1. 全文目录

2. 前言介绍

此处分为两部分：maven 使用入门以及 nexus 私服环境搭建

2.1. Maven 使用介绍

大部分内容是参考 maven 官方 wiki 以及 maven 权威指南一书整理的，希望了解更多细节内容的请参阅 maven 权威指南一书。

2.2. nexus 私服环境搭建

分别整理了在 windows 平台以及 Linux 平台上的搭建过程，此处通过一个 hellomaven 工程演示 nexus 的配置使用，希望了解更多 nexus 相关的可以参阅 nexus 的官方网站 http://www.sonatype.com/即可。

3. Maven 使用入门

3.1. Maven 基本介绍

3.1.1. Maven 是什么？

3.1.1.1. Apache maven 官方定义

Maven 是一个项目管理工具，它包含一个项目对象模型（Project Object Model），一组标准集合，一个项目生命周期（Project Lifecycle），一个依赖管理系统（Dependency Management System），和用来运行定义在生命周期阶段（Phase）中插件（Plugin）目标（Goal）的逻辑。

3.1.1.2. Maven 普遍认识定义

Maven 是一个采用纯 Java 编写的开源项目管理工具，Maven 采用了一个被称之为 Project Object Model(POM)概念来管理项目，所有的配置项目信息都被定义在一个叫做 pom.xml 的文件中，通过该文件，maven 可以管理项目的整个生命周期，包括编译、构建、测试、发布、报告等等。——将 maven 理解成类似 ant 的构建工具，但是 maven 不局限此。

关键词为：构建工具、项目对象模型 pom、依赖管理系统、插件 plugin、目标 goal

3.1.2. Maven 默认约定

在 maven 权威指南一书提到“约定优于配置”是一个简单的概念。系统、类库、框架应该假定合理的默认值，而非要求提供不必要的配置。Maven 默认约定为：

在没有自定义的情况下，源代码假定是在${basedir}/src/main/java/目录下，资源文件是在${basedir}/src/main/resources/目录下；

测试的源代码是在${basedir}/src/test/java/目录下，测试对应的资源文件是放在${basedir}/src/test/resources/目录下；

编译后的 class 文件是放在${basedir}/target/classes/目录下，打包后的 jar 文件是放在${basedir}/target/目录下。

Maven 的核心插件使用了一组通用的约定，以用来编译源代码、打包可分发的构建 jar 包或 war 包文件、生成 web 站点，还要许多其他过程。

3.1.3. 项目概念模型要点

Maven 的项目概念模型 POM 包括的要点为：

依赖管理：

由于项目是根据一个包含组标识符，构建标识符和版本的唯一的坐标定义的，项目之间可以使用这些坐标来声明依赖。

远程仓库：

和项目依赖相关的，我们可以使用定义在项目对象模型 POM 中的坐标来创建 maven 构建的仓库。

全局性构建逻辑重用：

插件被编写成和项目模型对象 POM 一起工作，它们没有被设计成操作某一个已知位置上的特定文件，一切都被抽象到模型中，插件配置和自定义行为都在模型中进行。

工具可移植性和集成

像 Eclipse 这样的工具现在又共同的地方来找到项目的信息，在 maven 出现之前，每个 IDE 都有不同的方法来存储实际上是自定义项目对象模型 POM 的信息，maven 只是标准化了这种描述，后边每个 IDE 仍旧继续维护它自定义项目文件，但是这些文件通过 maven 的插件很容易就生成了。即为 maven 工程到 Eclipse 工程的识别过程，创建出 IDE 识别文件。

便于搜索和过滤构件

像 nexus 私服这样的工具允许你使用存储在 POM 中的信息对仓库中的内容进行索引和搜索处理。一般建议在局域网内部搭建一个私服 nexus 环境。

此小节从《maven 权威指南》整理过来的，建议有空的事情过一遍原著。

3.1.4. Maven 和 Ant 对比

此处不再过多比较这两个构建工具，ant 能提供给你的，maven 也可以提供，并且 maven 功能更加强大和丰富。现在主流的开源项目都从 ant 迁移到了 maven 进行管理。

3.2. Maven 环境安装

Maven 已经支持各类操作系统平台，此处介绍常用平台。Windows 下 win7 以及 Linux 平台 centos 系统下 maven 的安装过程，其它平台比如 Ubuntu 下安装过程和 centos 系统下大致相同，此处不再介绍。

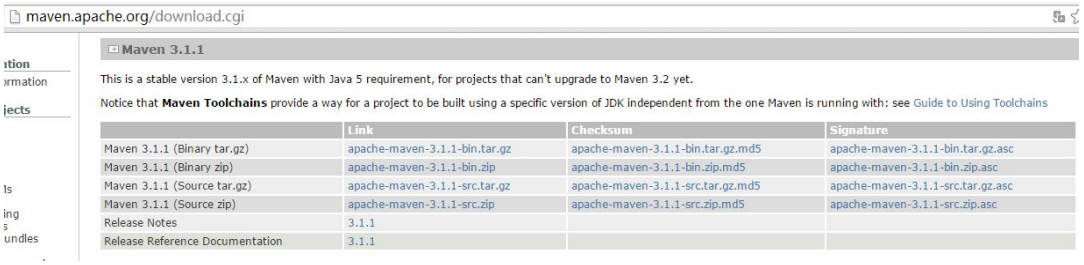
3.2.1. 前置安装

确保已经安装了配置好了 Java 开发环境，验证是否安装正确的 Java 环境可以执行 java -version 命令，另外查看环境变量，可以通过 echo %JAVA_HOME%/echo %PATH%/echo %CLASSPATH%（在 windows 平台），或者通过 echo $JAVA_HOME/echo $PATH/echo $CLASSPATH 命令（在 Linux 平台）查看 Java 环境变量。

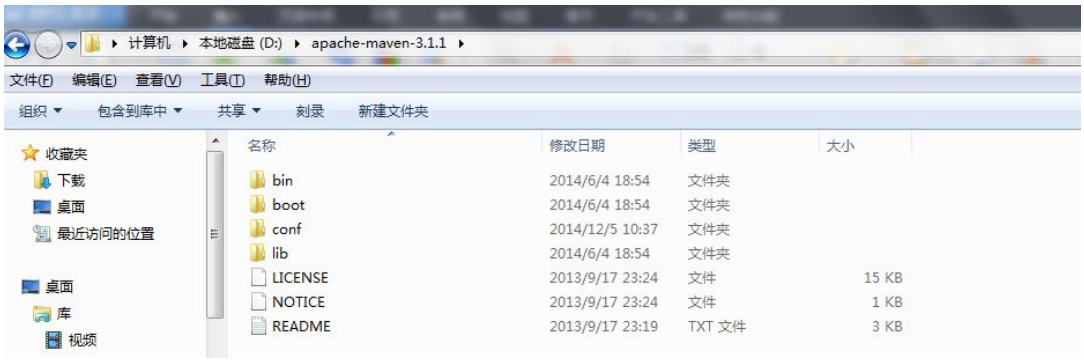
3.2.2. 在 windows7 系统上安装 maven

从 maven 官方地址下载对应的*.tar.gz 压缩包，此处下载为 3.1.1 版本。

下载地址为：http://maven.apache.org/download.cgi

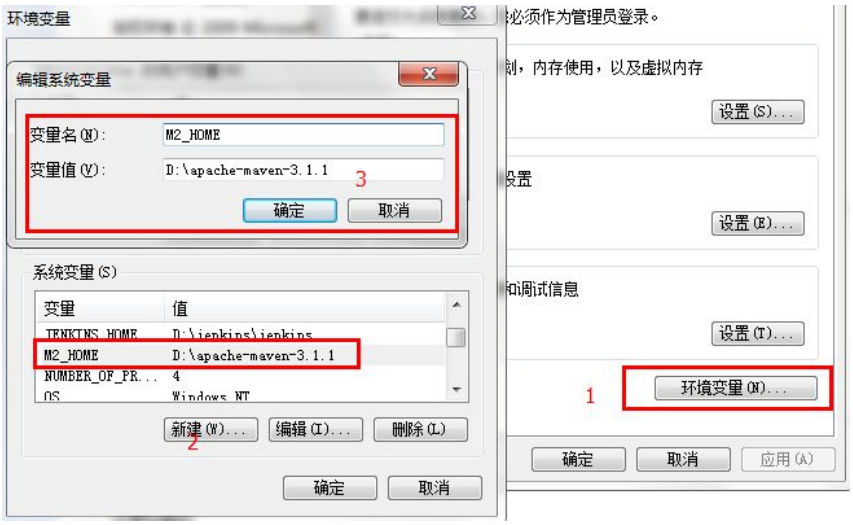


点击对应的版本下载即可，解压到本机 d 盘目录下为：

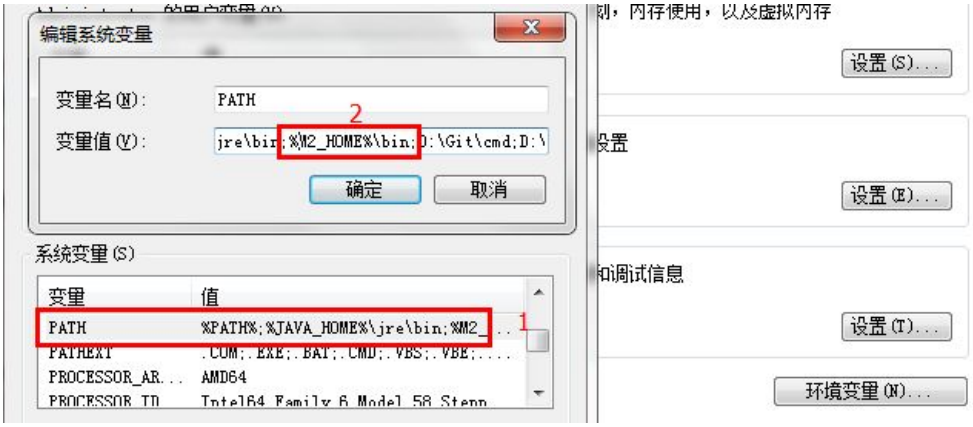


解压完毕后，设置 maven 环境变量为：

M2_HOME:D:\apache-maven-3.1.1

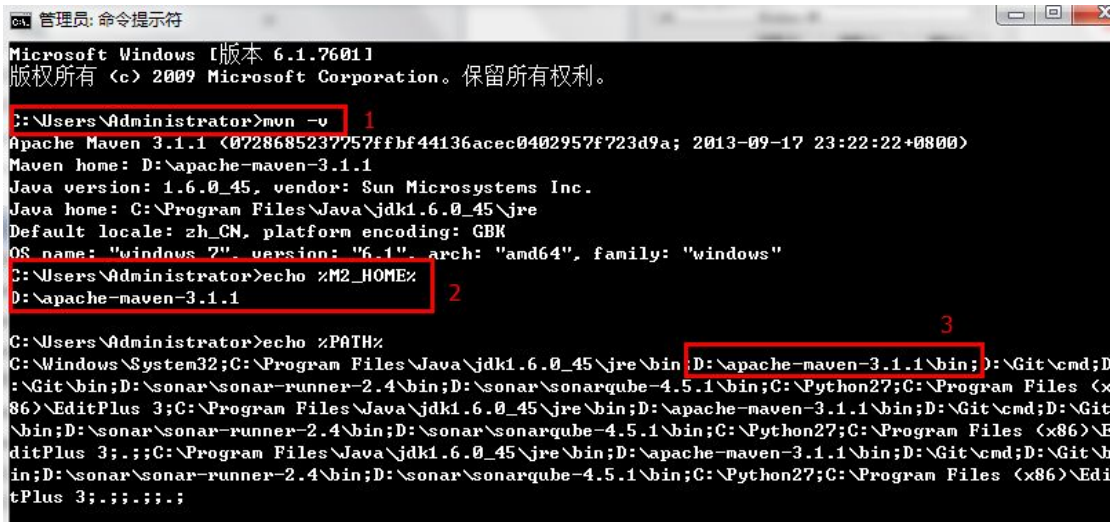


Path 后边追加上 M2_HOME/bin



3.2.3. 在 windows7 系统上验证 maven

设置完毕后，保存打开 cmd 窗口，检查下 maven 环境变量是否设置 ok



在 windows 平台上执行该命令观察到图示效果则代表 maven 环境搭建完成。

mvn -v 查看 maven 版本号信息

echo %M2_HOME% 查看 maven 设置的环境变量，此处指到 maven 的根目录

echo %PATH% 查看当前系统的 path 环境变量值，此处增加 maven\bin 目录，主要是为了可以 cmd 里边执行 mvn 脚本。

3.2.4. 在 Linux 系统上安装 maven

在 Linux 上安装 maven 下载过程和 windows 上一样，从官方网站下载回来的 apache-maven-3.1.1-bin.tar.gz 包，在 centos 系统下先将*.tar.gz 包上传到/usr/local 目录下，然后执行 tar -xzvf *.tar.gz 进行解压处理。解压后效果为：

```
[root@xf_test_77 ~]# cd /usr/local/
[root@xf_test_77 local]# ll
total 272392
drwxr-xr-x   7 root    root         4096 Dec  8 21:03 apache-maven-3.1.1
-rw-r--r--   1 root    root      5494427 Dec  7 21:37 apache-maven-3.1.1-bin.tar.gz
drwxr-xr-x   2 root    root         4096 Dec 11 13:49 bin
```

为了后期方便做 maven 版本升级，而又无需变更 maven 的环境变量，此处建一个软连接，通过 ln -s 源文件 软连接 命令进行设置处理。命令操作过程为：

```
[root@xf_test_77 local]# ll|grep maven
drwxr-xr-x   7 root    root         4096 Dec  8 21:03 apache-maven-3.1.1
-rw-r--r--   1 root    root      5494427 Dec  7 21:37 apache-maven-3.1.1-bin.tar.gz
lrwxrwxrwx   1 root    root           19 Dec  8 20:29 maven -> apache-maven-3.1.1/
```

因此 maven 的安装目录为：/usr/local/maven，接着我们在 centos 下设置 maven 的环境变量，设置过程为：编辑/etc/profile 文件在文件末尾追加上：

```
#java
export JAVA_HOME=/usr/local/java
export PATH=$PATH:$JAVA_HOME/bin
export CLASSPATH=./:$JAVA_HOME/lib:$JAVA_HOME/jre/lib
#jenkins
export JENKINS_HOME=/usr/local/jenkins
#maven
export M2_HOME=/usr/local/maven
export PATH=$PATH:$M2_HOME/bin
#nexus
export JRE_HOME=$JAVA_HOME/jre
export NEXUS_HOME=/usr/local/nexus
export RUN_AS_USER=root
```

此处观察到设置了 java 的环境变量以及 maven 的环境变量。

主要是设置 M2_HOME 以及在 PATH 上追加上$M2_HOME/bin 即可，设置完毕保存退出即可，接着通过 source /etc/profile 命令将刚设置的环境变量生效。

3.2.5. 在 Linux 系统上验证 maven

在上一步 Linux 上安装设置好了 maven 之后，此处验证下 maven 的安装。在命令上里边执行 mvn -v 参数即可，查看效果为：

```
[root@xf_test_77 local]# mvn -v
Apache Maven 3.1.1 (0728685237757ffbf44136acec0402957f723d9a; 2013-09-17 23:22:22+0800)
Maven home: /usr/local/maven
Java version: 1.7.0_71, vendor: Oracle Corporation
Java home: /usr/local/jdk1.7.0_71/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "2.6.32-431.el6.x86_64", arch: "amd64", family: "unix"
```

此处输出了当前 maven 的环境变量 M2_HOME 以及 Java 环境变量信息以及当前操作系统 os 的版本以及内核信息。

3.2.6. 获得 maven 帮助

碰到 maven 相关问题先到 Apache maven 官方网站查阅相关资料，地址为：http://maven.apache.org 上边有丰富的 maven wiki 相关资料。

另外可以使用 maven 提供的 help 插件了解相关细节。Maven 的 help 插件能让你列出活动的 maven profile 显示一个实际 POM（effective POM），打印出 settings（effective settings）或者列出 maven 插件的属性。

Maven 的 help 插件有四个目标分别为：

mvn help:active-profiles 列出当前构建中活动的 profile（项目的、用户的以及全局的）

mvn help:effective-pom 显示当前构建的实际 POM，包含活动的 profile

mvn help:effective-settings 打印出项目的实际 settings，包括从全局的 settings 和用户级别的 settings 继承的配置。

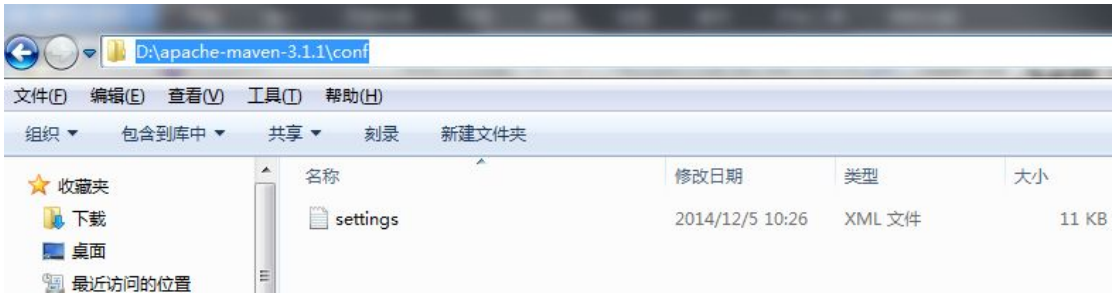
mvn help:describe 描述插件的属性，它不需要在项目目录下运行，但是你必须提供你想要描述插件的 groupId 和 artifactId。

3.3. Maven 配置文件

Maven 配置文件分全局的和用户的，全局放在 maven 的安装目录下，用户的放在用户目录下的~/.m2/settings.xml。

3.3.1. Maven 配置文件配置项

Maven 全局配置文件放在%M2_HOME%/conf/目录下，名称为：settings.xml



Maven settings.xml 配置文件大项分 10 类，配置结构为：

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">

    <localRepository/>

    <interactiveMode/>

    <usePluginRegistry/>

    <offline/>

    <pluginGroups/>

    <servers/>

    <mirrors/>

    <proxies/>

    <profiles/>

    <activeProfiles/>

</settings>
```

3.3.1.1.<localRepositoy/>

<localRepository/>配置 maven 本地仓库地址

3.3.1.2.<interactiveMode/>

<interactiveMode/>是否打开 maven 交互 interact 模式，默认为 false

3.3.1.3.<usePluginRegistry/>

<usePluginRegistry/>是否使用~/.m2/目录下的 plugin-registry.xml 文件来管理 maven plugin 依赖 jar 的 version 版本号，默认是 true

3.3.1.4.<offline/>

<offline/>假定不想每次 build 都去连外部仓库，则设置为 true，否则默认为 false，每次 build 都会去连接外部仓库，此处作用是在没有网络的情况下可以设置为 true

3.3.1.5.<pluginGroups/>

<pluginGroups/>plugin 插件 group 组标签，标签里边是<pluginGroup/>组合，举例用法为：

```
<pluginGroups>
    <pluginGroup>org.mortbay.jetty</pluginGroup>
</pluginGroups>
```

因此在执行 maven 命令的时候可从

org.mortbay.jetty:jetty-maven-plugin:run

简写成 mvn jetty:run，节省了不少事情。

此处默认装配了 org.apache.maven.plugins 和 org.codehaus.moj0 因此我们执行 maven 命令都是 mvn clean/mvn eclipse:eclipse/mvn install 简短的方式。

3.3.1.6.<servers/>

<servers/>

此处配置连接 nexus 私服身份信息，

<servers><server></server></servers>，配置参数介绍为：

<servers>

<server>

<id>server001</id> 此处 id 连接 neuxs 私服 mirror 镜像或者 repository 仓库 id

<username>my_login</username> 用户名

<password>my_password</password> 密码

<privateKey>${user.home}/.ssh/id_dsa</privateKey> 私钥

<passphrase>>some_passphrase</passphrase>

<filePermissions>664</filePermissions> 文件读写操作权限

<directoryPermissions>775</directoryPermissions> 目录权限

<configuration></configuration>

</server>

</servers>

此处注意假定你希望通过 privateKey 来验证身份，则不填写 password，否则 privatekey 将不会生效，这是 maven 约定的。

3.3.1.7.<mirrors/>

<mirrors/> maven 镜像配置，里边可以配置多个镜像

<mirrors>

<mirror>

<id>planetmirror.com</id> 镜像 id 标识

<name>PlanetMirror Australia</name> 镜像名称

<url>http://downloads.planetmirror.com/pub/maven2</url> 镜像地址

<mirrorOf>central</mirrorOf> 仓库设定，一般设定为*通配即可

</mirror>

</mirrors>

在执行 maven build 的过程中 maven 会通过 url 去请求 nexus 私服，此处 id 和 name 只要在各个 mirror 之间区分不重复即可，mirrorOf 匹配到 nexus 私服的 repository 仓库即可，比如中央仓库用 central，或者直接配置成通配的*代表私服 nexus 上的全部仓库。

此处 maven 官方文档指出没必要将 mirrorOf 配置成 mirror 的 id 值。

3.3.1.8.<proxies/>

<proxies/>配置 maven 代理相关内容，配置结构为：

```
<proxies>
      <proxy>
        <id>myproxy</id>
        <active>true</active>
        <protocol>http</protocol>
        <host>proxy.somewhere.com</host>
        <port>8080</port>
        <username>proxyuser</username>
        <password>somepassword</password>
        <nonProxyHosts>*.google.com|ibiblio.org</nonProxyHosts>
      </proxy>
  </proxies>
```

3.3.1.9.<profiles/>

在 maven 的 settings.xml 配置文件中<profiles/>是比较复杂的一项，profile 元素 element 标签配置可以理解为是 maven 工程 pom.xml 中 profile 元素配置项的缩减版，settings.xml 中的 profile 包含四大配置小项，分别为：

<activation/> 激活

<repositories/> 仓库

<pluginRepositories/> 插件仓库

<properties/> 属性元素

Maven 约定假定在一个 profile 在 settings.xml 文件中被激活 active，则它将覆盖/重载 maven 工程中的 pom.xml 或 profiles.xml 文件相同 profile-id 的参数配置。

此处理解起来比较困难，先简单理解为这边就是个性配置，profiles。

<activation/>

<profiles>

```
    <profile>
      <id>test</id>
      <activation>
        <activeByDefault>false</activeByDefault>
        <jdk>1.5</jdk>
        <os>
          <name>Windows XP</name>
          <family>Windows</family>
          <arch>x86</arch>
          <version>5.1.2600</version>
        </os>
        <property>
          <name>mavenVersion</name>
          <value>2.0.3</value>
        </property>
        <file>
          <exists>${basedir}/file2.properties</exists>
          <missing>${basedir}/file1.properties</missing>
        </file>
      </activation>
      ...
    </profile>
  </profiles>
```

一个 profile 要对应一个 activeProfile 激活，这边根据 profile-id 来标识，

更多关于 activation 参数配置，可以执行 mvn help:active-profiles 命令查看。

<repositories/>

此处仓库集合为 projects 的远程仓库集合，maven 构建 build 系统通过这些远程仓库集合来进行本地仓库的构建处理，对于本地仓库而言 maven 称之为插件 plugin 和依赖 dependency，不同的远程仓库包含不同的项目 project 集合，通过 active profile maven 可以查询匹配 release 版本或者 snapshot 版本的 artifact 信息。

```
<profiles>
      <profile>
        ...
        <repositories>
          <repository>
            <id>codehausSnapshots</id>
            <name>Codehaus Snapshots</name>
            <releases>
              <enabled>false</enabled>
              <updatePolicy>always</updatePolicy>
              <checksumPolicy>warn</checksumPolicy>
            </releases>
            <snapshots>
              <enabled>true</enabled>
              <updatePolicy>never</updatePolicy>
              <checksumPolicy>fail</checksumPolicy>
            </snapshots>
            <url>http://snapshots.maven.codehaus.org/maven2</url>
            <layout>default</layout>
          </repository>
        </repositories>
        <pluginRepositories>
          ...
        </pluginRepositories>
        ...
      </profile>
    </profiles>
<pluginRepositories/>
```

此处为插件仓库，配置项和 repositories 相同。

```
<properties/>
```

```
<profiles>
    <profile>
      ...
      <properties>
        <user.install>${user.home}/our-project</user.install>
      </properties>
      ...
    </profile>
  </profiles>
```

此处用的比较极少，有需要直接查询 maven 官方 wiki 即可。

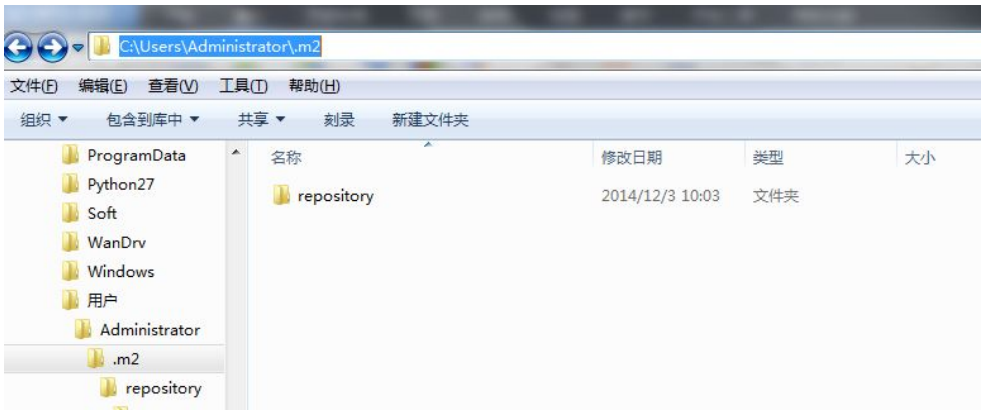
3.3.1.10. <activeProfiles/>

配合<profiles/> 使用，通过 profile-id 标识关联。

3.3.1.11. 配置文件其它

后续配置 maven 的 settings.xml 文件都是在此基础上不断添加更多内容。

这边主要配置下本机 maven 仓库，默认不设置的话在 win7 下是放置在 c 盘目录，这边建议不放在系统 c 盘下。



本机将 maven repository 配置在 d 盘 d:\\maven_repository 目录下

```
<!-- localRepository
 | The path to the local repository maven will use to store artifacts.
 |
 | Default: ${user.home}/.m2/repository
<localRepository>/path/to/local/repo</localRepository>
-->
<!-- 这边设置将本机仓库指向到d:盘目录下 -->
<localRepository>D:\\maven_repository</localRepository>
```

maven default默认行为

其他配置暂时不改动，默认即可。后边说到 nexus 私服如何在 maven 的 settings.xml 配置文件中配置参数项，后边会提到。

3.3.2. Maven 配置文件规则

在 windows 系统下安装完 maven 后一般有两个 settings.xml 配置文件，一个在%M2_HOME%/conf/settings.xml 另外一个在~/.m2/settings.xml，前者为 maven 全局 global 的配置文件，后者为用户级别的配置文件，下边举例说明加载规则：

Maven 作为 global 全局生效，当用户 user_1 使用 maven 在用户 user_1/.m2/目录下也配置了 settings.xml 配置文件，则 user_1/.m2/settings.xml 和 maven global settings.xml 进行 merged 处理，maven 会根据 user_1/.m2/settings.xml 为优先级规则。

以上规则在 Linux 下也亦如此，Linux 下也分用户和 global 级别的。

3.4. 一个简单 maven 项目

3.4.1. Maven demo 介绍

该章节通过一个 maven demo 例子来了解 maven 的基本使用，跑完整个流程后，提供扩展阅读&内部细节章节，此处为拓展阅读强烈建议您细读下从而理解 maven 的几个重点概念。诸如插件和目标、生命周期、坐标、仓库、依赖管理以及站点生成和报告。

3.4.2. 创建一个简单的 maven 项目

此处约定 maven 项目名称为 hellomaven，groupId 为 com.maven.demo，包名 packagename 采用 maven 默认约定的处理。

创建 maven 项目命令为：mvn archetype:create archetype 是一个 plugin 插件，create 是该插件的目标 goal，创建该 maven demo 项目完整命令为：

mvn archetype:create -DgroupId=com.maven.demo -DartifactId=hellomaven

```
[INFO] ------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:RELEASE
[INFO] ------------------------------------------------------------------------
[INFO] Parameter: groupId, Value: com.maven.demo
[INFO] Parameter: packageName, Value: com.maven.demo
[INFO] Parameter: package, Value: com.maven.demo
[INFO] Parameter: artifactId, Value: hellomaven
[INFO] Parameter: basedir, Value: /usr/local/test
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: /usr/local/test/hellomaven
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2.493s
[INFO] Finished at: Wed Dec 17 11:53:36 CST 2014
[INFO] Final Memory: 14M/480M
[INFO] ------------------------------------------------------------------------
```

执行命令输出信息到控制台，此处可以看到 maven demo 大致的结构目录，包括为：

groupId 组 id，设定为 com.maven.demo

artifactId 标识 id，设定为 hellomaven

packageName 包名，设定为 com.maven.demo，此处 maven 约定的规则

basedir 当前工程目录，此处为/usr/local/test

version 版本号默认规则为 snapshot 快照版本。

maven 会自动帮助创建 pom.xml 文件放在工程根目录下，另外也会创建 src/main/java 以及 src/main/resources，测试相关的 src/test/java 以及 src/test/resources

3.4.3. 构建一个简单的 maven 项目

创建完 maven 项目后，观察到 maven demo 项目工程结构为：

```
drwxr-xr-x 3 root root 4096 Dec 17 11:53 hellomaven
[root@xf_test_77 test]# cd hellomaven/
[root@xf_test_77 hellomaven]# ll
total 8
-rw-r--r-- 1 root root  756 Dec 17 11:53 pom.xml
drwxr-xr-x 4 root root 4096 Dec 17 11:53 src
```

构建一个 maven demo 项目，通过命令 mvn install 即可，该命令要在工程目录下执行，install 是 maven 约定默认的安装 install 阶段，maven 默认的生命周期阶段。

本地构建完成后，生成文件以及安装 install 到本地 maven 仓库为：

```
Running com.maven.demo.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.007 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ hellomaven ---
[INFO] Building jar: /usr/local/test/hellomaven/target/hellomaven-1.0-SNAPSHOT.jar
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ hellomaven ---
[INFO] Installing /usr/local/test/hellomaven/target/hellomaven-1.0-SNAPSHOT.jar to /usr/local/maven/repo/com/maven/demo/hellomaven/1.0-SNAPSHOT/hello
maven-1.0-SNAPSHOT.jar
[INFO] Installing /usr/local/test/hellomaven/pom.xml to /usr/local/maven/repo/com/maven/demo/hellomaven/1.0-SNAPSHOT/hellomaven-1.0-SNAPSHOT.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2.150s
[INFO] Finished at: Wed Dec 17 11:59:29 CST 2014
[INFO] Final Memory: 16M/606M
[INFO] ------------------------------------------------------------------------
```

输出信息观察到执行了一个单元测试 test，另外 install 安装到了本地的 maven 仓库，地址为 /usr/local/maven/repo/目录下的两个文件，一个为 jar 另外一个为 pom 文件：

com/maven/demo/hellomaven/1.0-SNAPSHOT/hellomaven-1.0-SNAPSHOT.jar

com/maven/demo/hellomaven/1.0-SNAPSHOT/hellomaven-1.0-SNAPSHOT.pom

截图为：

```
[root@xf_test_77 hellomaven]# cd  /usr/local/maven/repo/com/maven/demo/hellomaven
[root@xf_test_77 hellomaven]# ll
total 8
drwxr-xr-x 2 root root 4096 Dec 17 11:59 1.0-SNAPSHOT
-rw-r--r-- 1 root root  282 Dec 17 11:59 maven-metadata-local.xml
[root@xf_test_77 hellomaven]# cd 1.0-SNAPSHOT/
[root@xf_test_77 1.0-SNAPSHOT]# ll
total 16
-rw-r--r-- 1 root root 2297 Dec 17 11:59 hellomaven-1.0-SNAPSHOT.jar
-rw-r--r-- 1 root root  756 Dec 17 11:53 hellomaven-1.0-SNAPSHOT.pom
-rw-r--r-- 1 root root  704 Dec 17 11:59 maven-metadata-local.xml
-rw-r--r-- 1 root root  193 Dec 17 11:59 _remote.repositories
```

至此，构建一个简单的 maven 项目完成。

3.4.4. 项目模型 POM 文件介绍

Maven 创建后自动生成了一个 POM 文件，放在工程根目录下，pom 文件内容为：

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.maven.demo</groupId>
  <artifactId>hellomaven</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>hellomaven</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

该 pom 是最基础的 pom 文件，其他的 pom 都是在此基础上不断添加的结果集。你可以通过命令 mvn help:effective-pom 查看到一个大得多的 pom 文件，里边包括 maven 的默认设置信息。这其中就包括之前说到的全局的 settings 设置以及用户级别的 settings 配置文件里边定义的配置参数。

3.4.5. 打包部署到私服仓库

打包部署到私服仓库上，打包部署到本地命令在上一节中已经定义了，mvn install，部署到私服仓库命令为 mvn deploy。要完成部署到私服仓库，要配置下边两项内容：

在 maven demo 的 pom 文件中增加上 distribute management 相关配置；

在 maven 配置 settings.xml 文件中声明上 servers 的账号和密码配置。

接着，通过简单命令操作流程熟悉下 maven 的操作过程。

mvn deploy 经过了 compile、test、package、install 过程，即为编译、测试、打包、安装到了部署终点。

mvn compile 编译处理，使用 maven 的 compiler 编译插件。

mvn test 跑单元测试处理，使用 maven 的 surefire 测试插件。

mvn package 本地打包，此操作不会将 jar 包安装到本地仓库。

mvn install 本地构建安装处理。

此处重点演示将 maven demo 项目部署到私服仓库，私服此处不介绍，请见 nexus 私服使用手册，或者查阅官方相关 wiki 文档即可。为了支持部署到远程私服仓库，maven demo pom 文件里边添加定义内容为：

```
33      <distributionManagement>
34          <snapshotRepository>
35              <id>snapshots</id>
36              <name>Nexus Snapshot Repository</name>
37              <url>http://192.168.1.222:8081/nexus/content/repositories/snapshots</url>
38          </snapshotRepository>
39          <repository>
40              <id>releases</id>
41              <name>Nexus Release Repository</name>
42              <url>http://192.168.1.222:8081/nexus/content/repositories/releases</url>
43          </repository>
44      </distributionManagement>
```

在 maven 的全局配置文件中添加上 server 对应账号密码设置为：

配置 releases 和 snapshots 仓库的账号和密码，私服默认密码为 admin/admin123

```
    <servers>
        <server>
            <id>snapshots</id>
            <username>admin</username>
            <password>admin123</password>
            <filePermissions>664</filePermissions>
            <directoryPermissions>775</directoryPermissions>
        </server>
        <server>
            <id>releases</id>
            <username>admin</username>
            <password>admin123</password>
            <filePermissions>664</filePermissions>
            <directoryPermissions>775</directoryPermissions>
        </server>
    </servers>

    <mirrors>
        <mirror>
            <id>nexus</id>
            <mirrorOf>*</mirrorOf>
            <name>Localhost Nexus Server</name>
            <url>http://192.168.1.222:8081/nexus/content/groups/public</url>
        </mirror>
    </mirrors>
```

此处设置了 mirror 镜像配置，将 jar 包放在私服 nexus 服务上维护处理。

第一次从 central 中央仓库下载回来的 jar 包会放在 nexus 上，后续再次依赖的时候无需到 central 中央仓库下载，减轻对中央仓库的负担。

3.4.6. 生成项目站点文档

生成项目站点文档名称为 mvn site 即可，生成了 HTML 页面相关，放在 target/site 目录。

3.4.7. 导入 Eclipse IDE 处理

Maven 定义了 pom 项目模型文件，Eclipse 之类的 IDE 也有自身的描述项目的文件，maven 提供了 Eclipse 插件，通过命令 mvn eclipse:eclipse 可以创建出 Eclipse 对应的项目描述文件，执行结果为：

```
total 28
drwxr-xr-x 4 root root 4096 Dec 17 12:42 .
drwxr-xr-x 4 root root 4096 Dec 17 11:53 ..
-rw-r--r-- 1 root root  454 Dec 17 12:42 .classpath
-rw-r--r-- 1 root root  756 Dec 17 11:53 pom.xml
-rw-r--r-- 1 root root  446 Dec 17 12:42 .project
drwxr-xr-x 4 root root 4096 Dec 17 11:53 src
drwxr-xr-x 6 root root 4096 Dec 17 12:38 target
```

Eclipse 描述项目模型的文件为.classpath 和.project 文件，有兴趣的可以打开.project 文件观察下其描述内容。

```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
  <name>hellomaven</name>
  <comment>NO_M2ECLIPSE_SUPPORT: Project files created with the maven-eclipse-plugin are not supported in M2Eclipse.</comment>
  <projects/>
  <buildSpec>
    <buildCommand>
      <name>org.eclipse.jdt.core.javabuilder</name>
    </buildCommand>
  </buildSpec>
  <natures>
    <nature>org.eclipse.jdt.core.javanature</nature>
  </natures>
</projectDescription>
```

jar 包之间的依赖放在.classpath 文件里边

```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
  <classpathentry kind="src" path="src/test/java" output="target/test-classes" including="**/*.java"/>
  <classpathentry kind="src" path="src/main/java" including="**/*.java"/>
  <classpathentry kind="output" path="target/classes"/>
  <classpathentry kind="con" path="org.eclipse.jdt.launching.JRE_CONTAINER"/>
  <classpathentry kind="var" path="M2_REPO/junit/junit/3.8.1/junit-3.8.1.jar"/>
</classpath>
```

此处可以看到依赖的 jar 包 junit-3.8.1.jar 包，另外就是 src path 以及 output path 路径。

通过 mvn eclipse:clean 可以清理掉这两个文件，然后再执行 mvn eclipse:eclipse 可以再次生成 Eclipse 对应的项目模型描述文件。

处理完后打开 Eclipse IDE 就可以 import 导入 maven 工程了。

3.4.8. 扩展阅读&内部细节

通过前边章节我们已经跑完了一个 maven 的基本过程，该小节将介绍更多 maven 内部细节东西，在介绍之前我们回顾下前边的过程为：

我们通过 mvn archetype:create 创建了一个项目，之后用生命周期阶段 phase 作为参数来运行 maven，这个阶段会提示 maven 运行一系列 maven 插件的目标。最后，我们把 maven 构件 artifact 安装 install 到我们的本地仓库 repository。

3.4.8.1. Maven 插件和目标（Plugins and Goals）

一个 maven 插件是一个或者多个目标的集合，maven 插件的例子有一些简单但核心的插件，此处我们介绍 Jar 插件、Compiler 插件以及 Surefire 插件。

Jar 插件包含一组创建 jar 文件的目标；

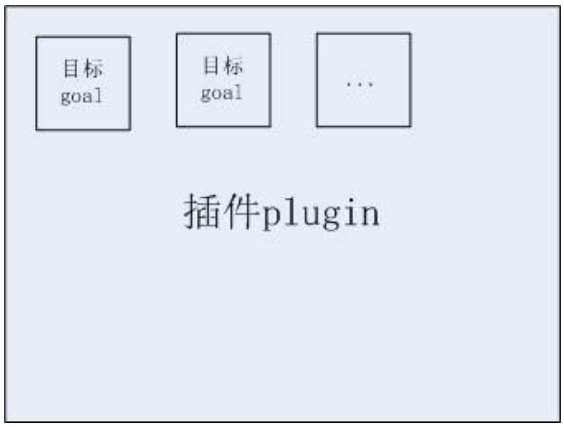
Compiler 插件包含一组编译源代码和测试代码的目标；

Surefire 插件包含一组运行单元测试和生成测试报告的目标。

回顾之前我们用的 mvn archetype:create 命令，archetype 是一个插件标识，create 是目标标识，当 maven 执行一个插件目标，它向标准输出打印出插件标识和目标标识。

另外一种是 mvn install 是一个生命周期阶段。

因此总结理解为：插件 plugin 和目标 goal 关系为：



插件=目标 1+目标 2+目标 n（插件为一组目标的集合）

简单理解为一个目标就是一个明确的任务，它可以作为单独的目标运行，或者作为一个大的构建的一部分和其它目标一起运行。一个目标是 maven 中一个工作单元（unit of work），比如 compiler 插件中的 compile 目标，用于编译项目的源代码和测试代码；surefire 插件的 test 目标，用于执行单元测试过程。速记为，插件标识符：目标标识符。

3.4.8.2. Maven 生命周期（Lifecycle）

此处先回顾 maven 命令的 mvn package，命令行并没有指定一个插件：目标，而是指定了一个 maven 生命周期阶段。一个阶段是在被 maven 称为构建生命周期中的一个步骤。

Maven 的生命周期是包含一个项目构建中的一系列有序的阶段。Maven 可以支持许多不同的生命周期，但是最常用的生命周期是默认的 maven 生命周期，这个生命周期中一开始的一个阶段是验证项目的基本完整性，最后一个阶段是把每一个项目发布成产品。

maven 通常约定的做法是将目标绑定在生命周期的阶段里边，比如 mvn package 是将 jar:jar 目标绑定到 package 打包阶段。

Maven 通常约定基本过程阶段为：验证 validation、测试 testing、发布 deployment。

3.4.8.3. Maven 坐标（Coordinates）

Maven 坐标定义了一组标识，他们可以用来唯一标识一个项目、一个依赖或者 maven pom 里边的一个插件。此处可以简单理解为 x/y/z 三维空间里边的三个坐标轴为唯一确定空间里

边的一个点。

groupId 为：团体、公司、小组、组织或者其他名称。建议用公司名称，比如 com.sonatype.xx

artifactId 为：在 groupId 下的表示一个单独的唯一标识符。

version 为：一个项目的特定版本。分为 release 版本和 snapshot 版本。

packaging 为：项目的类型，默认是 jar，也可以是其它，比如 war 或者 pom。

3.4.8.4. Maven 仓库（Repositories）

Maven 仓库理解为存放 jar 包的集合区，maven 提供了一个 central 中央仓库，中央仓库放置了所有的 jar 包集合，你可以在局域网内部搭建一个私服 nexus 仓库，将团队或者公司内部的 jar 包放在局域网私服仓库上，因为这部分 jar 包是私有的，你不应该上传到 central 中央仓库或者第三方开源的仓库中去。

Maven 约定寻找 jar 包的过程为，先在本地仓库寻找，然后依次是局域网内部的私服仓库（假定有搭建私服的话），在私服 nexus 上配置的 public 仓库集合（里边可以设定 central 中央仓库以及第三方仓库，比如 Google code、开源中国、Apache snapshot 仓库之类的）

当寻找完后还是无法定位下载到 jar 包则提示错误信息，依赖寻找到的 jar 下载完后会在私服以及本地仓库上做存放备份处理，下次就不需要再去外网拉取 jar 了。

3.4.8.5. Maven 依赖管理（Dependency Management）

Maven 的依赖管理通过在 POM 文件里边定义 dependency 参数配置即可，maven 依赖之间存在依赖传递性特点。Maven 仓库不仅存放二进制文件，也存储了这些构建的元数据 metadata，才使得传递性依赖成为可能。

Maven 也提供了不同的依赖范围 dependency scope，比如常见的单元测试 junit 设定的 scope 为 test，说明它在 compiler 插件运行 compile 目标的时候是不可用的，只有在 compiler 插件运行 testCompile 和 surefire 插件运行 test 目标的时候才会被加入到 classpath 中去。

另外，可以配置 maven 使用 provided 范围，让它排除 war 文件中特定的依赖，provided 范围告诉 maven 一个依赖在编译的时候需要，但是它不应该被捆绑在构建的输出中。当你开发 web 应用的时候 provided 范围变得有用，你需要通过 Servlet API 来编译你的代码，但是你不希望 Servlet API 的 jar 文件包含在你 web 应用的 WEB-INF/lib 目录中。

3.4.8.6.站点生成和报告（Site Generation and Reporting）

另外一个 maven 的重要特征是它能够生成文档以及报告。

命令为 mvn site，这将会运行 site 生命周期阶段，它不像默认生命周期那样，管理代码的生成，操作资源，编译，打包等等。Site 生命周期只关心处理在 src/site 目录下的 site 内容，还要生成报告。执行该命令你将会在 target/site 目录下看到一个项目 web 站点，打开 target/site/index.html 可以看到项目站点的基本外貌。

Maven 提供了许多有用的报告，分别为：

Clover 报告检查单元测试的覆盖率；

JXR 报告生成 HTML 源代码相互间的引用关系，这在代码审核的时候比较有用；

PMD 报告通过 PMD 规则分析项目的源代码；

JDepend 报告分析源代码中各个包之间的依赖。

我们通过在 pom.xml 中配置哪些报告被包含在构建中，站点报告就可以被定制了。

4. Nexus 私服搭建

4.1. Nexus 基本介绍

Nexus 私服仓库平台，在局域网内部搭建的 nexus 服务平台，方便团队成员分享依赖 jar 包，另外在局域网内部拉取 jar 包速度也比较快，不需要每次都到 maven 中央仓库拉取，减少对中央仓库的负担。

4.2. Nexus 环境安装

Nexus 私服从官方网站下载对应的*.tar.gz 包即可：

官方地址为：http://www.sonatype.org/nexus/

*.tar.gz 下载地址为：http://www.sonatype.org/nexus/downloads/

Linux 系统和 win7 系统下载的 tar.gz 包是一样的包，启动的脚本名称不同而已。

4.2.1. 在 Linux 平台搭建 nexus 私服

将 tar.gz 压缩包文件上传到/usr/local/目录下，然后建立一个软链接 nexus 到 2.10.0-02 版本的 nexus，启动 nexus，到 nexus 的 bin/目录下执行相应的脚本即可。

```
[root@xf_test_77 ~]# cd /usr/local/
[root@xf_test_77 local]# ll|grep nexus
lrwxrwxrwx   1 root    root          16 Dec  8 20:40 nexus -> nexus-2.10.0-02/
drwxr-xr-x   9 root    root        4096 Dec  8 20:45 nexus-2.10.0-02
-rw-r--r--   1 root    root    60815360 Dec  8 20:30 nexus-2.10.0-02.tar.gz
```

为了后续方便升级，参考 maven 此处建了一个软连接处理。

因此 nexus 私服安装目录为：/usr/local/nexus/，观察到 nexus 私服目录结构为：

```
[root@xf_test_77 local]# cd nexus
[root@xf_test_77 nexus]# ll
total 44
drwxr-xr-x  3 root root  4096 Oct 16 08:53 bin
drwxr-xr-x  2 root root  4096 Dec  9 10:46 conf
drwxr-xr-x  2 root root  4096 Oct 16 08:53 lib
-rw-r--r--  1 root root 10944 Sep 30 06:12 LICENSE.txt
drwxr-xr-x  2 root root  4096 Dec  8 20:43 logs
drwxr-xr-x  4 root root  4096 Oct 16 08:53 nexus
-rw-r--r--  1 root root   763 Sep 30 04:18 NOTICE.txt
drwxr-xr-x 12 root root  4096 Dec  8 22:16 sonatype-work
drwxr-xr-x  6 root root  4096 Dec 16 21:08 tmp
```

bin/目录下为脚本文件，conf/下为 nexus 配置文件

Logs/目录为日志文件，nexus/目录下为 web 站点需要的资源文件

```
[root@xf_test_77 nexus]# cd nexus/
[root@xf_test_77 nexus]# ll
total 32
-rw-r--r-- 1 root root  1150 Sep 30 04:18 favicon.ico
-rw-r--r-- 1 root root  1160 Sep 30 04:18 favicon.png
drwxr-xr-x 2 root root  4096 Oct 16 08:53 images
-rw-r--r-- 1 root root 11946 Sep 30 06:12 LICENSE.html
-rw-r--r-- 1 root root    86 Sep 30 04:18 robots.txt
drwxr-xr-x 6 root root  4096 Oct 16 08:53 WEB-INF
```

此处将 work 工作 space 目录放在/usr/local/nexus/sonatype-work 目录下。

到 conf/目录下查看 nexus 配置文件为：

```
# Sonatype Nexus
# ===============
# This is the most basic configuration of Nexus.

# Jetty section
application-port=8081
application-host=0.0.0.0
nexus-webapp=${bundleBasedir}/nexus
#nexus-webapp-context-path=/nexus
#使用nginx做反向代理此处设置为/
nexus-webapp-context-path=/

# Nexus section
nexus-work=/usr/local/nexus/sonatype-work
runtime=${bundleBasedir}/nexus/WEB-INF
```

Web 启动默认端口为 8081，设定了访问的 path 以及 nexus-work 目录。

Nexus 使用内置的 jetty 容器启动，默认启动的端口为 8081，启动成功后访问 http://localhost:8081/nexus/即可看到 nexus web 界面了。假定要变更端口，则修改此配置文件即可，然后重启 nexus 私服即可生效。

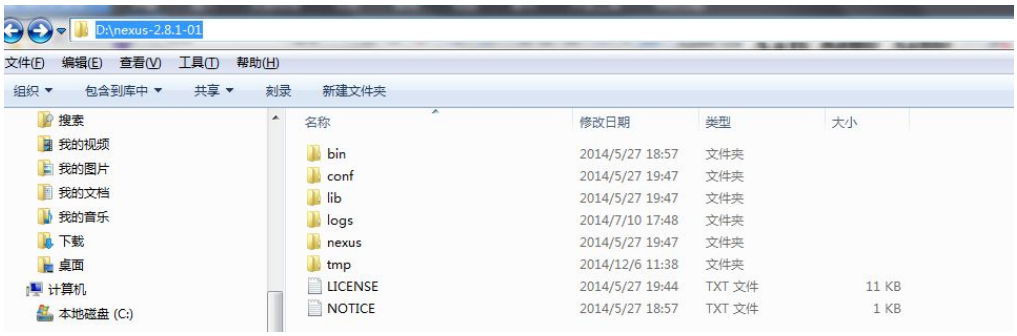
4.2.2. 在 windows 平台下搭建 nexus 私服

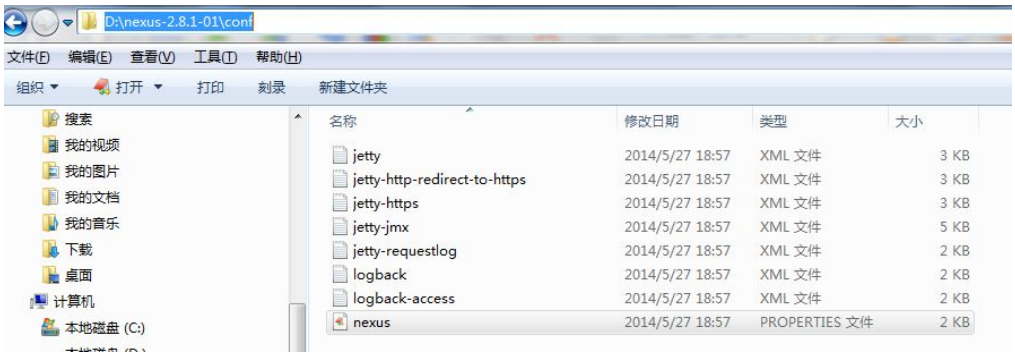
本机 win7 下载版本为：

解压放在 d 盘目录下，Linux 下同理，nexus 里边内置了一个 jetty 容器，因此可以直接启动无需再找其他的容器，不过你可以使用 tomcat 容器来跑 nexus 服务。

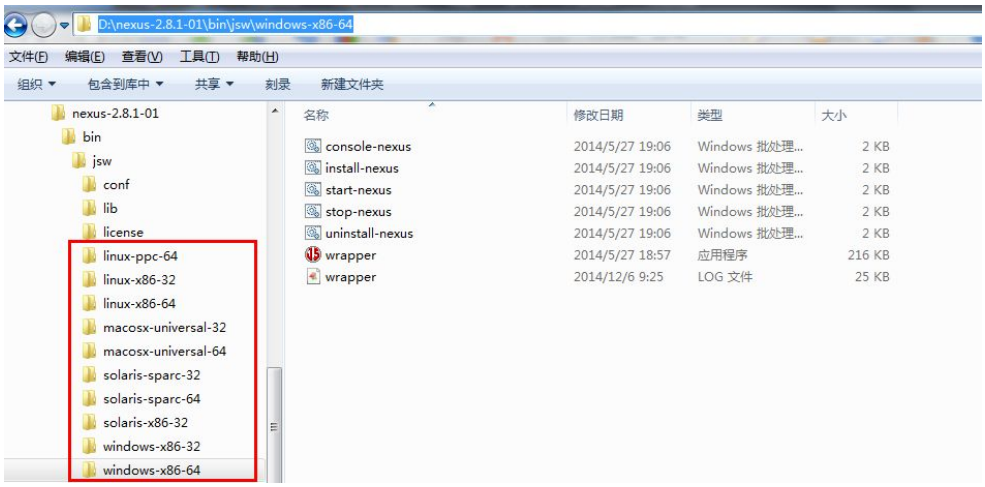
Nexus 配置文件放在 conf/目录下，通过配置文件名称可以了解大概的使用用途。主配置文件为：nexus.properties 文件，配置文件内容为：

```
# Sonatype Nexus
# ===============
# This is the most basic configuration of Nexus.

# Jetty section
application-port=8081
application-host=0.0.0.0
nexus-webapp=${bundleBasedir}/nexus
nexus-webapp-context-path=/nexus

# Nexus section
nexus-work=${bundleBasedir}/../sonatype-work/nexus
runtime=${bundleBasedir}/nexus/WEB-INF
```





这边设定 nexus jetty 启动的 http 端口以及访问的 uri 路径以及制定启动后 work 工作目录，本机放在../sonatype-work/nexus 目录下，同理 Linux 环境到 bin/目录下启动 nexus 服务即可。



在 bin/jsw/目录下放置了各个操作系统对应的安装、启动、卸载脚本。

本机 win7 64 位，执行 install 安装然后启动 nexus 即可，启动完毕后访问：

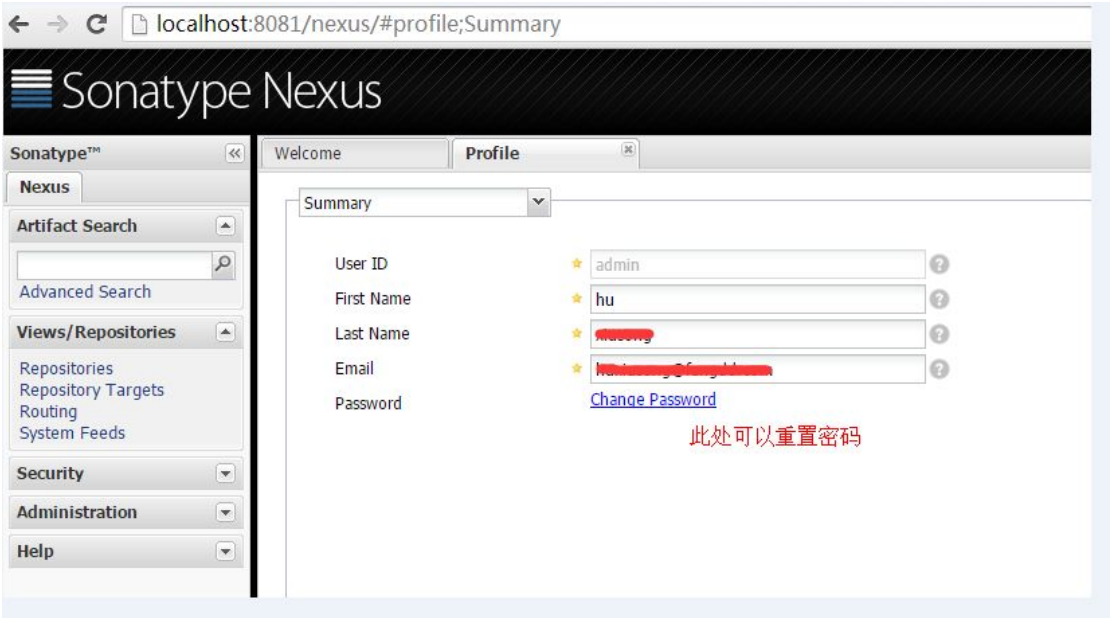
http://localhost:8081/nexus/

此处 tips 提示可以升级到最新版本的 nexus，2.11.x 版本，点击 download 升级即可。至此，nexus 在 win7 上搭建完毕，后边将会讲解基本的设置过程。

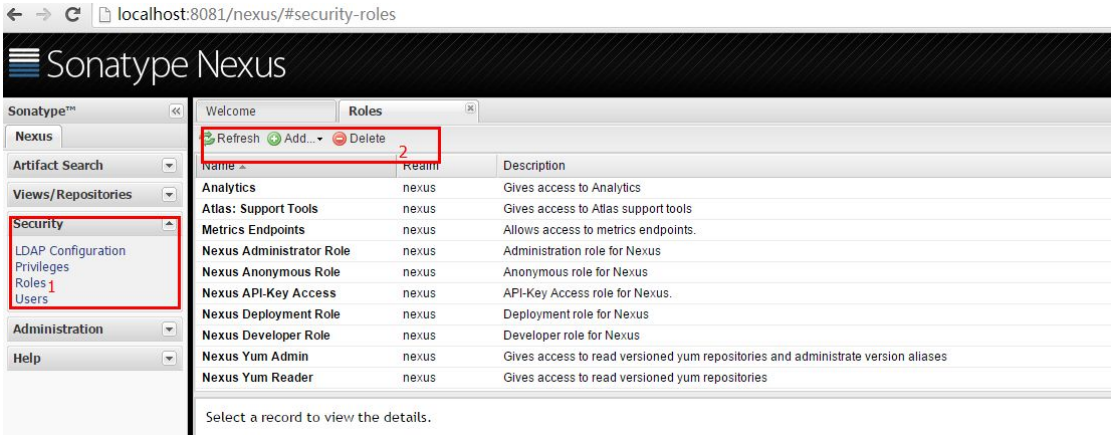
4.3. Nexus 私服设置

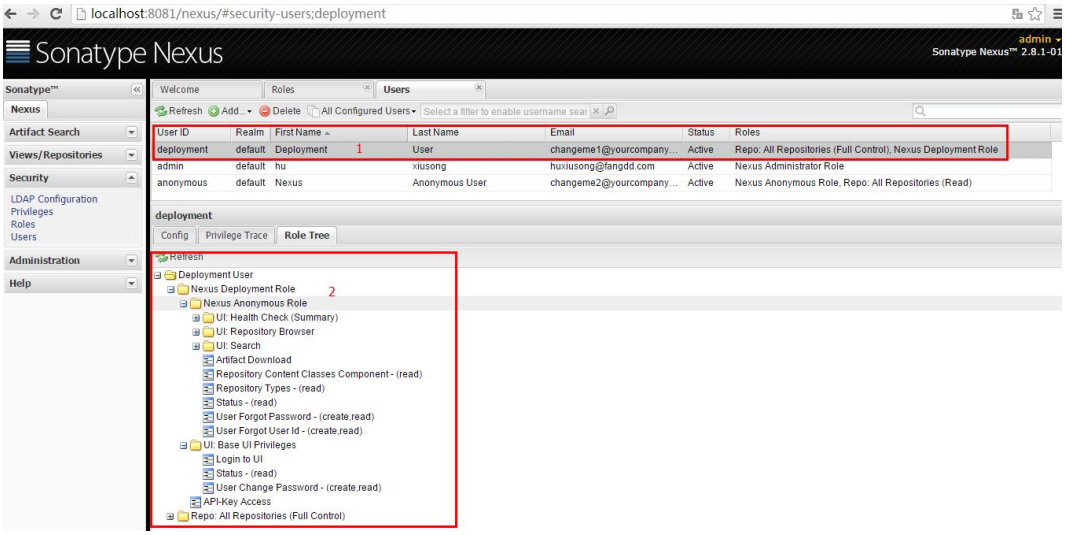
4.3.1. Nexus 账号设置

私服 nexus 默认账号密码为 admin/admin123，密码可以重置。



Nexus 提供了权限相关管理，左侧导航栏处的 Security 点开即可，基于 role 角色的权限分配逻辑。增加 role 以及删除 role 角色等操作，点击 Users 则为添加用户。



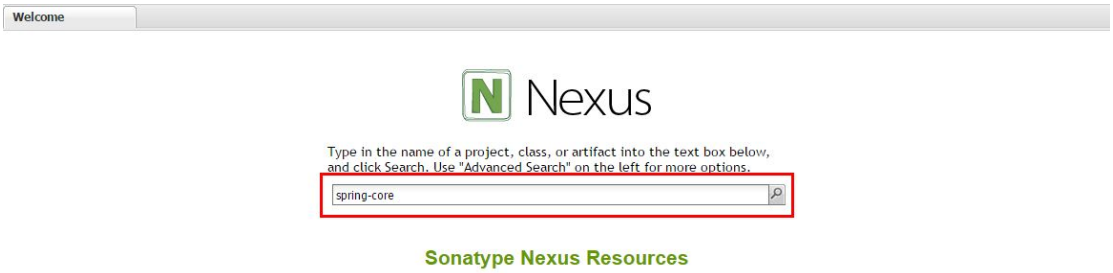


Nexus 默认为我们创建了三个用户，deployment 用户，admin 管理员用户以及 anonymous 匿名的用户，这边我们重置掉 admin 的默认密码，收回 admin 管理权限，分配 deployment 权限给组内开发同事即可。Nexus 默认分配的 deployment 账号密码为 deployment123。

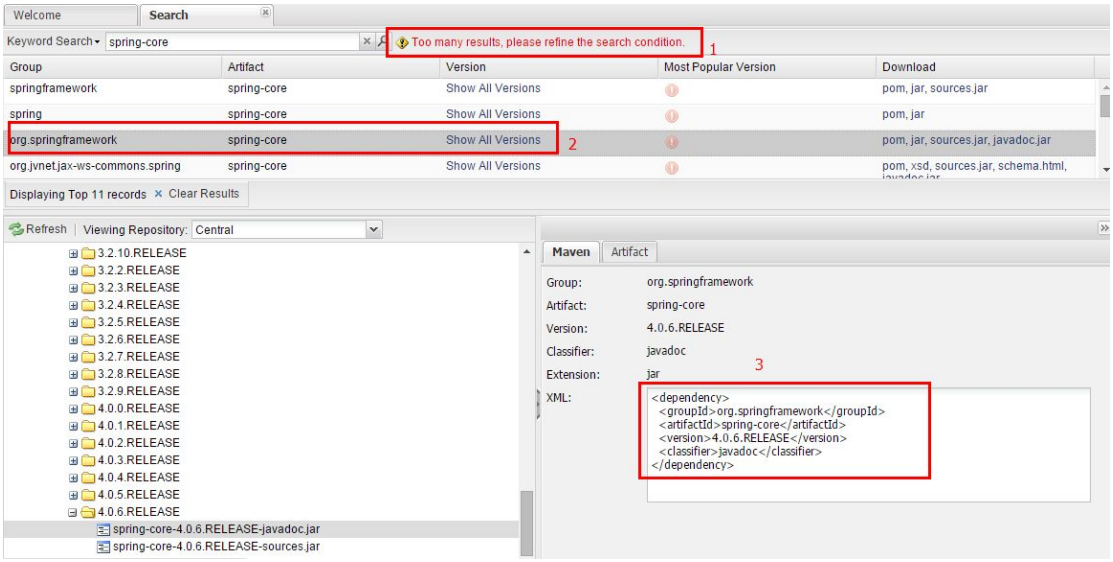
4.3.2. Nexus 搜索使用

此处将会使用未登录的用户状态，使用下搜索功能，即为类似匿名用户状态。

此处模拟搜索 spring-core jar 包



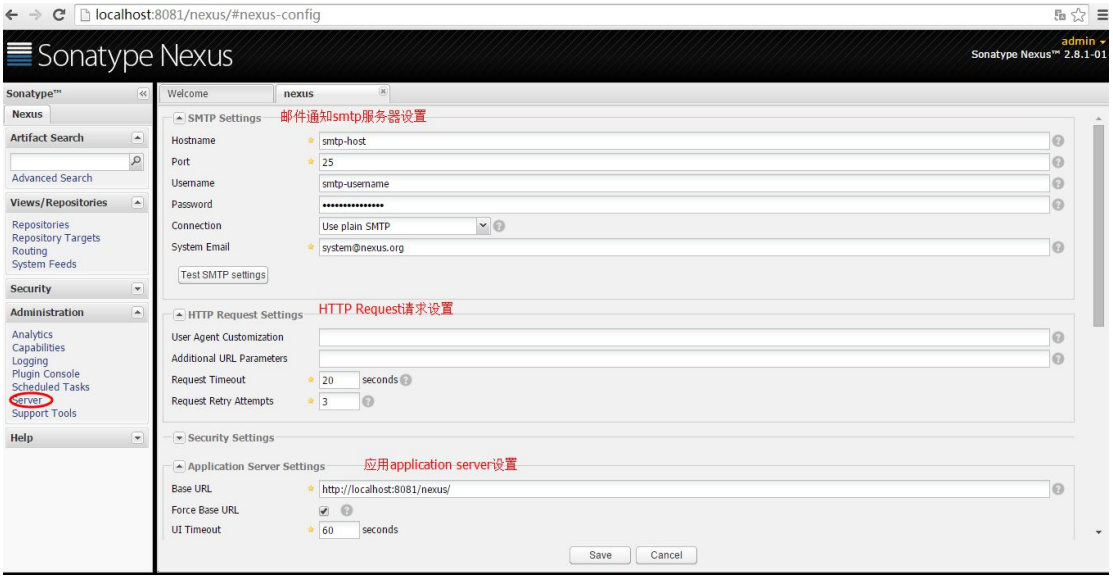
访问 http://localhost:8081/nexus/ 主界面输入 spring-core 搜索关键词回车即可



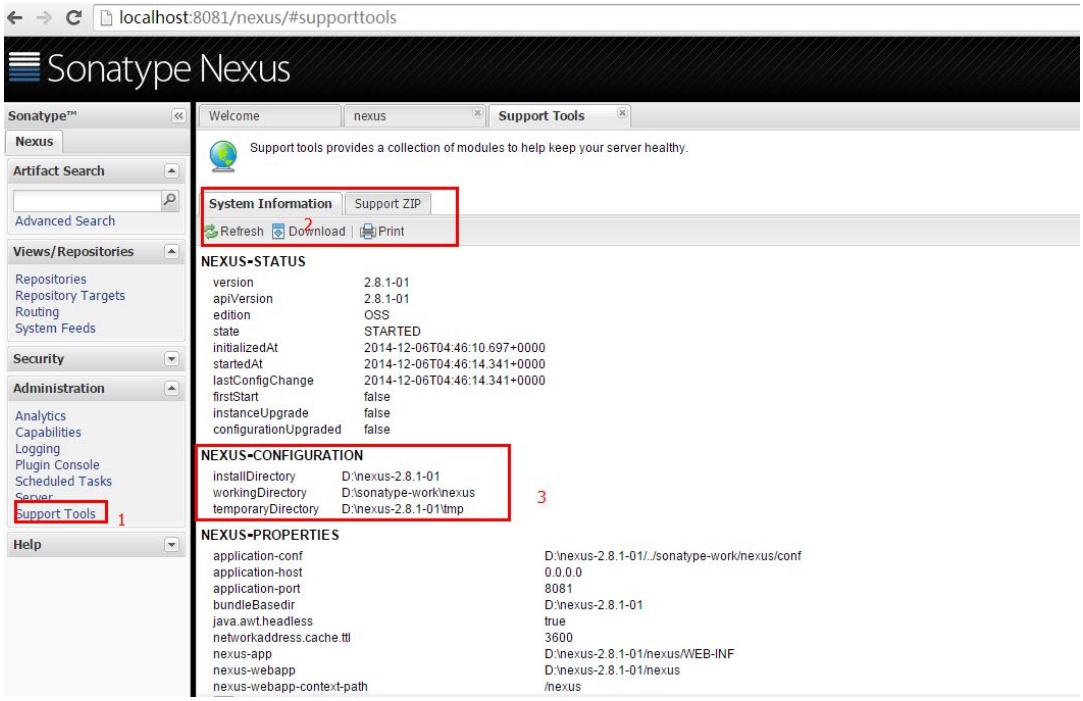
搜索结果提示结果集比较多，可以定制下过滤条件，我们计划是需要找的 spring 框架的 spring-core jar 包，罗列了全部版本的 spring-core，我们选择最新版本的，3 处即为对应的 maven dependency 依赖参数配置。复制到需要依赖 spring-core 的 maven 工程的 pom.xml 文件中即可。

4.3.3. Nexus 基本配置

点开左侧导航栏处的 Administrator，server 项目，里边是 nexus 的基本配置信息。



包括邮箱通知的 smtp 服务器设置，http request 请求设置以及应用服务器设置。

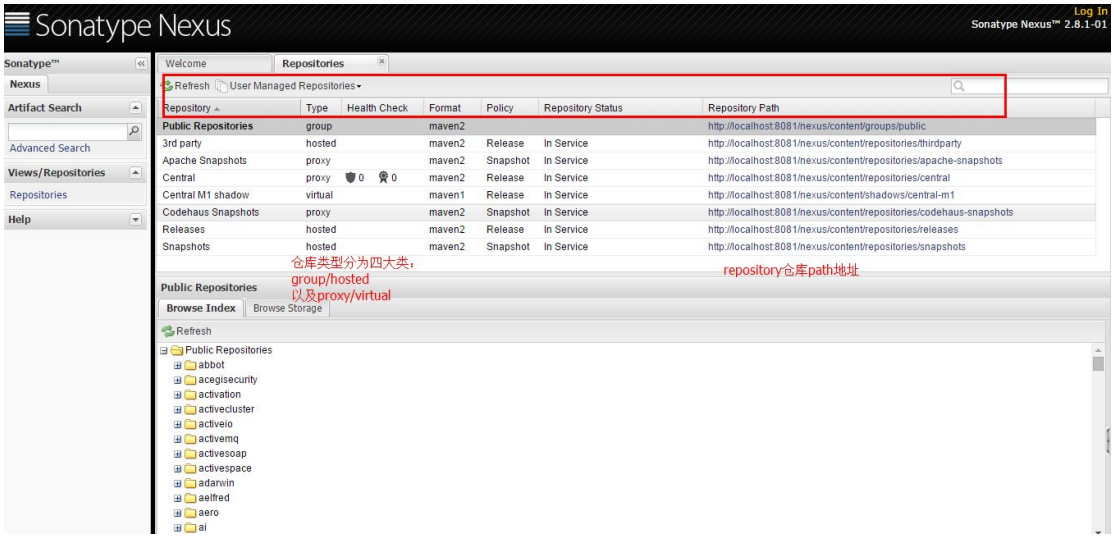


这些可以查看到 nexus 相关的 system 系统 info 信息，这边提供了 download 下载接口。

4.3.4. Nexus 仓库设置

Nexus 私服仓库类型分为四种：

Group 仓库组、Hosted 宿主、Proxy 代理以及 Virtual 虚拟四大类。



Public Repositories 仓库组

Central 用来代理 maven 中央仓库中发布版本构件的仓库

Central M1 shadow 用于提供中央仓库中 M1 格式的发布版本的构件镜像仓库

Codehaus Snapshots 用来代理 CodehausMaven 仓库的快照版本构件的仓库

3rd party 无法从公共仓库获得的第三方发布版本的构件仓库

Apache Snapshots 用了代理 Apache maven 仓库快照版本的构件仓库

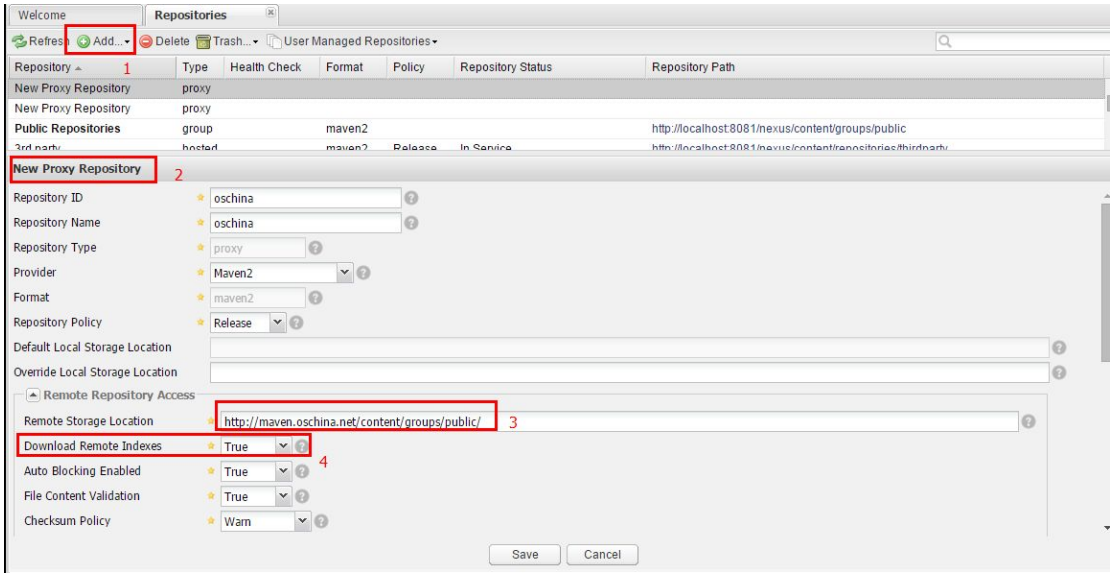
Releases 用来部署管理内部的发布版本构件的宿主类型仓库

Snashots 用来部署管理内部的快照版本构件的宿主类型仓库

此处，可以添加更多的私服仓库，比如 Google Code 以及 OSChina 之类的仓库。

举例添加 OSChina 过程为：

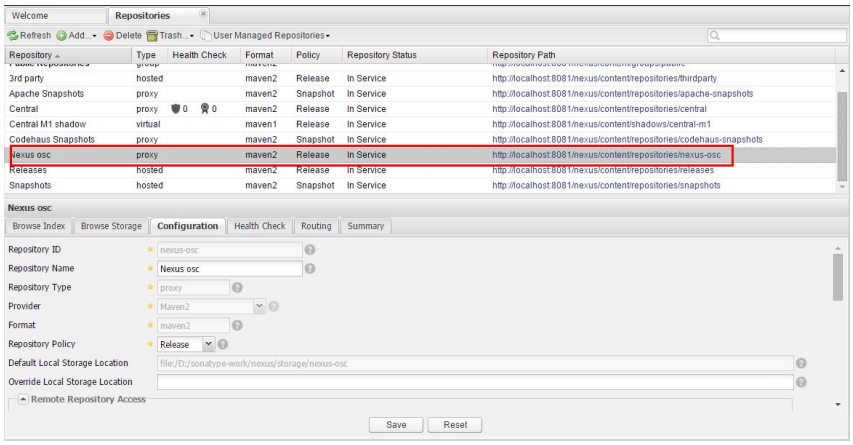
http://maven.oschina.net/content/groups/public/



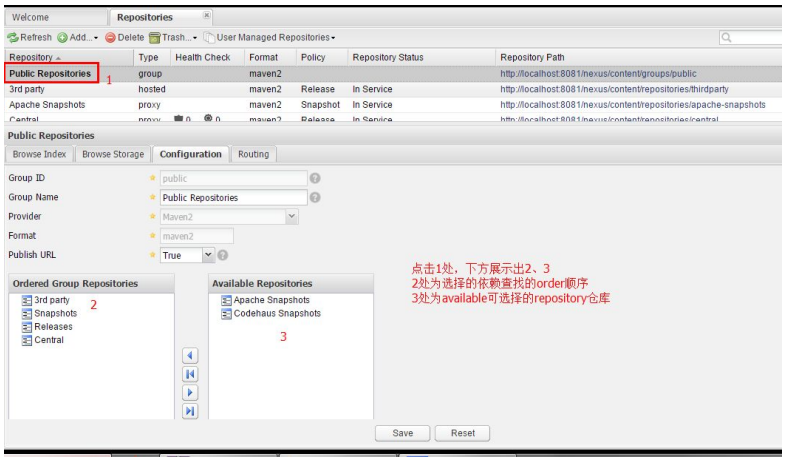
在 1 处点击 add 添加按钮，然后在 2 处填写相关 proxy repository 仓库信息

重点是在 3 处填上对应的 oschina repository 仓库地址，以及在 4 中开启 true 开关。

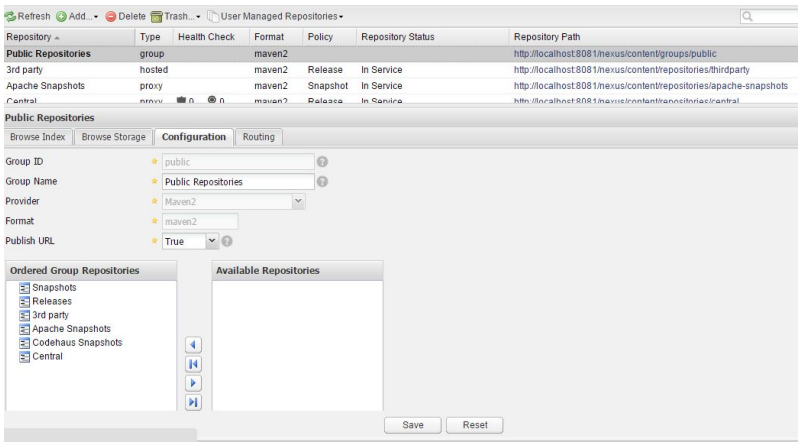
这边新建的是为 proxy 代理仓库 repository，同步更新的是 index 索引。



因此，此处约定局域网内部开发 dev 过程中若为 snapshot 快照 deploy 到 snapshot hosted 仓库，release 正式版本 deploy 到 release hosted 仓库中，统一规范不放在其它仓库里边。接着，我们将 public 公共仓库依赖顺序设置下为：



此处顺序调整为：

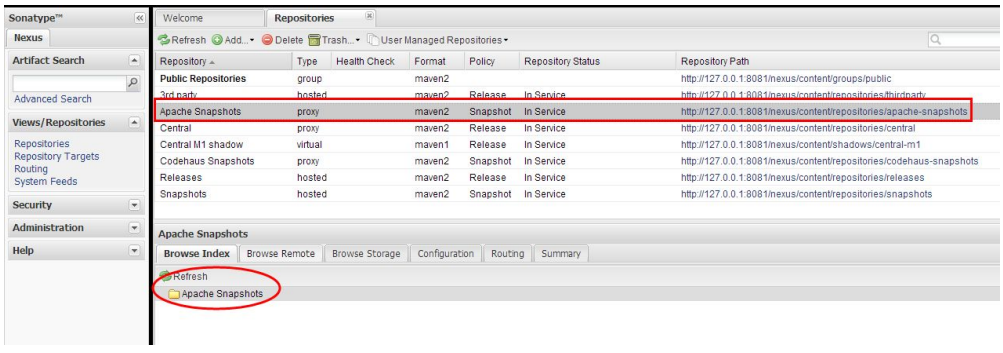


最后边才去查询 maven repository 中央仓库，其实中央仓库的速度也还可以。

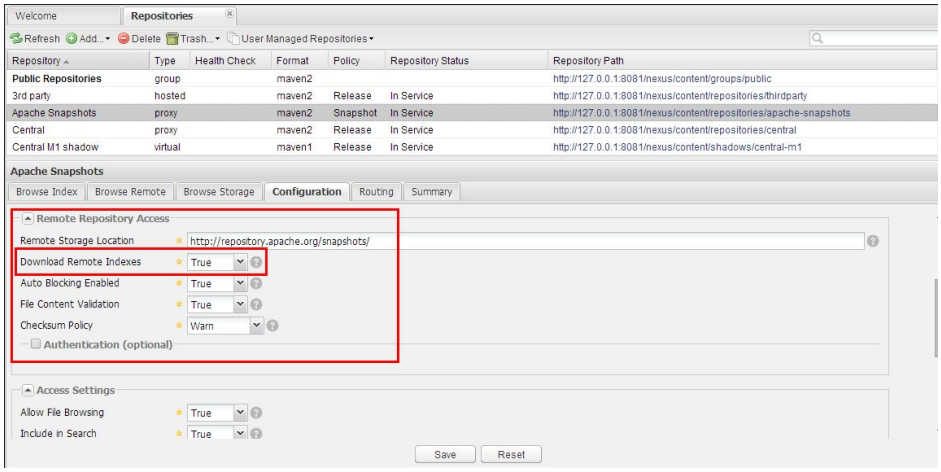
4.3.5. Nexus 索引设置

选择左侧 repository 仓库，点击 Apache Snapshots 仓库

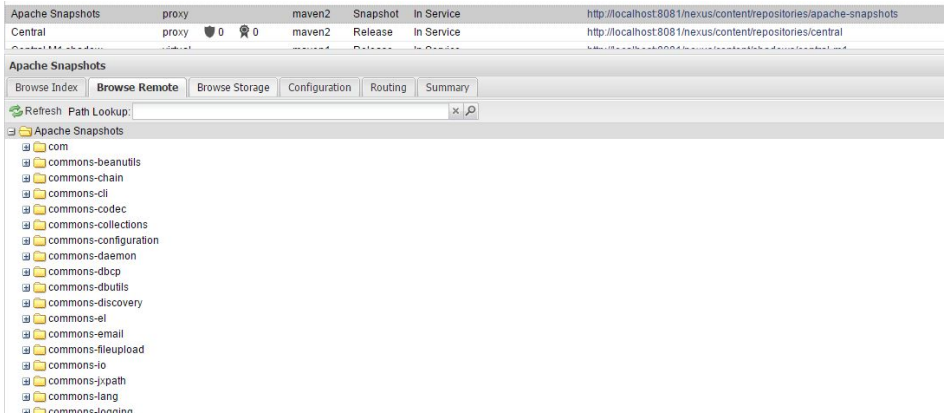
查看下方 Browse Index，点击发现并没有关联到 Apache Snapshots 仓库索引



选择 Configuration 设置，Download Remote Indexs 默认为 false，设置为 true 后点击保存。



然后，选中 apache snapshots 右键进行 repair index，从下载索引到 nexus 私服，其它仓库做相同的操作即可，处理完毕后，观察到



Browse remote 观察到远程 remote 仓库 jar 包集合

4.3.6. Maven 镜像配置

配置 maven conf/目录下的 setttings.xml 配置文件，添加上本机镜像地址为：

```
    <mirrors>
        <mirror>
            <id>nexus</id>
            <mirrorOf>*</mirrorOf>
            <name>Localhost Nexus Server</name>
            <url>http://localhost:8081/nexus/content/groups/public</url>
        </mirror>
    </mirrors>

    <profiles>
        <profile>
            <id>nexus</id>
            <repositories>
                <repository>
                    <id>central</id>
                    <url>http://central</url>
                    <releases><enabled>true</enabled></releases>
                    <snapshots><enabled>true</enabled></snapshots>
                </repository>
            </repositories>
            <pluginRepositories>
                <pluginRepository>
                    <id>central</id>
                    <url>http://central</url>
                    <releases><enabled>true</enabled></releases>
                    <snapshots><enabled>true</enabled></snapshots>
                </pluginRepository>
            </pluginRepositories>
        </profile>
    </profiles>
```

以上参考 nexus 官方 wiki 配置的，为了后续 deploy 部署此处添加上 server 配置。

```xml
<?xml version="1.0" encoding="UTF-8"?>

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
              http://maven.apache.org/xsd/settings-1.0.0.xsd">

    <localRepository>D:\\maven_repository</localRepository>
    <interactiveMode/>
    <usePluginRegistry/>
    <offline/>
    <pluginGroups/>
    <proxies />

    <servers>
        <server>
            <id>snapshots</id>
            <username>deployment</username>
            <password>deployment123</password>
            <filePermissions>664</filePermissions>
            <directoryPermissions>775</directoryPermissions>
        </server>
        <server>
            <id>releases</id>
            <username>deployment</username>
            <password>deployment123</password>
            <filePermissions>664</filePermissions>
            <directoryPermissions>775</directoryPermissions>
        </server>
    </servers>
```

4.4. Nexus 使用例子过程

4.4.1. 创建工程部署到私服

在 maven 章节 hellomaven 工程基础上，进一步做演示处理。

```xml
    <!-- maven dependency依赖关系设置 -->
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <!-- maven deploy远程部署配置 -->
    <distributionManagement>
        <snapshotRepository><!-- 部署到snapshot快照版本仓库 -->
            <id>snapshots</id>
            <name>Nexus Snapshot Repository</name>
            <url>http://localhost:8081/nexus/content/repositories/snapshots</url>
        </snapshotRepository>
        <repository><!-- 部署到release版本仓库 -->
            <id>releases</id>
            <name>Nexus Release Repository</name>
            <url>http://localhost:8081/nexus/content/repositories/releases</url>
        </repository>
    </distributionManagement>
</project>
```

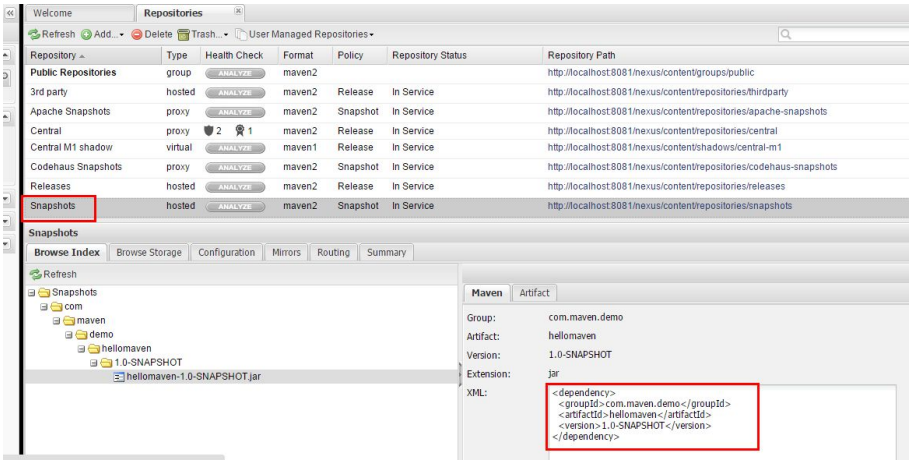
在 hellomaven 工程的 pom.xml 文件中添加上 maven deploy 远程部署配置，添加完毕后再次执行 mvn deploy 报错为：

```
[INFO] --- maven-deploy-plugin:2.7:deploy (default-deploy) @ hellomaven ---
Downloading: http://localhost:8081/nexus/content/repositories/snapshots/com/maven/demo/hellomaven/1.
0-SNAPSHOT/maven-metadata.xml
[WARNING] Could not transfer metadata com.maven.demo:hellomaven:1.0-SNAPSHOT/maven-metadata.xml from
/to snapshots (http://localhost:8081/nexus/content/repositories/snapshots): Access denied to: http:/
/localhost:8081/nexus/content/repositories/snapshots/com/maven/demo/hellomaven/1.0-SNAPSHOT/maven-me
tadata.xml , ReasonPhrase:Forbidden.
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 5.413s
[INFO] Finished at: Sat Dec 06 16:56:39 CST 2014
[INFO] Final Memory: 7M/152M
[INFO] ------------------------------------------------------------------------
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-deploy-plugin:2.7:deploy (default-depl
oy) on project hellomaven: Failed to retrieve remote metadata com.maven.demo:hellomaven:1.0-SNAPSHOT
/maven-metadata.xml: Could not transfer metadata com.maven.demo:hellomaven:1.0-SNAPSHOT/maven-metada
ta.xml from/to snapshots (http://localhost:8081/nexus/content/repositories/snapshots): Access denied
 to: http://localhost:8081/nexus/content/repositories/snapshots/com/maven/demo/hellomaven/1.0-SNAPSH
OT/maven-metadata.xml , ReasonPhrase:Forbidden. -> [Help 1]
[ERROR]
```

提示没有权限，检查下 maven 配置文件 settings.xml，添加上 server 配置参数为：

```
<servers>
    <!-- 添加账号信息，deploy过程要验证身份 -->
    <server>
        <id>releases</id>
        <username>deployment</username>
        <password>deployment123</password>
    </server>
    <server>
        <id>snapshot</id>
        <username>deployment</username>
        <password>deployment123</password>
    </server>
</servers>
```

设置完毕后，再次 deploy 部署 ok 了



执行 maven 部署命令：mvn deploy 之后就部署到 nexus 私服上，因为版本为 snapshot 因此部署到了私服 nexus 上的 snapshot 仓库上。对应配置在 pom.xml 里边的为：

```
<!-- maven deploy远程部署配置 -->
<distributionManagement>
    <snapshotRepository><!-- 部署到snapshot快照版本仓库 -->
        <id>snapshots</id>
        <name>Nexus Snapshot Repository</name>
        <url>http://localhost:8081/nexus/content/repositories/snapshots</url>
    </snapshotRepository>
    <repository><!-- 部署到release版本仓库 -->
        <id>releases</id>
        <name>Nexus Release Repository</name>
        <url>http://localhost:8081/nexus/content/repositories/releases</url>
    </repository>
</distributionManagement>
```

此处配置了 snapshot 和 release 版本的仓库 url 地址。

4.4.2. 创建新工程从私服上依赖

在本机新建一个 worldmaven maven 工程，里边添加上 dependency 上 hellomaven 工程部署在 nexus 私服上的 snapshot jar 包，worldmaven 工程 pom.xml 配置信息为：

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.maven.demo</groupId>
  <artifactId>worldmaven</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>worldmaven</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>com.maven.demo</groupId>
      <artifactId>hellomave</artifactId>
      <version>1.0-SNAPSHOT</version>
    </dependency>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>

```

此处增加上依赖hellomaven snapshot jar包

为了不受本地仓库干扰，将本地仓库hellomaven jar删除掉

保证worldmaven工程从nexus私服上拉取jar包

执行 maven 命令后 mvn eclipse:eclipse 增加依赖

```
[INFO]
[INFO] >>> maven-eclipse-plugin:2.9:eclipse (default-cli) @ worldmaven >>>
[INFO]
[INFO] <<< maven-eclipse-plugin:2.9:eclipse (default-cli) @ worldmaven <<<
[INFO]
[INFO] --- maven-eclipse-plugin:2.9:eclipse (default-cli) @ worldmaven ---
[INFO] Using Eclipse Workspace: null
[INFO] Adding default classpath container: org.eclipse.jdt.launching.JRE_CONTAINER
Downloading: http://localhost:8081/nexus/content/groups/public/com/maven/demo/hellomaven/1.0-SNAPSHO

Downloaded: http://localhost:8081/nexus/content/groups/public/com/maven/demo/hellomaven/1.0-SNAPSHOT
(768 B at 7.9 KB/sec)
Downloading: http://localhost:8081/nexus/content/groups/public/com/maven/demo/hellomaven/1.0-SNAPSHO
41207.092606-6.pom
Downloaded: http://localhost:8081/nexus/content/groups/public/com/maven/demo/hellomaven/1.0-SNAPSHOT
1207.092606-6.pom (2 KB at 16.9 KB/sec)
Downloading: http://localhost:8081/nexus/content/groups/public/com/maven/demo/hellomaven/1.0-SNAPSHO
41207.092606-6.jar
Downloaded: http://localhost:8081/nexus/content/groups/public/com/maven/demo/hellomaven/1.0-SNAPSHOT
1207.092606-6.jar (3 KB at 54.2 KB/sec)
[INFO] Not writing settings - defaults suffice
[INFO] Wrote Eclipse project for "worldmaven" to C:\Users\Administrator\worldmaven.
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 6.969s
[INFO] Finished at: Sun Dec 07 18:12:36 CST 2014
[INFO] Final Memory: 8M/152M
[INFO] ------------------------------------------------------------------------
C:\Users\Administrator\worldmaven>
```

提示 build 成功，从 nexus 私服上拉取了需要的 jar 包。

此处观察工程目录即可验证是否成功。

打开.classpath 文件观察为：

```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
  <classpathentry kind="src" path="src/test/java" output="target/test-classes" including="**/*.java"/>
  <classpathentry kind="src" path="src/main/java" including="**/*.java"/>
  <classpathentry kind="output" path="target/classes"/>
  <classpathentry kind="con" path="org.eclipse.jdt.launching.JRE_CONTAINER"/>
  <classpathentry kind="var" path="M2_REPO/com/maven/demo/hellomaven/1.0-SNAPSHOT/hellomaven-1.0-SNAPSHOT.jar"/>
  <classpathentry kind="var" path="M2_REPO/junit/junit/3.8.1/junit-3.8.1.jar"/>
</classpath>
```

此处表明 worldmaven 工程已经顺利从 nexus 私服下载到 jar 包，并且 install 到了本地仓中，至此，nexus 使用过程讲解完毕。