

# 一：NDK的安装

首先是安装NDK，安装之前需要安装CDT，具体的参照<http://blog.csdn.net/yanzi1225627/article/details/7736364>这里来完成。这里再详细记录下NDK的安装过程。google下载NDK，也可以点击[这里](#)下载：  
<http://download.csdn.NET/detail/yanzi1225627/5015893>，下载后输入tar -jxvf android-ndk-r8-Linux-x86.tar.bz2 -C /usr/local/android/将其解压到/usr/local/android/ 目录。

『注，我的android相关文件都安装在这里。这个目录不是死的。』然后gedit /etc/profile,在里面添加：export PATH=\$PATH:/usr/local/android/android-ndk-r8，从安装JDK到eclipse, android, NDK,在/etc/profile文件里，添加的命令一共有如下三条：  
export JAVA\_HOME=/usr/local/android/jdk1.7.0\_04  
export PATH=\$PATH:\$JAVA\_HOME/bin  
export PATH=\$PATH:/usr/local/android/android-ndk-r8

只要这三条就行了。然后source /etc/profile 使刚才的设置生效。  
在终端里输入：ndk-build，可以测试出ndk安装成功了么有。

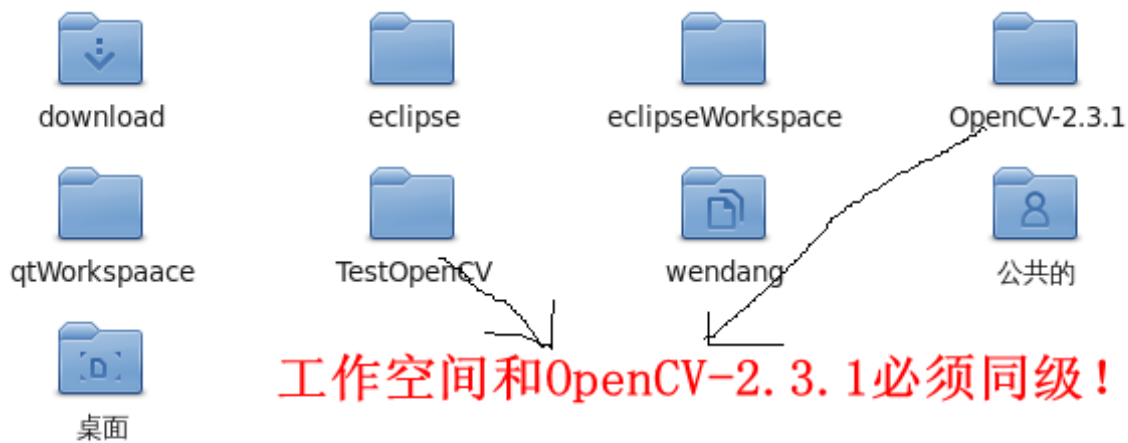
注：有的时候也可以通过在线安装方式，在eclipse里点help-install new software，网址输入这个：ADT - <https://dl-ssl.google.com/android/eclipse/>，也会看到安装NDK Plugins, 如果经过上面的步骤，NDK不能用就把这个也装上。但如果只装这个，好像不中，找不到ndk安装的目录，也无法配置路径。

## 二：OpenCV的移植

这里的移植不是像在qt下那么麻烦，这是因为OpenCV退出来直接支持Android的版本，所以不需要自己编译。直接下下来，解压缩就可以了。我用的OpenCV2.3.1,需要的可以直接到csdn资源里下载，链

接：<http://download.csdn.net/detail/yanzi1225627/5013701>，也可以自己到sourceforge上下载，链接：<http://sourceforge.net/projects/opencvlibrary/files/opencv-android/>，从这里可以看出从OpenCV2.3开始就有编译好的android版本。最新的是2.4.3版本，在2012年12月24发布的。牛逼阿！

接下来就是配置。事实上有两种方法在Android里调用OpenCV，一种是使用OpenCVJava Api，一种是通过JNI的方式。这里是针对后者。将OpenCV-2.3.1-android-bin.tar.bz2解压缩，然后将里面的OpenCV-2.3.1拷贝到Eclipse工作空间的平级目录。图示：



为此，我们先建一个工作空间。新建文件夹/home/yan/TestOpenCV，点eclipse里的File---Switch workspace---other，选中这个目录。切换到这个工作空间后，点Window---Preference--android，选中自己的android-sdk的安装目录，我的  
是：/usr/local/android/android-sdk-linux。然后就可以在这个工作空间里正常android开发了，如果不设置这个，新建的工程全是红叉叉。

新建一个项目HaveImgFun，包名是package com.testopencv.haveimgfun; 然后将刚才解压缩出来的**OpenCV-2.3.1-android-bin\samples**下的**includeOpenCV.mk**文件拷贝到和项目**HaveImgFun**同一级目录中。图示：



在 eclipse里选中那个项目，新建一个文件夹jni，然后新建文件：Android.mk,里面的内容是：

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
include ../includeOpenCV.mk
ifeq ("$(wildcard $(OPENCV_MK_PATH))", "")
#try to load OpenCV.mk from default install location
include $(TOOLCHAIN_PREBUILT_ROOT)/user/share/OpenCV/OpenCV.mk
else
include $(OPENCV_MK_PATH)
endif
LOCAL_MODULE := ImgFun
```

```
LOCAL_SRC_FILES := ImgFun.cpp
include $(BUILD_SHARED_LIBRARY)
```

再新建一个Application.mk, 内容输入:

```
APP_STL:=gnustl_static
APP_CPPFLAGS:=-frtti -fexceptions
APP_ABI:=armeabi armeabi-v7a
```

然后新建一个cpp文件, ImgFun.cpp, 这个测试程序是将一个图片的上半部分弄黑, 然后复原。具体的大家看源码。功能很简单。

内容是:

```
#include <jni.h>
#include <stdio.h>
#include <stdlib.h>
#include <opencv2/opencv.hpp>
using namespace cv;
extern "C" {
JNIEXPORT jintArray JNICALL
Java_com_example_haveimgfun2_LibImgFun_ImgFun(
    JNIEnv* env, jobject obj, jintArray buf, int w, int h);
JNIEXPORT jintArray JNICALL
Java_com_testopencv_haveimgfun_LibImgFun_ImgFun(
    JNIEnv* env, jobject obj, jintArray buf, int w, int h){
    jint *cbuf;
    cbuf = env->GetIntArrayElements(buf, false);
    if(cbuf == NULL)
    {
        return 0;
    }
    Mat myimg(h, w, CV_8UC4, (unsigned char*)cbuf);
    for(int j=0; j<myimg.rows/2; j++)
    {
        myimg.row(j).setTo(Scalar(0, 0, 0, 0));
    }
    int size=w*h;
    jintArray result = env->NewIntArray(size);
    env->SetIntArrayRegion(result, 0, size, cbuf);
    env->ReleaseIntArrayElements(buf, cbuf, 0);
    return result;
}
```

}



## jni文件夹下的三个文件

然后在终端里切换到HavelmgFun目录，也就是在目录/home/yan/TestOpenCV/HavelmgFun下，终端输入**ndk-build**，会生成相应的库。具体的大家下载源程序把！下载连

接：<http://download.csdn.net/detail/yanzi1225627/5016365>

【注，下载后将文件解压。然后eclipse里，切换到TestOpenCV空间，点File--import，就可以了。首先测下最后一步，也就是输入ndk-build, 会不会生成相应的库！】

参考：<http://www.cnblogs.com/ldr213/archive/2012/02/20/2359262.html>

来源：<http://blog.csdn.net/yanzi1225627/article/details/8525720>

OpenCV4Android释疑: 透析Android以JNI调OpenCV的三种方式(让OpenCVManager永不困扰)

前文曾详细探讨了关于**OpenCV**的使用，原本以为天下已太平。但不断有人反应依然配不好**OpenCV4Android**，不能得心应手的在**Android**上使用**OpenCV**，大量的精力都浪费在摸索配置上。尤其是**OpenCVManager**诞生之后，更让人无语，大家第一个反应就是如何才能不安装**OpenCVManager**，因为要多安装这个东西对客户来说体验太不好了。咱家昨夜研究至两点，今早七点起床，终于把头绪理清了。下面咱家以之前做过的一个基于**OpenCV2.3.1**，**android**通过**jni**调用**opencv**实现人脸检测的实例来逐个回答，如何在**Android**上使用**Java**接口而不安装**OpenCVManager**，及通过**jni**方式使用**OpenCV**的三种方式。

---

先来看**JNI**调**OpenCV**的三种方式。很多人会吃惊肿么**JNI**调**OpenCV**还会有3种方式，长久以来大量网上教程都说在**Android**上只有**Java**和**JNI**两种方式使用**OpenCV**，怎么又冒出来3种使用**JNI**的方式。经本人研究，确实有3种调**JNI**的方式，就连官网指导文档都模棱两可，所以让很多人不知所措。这三种方式分别是：

- 1、使用静态的**OpenCV**库的方式；
- 2、使用动态的**OpenCV**库的方式；
- 3、同时使用**Java**的**API**又使用**JNI**的接口的方式，此时编译时一般使用的是动态链接**OpenCV**库的方式。

要说明的是，这三种方式均**无需安装OpenCVManager**，区别在于**mk**文件的不同。个人最推崇的就是第一种方式，第一种方式也是和**OpenCV2.3.1**在**JNI**调**OpenCV**使用完全吻合的一种方式。本文是以**windows**平台最新的**OpenCV-2.4.9-android-sdk**为基础，使用**2.4.9**的**OpenCV4Android**需要使用**NDK**版本为**r9**，本人使用的是**android-ndk-r9d**的版本。之所以昨晚捣腾到2点，就是因为之前使用的**ndk r7**的版本，怎么编都编不过，因少东西报上千行错误。**android-ndk-r9d**安装十分简单，只需要解压缩配置一个环境变量即可。

一、**Android**以**JNI**调**OpenCV**的第一种配置方法：

**Application.mk**文件里的内容如下：

```
APP_STL:=gnustl_static
APP_CPPFLAGS:=-frtti -fexceptions
APP_ABI:= armeabi-v7a
```

这三种方式的**Application.mk**都一样，所以往后不说了。在**Application.mk**里还可以配置**APP\_PLATFORM=17**类似这种，当然不配置完全可以。

**Android.mk**内容如下：

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
```

```
OpenCV_INSTALL_MODULES:=on
```

```
OPENCV_CAMERA_MODULES:=off
```

```
OPENCV_LIB_TYPE:=STATIC
```

```
ifeq ("$(wildcard $(OPENCV_MK_PATH))", "")
include D:\ProgramFile\OpenCV-2.4.9-android-sdk\sdk\native\jni\OpenCV.mk
else
include $(OPENCV_MK_PATH)
endif
```

```
LOCAL_MODULE := ProcessImg
LOCAL_SRC_FILES := DetectFace_JNI.cpp \
                  src/copyToAssets.cpp \
                  src/detectFace.cpp
LOCAL_LDLIBS += -lm -llog
```

```
include $(BUILD_SHARED_LIBRARY)
```

逐一解释下，`OpenCV_INSTALL_MODULES:=on`的意思是自动将依赖的OpenCV的so库拷贝到libs目录下，但很遗憾的是，这个命令只对`OPENCV_CAMERA_MODULES`有效。只有当`OPENCV_CAMERA_MODULES:=on`时，可以看到他会自动将里面的带camera的so拷贝至工程下的libs文件夹下。`include D:\ProgramFile\OpenCV-2.4.9-android-sdk\sdk\native\jni\OpenCV.mk`这句话比较关键，这是我安装OpenCV-2.4.9-android-sdk的地方，我将其安装到了D盘。而我的工作空间在E盘也是ok的。而不用像OpenCV2.3.1使用时，限制这个解压缩包的位置了。`LOCAL_MODULE`是要生成的库的名字，`LOCAL_SRC_FILES`是jni文件夹下的cpp文件，其中的src说明我的jni下还有个文件夹名字是“src”，这块替换成自己的源码文件就ok了。

为了测试的严谨性，在工程里将libs文件夹的东西，和obj文件夹下的东西全删了。用cygwin进到工程，输入ndk-build，看到如下信息：

[plain] view plain copy print ?

```
01. <span style="font-family: 'Comic Sans MS';"><span style="font-
02. size:18px;">Administrator@yanzi /cygdrive/e/WorkSpaces/OpenCV4Android/FaceDetectLiu2
03. $ ndk-build
04. Android NDK: WARNING: APP_PLATFORM android-
    19 is larger than android:minSdkVersion 16 in ./AndroidManifest.xml
    [armeabi-v7a] Cygwin : Generating dependency file converter script
```

```

05. [armeabi-v7a] Compile++ thumb: ProcessImg <= DetectFace_JNI.cpp
06. jni/DetectFace_JNI.cpp: In function '__jstring* Java_org_yan_processlib_LibProcessImg_processIplImg
07. jni/DetectFace_JNI.cpp:99:44: warning: converting 'false' to pointer type for argument 2 of 'jint*
Wconversion-null]
08. jni/DetectFace_JNI.cpp: In function '__jstring* Java_org_yan_processlib_LibProcessImg_processStatic
09. jni/DetectFace_JNI.cpp:133:44: warning: converting 'false' to pointer type for argument 2 of 'jint
Wconversion-null]
10. [armeabi-v7a] Compile++ thumb: ProcessImg <= copyToAssets.cpp
11. [armeabi-v7a] Compile++ thumb: ProcessImg <= detectFace.cpp
12. [armeabi-v7a] SharedLibrary : libProcessImg.so
13. [armeabi-v7a] Install : libProcessImg.so => libs/armeabi-v7a/libProcessImg.so
14. </span></span>

```

上面两个警告么有关系，编译成功。生成的libProcessImg.so的大小为4M,整个apk大小为1.99M。

注意，如果将mk里的LOCAL\_LDLIBS += -lm -llog这一句错误的写为:LOCAL\_LDLIBS := -lm -llog，即将“+=”错写成了“:=”将会看到如下大量错误：

```

[plain] view plain copy print ? C
01. <span style="font-family: 'Comic Sans MS';"><span style="font-size:18px;">$ ndk-build
02. Android NDK: WARNING: APP_PLATFORM android-
19 is larger than android:minSdkVersion 16 in ./AndroidManifest.xml
03. [armeabi-v7a] Compile++ thumb: ProcessImg <= DetectFace_JNI.cpp
04. jni/DetectFace_JNI.cpp: In function '__jstring* Java_org_yan_processlib_LibProcessImg_processIplImg
05. jni/DetectFace_JNI.cpp:99:44: warning: converting 'false' to pointer type for argument 2 of 'jint*
Wconversion-null]
06. jni/DetectFace_JNI.cpp: In function '__jstring* Java_org_yan_processlib_LibProcessImg_processStatic
07. jni/DetectFace_JNI.cpp:133:44: warning: converting 'false' to pointer type for argument 2 of 'jint
Wconversion-null]
08. [armeabi-v7a] Compile++ thumb: ProcessImg <= copyToAssets.cpp
09. [armeabi-v7a] Compile++ thumb: ProcessImg <= detectFace.cpp
10. [armeabi-v7a] SharedLibrary : libProcessImg.so
11. D:/ProgramFile/android-ndk-r9d/toolchains/arm-linux-androideabi-
4.6/prebuilt/windows/bin/./lib/gcc/arm-linux-androideabi/4.6/../../../../arm-linux-
androideabi/bin/ld.exe: D:/ProgramFile\OpenCV-2.4.9-android-sdk\sdk\native\jni\./libs/armeabi-
v7a/libopencv_core.a(persistence.cpp.o): in function icvGets(CvFileStorage*, char*, int):persister
12. D:/ProgramFile/android-ndk-r9d/toolchains/arm-linux-androideabi-
4.6/prebuilt/windows/bin/./lib/gcc/arm-linux-androideabi/4.6/../../../../arm-linux-
androideabi/bin/ld.exe: D:/ProgramFile\OpenCV-2.4.9-android-sdk\sdk\native\jni\./libs/armeabi-
v7a/libopencv_core.a(persistence.cpp.o): in function icvXMLSkipSpaces(CvFileStorage*, char*, int):
13. D:/ProgramFile/android-ndk-r9d/toolchains/arm-linux-androideabi-
4.6/prebuilt/windows/bin/./lib/gcc/arm-linux-androideabi/4.6/../../../../arm-linux-
androideabi/bin/ld.exe: D:/ProgramFile\OpenCV-2.4.9-android-sdk\sdk\native\jni\./libs/armeabi-
v7a/libopencv_core.a(persistence.cpp.o): in function icvYMLSkipSpaces(CvFileStorage*, char*, int,
14. D:/ProgramFile/android-ndk-r9d/toolchains/arm-linux-androideabi-
4.6/prebuilt/windows/bin/./lib/gcc/arm-linux-androideabi/4.6/../../../../arm-linux-
androideabi/bin/ld.exe: D:/ProgramFile\OpenCV-2.4.9-android-sdk\sdk\native\jni\./libs/armeabi-
v7a/libopencv_core.a(persistence.cpp.o): in function icvPuts(CvFileStorage*, char const*):persiste
15. D:/ProgramFile/android-ndk-r9d/toolchains/arm-linux-androideabi-
4.6/prebuilt/windows/bin/./lib/gcc/arm-linux-androideabi/4.6/../../../../arm-linux-
androideabi/bin/ld.exe: D:/ProgramFile\OpenCV-2.4.9-android-sdk\sdk\native\jni\./libs/armeabi-
v7a/libopencv_core.a(persistence.cpp.o): in function icvClose(CvFileStorage*, std::string*):persis

```



```
16. D:/ProgramFile/android-ndk-r9d/toolchains/arm-linux-androideabi-  
4.6/prebuilt/windows/bin/./lib/gcc/arm-linux-androideabi/4.6/./././././arm-linux-  
androideabi/bin/ld.exe: D:/ProgramFile\OpenCV-2.4.9-android-sdk\sdk\native\jni\./libs/armeabi-  
v7a/libopencv_core.a(persistence.cpp.o): in function cvOpenFileStorage:persistence.cpp(.text.cvOpenFileStorage):  
17. D:/ProgramFile/android-ndk-r9d/toolchains/arm-linux-androideabi-  
4.6/prebuilt/windows/bin/./lib/gcc/arm-linux-androideabi/4.6/./././././arm-linux-  
androideabi/bin/ld.exe: D:/ProgramFile\OpenCV-2.4.9-android-sdk\sdk\native\jni\./libs/armeabi-  
v7a/libopencv_core.a(persistence.cpp.o): in function cvOpenFileStorage:persistence.cpp(.text.cvOpenFileStorage):  
18. D:/ProgramFile/android-ndk-r9d/toolchains/arm-linux-androideabi-  
4.6/prebuilt/windows/bin/./lib/gcc/arm-linux-androideabi/4.6/./././././arm-linux-  
androideabi/bin/ld.exe: D:/ProgramFile\OpenCV-2.4.9-android-sdk\sdk\native\jni\./libs/armeabi-  
v7a/libopencv_core.a(persistence.cpp.o): in function cvOpenFileStorage:persistence.cpp(.text.cvOpenFileStorage):  
19. D:/ProgramFile/android-ndk-r9d/toolchains/arm-linux-androideabi-  
4.6/prebuilt/windows/bin/./lib/gcc/arm-linux-androideabi/4.6/./././././arm-linux-  
androideabi/bin/ld.exe: D:/ProgramFile\OpenCV-2.4.9-android-sdk\sdk\native\jni\./libs/armeabi-  
v7a/libopencv_core.a(persistence.cpp.o): in function cvOpenFileStorage:persistence.cpp(.text.cvOpenFileStorage):  
20. D:/ProgramFile/android-ndk-r9d/toolchains/arm-linux-androideabi-  
4.6/prebuilt/windows/bin/./lib/gcc/arm-linux-androideabi/4.6/./././././arm-linux-  
androideabi/bin/ld.exe: D:/ProgramFile\OpenCV-2.4.9-android-  
sdk\sdk\native\jni\./3rdparty/libs/armeabi-  
v7a/liblibpng.a(pngread.c.o): in function png_create_read_struct_2:pngread.c(.text.png_create_read_struct_2):  
21. D:/ProgramFile/android-ndk-r9d/toolchains/arm-linux-androideabi-  
4.6/prebuilt/windows/bin/./lib/gcc/arm-linux-androideabi/4.6/./././././arm-linux-  
androideabi/bin/ld.exe: D:/ProgramFile\OpenCV-2.4.9-android-  
sdk\sdk\native\jni\./3rdparty/libs/armeabi-  
v7a/liblibpng.a(pngread.c.o): in function png_read_row:pngread.c(.text.png_read_row+0x218): error: undefined reference to `png_read_row`  
22. D:/ProgramFile/android-ndk-r9d/toolchains/arm-linux-androideabi-  
4.6/prebuilt/windows/bin/./lib/gcc/arm-linux-androideabi/4.6/./././././arm-linux-  
androideabi/bin/ld.exe: D:/ProgramFile\OpenCV-2.4.9-android-  
sdk\sdk\native\jni\./3rdparty/libs/armeabi-  
v7a/liblibpng.a(pngread.c.o): in function png_read_destroy:pngread.c(.text.png_read_destroy+0x96): error: undefined reference to `png_read_destroy`  
23. D:/ProgramFile/android-ndk-r9d/toolchains/arm-linux-androideabi-  
4.6/prebuilt/windows/bin/./lib/gcc/arm-linux-androideabi/4.6/./././././arm-linux-  
androideabi/bin/ld.exe: D:/ProgramFile\OpenCV-2.4.9-android-  
sdk\sdk\native\jni\./3rdparty/libs/armeabi-  
v7a/liblibpng.a(pngwrite.c.o): in function png_write_flush:pngwrite.c(.text.png_write_flush+0x1c): error: undefined reference to `png_write_flush`  
24. </span></span>
```

上两张运行效果图,分别是预览界面检测人脸和拍照后检测:





<http://blog.csdn.net/yanzi1225627>



切换至图库



## 二、**Android**以**JNI**调**OpenCV**的第二种配置方法

**Application.mk**文件同上，**Android.mk**文件如下：

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
```

```
OpenCV_INSTALL_MODULES:=on
OPENCV_CAMERA_MODULES:=off
```

```
OPENCV_LIB_TYPE:=SHARE
```

```
ifeq ("$(wildcard $(OPENCV_MK_PATH))", "")
```

```
include D:\ProgramFile\OpenCV-2.4.9-android-sdk\sdk\native\jni\OpenCV.mk
else
```

```
include $(OPENCV_MK_PATH)
endif
```

```
LOCAL_MODULE := ProcessImg
LOCAL_SRC_FILES := DetectFace_JNI.cpp \
                  src/copyToAssets.cpp \
                  src/detectFace.cpp
LOCAL_LDLIBS := -lm -llog
```

```
include $(BUILD_SHARED_LIBRARY)
```

唯一的变化时将`OPENCV_LIB_TYPE:=STATIC`变成了`SHARE`.即通过动态链接的方式连接OpenCV的so。编译信息如下:

```
[plain] view plain copy print ? C
01. <span style="font-
02. size:18px;">Administrator@yanzi /cygdrive/e/WorkSpaces/OpenCV4Android/FaceDetectLiu2
03. $ ndk-build
04. Android NDK: WARNING: APP_PLATFORM android-
19 is larger than android:minSdkVersi
05. Android NDK: WARNING:jni/Android.mk:ProcessImg: non-
system libraries in linker f
lopencv_java
06. Android NDK: This is likely to result in incorrect builds. Try using LOCAL_S
07. Android NDK: or LOCAL_SHARED_LIBRARIES instead to list the library dependenc
08. Android NDK: current module
09. [armeabi-v7a] Compile++ thumb: ProcessImg <= DetectFace_JNI.cpp
10. jni/DetectFace_JNI.cpp: In function '__jstring* Java_org_yan_processlib_LibProces
jni/DetectFace_JNI.cpp:99:44: warning: converting 'false' to pointer type for ar
Wconver
null]
11. jni/DetectFace_JNI.cpp: In function '__jstring* Java_org_yan_processlib_LibProces
12. jni/DetectFace_JNI.cpp:133:44: warning: converting 'false' to pointer type for a
Wconve
null]
13. [armeabi-v7a] Compile++ thumb: ProcessImg <= copyToAssets.cpp
14. [armeabi-v7a] Compile++ thumb: ProcessImg <= detectFace.cpp
15. [armeabi-v7a] SharedLibrary : libProcessImg.so
16. D:/ProgramFile/android-ndk-r9d/toolchains/arm-linux-androideabi-
4.6/prebuilt/windows/bin/./lib/gcc/arm-linux-androideabi/4.6/../../../../arm-linux-
androideabi/bin/ld.exe: warning: hidden symbol '__aeabi_atexit' in D:/ProgramFile/android-ndk-
r9d/sources/cxx-stl/gnu-libstdc++/4.6/libs/armeabi-
v7a/libgnustl_static.a(atexit_arm.o) is referenced by DSO D:/ProgramFile/OpenCV-2.4.9-android-
sdk\sdk\native\jni\./libs/armeabi-v7a/libopencv_java.so
17. [armeabi-v7a] Install : libProcessImg.so => libs/armeabi-v7a/libProcessImg.so
18. </span>
```

可以看到上面说找不到non-system libraries in linker flags: -lopencv\_java这个东西，关于这个问题我曾作如下尝试：

LOCAL\_LDLIBS += -lopencv\_java 或 LOCAL\_SHARED\_LIBRARIES += libopencv\_java均没有解决这个warning。原本运行正常的程序报错如下：

java.lang.UnsatisfiedLinkError: Cannot load library:  
soinfo\_link\_image(linker.cpp:1635): could not load library "libopencv\_java.so"  
needed by "libProcessImg.so"; caused by load\_library(linker.cpp:745): library  
"libopencv\_java.so" not found

说是自己编译的这个库libProcessImg.so依赖于libopencv\_java.so，没有找到它所以程序挂了。再看生成的libProcessImg.so大小为437KB，比第一种方式少了好几倍啊。肿么让程序正常运行呢？将安装目录D:\ProgramFile\OpenCV-2.4.9-android-sdk\sdk\native\libs\armeabi-v7a下的libopencv\_java.so拷贝到libs\armeabi-v7a文件夹下，然后再调用库的时候方法变更为：

```
[java] view plain copy print ? C
01. <span style="font-size:18px;">package org.yan.processlib;
02. public class LibProcessImg {
03.     static{
04.         System.loadLibrary("opencv_java");
05.         System.loadLibrary("ProcessImg");
06.     }
07.
08.     public static native void initProcessLib(String str);
09.     public static native String processIplImg(int[] buf, int w, int h);
10.     public static native String processStaticImg(int[] buf, int w, int h);
11. }
12. </span>
```

先调用这个依赖的库，然后调用我们自己的，注意这个libopencv\_java.so有9M多。如此程序又可以正常运行了。最终apk的大小为4.83M，是第一种2倍，但还没大的离谱，可以接受。

对比上面两种方法不难发现，虽然在mk里都有include \$(BUILD\_SHARED\_LIBRARY) 也即生成的库都是动态库，但这个库指的是我们自己写的，我们的库要进一步调用OpenCV的库，否则的话直接就能用OpenCV库里的函数，咋可能有这事呢。至于怎么调OpenCV的库，可以静态，也可以动态。这也就是为什么第二种方法生成的so的大小只有437KB，而第一种方法生成的库有4M的大小。事实上在我们第一种方式ndk-build的时候会发现有大量的各种.a .a被加载进去的情形，只不过这只出现一次，原因就在这。打开安装目录下的libs：

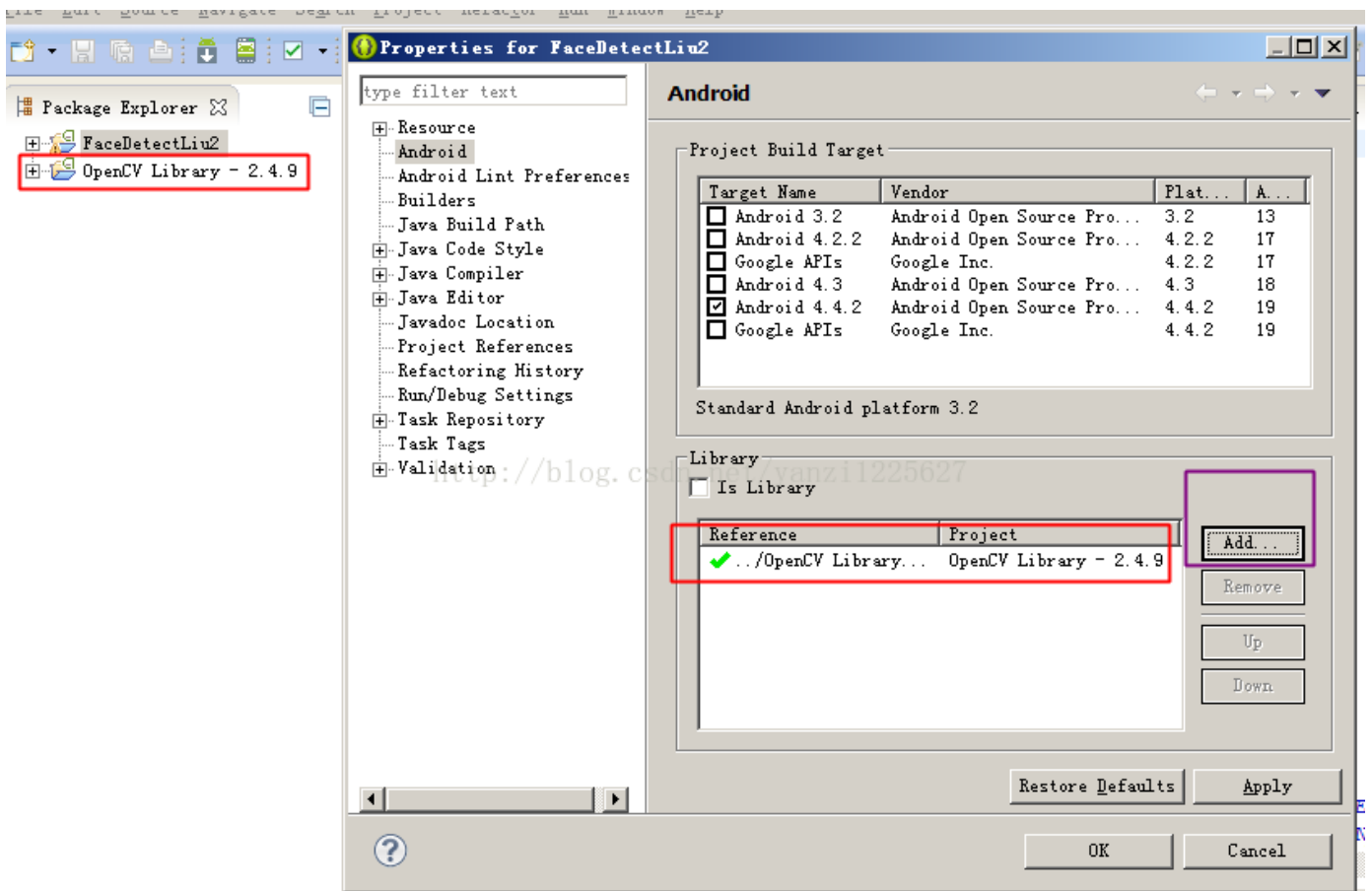


可以看到除了带Camera的so外，其他大量都是.a，而且这些.a是按包名划分的。而so只有libopencv\_java.so和libopencv\_info.so，在功能上这些.a静态调用时等同于动态加载这两个so。之所以这么多.a就是供我们采用第一种方法时使用的。关于静态和动态的优缺点参见[这里](#)

### 第三种方法：java和jni混用

搞完第二种，既然动态加载我表面上没用libopencv\_java.so，还要把它加载进来，那我干脆为啥不用用java的api呢？既然要用java的api那肯定要jar包弄进来，于是导进来OpenCV Library - 2.4.9工程如下图所示：





这样就可以用java的api了。如果你有强迫症，觉得OpenCV Library - 2.4.9这个工程一直开着心里不爽，那也可以将sdk bin目录生成的opencv library - 2.4.9.jar拷贝到自己工程的libs文件夹下，记得将刚才添加的Libraries remove掉。右键opencv library - 2.4.9.jar-----build path-----add to build path，这样照样使用Java的api。其实这块很明显，只要jar包弄进去了你就可以正常使用api了，即编译时不报错，但apk到手机上能不能正常运行则是另一码子事。

此时的mk文件跟第二种类似，记得把libopencv\_java.so拷贝到相应目录。相较于第二种，并没有增加什么，仅仅是开发时将jar包导入就可以正常编译了，能否正常运行还依赖于libopencv\_java.so。需注意的是，每clean一次，这个libopencv\_java.so就会不见一次，还要手动拷或者自己写个脚本拷。最终apk的大小为4.94M，相比第二种多点，原因是那个jar包的原因，以及我们代码里又加了几句：

[java] view plain copy print ?

```

01. <span style="font-size:18px;">package org.yan.processlib;
02.
03. import org.opencv.android.OpenCVLoader;
04.
05. import android.util.Log;
06.
07. public class LibProcessImg {
08.     static{
09.         if(!OpenCVLoader.initDebug()){
10.             Log.i("yanzi", "OpenCVLoader.initDebug() 失败");

```

```

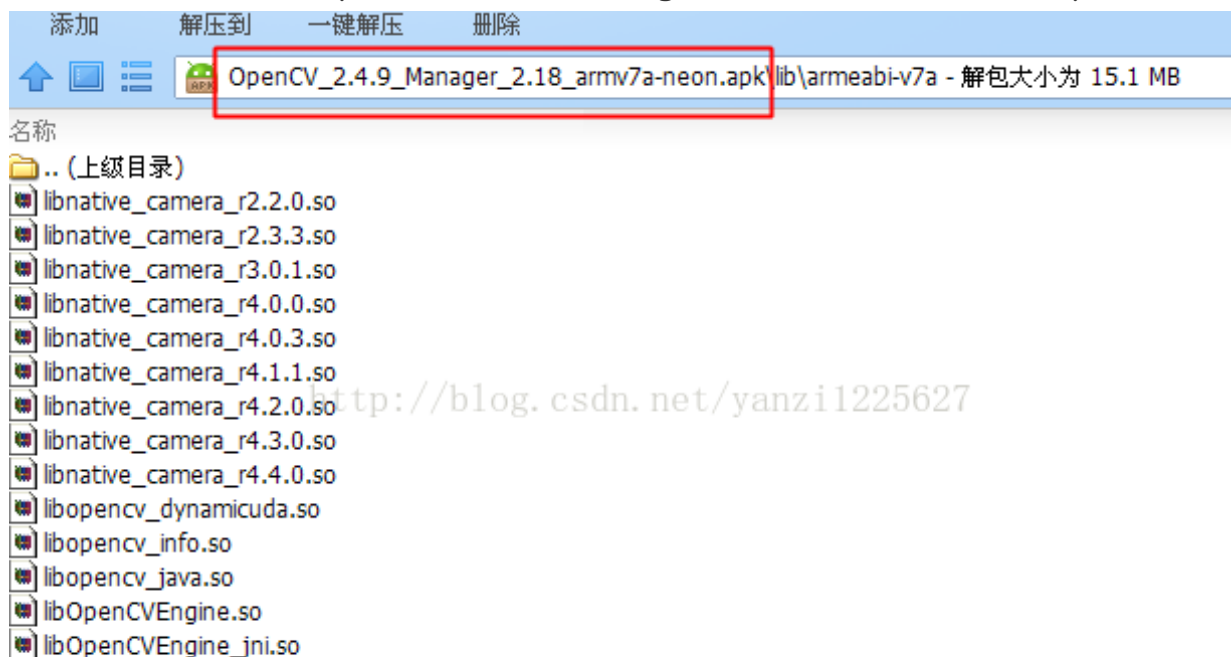
11.         }else{
12.             System.loadLibrary("opencv_java");
13.             System.loadLibrary("ProcessImg");
14.         }
15.     }
16.
17.     public static native void initProcessLib(String str);
18.     public static native String processIplImg(int[] buf, int w, int h);
19.     public static native String processStaticImg(int[] buf, int w, int h);
20. }
21. </span>

```

当然你可以加其他的很多OpenCV的java接口，比如Bitmap转mat，直接传Mat指针到jni等等，随便自己怎么玩。官网上的JNI使用OpenCV其实就是这种java和jni混用的情况，其实大多情况下么有啥必要，看个人了。至于动态加载OpenCV的库还是静态，也全看个人，我是倾向于第一种，以apk的体积最清爽为准。

我们用initDebug一下，其实这块你不写也行的。另外就是这个加载库用static方法跟放Activity里的onResume里差不多，我是习惯了放单独的一个静态方法里。记住千万不要用OpenCVLoader.initAsync()方法啊，本文的主线就是不用OpenCVManager!!!

最后我们打开一个OpenCV\_2.4.9\_Manager\_2.18\_armv7a-neon.apk来看一下：



哈哈，看到了吧，里面的精髓就是lib下的so以及那个引擎so。在使用OpenCVManager的情况下，这些库随着OpenCVManager.apk的装入都事先安装到手机了，不论是使用java也好，还是用jni再使用动态链接OpenCV库的方法(使自己的so体积最小)，都不用



往libs文件夹额外加so了，因为so随着OpenCVManager已安好了。这就是之所以加个OpenCVManager的半个初衷啊，另半个初衷是binder service 框架上的原因！！！  
最后补充3点：

1.有时Cygwin会有记忆效应，比如你修改了mk里从static变成share，但是它还是按照static来编译的。解决方法是退了重新进，或重启电脑吧，汗。

2.除了ndk-build命令外，还应该记住ndk-build -B 强制全编 和 ndk-build clean 清理这两个命令。

3.有些教程用到jni时还要把工程转成C++工程，再配置ndk-build.cmd命令，其实这个在前文也曾说过。个人觉得真心么必要啊。

-----本文系原创，转载请注明作者:yanzi1225627

欢迎大家加入OpenCV4Android联盟群：66320324 备注:yanzi

来源：<http://blog.csdn.net/yanzi1225627/article/details/27863615>

## Android 使用OpenCV的三种方式(Android Studio)

其实最早接触OpenCV是很久很久之前的事了，大概在2013年的5,6月份，当时还是个菜逼（虽然现在也是个菜逼），在那一段时间，学了一段时间的android（并不算学，一个月都不到），之后再也没接触android，而是一直在接触java web。那次接触OpenCV是因为一个学长的毕业设计，这次接触OpenCV是因为自己的毕业设计。2013年那年技术太菜，ndk环境都搭不好，当初还是eclipse环境，一直按照网上的教程去搭，下什么cygwin，简直就是个坑，网上的文章转来转去，都是过时的。后来一个机会看到了google官方的一个文档，就像发现了新大陆一样，发现ndk环境根本不需要装cygwin，装了你就坑了，装这个东西有好多G呢，时间浪费不说，简直误人子弟啊。后来在那年7月写下一篇博客

### NDK开发环境

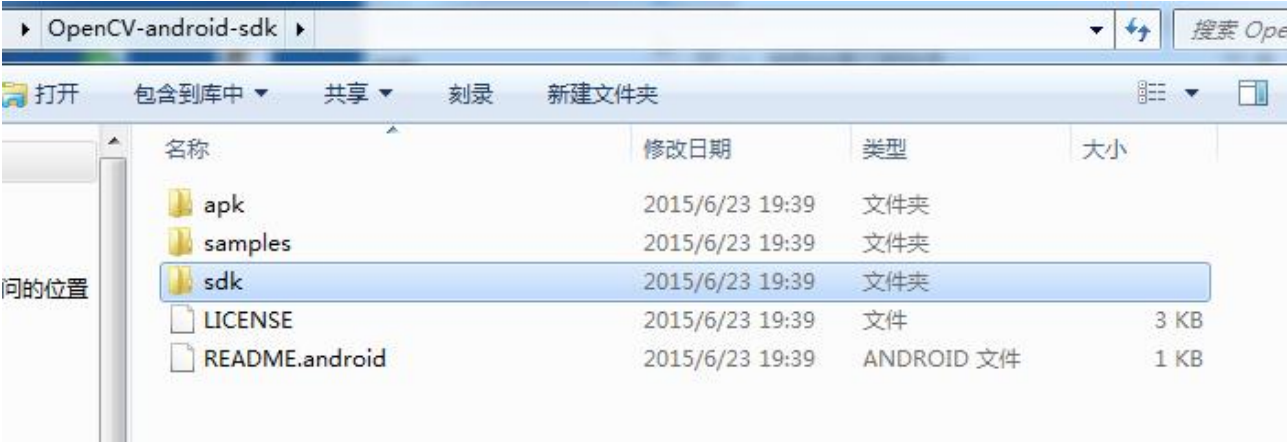
这段时间在填自己毕业设计的坑，要用到OpenCV，首先得下载到sdk吧，这个从官网下载就好了

### OpenCV for Android

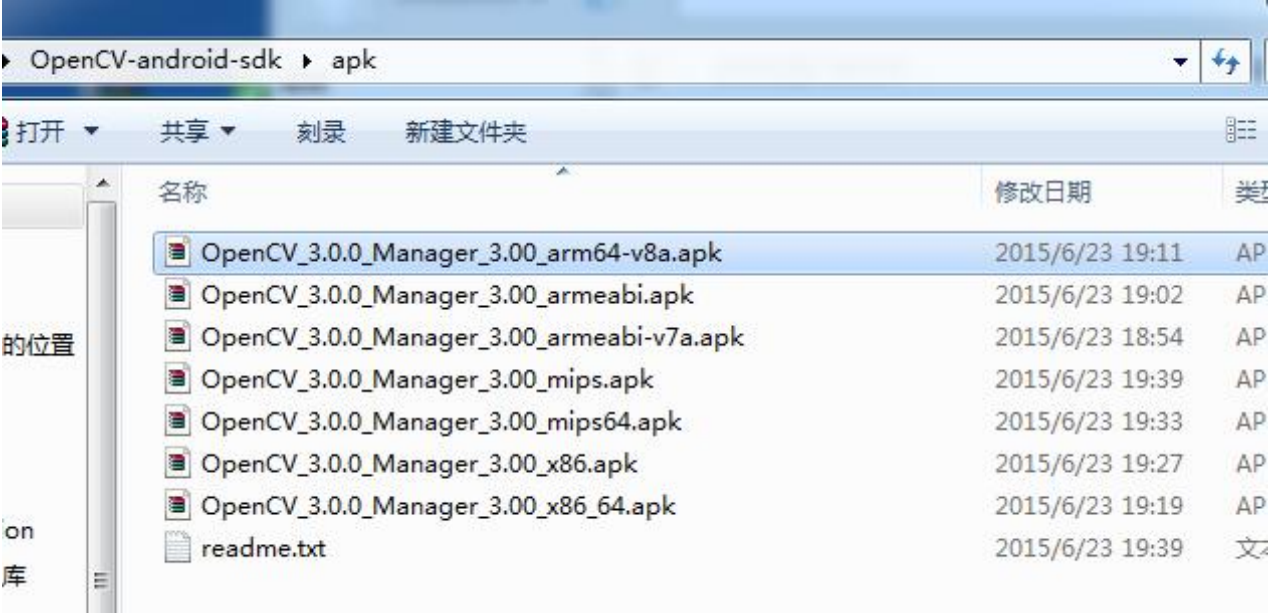
注意下载的是OpenCV for android。当前版本是3.0



解压后，里面的内容如下



samples目录下是样例代码，sdk目录下是我们需要用到的java层和jni层的代码。apk目录是manager的apk安装包  
其实OpenCV最简单的使用方式是使用manager，也就是使用apk目录下的安装包，安装对应的apk，将java层代码导入，使用**OpenCVLoader.initAsync()**加载库，之后你就可以直接用java代码调用Opencv相关的功能了。



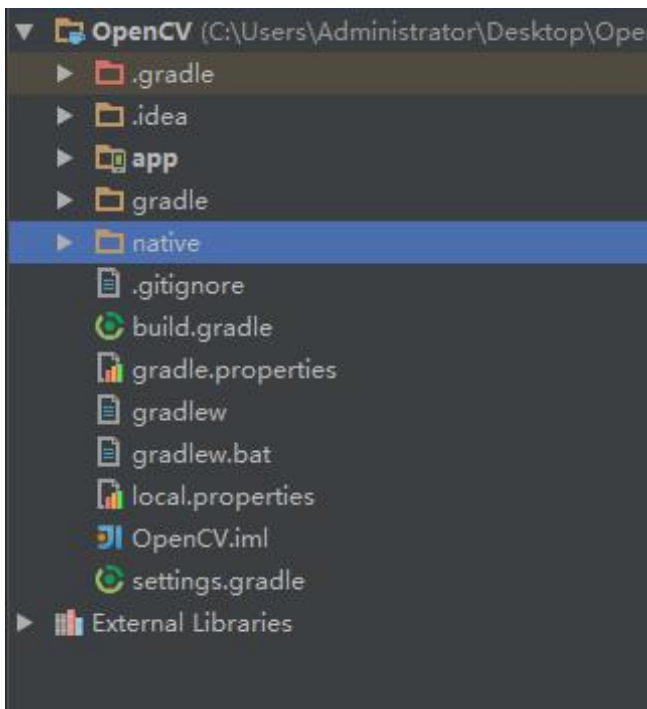
但是这种方式除了安装我们自己的apk还需要安装上面提到的manager的apk，用户体验十分不好，不推荐使用，本文的三种方式将完全脱离这个manager的apk。

本文下面的三种方式的内容参考自文章 [OpenCV4Android释疑: 透析Android以JNI调OpenCV的三种方式\(让OpenCVManager永不困扰\)](#)

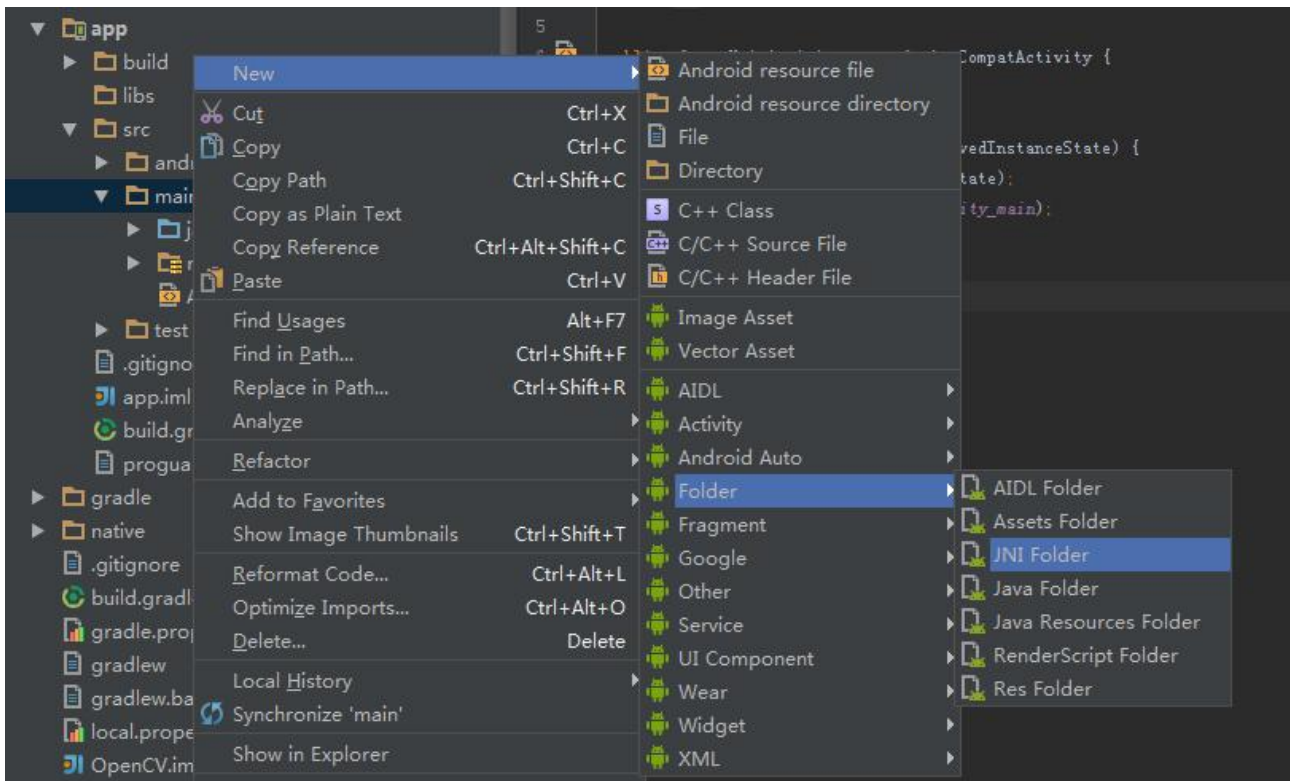
本篇文章使用android studio作为开发环境，由于实验性的构建工具对ndk支持还不好，所以使用旧的构建方式，在原来写的一篇博客基础上修改即可[android studio下ndk开发](#)

这正式介绍三种方式之前，我们需要做一些前期准备。

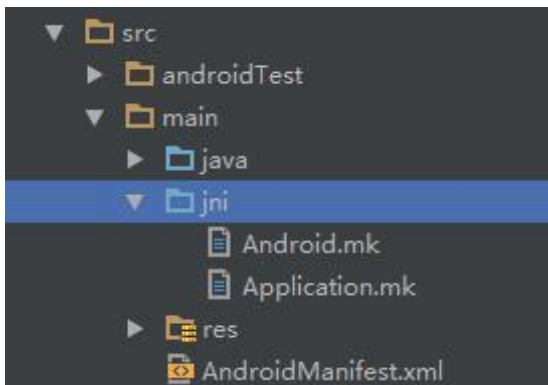
首先新建一个项目，将OpenCV中sdk目录下的native目录拷到项目根目录



然后新建Jni目录



在里面新建两个文件



编辑gradle.properties文件，增加下面的属性使用旧版的ndk功能（不添加会使用实验性的ndk构建工具）

```
android.useDeprecatedNdk=true
```

在local.properties文件中配置ndk目录

```
ndk.dir=D:\\AndroidSDK\\sdk\\ndk-bundle
```

编辑build.gradle，在android节点中增加下面的代码

```
sourceSets.main.jni.srcDirs = []
//禁止自带的ndk功能
sourceSets.main.jniLibs.srcDirs = ['src/main/libs','src/main/jniLibs']
//重定向so目录为src/main/libs和src/main/jniLibs，原来为src/main/jniLibs

task ndkBuild(type: Exec, description: 'Compile JNI source with NDK') {
    Properties properties = new Properties()
    properties.load(project.rootProject.file('local.properties').newDataInputStream())
    def ndkDir = properties.getProperty('ndk.dir')

    if (org.apache.tools.ant.taskdefs.condition.Os.isFamily(org.apache.tools.ant.taskdefs.condition.Os.FAMILY_WINDOWS)) {
        commandLine "$ndkDir/ndk-build.cmd", '-C', file('src/main/jni').absolutePath
    } else {
        commandLine "$ndkDir/ndk-build", '-C', file('src/main/jni').absolutePath
    }
}

tasks.withType(JavaCompile) {
    compileTask -> compileTask.dependsOn ndkBuild
}

task ndkClean(type: Exec, description: 'Clean NDK Binaries') {
    Properties properties = new Properties()
    properties.load(project.rootProject.file('local.properties').newDataInputStream())
    def ndkDir = properties.getProperty('ndk.dir')

    if (org.apache.tools.ant.taskdefs.condition.Os.isFamily(org.apache.tools.ant.taskdefs.condition.Os.FAMILY_WINDOWS)) {
        commandLine "$ndkDir/ndk-build.cmd", 'clean', '-C', file('src/main/jni').absolutePath
    } else {
        commandLine "$ndkDir/ndk-build", 'clean', '-C', file('src/main/jni').absolutePath
    }
}

clean.dependsOn 'ndkClean'
```

在之前新建的Application.mk中增加下面的内容

```
APP_STL := gnustl_static
APP_CPPFLAGS := -frtti -fexceptions
APP_ABI := armeabi armeabi-v7a
APP_PLATFORM := android-8
```

在Android.mk中增加下面的内容

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)
```

```
OpenCV_INSTALL_MODULES := on
OpenCV_CAMERA_MODULES := off

OPENCV_LIB_TYPE :=STATIC

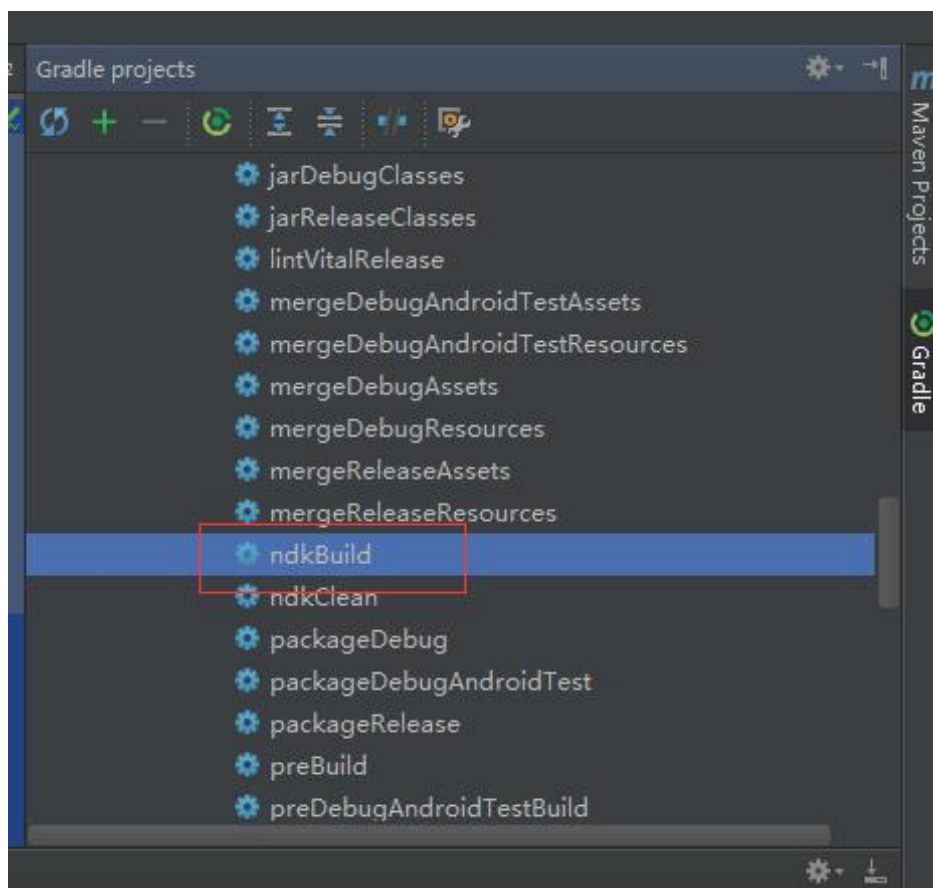
ifeq ("$(wildcard $(OPENCV_MK_PATH))", "")
include ../../../../native/jni/OpenCV.mk
elseinclude $(OPENCV_MK_PATH)
endif

LOCAL_MODULE := OpenCVLOCAL_SRC_FILES :=

LOCAL_LDLIBS += -lm -llog

include $(BUILD_SHARED_LIBRARY)
```

这时候，使用gradle构建一下，如果能成功构建出so，说明配置没问题，如下图，点击as右侧的gradle展开，双击ndkBuild进行构建





```

23:04:59: Executing external task 'ndkBuild'...
Configuration on demand is an incubating feature.
:app:ndkBuild
make.exe: Entering directory `C:/Users/Administrator/Desktop/OpenCV/app/src/main/jni'
[armeabi] Compile++ thumb: OpenCV <= cn_edu_zafu_opencv_OpenCVHelper.cpp
[armeabi] SharedLibrary : libOpenCV.so
[armeabi] Install : libOpenCV.so => libs/armeabi/libOpenCV.so
[armeabi-v7a] Compile++ thumb: OpenCV <= cn_edu_zafu_opencv_OpenCVHelper.cpp
[armeabi-v7a] SharedLibrary : libOpenCV.so
[armeabi-v7a] Install : libOpenCV.so => libs/armeabi-v7a/libOpenCV.so
make.exe: Leaving directory `C:/Users/Administrator/Desktop/OpenCV/app/src/main/jni'

BUILD SUCCESSFUL

Total time: 46.979 secs
23:05:48: External task execution finished 'ndkBuild'.

```

下面开始讲第一种方法，纯jni层的代码，该方法基于上面的所有步骤，为静态链接库声明java层的native方法

```

public class OpenCVHelper {
    static {
        System.loadLibrary("OpenCV");
    }
    public static native int[] gray(int[] buf, int w, int h);
}

```

使用javah命令生成头文件，内容如下

```

/* DO NOT EDIT THIS FILE - it is machine generated */#include <jni.h> /* Header for class cn_edu_zaf
extern "C" {
#ifdef __cplusplus
}
#endif
JNIEXPORT jintArray JNICALL Java_cn_edu_zafu_opencv_OpenCVHelper_gray
(JNIEnv *, jclass, jintArray, jint, jint);

#ifdef __cplusplus
}
#endif

```

新建cpp文件，实现对应的方法，就是灰度处理

```

#include "cn_edu_zafu_opencv_OpenCVHelper.h"#include <stdio.h>#include <stdlib.h>#include <opencv2/
using namespace cv;

extern "C" {

JNIEXPORT jintArray JNICALL Java_cn_edu_zafu_opencv_OpenCVHelper_gray(
    JNIEnv *env, jclass obj, jintArray buf, int w, int h);

JNIEXPORT jintArray JNICALL Java_cn_edu_zafu_opencv_OpenCVHelper_gray(
    JNIEnv *env, jclass obj, jintArray buf, int w, int h) {

```

```

jint *cbuf;
cbuf = env->GetIntArrayElements(buf, JNI_FALSE );
if (cbuf == NULL) {
    return 0;
}

Mat imgData(h, w, CV_8UC4, (unsigned char *) cbuf);

uchar* ptr = imgData.ptr(0);
for(int i = 0; i < w*h; i++){
    //计算公式: Y(亮度) = 0.299*R + 0.587*G + 0.114*B
    //对于一个int四字节, 其彩色值存储方式为: BGRA
    int grayScale = (int)(ptr[4*i+2]*0.299 + ptr[4*i+1]*0.587 + ptr[4*i+0]*0.114);
    ptr[4*i+1] = grayScale;
    ptr[4*i+2] = grayScale;
    ptr[4*i+0] = grayScale;
}

int size = w * h;
jintArray result = env->NewIntArray(size);
env->SetIntArrayRegion(result, 0, size, cbuf);
env->ReleaseIntArrayElements(buf, cbuf, 0);
return result;
}
}

```

之后, 需要将cpp文件编译进去, 在Andorid.mk文件中加入

```
LOCAL_SRC_FILES := cn_edu_zafu_opencv_OpenCVHelper.cpp
```

然后在java层写个测试方法测试一下是否进行灰度化了

```

Bitmap bitmap = ((BitmapDrawable) getResources().getDrawable(
    R.drawable.ic)).getBitmap();
int w = bitmap.getWidth(), h = bitmap.getHeight();
int[] pix = new int[w * h];
bitmap.getPixels(pix, 0, w, 0, 0, w, h);
int [] resultPixes=OpenCVHelper.gray(pix,w,h);
Bitmap result = Bitmap.createBitmap(w,h, Bitmap.Config.RGB_565);
result.setPixels(resultPixes, 0, w, 0, 0,w, h);
img.setImageBitmap(result);


```

运行效果如下, 灰度化后的结果





上面的这种方法生成的so库的大小见下图，大约有1.4M左右

 libOpenCV.so	2015/10/30 23:05	SO 文件	1,398 KB
--	------------------	-------	----------

第二种方法也是纯jni的，但是是动态链接库，在第一种基础上，修改Android.mk文件为

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

OpenCV_INSTALL_MODULES := on
OpenCV_CAMERA_MODULES := off

OPENCV_LIB_TYPE := SHARED

ifeq ("$(wildcard $(OPENCV_MK_PATH))", "")
include ../../../../native/jni/OpenCV.mk
else include $(OPENCV_MK_PATH)
endif
```

```
LOCAL_MODULE := OpenCVLOCAL_SRC_FILES := cn_edu_zafu_opencv_OpenCVHelper.cpp

LOCAL_LDLIBS += -lm -llog

include $(BUILD_SHARED_LIBRARY)
```

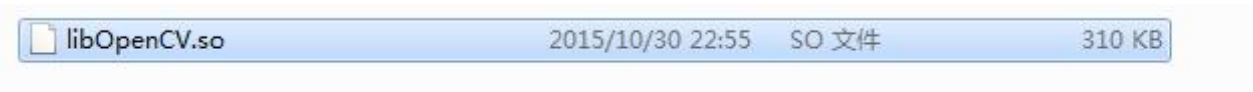
注意上面的OPENCV\_LIB\_TYPE属性的改动，从STATIC改为了SHARED，这时候再用ndkBuild一下，你会发现会输出一些警告以及一部分红色的内容

```
22:51:01: Executing external task 'ndkBuild'...
Configuration on demand is an incubating feature.
- app.ndkBuild
Android NDK: WARNING: C:/Users/Administrator/Desktop/OpenCV/app/src/main/jni/Android.mk: OpenCV: non-system libraries in linker flags: "-lopencv_java3
Android NDK: This is likely to result in incorrect builds. Try using LOCAL_STATIC_LIBRARIES
Android NDK: or LOCAL_SHARED_LIBRARIES instead to list the library dependencies of the
Android NDK: current module
Android NDK: WARNING: C:/Users/Administrator/Desktop/OpenCV/app/src/main/jni/Android.mk: OpenCV: non-system libraries in linker flags: "-lopencv_java3
Android NDK: This is likely to result in incorrect builds. Try using LOCAL_STATIC_LIBRARIES
Android NDK: or LOCAL_SHARED_LIBRARIES instead to list the library dependencies of the
Android NDK: current module
make.exe: Entering directory 'C:/Users/Administrator/Desktop/OpenCV/app/src/main/jni'
[armv7a] Compile++ thumb: OpenCV <= cn_edu_zafu_opencv_OpenCVHelper.cpp
[armv7a] SharedLibrary : libOpenCV.so
D:/AndroidSDK/sdk/ndk-bundle/toolchains/arm-linux-androideabi-4.8/prebuilt/windows-x86_64/bin/./lib/gcc/arm-linux-androideabi/4.8/../../../../arm-linux-androideabi/bin/ld.exe: warning: hidden symbol '__seal
[armv7a] Install : libOpenCV.so => libs/armv7a/libOpenCV.so
[armv7a-v7a] Compile++ thumb: OpenCV <= cn_edu_zafu_opencv_OpenCVHelper.cpp
[armv7a-v7a] SharedLibrary : libOpenCV.so
D:/AndroidSDK/sdk/ndk-bundle/toolchains/arm-linux-androideabi-4.8/prebuilt/windows-x86_64/bin/./lib/gcc/arm-linux-androideabi/4.8/../../../../arm-linux-androideabi/bin/ld.exe: warning: hidden symbol '__seal
[armv7a-v7a] Install : libOpenCV.so => libs/armv7a-v7a/libOpenCV.so
make.exe: Leaving directory 'C:/Users/Administrator/Desktop/OpenCV/app/src/main/jni'

BUILD SUCCESSFUL

Total time: 53.192 secs
22:51:56: External task execution finished 'ndkBuild'.
```

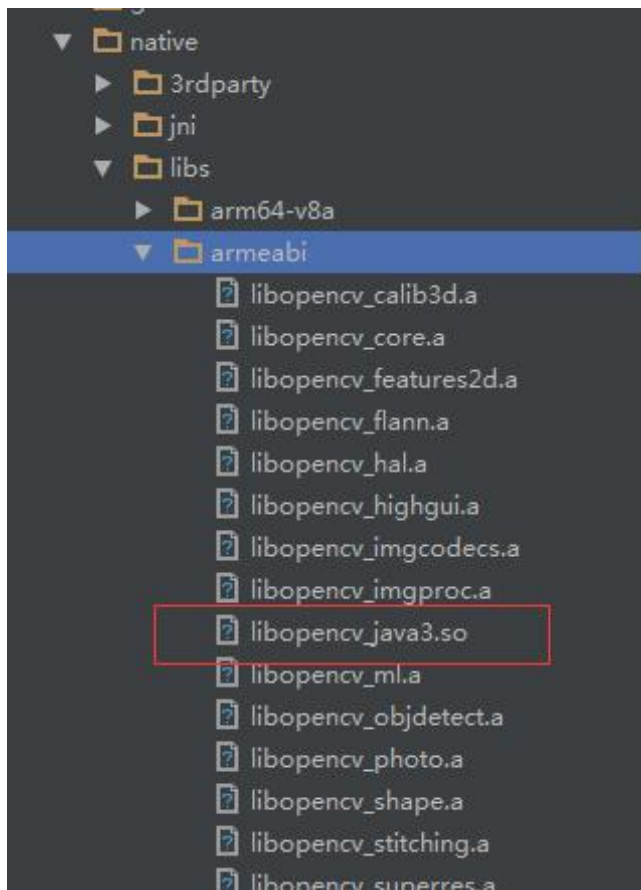
生成的so库的大小为310k，小了好几倍



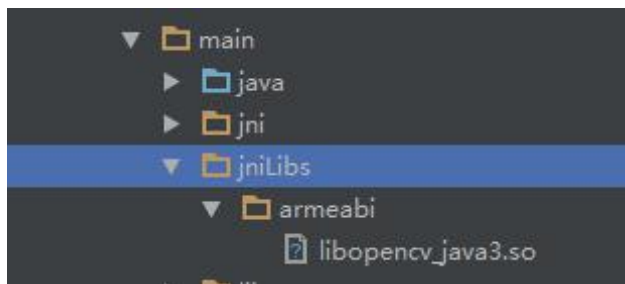
这时候如果你直接取运行程序，会报错误

```
FATAL EXCEPTION: main
Process: cn.edu.zafu.opencv, PID: 30534
java.lang.UnsatisfiedLinkError: dlopen failed: could not load library "libopencv_java3.so" needed by "libOpenCV.so": caused by library "libopencv_java3.so" not found
at java.lang.Runtime.loadLibrary(Runtime.java:364)
at java.lang.System.loadLibrary(System.java:526)
at cn.edu.zafu.opencv.OpenCVHelper.<clinit>(OpenCVHelper.java:10)
at cn.edu.zafu.opencv.MainActivity$1.onClick(MainActivity.java:32)
at android.view.View.performClick(View.java:4444)
at android.view.View$PerformClick.run(View.java:18440)
at android.os.Handler.handleCallback(Handler.java:733)
at android.os.Handler.dispatchMessage(Handler.java:95)
at android.os.Looper.loop(Looper.java:136)
at android.app.ActivityThread.main(ActivityThread.java:5016)
at java.lang.reflect.Method.invokeNative(Native Method) <1 internal calls>
at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:792)
at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:608)
at dalvik.system.NativeStart.main(Native Method)
```

原因是我们使用的是动态库加载方式，还需要将依赖的so加进去，这个so就是图中的libopencv\_java3.so，他在我们的最开始加到项目里的native目录中



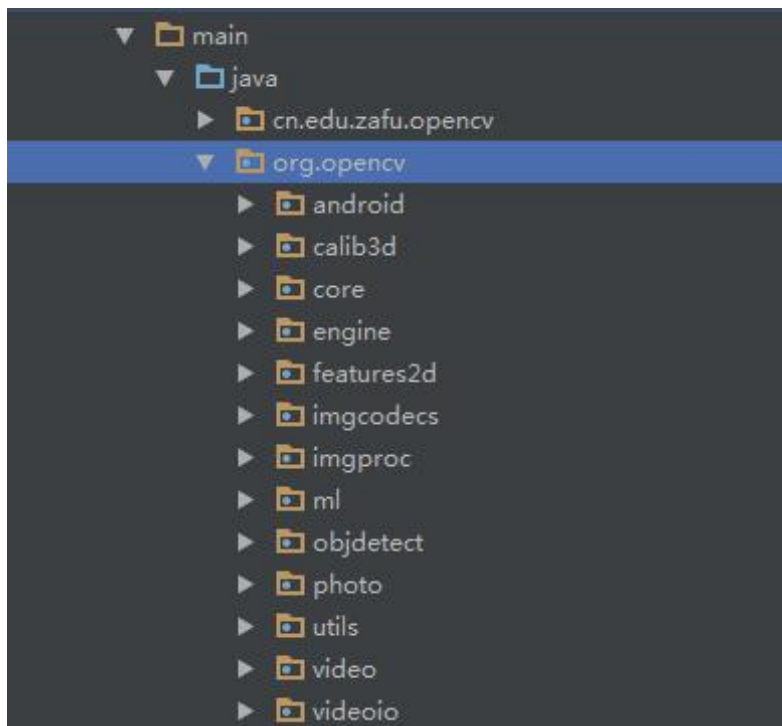
将它拷到我们的jniLibs目录中去，这里只拷贝armeabi和armeabi-v7a中的，至于其他的按需拷贝



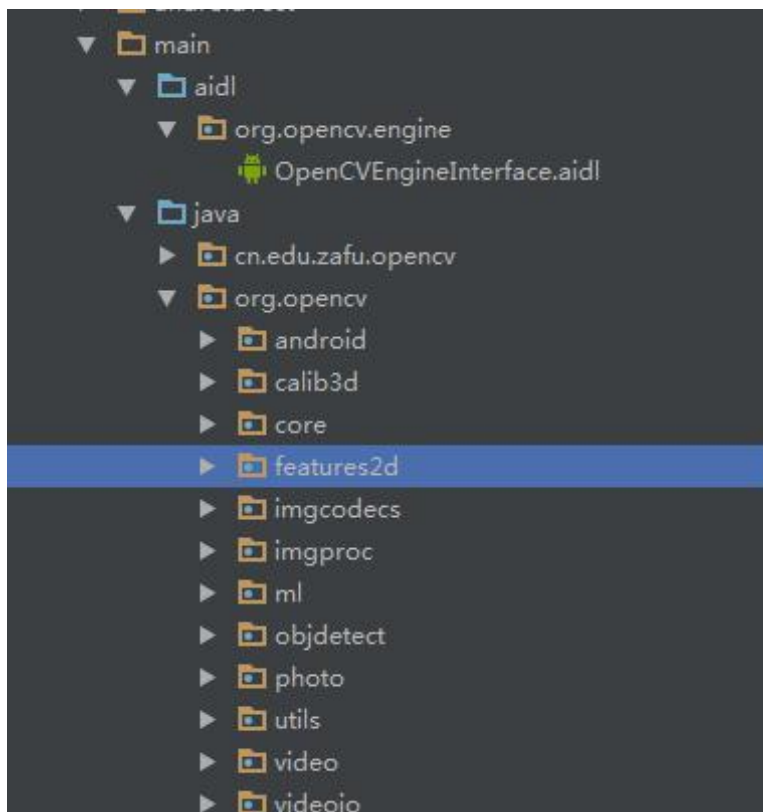
这时候运行就不会报错了。

既然我们使用了动态链接库，那么我们同样也可以使用java层的接口，优点是java开发速度相对快一点。第三种方法在第二种方法基础上，使用纯java层代码进行处理。

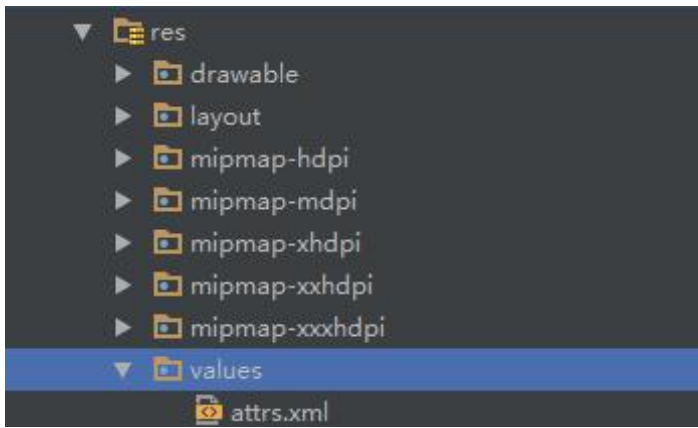
在此之前，我们需要将sdk目录中的java代码拷到项目中去



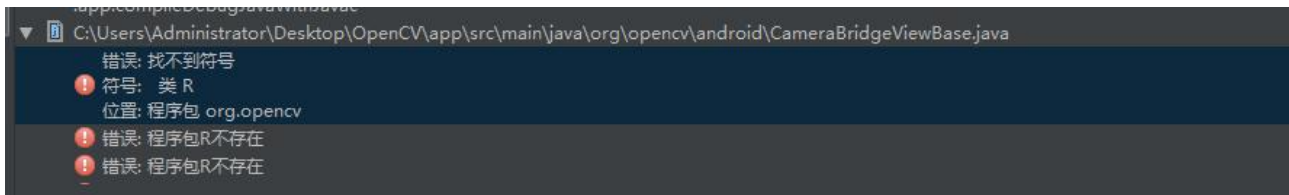
但是org.opencv.engine包中是一个aidl，我们需要将它剪贴到aidl目录中去，就像这样子



最后还有一个资源文件attrs.xml，拷过来



build一下项目，不出意外应该会报错，这时候找到该类，引入自己的R文件包就可以了



再次build应该就不会有什么问题了。

java层的测试方法

```
OpenCVLoader.initDebug();
Mat rgbMat = new Mat();
Mat grayMat = new Mat();
Bitmap srcBitmap = BitmapFactory.decodeResource(getResources(), R.drawable.ic);
Bitmap grayBitmap = Bitmap.createBitmap(srcBitmap.getWidth(), srcBitmap.getHeight(), Bitmap.Config.
Utils.bitmapToMat(srcBitmap, rgbMat); //convert original bitmap to Mat, R G B.
Imgproc.cvtColor(rgbMat, grayMat, Imgproc.COLOR_RGB2GRAY); //rgbMat to gray grayMat
Utils.matToBitmap(grayMat, grayBitmap); //convert mat to bitmap
img.setImageBitmap(grayBitmap);
```

注意使用**OpenCVLoader.initDebug();**进行初始化而不是使用**OpenCVLoader.initAsync()**

这种方法的特点是处理都在java层，不怎么会涉及jni层的代码，除非java层完成不了的工作会转移到jni层去。

三种方法各有各的优点，根据自己的情况进行选择。

- 如果c++特别好的，建议使用第一种方法
- 如果更习惯java代码的，并且java层的函数都能进行处理的，建议选择第三种方法
- 第二种方法建议在第三种方法不满足条件的情况下进行辅助使用，因为使用了第三种方法的前提是使用第二种方法的动态链接库。