# Maven第二天

## 回顾

## 1.什么是maven?

它是一个软件开发的管理工具，主要管理的工作是：依赖管理，项目构建

2.使用maven的好处？

能够集中管理jar包，提供一键构建

3.maven的安装及配置

配置：系统变量新增MAVEN\_HOME,PATH路径配置

本地仓库：<localRepository>

运行:mvn -v

4.常用的maven命令

clean,compile,test,package,install,deploy

5.maven工程是具有一定的目录结构

src

main

java(程序主要代码)

resources（配置文件）

webapps

WEB-INF

web.xml

test

java（测试代码）

resources（测试的配置文件）

pom.xml（Maven配置文件）

6.eclipse工具下的maven工程开发

1.windows->Preferences->Maven->Installations选择add增加Maven版本

2.windows->Preferences->Maven->User Settings选择setttings.xml文件，点击Update Settings

3.Window-->show view-->other-->maven Repositories-->右键本地仓库(重建索引)

7.在pom.xml文件中如何引入坐标

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>2.5</version>

<scope>provided</scope>

</dependency>

解决jdk版本默认1.5的问题

<build>

<pluginManagement>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.2</version>

<configuration>

<source>1.7</source>

<target>1.7</target>

<encoding>UTF-8</encoding>

<showWarnings>true</showWarnings>

</configuration>

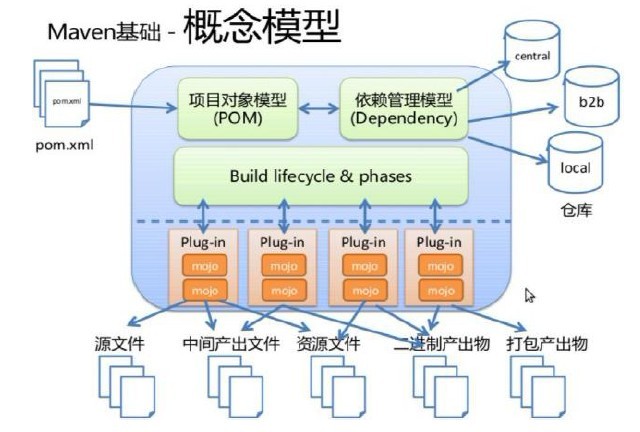
</plugin>

</plugins>

</pluginManagement>

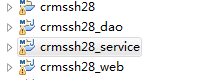
</build>

8.总结



## Maven工程的拆分与聚合（重点）

一个完整的早期开发好的crm项目，现在要使用maven工程对它进行拆分，这时候就可以将dao拆解出来形成表现独立的工程，同样service,action也都这样拆分



工程拆分之后，将来还要聚合（聚合就是将拆分的工程进一步组合在一起，又形成一个完整的项目）

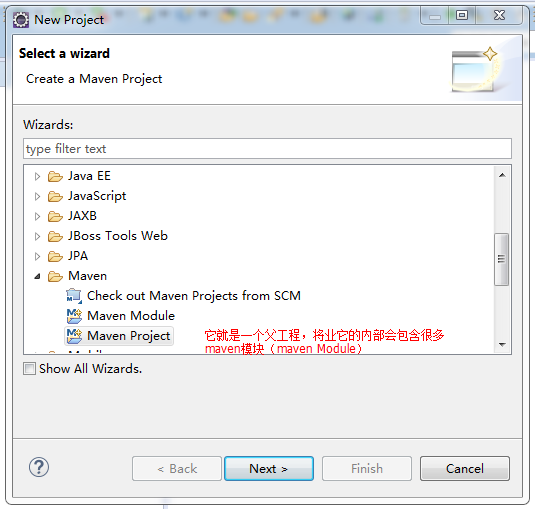
为了达到聚合的目标，所以今天会引入

父工程（maven project）

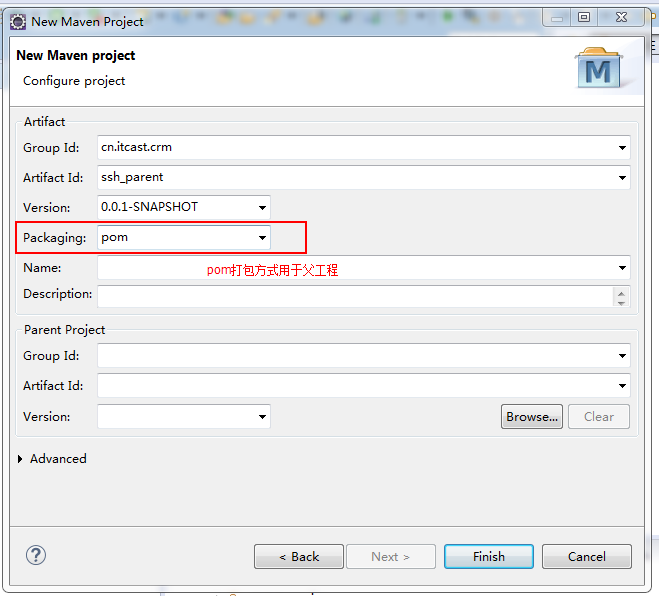
子模块(maven module) dao ,service, web

开发步骤：

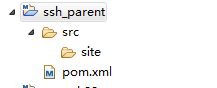
1. 创建一个maven父工程



点下一步：

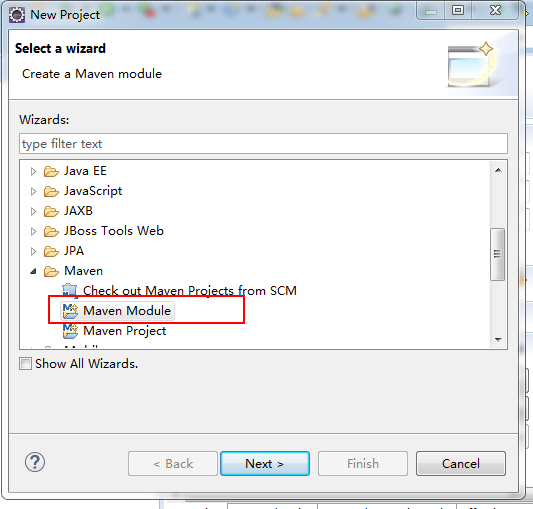


创建后的父工程如下：

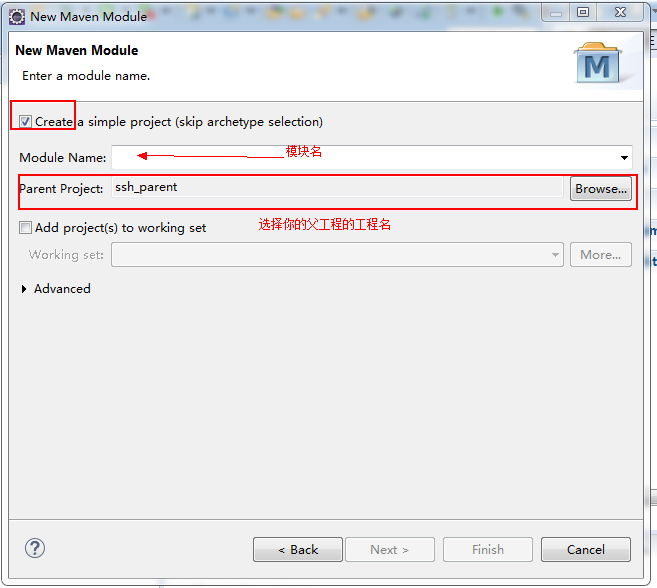


从它的目录结构可以看出，父工程本身不写代码，它里面有一个pom.xml文件，这个文件可以将多个子模块中通用的jar所对应的坐标，集中在父工程中配置，将来的子模块就可以不需要在pom.xml中配置通用jar的坐标了

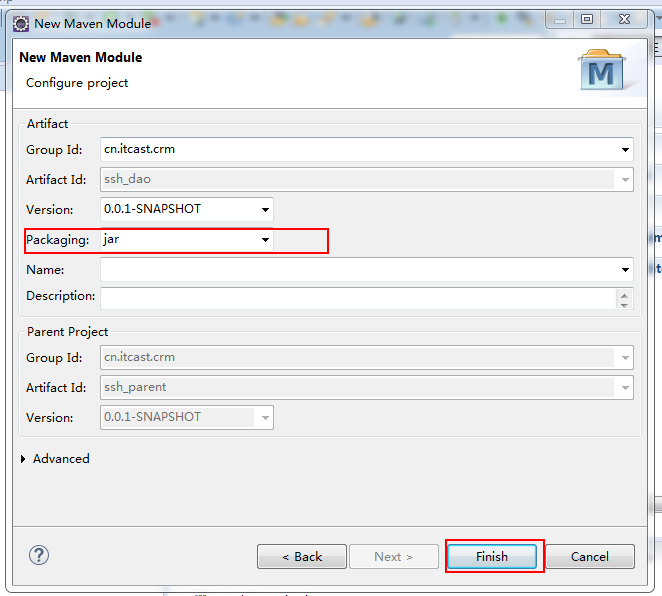
1. 如何创建这个父工程的一个子模块？



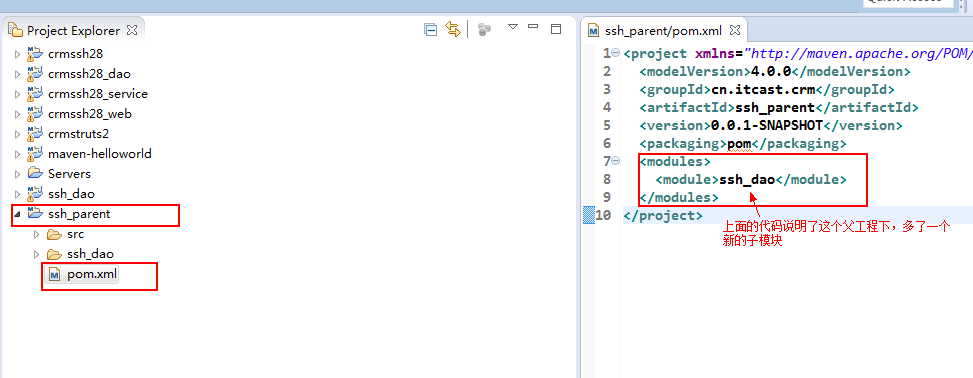
点next,进入如下图：



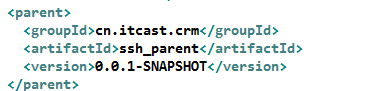
点next,进入如下图：



1. 再次查看父工程的pom.xml文件



1. 查看子模块的pom.xml，发现多了一个 parent结点



并且内部所包含的结点，其实就是父工程的坐标

坐标=groupId+artifactId+version

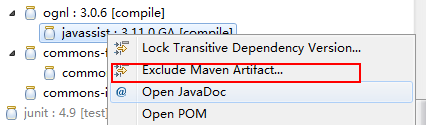
组织名 项目名 版本

## 三．冲突问题的解决

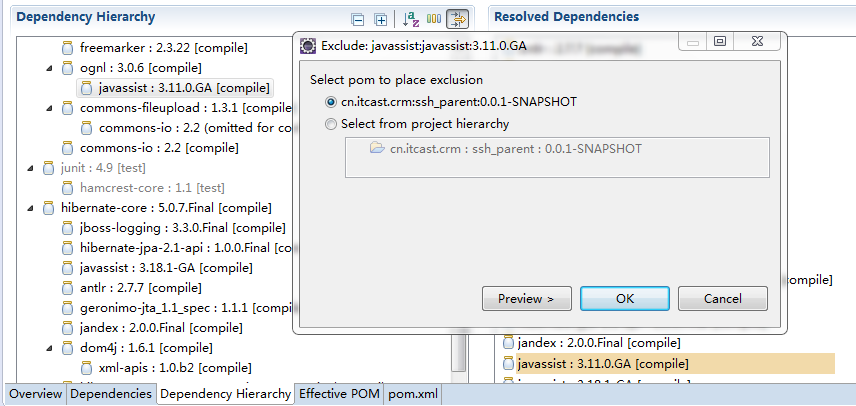
### 1.通过添加<exclusion>标签来解决冲突

在父工程中引入了struts-core,hibernate-core，就发现jar包是有冲突的

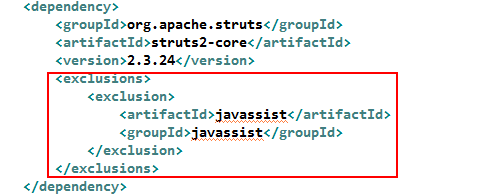
Javassist存在版本上冲突问题



进入下图：



背后的父工程的pom.xml文件中，添加的内容



2.依赖调解原则：

maven自动按照下边的原则调解：

* **1、第一声明者优先原则**

在pom文件定义依赖，先声明的依赖为准。

测试：

如果将上边struts-spring-plugins和spring-context顺序颠倒，系统将导入spring-beans-4.2.4。

分析：

由于spring-context在前边以spring-context依赖的spring-beans-4.2.4为准，所以最终spring-beans-4.2.4添加到了工程中。

* **2、路径近者优先原则**

例如：A依赖 spirng-beans-4.2.4，A依赖B依赖 spirng-beans-3.0.5，则spring-beans-4.2.4优先被依赖在A中，因为spring-beans-4.2.4相对spirng-beans-3.0.5被A依赖的路径最近。

测试：

在本工程中的pom中加入spirng-beans-4.2.4的依赖，根据路径近者优先原则，系统将导入spirng-beans-4.2.4：

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

### 2.使用版本锁定实现冲突解决

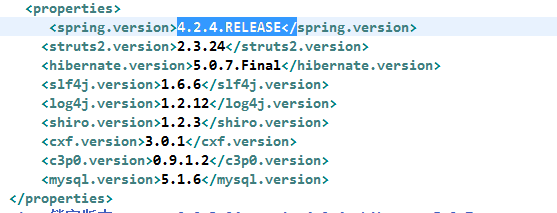
首先父工程中pom.xml文件添加：



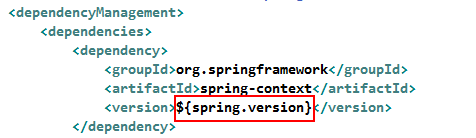


在使用坐标时，对于同一个框架，引入多次时，它的版本信息就会多次出现，所以

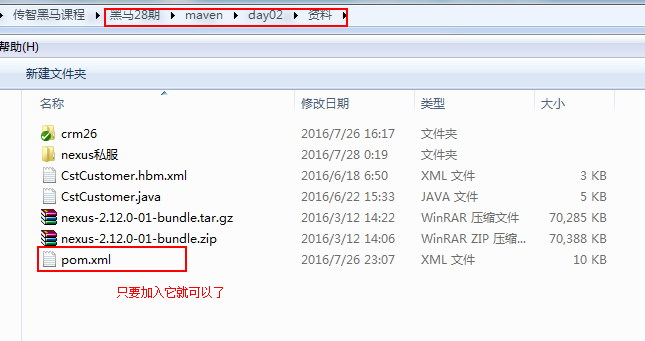
可以借用常量的思想，将这些版本号提取出来，在需要用到的时候，直接写版本的常量名称就可以了。



引用上面的常量

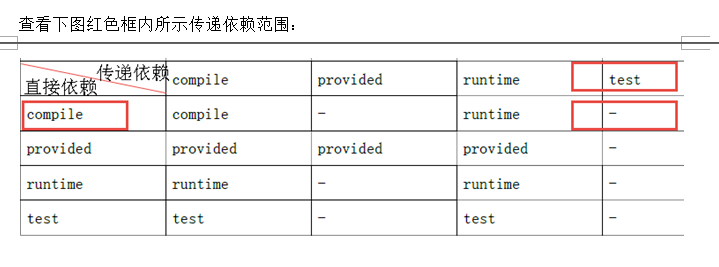


### 3.最终在ssh\_parent的pom.xml中引入的坐标



## 依赖关系

依赖具有传递性，但不是无限传递的，传递的规则如下：



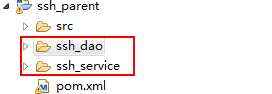
解决方法：

如果在依赖传递过程中，导致jar包丢失，我们的做法很简单，就是再导入一次坐标

## 四．编写子模块（Server模块举例）

1.创建一个maven module项目

创建结束后，父工程中结构如下：



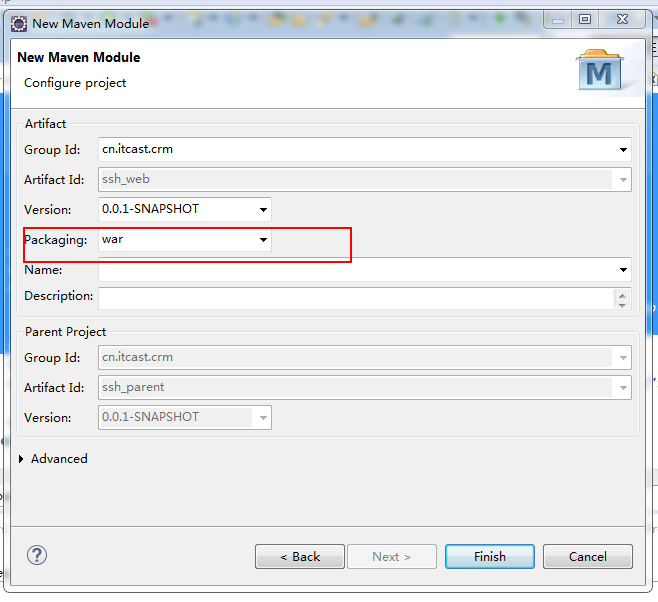
父工程的pom.xml文件如下

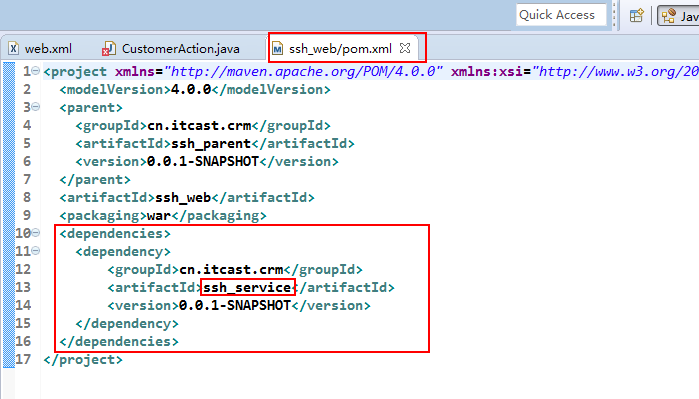


2.在service的pom.xml文件中引入dao的jar包



Web层的子模块创建：





## 五、私服搭建

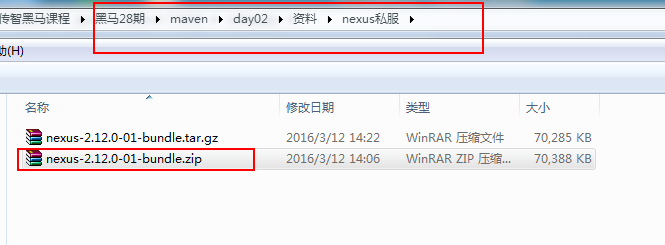
### 下载nexus

Nexus 是Maven仓库管理器，通过nexus可以搭建maven仓库，同时nexus还提供强大的仓库管理功能，构件搜索功能等。

下载Nexus， 下载地址：http://www.sonatype.org/nexus/archived/

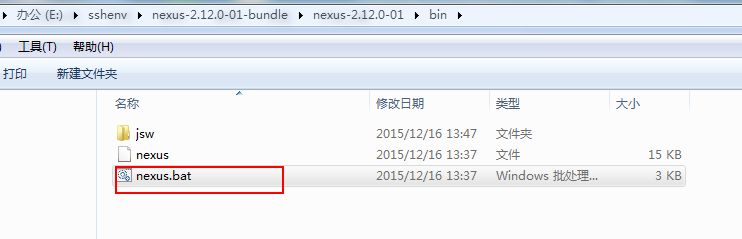


下载：nexus-2.12.0-01-bundle.zip



安装 ：

1.解压，进入指定的目录



2.安装并启动这个应用程序

cmd进入bin目录，执行nexus.bat install（尽量采用管理员模式）



安装成功在服务中查看有nexus服务：



通过从cmd端启动后如果出现问题提示，'findstr' 不是内部或外部命令，也不是可运行的程序或批处理文件；

这是PATH环境变量的问题，将windows命令的目录添加到PATH中就好了。

即：在path中追加：%SystemRoot%/system32;%SystemRoot%;这两个

### 卸载nexus

cmd进入nexus的bin目录，执行：nexus.bat uninstall



查看window服务列表nexus已被删除。

### 启动nexus

方法1：

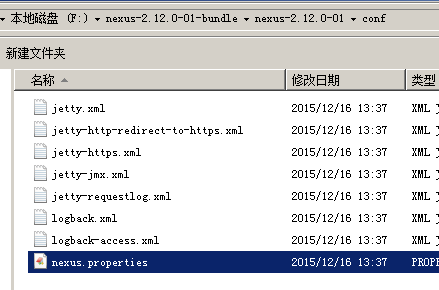
cmd进入bin目录，执行nexus.bat start

方法2：

直接启动nexus服务



查看nexus的配置文件conf/nexus.properties



# Jetty section

application-port=8081 # nexus的访问端口配置

application-host=0.0.0.0 # nexus主机监听配置(不用修改)

nexus-webapp=${bundleBasedir}/nexus # nexus工程目录

nexus-webapp-context-path=/nexus # nexus的web访问路径

# Nexus section

nexus-work=${bundleBasedir}/../sonatype-work/nexus # nexus仓库目录

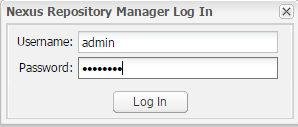
runtime=${bundleBasedir}/nexus/WEB-INF # nexus运行程序目录

访问：

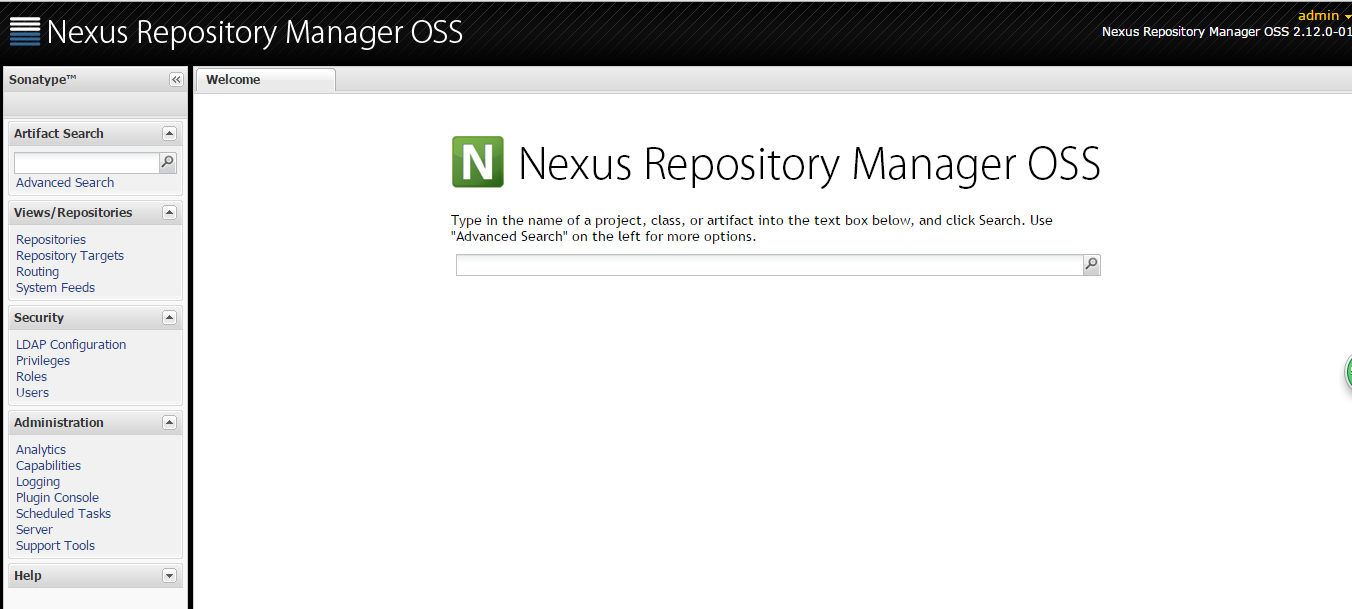
http://localhost:8081/nexus/

使用Nexus 内置账户admin/admin123登陆：

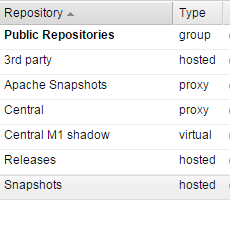
点击右上角的Log in，输入账号和密码 登陆



登陆成功：

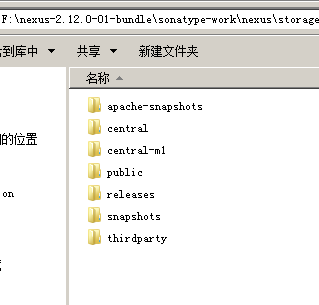


nexus的仓库有4种类型：



1. hosted，宿主仓库，部署自己的jar到这个类型的仓库，包括releases和snapshot两部分，Releases公司内部发布版本仓库、 Snapshots 公司内部测试版本仓库
2. proxy，代理仓库，用于代理远程的公共仓库，如maven中央仓库，用户连接私服，私服自动去中央仓库下载jar包或者插件。
3. **group，仓库组，用来合并多个hosted/proxy仓库，通常我们配置自己的maven连接仓库组。**
4. virtual(虚拟)：兼容Maven1 版本的jar或者插件

nexus仓库默认在sonatype-work目录中：



* **central：代理仓库，代理中央仓库**



* **apache-snapshots：代理仓库**

存储snapshots构件，代理地址https://repository.apache.org/snapshots/

* **central-m1：**virtual类型仓库，兼容Maven1 版本的jar或者插件
* **releases：本地仓库，存储releases构件。**
* **snapshots：本地仓库，存储snapshots构件。**
* **thirdparty：第三方仓库**
* **public：仓库组**

需求 ：将ssh\_dao的这个工程打成jar包，并放入到私服上去.

### 配置

第一步： 需要在客户端即部署dao工程的电脑上配置 maven环境，并修改 settings.xml 文件，配置连接私服的用户和密码 。

此用户名和密码用于私服校验，因为私服需要知道上传都 的账号和密码 是否和私服中的账号和密码 一致。

<server>

<id>releases</id>

<username>admin</username>

<password>admin123</password>

</server>

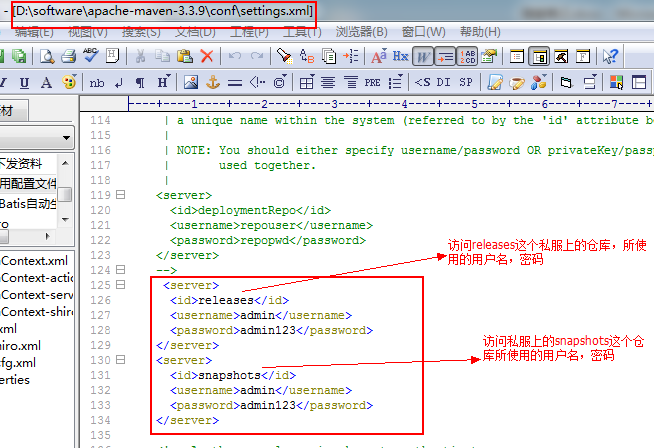
<server>

<id>snapshots</id>

<username>admin</username>

<password>admin123</password>

</server>



releases 连接发布版本项目仓库

snapshots 连接测试版本项目仓库



第二步： 配置项目pom.xml

配置私服仓库的地址，本公司的自己的jar包会上传到私服的宿主仓库，根据工程的版本号决定上传到哪个宿主仓库，如果版本为release则上传到私服的release仓库，如果版本为snapshot则上传到私服的snapshot仓库

<distributionManagement>

<repository>

<id>releases</id>

<url>http://localhost:8081/nexus/content/repositories/releases/</url>

</repository>

<snapshotRepository>

<id>snapshots</id>

<url>http://localhost:8081/nexus/content/repositories/snapshots/</url>

</snapshotRepository>

</distributionManagement>

注意：pom.xml这里<id> 和 settings.xml 配置 <id> 对应！

### 测试

将项目dao工程打成jar包发布到私服：

1、首先启动nexus

2、对dao工程执行deploy命令

## 从私服下载jar包

### 需求

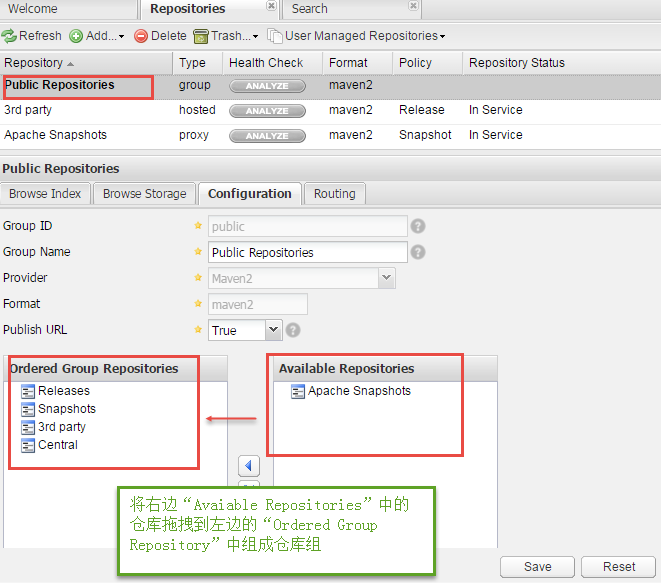
没有配置nexus之前，如果本地仓库没有，去中央仓库下载，通常在企业中会在局域网内部署一台私服服务器，有了私服本地项目首先去本地仓库找jar，如果没有找到则连接私服从私服下载jar包，如果私服没有jar包私服同时作为代理服务器从中央仓库下载jar包，这样做的好处是一方面由私服对公司项目的依赖jar包统一管理，一方面提高下载速度，项目连接私服下载jar包的速度要比项目连接中央仓库的速度快的多。

本例子测试从私服下载dao 工程jar包。

### 管理仓库组

nexus中包括很多仓库，hosted中存放的是企业自己发布的jar包及第三方公司的jar包，proxy中存放的是中央仓库的jar，为了方便从私服下载jar包可以将多个仓库组成一个仓库组，每个工程需要连接私服的仓库组下载jar包。

打开nexus配置仓库组，如下图：



上图中仓库组包括了本地仓库、代理仓库等。

### 在setting.xml中配置仓库

在客户端的setting.xml中配置私服的仓库，由于setting.xml中没有repositories的配置标签需要使用profile定义仓库。

<profile>

<!--profile的id-->

<id>dev</id>

<repositories>

<repository>

<!--仓库id，repositories可以配置多个仓库，保证id不重复-->

<id>nexus</id>

<!--仓库地址，即nexus仓库组的地址-->

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

<!--是否下载releases构件-->

<releases>

<enabled>true</enabled>

</releases>

<!--是否下载snapshots构件-->

<snapshots>

<enabled>true</enabled>

</snapshots>

</repository>

</repositories>

<pluginRepositories>

<!-- 插件仓库，maven的运行依赖插件，也需要从私服下载插件 -->

<pluginRepository>

<!-- 插件仓库的id不允许重复，如果重复后边配置会覆盖前边 -->

<id>public</id>

<name>Public Repositories</name>

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

</pluginRepository>

</pluginRepositories>

</profile>

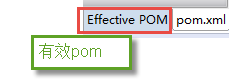
使用profile定义仓库需要激活才可生效。

<activeProfiles>

<activeProfile>dev</activeProfile>

</activeProfiles>

配置成功后通过eclipse查看有效pom，有效pom是maven软件最终使用的pom内容，程序员不直接编辑有效pom，打开有效pom



有效pom内容如下：

下边的pom内容中有两个仓库地址，maven会先从前边的仓库的找，如果找不到jar包再从下边的找，从而就实现了从私服下载jar包。

<repositories>

<repository>

<releases>

<enabled>true</enabled>

</releases>

<snapshots>

<enabled>true</enabled>

</snapshots>

<id>public</id>

<name>Public Repositories</name>

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

</repository>

<repository>

<snapshots>

<enabled>false</enabled>

</snapshots>

<id>central</id>

<name>Central Repository</name>

<url>https://repo.maven.apache.org/maven2</url>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<id>public</id>

<name>Public Repositories</name>

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

</pluginRepository>

<pluginRepository>

<releases>

<updatePolicy>never</updatePolicy>

</releases>

<snapshots>

<enabled>false</enabled>

</snapshots>

<id>central</id>

<name>Central Repository</name>

<url>https://repo.maven.apache.org/maven2</url>

</pluginRepository>

</pluginRepositories>