[Android 项目部署之Nexus私服搭建和应用](http://blog.csdn.net/l2show/article/details/48653949)

**一.概述**

       Nexus是一个基于maven的仓库管理的社区项目.主要的使用场景就是可以在局域网搭建一个maven私服,用来部署第三方公共构件或者作为远程仓库在该局域网的一个代理.简单举几个例子就是:

       1.第三方Jar包可以放在nexus上,项目可以直接通过Url和路径配置直接引用.方便进行统一管理.

       2.同时有多个项目在开发的时候,一些共用基础模块可以单独抽取到nexus上,需要用的项目直接从nexus上拉取就行(基础模块的实现,维护和部署可以交给专门的人员,其他项目不用关心代码实现,这样也可以达到保证核心代码不泄露).

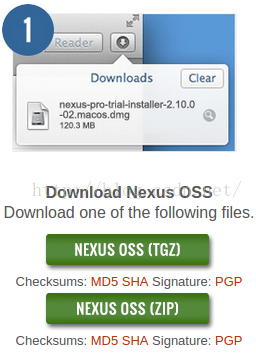
       3.一些封闭开发的过程中开发机是不能上公网的,所以连接central repository和下载jar就比较麻烦,这时就可以用nexus搭建起来一个介于公网和局域网之间的桥梁.

       接下来就简单介绍一下nexus的搭建以及在[**Android**](http://lib.csdn.net/base/android)项目中使用nexus的方式.

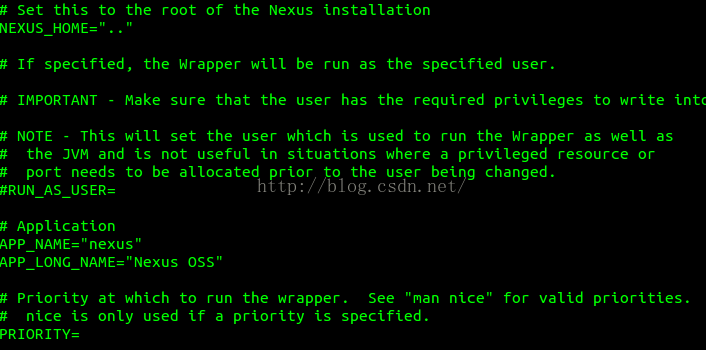
**二.搭建**

       1.下载&配置

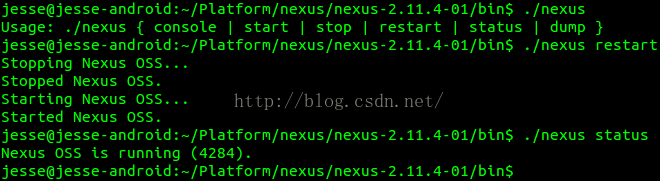
       这里使用的是Nexus OSS开源版,官网下载地址:<http://www.sonatype.org/nexus/go/> ,博主的开发环境是ubuntu,所以就下载的TGZ压缩的.



       把压缩包解压之后进入bin文件夹就可以运行shell脚本来启动nexus,通过查看bin/nexus脚本发现可以修改脚本来适配自己的需求,例如修改Nexus的root路径,如果需要以root身份来启动Nexus就需要设置RUN\_AS\_USER=root,设置app名字和登陆名字等.也可以去conf/nexus.properties文件修改端口之类的信息.

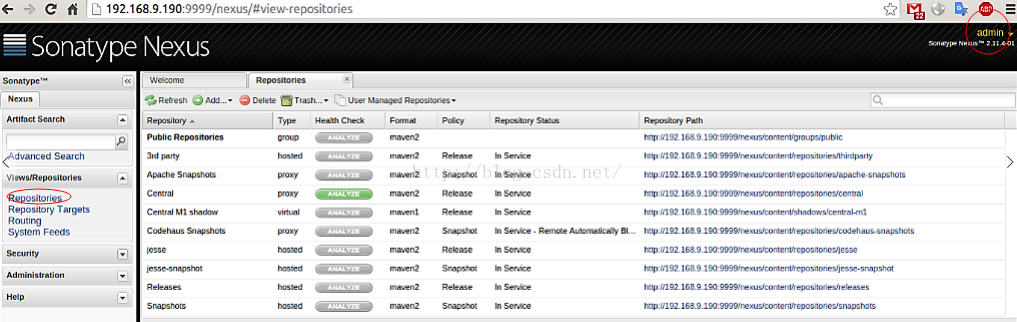


       接下来直接运行Nexus脚本,查看一下该脚本支持几个参数命令,将Nexus启动起来(在这之前博主的Nexus已经启动了所以就用的restart)



       2.创建仓库

       从状态上来看nexus已经启动起来了,Nexus启动之后就可以在浏览器访问类似 192.168.9.190:9999/nexus/ 地址,其中ip为当前服务器ip,端口Nexus默认为8081,可以在conf/nexus.properties文件中修改端口.进入web管理页面后右上角登陆,Nexus默认的账号密码是admin:admin123,然后点击左侧的repositories查看现有的仓库列表.



       这里的仓库分了四种类型

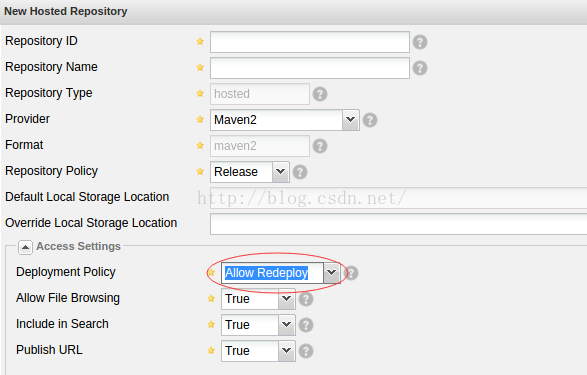
       1.hosted(宿主仓库):用来部署自己,第三方或者公共仓库的构件

       2.proxy(代理仓库):代理远程仓库

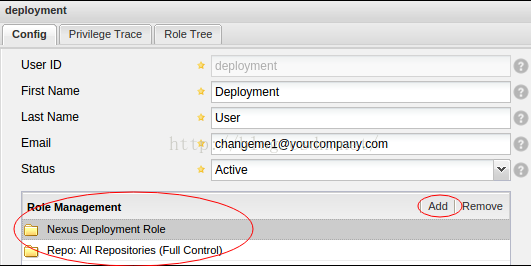
       3.virtual(虚拟仓库):默认提供了一个 Central M1虚拟仓库 用来将maven 2适配为maven 1

       4.group(仓库组):统一管理多个仓库

       这里以建立hosted仓库为例简单介绍Nexus在Android开发中的实际使用情况.点击repositories->add 键入ID(部署项目的标识) Name等属性,这里需要注意的是,如果该仓库有多次部署的情况的话,将policy设置为allow redeploy,不然后续在部署的时候会出现403错误.



       建立好新的仓库之后需要配置一下相关账号信息.在安全选项下选择用户选项,可以看到三个默认的账号,分别是管理员账号,部署账号和Nexus账号.正常访问仓库内容的时候是不需要这三个账户的,一般也就是把部署账号暴露出去,方便仓库项目维护人员部署项目使用.所以这里可以用默认的Deployment账户(记得reset下密码).也可以新建一个账号来使用,新建的时候可以通过add role management来控制该账号的权限.

       回到repositories选项就可以看到新建出来的仓库,点击仓库URL可以直接进入到仓库路径,当然现在还没有部署项目.到此为止搭建Maven私服Nexus端的配置和部署都已经完成,下面开始介绍Nexus Android端的配置部署和应用.

       3.Android端私服仓库应用

       这里就拿之前分析过的Android 4.4原生DeckClock程序举例.Android 4.4 中的DeckClock项目需要依赖framework下的一个名叫datetimepicker的库工程.这里就把这个datetimepicker工程部署到Nexus上,让DeskClock根据仓库url和路径进行访问和datetimepicker[**版本控制**](http://lib.csdn.net/base/git).

       首先在DeskClock项目的build.gradle中配置之前建立的仓库.

**[java]** [view plain](http://blog.csdn.net/l2show/article/details/48653949) [copy](http://blog.csdn.net/l2show/article/details/48653949)

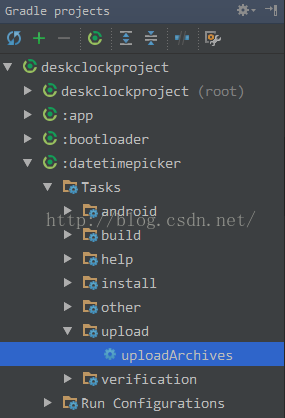
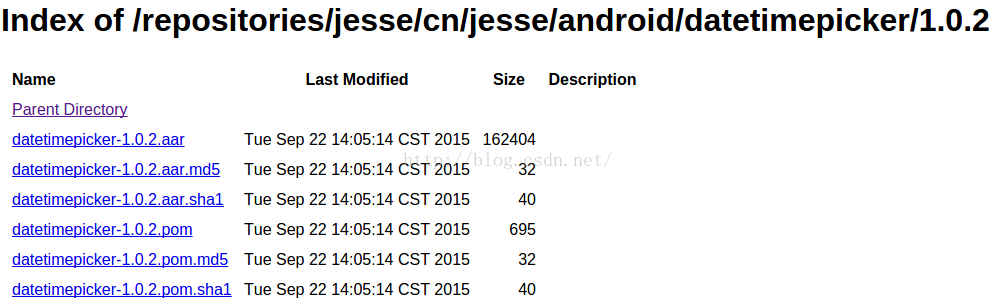
1. allprojects {
2. repositories {
3. jcenter()
4. mavenLocal()
5. }
6. dependencies{
7. repositories {
8. maven { url 'http://192.168.9.190:9999/nexus/content/repositories/jesse/' }
9. }
10. }
11. }

       接下来需要从项目中把datetimepicker模块剥离出来,因为我们要把它单独部署到刚刚配置的仓库里,DeskClock项目直接依赖datetimepicker模块改为通过Nexus依赖.剥离之后在datetimepicker模块中的build.gradle文件中使用uploadArchives方法向指定仓库上传有效的aar文件.并且配置pom文件信息.目前配置是默认走repository中的仓库,如果在版本号后面加上-SNAPSHOT就会走snapshotRepository中的仓库.artifactId是该模块的名字,groupId是仓库包名路径.

**[java]** [view plain](http://blog.csdn.net/l2show/article/details/48653949) [copy](http://blog.csdn.net/l2show/article/details/48653949)

1. uploadArchives {
2. repositories {
3. mavenDeployer {
4. snapshotRepository(url: 'http://192.168.9.190:8081/nexus/content/repositories/jesse-snapshot/') {
5. authentication(userName: 'Jesse', password: 'jesse')
6. }
8. repository(url: 'http://192.168.9.190:8081/nexus/content/repositories/jesse/') {
9. authentication(userName: 'Jesse', password: 'jesse')
10. }
12. pom.project {
13. version '1.0.2'
14. artifactId 'datetimepicker'
15. groupId 'cn.jesse.android'
16. packaging 'aar'
17. description 'dependences lib'
18. }
19. }
20. }
21. }
23. artifacts {
24. archives file('datetimepicker.aar')
25. }

       在Android Studio的Gradle project插件中选中datetimepicker模块运行uploadArchives任务就可以将该模块的aar文件以1.0.2的版本部署到Nexus上,供其他项目引用.

   
       将datetimepicker部署到Nexus上之后,就可以在主模块的build.gradle直接通过部署时设置的groupId和version建立与datetimepicker之间的依赖.

**[java]** [view plain](http://blog.csdn.net/l2show/article/details/48653949) [copy](http://blog.csdn.net/l2show/article/details/48653949)

1. dependencies {
2. compile 'cn.jesse.android:datetimepicker:1.0.2'
3. }

## 使用Nexus Respository搭建属于自己公司的私有maven服务器

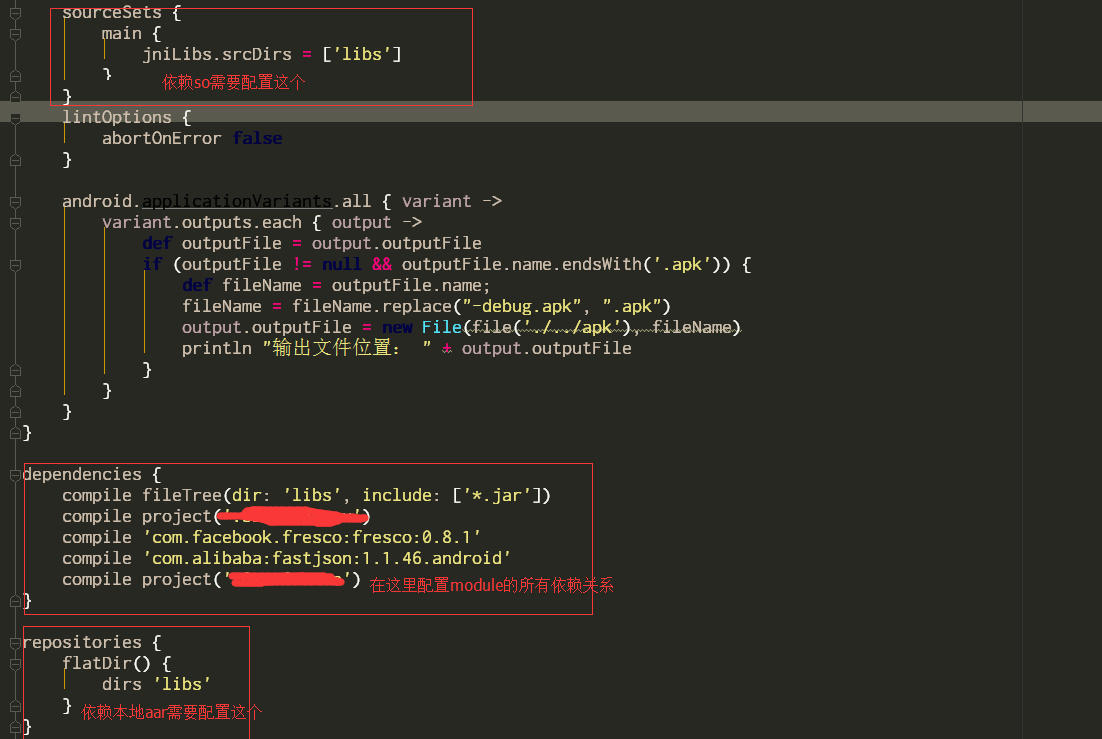
## 前言

  在Android应用开发过程中，不同IDE对工程的依赖方式不一样：

  使用Eclipse开发时，项目之间的依赖关系是这样的：一个主工程(project)可以依赖多个libproject、so、jar包，对jar包和so的依赖是直接将jar和so放在工程的libs文件夹下(老版本的ADT需要手动配置Build Path)，对libproject的依赖呈现在工程的“project.properties”文件中。



  使用Android Studio开发时，除了可以依赖module（对应Eclipse中的libproject）、jar和so，还可以依赖aar（aar和jar包不同之处在于可以将so和资源文件一起打包），as的依赖关系全部（jar、so、aar、libproject）在build.gradle文件中的android标签中管理。



  使用AS开发应用时，除了可以依赖本地的库之外，还可以依赖网上（公有maven服务器、私有maven服务器、jcenter等），如果是依赖本地的，必须要将依赖的module和主工程放在一个project里面，这就导致了每个project都需要配置这些依赖关系，如果是公司内多个工程依赖同一个公司内部的控件，控件有更新时，同步非常麻烦，但公司内部的控件不可能部署到公有maven服务器上，所以有必要搭建一个局域网内的maven服务器，方便管理公司内部的公共库。

## maven私有服务器搭建

  搭建maven私服使用得比较多的是Nexus，Nexus是基于maven仓库管理的社区项目，主要的使用场景就是可以在局域网搭建一个maven私服,用来部署第三方公共构件或者作为远程仓库在该局域网的一个代理。

  关于Nexus的介绍和配置很简单，具体可以查看这里：[Android 项目部署之Nexus私服搭建和应用](http://blog.csdn.net/l2show/article/details/48653949)。

## 上传库到私服

  上传库到私服有两种方式，一种是库中配置，配置完成后执行upload这个task，另外一种方式是直接上传。下面分别对这两种方式做介绍：

  在库中配置，步骤如下：

  1. 在project下的gradle.properties文件中定义通用属性，方便如果有多个库需要部署时，不需要修改每一个库中的配置：

#本地库

URLMAVEN\_URL= http://172.28.1.\*:8081/nexus/content/repositories/thirdparty/

MAVEN\_SNAPSHOT\_URL = http://172.28.1.\*:8081/nexus/content/repositories/thirdparty-snapshot/

#对应maven的groupId值

GROUP=common

#登录nexus oss的用户名

NEXUS\_USERNAME=admin

#登录nexus oss的密码

NEXUS\_PASSWORD=admin123

# groupid

GROUP\_ID = common

# type

TYPE = aar

# description

DESCRIPTION = dependences lib

  2. 修改module对应的build.gradle文件，配置以谁的名义上传这个库，上传到什么地方，这个库叫什么名字，属于哪个group，ID和version、description、packageing等信息

apply plugin: 'com.android.library'

apply plugin: 'maven'

android {

compileSdkVersion 17

buildToolsVersion "23.0.2"

defaultConfig {

minSdkVersion 15

targetSdkVersion 17

versionCode 1

versionName "1.0"

}

buildTypes {

release {

minifyEnabled **false**

proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'

}

}

sourceSets {

main {

jniLibs.srcDirs = ['libs']

}

}

lintOptions {

abortOnError **false**

}

}

dependencies {

compile fileTree(dir: 'libs', include: ['\*.jar'])

}

uploadArchives {

configuration = configurations.archives

repositories {

mavenDeployer {

snapshotRepository(url: MAVEN\_SNAPSHOT\_URL) {

authentication(userName: NEXUS\_USERNAME, password: NEXUS\_PASSWORD)

}

repository(url: MAVEN\_URL) {

authentication(userName: NEXUS\_USERNAME, password: NEXUS\_PASSWORD)

}

pom.project {

version '1.0.0'

artifactId 'TestLibrary'

groupId GROUP\_ID

packaging TYPE

description DESCRIPTION

}

}

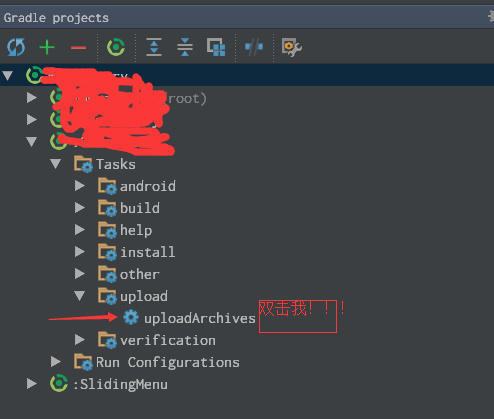
}

}

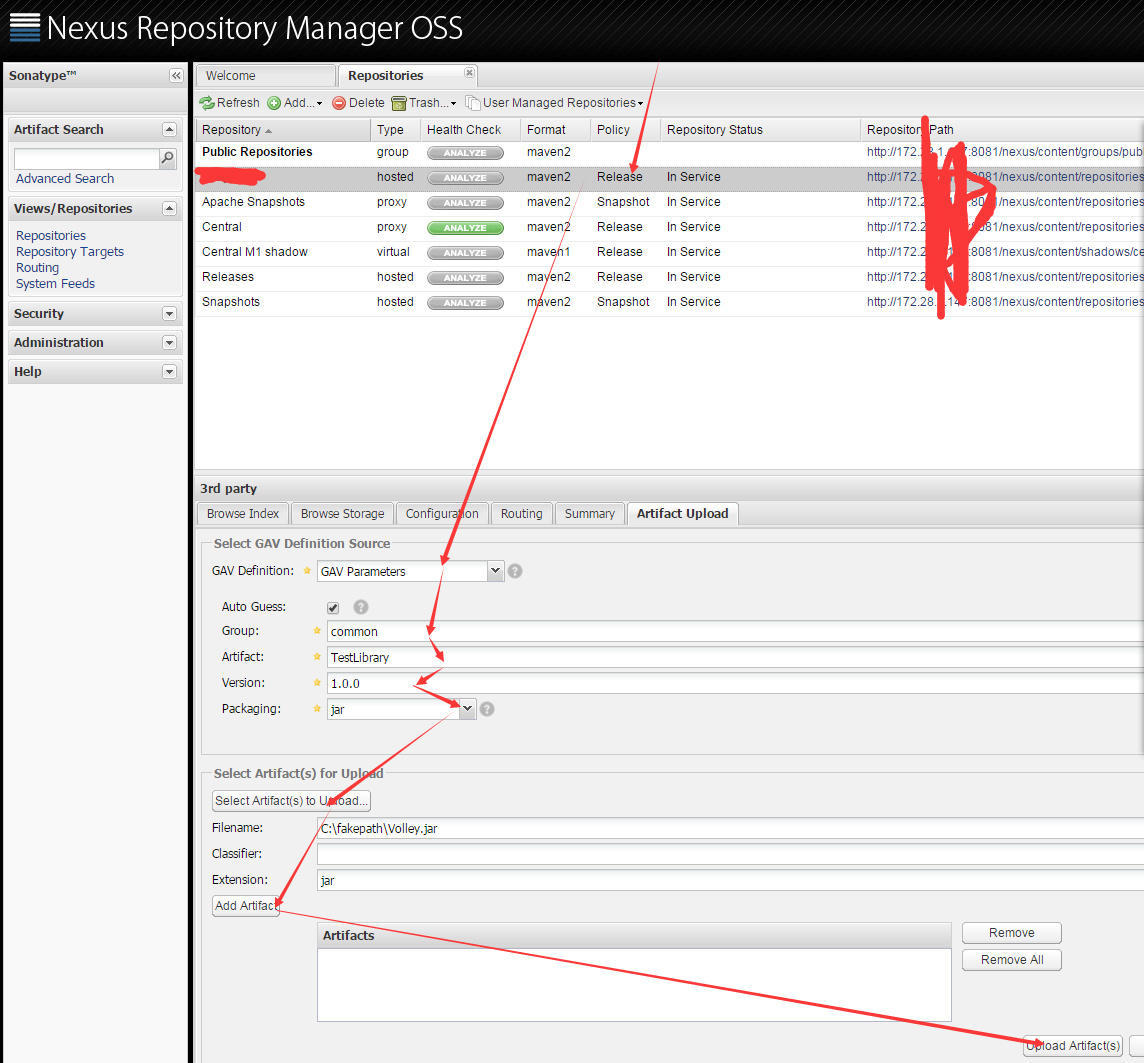
artifacts {

archives file('TestLibrary.aar') }

  3. 打开Gradle projects（在AS的右边栏），找到对应的module，展开，找到Tasks下面的upload标签并双击，在Gradle Console标签可以查看是否上传成功。



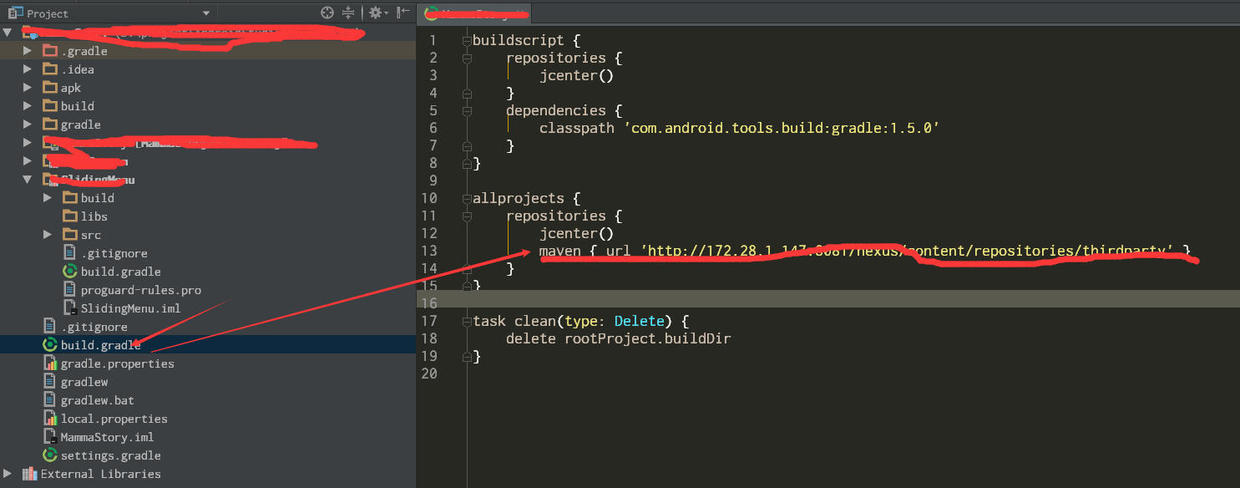
  直接上传：直接上传很简单，直接按照下图的箭头操作即可，如果上传aar还没有研究过，有兴趣的可以自己研究一下：



## 在项目中使用私有仓库

  在项目中使用私有仓库的步骤如下：

  1. 在project的build.gradle文件中指定私有仓库的地址，like this：



  2. 在需要依赖私有仓库的build.gradle文件中设置依赖关系：

