Maven: settings.xml、pom.xml完整配置

来自：彭世瑜的博客 2021-11-26 2364 举报

简介： Maven: settings.xml、pom.xml完整配置

完整配置

settings.xml

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                          https://maven.apache.org/xsd/settings-1.0.0.xsd">
  <!-- 本地仓库配置：默认~/.m2/repository[店家推荐修改配置] -->
  <localRepository>${user.home}/.m2/repository</localRepository>

  <!-- 交互方式配置，读取用户输入信息[使用默认即可，很少修改] -->
  <interactiveMode>true</interactiveMode>

  <!-- 是否启用独立的插件配置文件，一般很少启用[默认即可，很少修改] -->
  <usePluginRegistry>false</usePluginRegistry>

  <!-- 是否启用离线构建模式，一般很少修改[如果长时间不能联网的情况下可以修改] -->
  <offline>false</offline>

  <!-- 是否启用插件groupId自动扫描[很少使用，配置插件时建议全信息配置] -->
  <pluginGroups>
    <pluginGroup>org.apache.maven.plugins</pluginGroup>
  </pluginGroups>

  <!--配置服务端的一些设置如身份认证信息(eg: 账号、密码) -->
  <servers>
    <!--服务器元素包含配置服务器时需要的信息 -->
    <server>
      <!--这是server的id（注意不是用户登陆的id）
      该id与distributionManagement中repository元素的id相匹配。
      -->
      <id>server_001</id>
      <!--身份鉴权令牌。鉴权/认证用户名和鉴权密码表示服务器认证所需要的登录名和密码。 -->
      <username>my_login</username>
      <!--身份鉴权密码 。鉴权/认证用户名和鉴权密码表示服务器认证所需要的登录名和密码-->
      <password>my_password</password>
      <!--鉴权/认证时使用的私钥文件位置。和前两个元素类似
      私钥位置和私钥密码指定了一个私钥的路径（默认是${user.home}/.ssh/id_dsa）==>
```

```
      <privateKey>${usr.home}/.ssh/id_dsa</privateKey>
      <!--鉴权/认证时使用的私钥密码。 -->
      <passphrase>some_passphrase</passphrase>
      <!--文件被创建时的权限。如果在部署的时候会创建一个仓库文件或者目录，这时候就可以使用权限（permission）。这两个元素合法的值是一个三位数字，其对应了unix文件系统的权限，如664，或者775。 -->
      <filePermissions>664</filePermissions>
      <!--目录被创建时的权限。 -->
      <directoryPermissions>775</directoryPermissions>
    </server>
  </servers>

   <mirrors>
    <!-- 默认仓库配置给定的下载镜像位置 -->
    <mirror>
      <!-- 该镜像的唯一标识符。id用来区分不同的mirror元素。 -->
      <id>nexus aliyun</id>
      <!-- 镜像名称 -->
      <name>Nexus Aliyun</name>
      <!-- 该镜像的URL。构建系统会优先考虑使用该URL，而非使用默认的服务器URL。 -->
      <url>http://downloads.planetmirror.com/pub/maven2</url>
      <!-- 被镜像的服务器的id。
      如果我们要设置了一个Maven中央仓库（http://repo.maven.apache.org/maven2/）的镜像
      就需要将mirrorOf设置成central。
      保持和中央仓库的id central一致。 这样就能替代中央仓库的功能了-->
      <mirrorOf>central</mirrorOf>
    </mirror>
  </mirrors>

  <proxies>
    <!--代理元素包含配置代理时需要的信息 -->
    <proxy>
      <!--代理的唯一定义符，用来区分不同的代理元素。 -->
      <id>myproxy</id>
      <!--该代理是否是激活的那个。true则激活代理。当我们声明了一组代理，而某个时候只需要激活一个代理的时候，该元素就可以派上用处。 -->
      <active>true</active>
      <!--代理的协议。 协议://主机名:端口，分隔成离散的元素以方便配置。 -->
      <protocol>http</protocol>
      <!--代理的主机名。协议://主机名:端口，分隔成离散的元素以方便配置。 -->
      <host>proxy.somewhere.com</host>
      <!--代理的端口。协议://主机名:端口，分隔成离散的元素以方便配置。 -->
      <port>8080</port>
      <!--代理的用户名，用户名和密码表示代理服务器认证的登录名和密码。 -->
      <username>proxyuser</username>
      <!--代理的密码，用户名和密码表示代理服务器认证的登录名和密码。 -->
      <password>somepassword</password>
      <!--不该被代理的主机名列表。该列表的分隔符由代理服务器指定；例子中使用了竖线分隔符，使用逗号分隔也很常见。 -->
      <nonProxyHosts>*.google.com|ibiblio.org</nonProxyHosts>
    </proxy>
```

```
  </proxies>

  <profiles>
    <profile>
      <!-- profile的唯一标识 -->
      <id>test</id>
      <!-- 自动触发profile的条件逻辑 -->
      <activation />
      <!-- 扩展属性列表 -->
      <properties />
      <!-- 远程仓库列表 -->
      <repositories />
      <!-- 插件仓库列表 -->
      <pluginRepositories />
    </profile>
  </profiles>


  <activeProfiles>
    <!-- 要激活的profile id -->
    <activeProfile>env-test</activeProfile>
  </activeProfiles>

  <activation>
    <!--profile默认是否激活的标识 -->
    <activeByDefault>false</activeByDefault>
    <!--当匹配的jdk被检测到，profile被激活。例如，1.4激活JDK1.4，1.4.0_2，而!1.4激活所有版本不是以1.4开头的JDK。 -->
    <jdk>1.5</jdk>
    <!--当匹配的操作系统属性被检测到，profile被激活。os元素可以定义一些操作系统相关的属性。-->
    <os>
      <!--激活profile的操作系统的名字 -->
      <name>Windows XP</name>
      <!--激活profile的操作系统所属家族(如 'windows') -->
      <family>Windows</family>
      <!--激活profile的操作系统体系结构 -->
      <arch>x86</arch>
      <!--激活profile的操作系统版本 -->
      <version>5.1.2600</version>
    </os>
    <!--如果Maven检测到某一个属性（其值可以在POM中通过${name}引用），其拥有对应的name = 值，Profile就会被激活。如果值字段是空的，那么存在属性名称字段就会激活profile，否则按区分大小写方式匹配属性值字段 -->
    <property>
      <!--激活profile的属性的名称 -->
      <name>mavenVersion</name>
      <!--激活profile的属性的值 -->
      <value>2.0.3</value>
```

```
    </property>
    <!--提供一个文件名，通过检测该文件的存在或不存在来激活profile。missing检查文件是否存在，如果不存在则激活profile。另一方面，exists则会检查文件是否存在，如果存在则激活profile。 -->
    <file>
      <!--如果指定的文件存在，则激活profile。 -->
      <exists>${basedir}/file2.properties</exists>
      <!--如果指定的文件不存在，则激活profile。 -->
      <missing>${basedir}/file1.properties</missing>
    </file>
  </activation>

<properties>
  <spring.Version>5.2.8</spring.Version>
</properties>

<repositories>
  <!--包含需要连接到远程仓库的信息 -->
  <repository>
    <!--远程仓库唯一标识 -->
    <id>codehausSnapshots</id>
    <!--远程仓库名称 -->
    <name>Codehaus Snapshots</name>
    <!--如何处理远程仓库里发布版本的下载 -->
    <releases>
      <!--true或者false表示该仓库是否为下载某种类型构件（发布版，快照版）开启。 -->
      <enabled>false</enabled>
      <!--该元素指定更新发生的频率。Maven会比较本地POM和远程POM的时间戳。这里的选项是：always（一直），daily（默认，每日），interval：X（这里X是以分钟为单位的时间间隔），或者never（从不）。 -->
      <updatePolicy>always</updatePolicy>
      <!--当Maven验证构件校验文件失败时该怎么做-ignore（忽略），fail（失败），或者warn（警告）。 -->
      <checksumPolicy>warn</checksumPolicy>
    </releases>
    <!--如何处理远程仓库里快照版本的下载。有了releases和snapshots这两组配置，POM就可以在每个单独的仓库中，为每种类型的构件采取不同的策略。例如，可能有人会决定只为开发目的开启对快照版本下载的支持。参见repositories/repository/releases元素 -->
    <snapshots>
      <enabled />
      <updatePolicy />
      <checksumPolicy />
    </snapshots>
    <!--远程仓库URL，按protocol://hostname/path形式 -->
    <url>http://snapshots.maven.codehaus.org/maven2</url>
    <!--用于定位和排序构件的仓库布局类型-可以是default（默认）或者legacy（遗留）。Maven 2为其仓库提供了一个默认的布局；然而，Maven 1.x有一种不同的布局。我们可以使用该元素指定布局是default（默认）还是legacy（遗留）。 -->
    <layout>default</layout>
  </repository>
</repositories>

```

```
</settings>
```

pom.xml

```
<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">
    <!--
    1、项目基本信息配置

    -->
    <!--父项目的坐标。如果项目中没有规定某个元素的值，那么父项目中的对应值即为项目的默认值。坐标包括group ID，artifact ID和 version。 -->
    <parent>
        <!--被继承的父项目的构件标识符 -->
        <artifactId />
        <!--被继承的父项目的全球唯一标识符 -->
        <groupId />
        <!--被继承的父项目的版本 -->
        <version />
        <!--父项目的pom.xml文件的相对路径。相对路径允许你选择一个不同的路径。默认值是../pom.xml。Maven首先在构建当前项目的地方寻找父项目的pom，其次在文件系统的这个位置（relativePath位置），然后在本地仓库，最后在远程仓库寻找父项目的pom。 -->
        <relativePath />
    </parent>
    <!--声明项目描述符遵循哪一个POM模型版本。模型本身的版本很少改变，虽然如此，但它仍然是必不可少的，这是为了当Maven引入了新的特性或者其他模型变更的时候，确保稳定性。 -->
    <modelVersion>4.0.0</modelVersion>
    <!--项目的全球唯一标识符，通常使用全限定的包名区分该项目和其他项目。并且构建时生成的路径也是由此生成， 如com.mycompany.app生成的相对路径为：/com/mycompany/app -->
    <groupId>asia.banseon</groupId>
    <!--构件的标识符，它和group ID一起唯一标识一个构件。换句话说，你不能有两个不同的项目拥有同样的artifact ID和groupID；在某个特定的group
        ID下，artifact ID也必须是唯一的。构件是项目产生的或使用的一个东西，Maven为项目产生的构件包括：JARs，源码，二进制发布和WARs等。 -->
    <artifactId>banseon-maven2</artifactId>
    <!--项目产生的构件类型，例如jar、war、ear、pom。插件可以创建他们自己的构件类型，所以前面列的不是全部构件类型 -->
    <packaging>jar</packaging>
    <!--项目当前版本，格式为:主版本.次版本.增量版本-限定版本号 -->
    <version>1.0-SNAPSHOT</version>
    <!--项目的名称，Maven产生的文档用 -->
    <name>banseon-maven</name>
    <!--项目主页的URL，Maven产生的文档用 -->
    <url>http://www.baidu.com/banseon</url>
    <!--项目的详细描述，Maven 产生的文档用。 当这个元素能够用HTML格式描述时（例如，CDATA中的文本会被解析器忽略，就可以包含HTML标签），
```

```
36          不鼓励使用纯文本描述。如果你需要修改产生的web站点的索引页面，你应该修改你自己的索引页
文件，而不是调整这里的文档。 -->
37      <description>A maven project to study maven.</description>
38      <!--项目创建年份，4位数字。当产生版权信息时需要使用这个值。 -->
39      <inceptionYear />
40      <!--项目相关邮件列表信息 -->
41      <mailingLists>
42          <!--该元素描述了项目相关的所有邮件列表。自动产生的网站引用这些信息。 -->
43          <mailingList>
44              <!--邮件的名称 -->
45              <name>Demo</name>
46              <!--发送邮件的地址或链接，如果是邮件地址，创建文档时，mailto: 链接会被自动创建
-->
47              <post>Demo@126.com</post>
48              <!--订阅邮件的地址或链接，如果是邮件地址，创建文档时，mailto: 链接会被自动创建
-->
49              <subscribe>Demo@126.com</subscribe>
50              <!--取消订阅邮件的地址或链接，如果是邮件地址，创建文档时，mailto: 链接会被自动
创建 -->
51              <unsubscribe>Demo@126.com</unsubscribe>
52              <!--你可以浏览邮件信息的URL -->
53              <archive>http://localhost:8080/demo/dev/</archive>
54          </mailingList>
55      </mailingLists>
56      <!--项目开发者列表 -->
57      <developers>
58          <!--某个项目开发者的信息 -->
59          <developer>
60              <!--SCM里项目开发者的唯一标识符 -->
61              <id>HELLO WORLD</id>
62              <!--项目开发者的全名 -->
63              <name>youname</name>
64              <!--项目开发者的email -->
65              <email>youname@qq.com</email>
66              <!--项目开发者的主页的URL -->
67              <url />
68              <!--项目开发者在项目中扮演的角色，角色元素描述了各种角色 -->
69              <roles>
70                  <role>Project Manager</role>
71                  <role>Architect</role>
72              </roles>
73              <!--项目开发者所属组织 -->
74              <organization>demo</organization>
75              <!--项目开发者所属组织的URL -->
76              <organizationUrl>http://www.xxx.com/</organizationUrl>
77              <!--项目开发者属性，如即时消息如何处理等 -->
78              <properties>
79                  <dept>No</dept>
80              </properties>
81              <!--项目开发者所在时区， -11到12范围内的整数。 -->
82              <timezone>+8</timezone>
```

```
        </developer>
    </developers>
    <!--项目的其他贡献者列表 -->
    <contributors>
        <!--项目的其他贡献者。参见developers/developer元素 -->
        <contributor>
            <name />
            <email />
            <url />
            <organization />
            <organizationUrl />
            <roles />
            <timezone />
            <properties />
        </contributor>
    </contributors>
    <!--该元素描述了项目所有License列表。 应该只列出该项目的license列表，不要列出依赖项目的license列表。如果列出多个license，用户可以选择它们中的一个而不是接受所有license。 -->
    <licenses>
        <!--描述了项目的license，用于生成项目的web站点的license页面，其他一些报表和validation也会用到该元素。 -->
        <license>
            <!--license用于法律上的名称 -->
            <name>Apache 2</name>
            <!--官方的license正文页面的URL -->
            <url>http://www.xxxx.com/LICENSE-2.0.txt</url>
            <!--项目分发的主要方式： repo，可以从Maven库下载 manual， 用户必须手动下载和安装依赖 -->
            <distribution>repo</distribution>
            <!--关于license的补充信息 -->
            <comments>A business-friendly OSS license</comments>
        </license>
    </licenses>
    <!--SCM(Source Control Management)标签允许你配置你的代码库，供Maven web站点和其它插件使用。 -->
    <scm>
        <!--SCM的URL,该URL描述了版本库和如何连接到版本库。欲知详情，请看SCMs提供的URL格式和列表。该连接只读。 -->
        <connection>
            scm:svn:http://svn.xxxx.com/maven/xxxxx-maven2-trunk(dao-trunk)
        </connection>
        <!--给开发者使用的，类似connection元素。即该连接不仅仅只读 -->
        <developerConnection>
            scm:svn:http://svn.xxxx.com/maven/dao-trunk
        </developerConnection>
        <!--当前代码的标签，在开发阶段默认为HEAD -->
        <tag />
        <!--指向项目的可浏览SCM库（例如ViewVC或者Fisheye）的URL。 -->
        <url>http://svn.xxxxx.com/</url>
    </scm>
    <!--描述项目所属组织的各种属性。Maven产生的文档用 -->
```

```
    <organization>
        <!--组织的全名 -->
        <name>demo</name>
        <!--组织主页的URL -->
        <url>http://www.xxxxxx.com/</url>
    </organization>



    <!--
    2、项目构建环境配置

    -->
    <!--描述了这个项目构建环境中的前提条件。 -->
    <prerequisites>
        <!--构建该项目或使用该插件所需要的Maven的最低版本 -->
        <maven />
    </prerequisites>
    <!--项目的问题管理系统(Bugzilla, Jira, Scarab,或任何你喜欢的问题管理系统)的名称和URL，本例为 jira -->
    <issueManagement>
        <!--问题管理系统（例如jira）的名字， -->
        <system>jira</system>
        <!--该项目使用的问题管理系统的URL -->
        <url>http://jira.xxxx.com/xxxx</url>
    </issueManagement>
    <!--项目持续集成信息 -->
    <ciManagement>
        <!--持续集成系统的名字，例如continuum -->
        <system />
        <!--该项目使用的持续集成系统的URL（如果持续集成系统有web接口的话）。 -->
        <url />
        <!--构建完成时，需要通知的开发者/用户的配置项。包括被通知者信息和通知条件（错误，失败，成功，警告） -->
        <notifiers>
            <!--配置一种方式，当构建中断时，以该方式通知用户/开发者 -->
            <notifier>
                <!--传送通知的途径 -->
                <type />
                <!--发生错误时是否通知 -->
                <sendOnError />
                <!--构建失败时是否通知 -->
                <sendOnFailure />
                <!--构建成功时是否通知 -->
                <sendOnSuccess />
                <!--发生警告时是否通知 -->
                <sendOnWarning />
                <!--不赞成使用。通知发送到哪里 -->
                <address />
                <!--扩展配置项 -->
```

```
                <configuration />
            </notifier>
        </notifiers>
    </ciManagement>
    <!--模块（有时称作子项目） 被构建成项目的一部分。列出的每个模块元素是指向该模块的目录的相对路径 -->
    <modules />
    <!--构建项目需要的信息 -->
    <build>
        <!--该元素设置了项目源码目录，当构建项目的时候，构建系统会编译目录里的源码。该路径是相对于pom.xml的相对路径。 -->
        <sourceDirectory />
        <!--该元素设置了项目脚本源码目录，该目录和源码目录不同：绝大多数情况下，该目录下的内容 会被拷贝到输出目录(因为脚本是被解释的，而不是被编译的)。 -->
        <scriptSourceDirectory />
        <!--该元素设置了项目单元测试使用的源码目录，当测试项目的时候，构建系统会编译目录里的源码。该路径是相对于pom.xml的相对路径。 -->
        <testSourceDirectory />
        <!--被编译过的应用程序class文件存放的目录。 -->
        <outputDirectory />
        <!--被编译过的测试class文件存放的目录。 -->
        <testOutputDirectory />
        <!--使用来自该项目的一系列构建扩展 -->
        <extensions>
            <!--描述使用到的构建扩展。 -->
            <extension>
                <!--构建扩展的groupId -->
                <groupId />
                <!--构建扩展的artifactId -->
                <artifactId />
                <!--构建扩展的版本 -->
                <version />
            </extension>
        </extensions>
        <!--当项目没有规定目标（Maven2 叫做阶段）时的默认值 -->
        <defaultGoal />
        <!--这个元素描述了项目相关的所有资源路径列表，例如和项目相关的属性文件，这些资源被包含在最终的打包文件里。 -->
        <resources>
            <!--这个元素描述了项目相关或测试相关的所有资源路径 -->
            <resource>
                <!--描述了资源的目标路径。该路径相对target/classes目录（例如${project.build.outputDirectory}）。举个例子，如果你想资源在特定的包里(org.apache.maven.messages)，你就必须该元素设置为org/apache/maven/messages。然而，如果你只是想把资源放到源码目录结构里，就不需要该配置。 -->
                <targetPath />
                <!--是否使用参数值代替参数名。参数值取自properties元素或者文件里配置的属性，文件在filters元素里列出。 -->
                <filtering />
                <!--描述存放资源的目录，该路径相对POM路径 -->
```

```
                <directory />
                <!--包含的模式列表，例如**/*.xml. -->
                <includes />
                <!--排除的模式列表，例如**/*.xml -->
                <excludes />
            </resource>
        </resources>
        <!--这个元素描述了单元测试相关的所有资源路径，例如和单元测试相关的属性文件。 -->
        <testResources>
            <!--这个元素描述了测试相关的所有资源路径，参见build/resources/resource元素的
说明 -->
            <testResource>
                <targetPath />
                <filtering />
                <directory />
                <includes />
                <excludes />
            </testResource>
        </testResources>
        <!--构建产生的所有文件存放的目录 -->
        <directory />
        <!--产生的构件的文件名，默认值是${artifactId}-${version}。 -->
        <finalName />
        <!--当filtering开关打开时，使用到的过滤器属性文件列表 -->
        <filters />
        <!--子项目可以引用的默认插件信息。该插件配置项直到被引用时才会被解析或绑定到生命周
期。给定插件的任何本地配置都会覆盖这里的配置 -->
        <pluginManagement>
            <!--使用的插件列表 。 -->
            <plugins>
                <!--plugin元素包含描述插件所需要的信息。 -->
                <plugin>
                    <!--插件在仓库里的group ID -->
                    <groupId />
                    <!--插件在仓库里的artifact ID -->
```

```
                    <artifactId />
                    <!--被使用的插件的版本（或版本范围） -->
                    <version />
                    <!--是否从该插件下载Maven扩展（例如打包和类型处理器），由于性能原因，只有在真需要下载时，该元素才被设置成enabled。 -->
                    <extensions />
                    <!--在构建生命周期中执行一组目标的配置。每个目标可能有不同的配置。 -->
                    <executions>
                        <!--execution元素包含了插件执行需要的信息 -->
                        <execution>
                            <!--执行目标的标识符，用于标识构建过程中的目标，或者匹配继承过程中需要合并的执行目标 -->
                            <id />
                            <!--绑定了目标的构建生命周期阶段，如果省略，目标会被绑定到源数据里配置的默认阶段 -->
                            <phase />
                            <!--配置的执行目标 -->
                            <goals />
                            <!--配置是否被传播到子POM -->
                            <inherited />
                            <!--作为DOM对象的配置 -->
                            <configuration />
                        </execution>
                    </executions>
                    <!--项目引入插件所需要的额外依赖 -->
                    <dependencies>
                        <!--参见dependencies/dependency元素 -->
                        <dependency>......</dependency>
                    </dependencies>
                    <!--任何配置是否被传播到子项目 -->
                    <inherited />
                    <!--作为DOM对象的配置 -->
                    <configuration />
                </plugin>
            </plugins>
        </pluginManagement>
        <!--使用的插件列表 -->
        <plugins>
            <!--参见build/pluginManagement/plugins/plugin元素 -->
            <plugin>
                <groupId />
                <artifactId />
                <version />
                <extensions />
                <executions>
                    <execution>
                        <id />
                        <phase />
                        <goals />
                        <inherited />
```

```
                        <configuration />
                    </execution>
                </executions>
                <dependencies>
                    <!--参见dependencies/dependency元素 -->
                    <dependency>......</dependency>
                </dependencies>
                <goals />
                <inherited />
                <configuration />
            </plugin>
        </plugins>
    </build>
    <!--在列的项目构建profile，如果被激活，会修改构建处理 -->
    <profiles>
        <!--根据环境参数或命令行参数激活某个构建处理 -->
        <profile>
            <!--构建配置的唯一标识符。即用于命令行激活，也用于在继承时合并具有相同标识符的profile。 -->
            <id />
            <!--自动触发profile的条件逻辑。Activation是profile的开启钥匙。profile的力量来自于它 能够在某些特定的环境中自动使用某些特定的值；这些环境通过activation元素指定。activation元素并不是激活profile的唯一方式。 -->
            <activation>
                <!--profile默认是否激活的标志 -->
                <activeByDefault />
                <!--当匹配的jdk被检测到，profile被激活。例如，1.4激活JDK1.4，1.4.0_2，而!1.4激活所有版本不是以1.4开头的JDK。 -->
                <jdk />
                <!--当匹配的操作系统属性被检测到，profile被激活。os元素可以定义一些操作系统相关的属性。 -->
                <os>
                    <!--激活profile的操作系统的名字 -->
                    <name>Windows XP</name>
                    <!--激活profile的操作系统所属家族(如 'windows') -->
                    <family>Windows</family>
                    <!--激活profile的操作系统体系结构 -->
                    <arch>x64</arch>
                    <!--激活profile的操作系统版本 -->
                    <version>6.1.7100</version>
                </os>
                <!--如果Maven检测到某一个属性（其值可以在POM中通过${名称}引用），其拥有对应的名称和值，Profile就会被激活。如果值 字段是空的，那么存在属性名称字段就会激活profile，否则按区分大小写方式匹配属性值字段 -->
                <property>
                    <!--激活profile的属性的名称 -->
                    <name>mavenVersion</name>
                    <!--激活profile的属性的值 -->
                    <value>2.0.3</value>
```

```
                </property>
                <!--提供一个文件名，通过检测该文件的存在或不存在来激活profile。missing检查文件是否存在，如果不存在则激活 profile。另一方面，exists则会检查文件是否存在，如果存在则激活profile。 -->
                <file>
                    <!--如果指定的文件存在，则激活profile。 -->
                    <exists>/usr/local/xxxx/xxxx-home/tomcat/maven-guide-zh-to-production/workspace/
                    </exists>
                    <!--如果指定的文件不存在，则激活profile。 -->
                    <missing>/usr/local/xxxx/xxxx-home/tomcat/maven-guide-zh-to-production/workspace/
                    </missing>
                </file>
            </activation>
            <!--构建项目所需要的信息。参见build元素 -->
            <build>
                <defaultGoal />
                <resources>
                    <resource>
                        <targetPath />
                        <filtering />
                        <directory />
                        <includes />
                        <excludes />
                    </resource>
                </resources>
                <testResources>
                    <testResource>
                        <targetPath />
                        <filtering />
                        <directory />
                        <includes />
                        <excludes />
                    </testResource>
                </testResources>
                <directory />
                <finalName />
                <filters />
                <pluginManagement>
                    <plugins>
                        <!--参见build/pluginManagement/plugins/plugin元素 -->
                        <plugin>
                            <groupId />
                            <artifactId />
                            <version />
                            <extensions />
                            <executions>
                                <execution>
```

```
                                    <id />
                                    <phase />
                                    <goals />
                                    <inherited />
                                    <configuration />
                                </execution>
                            </executions>
                            <dependencies>
                                <!--参见dependencies/dependency元素 -->
                                <dependency>......</dependency>
                            </dependencies>
                            <goals />
                            <inherited />
                            <configuration />
                        </plugin>
                    </plugins>
                </pluginManagement>
                <plugins>
                    <!--参见build/pluginManagement/plugins/plugin元素 -->
                    <plugin>
                        <groupId />
                        <artifactId />
                        <version />
                        <extensions />
                        <executions>
                            <execution>
                                <id />
                                <phase />
                                <goals />
                                <inherited />
                                <configuration />
                            </execution>
                        </executions>
                        <dependencies>
                            <!--参见dependencies/dependency元素 -->
                            <dependency>......</dependency>
                        </dependencies>
                        <goals />
                        <inherited />
                        <configuration />
                    </plugin>
                </plugins>
            </build>
            <!--模块（有时称作子项目） 被构建成项目的一部分。列出的每个模块元素是指向该模块
的目录的相对路径 -->
            <modules />
            <!--发现依赖和扩展的远程仓库列表。 -->
            <repositories>
                <!--参见repositories/repository元素 -->
                <repository>
                    <releases>
```

```
                        <enabled />
                        <updatePolicy />
                        <checksumPolicy />
                    </releases>
                    <snapshots>
                        <enabled />
                        <updatePolicy />
                        <checksumPolicy />
                    </snapshots>
                    <id />
                    <name />
                    <url />
                    <layout />
                </repository>
            </repositories>
            <!--发现插件的远程仓库列表，这些插件用于构建和报表 -->
            <pluginRepositories>
                <!--包含需要连接到远程插件仓库的信息.参见repositories/repository元素
    -->
                <pluginRepository>
                    <releases>
                        <enabled />
                        <updatePolicy />
                        <checksumPolicy />
                    </releases>
                    <snapshots>
                        <enabled />
                        <updatePolicy />
                        <checksumPolicy />
                    </snapshots>
                    <id />
                    <name />
                    <url />
                    <layout />
                </pluginRepository>
            </pluginRepositories>
            <!--该元素描述了项目相关的所有依赖。 这些依赖组成了项目构建过程中的一个个环节。
    它们自动从项目定义的仓库中下载。要获取更多信息，请看项目依赖机制。 -->
            <dependencies>
                <!--参见dependencies/dependency元素 -->
                <dependency>......</dependency>
            </dependencies>
            <!--不赞成使用. 现在Maven忽略该元素. -->
            <reports />
            <!--该元素包括使用报表插件产生报表的规范。当用户执行“mvn site”，这些报表就会运
    行。 在页面导航栏能看到所有报表的链接。参见reporting元素 -->
            <reporting>......</reporting>
            <!--参见dependencyManagement元素 -->
            <dependencyManagement>
                <dependencies>
                    <!--参见dependencies/dependency元素 -->
```

```
                    <dependency>......</dependency>
                </dependencies>
            </dependencyManagement>
            <!--参见distributionManagement元素 -->
            <distributionManagement>......</distributionManagement>
            <!--参见properties元素 -->
            <properties />
        </profile>
    </profiles>


    <!--
    3、项目仓库管理配置

    -->
    <!--发现依赖和扩展的远程仓库列表。 -->
    <repositories>
        <!--包含需要连接到远程仓库的信息 -->
        <repository>
            <!--如何处理远程仓库里发布版本的下载 -->
            <releases>
                <!--true或者false表示该仓库是否为下载某种类型构件（发布版，快照版）开启。
-->
                <enabled />
                <!--该元素指定更新发生的频率。Maven会比较本地POM和远程POM的时间戳。这里的
选项是：always（一直），daily（默认，每日），interval：X（这里X是以分钟为单位的时间间隔），
或者never（从不）。 -->
                <updatePolicy />
                <!--当Maven验证构件校验文件失败时该怎么做：ignore（忽略），fail（失败），
或者warn（警告）。 -->
                <checksumPolicy />
            </releases>
            <!--如何处理远程仓库里快照版本的下载。有了releases和snapshots这两组配置，POM
就可以在每个单独的仓库中，为每种类型的构件采取不同的策略。例如，可能有人会决定只为开发目的开启
对快照版本下载的支持。参见repositories/repository/releases元素 -->
            <snapshots>
                <enabled />
                <updatePolicy />
                <checksumPolicy />
            </snapshots>
            <!--远程仓库唯一标识符。可以用来匹配在settings.xml文件里配置的远程仓库 -->
            <id>banseon-repository-proxy</id>
            <!--远程仓库名称 -->
            <name>banseon-repository-proxy</name>
            <!--远程仓库URL，按protocol://hostname/path形式 -->
            <url>http://10.10.10.123:8080/repository/</url>
            <!--用于定位和排序构件的仓库布局类型-可以是default（默认）或者legacy（遗留）。
Maven 2为其仓库提供了一个默认的布局；然而，Maven
                1.x有一种不同的布局。我们可以使用该元素指定布局是default（默认）还是
legacy（遗留）。 -->
            <layout>default</layout>
```

```
        </repository>
    </repositories>

    <!--发现插件的远程仓库列表，这些插件用于构建和报表 -->
    <pluginRepositories>
        <!--包含需要连接到远程插件仓库的信息.参见repositories/repository元素 -->
        <pluginRepository>......</pluginRepository>
    </pluginRepositories>

    <!--
    4、项目依赖管理配置

    -->
    <!--继承自该项目的所有子项目的默认依赖信息。这部分的依赖信息不会被立即解析,而是当子项目声明一个依赖（必须描述group ID和artifact
        ID信息），如果group ID和artifact ID以外的一些信息没有描述，则通过group ID和artifact ID匹配到这里的依赖，并使用这里的依赖信息。 -->
    <dependencyManagement>
        <dependencies>
```