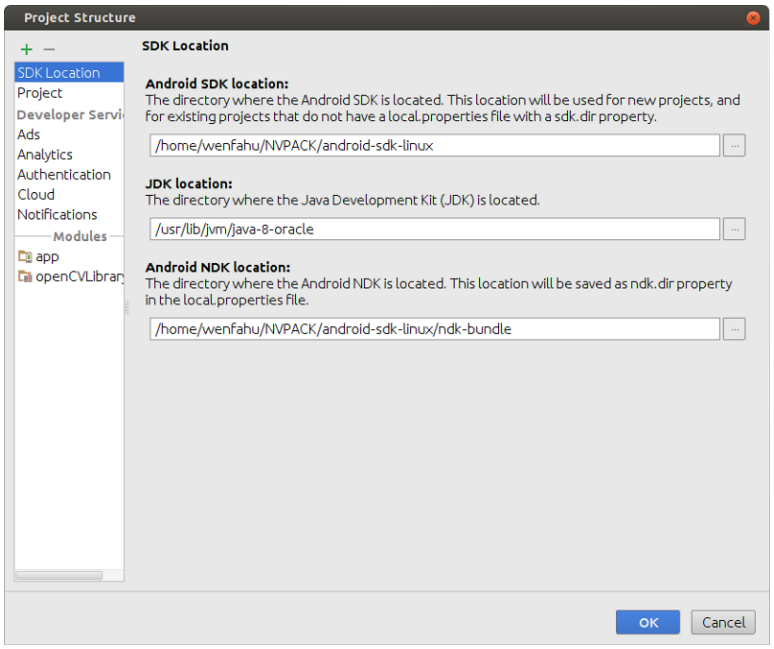
温发琥 Follow
Sep 30, 2016 · 4 min read

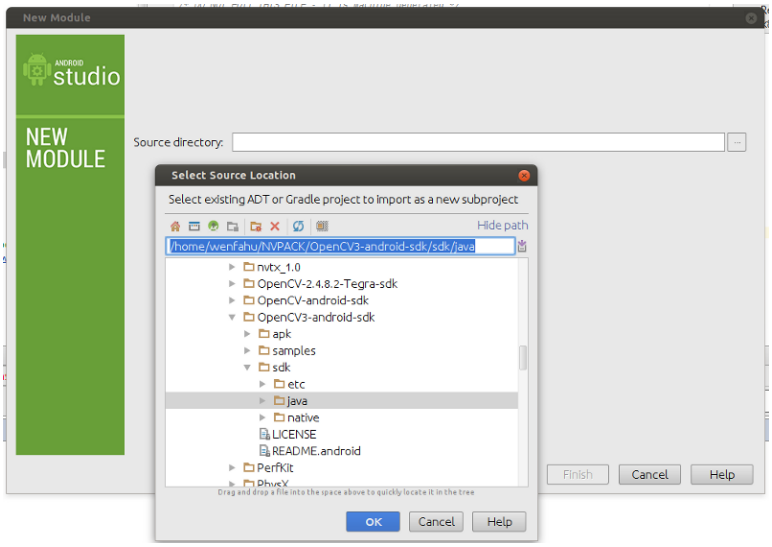
# OpenCV and Android NDK integration in Android Studio

. . .

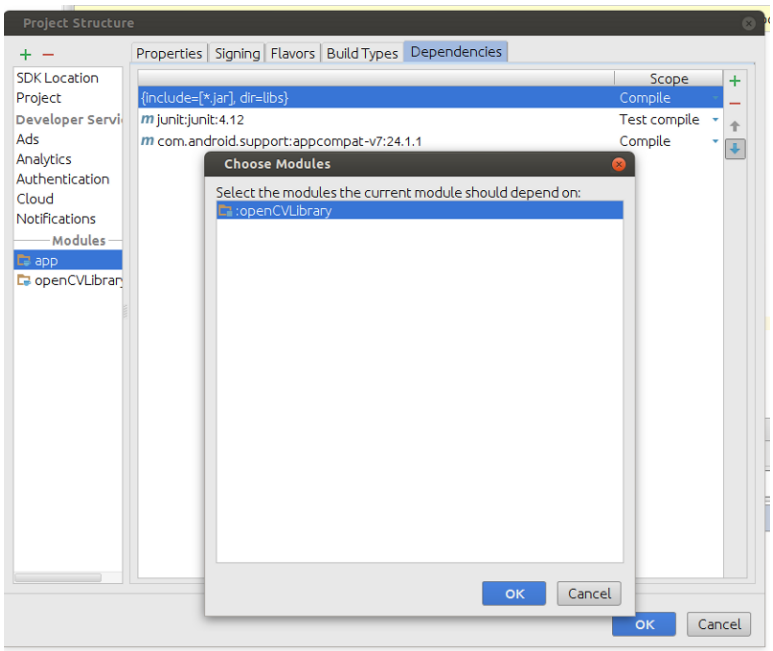
1. Download the OpenCV4Android SDK from this site（here I use version 3.1.1). Create the project in your Android Studio (I use version 2.1.2). Config your project SDK, JDK (I personally recommend Oracle JDK rather than Open JDK) and NDK in File/Project Structure/SDK location




2. Import OpenCV4Android as a module dependency of your app. In your Android Studio Project, select File/New/Import Module.. . Then in the New Module dialog, navigate to the path of OpenCV4Android Java SDK. For instance, my path is ***/home/wenfahu/NVPACK/OpenCV3-android-sdk/sdk/java***




Then right click your project in the project panel and select **Open Module Settings.** Add the OpenCV as a module dependency of your app.




You may get error messages after this, most of the time you can just change the *compileSdkVersion* in **build.gradle** file of the **openCVLibrary** in Android Studio

```
apply plugin: 'com.android.library'
```

```
android {
    compileSdkVersion 21 // change this to higher version as
21
    buildToolsVersion "19.1.0"

    defaultConfig {
        minSdkVersion 8
        targetSdkVersion 21
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-
android.txt'), 'proguard-rules.txt'
        }
    }
}
```

And then sync the project.

3. If you would like to use OpenCV in C++ code via Android NDK, Plz ignore this step and read on.

If you would use OpenCV in java code, you can just create a folder named **jniLibs** in **src/main** and create sub-directories of corresponding arch of your Android devices (such as arm64-v8a, armeabi-v7a …) and copy the corresponding **libopencv_java3.so** file from the

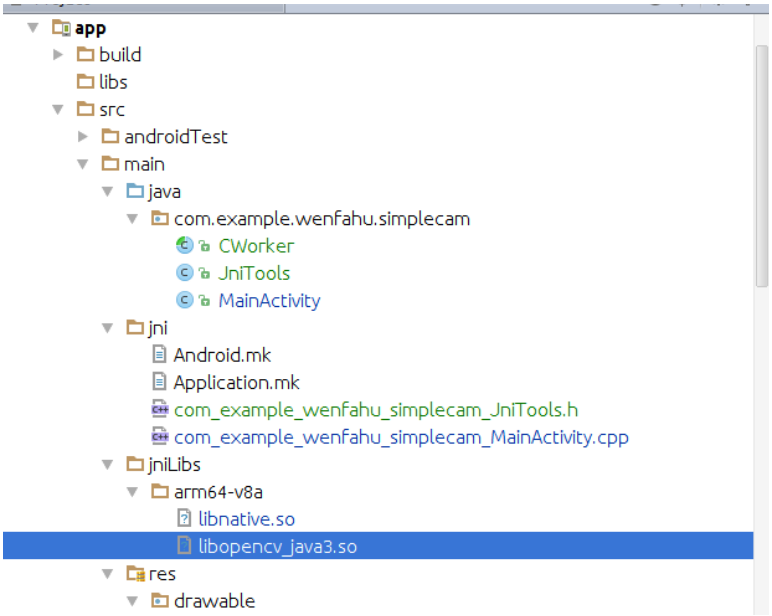
```
<OpenCV4Android_sdk_path>/sdk/native/libs/<arch>
```

into those folders.

For example, my **libopencv_java3.so** file for arm64v8a located in

```
/home/wenfahu/NVPACK/OpenCV3-android-sdk/sdk/native
/libs/arm64-v8a
```

And I copy it to **jniLibs/arm64-v8a** folder in my Android Studio project.




And you can use OpenCV staff in the Java Classes.

4. Create directory **src/main/jni**. Create files **Android.mk** and **Application.mk** in the **jni** directory you just created. You need these files to instruct NDK tool to build the native part of your project. The details of configuration can be found here. My configuration is as follows:

**Android.mk**

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

include $(CLEAR_VARS)

OPENCV_ROOT:=/home/wenfahu/NVPACK/OpenCV3-android-sdk
OPENCV_CAMERA_MODULES:=on
OPENCV_INSTALL_MODULES:=on
OPENCV_LIB_TYPE:=SHARED
include ${OPENCV_ROOT}/sdk/native/jni/OpenCV.mk

NDK_MODULE_PATH=/home/wenfahu/NVPACK/android-sdk-linux/ndk-
bundle
LOCAL_ARM_NEON := true
LOCAL_SRC_FILES :=
com_example_wenfahu_simplecam_MainActivity.cpp
LOCAL_CPPFLAGS := -std=gnu++0x
LOCAL_CFLAGS += -O2
LOCAL_LDLIBS += -llog -ldl
LOCAL_MODULE := native


include $(BUILD_SHARED_LIBRARY)
```

You may need to replace the **OPENCV_ROOT** and **NDK_MODULE_PATH** with the actual path of OpenCV4Android SDK and NDK on your machine.

**Application.mk**

```
APP_STL := gnustl_static
APP_CPPFLAGS := -frtti -fexceptions
NDK_TOOLCHAIN_VERSION=4.9
APP_ABI := arm64-v8a
APP_PLATFORM := android-16
APP_OPTIM := release
```

Remember to replace the values of **NDK_TOOLCHAIN_VERSION** , **APP_ABI** and **APP_PLATFORM** with the corresponding values of your project.

**5.** Configure the **build.gradle** file of your app. Here is my **build.gradle** for OpenCV integration with Android NDK (my gradle version is 2.1.2）

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 24
    buildToolsVersion "24.0.0"

    defaultConfig {
        applicationId "com.example.wenfahu.simplecam"
        minSdkVersion 21
        targetSdkVersion 24
        versionCode 1
        versionName "1.0"
    }

    sourceSets.main.jni.srcDirs = []

    task ndkBuild(type: Exec, description: 'Compile JNI
source via NDK') {
        def ndkDir = "/home/wenfahu/NVPACK/android-sdk-
linux/ndk-bundle"
        commandLine "$ndkDir/ndk-build",
                'NDK_PROJECT_PATH=build/intermediates/ndk',
                'NDK_LIBS_OUT=src/main/jniLibs',
                'APP_BUILD_SCRIPT=src/main/jni/Android.mk',
                'NDK_APPLICATION_MK=src/main
/jni/Application.mk'
    }

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

```
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-
android.txt'), 'proguard-rules.pro'
        }
        debug {
            debuggable = true
            jniDebuggable true

        }
    }
}

dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:24.1.1'
    compile project(':openCVLibrary')
}
```

The key point is that you need to create a gradle task named *ndkBuild* manually to compile C++ code to **.so** files in the **jniLibs** folder:

```
task ndkBuild(type: Exec, description: 'Compile JNI source
via NDK') {
    def ndkDir = "/home/wenfahu/NVPACK/android-sdk-
linux/ndk-bundle"
    commandLine "$ndkDir/ndk-build",
            'NDK_PROJECT_PATH=build/intermediates/ndk',
            'NDK_LIBS_OUT=src/main/jniLibs',
            'APP_BUILD_SCRIPT=src/main/jni/Android.mk',
            'NDK_APPLICATION_MK=src/main/jni/Application.mk'
}
```

Based on the configuration of **Android.mk** above, the *ndkBuild* task would generate **libnative.so** in **src/main/jniLibs/<arch>/** directory**.**

**6.** In the Java Class file, you could load the library via

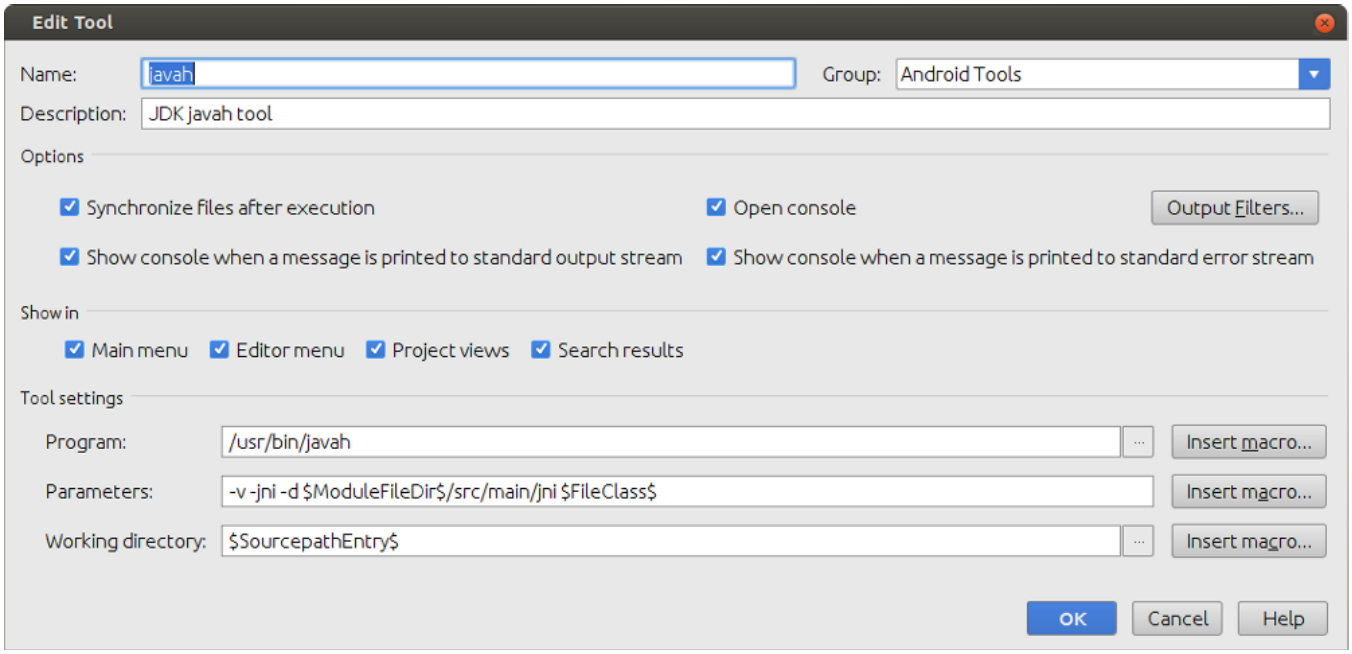
```
static {
    System.loadLibrary("native");
}
```

And you can declare a native method

```
public static native int test();
```

You could generate the declaration of JNI calls in the C++ headers files using *javah*. For convenience, you can just add *javah* as a external tool in Android Studio.

In your Android Studio, navigate to File/settings/Tools/External Tools . Add javah tool as follows.




The Tool Settings :

Program: the path of javah command on your machine. On a linux machine, you can get it by type *which javah* in the terminal after JDK installation. For example, mine is **/usr/bin/javah**

Parameters : the parameters of javah command. In my project, I set it to

```
-v -jni -d $ModuleFileDir$/src/main/jni $FileClass$
```

Working directory: **$SourcepathEntry$**

Now, after you declare a native method in the java class file, you may right click the class in the project panel of Android Studio, click **Android Tools/javah**. The corresponding header file would be generated in the jni folder with the declaration of jni calls.