MAVEN 版本管理

作者:yinhongzhan@gmail.com

概要

为了方便团队合作,在项目开发的过程中,大家都应该使用快照版本,Maven 能很智能的处理这种特殊的版本,解析项目各个模块的最新的"快照".快照版本能促进团队内部交流,但是项目需要对外发布时,我们显然要提供非常稳定的版本,使用该版本应当永远只能够定位到唯一的构件,而不是像快照版本一样,定位构件随时可能发生变化. 当我们发布一个项目正式版本后,就可以进入下一个开发阶段,项目也就自然的转化到新的快照版本中.

项目版本管理关心的问题之一就是上面提到的"快照版"和"正式发布版"之间的切换.

首先来说明下理想的发布状态:

- ✧ 所有自动化测试全部通过
- ✧ 项目内部没有任何的快照版本依赖
- ✧ 项目中没有配置任何快照版本的插件
- ✧ 项目的代码应该已经全部提交到版本控制系统中

理论上来说应该满足以上几点才能发布版本.

版本号定义约定

在使用一些开源框架时,大家都会看到很多版本号,如 springfrxxxx3.1.1,sprngfrxxxx3.0.1,junit3.8 等

Maven 中的版本号约定是这样的:

<主版本>.<次版本>.<增量版本>-<里程碑版本>

可以看到在除了里程碑版本外其他版本之间使用点号分割的.下面来说明下每种的意义:

主版本:

表示项目的重大架构变更. 例如 Maven2 和 Maven1 相去甚远;Struts1 和 2 则更是采用了不用的架构.JUnit4 比 JUnit3 增加了注解标记

次版本:

表示较大范围的功能增加和变化,及 BUG 修复.但从总体架构上来说没有什么变化.

增量版本:

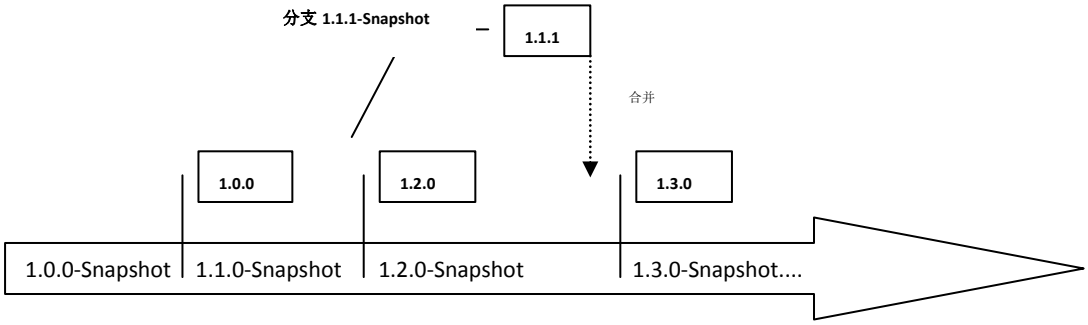
一般表示重大的 BUG 的修复.例如项目发布了 1.0.0 版本后发现了一个影响功能的重大 BUG,则应快速修复并发布为 1.0.1

里程碑版本:

这些版本往往表示还不是非常稳定,需要很多测试.如 1.0.0-beta1,1.0.0-alpha1 等

我们的使用版本号也按照此约定来进行.

主干、标签和分支



画了一个图方便大家理解,我举个例子来说明这几个概念,假如某个项目要开始开发,这时候的版本即是 1.0.0-Snapshot 状态,经过大家一个月的努力,功能全部完成了,这时候可以发布一个版本 1.0.0,然后项目就进入了 1.1.0-Snapshot 状态,添加了一些新特性和修复了很多 BUG 后团队很开心的宣布 1.2.0 诞生了,但是过了几天后,测试部反映上来了一个重大的 BUG,需要尽快修复,而这时候开发们也早已经进入了 1.2.0-Snapshot 状态了,这时候我们既不能停止 1.2.0-Snapshot 的开发,又由于主干的变数太多,这时候我们就可以创建一个 1.1.1-Snapshot 分支,在这里进行 BUG 的修复,然后为用户发布一个 1.1.1 的增量版本,同时打上标签,当然不能忘了把 BUG 修复设计的变更合并到 1.2.0-Snapshot 中,主干的 1.2.0-Snapshot 在开发一段时间后发布了 1.2.0 版本,然后进入了 1.3.0-Snapshot 的开发过程中.

通过上面的这个例子大家也应该明白了这个 3 个概念的意义,这也是一个不成文的行业规定,对大家学习开源框架也比较好

自动化发布

前面说的内容主要是为了方便大家理解后面所述.所以现在才是重头戏.

在上面介绍了一下标准的发布流程,作为一个程序员当然会希望如何自动化实现这一功能,所幸的是 Maven 的插件 Maven Release Plugin 提供了这样的功能.

该插件主要有 3 种命令:release:prepare,release:rollback,release:perform.从字面的意思大家就可以猜想到每条命令的作用.现在来大致介绍下:

大家回想下前面讲到的一个版本需要成为正式发布版有四个条件,其中命令 **release:prepare** 准备阶段首先做的事就是和这个相关:

- 检查项目是否有未提交的代码
- 检查项目是否有快照版本的依赖
- 根据用户的输入将快照版本升级为发布版(1.0.0-Snapshot-------> 1.0.0)
- 将 POM 中的 SCM 信息更新为标签地址
- 基于修改后的 POM 执行 MAVEN 构建
- 提交 POM 变更
- 基于用户输入为代码打标签
- 将代码从发布版升级为新的快照版
- 提交 POM 变更

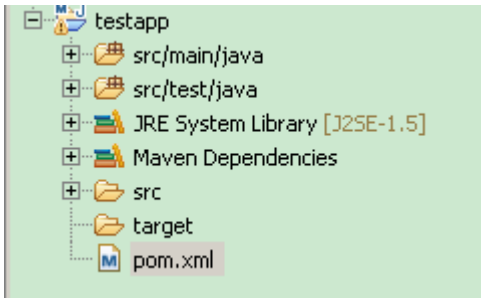
release:rollback:该步骤主要用来回退 prepare 所执行的操作,但是由 prepare 生成的标签他不会删除还是需要我们自己去主动删除的

release:perform:执行版本发布.迁出 renlease:prepare 生成的标签中的源码,并在此基础上执

行 mvn deploy 命令并打包部署构件至仓库

再介绍了上面的一些概念后我们进入实战,这边我会边举例边说:

根据上面的图我们首先在svn目录下面建立2个目录一个名为tags,一个名为trunk前一个用来存放标签的信息,后一个则是主干存放程序所有代码.现在我们新建一个名为testapp的maven工程,如下:



Pom 文件里的内容非常简单只引入了一个 junit3.8.1, 现在我们将它改造使其支持自动和 SVN 结合发布版本,并上传至我们的 MAVEN 私服.这里我就省略 setting.xml 的配置了这个文件主要是要加上你的有权限部署到私服的账号和密码,以及配置镜像.

先加上 SCM

```
<scm>
    <connection>scm:svn:http://10.0.2.143/svn/infos/trunk/testapp</connection>
    <developerConnection>scm:svn:http://10.0.2.143/svn/infos/trunk/testapp</developerConnection>
    <url>http://10.0.2.143/svn/infos/trunk/testapp</url>
</scm>
```

SCM 的作用主要是告诉 Maven 主干在哪里,我们还要告诉他标签的位置,并配置 Maven Release Plugin 插件,如下:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-release-plugin</artifactId>
    <version>2.2.2</version>
    <configuration>
        <tagBase>http://10.0.2.143/svn/infos/tags/</tagBase>
        <releaseProfiles>release</releaseProfiles>
    </configuration>
</plugin>
```

再加上我们私服部署的位置:

```
<distributionManagement>
    <repository>
        <id>HEXINB2B_RELEASE</id>
        <name>HEXINB2B_RELEASE</name>
        <url>http://10.0.2.143:8081/nexus/content/repositories/HEXINB2B_RELEASE/</url>
    </repository>
    <snapshotRepository>
        <id>HEXINB2B_Snapshots</id>
        <name>HEXINB2B_Snapshots</name>
        <url>http://10.0.2.143:8081/nexus/content/repositories/HEXINB2B_Snapshots/</url>
    </snapshotRepository>
</distributionManagement>
```

到这里我们就可以开始执行 release:prepare 命令了,但是还有一点要注意的是系统要提供 SVN 命令工具.上传 testapp 上传至 SVN 的 trunk 下,请不要将 target 加入版本控制.进入 testapp 的 POM 文件所在目录执行 mvnrelease:prepare–DautoVersionSubmodules=true 命令

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building testapp 1.0.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-release-plugin:2.0:prepare (default-cli) @ testapp ---
[INFO] Verifying that there are no local modifications...
[INFO] Executing: cmd.exe /X /C "svn --non-interactive status"
[INFO] Working directory: D:\workspace\first\testapp
[INFO] Checking dependencies and plugins for snapshots ...
What is the release version for "testapp"? (com.test:testapp) 1.0.0: :
What is SCM release tag or label for "testapp"? (com.test:testapp) testapp-1.0.0
: :
What is the new development version for "testapp"? (com.test:testapp) 1.0.1-SNAP
SHOT: : 1.1.0-SNAPSHOT
```

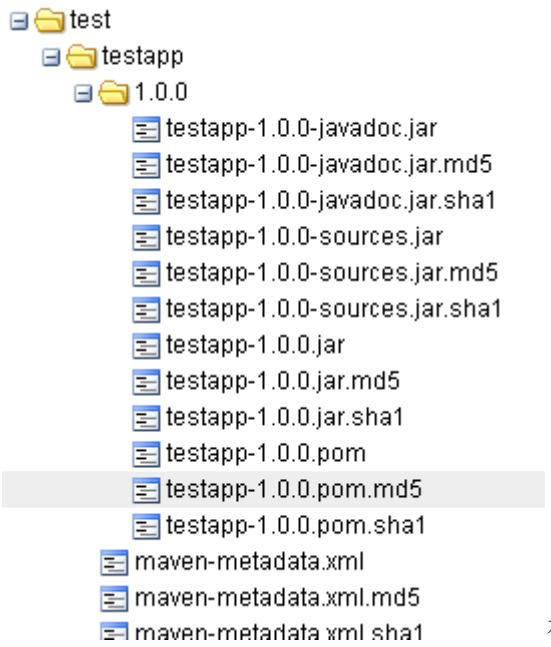
在最后那项里面填入你下一个希望的版本序号

```
[INFO] Release preparation complete.
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 14.750s
[INFO] Finished at: Tue May 08 23:37:44 CST 2012
[INFO] Final Memory: 5M/15M
[INFO] ------------------------------------------------------------------------
```

看到以上标签就表明准备成功了,里面的详细过程大家可以去看一下主要执行的内容和上面所诉一致.这时候可以执行 mvnrelease:perform

```
[INFO] Uploaded: http://10.0.2.143:8081/nexus/content/repositories/HEXINB2B_RELE
ASE/com/test/testapp/1.0.0/testapp-1.0.0-javadoc.jar (22 KB at 347.4 KB/sec)
[INFO] [INFO] ------------------------------------------------------------------
------
[INFO] [INFO] BUILD SUCCESS
[INFO] [INFO] ------------------------------------------------------------------
------
[INFO] [INFO] Total time: 7.813s
[INFO] [INFO] Finished at: Tue May 08 23:39:18 CST 2012
[INFO] [INFO] Final Memory: 13M/33M
[INFO] [INFO] ------------------------------------------------------------------
------
[INFO] Cleaning up after release...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 12.078s
[INFO] Finished at: Tue May 08 23:39:18 CST 2012
[INFO] Final Memory: 5M/15M
[INFO] ------------------------------------------------------------------------
```

大家看到这个就表明上传到仓库成功了.



在 maven 库里的效果大家可以看到同时生成了 sources 和 doc 的 JAR,这是因为在超级 POM 里有,最好大家在自己的 POM 历显示的声明一下,并制定下插件的版本增加稳定性.我们再看下 svn 库

可以看到在标签地址下保存了该版本的信息

至此版本自动化发布 OK 了,回想一下那个箭头图我们还少了分支一环,下面我就介绍下如何自动创建分支.

自动化创建分支

在 svn 上创建 branchs 目录

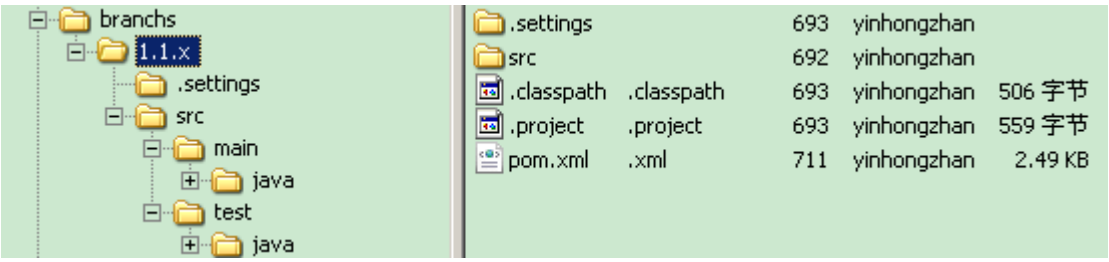
再 release 插件中加入

```
<branchBase>http://10.0.2.143/svn/infos/branchs/</branchBase>
```

其实这个你不加也没关系 maven 会找和 trunk 平级的 branchs,tags 也是

然 后 输 入 mvnrelease:branch -DbranchName=1.1.x -DupdateBranchVersions=true -DupdateWorkingCopyVersions=false

此时插件就会帮我们完成:检查本地有无未提交代码、为分支更改 POM 的信息、将 POM 中的信息改为分支的信息、提交以上更改、并将主干的代码复制到分支中、然后修改本地的代码使其还原到分之前的版本、提交本地更改,结果可以再 SVN 上可以看到如下效果:



这里的 POM 的版本已经升级为 1.1.1-SNAPSHOT 了

小结

项目版本发布的问题是非常重要的,这里主要介绍了版本的管理方式,包括各种不同种类版本的切换、版本号的意义、以及 SVN 和 MAVEN 的结合方式,最后还介绍了自动化发布和自动化创建分支的操作.