## https://www.runoob.com/maven/maven-build-profiles.html

## 约定配置

Maven 提倡使用一个共同的标准目录结构，Maven 使用约定优于配置的原则，大家尽可能的遵守这样的目录结构。如下所示：

| **目录** | **目的** |
| --- | --- |
| ${basedir} | 存放pom.xml和所有的子目录 |
| ${basedir}/src/main/java | 项目的java源代码 |
| ${basedir}/src/main/resources | 项目的资源，比如说property文件，springmvc.xml |
| ${basedir}/src/test/java | 项目的测试类，比如说Junit代码 |
| ${basedir}/src/test/resources | 测试用的资源 |
| ${basedir}/src/main/webapp/WEB-INF | web应用文件目录，web项目的信息，比如存放web.xml、本地图片、jsp视图页面 |
| ${basedir}/target | 打包输出目录 |
| ${basedir}/target/classes | 编译输出目录 |
| ${basedir}/target/test-classes | 测试编译输出目录 |
| Test.java | Maven只会自动运行符合该命名规则的测试类 |
| ~/.m2/repository | Maven默认的本地仓库目录位置 |

## Maven POM

POM( Project Object Model，项目对象模型 ) 是 Maven 工程的基本工作单元，是一个XML文件，包含了项目的基本信息，用于描述项目如何构建，声明项目依赖，等等。

执行任务或目标时，Maven 会在当前目录中查找 POM。它读取 POM，获取所需的配置信息，然后执行目标。

POM 中可以指定以下配置：

* 项目依赖
* 插件
* 执行目标
* 项目构建 profile
* 项目版本
* 项目开发者列表
* 相关邮件列表信息



有一个默认的父（Super）POM，也就是上图的配置，可以被覆写。

## [POM 标签大全详解](maven配置管理说明附加/pom.xml)

## Maven 构建生命周期

Maven 构建生命周期定义了一个项目构建跟发布的过程。

一个典型的 Maven 构建（build）生命周期是由以下几个阶段的序列组成的：

| **阶段** | **处理** | **描述** |
| --- | --- | --- |
| 验证 validate | 验证项目 | 验证项目是否正确且所有必须信息是可用的 |
| 编译 compile | 执行编译 | 源代码编译在此阶段完成 |
| 测试 Test | 测试 | 使用适当的单元测试框架（例如JUnit）运行测试。 |
| 包装 package | 打包 | 创建JAR/WAR包如在 pom.xml 中定义提及的包 |
| 检查 verify | 检查 | 对集成测试的结果进行检查，以保证质量达标 |
| 安装 install | 安装 | 安装打包的项目到本地仓库，以供其他项目使用 |
| 部署 deploy | 部署 | 拷贝最终的工程包到远程仓库中，以共享给其他开发人员和工程 |

Maven 有以下三个标准的生命周期：

* **clean**：项目清理的处理
* **default(或 build)**：项目部署的处理
* **site**：项目站点文档创建的处理

### Clean 生命周期

当我们执行 mvn post-clean 命令时，Maven 调用 clean 生命周期，它包含以下阶段：

* 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

mvn clean 中的 clean 就是上面的 clean，在一个生命周期中，运行某个阶段的时候，它之前的所有阶段都会被运行，也就是说，如果执行 mvn clean 将运行以下两个生命周期阶段：

pre-clean, clean

如果我们运行 mvn post-clean ，则运行以下三个生命周期阶段：

pre-clean, clean, post-clean

我们可以通过在上面的 clean 生命周期的任何阶段定义目标来修改这部分的操作行为。

### Default (Build) 生命周期

这是 Maven 的主要生命周期，被用于构建应用，包括下面的 23 个阶段：

| **生命周期阶段** | **描述** |
| --- | --- |
| validate | 检查工程配置是否正确，完成构建过程的所有必要信息是否能够获取到。 |
| initialize | 初始化构建状态，例如设置属性。 |
| generate-sources | 生成编译阶段需要包含的任何源码文件。 |
| process-sources | 处理源代码，例如，过滤任何值（filter any value）。 |
| generate-resources | 生成工程包中需要包含的资源文件。 |
| process-resources | 拷贝和处理资源文件到目的目录中，为打包阶段做准备。 |
| compile | 编译工程源码。 |
| process-classes | 处理编译生成的文件，例如 Java Class 字节码的加强和优化。 |
| generate-test-sources | 生成编译阶段需要包含的任何测试源代码。 |
| process-test-sources | 处理测试源代码，例如，过滤任何值（filter any values)。 |
| test-compile | 编译测试源代码到测试目的目录。 |
| process-test-classes | 处理测试代码文件编译后生成的文件。 |
| test | 使用适当的单元测试框架（例如JUnit）运行测试。 |
| prepare-package | 在真正打包之前，为准备打包执行任何必要的操作。 |
| package | 获取编译后的代码，并按照可发布的格式进行打包，例如 JAR、WAR 或者 EAR 文件。 |
| pre-integration-test | 在集成测试执行之前，执行所需的操作。例如，设置所需的环境变量。 |
| integration-test | 处理和部署必须的工程包到集成测试能够运行的环境中。 |
| post-integration-test | 在集成测试被执行后执行必要的操作。例如，清理环境。 |
| verify | 运行检查操作来验证工程包是有效的，并满足质量要求。 |
| install | 安装工程包到本地仓库中，该仓库可以作为本地其他工程的依赖。 |
| deploy | 拷贝最终的工程包到远程仓库中，以共享给其他开发人员和工程。 |

### Site 生命周期

Maven Site 插件一般用来创建新的报告文档、部署站点等。

* 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

## 构建配置文件的类型

构建配置文件大体上有三种类型:

| **类型** | **在哪定义** |
| --- | --- |
| 项目级（Per Project） | 定义在项目的POM文件pom.xml中 |
| 用户级 （Per User） | 定义在Maven的设置xml文件中 (%USER\_HOME%/.m2/settings.xml) |
| 全局（Global） | 定义在 Maven 全局的设置 xml 文件中 (%M2\_HOME%/conf/settings.xml) |