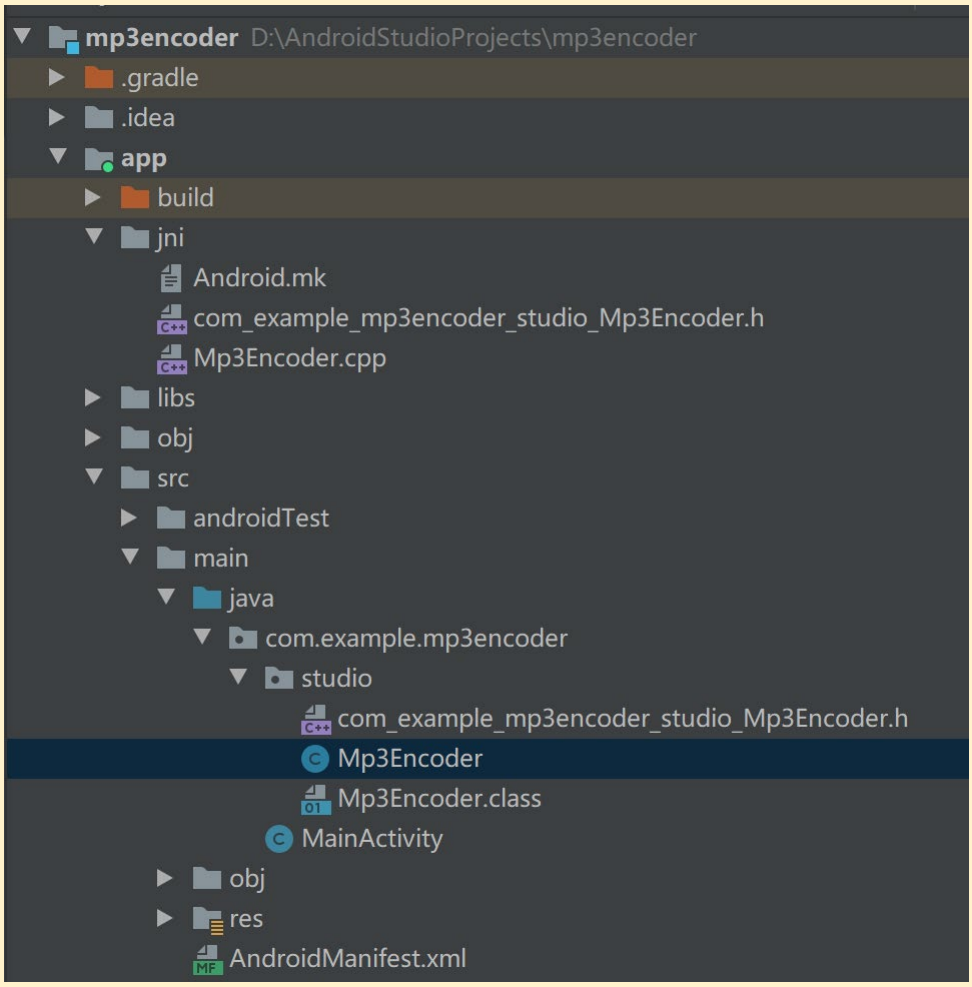
Jni 使用教程

下载 android studio，随便找一个教程配置好安卓环境

android studio 的 Settings 里面搜索 sdk，进入 Android SDK-》SDK Tools，勾选上 NDK，应用，等待安装，将安装目录加入环境变脸（ndk-build 所在的目录）。

新建一个项目，文件目录：



先写一个 java 类（Mp3Encoder）

```
package com.example.mp3encoder.studio;

public class Mp3Encoder {
    public native void encode();
}
```

在文件目录下运行 javac -h . .\Mp3Encoder.java

第一个点代表生成.h 文件在当前目录下。

新建 jni 文件夹位置，将.h 文件（那个名字很长的）移动到那里（如目录图）

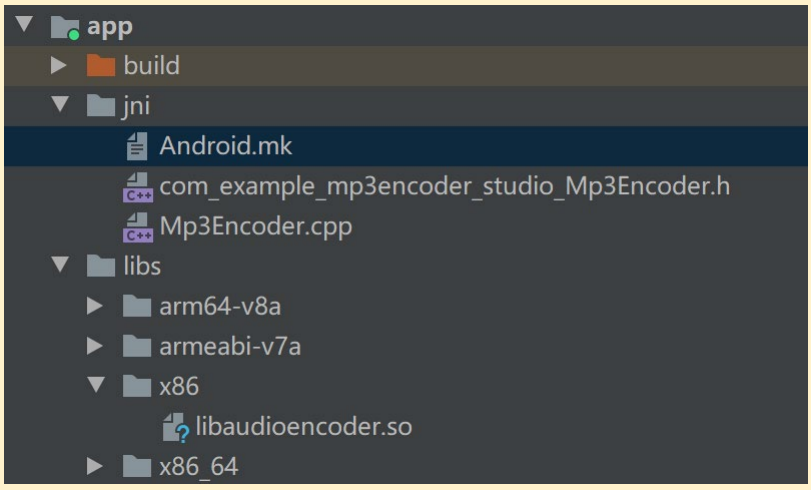
在 jni 文件夹里新建.cpp 文件实现方法

```
#include <com_example_mp3encoder_studio_Mp3Encoder.h>
//#include <android/log.h>

#define  LOG_TAG    "h264AndroidJNI"
#define LOGI(...) __android_log_print(ANDROID_LOG_INFO, LOG_TAG, __VA_ARGS__)


JNIEXPORT void JNICALL Java_com_example_mp3encoder_studio_Mp3Encoder_encode(JNIEnv *env, jobject obj){
    LOGI("encoder encode");
}
```

新建 mk 文件

```
Plugins supporting *.mk files found.
1  LOCAL_PATH := $(call my-dir)
2  APP_PLATFORM := android-8
3  include $(CLEAR_VARS)
4  LOCAL_SRC_FILES = ./Mp3Encoder.cpp
5  LOCAL_LDLIBS := -L$(SYSROOT)/user/lib -llog
6  LOCAL_MODULE := libaudioencoder
7  include $(BUILD_SHARED_LIBRARY)
```

控制台运行 ndk-build，会自动生成.so 文件



在 build.gradle 文件里，defaultConfig 内加上

```
multiDexEnabled true
ndk {
    abiFilters "arm64-v8a", "armeabi-v7a", "x86", "x86_64"
}
```

在 android 内加上

```
sourceSets {
    main {
        jniLibs.srcDirs = ['libs']
    }
}
```

进入 mainActivity，在构造函数前加上加载语句，就可以声明对象使用方法了

```
public class MainActivity extends AppCompatActivity {
    static{
        System.loadLibrary( libname: "audioencoder");
    }
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Mp3Encoder encoder = new Mp3Encoder();
        encoder.encode();
    }
}
```

我们愉快的可以在安卓 java 代码里调用 c 了，但是如何用 react native 实现有待进一步研究。目前了解关键词：JNI 智能指针

Ffmpeg 交叉编译
在 linux 下进行编译较为方便（虚拟机）

运行以下脚本以完成对 ffmpeg 的 makefile 文件的生成和编译工作。

```
#!/bin/bash

# 以下路径需要修改成自己的 NDK 目录

TOOLCHAIN=/root/ff/android-ndk-r22/toolchains/llvm/prebuilt/linux-x86_64

# 最低支持的 android sdk 版本

API=29

function build_android

{

# 打印

echo "Compiling FFmpeg for $CPU"
```

```
# 调用同级目录下的 configure 文件
./configure \
--prefix=$PREFIX \
--enable-neon \
--enable-hwaccels \
--enable-gpl \
--disable-postproc \
--enable-shared \
--enable-jni \
--enable-mediacodec \
--enable-decoder=h264_mediacodec \
--disable-static \
--disable-doc \
--disable-ffmpeg \
--disable-ffplay \
--disable-ffprobe \
--disable-avdevice \
--disable-doc \
--disable-symver \
--cross-prefix=$CROSS_PREFIX \
--target-os=android \
--arch=$ARCH \
--cpu=$CPU \
--cc=$CC \
--cxx=$CXX \
--enable-cross-compile \
--sysroot=$SYSROOT \
--extra-cflags="-Os -fpic $OPTIMIZE_CFLAGS" \
```

```
--extra-ldflags="$ADDI_LDFLAGS" \
$ADDITIONAL_CONFIGURE_FLAG
make clean
make -j16
make install
echo "The Compilation of FFmpeg for $CPU is completed"
}
#armv8-a
ARCH=arm64
CPU=armv8-a
# r21 版本的 ndk 中所有的编译器都在 /ndk/21.3.6528147/toolchains/llvm/prebuilt/darwin-x86_64/目录下（clang）
CC=$TOOLCHAIN/bin/aarch64-linux-android$API-clang
CXX=$TOOLCHAIN/bin/aarch64-linux-android$API-clang++
# NDK 头文件环境
SYSROOT=$TOOLCHAIN/sysroot
CROSS_PREFIX=$TOOLCHAIN/bin/aarch64-linux-android-
# so 输出路径
PREFIX=$(pwd)/android/$CPU
OPTIMIZE_CFLAGS="-march=$CPU"
build_android
#  交叉编译工具目录,对应关系如下
# armv8a -> arm64 -> aarch64-linux-android-
# armv7a -> arm -> arm-linux-androideabi-
# x86 -> x86 -> i686-linux-android-
# x86_64 -> x86_64 -> x86_64-linux-android-
# CPU 架构
#armv7-a
```

```
ARCH=arm
CPU=armv7-a
CC=$TOOLCHAIN/bin/armv7a-linux-androideabi$API-clang
CXX=$TOOLCHAIN/bin/armv7a-linux-androideabi$API-clang++
SYSROOT=$TOOLCHAIN/sysroot
CROSS_PREFIX=$TOOLCHAIN/bin/arm-linux-androideabi-
PREFIX=$(pwd)/android/$CPU
OPTIMIZE_CFLAGS="-mfloat-abi=softfp -mfpu=neon -marm -march=$CPU "
build_android
#x86
ARCH=x86
CPU=x86
CC=$TOOLCHAIN/bin/i686-linux-android$API-clang
CXX=$TOOLCHAIN/bin/i686-linux-android$API-clang++
SYSROOT=$TOOLCHAIN/sysroot
CROSS_PREFIX=$TOOLCHAIN/bin/i686-linux-android-
PREFIX=$(pwd)/android/$CPU
OPTIMIZE_CFLAGS="-march=i686 -mtune=intel -mssse3 -mfpmath=sse -
m32"
build_android
#x86_64
ARCH=x86_64
CPU=x86-64
CC=$TOOLCHAIN/bin/x86_64-linux-android$API-clang
CXX=$TOOLCHAIN/bin/x86_64-linux-android$API-clang++
SYSROOT=$TOOLCHAIN/sysroot
CROSS_PREFIX=$TOOLCHAIN/bin/x86_64-linux-android-
PREFIX=$(pwd)/android/$CPU
```

```
OPTIMIZE_CFLAGS="-march=$CPU -msse4.2 -mpopcnt -m64 -mtune=intel"
# 方法调用
build_android
```