Maven2的配置文件settings.xml - Yakov

简介：概览当Maven运行过程中的各种配置，例如pom.xml，不想绑定到一个固定的project或者要分配给用户时，我们使用settings.xml中的settings元素来确定这些配置。这包含了本地仓库位置，远程仓库服务器以及认证信息等。settings.xml存在于两个地方：1.安装的地方：$M2_HOME/conf/settings.xml2.用户的目录：${user.home}/.m2/settings.xml前者又被叫做全局配置，后者被称为用户配置。如果两者都存在，它们的内容将被合并，并且用户范围的settings.xml优先。如果你偶尔需要创建用户范围的settings，你可以简单的copy Maven安装路径下的settings到目录${user.home}/.m2。Maven默认的settings.xml是一个包含了注释和例子的模板，你可以快速的修改它来达到你的要求。下面是settings下的顶层元素的一个概览：

```
 1 <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
 2           xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
 3
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
 4               http://maven.apache.org/xsd/settings-
1.0.0.xsd">
 5     <localRepository/>
 6     <interactiveMode/>
 7     <usePluginRegistry/>
 8     <offline/>
 9     <pluginGroups/>
10     <servers/>
11     <mirrors/>
12     <proxies/>
13     <profiles/>
14     <activeProfiles/>
15 </settings>
```

settings的内容可以在下面这些地方篡改：1.${user.home}和所有其他的系统属性

2.${env.HOME}等环境变量注意：settins.xml中profiles下定义的属性不能被篡改。配置细节：简单的值一半以上的顶级settings元素师简单的值，代表了一直处于活跃的构建系统的元素的取值范围。

```
 1 <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
 2           xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
 3
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
 4             http://maven.apache.org/xsd/settings-
1.0.0.xsd">
 5
<localRepository>${user.home}/.m2/repository</localRepositor
y>
 6     <interactiveMode>true</interactiveMode>
 7     <usePluginRegistry>false</usePluginRegistry>
 8     <offline>false</offline>
 9     ...
10 </settings>
```

localRepository：这个值是构建系统的本地仓库的路径。默认的值是${user.home}/.m2/repository.如果一个系统想让所有登陆的用户都用同一个本地仓库的话，这个值是极其有用的。interactiveMode：如果Maven要试图与用户交互来得到输入就设置为true，否则就设置为false，默认为true。usePluginRegistry：如果Maven使用${user.home}/.m2/plugin-registry.xml来管理plugin的版本，就设置为true，默认为false。offline：如果构建系统要在离线模式下工作，设置为true，默认为false。如果构建服务器因为网络故障或者安全问题不能与远程仓库相连，那么这个设置是非常有用的。插件组这个元素包含了一系列pluginGroup元素，每个又包含了一个groupId。当一个plugin被使用，而它的groupId没有被提供的时候，这个列表将被搜索。这个列表自动的包含了org.apache.maven.plugins和org.codehaus.mojo。

```
 1 <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
```

```
 2          xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
 3
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
 4            http://maven.apache.org/xsd/settings-
1.0.0.xsd">
 5     ...
 6     <pluginGroups>
 7         <pluginGroup>org.mortbay.jetty</pluginGroup>
 8     </pluginGroups>
 9     ...
10 </settings>
```

例如，有了上面的配置，Maven命令行可以使用简单的命令执行org.morbay.jetty:jetty-maven-plugin:run，如下mvn jetty run服务器用来下载和部署的仓库是用POM中的repositories和distributionManagement元素来定义的。但是某些配置例如username和password就不应该随着pom.xml来分配了。这种类型的信息应该保存在构建服务器中的settings.xml中。

```
 1 <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
 2          xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
 3
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
 4            http://maven.apache.org/xsd/settings-
1.0.0.xsd">
 5     ...
 6     <servers>
 7     <server>
 8     <id>server001</id>
 9     <username>my_login</username>
10     <password>my_password</password>
11     <privateKey>${user.home}/.ssh/id_dsa</privateKey>
12     <passphrase>some_passphrase</passphrase>
```

```
13     <filePermissions>664</filePermissions>
14     <directoryPermissions>775</directoryPermissions>
15     <configuration></configuration>
16     </server>
17     </servers>
18     ...
19 </settings>
```

id：这是Server的ID(不是登录进来的user)，与Maven想要连接上的repository/mirror中的id元素相匹配。username，password：这两个元素成对出现，表示连接这个server需要验证username和password。privateKey，passphrase：与前两个元素一样，这两个成对出现，分别指向了一个私钥(默认的是${user.home}/.ssh/id_dsa)和一个passphrase。passphrase和password元素可能在将来被客观化，但是现在必须以文本形式在settings.xml中设置。filePermissions，directoryPermissions：当一个仓库文件或者目录在部署阶段被创建的时候，就必须用到权限许可。他们合法的值是三个数字，就像*nix中的文件权限，例如：664，775.注意：如果你使用了一个私钥来登录server，那么password元素必须被省略，否则私钥将被忽视。密码加密一个新特征：服务器password和passphrase加密已经被升到2.1.0+镜像

```
 1 <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
 2           xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
 3
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
 4             http://maven.apache.org/xsd/settings-
1.0.0.xsd">
 5     ...
 6     <mirrors>
 7         <mirror>
 8             <id>planetmirror.com</id>
 9             <name>PlanetMirror Australia</name>
10
<url>http://downloads.planetmirror.com/pub/maven2</url>
```

```
11             <mirrorOf>central</mirrorOf>
12         </mirror>
13     </mirrors>
14     ...
15 </settings>
```

id，name：唯一的镜像标识和用户友好的镜像名称。id被用来区分mirror元素，并且当连接时候被用来获得相应的证书。url：镜像基本的URL，构建系统敬将使用这个URL来连接仓库，而不是原来的仓库URL。mirrorOf：镜像所包含的仓库的Id。例如，指向Maven central仓库的镜像(http://repo1.maven.org/maven2/)，设置这个元素为central。更多的高级映射例如repo1,repo2 或者*,!inhouse都是可以的。没必要一定和mirror的id相匹配。代理

```
 1 <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
 2           xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
 3
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
 4             http://maven.apache.org/xsd/settings-
1.0.0.xsd">
 5     ...
 6     <proxies>
 7         <proxy>
 8             <id>myproxy</id>
 9             <active>true</active>
10             <protocol>http</protocol>
11             <host>proxy.somewhere.com</host>
12             <port>8080</port>
13             <username>proxyuser</username>
14             <password>somepassword</password>
15
<nonProxyHosts>*.google.com|ibiblio.org</nonProxyHosts>
16         </proxy>
17     </proxies>
```

```
18     ...
19 </settings>
```

id：proxy的唯一标识，用来区别proxy元素。active：当proxy被激活的时候为true。当申明的代理很多的时候，这个很有用，但是同一时间仅有一个被激活。protocol，host，port：代理地址protocol://host:port的分散形式。username，password：两个元素成对出现，提供连接proxy服务器时的认证。nonProxyHosts：这里列出了不需要使用代理的hosts。列表的分隔符是proxy服务器想要的类型。上面例子使用了pipe分隔符，逗号分隔符也比较通用。配置文件settings.xml中的profile是pom.xml中的profile的简洁形式。它包含了激活(activation)，仓库(repositories)，插件仓库(pluginRepositories)和属性(properties)元素。profile元素仅包含这四个元素是因为他们涉及到整个的构建系统，而不是个别的POM配置。如果settings中的profile被激活，那么它的值将重载POM或者profiles.xml中的任何相等ID的profiles。激活(activation)activations是profile的关键，就像POM中的profiles，profile的能力在于它在特定情况下可以修改一些值。而这些情况是通过activation来指定的。

```
 1 <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
 2           xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
 3
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
 4             http://maven.apache.org/xsd/settings-
1.0.0.xsd">
 5     ...
 6     <profiles>
 7         <profile>
 8             <id>test</id>
 9             <activation>
10                 <activeByDefault>false</activeByDefault>
11                 <jdk>1.5</jdk>
12                 <os>
13                     <name>Windows XP</name>
14                     <family>Windows</family>
```

```
15                      <arch>x86</arch>
16                      <version>5.1.2600</version>
17                  </os>
18                  <property>
19                      <name>mavenVersion</name>
20                      <value>2.0.3</value>
21                  </property>
22                  <file>
23
<exists>${basedir}/file2.properties</exists>
24
<missing>${basedir}/file1.properties</missing>
25                  </file>
26              </activation>
27              ...
28          </profile>
29      </profiles>
30      ...
31 </settings>
```

如果所有指定的条件都达到了，那么，activation就被触发，而且不需要一次性全部达到。jdk：在jdk元素中，activation有一个内建的，java版本检测。如果检测到jdk版本与期待的一样，那么就激活。在上面的例子中，1.5.0_06是满足的。os：os元素可以定义一些上面所示的操作系统特定的属性。property：如果Maven检测到相应的名值对的属性，那么，这个profile将被激活。file：如果给定的文件存在，或者不存在那么将激活这个profile。activation并不是唯一激活profile的途径。settings.xml中的activeProfile包含了profile的id。他们也可以通过命令行来显式的激活，例如-P test。如果你想查看在一个构建过程中有哪些profile会被激活。就使用maven-help-pluginmvn help:active-profiles属性(properites)Maven的属性是值占位符，就像Ant中的属性。如果X是一个属性的话，那么它的值在POM中可以使用${X}来进行任意地方的访问。他们来自于五种不同的风格，所有都可以从settings.xml文件中访问到。1.env.X：使用“env.”前缀将会返回当前的环境变量。例如${env.PATH}就是使用了$path环境变量。2.project.X：一个点“.”分割的路径，在POM中就是相关的元素的值。例如：<project><version>1.0</version></project>就可以通过${project.version}来访

问。3.settings.X：一个点“.”分割的路径，在settings.xml中就是相对应的元素的值，例如：<settings><offline>false</offline></settings>就可以通过${settings.offline}来访问。4.Java系统属性：所有通过java.lang.System.getProperties()来访问的属性都可以像POM中的属性一样访问，例如：${java.home}5.X：被<properties/>或者外部文件定义的属性，值可以这样访问${someVar}

```
 1 <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
 2           xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
 3
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
 4             http://maven.apache.org/xsd/settings-
1.0.0.xsd">
 5     ...
 6     <profiles>
 7         <profile>
 8             ...
 9             <properties>
10                 <user.install>${user.home}/our-
project</user.install>
11             </properties>
12             ...
13         </profile>
14     </profiles>
15     ...
16 </settings>
```

如果这个profile被激活，那么属性${user.install}就可以被访问了。仓库(repositories)仓库是Maven用来构筑构建系统的本地仓库的远程项目集合。它来自于被Maven叫做插件和依赖的本地仓库。不同的远程仓库包含不同的项目，当profile被激活，他们就会需找匹配的release或者snapshot构件。

```
 1 <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
```

```
 2          xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
 3
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
 4            http://maven.apache.org/xsd/settings-
1.0.0.xsd">
 5    ...
 6    <profiles>
 7        <profile>
 8        ...
 9        <repositories>
10            <repository>
11                <id>codehausSnapshots</id>
12                <name>Codehaus Snapshots</name>
13                <releases>
14                    <enabled>false</enabled>
15                    <updatePolicy>always</updatePolicy>
16                    <checksumPolicy>warn</checksumPolicy>
17                </releases>
18                <snapshots>
19                    <enabled>true</enabled>
20                    <updatePolicy>never</updatePolicy>
21                    <checksumPolicy>fail</checksumPolicy>
22                </snapshots>
23
<url>http://snapshots.maven.codehaus.org/maven2</url>
24                <layout>default</layout>
25            </repository>
26        </repositories>
27        <pluginRepositories>
28        ...
29        </pluginRepositories>
30        ...
31        </profile>
32    </profiles>
33    ...
```

```
34 </settings>
```

releases，snapshots：这是各种构件的策略，release或者snapshot。因了这两个集合，POM可以在单个的仓库中不依赖于另外一个的策略而改变当前策略。例如：一个人可能只下载snapshot用来开发。enable：true或者false，来标记仓库是否为各自的类型激活(release 或者 snapshot)。updatePolicy：这个元素指明了更新的频率。Maven会比较本地POM与远程的时间戳。可选的项目为：always，daily，interval:X，nerver。checksumPolicy：当Maven向仓库部署文件的时候，它也部署了相应的校验和文件。可选的为：ignore，fail，warn，或者不正确的校验和。layout：在上面描述仓库的时候，我们提到他们有统一的布局。这完全正确。使用这个来表明它是default还是legacy。插件仓库(plugin repositories)仓库包含了两种重要类型的构件。第一种是用来做其他构件依赖的构件，这是在中央仓库中的大多数插件。另外一种类型的构件就是插件。Maven的插件本身就是一种特殊的构件。因此，插件仓库被从其他仓库中分离出来。无论怎么说，pluginRepositories元素模块的结构与repositories模块很相似。pluginRepository元素指向一个可以找到新插件的远程地址。**激活配置(Active Profiles)**

```
1 <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
2           xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
3
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
4             http://maven.apache.org/xsd/settings-
1.0.0.xsd">
5     ...
6     <activeProfiles>
7         <activeProfile>env-test</activeProfile>
8     </activeProfiles>
9 </settings>
```

settings.xml最后一个谜题是activeProfiles元素。它包含一系列的activeProfile元素，每个都有一个profile id的值，任何profile id被定义到activeProfile的profile将被激活，不

管其他的环境设置怎么样。如果没有匹配的profile被找到，那么就什么事情也不做。例如：如果env-test是一个activeProfile，一个在pom.xml或者profile.xml中的具有相应id的profile将被激活。如果没有这样的profile被找到，就什么事也不做，一切照常。

原文地址：http://maven.apache.org/settings.html