总结

原 android opencv linux lavnFan 9月5日发布

- Android 中写的 JNI 如何调用 OpenCV ？
- OpenCV 如何配置到 Linux 服务器上？

OpenCV for Android

如果想实现图片的高斯模糊，图片比较，人脸识别等算法，OpenCV 可能是现成库里比较好的选择。
使用 OpenCV 的优缺点：

- 现成库 C++ 调用，封装很好，实现较为简单，上层 JNI 调用性能较好
- 失去 Java 的跨平台特性

下边我们来看看在 Android 的 JNI 中如何使用 OpenCV ？

1、下载 OpenCV for Android 的 SDK

OpenCV for Android SDK

解压后有：

- apk：manger 的 apk，这种方法不需要导入 OpenCV 的 sdk ，但需要在安装自己的apk外，还需安装 这里边的 manager 的 apk，体验不好,不建议使用。
- samples：样例代码
- sdk：需要用到的 java 层 或 jni 层的代码

具体的 IDE 配置方法

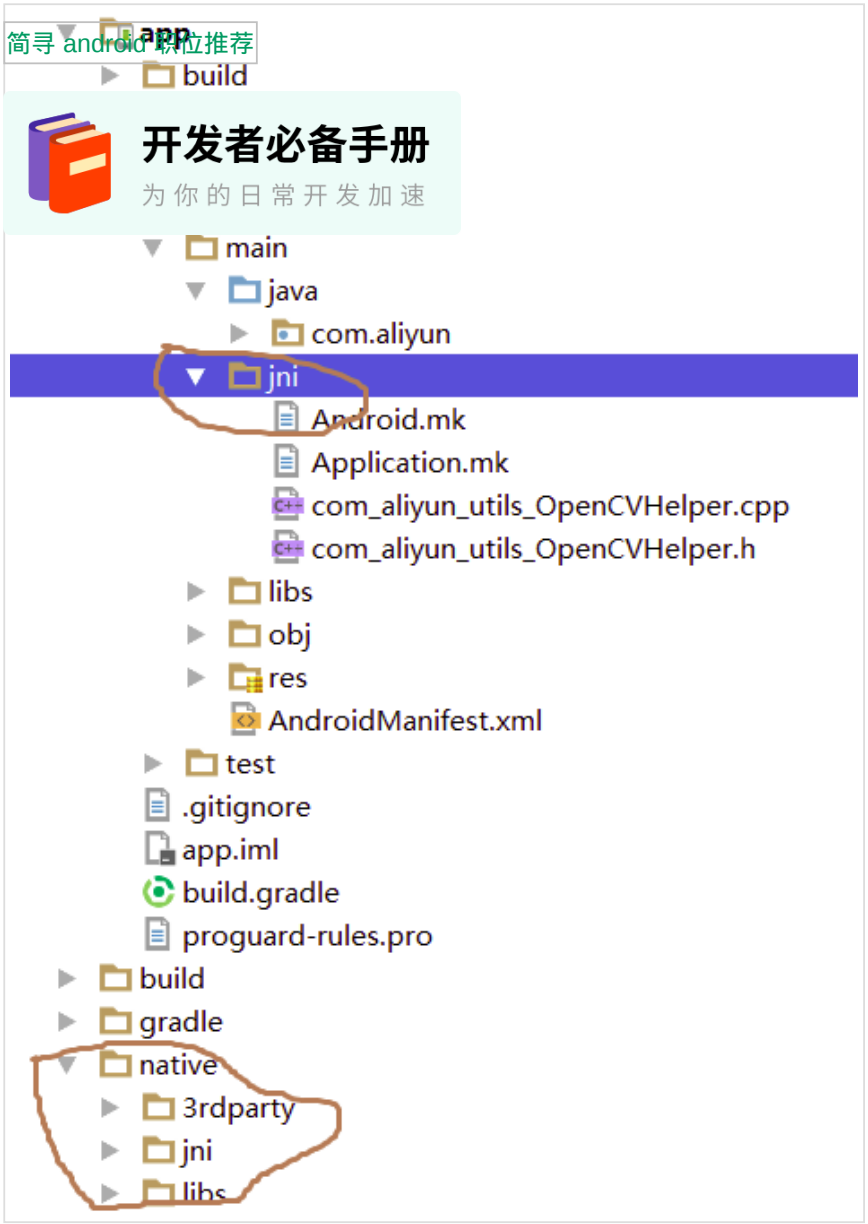
4/doc/tutorials/introduction/android_binary_package/dev_with_OCV_on_An

2、Android studio 配置

2.1、拷贝 native

- 将 OpenCV 中 sdk 目录下的 native 全拷到项目根目录下
- 新建 JNI Folder，并在目录下创建俩个文件：Android.mk 和 Application.mk

建立在Project展开如下：



2.2、 文件配置

- Android.mk

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



```
on
OPENCV_CAMERA_MODULES := off

OPENCV_LIB_TYPE :=STATIC

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

LOCAL_MODULE := OpenCV

LOCAL_SRC_FILES :=

LOCAL_LDLIBS +=  -lm -llog

include $(BUILD_SHARED_LIBRARY)
```

这里是以静态链接导入 OpenCV 库，然后编译成 共享的 so 库，所以会导致最终的 so 库体积比较大。
注意： `include ..\..\..\..\native\jni\OpenCV.mk` **的路径一定要对！！！**

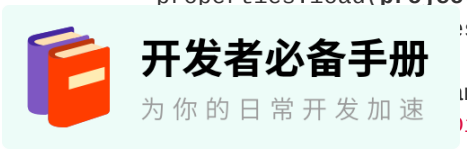
- Application.mk

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

这里配置了 OpenCV 需要用到的 **STL** 库，以及编译的硬件平台等。

- build.gradle

```
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，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()
                    s.getProperty('ndk.dir')

    if (org.apache.tools.ant.taskdefs.condition.Os.isFamily(org.apache.tools.ant.task
        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
}
```

- local.properties

```
ndk.dir=D\:\\program\\Android\\sdk\\ndk-bundle
```

这里配置 NDK 的路径

2.3、JNI 调用

- 声明 java 层的 native 方法

```
public class OpenCVHelper {

        static {
            System.loadLibrary("OpenCV");
        }

    public static native double compareImages(String old_image, String new_image,int
}
```

- 使用 javah 命令生成头文件

```
javah -jni com.aliyun.utils.OpenCVHelper
```

其中，javah的环境需要配置到电脑的 `path` 中去；
生成了 `com_aliyun_utils_OpenCVHelper.h` 的头文件。

`_OpenCVHelper.cpp` ，

修改 `LOCAL_SRC_FILES := com_aliyun_utils_OpenCVHelper.cpp`

2.4、ndk-build

- 使用 Android studio 的 ndk-build 工具构建so库，点击右侧的 gradle ，展开在 other 下边找到 ndk-build。
- 然后在 java 层中就可以使用 JNI 层的功能了。

3、静态还是共享链接 OpenCV

上边我们在 Android.mk 中，使得 `OPENCV_LIB_TYPE :=STATIC` ，以静态库的方式导入 OpenCV ，所以生成的 so 库比较大，达到好几M。

另一种方式是使用动态库的方式引入 OpenCV ，即把 `OPENCV_LIB_TYPE :=SHARED` ，动态加载所需要的库，在 ndk-build 时，会报 `-lopencv_java3` 的 warning :

```
Android NDK: WARNING:D:/code/demo/OpenCVHelper/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:D:/code/demo/OpenCVHelper/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
```

这时我们把 native-jni-编译平台下的 `libopencv_java3.so` 导入即可。

OpenCV for Linux

OpenCV for Android 的 demo 完成了，但配置到服务器上，就是各种坑啊，一个 bug 接着一个 bug，所幸最终都依依解决了，现在把那些 bug 场景重现！

!就是需要把 **OpenCV 库导入**进来，不管是静态方式还是动态方式。

- 方式一：利用已写好的 OpenCV.mk 引入
- 方式二：在自己写的 Android.mk 中引入

俩种方式并无本质差别，但用已有的 OpenCV.mk 引入时，会出现**更多的问题**，所以就单独作一类方法，来总结遇到的坑。

方法一、利用原来的 OpenCV.mk 引入

1、OpenCV.mk 的路径问题

代码：

```
include ..\..\native\jni\OpenCV.mk
```

报错：

```
packages/apps/DVRRecorder/jni/OpenCV/Android.mk:9: ..\..\native\jni\OpenCV.mk: No such file
```

代码修改成这样：

```
$(LOCAL_PATH)\..\..\native\jni\OpenCV.mk
```

还是报同样的错误。。。
认真思考一番,有办法了！！！

解决：

可以用 **ls 命令在服务器上看能不能进入这个目录~**，发现进不了，再细细思考一下，应该把斜杠给改反一下。


```
include $(LOCAL_PATH)/../../native/jni/OpenCV.mk
```

恩，没有报这个错误了，但紧接着又报了第二个错误。。。

2、TARGET_ARCH_ABI 的问题

报错：

```
packages/apps/DVRRecorder/jni/OpenCV/../../native/jni/OpenCV.mk:40: packages/apps/DVRRecorc
```

OpenCV-.mk 是个什么鬼，但明显已经解决了 OpenCV.mk 的路径问题，怎么又出来 OpenCV-.mk 呢？

深入：

这时深入 OpenCV.mk，发现有这样的一段代码：

```
OPENCV_SUB_MK:=$(call my-dir)/OpenCV-$(TARGET_ARCH_ABI).mk
```

这里找到了 - 号，很有可能是 `TARGET_ARCH_ABI` 的问题，于是在 导入的 native 目录下，全局搜索，发现

```
jni/OpenCV.mk:14:OPENCV_TARGET_ARCH_ABI:=$(TARGET_ARCH_ABI)
jni/OpenCV.mk:17:OPENCV_LIBS_DIR:=$(OPENCV_THIS_DIR)/../libs/$(OPENCV_TARGET_ARC
jni/OpenCV.mk:18:OPENCV_3RDPARTY_LIBS_DIR:=$(OPENCV_THIS_DIR)/../3rdparty/libs/$
jni/OpenCV.mk:22:OPENCV_SUB_MK:=$(call my-dir)/OpenCV-$(TARGET_ARCH_ABI).mk
jni/OpenCV.mk:64:ifeq ($(OPENCV_MK_$(OPENCV_TARGET_ARCH_ABI)_ALREADY_INCLUDED),)
jni/OpenCV.mk:76:    OPENCV_MK_$(OPENCV_TARGET_ARCH_ABI)_ALREADY_INCLUDED:=on
```

搜索后发现**只有使用，但没有定义！！！**

问题就很明显了，**没有导入编译的硬件平台**，很有可能也是路径问题。

BI 的路径了，直接根据所编译的平台，合理取值。

在 Android developer Guide上有说明：https://developer.android.com/ndk/guides/android_mk.html

TARGET_ARCH_ABI

This variable stores the name of the CPU and architecture to target when the build system parses this `Android.mk` file. You can specify one or more of the following values, using a space as a delimiter between multiple targets. Table 1 shows the ABI setting to use for each supported CPU and architecture.

Table 1. ABI settings for different CPUs and architectures.

CPU and architecture	Setting
ARMv5TE	`armeabi`
ARMv7	`armeabi-v7a`
ARMv8 AArch64	`arm64-v8a`
i686	`x86`
x86-64	`x86_64`
mips32 (r1)	`mips`
mips64 (r6)	`mips64`
All	`all`

用 `mm -B` 编译时，可以发现所编译的平台，根据使用的 CPU 架构来选取一个合理值，比如：

```
TARGET_ARCH_ABI := armeabi-v7a
```

恩，以为没问题了，但下一个错误马上来~~~

3、编译，连接的问题

报错：

```
No rule to make target `out/target/product/aeon6735_65c_s_l1/obj_arm/STATIC_LIBRARIES/oper
```


追踪：

但有关键字， `STATIC_LIBRARIES` ， `SHARED_LIBRARIES` ，且也肯定是 OpenCV.mk 出问题了，在 Android.mk 中我们声明了这俩个条件：

```
OpenCV_INSTALL_MODULES := on
OpenCV_CAMERA_MODULES := off
OPENCV_LIB_TYPE :=STATIC
```

于是追踪到 OpenCV.mk 中发现：

```
ifeq ($(OPENCV_INSTALL_MODULES),on)
        LOCAL_$(OPENCV_LIB_TYPE)_LIBRARIES += $(foreach mod, $(OPENCV_LIBS), opencv_$(mod))
else
        LOCAL_LDLIBS += -L$(call host-path,$(LOCAL_PATH)/$(OPENCV_LIBS_DIR)) $(foreach lib,
endif

ifeq ($(OPENCV_LIB_TYPE),STATIC)
        LOCAL_STATIC_LIBRARIES += $(OPENCV_3RDPARTY_COMPONENTS)
endif
```

猜测是这俩个条件判断后进去的语句出问题了，

- 当为 `on` 时，以模块方式加载 `LOCAL_STATIC_LIBRARIES` ，当为 `off` 时，以 `LOCAL_LDLIBS` 方式加载lib下的库。
- 当 `OPENCV_LIB_TYPE==STATIC` 时，第三方的lib加载到 `LOCAL_STATIC_LIBRARIES` 中。

解决：

既然出错了，那把这俩句话都注释掉试试~

，既然我们把引入库的语句给删了，那接下来，肯定是需要我们把库的链接

报错：

```
packages/apps/DVRRecorder/jni/OpenCV/../../native/jni/include/opencv2/core/base.hpp:53:21:
 #include <algorithm>
```

STL 模板库的头文件没找到，所以需要引入 `algorithm` 的头文件。

解决：

找到头文件所在的位置，并以如下方式引入在 Android.mk 中：

```
LOCAL_C_INCLUDES := external/stlport/stlport/
LOCAL_C_INCLUDES += bionic
```

解决了这个问题，发现又引入了更多的 bug ，但都是库的链接问题，接下来的库引入和第二种方式一样，只是第二种，就直接去掉了 OpenCV.mk，直接在 Android.mk 中一步一步的引入链接库。

方法二、另启炉灶，在 Android.mk 中一个库一个库的引入

1、静态库的引入

报错：

```
packages/apps/DVRRecorder/jni/OpenCV/com_aliyun_utils_OpenCVHelper.cpp:30: error: undefinec
packages/apps/DVRRecorder/jni/OpenCV/../../native/jni/include/opencv2/core/cvstd.hpp:667: e
```

```
packages/apps/DVRRecorder/jni/OpenCV/com_aliyun_utils_OpenCVHelper.cpp:31: error: undefinec
packages/apps/DVRRecorder/jni/OpenCV/../../native/jni/include/opencv2/core/cvstd.hpp:667: e
packages/apps/DVRRecorder/jni/OpenCV/com_aliyun_utils_OpenCVHelper.cpp:34: error: undefinec
```


链接，而且这些链接应该是 OpenCV 库中的。

解决：

这里我们以静态库的方式引入 OpenCV 中的静态链接库，把 lib/armeabi-v7a 和 3rdparty/libs/armeabi-v7a 目录下的库全引入。

shell pwd：显示当前工作目录
这里我们以全路径的方式加入对应的库：

```
PATHD=$(shell pwd)
LOCAL_LDFLAGS +=-L$(PATHD)/$(LOCAL_PATH)/lib/  -l$(PATHD)/$(LOCAL_PATH)/lib/libopencv_core.
            -l$(PATHD)/$(LOCAL_PATH)/lib/libopencv_imgproc.a\
            -l$(PATHD)/$(LOCAL_PATH)/lib/libopencv_highgui.a\
            -l$(PATHD)/$(LOCAL_PATH)/lib/libopencv_imgcodecs.a \
            -l$(PATHD)/$(LOCAL_PATH)/lib/libIlmImf.a  \
            -l$(PATHD)/$(LOCAL_PATH)/lib/liblibjpeg.a  \
            -l$(PATHD)/$(LOCAL_PATH)/lib/liblibwebp.a  \
            -l$(PATHD)/$(LOCAL_PATH)/lib/liblibtiff.a  \
            -l$(PATHD)/$(LOCAL_PATH)/lib/liblibpng.a  \
            -l$(PATHD)/$(LOCAL_PATH)/lib/liblibjasper.a   \
            -l$(PATHD)/$(LOCAL_PATH)/lib/libtbb.a  \
            -l$(PATHD)/$(LOCAL_PATH)/lib/libopencv_java3.so \
            -lstdc++
```

2、STL 库的引入

报错

已经解决了 OpenCV 库的链接错误，但又报了如下错误：

```
error: undefined reference to 'vtable for std::basic_istringstream<char, std::char_traits<c
```


我们还要引入关于 **libgnustl libsupc++ libstlport**。

解决：

我们以静态的链接库引入 stl 等一系列库，如下：

```
prebuilt_stdcxx_PATH := prebuilts/ndk/current/sources/cxx-stl/gnu-libstdc++
LOCAL_LDFLAGS += -L$(prebuilt_stdcxx_PATH)/libs/armeabi-v7a -lgnustl_static -lsupc++
```

- 先定义一个路径的变量
- 然后再在 LOCAL_LDFLAGS 中引入该链接库

！！！但是，错误还是没变，但通过 `$(warning $(LOCAL_LDFLAGS))` 打印，确实可以找到对应的链接，这又是什么原因？？？

找了很久，决定在对应路径下再看看有没有其他的相同链接库，诶，发现有**同名的共享 so 库**，换成动态链接库试试~~

```
LOCAL_LDFLAGS += -L$(prebuilt_stdcxx_PATH)/libs/armeabi-v7a -lgnustl_shared -lsupc++
```

竟然发现报该错误的链接消失了！！神奇。

猜测原因：

可能是我们在 Android.mk 中要编译的是动态 so 库，所以在既有静态库，也有动态库的情况下，需要引入和要编译的库相同类型的。

3、LOCAL_LDIBS 的引入

报错：


```
.cpp:function initOpenCLAndLoad: error: undefined reference
```

此处省略一堆同样的错误~~~

解决：

```
LOCAL_LDLIBS +=  -lm -llog  -ldl -lz
```

上边的的错误是没有引入 -ldl 的原因。

总结

这次使用 OpenCV 的经历，bug 是一浪接着一浪。有些命令很有用，总结如下：

- 强大的搜索：find * -exec grep -iHn '搜索名' {} ;
- $(warning $(PATHD))：打印 makefile 中的 PATHD 变量
- PATHD=$(shell pwd)：获取当前工作目录，而不用手动书写全路径。
- LOCAL_C_INCLUDES：额外的 C/C++ 编译头文件路径
- LOCAL_LDLIBS：即 ldlibs，就是指定那些存在于**系统目录下本模块需要连接的库**。如果某一个库既有动态库又有静态库，那么在默认情况下是链接的动态库而非静态库
- LOCAL_LDFLAGS：这个编译变量传递给链接器一个一些额外的参数，比如想**传递给外面的库和库路径给ld**，那么就要加到这个上面，如：

  ```
  LOCAL_LDFLAGS += -L$(prebuilt_stdcxx_PATH1)/libs/armeabi-v7a -lstlport_shared
  ```

  或者直接加上绝对路径库的全名：

```
LOCAL_LDFLAGS +=-L$(PATHD)/$(LOCAL_PATH)/lib/  -l$(PATHD)/$(LOCAL_PATH)/lib/libopencv
-l$(PATHD)/$(LOCAL_PATH)/lib/libopencv_imgproc.a\
.OCAL_PATH)/lib/libopencv_highgui.a\
.OCAL_PATH)/lib/libopencv_imgcodecs.a \
```


且需要注意：如果是非系统的第三方库，只能用LOCAL_LDFLAGS方式，LOCAL_LDLIBS方式不行。

本文发表于个人博客：http://lavnfan.github.io/，欢迎指教。

9月5日发布 ···

你可能感兴趣的文章

OpenCV 静态链接 libstdc++ 103 浏览

Android NDK和OpenCV整合开发 (3) OpenCV 5 收藏，4.8k 浏览

Differences between OpenCV JavaCV and OpenCV4Android 8 收藏，4k 浏览

评论

默认排序 时间排序

lavnFan
314 声望

关注作者

发布于专栏

个人总结

有关Android开发、数据结构与算法等等总结，详情请见个人博客：http://lavnfan.github.io/ 。

5 人关注

关注专栏