Maven仓库  
在以前使用Ant的时候，我们会建立一个lib目录在存放我们的jar包，比如项目所依赖的第三方包，每建立一个项目都要建 立一个lib，不停的做copy工作，不仅是对于磁盘的浪费，而且也造成了版本管理上的麻烦。而且我们还需要通过提交到svn上来对lib进行管理，但是 svn对于这种二进制文件的管理并不出色。

Maven仓库的初中就是为了解决这个问题，是所有常用的第三方包的集 中营。这样所有的Maven项目就可以从这个仓库中获取所需要的资源，Maven仓库中对jar通过Group Id, Atifact Id, version 来管理，所以Maven项目可以很方便的进行依赖管理。你不需要自己来管理这个庞大的资源仓库，当然你可以创建一个公司层面的仓库管理器，这个我在这个章 节的后面会介绍。

Maven仓库的两个概念：**本地仓库**和**远程仓库**

本地仓库是远程仓库的一个缓冲和子集，当你构建Maven项目的时候，首先会从本地仓库查找资源，如果没有，那么Maven会从远程仓库下载到你本地仓库。这样在 你下次使用的时候就不需要从远程下载了。如果你所需要的jar包版本在本地仓库没有，而且也不存在于远程仓库，Maven在构建的时候会报错，这种情况可 能发生在有些jar包的新版本没有在Maven仓库中及时更新。

Maven缺省的本地仓库地址为${user.home}/.m2/repository 。也就是说，一个用户会对应的拥有一个本地仓库。当然你可以通过修改${user.home}/.m2/settings.xml配置这个地址：

Xml代码

<settings>     
  ...      
  <localRepository>D:\java\repository</localRepository>     
  ...      
</settings>     
如果你想让本计算机所有的用户使用统一的配置那么你可以修改Maven主目录下的setting.xml:

${M2\_HOME}/conf/setting.xml

还可以通过在运行时指定目录，但是并不推荐这么做：

mvn clean install -Dmaven.repo.local=/home/juven/myrepo/

当 我们创建一个简单的Maven项目后(只需要在pom.xml配置好依赖)，运行mvn clean install就可以把项目构建好，不需要我们手工下载任何jar，这全靠中央仓库的存在，它会自动从仓库下载。这个仓库的定义是 在${M2\_HOME}/lib/maven-2.0.10-uber.jar里面。你可以在里面找到\org\apache\maven \project\pom-4.0.0.xml这个文件，在这个文件里面定义了中央仓库的地址：

<repositories>     
  <repository>     
    <id>central</id>     
    <name>Maven Repository Switchboard</name>     
    <layout>default</layout>     
    <url>http://repo1.maven.org/maven2</url>     
    <snapshots>     
      <enabled>false</enabled>     
    </snapshots>     
  </repository>     
</repositories>     
在POM中配置远程仓库

下 面我介绍下如何在pom.xml里面配置远程仓库，我们需要在什么时候配置远程仓库呢？当你连接中央仓库的速度比较慢时，或者你为你的公司搭建了自己的仓 库，比如Nexus仓库管理(后面我会介绍)，又或者你苏需要的jar存在另外一个公共仓库，比如我们配置一个国内的镜像地址：

<project>     
...      
  <repositories>     
    <repository>     
      <id>maven-net-cn</id>     
      <name>Maven China Mirror</name>     
      <url>http://maven.net.cn/content/groups/public/</url>     
      <releases>     
        <enabled>true</enabled>     
      </releases>     
      <snapshots>     
        <enabled>false</enabled>     
      </snapshots>     
    </repository>     
  </repositories>     
  <pluginRepositories>     
    <pluginRepository>     
      <id>maven-net-cn</id>     
      <name>Maven China Mirror</name>     
      <url>http://maven.net.cn/content/groups/public/</url>     
      <releases>     
        <enabled>true</enabled>     
      </releases>     
      <snapshots>     
        <enabled>false</enabled>     
      </snapshots>          
    </pluginRepository>     
  </pluginRepositories>     
...      
</project>    
这 里我们可以看到，允许配置多个repository和 plugin repository，

<releases><enabled>true</enabled>< /releases>告诉Maven可以从这个仓库下载releases版本的构件， 而<snapshots><enabled>false</enabled></snapshots> 告诉Maven不要从这个仓库下载snapshot版本的构件

,之所以不让你从这个仓库下载snapshot版本，是因为这些版本是不稳定的，但是 snapshot版本在我们内部项目开发的时候可是作用巨大，后面的问文章我会讨论这个问题。至 于<pluginRepositories>，这是配置Maven从什么地方下载插件构件，Maven的所有行为都是通过插件来完成的，其内 部配置与<repository>类似，这里就不多说了。

尽管pom.xml中可以方便的配置中央仓库，但是我并不推荐大家这么做，尤其是在大的公司中，因为一个公司会有很多的项目，如果每个项目都这样配置，那么又开始做重复的copy工作了，如何解决呢，我们往下走

在settings.xml中配置远程仓库

Pom.xml 的作用范围是一个项目，一个公司不可能只做一个项目，那么为了避免重复配置，那么我们需要把一些公共信息配置在setting.xml中。但是 setting.xml中并不支持<repositories>及<pluginRepositories>，为了解决这个问题 我们使用profile：

<settings>     
  ...      
  <profiles>     
    <profile>     
      <id>myProfiel</id>     
      <!—在这里加入<repositories>及<pluginRepositories>-->     
    </profile>     
  </profiles>     
  <activeProfiles>     
    <activeProfile> myProfiel </activeProfile>     
  </activeProfiles>     
  ...      
</settings>     
这里通过<activeProfile>元素来激活这个profile，这样我们就可以全局的使用这个配置，不再需要为每个POM做重复的配置了。

在实际的操作过程中，这里我们最好不要修改远程仓库配置，最好能够通过nexus建立公司或者组织自己的仓库，然后这把把地址指向自己的仓库，后面我会介绍为什么要这么做，怎么做。

配置镜像

如果你想覆盖中央仓库的默认地址，那么这里我们就会使用的镜像了，还在setting.xml里面配置：

<settings>     
...      
  <mirrors>     
    <mirror>     
      <id>maven-net-cn</id>     
      <name>Maven China Mirror</name>     
      <url>http://maven.net.cn/content/groups/public/</url>     
      <mirrorOf>central</mirrorOf>     
    </mirror>     
  </mirrors>     
...      
</settings>     
这里解释一下<mirrorOf>,表示只为central仓库做镜像，如果想为所有的仓库做镜像那么可以改为：<mirrorOf>\*</mirrorOf>

如果你看到这里，请先不要着急，实际的项目经验告诉我，只是这些还不够，我们需要更快捷和高效的管理：

利用Nexus来构建企业级Maven仓库  
Nexus简介

Nexus是Maven仓库管理器，用来搭建一个本地仓库服务器，这样做的好处是便于管理，节省网络资源，速度快，还有一个非常有用的功能就是可以通过项目的SNAPSHOT版本管理，来进行模块间的高效依赖开发，下面会一一描述。

虽然你可以通过中央仓库来获取你所需要的jar包，但是现实往往是存在很多问题：

网速慢，你可能需要花很长的时间来下载你所需要的jar   
如果你的公司很大，有几百甚至几千人再用Maven，那么这些人都去通过中央仓库来获取jar，那么这是一个很大的资源浪费   
如果存在模块之间的依赖开发，你的snapshot版本是不能够被你的伙伴很方便的获取。   
在实际开发过程中，有些jar的版本可能在中央仓库里面不存在，或者更新不及时，你是获取不到这个jar的。   
所有以上问题，通过Nexus这个日益流行的仓库管理器可以轻松的解决。

这个仓库是本地的，下载的速度是从远程下载不可比的。   
可以为你公司所有的Maven使用者服务，可以进行统一管理   
后面我会介绍如何通过nexus来进行存在模块依赖的项目的开发   
你可以添加自己的第三方包。

## 安装Nexus

我们从<http://nexus.sonatype.org/downloads/> 来获取最新版本，目前最新版本为1.3.4

Nexus提供了两种安装方式:

一种是内嵌Jetty的bundle，只要你有JRE就能直接运行。

第二种方式是WAR，你只须简单的将其发布到web容器中即可使用。

**Bundle方式安装**

解 压nexus-webapp-1.3.4-bundle.zip 至任意目录，如D:\tools ，转到目录D:\tools\nexus-webapp-1.3.4\bin\jsw\windows-x86-32 ，运行Nexus.bat ，如果你是在linux下安装，那么就下载nexus-webapp-1.3.4-bundle.tar.gz，解压后转到${NEXUS\_HOME}/ nexus-webapp-1.3.3\bin\jsw\linux-x86-32,它还支持solaris,macos等操作系统。当你看到 “Started [SelectChannelConnector@0.0.0.0:8081](mailto:SelectChannelConnector@0.0.0.0:8081)”之后，说明Nexus启动成功了，然后打开浏览器，访问<http://127.0.0.1:8081/nexus>，通过admin的帐号(admin)和密码(admin123)登录你会看到如下的页面：

如果有新版本发布，会有提示在默认页面上。

这里，可以管理仓库，配置Nexus系统，管理任务，管理用户，角色，权限，查看系统的RSS源，管理及查看系统日志，等等。

**War方式安装**

你可以同过war的方式以web应用的形式发布到你的应用服务器，比如tomcat。你所要做的就是下载war版本的文件，然后放到应用服务器的发布目录即可，这里就不多讲了。

到此我们已经安装好Nexus,下面我来介绍下一些我们常用的功能和使用：

## 配置中央仓库

先看一下界面：

在左边菜单栏里选择Repositories，然后会出现右边的画面，右边上半部分是列出来的repository，黑体字是类型为group的repository. 这里简单介绍下几种repository的类型:

hosted，本地仓库，通常我们会部署自己的构件到这一类型的仓库。比如公司的第二方库。   
proxy，代理仓库，它们被用来代理远程的公共仓库，如maven中央仓库。   
group，仓库组，用来合并多个hosted/proxy仓库，当你的项目希望在多个repository使用资源时就不需要多次引用了，只需要引用一个group即可。   
Maven central是Maven的中央仓库，点击它并选择configuration标签栏，我们会看到下面的页面：

这里有几个项目是我们可能会经常用到的：

Override local storage location: 在这个选项你可以配置你的Nexus本地仓库的存放地址，用来覆盖其默认的存放地址   
Remote storage location: 这里是远程仓库的地址，为了提高代理速度，你可以修改为国内的镜像地址。默认值是<http://repo1.maven.org/maven2/>   
Download remote indexes: 这里配置是否下载远程索引文件，模式是false, 建议配置为true，这样我们便可以通过索引文件来搜索我们需要的构件。

## 添加代理仓库

Maven central是一个比较大的代理仓库，如果你需要添加的一个代理仓库，那么可以在点击左边惨淡栏里面的Repositories,然后右边的页面点击add -> add proxy repository, 之后出现以下页面：

填写相关信息保存即可。通常情况下,使用预设的代理仓库已经能够满足大部分项目的需求了, 只有在特殊需求的情况下才会参加代理仓库.

## 管理本地仓库

我们前面讲到类型为hosted的为本地仓库，Nexus预定义了3个本地仓库，分别是Releases, Snapshots, 3rd Party. 分别讲一下这三个预置的仓库都是做什么用的:

Releases: 这里存放我们自己项目中发布的构建, 通常是Release版本的, 比如我们自己做了一个FTP Server的项目, 生成的构件为ftpserver.war, 我们就可以把这个构建发布到Nexus的Releases本地仓库. 关于符合发布后面会有介绍.   
Snapshots: 这个仓库非常的有用, 它的目的是让我们可以发布那些非release版本, 非稳定版本, 比如我们在trunk下开发一个项目,在正式release之前你可能需要临时发布一个版本给你的同伴使用, 因为你的同伴正在依赖你的模块开发, 那么这个时候我们就可以发布Snapshot版本到这个仓库, 你的同伴就可以通过简单的命令来获取和使用这个临时版本.   
3rd Party: 顾名思义, 第三方库, 你可能会问不是有中央仓库来管理第三方库嘛, 没错, 这里的是指可以让你添加自己的第三方库, 比如有些构件在中央仓库是不存在的. 比如你在中央仓库找不到Oracle 的JDBC驱动, 这个时候我们就需要自己添加到3rd party仓库.   
你也可以创建自己的本地仓库，点击Repository面板上方的Add按钮，然后选择Hosted Repository，然后在下方的配置面板中输入相关信息, 这里我们不再需要填写远程仓库地址，根据自己的需要选择Release或者Snapshot，如图：

## Maven仓库组

仓 库组的概念是Maven没有的,通过前面介绍可以了解到, 我们可以建立多个proxy代理仓库,hosted本地仓库, 如果没有仓库组的概念,我们如果需要引用这些仓库的是时候需要一一加入到我们的setting.xml里面去, 有了仓库的组的概念, 我们只需要做一次引用就可以了,把我们需要的仓库加入到仓库组即可.像这样:

    <repositories>

     <repository>

        <id>nexus</id>

          <url>http://127.0.0.1:8081/nexus/content/groups/public/</url>

            <releases>

            <enabled>true</enabled>

    </releases>

     <snapshots>

       <enabled>true</enabled>

     </snapshots>

</repository>

  </repositories>

Nexus中预设了2个仓库组,public repositories和public snapshot repositories. 如图:

这个仓库组默认包含本地仓库Releases, snapshots, 3rd party和代理仓库Maven Central. 你可以在Configuration配置页添加仓库到这个仓库组. 如果需要你还可以创建一个仓库组,如图:

点击 Add->Repository Group

出现New Repository Group界面后,填入ID, name 等相关信息, 在右边Available Repositories 栏里可以选择你要添加的仓库到你新建的仓库组.

## 通过Nexus搜索构件

在我们实际使用构件的过程中通常遇到一个问题,有时候我紧紧知道我所需要的构建的大致名字,并不知道全称或group id, 这是件非常头疼的事情. Nexus基于Nexus indexer的搜索功能帮我们解决了这个问题.

要 是用搜索功能, 必须要有索引文件, Nexus默认是不建立索引文件的,因为像中央仓库这样的索引文件的建立需要耗费比较大的网络资源,仅索引文件就要几十兆. 要开启中央仓库的索引文件下载功能需要在Maven Central的配置页面, 把Download Remote Indexes选择true. 如图:

这样设置以后, Nexus会自动从远程中央仓库下载索引文件, 为了检验索引文件自动下载是否生效,可以却换到Browse:

如果出现先以上文件夹,那说明索引文件已经建立.

下面我试一下搜索功能, 在左边菜单栏里面有个Artifact Search, 在输入框里面输入你想要搜索的构件名字,比如: testing, 那么查询结果如下：

这是模糊查询的结果，当然如果你知道更多信息，比如版本号等，你可以使用高级搜索，点击高级搜索后，右边界面会提供集中搜索方式：keyword, classname, GAV, checksum

比如我们这里选择GAV模式, 而且我只知道artiface name : testng和版本号5.8, 其他的我不知道, 那么就在artifact 和 version处分别输入testng 和 5.8 , 搜索结果如下:

你如果你不知道知道构件的名称, 只知道classname, 那么你也可以通过class name 的方式搜索,这里就不再赘述

当你选择一项搜索结果,在页面的下方会出现这个构件的详细信息, 并且会列出这个构件的Maven依赖配置, 你可以直接拷贝到你的pom文件中使用,这是个非常实用的功能:

在Maven中使用Nexus

   到此为止我们介绍了如何安装和使用Nexus以及其基本配置, 下面我们介绍下如何让Maven来使用Nexus本地仓库用来替代使用远程仓库. 在Maven使用Nexus本地仓库只需稍作配置, 在settings.xml中加入以下代码:

    <profile>

       <id>dev</id>

          <repositories>

             <repository>

                 <id>nexus</id>

                 <url>http://127.0.0.1:8081/nexus/content/groups/public/</url>

                 <releases>

                    <enabled>true</enabled>

                 </releases>

                 <snapshots>

                    <enabled>true</enabled>

                 </snapshots>

              </repository>

            </repositories>

            <pluginRepositories>

                <pluginRepository>

                    <id>nexus</id>

                    <url>http://127.0.0.1:8081/nexus/content/groups/public</url>

                    <releases>

                        <enabled>true</enabled>

                    </releases>

                    <snapshots>

                        <enabled>true</enabled>

                    </snapshots>

                </pluginRepository>

            </pluginRepositories>

        </profile>

    </profiles>

    <activeProfiles>

        <activeProfile>dev</activeProfile>

    </activeProfiles>

这里配置了repository和pluginRepository, Maven在使用第三方构件和插件时是分开来配置的,所以如果我们也希望插件的下载也通过我们的本地仓库来下载,那么我们就需要配置pluginRepository.

红色字体部分就是我们之前安装的Nexus的地址, 这个地址可以是你们公司局域网内部的一台仓库服务器.

<releases> <enabled>true</enabled></releases>这个标签的作用是设定是否允许下载

release版本的载构件, 同样snapshots标签可以设定是否允许下载snapshot版本的构件.

通常,我们不建议下载snapshot版本的构件,因为它是不稳定的版本, 除非你有特殊的需

求.

## 构件部署

有些时候我们需要部署构件到Nexus的3rd party, 比如我们在中央仓库找不到我们需要的构件, 我们可以通过Nexus的UI来上传构件:

点击左边菜单栏的 Repositories, 然后点击右边界面的3rd party, 选择界面下方的Artifact Upload, 这个时候出现以下界面:

上传构件需要两个步骤,一个是定义文件的上传,再就是构件的实体文件.

第一部分定义文件可以是POM文件, 这也是比较推荐的方式, 如果没有pom文件,可以

选择以参数的形式输入.

第二部分是上传构件的实体文件,这里简单说一下Classifier和Extension, 这两个都是选

填相, Classifier用来区别同功能的构件用于不同的场景, 比如这个构件是分别针对JDK14

和JDK15做了2个功能一样的Jar, 这个时候你就需要指定这个构件的Classifier为JDK14

还是JDK15. Extension是指扩展名,如果不提供,那么会自动取这个构件的Packaging Type

作为扩展名, 比如 ear, jar, war 等等. (Packaging Type是在第一步中通过pom文件或者手

工输入得到的)

刚才说了3rd party的部署, 关于releases 和 snapshots的UI部署也是一样的操作过程.

我们之前也讲过, 这里的releases和snapshots是用来部署我们自己的项目构件的, 通过

UI部署是可以,但是不是最高效的, 我们可以通过配置Maven来自动部署我们的项目构

件,这也是我们建立自己的仓库的一个非常重要的原因, 下面就让我们看看如何配置:

首先需要在POM文件中加入以下代码:

<project>     
...      
<distributionManagement>     
  <repository>     
    <id>nexus-releases</id>     
      <name>Nexus Release Repository</name>     
      <url>http://127.0.0.1:8081/nexus/content/repositories/releases/</url>     
  </repository>     
  <snapshotRepository>     
    <id>nexus-snapshots</id>     
    <name>Nexus Snapshot Repository</name>     
    <url>http://127.0.0.1:8081/nexus/content/repositories/snapshots/</url>     
  </snapshotRepository>     
</distributionManagement>     
...      
</project>     
这里配置,让Maven知道当我要发布release版本或者snapshot版本是需要发布到哪个地址.

然后我们需要在setting.xml里面配置一下Nexus的帐号和密码:

<settings>     
...      
<servers>     
  <server>     
    <id>nexus-releases</id>     
    <username>admin</username>     
    <password>admin123</password>     
  </server>     
  <server>     
    <id>nexus-snapshots</id>     
    <username>admin</username>     
    <password>admin123</password>     
  </server>        
</servers>     
...      
</settings>   
到此为止, 我们就可以通过命令mvn deploy或者通过IDE的可视化界面点击deploy来发布我们项目到本地仓库了. 通过这种方式我们可以很方便的进行模块间的依赖开发, 在后面的文章中我会详细介绍如何通过snapshot来让我们的依赖开发变得简单.

小结  
本文介绍了Maven仓库, 如何通过Nexus建立自己本地仓库, 通过和远程仓库的比较, 我们知道Nexus给我带来很多方便之处,  方便我们管理, 方便我们的项目构件部署, 项目的依赖开发等. 还在等什么, 创建你自己的仓库吧.

本文来自CSDN博客，转载请标明出处：<http://blog.csdn.net/arvinzhuo/archive/2009/06/05/4244061.aspx>