

# 关于视频的编解码与传输技术，你想知道的都在这里

## 一、如何编译 FFmpeg

### 准备工作

- 下载 FFmpeg 源码：下载地址
- 下载 NDK：下载地址

下载后文件在 Mac 中的存放路径如下：

```
ChendeMacBook-Pro: compileFF
ChendeMacBook-Pro: compileFF chenzongwen$ pwd
/Users/chenzongwen/compileFF //文件的存放路径 NDK 与 FFmpeg 源码
```

### 如何编译 ffmpeg

首先进入 ffmpeg-3.0 文件夹，在文件夹中增加 编译脚本 ffmpegConfig 文件如下图：

```
ChendeMacBook-Pro:ffmpeg-3.0 chenzongwen$ ls
COPYING.GPLv2  RELEASE_NOTES  config.h      ffmpeg_filter.c  ffprobe.c
COPYING.GPLv3  VERSION       config.log    ffmpeg_filter.d  ffprobe.d  libpostproc
COPYING.LGPLv2.1 arch.mak      config.mak    ffmpeg_filter.o  ffprobe.o
COPYING.LGPLv3 cmdutils.c    configure     ffmpeg_g        ffprobe_g        libswresample
CREDITS        cmdutils.d    doc           ffmpeg_opt.c     ffserver.c  libswscale
Changelog      cmdutils.h    ffmpeg        ffmpeg_opt.d     ffserver_config.c  presets
INSTALL.md     cmdutils.o    ffmpeg.c      ffmpeg_opt.o     ffserver_config.h  tests
LICENSE.md     cmdutils_common_opts.h  ffmpeg.d      ffmpeg_qsv.c     libavcodec  tools
MAINTAINERS    cmdutils_openssl.c  ffmpeg.h      ffmpeg_vdpau.c   libavdevice
Makefile       common.mak      ffmpeg.o      ffmpeg_videotoolbox.c  libavfilter
README.md      compat          ffmpegConfig  ffplay.c         libavformat
RELEASE        config.fate      ffmpeg_dxva2.c  ffprobe         libavresample
ChendeMacBook-Pro:ffmpeg-3.0 chenzongwen$
```

ffmpegConfig 文件内容如下：

```
#!/bin/bash

NDK=/Users/chenzongwen/compileFF/android-ndk-r11b
export PATH=$PATH:$NDK
SYSROOT=$NDK/platforms/android-19/arch-arm/
TOOLCHAIN=$NDK/toolchains/arm-linux-androideabi-4.9/prebuilt/darwin-x86_64
function build_one
{
bash ./configure \
--prefix=$PREFIX \
--enable-shared \
--disable-static \
--disable-doc \
--disable-ffserver \
--enable-cross-compile \
--cross-prefix=$TOOLCHAIN/bin/arm-linux-androideabi- \
--target-os=linux \
--arch=arm \
--extra-libs=-lgcc \
--sysroot=$SYSROOT \
--enable-asm \
--enable-neon \
--extra-cflags="-O3 -fPIC $ADDI_CFLAGS" \
--extra-ldflags="$ADDI_LDFLAGS" \
$ADDITIONAL_CONFIGURE_FLAG
}
CPU=arm
PREFIX=$(pwd)/android/$CPU
ADDI_CFLAGS="-marm -mcpu=neon"
build_one
~
~
```

然后在当前文件夹下执行如下命令：

1. chenzongwen ./ffmpegConfig
2. chenzongwen make -j4 // 4 为用 4 个 CPU 进行编译
3. chenzongwen\$ make install

命令执行完之后会在当前文件夹下生成一个 android 文件夹 android/arm 文件夹中的内容如下几个文件夹：

```
bin
include
lib
share
```

include 中存放的是头文件，lib 中存放的是 so 文件 整个编译过程结束。

### 如何使用 FFmpeg

1. 在跟 android-ndk-r11b ffmpeg-3.0 同级的目录下 创建 jni 文件夹 执行如下命令：

```
mkdir jni
```

2. 将之前编译的头文件（在 include 文件夹下）拷贝到 jni 文件夹下 并且在 jni 文件夹下创建 prebuilt 文件夹 并将之前生成的 so（在 lib 文件夹下）拷贝到 prebuilt 文件夹下，拷贝完成后如下图：

```
ChendeMacBook-Pro:jni chenzongwen$ ls
libavcodec  libavdevice  libavfilter  libavformat  libavutil  libswresample  libswscale  prebuild
ChendeMacBook-Pro:jni chenzongwen$ ls prebuild/
libavcodec-57.so  libavfilter-6.so  libavutil-55.so  libswscale-4.so
libavdevice-57.so  libavformat-57.so  libswresample-2.so  libswscale.so
```

3. 调用 so 方法 在当前文件夹下添加两个文件 Android.mk 和 Application.mk 内容分别如下  
Android.mk 内容

```

LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := avcodec-56-prebuilt
LOCAL_SRC_FILES := prebuilt/libavcodec-57.so
include $(PREBUILT_SHARED_LIBRARY)

#include $(CLEAR_VARS)

#LOCAL_MODULE := avdevice-56-prebuilt

#LOCAL_SRC_FILES := prebuilt/libavdevice-57.so

#include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := avfilter-5-prebuilt
LOCAL_SRC_FILES := prebuilt/libavfilter-6.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := avformat-56-prebuilt
LOCAL_SRC_FILES := prebuilt/libavformat-57.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := avutil-54-prebuilt
LOCAL_SRC_FILES := prebuilt/libavutil-55.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := avswresample-1-prebuilt
LOCAL_SRC_FILES := prebuilt/libswresample-2.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := swscale-3-prebuilt
LOCAL_SRC_FILES := prebuilt/libswscale-4.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)

```

Application.mk 的内容如下：

```

ChendeMacBook-Pro:jni chenzongwen$ vim Application.mk
APP_ABI := armeabi

#APP_ABI := armeabi-v7a

APP_PLATFORM := android-10

```

编写调用文件在当前文件夹下创建 \*.c 文件 内容如下（文章结束会给出全部源码）：

```

JNIEXPORT jint JNICALL Java_tan_h264_FfmpegNative_decode_1file
(JNIEnv *env, jobject obj, jstring filePath) { // jni 方法的定义
//todo
}

JNIEXPORT jint JNICALL Java_tan_h264_FfmpegNative_decodeFrame
(JNIEnv *env, jobject obj, jbyteArray in, jint inSize)
{
//todo
}

```

开始编译 在 jni 目录下执行：

```

ChendeMacBook-Pro:jni chenzongwen$ ../android-ndk-r11b/ndk-build

```

最后将 如下目录下的so 拷贝到工程中就可以使用了。

```

ChendeMacBook-Pro:armeabi chenzongwen$ pwd
/Users/chenzongwen/compileFF/libs/armeabi
ChendeMacBook-Pro:armeabi chenzongwen$ ls
libavcodec-57.so  libavutil-55.so  libswscale-4.so
libavfilter-6.so  libowenchan_Test.so
libavformat-57.so  libswresample-2.so
ChendeMacBook-Pro:armeabi chenzongwen$

```

## 二、Android App 中如何调用 Ffmpeg so,jni 技术的讲解

java 层代码调用如下：

```

import android.graphics.Bitmap;

import java.nio.ByteBuffer;

public class FfmpegNative {
    static {
        System.loadLibrary("avutil-54");
        System.loadLibrary("swresample-1");
        System.loadLibrary("avcodec-56");
        System.loadLibrary("avformat-56");
        System.loadLibrary("swscale-3");
        System.loadLibrary("avfilter-5");
        System.loadLibrary("avdevice-56");
        System.loadLibrary("ffmpeg_codec");
    }

    public native int decode_init();

    public native int decode_file(String filePath);

    public native int decodeFrame(ByteBuffer in, int inSize);

    public native int decodeFrame2(ByteBuffer in, int inSize);

    public native int decodeFrameRGB(ByteBuffer in);
}

```

```
public native int copyFrameH264(ByteBuffer out);

public native int copyFrameYUV420p(ByteBuffer out);

public native int copyFrame2(ByteBuffer outY, ByteBuffer outU, ByteBuffer outV);
}
```

### 三、H264 格式

H.264分为两层

(一) H264 分为两层

1. 视频编码层 (VCL: Video Coding Layer): 进行视频编解码, 包括运动补偿预测, 变换编码和熵编码等功能;
2. 网络取层 (NAL: Network Abstraction Layer): 用于采用适当的格式对VCL 视频数据进行封装打包;VCL需要打包成NAL,才能用于传输或存储。

(二) 分层的目的

1. 可以定义VCL视频压缩处理与NAL网络传输机制的接口, 这样允许视频编码层VCL的设计可以在不同的处理器平台进行移植, 而与NAL层的数据封装格式无关;
2. VCL和NAL都被设计成工作于不同的传输环境, 异构的网络环境并不需要 对VCL比特流进行重构和重编码。

(三) NAL单元(NAL Unit)

H264基本码流由一系列的NALU组成, 组成结构如下

**SODE + RBSP trailing bits = RBSP**

**NAL header(1 byte) + RBSP = NALU**

**Start Code + NALU + Start Code + NALU + ... = H.264BitsStream**

- NALU: Coded H.264 data is stored or transmitted as a series of packets known as Network Abstraction LayerUnits. (NALU单元)
- RBSP: A NALU contains a Raw Byte Sequence Payload, a sequence of bytes containing syntax elements.(原始数据字节流)
- SODE: String Of Data Bits (原始数据比特流, 长度不一定是8的倍数, 需要补齐)

Start code

一共有两种起始码:3字节的0x000001和4字节的0x00000001;如果NALU对应的Slice为一帧的开始, 则用4字节表示, 即0x00000001; 否则用3字节0x000001表示, 就是一个完整的帧被编为多个slice的时候, 包含这些 slice的nalu使用3字节起始码。由于NAL的语法中没有给出长度信息, 实际的传输、存储系统需要增加额外的起始头实现各个NAL单元的界定。

先识别H264起始码0x00000001; 接着读取NALU的header 字节, 判断后 RBSP类型, 相应的 六进制类型定义如下:

```
0x67: SPS
0x68: PPS
0x65: IDR
0x61: non-IDR Slice 0x01: B Slice
0x06: SEI
0x09: AU Delimiter
```

从 读出的 个H.264视频帧以下的形式存在: 0000000167...SPS

```
0000000168... PPS
00 00 00 01 65 ... IDR Slice
```



剩下的几个部分是视频的传输压缩与解压, 我做分享的时候对着代码来分析。

代码下载地址

界面如下:



Start

Record

127.0.0.1

# GitChat