标签：Android Studio 3.0 JNI NDK
转载请注明：http://blog.csdn.net/feather_wch/article/details/79548978

Android Stduio 3.0进行JNI和NDK开发

版本：2018/3/17-1

本文主要包括两方面的内容：

1. 如何在Android Studio 3.0中在新工程和老工程中进行JNI和NDK相关的支持
2. 如何在新工程和老工程中对JNI进行开发

Android Studio 3.0进行JNI/NDK开发时，如何让项目能支持JNI功能主要有两种方法。第一种是Android Stduio 3.0中新建工程时进行C++的支持。第二种是在老项目中添加C++的支持。

前提

下载 `NDK、LLDB、CMake` -位于 `settings->Android SDK->SDK Tools` 中

第一种：新建工程中选择C++支持

步骤：

1. 新建工程，New->New Project
2. 选择 `include C++ Support`
3. 一路默认选择，直到选择完 `finish`

支持C++相关的重要文件

- `app\src\main\cpp` 文件夹
  > 里面存放的 `native-lib.cpp` 就是系统默认的 `C++` 文件，用于编写 `native` 代码

- app/src/CMakeLists.txt

  > 生成so库的配置文件，描述了so库的名称，以及所依赖的具体源文件(如： native-lib.cpp)

```
# [CMakeLists.txt] 1. 设置构建native库所需要的CMake的最小版本
cmake_minimum_required(VERSION 3.4.1)

add_library(
             # 2. 设置库的名字
             native-lib

             # 3. 将库设置为共享型库(Gradle会自动将共享库打包到APK中)
             SHARED

             # 4. 提供源码文件的相对路径
             src/main/cpp/native-lib.cpp )

find_library(
              log-lib
              log )

target_link_libraries(
                       # 5. 目标库的名称
                       native-lib

                       # 6. 将目标库和NDK中的log库连接起来
                       ${log-lib} )
```

- build.gradle 文件新增配置内容

```
android {
    compileSdkVersion 26
    defaultConfig {
        ...
        //===增加如下内容===
        externalNativeBuild {
            cmake {
                cppFlags ""
            }
        }
    }
    buildTypes {
        ...
    }
    //===增加如下内容，指明CMake的配置文件===
    externalNativeBuild {
        cmake {
            path "CMakeLists.txt"
        }
    }
}

dependencies {
    ...
}
```

第二种：老项目中添加C++的支持

步骤

1. AS3.0左上角File->选择Link C++ project with Gradle
2. 选择 CMake
3. 将 方法一 生成的 CMakeLists.txt 复制到本工程的 app/src 目录下和 main 平级
4. Project Path 中选择复制好的 CMakeLists.txt 文件
5. 完成导入

在新老工程中如何进行C++开发

1、新建一个JniUtils2.java文件-编写需要native实现的Java类和方法

```
public class JniUtils2 {
    static {
        System.loadLibrary("native-lib"); //CMakeLists.txt中指定的仓库名
    }
    public native String getStringFromC();
}
```

2、生成 JniUtils2的native方法 对应的头文件

1. 进入 app/build/intermediates/classes/debug 目录
2. 执行命令 javah -d jni com.hao.jniapp.JniUtils2 也就是 Javah 包名.类名
3. 最终在 app/build/intermediates/classes/debug/jni 目录中生成头文件 com_hao_jniapp_JniUtils2.h
4. 将 头文件 移动到 cpp 目录中

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class com_hao_jniapp_JniUtils2 */

#ifndef _Included_com_hao_jniapp_JniUtils2
#define _Included_com_hao_jniapp_JniUtils2
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     com_hao_jniapp_JniUtils2
 * Method:    getStringFromC
 * Signature: ()Ljava/lang/String;
 */
JNIEXPORT jstring JNICALL Java_com_hao_jniapp_JniUtils2_getStringFromC
  (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

3、创建C++源代码 xxx.cpp (自定义，例如：abc.cpp或者com_haoe_jniapp_JniUtils2.cpp均可)

1. 实现 native方法

```
//com_haoe_jniapp_JniUtils2.cpp-拼写一定要正确！！！
#include "com_hao_jniapp_JniUtils2.h"

extern "C" {
JNIEXPORT jstring JNICALL Java_com_hao_jniapp_JniUtils2_getStringFromC
        (JNIEnv *env, jobject jobject1) {
    return env->NewStringUTF("我是来自JniUtils2的Native方法");
}
}
```

4、CMakeLists.txt将源码引入

```
cmake_minimum_required(VERSION 3.4.1)

add_library(
            #1. 库名不变
             native-lib
             SHARED

             src/main/cpp/native-lib.cpp
             src/main/cpp/main.cpp

            #2. 新增新功能cpp源码-拼写一定要正确！！！
             src/main/cpp/com_hao_jniapp_JniUtils2.cpp
             )

             //......
```

5、Clean Project

> 或者将 `app/build/intermediates/cmake` 中的debug和realease中所有so文件删除

6、Make Project

> 整个项目就都可以编译运行了，如果有CMake的错误，很可能是有用
> 到 `Java_com_hao_jniapp_JniUtils2_getStringFromC` 等手写的地方出现拼写错误。请按照顺序仔细排查。