



放弃我。抓紧我
STAY WITH ME

王凯陈乔恩演绎时尚圈爱情童话
搜狐视频 正在热播



simonyee's blog 首页 日志

simonyee's blog



登录 注册

博文广场



个人资料

日志正文



博客年龄：10年10个月
访问：27396次
文章：201篇

个人描述

姓名：***
职业：**
年龄：**
位置：中国，**
个性介绍：

博主最新文章

Maven最佳实践：版本管理

2013-11-03 23:49

阅读(3145) 评论(0)

什么是版本管理

首先，这里说的版本管理（version management）不是指版本控制（version control），但是本文假设你拥有基本的版本控制的知识，了解subversion的基本用法。版本管理中说得版本是指构件（artifact）的版本，而非源码的版本（如subversion中常见的rXXX，或者git中一次提交都有个sha1的commit号）。

比如我有一个项目，其artifactId为myapp，随着项目的进展，我们会生成这样一些jar：myapp-1.0-SNAPSHOT.jar，myapp-1.0.jar，myapp-1.1-SNAPSHOT.jar，myapp-1.0.1.jar等等。你可能会说，这很简单啊，我在POM中改个version，mvn clean install不就完了？但这只是表面，本文我将讲述，snapshot和release版本的区别，如何自动化版本发布（如果你的项目有几十个module，你就会觉得手工改POM来升级版本是很痛苦的事情），结合自动化发布的过程，这里还会介绍maven-release-plugin。此外，一些scm概念也会被涉及到，比如tag和branch。

前提：版本控制

不管怎样，我们都需要建立一个项目并提交到SCM中，这里我以subversion为例。你得有一个配置好的subversion repository，这里我建立了一个空的svn仓库，其地址为：https://192.168.1.100:8443/svn/myapp/ 现在，该目录下只有三个空的典型的子目录：/trunk/，branches/，tags/。分别用来存放主干，分支，以及标签。

接着将项目导入到svn仓库中，到项目根目录，运行如下命令：

[Maven最佳实践：版本管理](#)

[Maven最佳实践：划分模块](#)

[CSS盒子模型之firefox与IE区别篇](#)

[如何解决Java WEB应用中的乱码问题](#)

[Maven2:OutOfMemory when execute mvn command](#)

[更多文章>>](#)

svn import -m 'project initialization' https://192.168.1.100:8443/svn/myapp/trunk
（注意，这么做你会将目录下所有文件导入到svn库中，但是这其中某些目录和文件是不应该被导入的，如/target目录，以及eclipse相关的项目文件）

目前，我们将项目的版本设置为1.0-SNAPSHOT。

为什么用SNAPSHOT？

我先说说如果没有SNAPSHOT会是什么样子。假设你的项目有2个模块，A,B，其中A依赖B。这三个模块分别由甲，乙两个人负责开发。在开发过程中，因为A是依赖于B的，因此乙每次做一个改动都会影响到甲，于是，乙提交了一些更改后，需要让甲看到。这个时候，怎么做呢？乙对甲说，“你签出我的代码，build一下就OK了”，甲有点不情愿，但还是照做了，签出代码，**svn clean install**，然后，发现build出错了，有个测试没有**pass**。甲郁闷了，对乙说，“你的代码根本不能用，我不想build，你build好了给我”，乙看了看确实自己的代码build不过，于是回去解决了，然后打了个jar包，扔给甲，甲对了groupId，artifactId，放到了自己的.m2/repository/目录下，OK，能用了。

于是乙每次更新都这样做，打包，复制，然后甲粘贴，使用.....渐渐的，大家发现这是个很笨的办法，这是纯手工劳动阿，程序员最BS的就是重复劳动。一天，甲对乙说，“你知道nexus么？你把你的jar发布到nexus上就可以了，我要用就自动去下载，这多棒！”乙说“哦？有这好东西，我去看看”于是乙发现了nexus这块新大陆，并成功的发布了B到nexus上。（见，[Nexus入门指南，（图文）](#)）。

但是，请注意，我们这里的一切都假设没有SNAPSHOT，因此如果乙不更改版本，甲下载一次如B-1.0.jar之后，maven认为它已经有了正确的B的版本，就不会再重新下载。甲发现了这个问题，对乙说“你的更新我看不到，你更新了么？”乙说“不可能！我看看”，于是检查一下甲下载的C- 1.0.jar，发现那是几天前的。乙一拍脑袋，说“这简单，我更新一下我的版本就好了，我发布个B-1.1.jar上去，你更新下依赖版本”，甲照做了，似乎这么做是可行的。

这里有一个问题，一次提交就更新一个版本，这明显不是正确的管理办法，此外，乙得不停的通知甲更新对B的依赖版本，累不累阿？1.0，或者说1.1，2.0，都代表了稳定，这样随随便便的改版本，能稳定么？

所以Maven有SNAPSHOT版本的概念，它与release版本对应，后者是指1.0，1.1，2.0这样稳定的发布版本。

现在乙可以将B的版本设置成1.0-SNAPSHOT，每次更改后，都mvn deploy到nexus中，每次deploy，maven都会将SNAPSHOT改成一个当前时间的timestamp，比如B-1.0- SNAPSHOT.jar到nexus中后，会成为这个样子：B-1.0-20081017-020325-13.jar。Maven在处理A中对于B的SNAPSHOT依赖时，会根据这样的timestamp下载最新的jar，默认Maven每天更新一次，如果你想让Maven强制更新，可以使用-U参数，如：**mvn clean install -U**。

现在事情简化成了这个样子：乙做更改，然后mvn deploy，甲用最简单的maven命令就能得到最新的B。

从1.0-SNAPSHOT到1.0到1.1-SNAPSHOT

SNAPSHOT是快照的意思，项目到一个阶段后，就需要发布一个正式的版本（**release**版本）。一次正式的发布需要这样一些工作：

1. 在trunk中，更新pom版本从1.0-SNAPSHOT到1.0
2. 对1.0打一个svn tag
3. 针对tag进行mvn deploy，发布正式版本
4. 更新trunk从1.0到1.1-SNAPSHOT

你可以手工一步步的做这些事情，无非就是一些svn操作，一些pom编辑，还有一些mvn操作。但是你应该明白，手工做这些事情，一来繁琐，而来容易出错。因此这里我介绍使用maven插件来自动化这一系列动作。

SCM

首先我们需要在POM中加入scm信息，这样Maven才能够替你完成svn操作，这里我的配置如下：

Xml代码

1. <scm>
2. <connection>scm:svn:http://192.168.1.100:8443/svn/myapp/trunk/</connection>
3. <developerConnection>scm:svn:https://192.168.1.100:8443/svn/myapp/trunk/</developerConnection>
4. </scm>

需要注意的是，很多windows使用的tortoiseSVN客户端，而没有svn命令行客户端，这会导致Maven所有svn相关的工作失败，因此，你首先确保svn --version能够运行。

分发仓库

想要让Maven帮我们自动发布，首先我们需要配置好分发仓库。关于这一点，见[Maven最佳实践：Maven仓库](#)——分发构件至远程仓库。

maven-release-plugin

紧接着，我们需要配置maven-release-plugin，这个插件会帮助我们升级pom版本，提交，打tag，然后再升级版本，再提交，等等。基本配置如下：

Xml代码

1. <plugin>
2. <groupId>org.apache.maven.plugins</groupId>
3. <artifactId>maven-release-plugin</artifactId>
4. <version>2.0-beta-7</version>
5. <configuration>

- 6. <tagBase>https://192.168.1.100:8443/svn/myapp/tags/</tagBase>
- 7. </configuration>
- 8. </plugin>

GAV我就不多解释了，这里我们需要注意的是configuration元素下的tagBase元素，它代表了我们的svn中的tag目录，也就是说，maven-release-plugin帮我们打tag的时候，其基础目录是什么。这里，我填写了svn仓库中的标准的tags目录。

提交代码

接着，确保你的所有代码都提交了，如果你有未提交代码，release插件会报错，既然你要发布版本了，就表示代码是稳定的，所以要么要么把代码提交了，要么把本地的更改抛弃了。

开始工作

现在，屏住呼吸，执行：

mvn release:prepare

执行过程中，你会遇到这样的提示：

What is the release version for "Unnamed - org.myorg:myapp:jar:1.0-SNAPSHOT"? (org.myorg:myapp) 1.0: :

——“你想将1.0-SNAPSHOT发布为什么版本？默认是1.0。”我要的就是1.0，直接回车。

What is SCM release tag or label for "Unnamed - org.myorg:myapp:jar:1.0-SNAPSHOT"? (org.myorg:myapp) myapp-1.0: :

——“发布的tag标签名称是什么？默认为myapp-1.0。”我还是要默认值，直接回车。

What is the new development version for "Unnamed - org.myorg:myapp:jar:1.0-SNAPSHOT"? (org.myorg:myapp) 1.1-SNAPSHOT: :

——“主干上新的版本是什么？默认为1.1-SNAPSHOT。”哈，release插件会自动帮我更新版本到1.1-SNAPSHOT，很好，直接回车。

然后屏幕刷阿刷，maven在build我们的项目，并进行了一些svn操作，你可以仔细查看下日志。

那么结果是什么呢？你可以浏览下svn仓库：

- 我们多了一个tag：https://192.168.1.100:8443/svn/myapp/tags/myapp-1.0/，这就是需要发布的版本1.0。
- 再看看trunk中的POM，其版本自动升级成了1.1-SNAPSHOT。

这不正是我们想要的么？等等，好像缺了点什么，对了，1.0还没有发布到仓库中呢。

再一次屏住呼吸，执行：

mvn release:perform

maven-release-plugin会自动帮我们签出刚才打的tag，然后打包，分发到远程Maven仓库中，至此，整个版本的升级，打标签，发布等工作全部完成。我们可以在远程Maven仓库中看到正式发布的1.0版本。

这可是自动化的，正式的 版本发布！

Maven的版本规则

前面我们提到了SNAPSHOT和Release版本的区别，现在看一下，为什么要有1.0，1.1，1.1.1这样的版本，这里的规则是什么。

Maven主要是这样定义版本规则的：

<主版本>.<次版本>.<增量版本>

比如说1.2.3，主版本是1，次版本是2，增量版本是3。

主版本一般来说代表了项目的重大的架构变更，比如说Maven 1和Maven 2，在架构上已经两样了，将来的Maven 3和Maven 2也会有很大的变化。次版本一般代表了一些功能的增加或变化，但没有架构的变化，比如说Nexus 1.3较之于Nexus 1.2来说，增加了一系列新的或者改进的功能（仓库镜像支持，改进的仓库管理界面等等），但从大的架构上来说，1.3和1.2没什么区别。至于增量版本，一般是一些小的bug fix，不会有重大的功能变化。

一般来说，在我们发布一次重要的版本之后，随之会开发新的版本，比如说，myapp-1.1发布之后，就着手开发myapp-1.2了。由于 myapp-1.2有新的主要功能的添加和变化，在发布测试前，它会变得不稳定，而myapp-1.1是一个比较稳定的版本，现在的问题是，我们在 myapp-1.1中发现了一些bug（当然在1.2中也存在），为了能够在段时间内修复bug并仍然发布稳定的版本，我们会用到分支（branch），我们基于1.1开启一个分支1.1.1，在这个分支中修复bug，并快速发布。这既保证了版本的稳定，也能够使bug得到快速修复，也 不同停止1.2的开发。只是，每次修复分支1.1.1中的bug后，需要merge代码到1.2（主干）中。

上面讲的就是我们为什么要用增量版本。

实战分支

目前我们trunk的版本是1.1-SNAPSHOT，其实按照前面解释的版本规则，应该是1.1.0-SNAPSHOT。

现在我们想要发布1.1.0，然后将主干升级为1.2.0-SNAPSHOT，同时开启一个1.1.x的分支，用来修复1.1.0中的bug。

首先，在发布1.1.0之前，我们创建1.1.x分支，运行如下命令：

mvn release:branch -DbranchName=1.1.x -DupdateBranchVersions=true -DupdateWorkingCopyVersions=false

这是maven-release-plugin的branch目标，我们指定branch的名称为1.1.x，表示这里会有版本1.1.1, 1.1.2等等。updateBranchVersions=true的意思是在分支中更新版本，而 updateWorkingCopyVersions=false是指不更改当前工作目录（这里是trunk）的版本。

在运行该命令后，我们会遇到这样的提示：

What is the branch version for "Unnamed - org.myorg:myapp:jar:1.1-SNAPSHOT"? (org.myorg:myapp) 1.1-SNAPSHOT: :

——"分支中的版本号是多少？默认为1.1-SNAPSHOT" 这时我们想要的版本是1.1.1-SNAPSHOT，因此输入1.1.1-SNAPSHOT，回车，maven继续执行直至结束。

接着，我们浏览svn仓库，会看到这样的目录：https://192.168.1.100:8443/svn/myapp/branches/1.1.x/，打开其中的POM文件，其版本已经是1.1.1-SNAPSHOT。

分支创建好了，就可以使用release:prepare和release:perform为1.1.0打标签，升级trunk至1.2.0-SNAPSHOT，然后分发1.1.0。

至此，一切OK。

小结

本文讲述了如何使用Maven结合svn进行版本管理。解释了Maven中SNAPSHOT版本的来由，以及Maven管理版本的规则。并结合 SCM的tag和branch概念展示了如何使用maven-release-plugin发布版本，以及创建分支。本文涉及的内容比较多，且略显复杂， 不过掌握版本管理的技巧对于项目的正规化管理来说十分重要。Maven为我们提供了一些一套比较成熟的机制，值得掌握。

分享到：

阅读(3145) 评论(0)

下一篇：[Maven最佳实践：划分模块](#)

评论  想第一时间抢沙发么？

由于最近广告泛滥，暂只允许登录用户对此文评论。[登录](#)