

# Maven Study

## 1. Maven 功能

### Maven 功能

Maven 能够帮助开发者完成以下工作：

- 构建
- 文档生成
- 报告
- 依赖
- SCMs
- 发布
- 分发
- 邮件列表

## 2. 约定配置： 很重要！！文件夹的命名需要按照这个来！！！！

### 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

## 3. Maven POM(Project Object Model, 项目对象模型)

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。

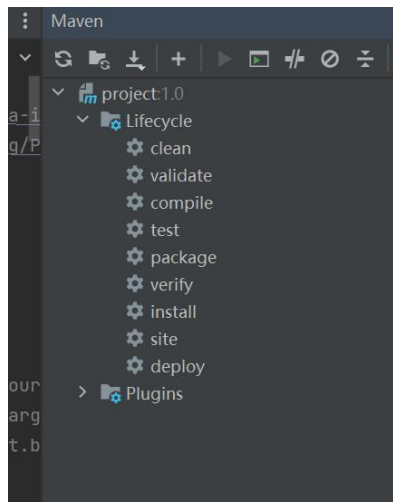
执行任务或目标时，Maven 会在当前目录中查找 POM。它读取 POM，获取所需的配置信息，然后执行目标。

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

#### 4. Maven 构建生命周期





## 4.1. Clean 生命周期

Input:

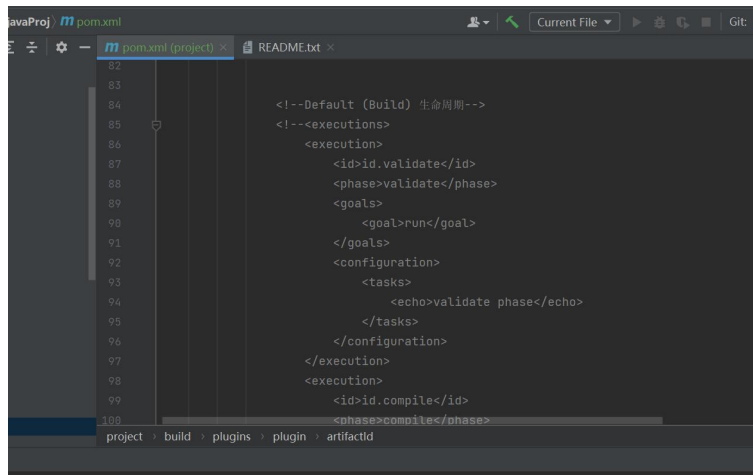
```
<!--Clean 生命周期-->
<!--<executions>
  <execution>
    <id>id.pre-clean</id>
    <phase>pre-clean</phase>
    <goals>
      <goal>run</goal>
    </goals>
    <configuration>
      <tasks>
        <echo>pre-clean phase</echo>
      </tasks>
    </configuration>
  </execution>
  <execution>
    <id>id.clean</id>
    <phase>clean</phase>
    <goals>
      <goal>run</goal>
    </goals>
    <configuration>
```

Output:

```
[INFO] Executed tasks
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
```

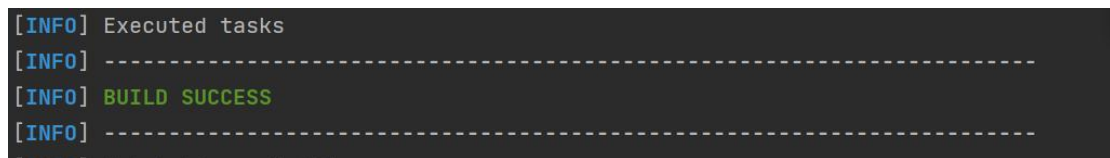
## 4.2. Default (Build) 生命周期

Input:



```
82
83
84      <!-- Default (Build) 生命周期-->
85      <!--executions>
86          <execution>
87              <id>id.validate</id>
88              <phase>validate</phase>
89              <goals>
90                  <goal>run</goal>
91              </goals>
92              <configuration>
93                  <tasks>
94                      <echo>validate phase</echo>
95                  </tasks>
96              </configuration>
97          </execution>
98          <execution>
99              <id>id.compile</id>
100             <phase>compile</phase>
```

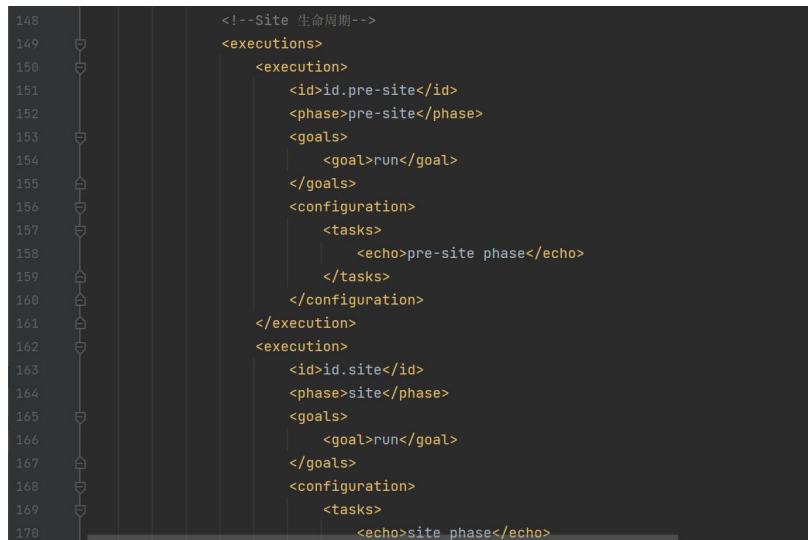
Output:



```
[INFO] Executed tasks
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
```

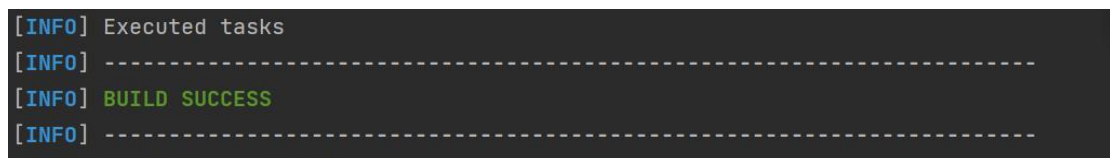
## 4.3. Site 生命周期

Input:

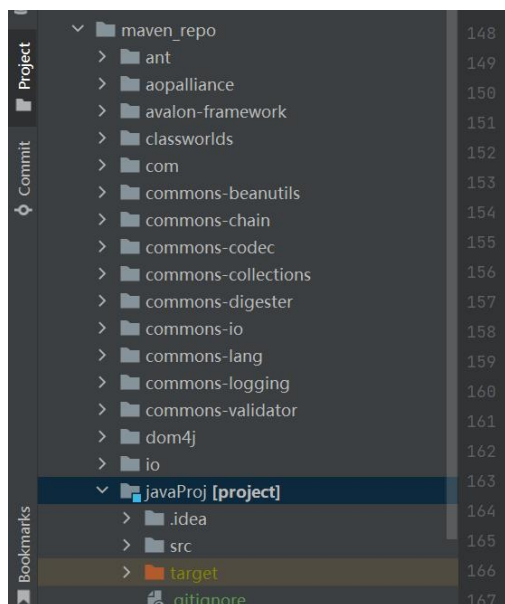


```
148
149      <!-- Site 生命周期-->
150      <executions>
151          <execution>
152              <id>id.pre-site</id>
153              <phase>pre-site</phase>
154              <goals>
155                  <goal>run</goal>
156              </goals>
157              <configuration>
158                  <tasks>
159                      <echo>pre-site phase</echo>
160                  </tasks>
161              </configuration>
162          </execution>
163          <execution>
164              <id>id.site</id>
165              <phase>site</phase>
166              <goals>
167                  <goal>run</goal>
168              </goals>
169              <configuration>
170                  <tasks>
171                      <echo>site phase</echo>
```

Output:



```
[INFO] Executed tasks
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
```



## 5. Maven 构建配置文件

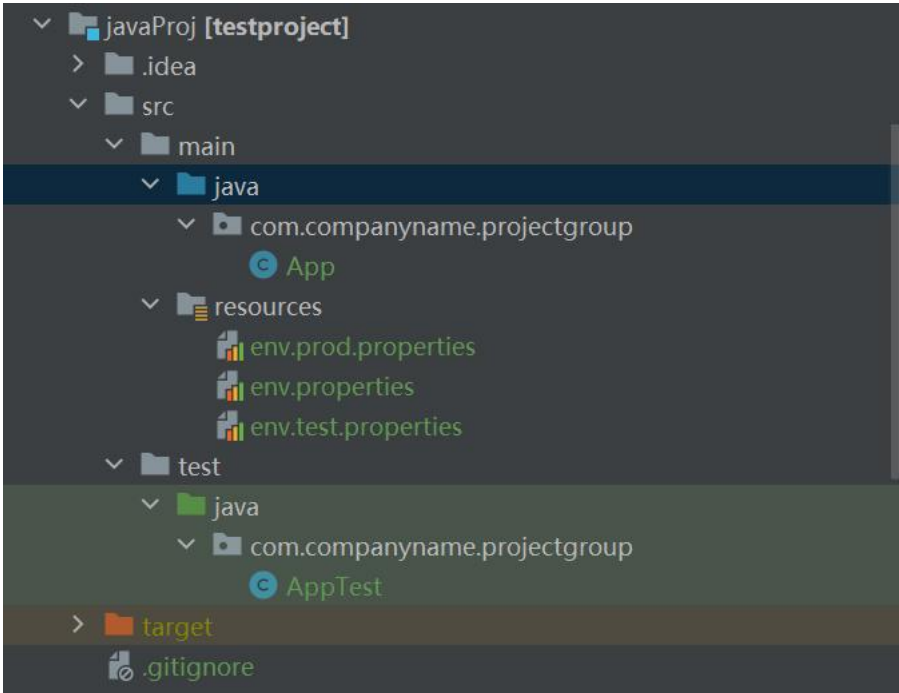
### 构建配置文件的类型

#### 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)

假定项目结构如下:



其中在src/main/resources文件夹下有三个用于测试文件：

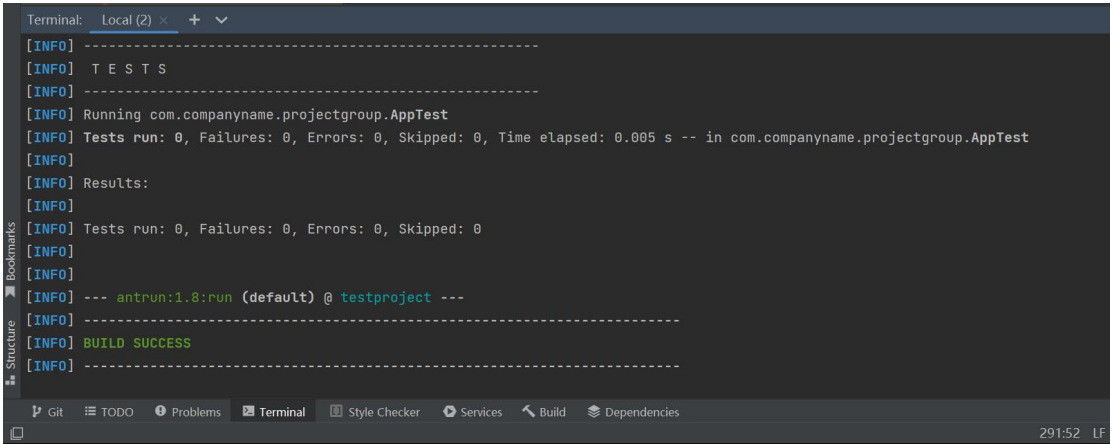
文件名	描述
env.properties	如果未指定配置文件时默认使用的配置。
env.test.properties	当测试配置文件使用时的测试配置。
env.prod.properties	当生产配置文件使用时的生产配置。

**注意：**这三个配置文件并不是代表构建配置文件的功能，而是用于本次测试的目的；比如，我指定了构建配置文件为 `prod` 时，项目就使用 `env.prod.properties` 文件。

**注意：**下面的例子仍然是使用 `AntRun` 插件，因为此插件能绑定 `Maven` 生命周期阶段，并通过 `Ant` 的标签不用编写一点代码即可输出信息、复制文件等，经此而已。其余的与本次构建配置文件无关。

配置文件激活：

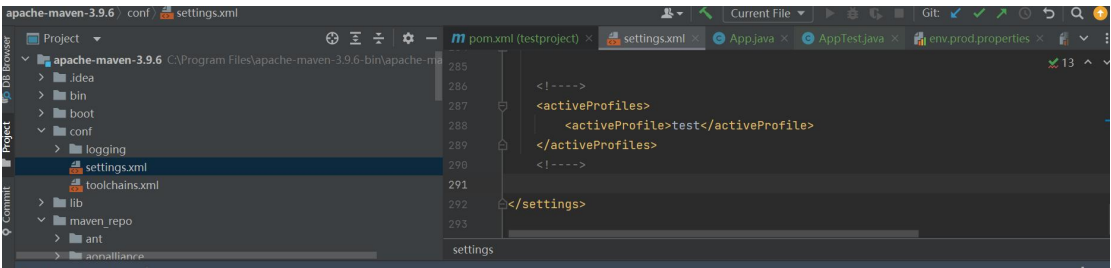
Output: （）



```
[WARNING] Parameter 'tasks' is deprecated: Use target instead
[WARNING] Parameter tasks is deprecated, use target instead
[INFO] Executing tasks

main:
  [echo] Using env.test.properties
  [copy] Copying 1 file to C:\Program Files\apache-maven-3.9.6-bin\apache-maven-3.9.6\maven_repo\javaProj\target\classes
[INFO] Executed tasks
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.883 s
[INFO] Finished at: 2024-01-11T10:35:17+08:00
[INFO] -----
PS C:\Program Files\apache-maven-3.9.6-bin\apache-maven-3.9.6\maven_repo\javaProj> $
```

## 2.通过 Maven 设置



### 2、通过Maven设置激活配置文件

打开 %USER\_HOME%\m2 目录下的 settings.xml 文件，其中 %USER\_HOME% 代表用户主目录。如果 settings.xml 文件不存在就复制 %M2\_HOME%\conf\settings.xml 到 m2 目录，其中 %M2\_HOME% 代表 Maven 的安装目录。

配置 settings.xml 文件，增加 <activeProfiles>属性：

```
<settings 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/xsd/settings-1.0.0.xsd">
  ...
  <activeProfiles>
    <activeProfile>test</activeProfile>
  </activeProfiles>
</settings>
```

执行命令：

```
mvn test
```

提示 1：此时不需要使用 -Ptest 来输入参数了，上面的 settings.xml 文件的 <activeprofile> 已经指定了 test 参数代替了。

提示 2：同样可以使用在 %M2\_HOME%\conf\settings.xml 的文件进行配置，效果一致。

执行结果：

```
D:\开发工程\Github\5_java_example\maventest\test4\testproject
> mvn test
[INFO] Scanning for projects...
[INFO]
[INFO] Building testproject 0.1-SNAPSHOT
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.239 s
[INFO] Finished at: 2024-01-11T10:35:17+08:00
```

Output:

```
Terminal: Local < + v
[WARNING] Parameter tasks is deprecated, use target instead
[INFO] Executing tasks

main:
  [echo] Using env.test.properties
  [copy] Copying 1 file to C:\Program Files\apache-maven-3.9.6-bin\apache-maven-3.9.6\maven_repo\javaProj\target\classes
[INFO] Executed tasks
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.239 s
[INFO] Finished at: 2024-01-11T10:35:17+08:00
```

### 3. 通过环境变量激活配置环境：

### 3、通过环境变量激活配置文件

先把上一步测试的 `setting.xml` 值全部去掉。

然后在 `pom.xml` 里面的 `<id>` 为 `test` 的 `<profile>` 节点，加入 `<activation>` 节点：

```
<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.jsoft.test</groupId>
  <artifactId>testproject</artifactId>
  <packaging>jar</packaging>
  <version>0.1-SNAPSHOT</version>
  <name>testproject</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <profiles>
    <profile>
      <id>test</id>
      <activation>
        <property>
          <name>env</name>
```

.....  
.....  
.....

## 6. Maven 插件

### Maven 插件

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

每个生命周期中都包含着一系列的阶段(phase)。这些 phase 就相当于 Maven 提供的统一的接口，然后这些 phase 的实现由 Maven 的插件来完成。

我们在输入 `mvn` 命令的时候 比如 `mvn clean`，`clean` 对应的就是 Clean 生命周期中的 `clean` 阶段。但是 `clean` 的具体操作是由 `maven-clean-plugin` 来实现的。

所以说 Maven 生命周期的每一个阶段的具体实现都是由 Maven 插件实现的。

Maven 实际上是一个依赖插件执行的框架，每个任务实际上是由插件完成。Maven 插件通常被用来：

- 创建 `jar` 文件
- 创建 `war` 文件
- 编译代码文件
- 代码单元测试
- 创建工程文档



# 7. Maven 构建 Java 项目

## Maven 构建 Java 项目：

Maven 使用原型 **archetype** 插件创建项目。要创建一个简单的 Java 应用，我们将使用 **maven-archetype-quickstart** 插件。

在下面的例子中，我们将在 C:\MVN 文件夹下创建一个基于 maven 的 java 应用项目。

命令格式如下：

```
mvn archetype:generate "-DgroupId=com.companyname.bank" "-DartifactId=consumerBanking" "-DarchetypeArtifactId=maven-archetype-quickstart" "-DinteractiveMode=false"
```

参数说明：

- **-DgroupId**: 组织名，公司网址的反写 + 项目名称
- **-DartifactId**: 项目名-模块名
- **-DarchetypeArtifactId**: 指定 ArchetypeId, maven-archetype-quickstart, 创建一个简单的 Java 应用
- **-DinteractiveMode**: 是否使用交互模式

Input:

```
PS C:\Program Files\apache-maven-3.9.6-bin\apache-maven-3.9.6\maven_repo> mvn archetype:generate "-DgroupId=com.companyname.bank" "-DartifactId=consumerBanking" "-DarchetypeArtifactId=maven-archetype-quickstart" "-DinteractiveMode=false"
```

Output: (文件结构如下)

