Maven 教程之 pom.xml 详解

小知

> 作者：dunwu
> github.com/dunwu/blog

简介

什么是 pom?

POM 是 Project Object Model 的缩写，即项目对象模型。

pom.xml 就是 maven 的配置文件，用以描述项目的各种信息。

pom 配置一览

```
<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/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <!-- The Basics -->
  <groupId>...</groupId>
  <artifactId>...</artifactId>
  <version>...</version>
  <packaging>...</packaging>
  <dependencies>...</dependencies>
  <parent>...</parent>
  <dependencyManagement>...</dependencyManagement>
  <modules>...</modules>
  <properties>...</properties>

  <!-- Build Settings -->
  <build>...</build>
  <reporting>...</reporting>

  <!-- More Project I
  <name>...</name>
```

```
  <description>...</description>
  <url>...</url>
  <inceptionYear>...</inceptionYear>
  <licenses>...</licenses>
  <organization>...</organization>
  <developers>...</developers>
  <contributors>...</contributors>

  <!-- Environment Settings -->
  <issueManagement>...</issueManagement>
  <ciManagement>...</ciManagement>
  <mailingLists>...</mailingLists>
  <scm>...</scm>
  <prerequisites>...</prerequisites>
  <repositories>...</repositories>
  <pluginRepositories>...</pluginRepositories>
  <distributionManagement>...</distributionManagement>
  <profiles>...</profiles>
</project>
```

基本配置

- **project** - `project` 是 pom.xml 中描述符的根。
- **modelVersion** - `modelVersion` 指定 pom.xml 符合哪个版本的描述符。maven 2 和 3 只能为 4.0.0。

一般 jar 包被识别为： `groupId:artifactId:version` 的形式。

```
<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/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.codehaus.mojo</groupId>
  <artifactId>my-project</artifactId>
  <version>1.0</version>
  <packaging>war</packaging>
</project>
```

maven 坐标

在 maven 中，根据 `groupId` 、 `artifactId` 、 `version` 组合成 `groupId:artifactId:version` 来唯一识别一个 jar 包。

- **groupId** - 团体、组织的标识符。团体标识的约定是，它以创建这个项目的组织名称的逆向域名(reverse domain name)开头。一般对应着 java 的包结构。
- **artifactId** - 单独项目的唯一标识符。比如我们的 tomcat、commons 等。不要在 artifactId 中包含点号(.)。
- **version** - 一个项目的特定版本。
- maven 有自己的版本规范，一般是如下定义 major version、minor version、incremental version-qualifier ，比如 1.2.3-beta-01。要说明的是，maven 自己判断版本的算法是 major、minor、incremental 部分用数字比较，qualifier 部分用字符串比较，所以要小心 alpha-2 和 alpha-15 的比较关系，最好用 alpha-02 的格式。
- maven 在版本管理时候可以使用几个特殊的字符串 SNAPSHOT、LATEST、RELEASE。比如 `1.0-SNAPSHOT` 。各个部分的含义和处理逻辑如下说明：
  - **SNAPSHOT** - 这个版本一般用于开发过程中，表示不稳定的版本。
  - **LATEST** - 指某个特定构件的最新发布，这个发布可能是一个发布版，也可能是一个 snapshot 版，具体看哪个时间最后。
  - **RELEASE** ： 指最后一个发布版。
- **packaging** - 项目的类
  jar, maven-plugin, ejb

依赖配置

dependencies

```
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  ...
  <dependencies>
    <dependency>
     <groupId>org.apache.maven</groupId>
      <artifactId>maven-embedder</artifactId>
      <version>2.0</version>
      <type>jar</type>
      <scope>test</scope>
      <optional>true</optional>
      <exclusions>
        <exclusion>
          <groupId>org.apache.maven</groupId>
          <artifactId>maven-core</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
    ...
  </dependencies>
  ...
</project>
```

- **groupId**, **artifactId**, **version** - 和基本配置中的 `groupId` 、 `artifactId` 、 `version` 意义相同。
- **type** - 对应 `packaging` 的类型，如果不使用 `type` 标签，maven 默认为 jar。
- **scope** - 此元素指的是任务的类路径（编译和运行时，测试等）以及如何限制依赖关系的传递性。有 5 种可用的限定范围：
- **compile** - 如果没有指定 `scope` 标签，maven 默认为这个范围。编译依赖关系在所有 classpath 中都可用。此外，这些依赖关系被传播到依赖项目。
- **provided** - 与 compile 类似，但是表示您希望 jdk 或容器在运行时提供它。它只适用于编译和测试 classpath，不可传递。
- **runtime** - 此范围表示编译不需要依赖关系，而是用于执行。它是在运行时和测试 classpath，但不是编译 classpath。
- **test** - 此范围表示正常使用应用程序不需要依赖关系，仅适用于测试编译和执行阶段。它不是传递的。
- **system** - 此范围与 provided 类似，除了您必须提供明确包含它的 jar。该 artifact 始终可用，并且不是在仓库中查找。
- **systemPath** - 仅当依赖范围是系统时才使用。否则，如果设置此元素，构建将失败。该路径必须是绝对路径，因此建议使用 `propertie` 来指定特定的路径，如 `\$ {java.home} / lib` 。由于假定先前安装了系统范围依赖关系，maven 将不会检查项目的仓库，而是检查库文件是否存在。如果没有，maven 将会失败，并建议您手动下载安装。
- **optional** - `optional` 让其他项目知道，当您使用此项目时，您不需要这种依赖性才能正常工作。
- **exclusions** - 包含一个或多个排除元素，每个排除元素都包含一个表示要排除的依赖关系的 `groupId` 和 `artifactId` 。与可选项不同，可能或可能不会安装和使用，排除主动从依赖关系树中删除自己。

parent

maven 支持继承功能。子 POM 可以使用 `parent` 指定父 POM ，然后继承其配置。

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
```

```
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>my-parent</artifactId>
    <version>2.0</version>
    <relativePath>../my-parent</relativePath>
  </parent>

  <artifactId>my-project</artifactId>
</project>
```

- **relativePath** - 注意 `relativePath` 元素。在搜索本地和远程存储库之前，它不是必需的，但可以用作 maven 的指示符，以首先搜索给定该项目父级的路径。

dependencyManagement

`dependencyManagement` 是表示依赖 jar 包的声明。即你在项目中的 `dependencyManagement` 下声明了依赖，maven 不会加载该依赖， `dependencyManagement` 声明可以被子 POM 继承。

`dependencyManagement` 的一个使用案例是当有父子项目的时候，父项目中可以利用 `dependencyManagement` 声明子项目中需要用到的依赖 jar 包，之后，当某个或者某几个子项目需要加载该依赖的时候，就可以在子项目中 `dependencies` 节点只配置 `groupId` 和 `artifactId` 就可以完成依赖的引用。

`dependencyManagement` 主要是为了统一管理依赖包的版本，确保所有子项目使用的版本一致，类似的还有 `plugins` 和 `pluginManagement` 。

modules

子模块列表。

```
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.codehaus.mojo</groupId>
  <artifactId>my-parent</artifactId>
  <version>2.0</version>
  <packaging>pom</packaging>

  <modules>
    <module>my-project</module>
    <module>another-project</module>
    <module>third-project/pom-example.xml</module>
  </modules>
</project>
```

properties

属性列表。定义的属性可以在 pom.xml 文件中任意处使用。使用方式为 `${propertie}` 。

```
<project>
  ...
  <properties>
    <maven.compiler.source>1.7<maven.compiler.source>
    <maven.compiler.target>1.7<maven.compiler.target>
    <project.build.so
    <project.reportin
```

```
  </properties>
  ...
</project>
```

构建配置

build

build 可以分为 "project build" 和 "profile build"。

```
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  ...
  <!-- "Project Build" contains more elements than just the BaseBuild set -->
  <build>...</build>

  <profiles>
    <profile>
      <!-- "Profile Build" contains a subset of "Project Build"s elements -->
      <build>...</build>
    </profile>
  </profiles>
</project>
```

基本构建配置：

```
<build>
  <defaultGoal>install</defaultGoal>
  <directory>${basedir}/target</directory>
  <finalName>${artifactId}-${version}</finalName>
  <filters>
    <filter>filters/filter1.properties</filter>
  </filters>
  ...
</build>
```

defaultGoal：默认执行目标或阶段。如果给出了一个目标，它应该被定义为它在命令行中（如 jar：jar）。如果定义了一个阶段（如安装），也是如此。

directory： 构建时的输出路径。默认为： `${basedir}/target` 。

finalName： 这是项目的最终构建名称（不包括文件扩展名，例如：my-project-1.0.jar）

filter： 定义 `*.properties` 文件，其中包含适用于接受其设置的资源的属性列表（如下所述）。换句话说，过滤器文件中定义的“name = value”对在代码中替换 `\${name}` 字符串。

resources

资源的配置。资源文件通常不是代码，不需要编译，而是在项目需要捆绑使用的内容。

```
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <build>
    ...
    <resources>
      <resource>
        <targetPath>M
        <filtering>fa
```

```
        <directory>${basedir}/src/main/plexus</directory>
        <includes>
          <include>configuration.xml</include>
        </includes>
        <excludes>
          <exclude>**/*.properties</exclude>
        </excludes>
      </resource>
    </resources>
    <testResources>
      ...
    </testResources>
    ...
  </build>
</project>
```

- **resources**: 资源元素的列表，每个资源元素描述与此项目关联的文件和何处包含文件。
- **targetPath**: 指定从构建中放置资源集的目录结构。目标路径默认为基本目录。将要包装在 jar 中的资源的通常指定的目标路径是 META-INF。
- **filtering**: 值为 true 或 false。表示是否要为此资源启用过滤。请注意，该过滤器 `*.properties` 文件不必定义为进行过滤 - 资源还可以使用默认情况下在 POM 中定义的属性（例如\$ {project.version}），并将其传递到命令行中“-D”标志（例如，“-Dname = value”）或由 properties 元素显式定义。过滤文件覆盖上面。
- **directory**: 值定义了资源的路径。构建的默认目录是 `${basedir}/src/main/resources` 。
- **includes**: 一组文件匹配模式，指定目录中要包括的文件，使用*作为通配符。
- **excludes**: 与 `includes` 类似，指定目录中要排除的文件，使用*作为通配符。注意：如果 `include` 和 `exclude` 发生冲突，maven 会以 `exclude` 作为有效项。
- **testResources**: `testResources` 与 `resources` 功能类似，区别仅在于： `testResources` 指定的资源仅用于 test 阶段，并且其默认资源目录为： `${basedir}/src/test/resources` 。

plugins

```
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <build>
    ...
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <version>2.6</version>
        <extensions>false</extensions>
        <inherited>true</inherited>
        <configuration>
          <classifier>test</classifier>
        </configuration>
        <dependencies>...</dependencies>
        <executions>...</executions>
      </plugin>
    </plugins>
  </build>
</project>
```

- **groupId**, **artifactId**, **version**： 和基本配置中的 `groupId` 、 `artifactId` 、 `version` 意义相同。
- **extensions**： 值为 true 或 false。是否加载此插件的扩展名。默认为 false。
- **inherited**： 值为 true 或 false。这个插件配置是否应该适用于继承自这个插件的 POM。默认值为 true。
- **configuration** - 这是针对个人插件的配置，这里不扩散讲解。
- **dependencies**： 这里

- **executions**： 需要记住的是，插件可能有多个目标。每个目标可能有一个单独的配置，甚至可能将插件的目标完全绑定到不同的阶段。执行配置插件的目标的执行。
- **id**: 执行目标的标识。
- **goals**: 像所有多元化的 POM 元素一样，它包含单个元素的列表。在这种情况下，这个执行块指定的插件目标列表。
- **phase**: 这是执行目标列表的阶段。这是一个非常强大的选项，允许将任何目标绑定到构建生命周期中的任何阶段，从而改变 maven 的默认行为。
- **inherited**: 像上面的继承元素一样，设置这个 false 会阻止 maven 将这个执行传递给它的子代。此元素仅对父 POM 有意义。
- **configuration**: 与上述相同，但将配置限制在此特定目标列表中，而不是插件下的所有目标。

```
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  ...
  <build>
    <plugins>
      <plugin>
        <artifactId>maven-antrun-plugin</artifactId>
        <version>1.1</version>
        <executions>
          <execution>
            <id>echodir</id>
            <goals>
              <goal>run</goal>
            </goals>
            <phase>verify</phase>
            <inherited>false</inherited>
            <configuration>
              <tasks>
                <echo>Build Dir: ${project.build.directory}</echo>
              </tasks>
            </configuration>
          </execution>
        </executions>

      </plugin>
    </plugins>
  </build>
</project>
```

pluginManagement

与 `dependencyManagement` 很相似，在当前 POM 中仅声明插件，而不是实际引入插件。子 POM 中只配置 `groupId` 和 `artifactId` 就可以完成插件的引用，且子 POM 有权重写 pluginManagement 定义。

它的目的在于统一所有子 POM 的插件版本。

directories

```
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  ...
  <build>
    <sourceDirectory>${basedir}/src/main/java</sourceDirectory>
    <scriptSourceDirectory>${basedir}/src/main/scripts</scriptSourceDirectory>
    <testSourceDirectory>${basedir}/src/test/java</testSourceDirectory>
    <outputDirectory>${basedir}/target/classes</outputDirectory>
    <testOutputDirect
    ...
```

```
  </build>
</project>
```

目录元素集合存在于 `build` 元素中，它为整个 POM 设置了各种目录结构。由于它们在配置文件构建中不存在，所以这些不能由配置文件更改。

如果上述目录元素的值设置为绝对路径（扩展属性时），则使用该目录。否则，它是相对于基础构建目录： `${basedir}` 。

extensions

扩展是在此构建中使用的 artifacts 的列表。它们将被包含在运行构建的 classpath 中。它们可以启用对构建过程的扩展（例如为 Wagon 传输机制添加一个 ftp 提供程序），并使活动的插件能够对构建生命周期进行更改。简而言之，扩展是在构建期间激活的 artifacts。扩展不需要实际执行任何操作，也不包含 Mojo。因此，扩展对于指定普通插件接口的多个实现中的一个是非常好的。

```
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  ...
  <build>
    ...
    <extensions>
      <extension>
        <groupId>org.apache.maven.wagon</groupId>
        <artifactId>wagon-ftp</artifactId>
        <version>1.0-alpha-3</version>
      </extension>
    </extensions>
    ...
  </build>
</project>
```

reporting

报告包含特定针对 `site` 生成阶段的元素。某些 maven 插件可以生成 `reporting` 元素下配置的报告，例如：生成 javadoc 报告。 `reporting` 与 `build` 元素配置插件的能力相似。明显的区别在于：在执行块中插件目标的控制不是细粒度的，报表通过配置 `reportSet` 元素来精细控制。

而微妙的区别在于 `reporting` 元素下的 `configuration` 元素可以用作 `build` 下的 `configuration` ，尽管相反的情况并非如此（ `build` 下的 `configuration` 不影响 `reporting` 元素下的 `configuration` ）。

另一个区别就是 `plugin` 下的 `outputDirectory` 元素。在报告的情况下，默认输出目录为 `${basedir}/target/site` 。

```
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  ...
  <reporting>
    <plugins>
      <plugin>
        ...
        <reportSets>
          <reportSet>
            <id>sunlink</id>
            <reports>
              <report>javadoc</report>
            </reports
            <inherite
```

```
            <configuration>
              <links>
                <link>http://java.sun.com/j2se/1.5.0/docs/api/</link>
              </links>
            </configuration>
          </reportSet>
        </reportSets>
      </plugin>
    </plugins>
  </reporting>
  ...
</project>
```

项目信息

项目信息相关的这部分标签**都不是必要的**，也就是说完全可以不填写。

它的作用仅限于描述项目的详细信息。

下面的示例是项目信息相关标签的清单：

```
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  ...

  <!-- 项目信息 begin -->

  <!--项目名-->
  <name>maven-notes</name>

  <!--项目描述-->
  <description>maven 学习笔记</description>

  <!--项目url-->
  <url>https://github.com/dunwu/maven-notes</url>

  <!--项目开发年份-->
  <inceptionYear>2017</inceptionYear>

  <!--开源协议-->
  <licenses>
    <license>
      <name>Apache License, Version 2.0</name>
      <url>https://www.apache.org/licenses/LICENSE-2.0.txt</url>
      <distribution>repo</distribution>
      <comments>A business-friendly OSS license</comments>
    </license>
  </licenses>

  <!--组织信息(如公司、开源组织等)-->
  <organization>
    <name>...</name>
    <url>...</url>
  </organization>

  <!--开发者列表-->
  <developers>
    <developer>
      <id>victor</id>
      <name>Zhang Peng</name>
      <email>forbreak at 163.com</email>
      <url>https://github.com/dunwu</url>
      <organization>.
```

```
      <organizationUrl>...</organizationUrl>
      <roles>
        <role>architect</role>
        <role>developer</role>
      </roles>
      <timezone>+8</timezone>
      <properties>...</properties>
    </developer>
  </developers>

  <!--代码贡献者列表-->
   <contributors>
    <contributor>
      <!--标签内容和<developer>相同-->
    </contributor>
  </contributors>

  <!-- 项目信息 end -->

  ...
</project>
```

这部分标签都非常简单，基本都能做到顾名思义，且都属于可有可无的标签，所以这里仅简单介绍一下：

- **name** - 项目完整名称
- **description** - 项目描述
- **url** - 一般为项目仓库的 host
- **inceptionYear** - 开发年份
- **licenses** - 开源协议
- **organization** - 项目所属组织信息
- **developers** - 项目开发者列表
- **contributors** - 项目贡献者列表， 的子标签和 的完全相同。

环境配置

issueManagement

这定义了所使用的缺陷跟踪系统（Bugzilla，TestTrack，ClearQuest 等）。虽然没有什么可以阻止插件使用这些信息的东西，但它主要用于生成项目文档。

```
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  ...
  <issueManagement>
    <system>Bugzilla</system>
    <url>http://127.0.0.1/bugzilla/</url>
  </issueManagement>
  ...
</project>
```

ciManagement

CI 构建系统配置，主要是指定通知机制以及被通知的邮箱。

```
<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
                      [illegible]

  ...
```

```
  <ciManagement>
    <system>continuum</system>
    <url>http://127.0.0.1:8080/continuum</url>
    <notifiers>
      <notifier>
        <type>mail</type>
        <sendOnError>true</sendOnError>
        <sendOnFailure>true</sendOnFailure>
        <sendOnSuccess>false</sendOnSuccess>
        <sendOnWarning>false</sendOnWarning>
        <configuration><address>continuum@127.0.0.1</address></configuration>
      </notifier>
    </notifiers>
  </ciManagement>
  ...
</project>
```

mailingLists

邮件列表

```
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  ...
  <mailingLists>
    <mailingList>
      <name>User List</name>
      <subscribe>user-subscribe@127.0.0.1</subscribe>
      <unsubscribe>user-unsubscribe@127.0.0.1</unsubscribe>
      <post>user@127.0.0.1</post>
      <archive>http://127.0.0.1/user/</archive>
      <otherArchives>
        <otherArchive>http://base.google.com/base/1/127.0.0.1</otherArchive>
      </otherArchives>
    </mailingList>
  </mailingLists>
  ...
</project>
```

scm

SCM（软件配置管理，也称为源代码/控制管理或简洁的版本控制）。常见的 scm 有 svn 和 git 。

```
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  ...
  <scm>
    <connection>scm:svn:http://127.0.0.1/svn/my-project</connection>
    <developerConnection>scm:svn:https://127.0.0.1/svn/my-project</developerConnection
    <tag>HEAD</tag>
    <url>http://127.0.0.1/websvn/my-project</url>
  </scm>
  ...
</project>
```

prerequisites

POM 执行的预设条件。

```
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  ...
  <prerequisites>
    <maven>2.0.6</maven>
  </prerequisites>
  ...
</project>
```

repositories

`repositories` 是遵循 Maven 存储库目录布局的 artifacts 集合。默认的 Maven 中央存储库位于 repo.maven.apache.org/m...上。

```
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  ...
  <repositories>
    <repository>
      <releases>
        <enabled>false</enabled>
        <updatePolicy>always</updatePolicy>
        <checksumPolicy>warn</checksumPolicy>
      </releases>
      <snapshots>
        <enabled>true</enabled>
        <updatePolicy>never</updatePolicy>
        <checksumPolicy>fail</checksumPolicy>
      </snapshots>
      <id>codehausSnapshots</id>
      <name>Codehaus Snapshots</name>
      <url>http://snapshots.maven.codehaus.org/maven2</url>
      <layout>default</layout>
    </repository>
  </repositories>
  <pluginRepositories>
    ...
  </pluginRepositories>
  ...
</project>
```

pluginRepositories

与 `repositories` 差不多。

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  ...
  <distributionManagement>
    ...
    <downloadUrl>http://mojo.codehaus.org/my-project</downloadUrl>
    <status>deployed</status>
  </distributionManagement>
  ...
</project>
```

distributionManagement

它管理在整个构建过程中生成的 artifact 和支持文件的分布。从最后的元素开始：

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  ...
  <distributionManagement>
    ...
    <downloadUrl>http://mojo.codehaus.org/my-project</downloadUrl>
    <status>deployed</status>
  </distributionManagement>
  ...
</project>
```

- **repository** - 与 `repositories` 相似
- **site** - 站点信息
- **relocation** - 项目迁移位置

profiles

`activation` 是一个 `profile` 的关键。配置文件的功能来自于在某些情况下仅修改基本 POM 的功能。这些情况通过 `activation` 元素指定。

```
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  ...
  <profiles>
    <profile>
      <id>test</id>
      <activation>
        <activeByDefault>false</activeByDefault>
        <jdk>1.5</jdk>
        <os>
          <name>Windows XP</name>
          <family>Windows</family>
          <arch>x86</arch>
          <version>5.1.2600</version>
        </os>
        <property>
          <name>sparrow-type</name>
          <value>African</value>
        </property>
        <file>
          <exists>${basedir}/file2.properties</exists>
          <missing>${basedir}/file1.properties</missing>
        </file>
      </activation>
      ...
    </profile>
  </profiles>
</project>
```

参考资料

推荐阅读(点击即可跳转阅读)

1. SpringBoot内容聚合
2. 面试题内容聚合
3. 设计模式内容聚合
4. Mybatis内容聚合
5. 多线程内容聚合

发布于 2019-08-06 06:06

Maven

写下你的评论...

5 条评论 默认 最新

季夏
想问一下，利用Android Studio基于Java编写开发程序的时候，弹出日志警告SLF4J: See slf4j.org/codes.html#... for further details.
2023-04-24 回复 喜欢

季夏
这个具体的解决方法是什么呢
2023-04-24 回复 喜欢

gamble life
强强强
2022-08-06 回复 喜欢

Jokerdos
万分感谢
2022-05-07 回复 喜欢

欢hhhhh
nb啊，写的清晰明了
2021-11-26 回复 喜欢

文章被以下专栏收录

Java知音
Java知音在知乎的分类总结

推荐阅读

Maven POM配置详解
看了一篇大佬的文章,对maven各个标签讲解很全面,分享给大家 Maven项目的核心是pom.xml，POM（project object model，项目对象模型）定义了项目的基本信息，用于描述项目如何构建，如何声...
龙神之飞绝

XML 解析详解 - 四种解析方式

效率比拟RapidXml的XML解析库Textcat::XML即将完工

最简单的
Pagic
GitHub: xc
easiest wa
html page
很多地方用
Jekyll 和 H
不需要它的