使用Nexus搭建Maven私服

在实际的企业开发中经常会遇到的问题：在进行Maven项目开发时，所需要的构件都是通过Maven的中央仓库或者第三方的Maven仓库下载到本地，那么企业内部开发使用也会遇到一些问题，团队内的每个使用者都需要去下载相应的依赖包或者插件，效率低下，浪费带宽，网速慢，你可能需要花很长的时间来下载所需的jar，工作效率低，应该项目进度。并且若存在模块之间的依赖开发，那么snapshot版本是不能够被团队伙伴很方便的获取，即：不利于公共构件的管理和维护。另外在实际开发过程中，有些第三方的jar包版本可能在中央仓库里面不存在，或者更新不及时，则获取不到这个jar包。那这些实际问题需要如何解决呢？

解决方案：搭建企业内部的Maven私服。

优点：

1. 私服仓库是本地的，下载的速度远高于从远程下载。
2. 可自行进行构件的管理和维护，包括第三方构件以及项目模块中所依赖的自助开发的公共构件。具体流程如下图1所示：

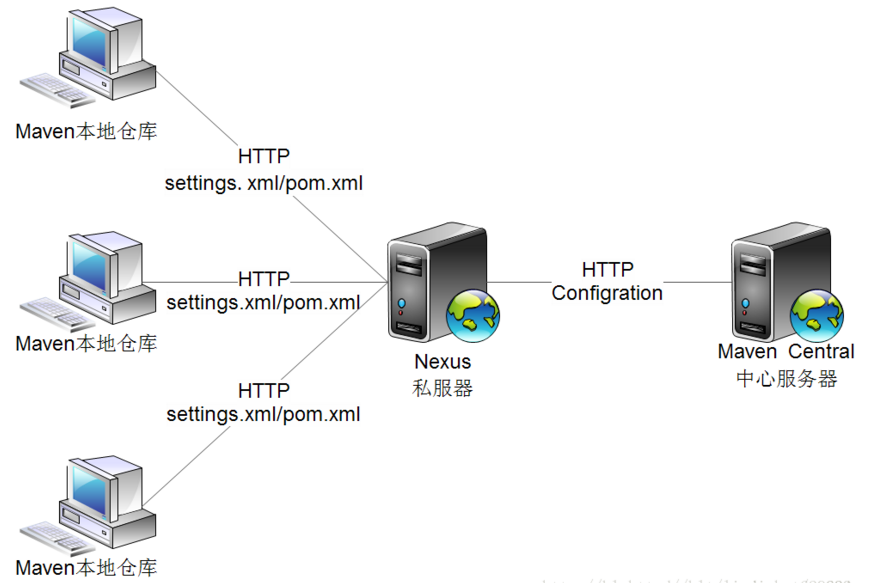


图1

在上图中，搭建私服后，所有的依赖从私服下载，私服会自动判定，若私服库里没有所需的资源，则私服会自动去远程中央仓库下载，若私服已包含所需资源，则可以通过内网提供给使用者，大大提高工作效率。

下面将根据解决方案，学习如何搭建Maven私服。

## Sonatype Nexus

我们使用专门的 Maven 仓库管理软件来完成私服的搭建，比如：Apache Archiva，Artifactory，Sonatype Nexus。下面我们重点介绍如何使用 Sonatype Nexus进行私服搭建。

Nexus是一个非常强大的Maven仓库管理器，极大地简化了自己内部仓库的维护和外部仓库的访问。利用Nexus就可以只在一个地方就能够完全控制访问和部署在所维护仓库中的每个Artifact。并且它不需要数据库，它使用文件系统加Lucene来组织数据。最大的一个功能就是通过项目的SNAPSHOT版本管理，来进行模块间的高效依赖开发。官网：http://www.sonatype.org/nexus/。注意：Nexus 专业版是需要付费的，这里我们下载开源版 Nexus OSS。

Nexus 提供两种安装包：（下载地址：http://www.sonatype.org/nexus/go）

1. 内嵌 Jetty 容器的 bundle 包（只需有JRE就可直接运行）
2. war包（须将其发布到Web容器中即可使用）

补充说明：提供所有版本的下载列表（<http://www.sonatype.org/nexus/archived/>），可选择不同版本，个人建议不使用3.x的版本，因为无法更新索引，建议使用的2.x的版本。

## 1.2 Maven私服搭建

### 1.2.1 安装配置Nexus环境

1. 安装环境准备
2. 服务器环境
   1. CentOS 6.4 64位
   2. JDK 1.8
3. nexus-2.12.0-01-bundle.tar.gz
4. 安装配置
5. 解压tar

tar -zxvf  nexus-2.12.0-01-bundle.tar.gz

1. 配置文件：conf/nexus.properties
2. 开放8081端口

vim /etc/sysconfig/iptables

service iptables restart  #最后重启防火墙使配置生效

1. 启动Nexus

命令：./usr/local/nexus2.12/nexus-2.12.0-01/bin/nexus start

注意：配置 bin/nexus (RUN\_AS\_USER=root)

1. 访问Nexus

http://服务器IP:8081/nexus/

默认端口：8081

默认用户名/密码：admin/admin123

### 1.2.2 Nexus仓库的设置

因为新搭建的Neuxs环境只是一个空的仓库，那么首先就需要与远程中心库进行同步，进行私有仓库索引更新。更新的方式有两种：自动方式、手动方式。

1. 自动更新方式

开启远程索引自动更新下载（Central Repository），设置Download Remote Indexes：True（远程下载Central仓库的索引）。Nexus会自动从远程中央仓库下载索引文件, 在Central上右键点击Update/Rapair Index，下载index文件。一般远程仓库都比较大，构件较多，索引文件会很大，比如：从http://repo1.maven.org/maven2下载后 就有2G，因此需要的时间就比较长。可以进入Scheduled Tasks查看任务的执行情况，当执行完成时，远程仓库的索引就已经建立完毕了。更新完索引后可在Browse Index页签点击刷新按钮就可显示所有更新后的索引。那么在这里由于网络原因，我们不推荐自动方式，强烈推荐手动方式。

1. 手动更新方式
2. **下载索引文件**

* nexus-maven-repository-index.gz

下载地址：http://repo.maven.apache.org/maven2/.index/

* nexus-maven-repository-index.properties

下载地址：http://repo.maven.apache.org/maven2/.index/

* indexer-cli-5.1.0.jar

下载地址：http://maven.outofmemory.cn/org.apache.maven.indexer/indexer-cli/5.1.0/

1. **解压缩索引文件**

将上面三个文件（.gz & .properties & .jar）放置到同一目录下，运行命令：  
java -jar indexer-cli-5.1.0.jar -u nexus-maven-repository-index.gz -d indexer

1. **停止Nexus**

./usr/local/nexus2.12/nexus-2.12.0-01/bin/nexus stop

1. **删除原有的索引文件**

将{nexus\_home}\sonatype-work\nexus\indexer\central-ctx下的文件全部删掉，执行命令：rm \*

1. **拷贝索引至central-ctx目录下**

将nexus-maven-repository-index.gz解压后的indexer目录中所有文件，放到sonatype-work\nexus\indexer\central-ctx下面，执行命令：

cp -r \* /usr/local/nexus2.12/sonatype-work/nexus/indexer/central-ctx/

1. **启动Nexus即自动更新索引**

更新完索引后可在Browse Index页签点击刷新按钮就可显示所有更新后的索引，展开索引树，点击到叶子节点--依赖jar包，可查看其依赖配置坐标。

### 1.2.3 Nexus仓库及常规操作介绍

#### 1.2.3.1 Nexus仓库

Nexus 的Repositories，一共默认有8个仓库，如下图2所示：



图2

上图中Nexus的预定义仓库的type类型、policy策略、格式均不同，下面依次进行介绍。

* 四种仓库类型：

1. Hosted Repository：本地仓库，通常我们会部署自己的构件到这一类型的仓库。比如公司的一些构件库等等。简单理解就是：内部项目的发布仓库（如：执行mvn clean install命令发布项目，最终发布的jar包会放入Hosted Repository）
2. Proxy Repository：代理仓库，它们被用来代理远程的公共仓库，如Maven中央仓库。从远程中央仓库中寻找数据的仓库（可以点击对应的仓库的Configuration页签下Remote Storage Location属性的值即被代理的远程仓库的路径）
3. Virtual Repository：虚拟仓库
4. Repository Group：仓库组，用来合并多个hosted/proxy仓库。组仓库用来方便开发人员进行设置的仓库, group仓库可以自由组合其他的仓库为一个组，然后在项目的pom文件的repository可以配置这个仓库。可以说仓库组是为了简化仓库的引用而提出的。

* 两种仓库格式：

1. maven2
2. maven1

* 两种仓库Policy：

1. Release（发布版本，即稳定版本）
2. Snapshot（资源快照，版本相对不稳定）

下面我们再重点介绍几个Nexus预定义仓库：

1. 3rd party

存放第三方构件，一些无法从公共仓库获得的构件，比如一些早期开源项目，没有采用Maven方式构建的，还有一些非开源的组建等等，这些构件中央仓库里没有，需要用户手动上传的构件，那么都会存放在这个仓库里。在仓库的Artifact Upload页签中可以执行上传操作。

1. Apache Snapshots: 使用代理Apache Maven仓库快照版本的构件仓库
2. Central: 用来代理Maven中央仓库中发布版本构件的仓库，策略为Release、只会下载和缓存中央仓库中的发布版本构件。

注意:该库是私服上的中央仓库，并不是Maven的中央库。可以把常用的jar包拷进去，这样客户端就可以从这里下载了。也可作为Maven中央库的代理。

1. Releases: 存放稳定版本的构件。比如：我们完成了一个版本下的组件开发,就可以把它发布到这里
2. Snapshots: 存放快照版本的构件。 比如：在下一个组件版本的所有开发和测试工作完成之前，是不应该发布到release仓库的，但可能其他项目只需用到这个组件的某些接口，只要这些接口完成了开发并通过测试，就可以拿来使用, 从而实现多个项目并行开发。
3. PublicRepositories:  仓库组

如下图3所示，Configuration里面，Ordered Group Repositories，包含了三个仓库Releases，3rd party和Central。也就是说我们在pom.xml引用这个仓库组，其包含的仓库也被引用。

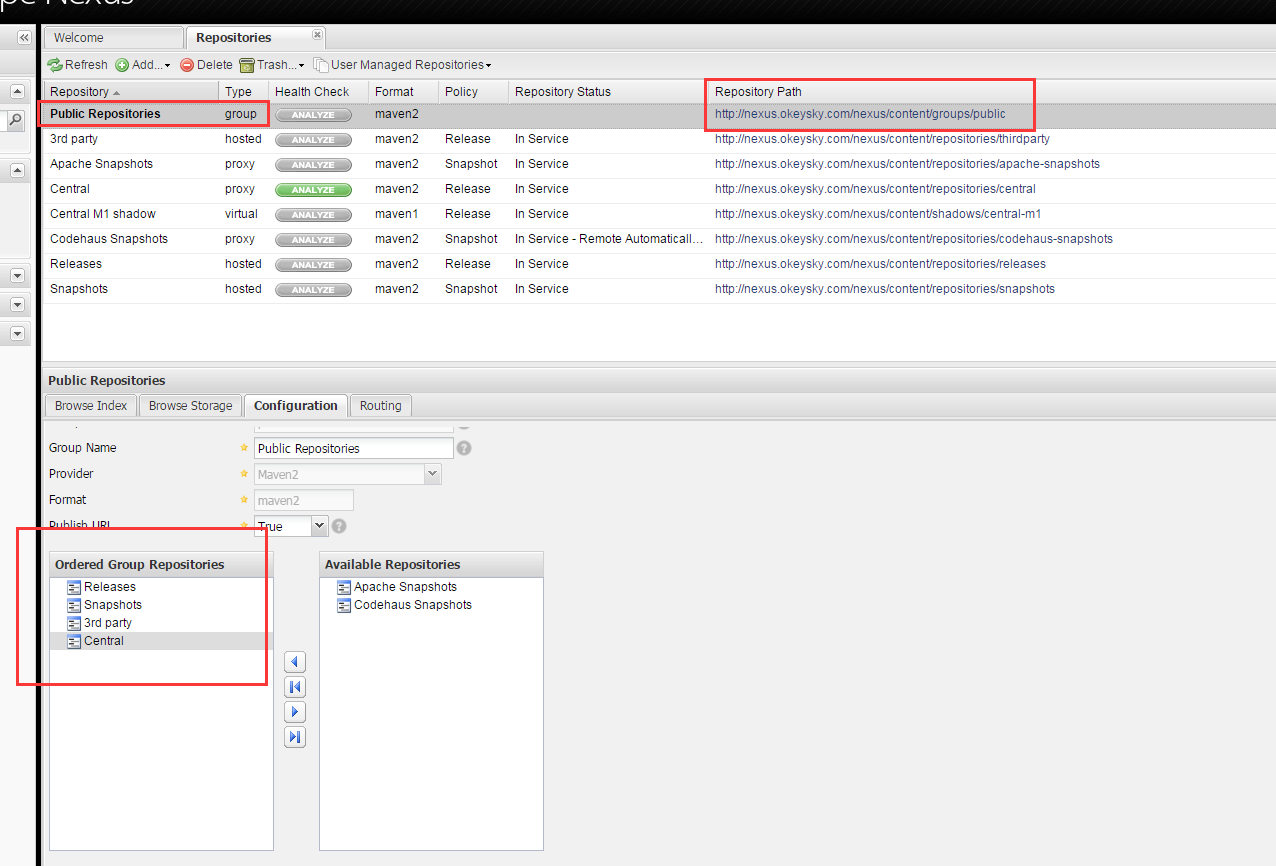


图3

#### 1.2.3.2 常规操作

1. 修改admin用户密码

点击左侧菜单栏的Security——>Users，可看到系统默认的用户。点击set pwd，就可进行新密码的设置，如下图4所示。(需要注意：在发布构件一般需要帐号和密码)

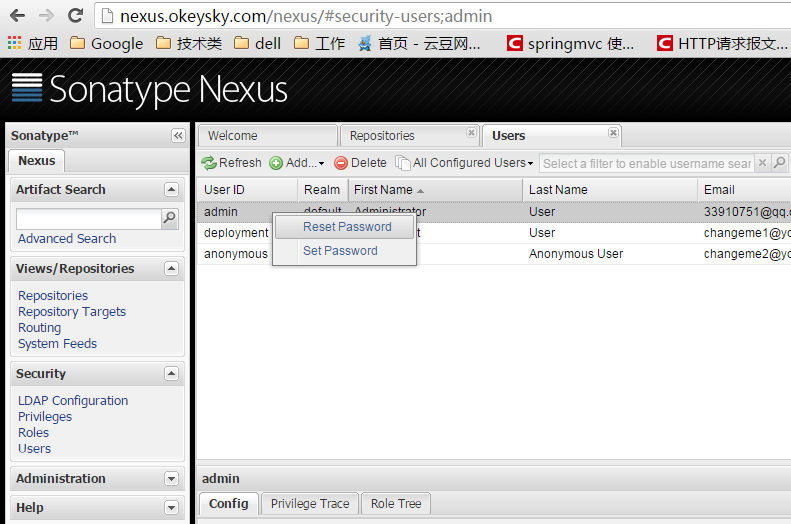


图4

1. 新增用户

点击上方的add按钮，在弹出的界面中，设置必填项：角色，状态为激活。如下图5所示。

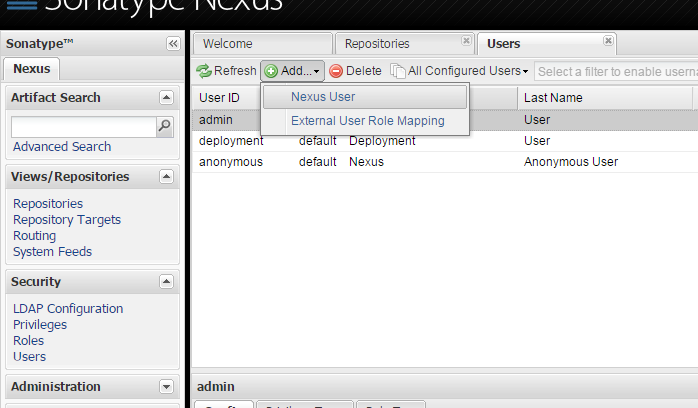


图5

## 1.3 配置本地项目引用并使用私服

### 1.3.1 自动发布构件到Nexus仓库

1. 在工程的pom.xml中添加：distributionManagement

发布仓库（distributionManagement）:负责管理构件的发布包和其他编译生成的支撑文件，那么我们自主开发的jar需要上传到Nexus的时候需要在pom中增加distributionManagement节点进行自动打包上传。

配置上传的仓库有两个：snapshot快照仓库和release发布仓库。snapshot快照仓库用于保存开发过程中的不稳定版本，release正式仓库则是用来保存稳定的发行版本。那么我们在项目中定义一个组件/模块为快照版本，只需要在pom文件中在该模块的版本号后加上**-SNAPSHOT**即可(注意：这里必须是大写)，在开发阶段，我们一般都是设置为快照版本。 maven2会根据模块的版本号(pom文件中的version)中是否带有-SNAPSHOT来判断是快照版本还是正式版本。如果是快照版本，那么在mvn deploy时会自动发布到快照版本库中。关键配置代码如下：

|  |
| --- |
| <distributionManagement>  <repository>  <id>releases</id><!--ID需要与你的release仓库的Repository ID一致-->  <url>http://192.168.74.163:8081/nexus/content/repositories/releases</url>  </repository>  <snapshotRepository>  <id>snapshots</id><!-- ID需要与你的snapshots仓库的Repository ID一致-->  <url>http://192.168.74.163:8081/nexus/content/repositories/snapshots</url>  </snapshotRepository>  </distributionManagement> |

1. 修改本地maven的settings.xml文件，配置用户名/密码，关键配置代码如下：

|  |
| --- |
| <servers>  <server>  <id>releases</id>  <username>admin</username>  <password>admin123</password>  </server>  <server>  <id>snapshots</id>  <username>admin</username>  <password>admin123</password>  </server>  </servers> |

当然若你要上传构件到其他仓库，可依照如下代码进行配置：

|  |
| --- |
| <server>    <id>thirdparty</id>    <username>admin</username> <password>admin123</password>  </server>  <server>    <id>central</id>    <username>admin</username>    <password>admin123</password>  </server> |

1. 在idea中，工程所在目录下执行mvn命令（mvn:deploy）

将最终版本的包拷贝到远程的Repository，使得其他的开发者或者工程项目可以共享。所部署的包就自动上传到了Nexus指定的仓库里。比如：上传jar包到snapshots仓库。mvn:deploy在整合或者发布环境下执行：

|  |
| --- |
| mvn deploy:deploy-file -DgroupId=cn.itrip -DartifactId=itrip-beans -Dversion=1.0-SNAPSHOT -Dpackaging=jar -Dfile=D:\ideaworkspace\itripbackend\itripbeans\target\itrip-beans-1.0-SNAPSHOT.jar  -Durl=http://192.168.74.163:8081/nexus/content/repositories/snapshots/ -DrepositoryId=**snapshots** |

* DgroupId和DartifactId构成了该jar包在pom.xml的坐标，项目就是依靠这两个属性定位。
* Dfile表示需要上传的jar包的绝对路径。
* Durl私服上仓库的位置，打开nexus——>repositories菜单，可以看到该路径。
* DrepositoryId服务器的表示id，在Nexus的configuration可以看到。
* Dversion表示版本信息

上传成功后，在Nexus界面点击**snapshots**仓库可以看到这个包。还有一种上传构件的方式是手动上传，即通过Nexus的界面操作，较为简单，在此不再赘述。

### 1.3.2 配置Maven从Nexus下载构件

1. 首先配置Maven镜像

配置镜像目的就是让Maven只使用私服，修改Maven的setting.xml文件，具体配置如下。

|  |
| --- |
| <mirror>  <id>nexus</id>  <mirrorOf>\*</mirrorOf>  <name>local nexus repository</name>  <url>http://192.168.74.163:8081/nexus/content/groups/public/</url>  </mirror> |

在pom文件不做任何配置的情况下，默认是使用id为central的Maven中央库进行配置的。那么以上配置只取代了pom中的下载仓库，覆盖了maven默认的配置。mirrorOf配置为\*，表示这个镜像配置适用于所有的仓库，是所有仓库的一个镜像，maven无论要去那个仓库下载构件都会跳转到镜像URL（此时配置的是nexus的group类型的仓库地址，该组仓库包含了其他几个类型的仓库）去下载，不会到中央资源库下载。当然mirrorOf也可以配置一个或多个仓库，多个仓库配置值要与对应的仓库id一致，以逗号分隔。

1. 修改项目的pom文件，设置下载仓库（repostories）

|  |
| --- |
| <repositories>  <repository>  <id>public</id>  <name>Team Maven Repository</name>  <url>http://192.168.74.163:8081/nexus/content/groups/public/</url>  <releases>  <enabled>true</enabled><!—是否支持更新—>  </releases>  <snapshots>  <enabled>true</enabled>  </snapshots>  </repository>  </repositories> |

注：上述配置只对当前项目有效，若需让本机所有Maven项目均使用该Mavne私服，应该在setting.xml中进行配置。

1. 通过在settings.xml中配置profile元素，可以让本机所有的Maven项目都使用Maven私服。

|  |
| --- |
| <profiles>                 <profile>                     <id>public</id>                     <repositories>                         <repository>                           <id>public</id>                           <name>Nexus</name>                           <url>http://192.168.74.163:8081/nexus/content/groups/public/<url>                           <release><enabled>true</enabled></release>                           <snapshots><enabled>true></enabled></snapshots>                         </repository>                      </repositories>  </profile>  <activeProfiles><!-- activeProfiles用来激活—>                <activeProfile>public</activeProfiles>   </activaProfiles> |