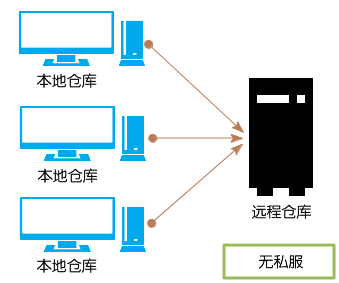
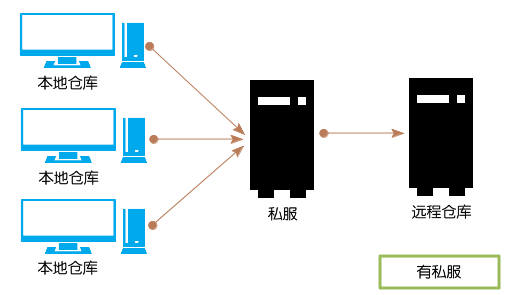
# Nexus搭建Maven私服

**一. 私服搭建及配置**

**1 . 私服简介**

　　私服是架设在局域网的一种特殊的远程仓库，目的是代理远程仓库及部署第三方构件。有了私服之后，当 Maven 需要下载构件时，直接请求私服，私服上存在则下载到本地仓库；否则，私服请求外部的远程仓库，将构件下载到私服，再提供给本地仓库下载。

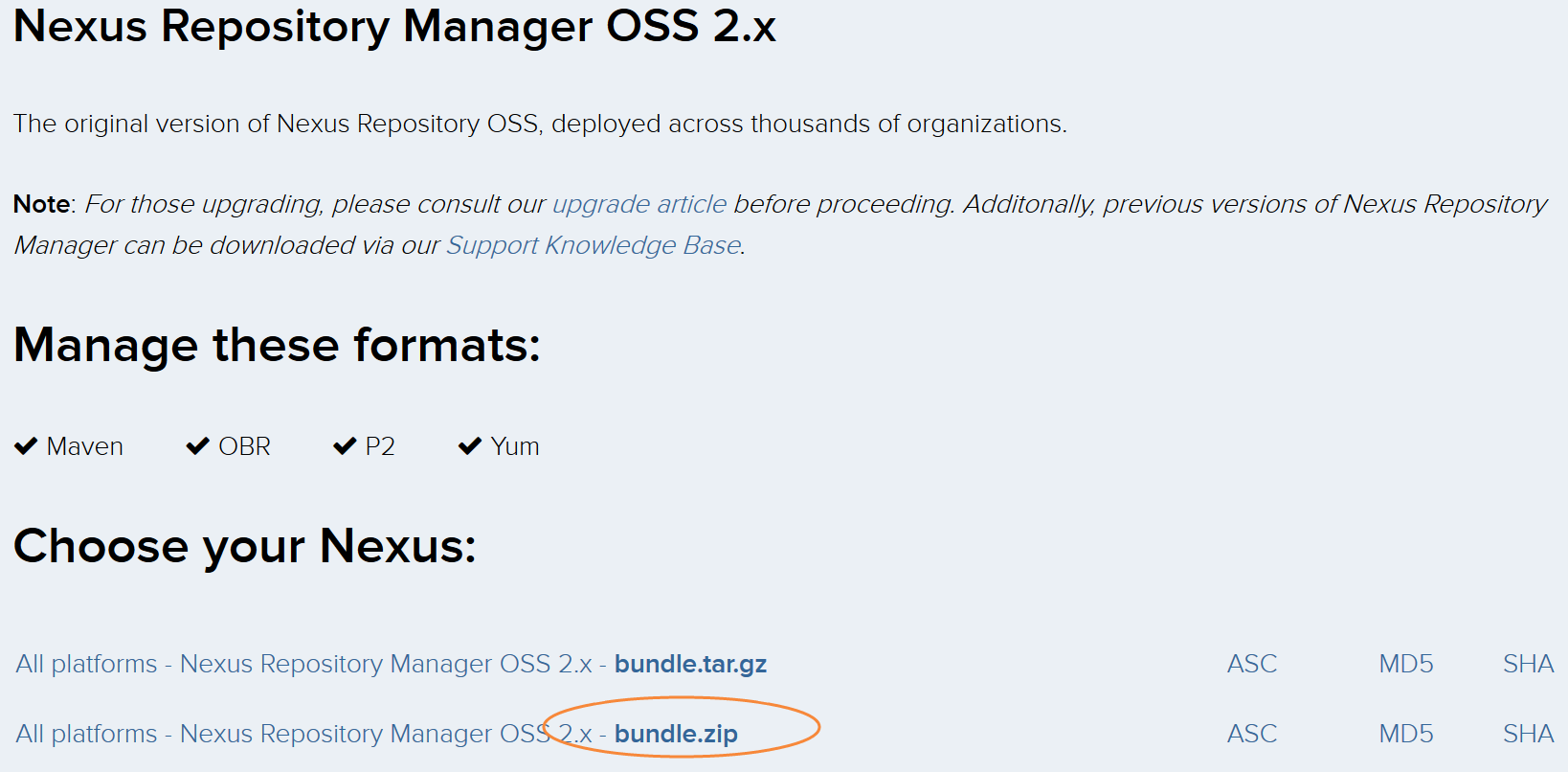
                                                

　　我们可以使用专门的 Maven 仓库管理软件来搭建私服，比如：[Apache Archiva](http://archiva.apache.org/index.cgi)，[Artifactory](http://www.jfrog.com/home/v_artifactory_opensource_overview/)，[Sonatype Nexus](http://www.sonatype.org/nexus/)。这里我们使用 Sonatype Nexus。

**2 . 安装Nexus**

**2 . 1 . 下载Nexus**

　　Nexus 专业版是需要付费的，这里我们下载开源版 Nexus OSS。下载地址：[http://www.sonatype.org/nexus/go,最新的为OSS3.X，我这里选择稳定版本2.x,点击后自动下载](http://www.sonatype.org/nexus/go)

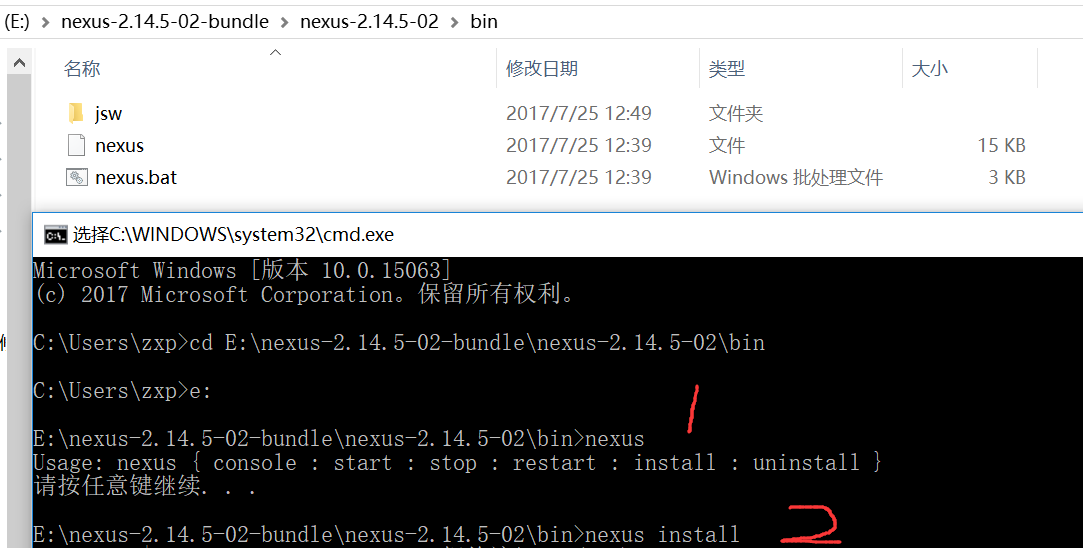


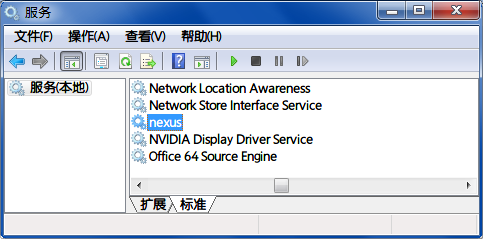
**2 . 2 . 使用bundle安装包安装Nexus**

解压安装包nexus-2.14.5-02.zip，打开命令提示符，进入nexus-2.14.5-02/bin目录 （为方便启动和退出Nexus，可将bin目录添加到环境变量），

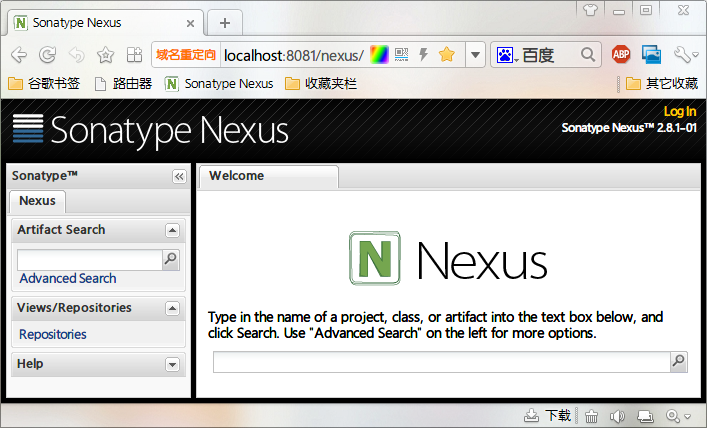
1.执行nexus命令

2.执行 nexus install 将Nexus安装为Windows服务





**3.打开浏览器，访问：http://localhost:8081/nexus/：**



点击右上角 Log In，使用用户名：admin ，密码：admin123 登录，可使用更多功能：

**4 . Nexus预置的仓库**

**4.1  类型介绍**

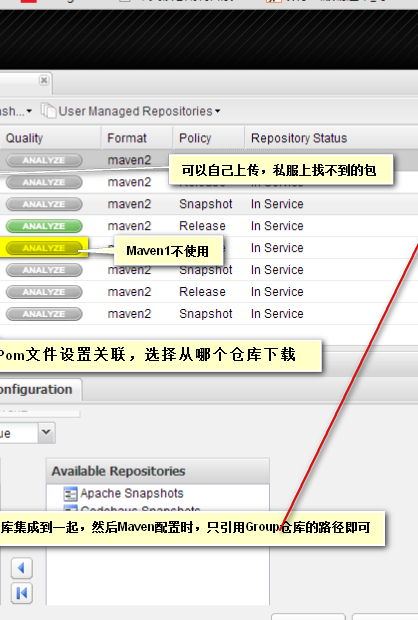
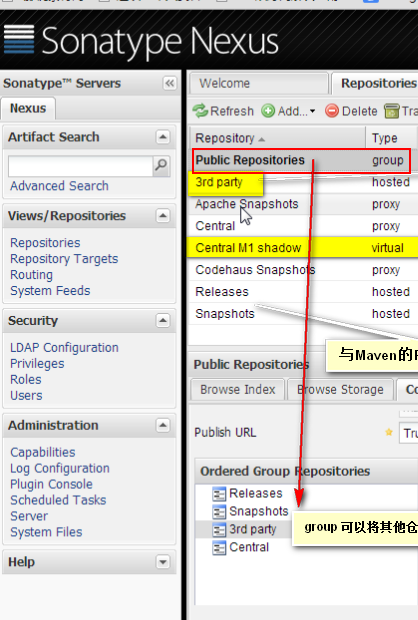
　　　　登陆Nexus，在左边菜单栏里选择Repositories，然后会出现右边的画面，右边上半部分是列出来的repository，黑体字是类型为group的repository。

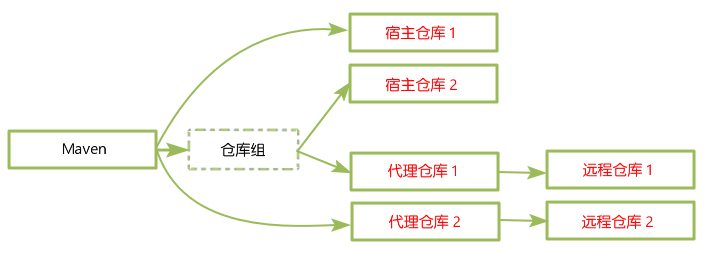
　　　　这里简单介绍下几种repository的类型：

　　　　Ø  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。

　　　　Ø  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。

**Ø  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。**





4.2  **管理本地仓库**

　　　　我们前面讲到类型为hosted的为本地仓库，Nexus预置了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.

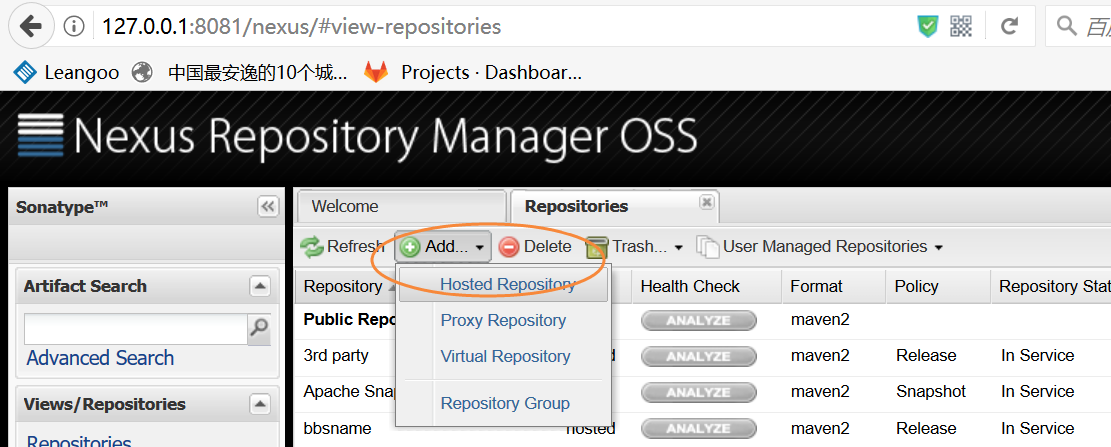
**3rd Party:**

　　　　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3rdparty仓库。

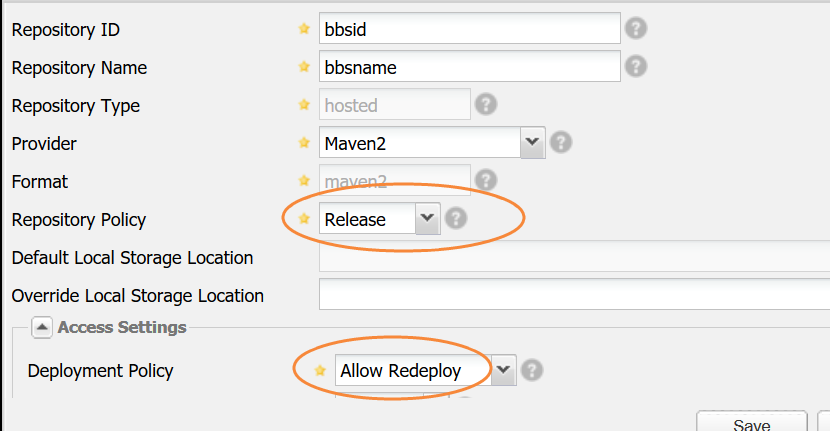
## 

## 4.3  在Nexus为自己的项目手动单独建库

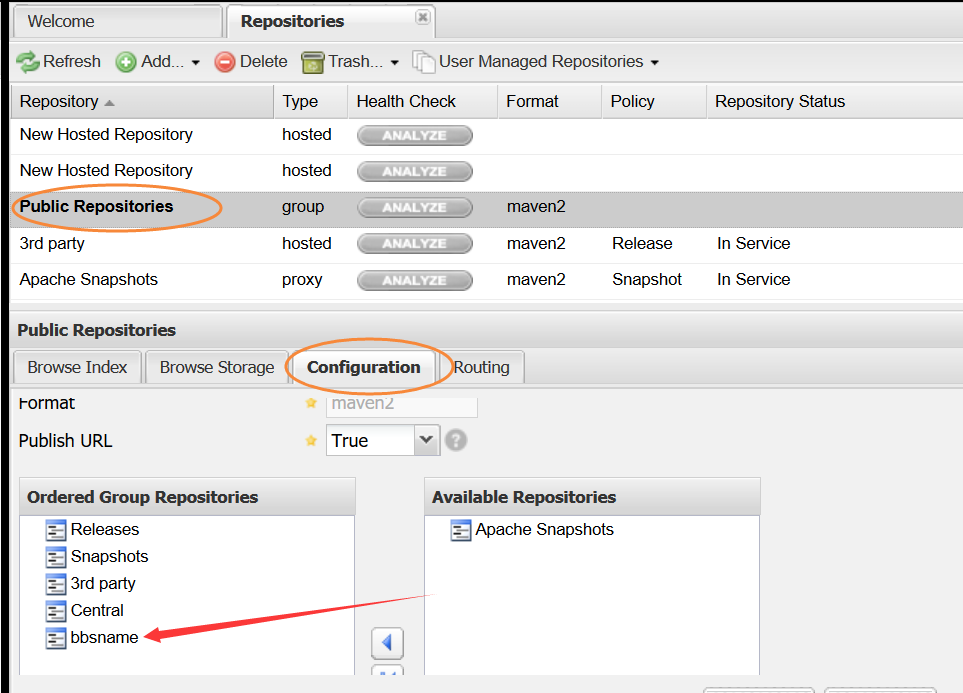
　　1. 建库，Add-- >HostedRepository



按照下图填写信息，注意圈的地方，填写完成后save



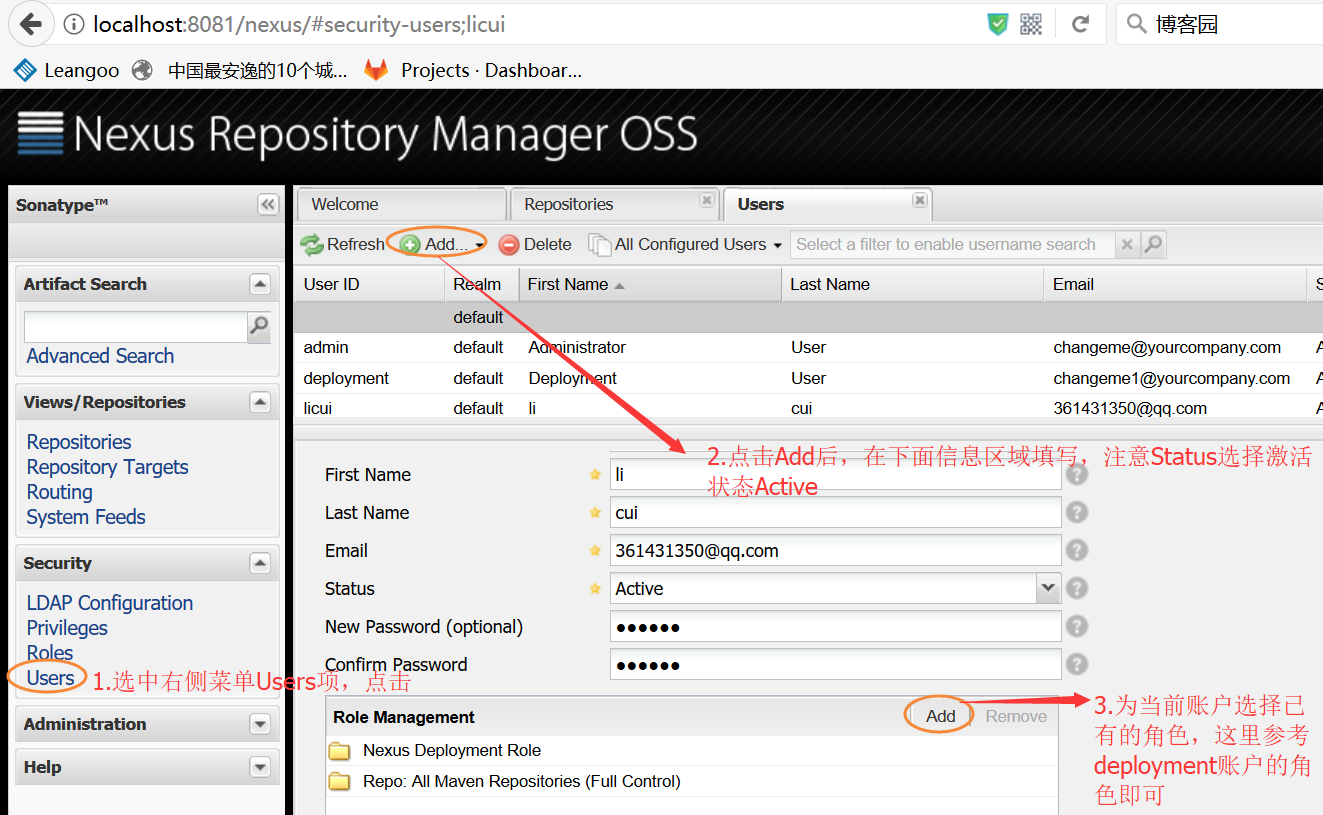
      2. 然后选择PublicRepositories,打开configuration选项卡



   3.  将自己创建的仓库添加到group,如图从右侧移到左侧，点击save，保存。至此，已经成功搭建好我们自定义的仓库了。

   注：如果需要你还可以创建一个仓库组,在Repositories菜单处选add-->Repository Group即可

## 5.  在Nexus创建账户

****

     最后点击保存。当然自己也可以创建角色，分配权限等，这里直接用已有的角色，省略了一些步骤。

**二. 利用搭建的私服，进行上传和下载jar包**

**1. 远程仓库的认证**

大部分公共的远程仓库无须认证就可以直接访问，但我们在平时的开发中往往会架设自己的Maven远程仓库，出于安全方面的考虑，我们需要提供认证信息才能访问这样的远程仓库。配置认证信息和配置远程仓库不同，远程仓库可以配置在settings.xml文件中，也可直接在pom.xml中配置，但是认证信息必须配置在settings.xml文件中。在settings.xml中配置认证信息更为安全。配置用的账号为上面我们创建的账户。

[复制代码](javascript:void(0);)

复制代码

1 <settings>

2 2 ...

3 3 <!--配置远程仓库认证信息-->

4 4 <servers>

5 5 <server>

6 6 <id>bbsnexus</id>

7 7 <username>licui</username>

8 8 <password>a111111</password>

9 9 </server>

10 10 </servers>

11 11 ...

12 12 </settings>

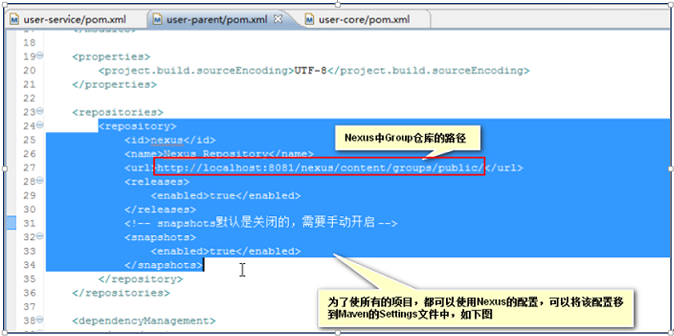
复制代码

[复制代码](javascript:void(0);)

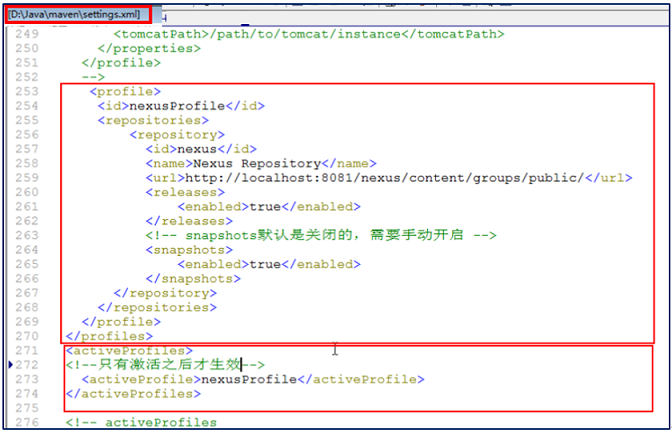
上面代码我们配置了一个id为bbsnexus的远程仓库认证信息。Maven使用settings.xml文件中的servers元素及其子元素server配置仓库认证信息。认证用户名为licui，认证密码为a111111。这里的关键是id元素，settings.xml中server元素的id必须与pom.xml中需要认证的repository元素的id完全一致。正是这个id将认证信息与仓库配置联系在了一起。

**2.  配置远程仓库**

        如果只有一个项目，可以配置在项目的pom文件中：



　　如果有多个项目都使用Nexus的配置，可以将如上皮遏制放到Maven的Settings文件中：



**repository：**在repositories元素下，可以使用repository子元素声明一个或者多个远程仓库。

**id：**仓库声明的唯一id，尤其需要注意的是，Maven自带的中央仓库使用的id为central，如果其他仓库声明也使用该id，就会覆盖中央仓库的配置。

**name：**仓库的名称，让我们直观方便的知道仓库是哪个，暂时没发现其他太大的含义。

**url：**指向了仓库的地址，一般来说，该地址都基于http协议，Maven用户都可以在浏览器中打开仓库地址浏览构件。

**releases和snapshots：**用来控制Maven对于发布版构件和快照版构件的下载权限。需要注意的是**enabled**子元素，该例中releases的enabled值为true，表示开启JBoss仓库的发布版本下载支持，而snapshots的enabled值为false，表示关闭JBoss仓库的快照版本的下载支持。根据该配置，Maven只会从JBoss仓库下载发布版的构件，而不会下载快照版的构件。

**layout：**元素值default表示仓库的布局是Maven2及Maven3的默认布局，而不是Maven1的布局。基本不会用到Maven1的布局。

**其他：**对于releases和snapshots来说，除了enabled，它们还包含另外两个子元素updatePolicy和checksumPolicy。

元素**updatePolicy**用来配置Maven从远处仓库检查更新的频率，默认值是daily，表示Maven每天检查一次。其他可用的值包括：never-从不检查更新；always-每次构建都检查更新；interval：X-每隔X分钟检查一次更新（X为任意整数）。

元素**checksumPolicy**用来配置Maven检查校验和文件的策略。当构建被部署到Maven仓库中时，会同时部署对应的检验和文件。在下载构件的时候，Maven会验证校验和文件，如果校验和验证失败，当checksumPolicy的值为默认的warn时，Maven会在执行构建时输出警告信息，其他可用的值包括：fail-Maven遇到校验和错误就让构建失败；ignore-使Maven完全忽略校验和错误。

# 3.    配置仓库镜像

　　默认的，如果本地仓库找不到依赖的构件，这时需要东西时先到Nexus上找，如果发现Nexus服务关闭后，会自动到中央仓库找。

　　如果我们想覆盖中央仓库的默认地址，强制依赖的东西都到Nexus中去找，即使Nexus关闭也不会到中央工厂去下载：

　　修改Maven的Settings文件：



至此，我们已经可以从自己搭建的私服下载jar了。

**4.上传jar包到远程仓库**

**1）将已有的项目打成jar包上传到私服服务器**

首先需要在pom.xml中配置上传仓库的地址，配置distributionManagement元素，仓库地址指向前面自定义的仓库bbsid

[复制代码](javascript:void(0);)

复制代码

1 <distributionManagement>

2 <repository>

3 <id>bbsnexus</id>

4 <name>bbsidrepository</name>

5 <url>http://127.0.0.1:8081/nexus/content/repositories/bbsid</url>

6 </repository>

7 </distributionManagement>

复制代码

[复制代码](javascript:void(0);)

distributionManagement包含repository和snapshotRepository子元素，前者表示发布版本（稳定版本）jar包的仓库，后者表示快照版本（开发测试版本）的仓库。

这两个元素都需要配置id、name和url，id为远程仓库的唯一标识,很重要，name只是为了方便人阅读，关键的url表示该仓库的地址。

往远程仓库部署jar包的时候，需要认证，配置认证的方式为id,一定要与server的id必须一致。

如果项目当前的版本是快照版本，则部署到快照版本的仓库地址，否则就部署到发布版本的仓库地址，因为这里只是测试演示，前面只创建了Release版本的仓库bbsid，所以省略了snapshotRepository。

配置正确后，定位到要上传的项目目录，运行命令**mvn clean deploy**，Maven就会将项目打包生成的jar包部署到配置对应的远程仓库中。

2）**上传第三方jar包到远程仓库**

**方式一  (假设发布仓库为bbsid,发布Jar包为zbb-sms-0.0.1.jar)：**

      A.  settings.xml中配置认证信息。

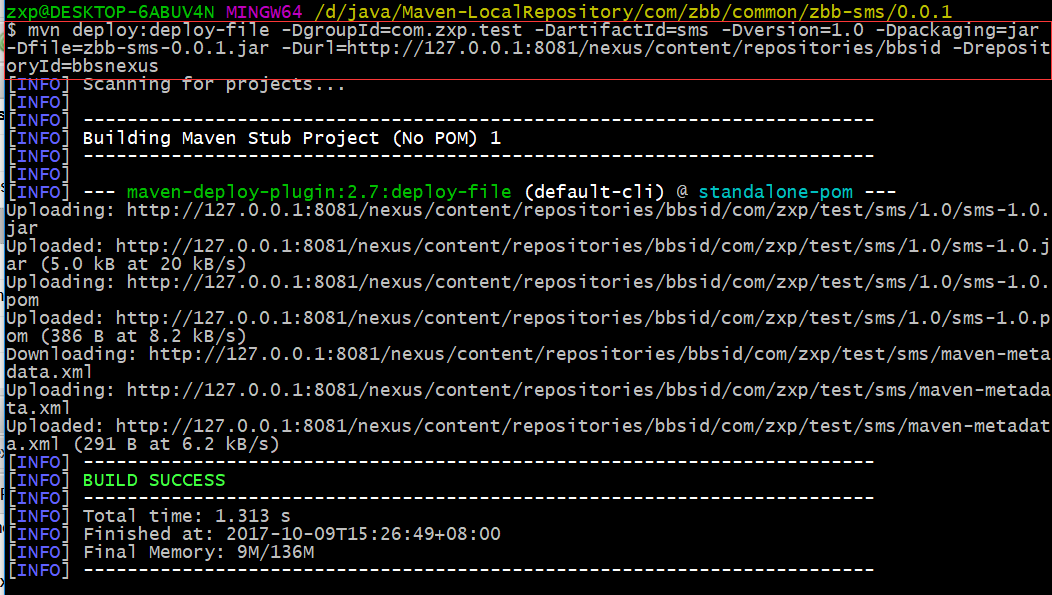
      B. 定位到要上传的jar包的目录，执行“ mvn deploy:deploy-file -DgroupId=com.zxp.test -DartifactId=sms -Dversion=1.0 -Dpackaging=jar -Dfile=zbb-sms-0.0.1.jar -Durl=http://127.0.0.1:8081/nexus/content/repositories/bbsid -DrepositoryId=bbsnexus”命令。

        说明：deploy:deploy-file表示发布独立的文件。

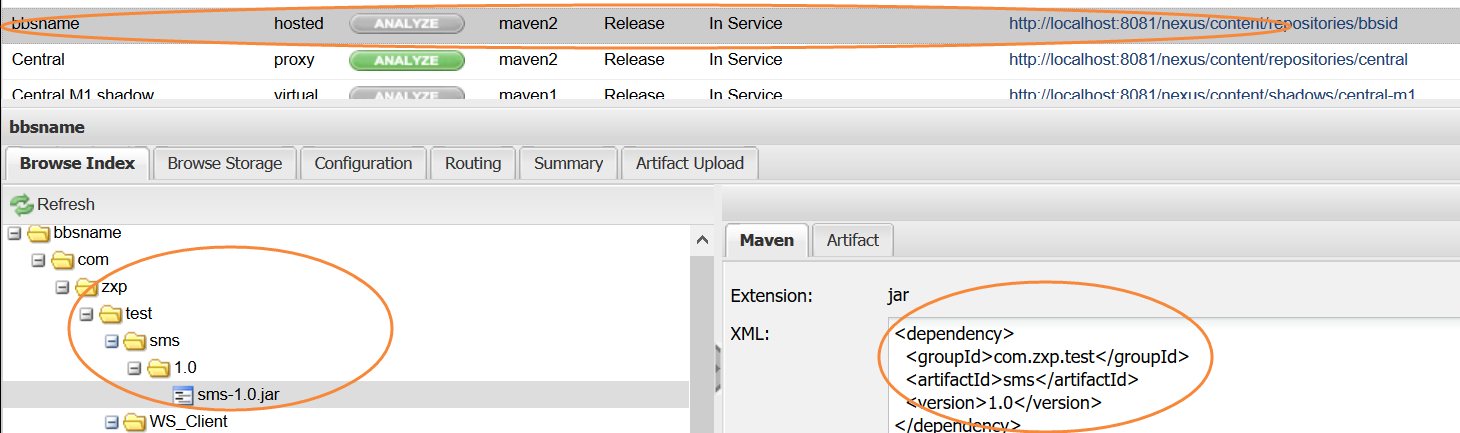
                 groupId、artifactId和version可根据需要设定。（我们要传的包为**zbb-sms-0.0.1.jar**，但是命令里指定**-Dversion=1.0**，**-DartifactId=sms**，所以最终上传到仓库后的名称为**sms-1.0.jar**）

                 url为Nexus服务器中需要上传的仓库路径。

                repositoryId与server的id必须一致。

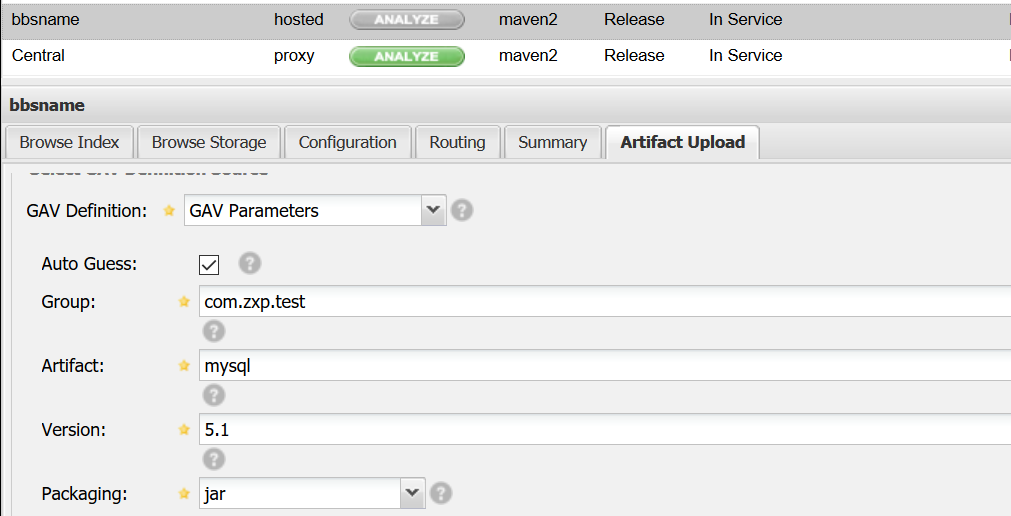


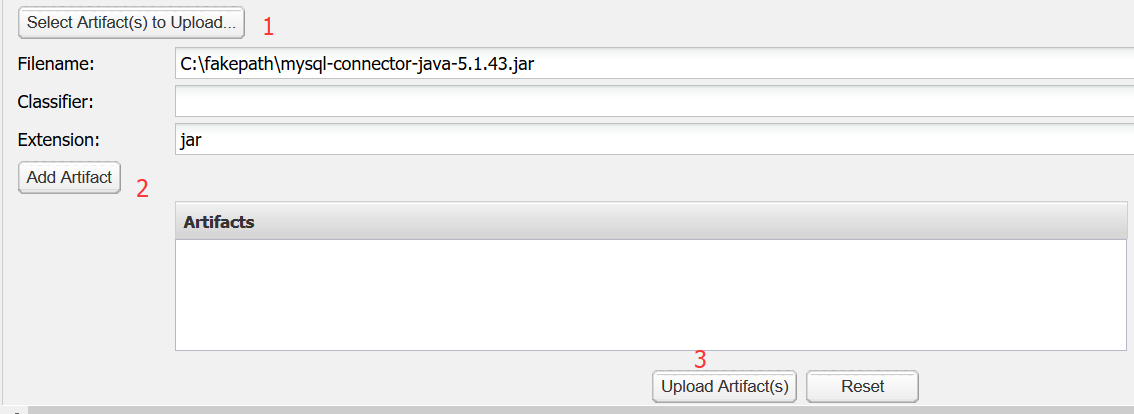
上传到仓库中的结果如下图示：



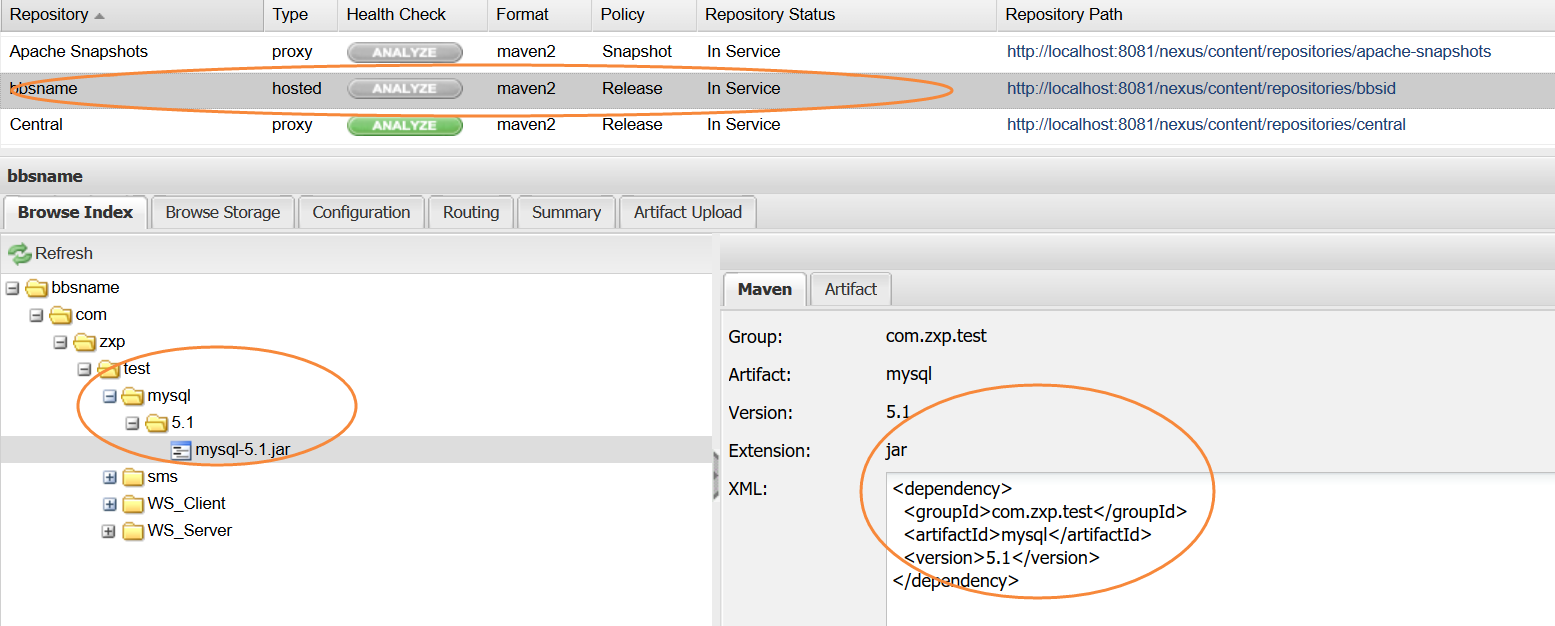
**方式二 Nexus控制台直接上传jar包 (假设发布仓库为bbsid,发布Jar包为mysql-connector-java-5.1.43.jar)：**

 在Repositories列表中选择Releases，点Artifact Upload,指定GAV Definition为“GAV Parameters”，然后输入相应的groupId、artifactId和version、Packaging，再点击“Select Artifact(s) to Upload...”选择指定的Jar文件，点击“Add ActifactId”添加到Actifacts框，最后点“Upload Artifact(s)”即可。

****



上传到仓库中的jar如下图示：



参考文章：http://www.cnblogs.com/luotaoyeah/p/3791966.html

                  http://www.cnblogs.com/AlanLee/p/6198413.html

                  http://blog.csdn.net/wang379275614/article/details/43940259