maven私服nexus3.x环境配置

开发工具 (/categories/开发工具) nexus (/tags/nexus) 2017/09/02 921

私服是指私有服务器，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建。 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。

Nexus是一个强大的Maven仓库管理器，它极大地简化了本地内部仓库的维护和外部仓库的访问。 如果使用了公共的Maven仓库服务器，可以从Maven中央仓库下载所需要的构件（Artifact），但这通常不是一个好的做法。 正常做法是在本地架设一个本地Maven仓库服务器，利用Nexus私服可以只在一个地方就能够完全控制访问和部署在你所维护仓库中的每个Artifact。

Nexus优点

为什么要构建Nexus私服？好处我随便列几点：

- Nexus在代理远程仓库的同时维护本地仓库，以降低中央仓库的负荷,节省外网带宽和时间，Nexus私服就可以满足这样的需要。
- Nexus是一套“开箱即用”的系统不需要数据库，它使用文件系统加Lucene来组织数据。
- Nexus使用ExtJS来开发界面，利用Restlet来提供完整的REST APIs，并能通过插件和各种IDE集成。
- Nexus支持WebDAV与LDAP安全身份认证。
- Nexus还提供了强大的仓库管理功能，构件搜索功能，它基于REST，提供友好的UI，占用较少的内存，基于简单文件系统而非数据库。

安装

我的环境是centos7.2 + JDK8 + Maven3，首先需要安装 JDK8 和 Maven3，这里不多讲。

去官网下载最新的 download nexus (https://www.sonatype.com/download-oss-sonatype)

下载文件nexus-3.6.0-02-unix.tar.gz，解压缩。

```
tar zxf nexus-3.6.0-02-unix.tar.gz -C /usr/local/
cd /usr/local/
mv nexus-3.6.0-02/ nexus
```

启动：

```
./nexus/bin/nexus start
```

默认端口8081，如果开了防火墙，就把这个端口放开：

```
firewall-cmd --zone=public --add-port=8081/tcp --permanent
```

启动之后，运行命令`lsof -i:8081`可查看是否成功启动了。

Nexus默认的端口是8081，可以在etc/nexus-default.properties配置中修改。

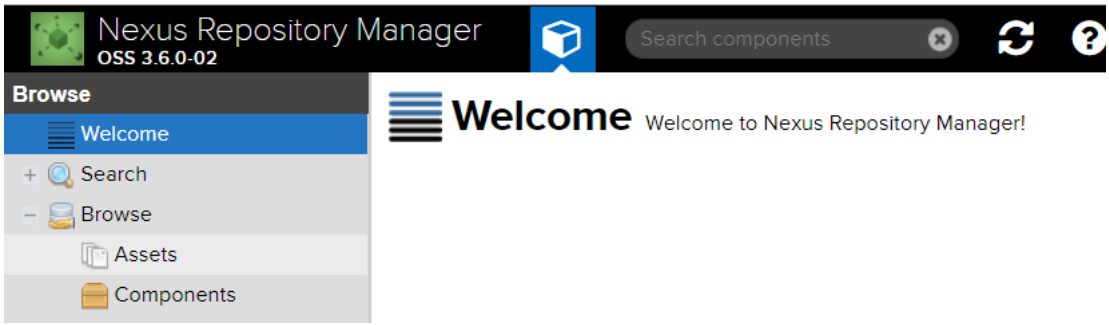
Nexus默认的用户名密码是admin/admin123

当遇到奇怪问题时，重启nexus，重启后web界面要1分钟左右后才能访问。

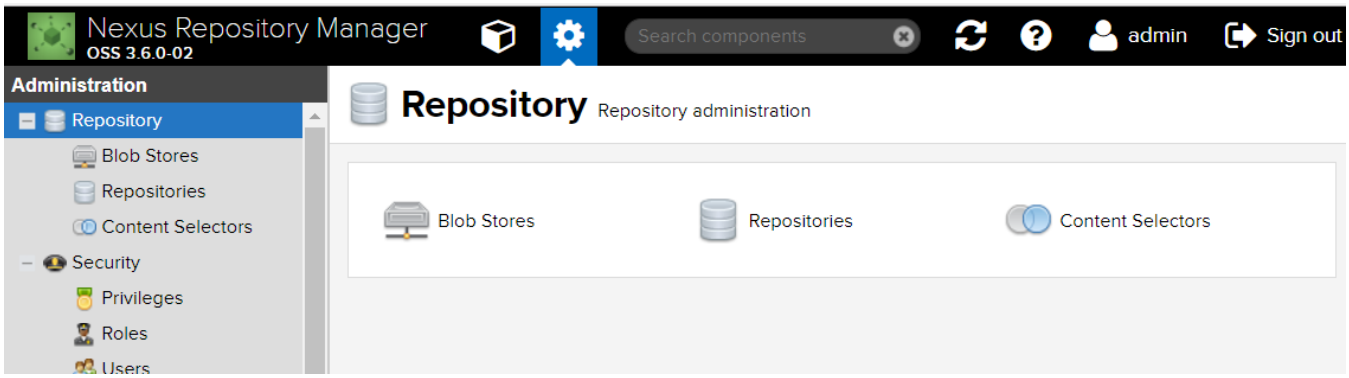
Nexus的工作目录是`sonatype-work`（路径一般在nexus同级目录下），日志文件也在这里。

```
ls sonatype-work/nexus3/
backup  blobs  cache  db  elasticsearch  etc  generated-bundles
health-check  instances  keystores  lock  log  orient  port  tmp
```

访问：http://localhost:8081 (http://localhost:8081)，效果如下：

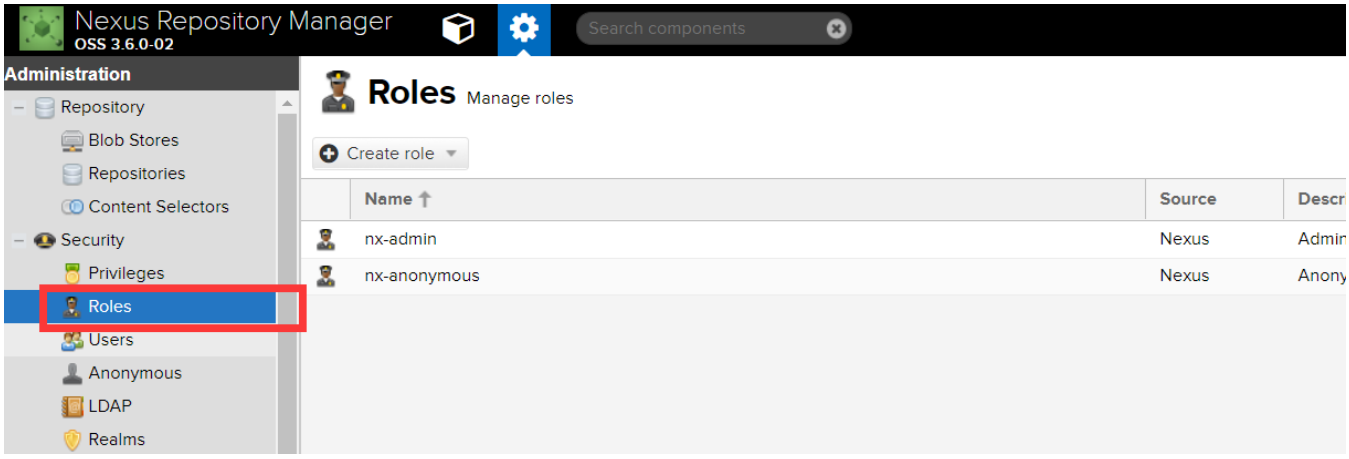


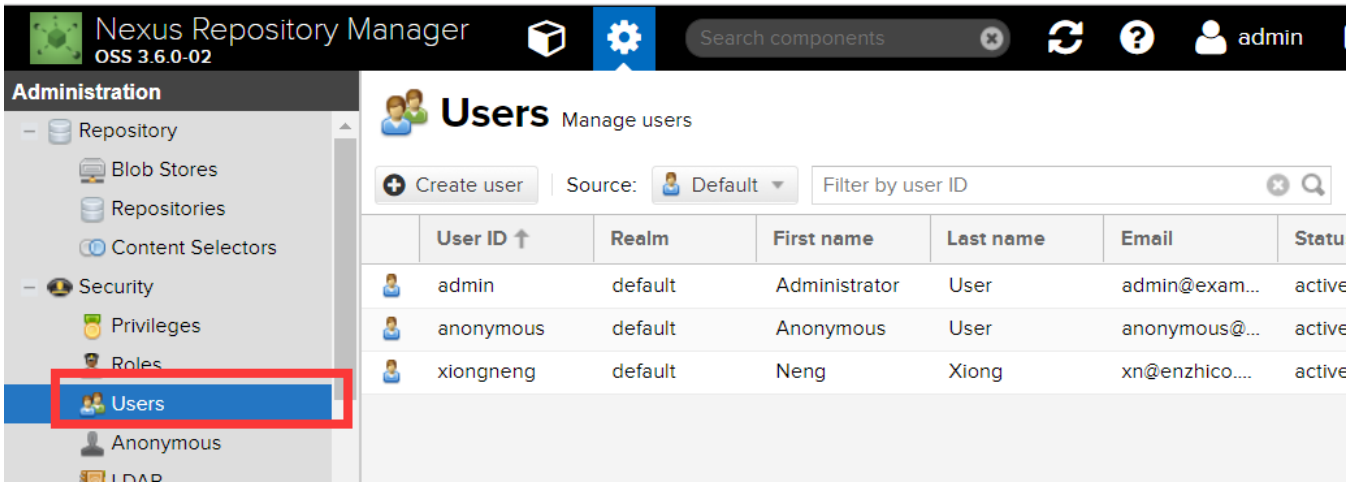
使用默认的管理员admin/admin123登录，进入管理界面：



用户和角色

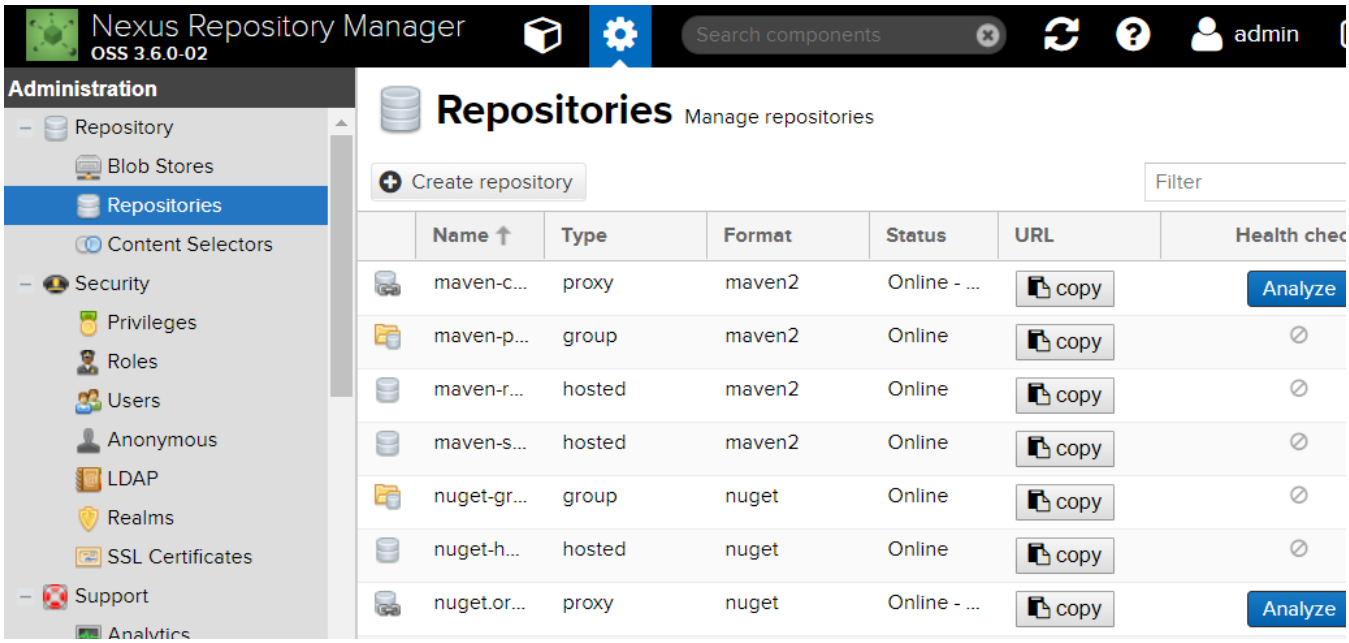
可以点击上面的“设置”图标，在“设置”里可以添加用户、角色，对接LDAP等的设置。





仓库

最核心的是仓库管理



默认的这几个仓库我解释一下：

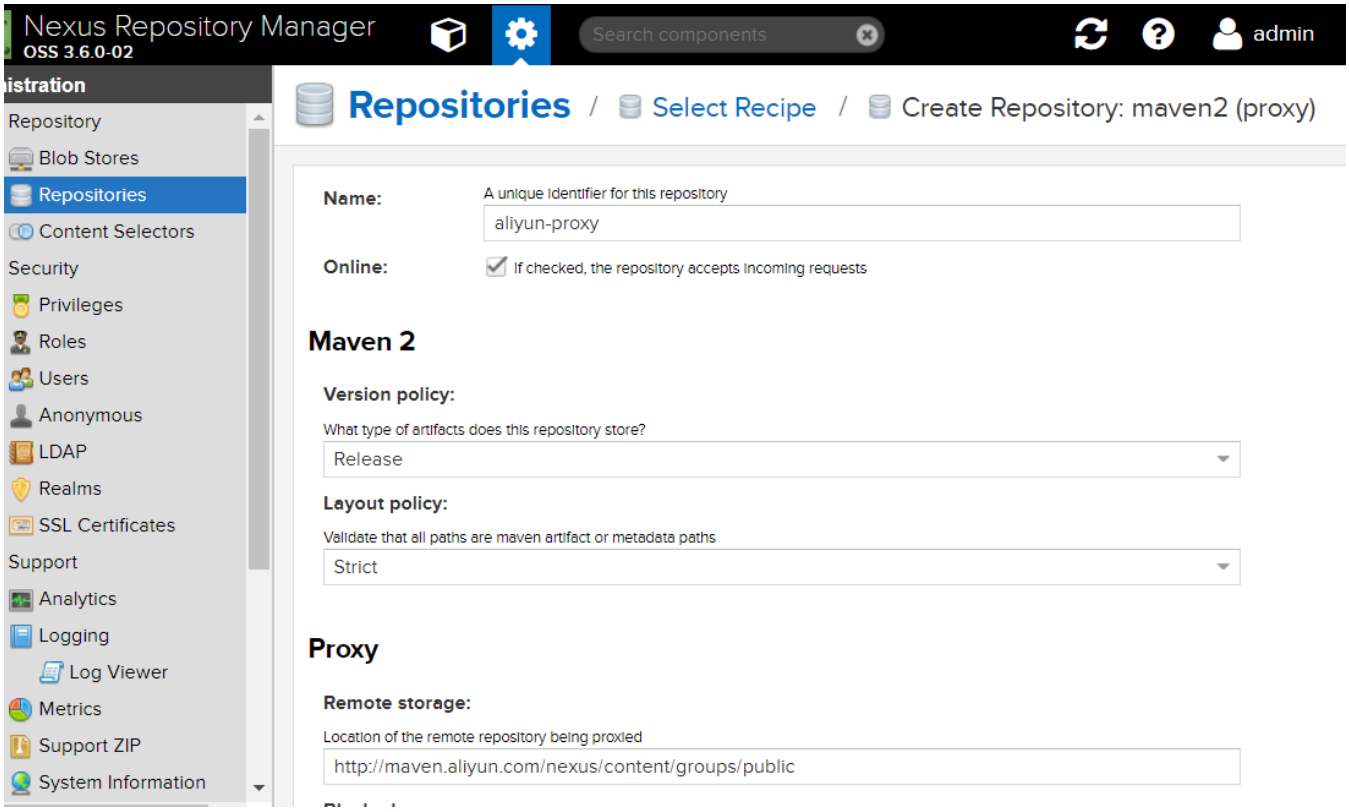
1. maven-central：maven中央库，默认从https://repo1.maven.org/maven2/拉取jar (https://repo1.maven.org/maven2/拉取jar)
2. maven-releases：私库发行版jar，初次安装请将`Deployment policy`设置为`Allow redeploy`
3. maven-snapshots：私库快照（调试版本）jar
4. maven-public：仓库分组，把上面三个仓库组合在一起对外提供服务，在本地maven基础配置`settings.xml`中使用。

Nexus默认的仓库类型有以下四种：

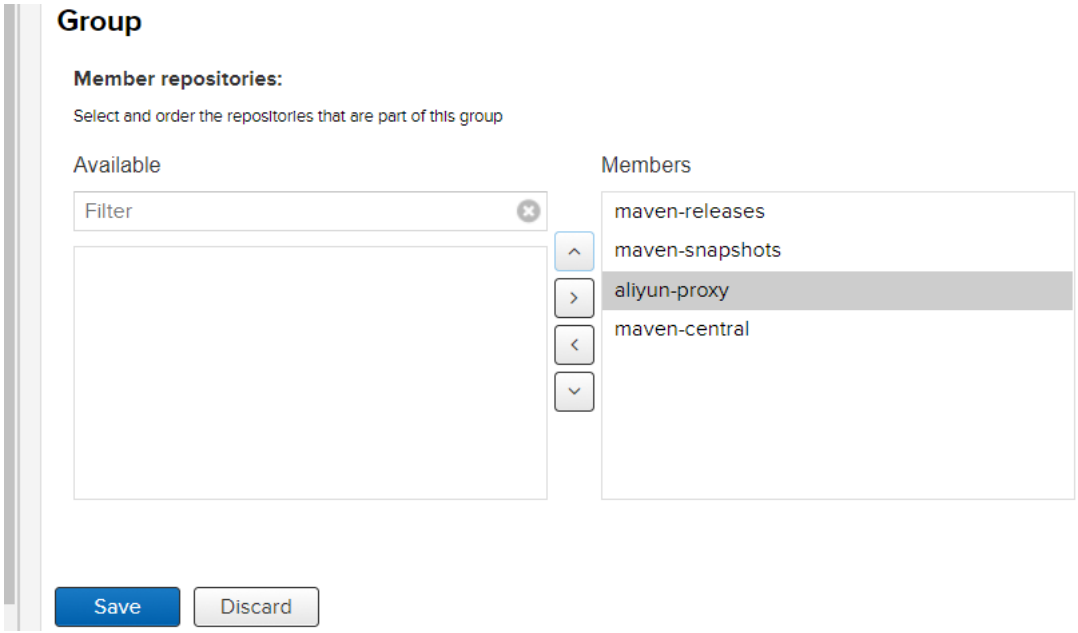
1. group(仓库组类型)：又叫组仓库，用于方便开发人员自己设定的仓库；
2. hosted(宿主类型)：内部项目的发布仓库（内部开发人员，发布上去存放的仓库）；
3. proxy(代理类型)：从远程中央仓库中寻找数据的仓库（可以点击对应的仓库的Configuration页签下Remote Storage属性的值即被代理的远程仓库的路径）；
4. virtual(虚拟类型)：虚拟仓库（这个基本用不到，重点关注上面三个仓库的使用）；

Policy(策略): 表示该仓库为发布(Release)版本仓库还是快照(Snapshot)版本仓库；

由于访问中央仓库有时候会比较慢，这里我添加一个阿里云的代理仓库，然后优先级放到默认中央库之前，， 阿里云的maven仓库url为`http://maven.aliyun.com/nexus/content/groups/public`



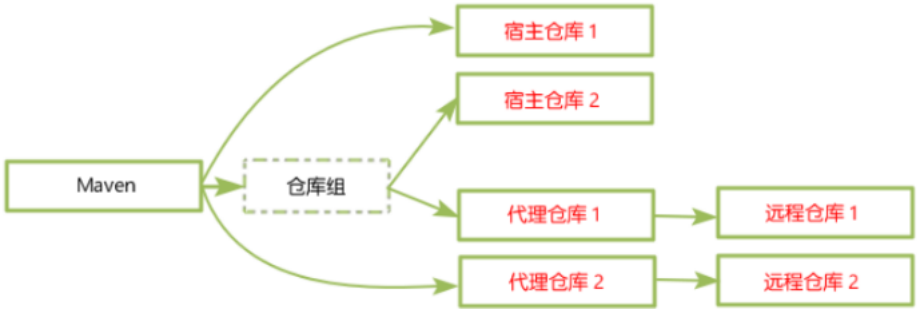
然后再public组里面讲这个`aliyun-proxy`仓库加入，排在`maven-central`之前即可。



Nexus仓库分类的概念

1）Maven可直接从宿主仓库下载构件,也可以从代理仓库下载构件,而代理仓库间接的从远程仓库下载并缓存构件

2）为了方便,Maven可以从仓库组下载构件,而仓库组并没有时间的内容(下图中用虚线表示,它会转向包含的宿主仓库或者代理仓库获得实际构件的内容)

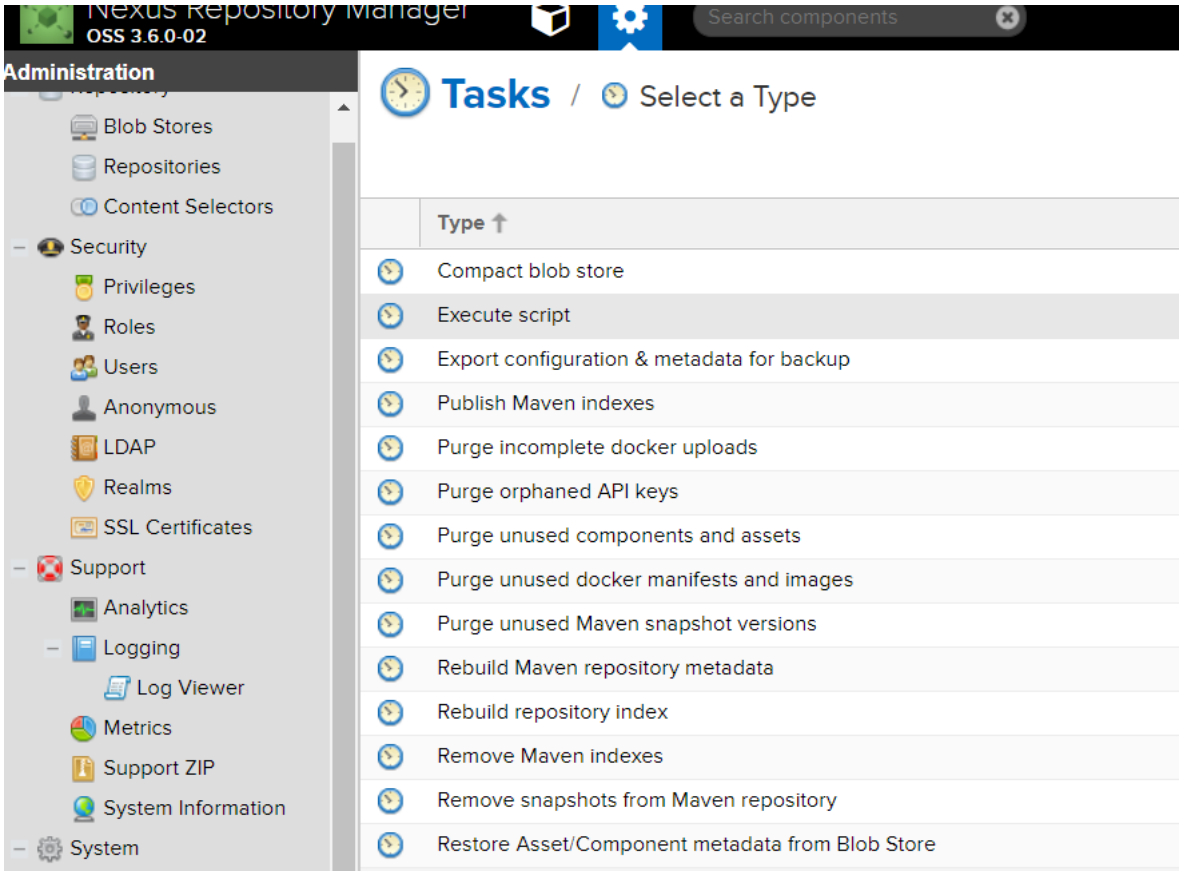


Nexus的调度任务

默认安装好之后是没有索引和jar文件的，因为你要自己定义任务去执行。

Nexus提供了一系列可配置的调度任务来方便用户管理系统。用户可以设定这些任务运行的方式，例如每天、每周等。调度任务会在适当的时候在后台运行。

要建立一个调度任务，单击左边导航菜单中的Tasks，点击`Create Task`，然后选择一个任务类型。



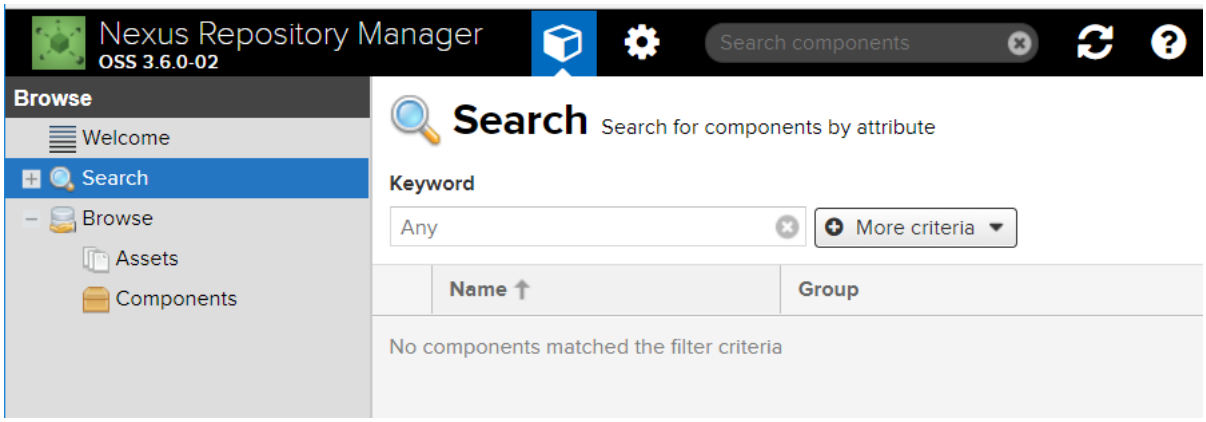
以下几种常用类型的调度任务：

- Execute script：执行自定义脚本
- Purge开头：清理一些不使用的资源。
- Rebuild repository index：为仓库重新编纂索引，从远仓库下载最新的索引。
- Rebuild Maven repository metadata：基于仓库内容重新创建仓库元数据文件，同时重新创建每个文件的校验和md5与sha1。
- Remove snapshots from Maven repository：把快照删了，这个是在稳定版发布后清除

比如我新建一个重构索引的任务，然后选择aliyun仓库，让它把远程索引取下来，手动执行。不过最好别这样做，因为需要很大的硬盘空间。最好是让它自己去维护，请求一个依赖的时候如果私服没有会自动去远仓库取的。

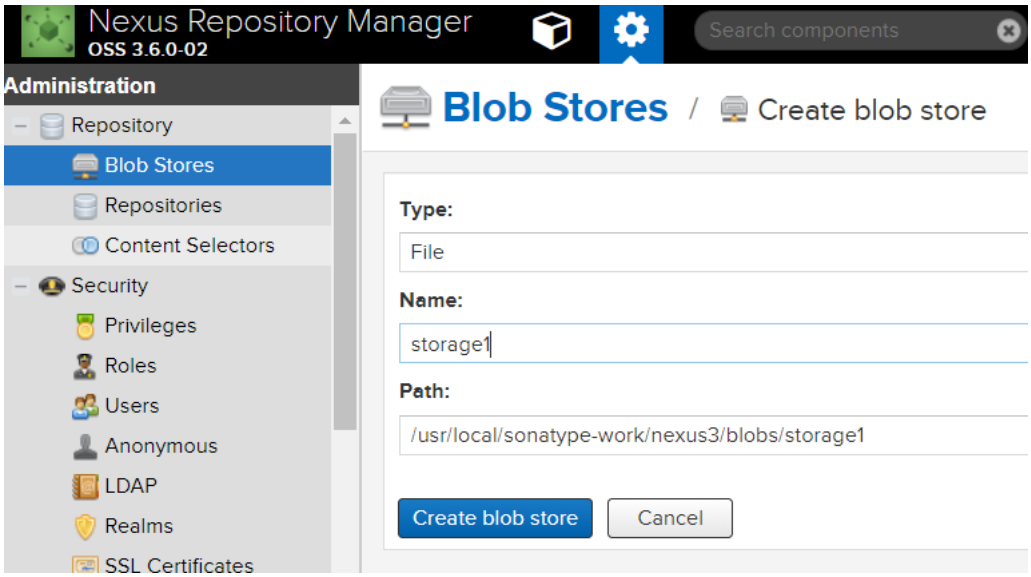
Nexus搜索页

这个不需要登录就可以访问，用来查询jar包。支持模糊查询



Blob Stores

文件存储的地方，创建一个目录的话，对应文件系统的一个目录，可供仓库上传文件使用，如图所示：



本地Maven使用私服

安装和配置好之后，在开发中如何使用呢。可在maven的默认配置`settings.xml`中修改如下：

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

<mirrors>
    <mirror>
        <id>nexus</id>
        <mirrorOf>*</mirrorOf>
        <url>http://123.207.66.156:8081/repository/maven-public/</url>
    </mirror>
</mirrors>

<profiles>
    <profile>
      <id>dev</id>
      <repositories>
        <repository>
          <id>Nexus</id>
          <url>http://123.207.66.156:8081/repository/maven-public/</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>true</enabled>
            <updatePolicy>always</updatePolicy>
          </snapshots>
        </repository>
      </repositories>
      <activation>
        <activeByDefault>true</activeByDefault>
        <jdk>1.8</jdk>
      </activation>
      <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
        <maven.compiler.compilerVersion>1.8</maven.compiler.compilerVersion>
      </properties>
    </profile>
</profiles>
<activeProfiles>
    <activeProfile>dev</activeProfile>
</activeProfiles>
```

如果要发布自己的jar到私服，就需要修改工程的`pom.xml`，添加如下内容，否则什么都不用做：

```
<distributionManagement>
    <repository>
        <id>releases</id>
        <name>Releases</name>
        <url>http://123.207.66.156:8081/repository/maven-releases/</url>
    </repository>
    <snapshotRepository>
        <id>snapshots</id>
        <name>Snapshot</name>
        <url>http://123.207.66.156:8081/repository/maven-snapshots/</url>
    </snapshotRepository>
</distributionManagement>
```

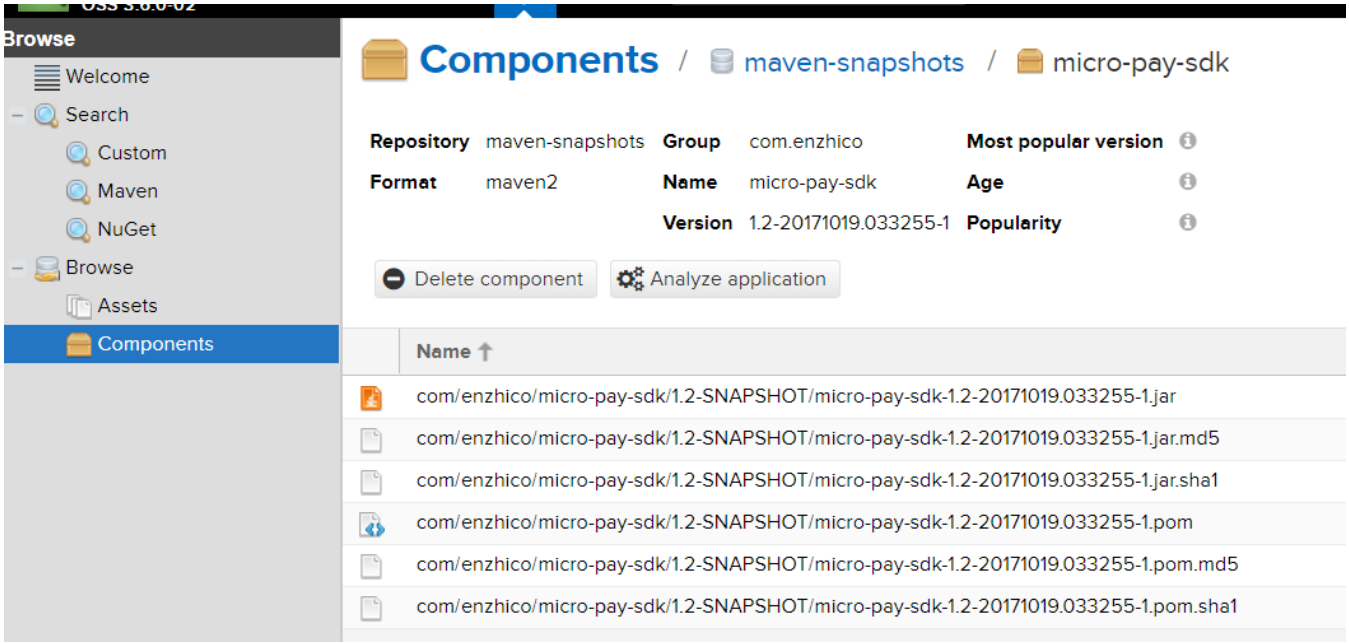
注意上面的repository的id值一定要跟`settings.xml`文件中配置的server一致。

上传到Nexus上，使用`mvn deploy`即可，开发的时候请使用snapshot版本，也就是version的后缀必须是`-SNAPSHOT`。

```
<groupId>com.enzhico</groupId>
<artifactId>micro-pay-sdk</artifactId>
<version>1.2-SNAPSHOT</version>
<packaging>jar</packaging>
```

```
Downloading: http://123.207.66.156:8081/repository/maven-snapshots/com/enzhico/micro-pay-sd
Uploading: http://123.207.66.156:8081/repository/maven-snapshots/com/enzhico/micro-pay-sdk/
Uploaded: http://123.207.66.156:8081/repository/maven-snapshots/com/enzhico/micro-pay-sdk/1
Uploading: http://123.207.66.156:8081/repository/maven-snapshots/com/enzhico/micro-pay-sdk/
Uploaded: http://123.207.66.156:8081/repository/maven-snapshots/com/enzhico/micro-pay-sdk/m
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 6.555 s
[INFO] Finished at: 2017-10-19T11:32:56+08:00
[INFO] Final Memory: 16M/190M
[INFO] ------------------------------------------------------------------------
```

上次成功后再去私服仓库查询快照版发现已经成功上传：



第三方Jar上传到Nexus

```
mvn deploy:deploy-file \
    -DgroupId=<group-id> \
    -DartifactId=<artifact-id> \
    -Dversion=<version> \
    -Dpackaging=<type-of-packaging> \
    -Dfile=<path-to-file> \
    -DrepositoryId=<server-id-settings.xml> \
    -Durl=<url-of-the-repository-to-deploy>
```

`-DrepositoryId`的值即为在`setttings.xml`里面配置的server id。

上次不同JDK版本

pom.xml里面配置多个profile，其中一个默认的：

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.6.1</version>
            <configuration>
                <source>${jar.source}</source>
                <target>${jar.target}</target>
                <encoding>UTF-8</encoding>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-deploy-plugin</artifactId>
            <version>2.8.2</version>
            <executions>
                <execution>
                    <id>deploy</id>
                    <phase>deploy</phase>
                    <goals>
                        <goal>deploy</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>

<profiles>
    <profile>
        <id>default</id>
        <activation>
            <activeByDefault>true</activeByDefault>
        </activation>
        <properties>
            <jar.source>1.8</jar.source>
            <jar.target>1.8</jar.target>
        </properties>
    </profile>
    <profile>
        <id>jdk16</id>
        <build>
            <plugins>
                <plugin>
                    <artifactId>maven-jar-plugin</artifactId>
                    <executions>
                        <execution>
                            <phase>package</phase>
                            <goals>
                                <goal>jar</goal>
                            </goals>
                            <configuration>
                                <classifier>jdk16</classifier>
                            </configuration>
                        </execution>
                    </executions>
                </plugin>
            </plugins>
        </build>
        <properties>
            <jar.source>1.6</jar.source>
            <jar.target>1.6</jar.target>
        </properties>
```

```
    </profile>
</profiles>
```

上面我定义了两个profile，那么打包或者发布的时候可指定不同的JDK版本：

```
# 默认版本JDK1.8
mvn clean && mvn deploy
# JDK1.6版本
mvn clean && mvn deploy -P jdk16
```

第一条命令打包使用默认的profile，编译的版本是1.8，生成的文件是xxx-SNAPSHOT.jar； 而第二条命令打包指定使用jdk16这个profile，编译版本是1.6，生成的文件是xxx-SNAPSHOT-jdk16.jar。

项目中引用的时候可通过指定classifier：

```
<dependency>
    <groupId>com.enzhico</groupId>
    <artifactId>adm-traffic-common-model</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <classifier>jdk16</classifier>
</dependency>
```

发布源码和文档

如果你还想发布源码和javadoc，那么需要使用maven插件，我把插件配置列出来：

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.7.0</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
                <encoding>UTF-8</encoding>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-resources-plugin</artifactId>
            <configuration>
                <encoding>UTF-8</encoding>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-source-plugin</artifactId>
            <version>3.0.1</version>
            <executions>
                <execution>
                    <id>attach-sources</id>
                    <goals>
                        <goal>jar</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-javadoc-plugin</artifactId>
            <version>2.10.4</version>
            <configuration>
                <encoding>UTF-8</encoding>
                <aggregate>true</aggregate>
                <charset>UTF-8</charset>
                <docencoding>UTF-8</docencoding>
            </configuration>
            <executions>
                <execution>
                    <id>attach-javadocs</id>
                    <goals>
                        <goal>jar</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-deploy-plugin</artifactId>
            <version>2.8.2</version>
            <executions>
                <execution>
                    <id>deploy</id>
                    <phase>deploy</phase>
                    <goals>
                        <goal>deploy</goal>
                    </goals>
                </execution>
            </executions>
```

```
        </plugin>
    </plugins>
</build>
```

发布javadoc的时候，每个方法注释必须遵循规范，比如参数、返回值、异常都应该有说明。

打包或发布的时候如果想跳过测试，加一个参数：

```
mvn clean && mvn deploy -DskipTests=true
```


‹ 上一篇 (/2017/09/06/spring/spring-axis2.html)

下一篇 › (/2017/08/19/spring/sb-echarts.html)