下面是课程大纲的介绍：

```
RTMP直播音频推流实战(NDK第40节课)

1.上节课回顾。
2.上层Audio编码工作。
3.FAAC交叉编译。
4.C++层音频编码器工作。
5.音频推流编码工作。

预习资料：同学们，暂无预习资料哦。
```

上节课39节课的预习资料

NV21与I420

Android Camera对象通过setPreviewCallback 函数，在onPreviewFrame(byte[] data,Camera camera)中回调采集的数据就是NV21格式。而x264编码的输入数据却为I420格式。

因此，当我们采集到摄像头数据之后需要将NV21转为I420。

NV21和I420都是属于YUV420格式。而NV21是一种two-plane模式，即Y和UV分为两个Plane(平面)，但是UV（CbCr）交错存储，2个平面，而不是分为三个。这种排列方式被称之为YUV420SP，而I420则称之为YUV420P。(Y:明亮度、灰度，UV:色度、饱和度)

下图是大小为4x4的NV21数据:Y1、Y2、Y5、Y6共用V1与U1,.....

而I420则是

y1	y2	y3	y4
y5	y6	y7	y8
y9	y10	y11	y12
y13	y14	y15	y16
u1	u2	u3	u4
v1	v2	v3	v4

可以看出无论是哪种排列方式，YUV420的数据量都为: w*h*+*w*/2h/2+w/2*h*/2 *即为*wh*3/2

将NV21转位I420则为：

Y数据按顺序完整复制,U数据则是从整个Y数据之后加一个字节再每隔一个字节取一次。

手机摄像头的图像数据来源于摄像头硬件的图像传感器，这个图像传感器被固定到手机上后会有一个默认的取景方向，这个取景方向坐标原点于手机横放时的左上角。当应用是横屏时候：图像传感器方向与屏幕自然方向原点一致。而当手机为竖屏时:

传感器与屏幕自然方向不一致，将图像传感器的坐标系逆时针旋转90度，才能显示到屏幕的坐标系上。所以看到的画面是逆时针旋转了90度的，因此我们需要将图像顺时针旋转90度才能看到正常的画面。而Camera对象提供一个setDisplayOrientation

接口能够设置预览显示的角度：

```
* public static void setCameraDisplayOrientation(Activity activity,
*         int cameraId, android.hardware.Camera camera) {
*     android.hardware.Camera.CameraInfo info =
*             new android.hardware.Camera.CameraInfo();
*     android.hardware.Camera.getCameraInfo(cameraId, info);
*     int rotation = activity.getWindowManager().getDefaultDisplay()
*             .getRotation();
*     int degrees = 0;
*     switch (rotation) {
*         case Surface.ROTATION_0: degrees = 0; break;
*         case Surface.ROTATION_90: degrees = 90; break;
*         case Surface.ROTATION_180: degrees = 180; break;
*         case Surface.ROTATION_270: degrees = 270; break;
*     }
*
*     int result;
*     if (info.facing == Camera.CameraInfo.CAMERA_FACING_FRONT) {
*         result = (info.orientation + degrees) % 360;
*         result = (360 - result) % 360;  // compensate the mirror
*     } else {  // back-facing
*         result = (info.orientation - degrees + 360) % 360;
*     }
*     camera.setDisplayOrientation(result);
* }
```

根据文档，配置完Camera之后预览确实正常了，但是在onPreviewFrame中回调获得的数据依然是逆时针旋转了90度的。所以如果需要使用预览回调的数据，还需要对onPreviewFrame回调的byte[] 进行旋转。

即对NV21数据顺时针旋转90度。

旋转前：

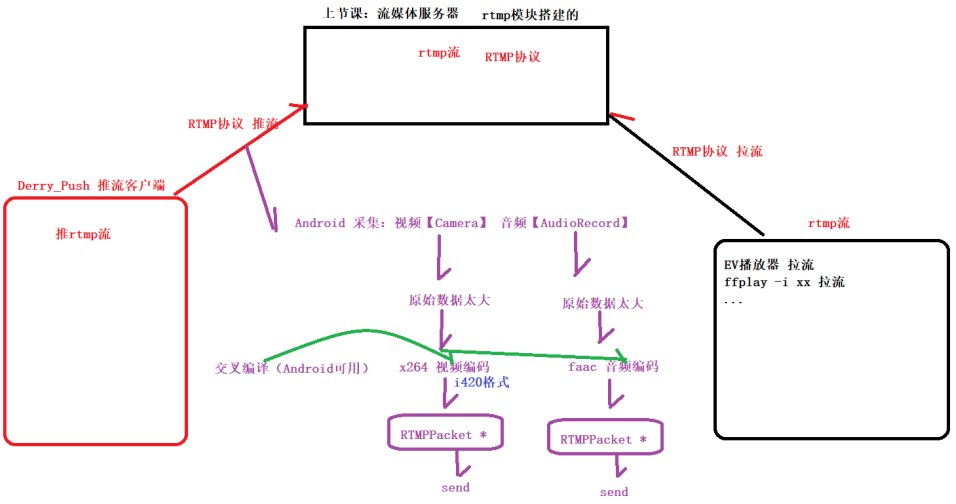
y1	y2	y3	y4
y5	y6	y7	y8
y9	y10	y11	y12
y13	y14	y15	y16
v1	u1	v2	u2
v3	u3	v4	u4

旋转后:

y13	u9	y5	y1
y14	y10	y6	y2
y15	y11	y7	y3
y16	y16	y8	y4
v3	u3	v1	u1
v4	u4	v2	u2

0.流程图&RTMP视频格式

01-流程细节图



02-RTMP视频数据格式

RTMP视频流数据的格式

一般情况下，组装的RTMPPacket(RTMPDump中的结构体)为：

关键帧	0x17	0x01	0x00	0x00	0x00	4字节数据长度	h264裸数据
非关键帧	0x27	0x01	0x00	0x00	0x00	4字节数据长度	h264裸数据
sps与pps	0x17	0x00	0x00	0x00	0x00	sps+pps数据	

类型	长度	说明
configurationVersion	1	0x01 版本
avcProfileIndication	1	sps[1] Prifile
profile_compatibility	1	sps[2] 兼容性
profile_level	1	sps[3] Profile Level
lengthSizeMinusOne	1	0xff 包长数据所使用的字节数,通常为oxf
numOfSequenceParameterSets	1	0xe1 SPS个数,通常为0xe1
sequenceParameterSetLength	2	sps长度
sequenceParameterSetNALUnits	sequenceParameterSetLength	sps内容
numOfPictureParameterSets	1	0x01 pps个数
pictureParameterSetLength	2	pps长度
pictureParameterSetNALUnits	pictureParameterSetLength	pps内容

03-Video Tag Data

字段	占位	描述
FrameType	4	帧类型。 1: keyframe(for AVC, a seekable frame) 2: inter frame(for AVC, a non-seekable frame) 3: disposable inter frame(H.263 only) 4: generated keyframe(reserved for server use only) 5: video info/command frame
CodecID	4	编码类型。 1: JPEG(目前未用到) 2: Sorenson H.263 3: Screen video 4: On2 VP6 5: On2 VP6 with alpha channel 6: Screen video version 2 7: AVC(高级视频编码)
VideoData	n	数据部分（AVC则需要参考下面AVCVIDEOPACKET部

AVCVIDEOPACKET

字段	字节	描述
AVCPacketType	1	0：AVC sequence header 1：AVC NALU 2：AVC end of sequence
CompositionTime	3	合成时间。 AVCPacketType==1，表示 合成时间(单位毫秒); 否则为0
data	n	如果AVCPacketType==0，数据部分为AVCDecoderConfigurationRecord； 如果AVCPacketType==1，数据部分为1个或多个NALU 如果AVCPacketType==2，数据部分为空

AVCDecoderConfigurationRecord

AVCDecoderConfigurationRecord 包含了H.264解码相关比较重要的sps和pps信息，再给AVC解码器送数据流之前一定要把sps和pps信息送出，否则的话解码器不能正常解码。而且在解码器stop之后再次start之前，如seek、快进快退状态切换等，都需要重新送一遍sps和pps的信息。AVCDecoderConfigurationRecord一般情况也是出现1次，也就是第一个video tag。

字段	字节	描述
版本	1	0x01，版本号为1
编码规格	3	sps[1]+sps[2]+sps[3]
NALU 的长度	1	0xFF，包长为 （0xFF& 3） + 1，也就是4字节表
SPS个数	1	0xE1，个数为0xE1 & 0x1F 也就是1
SPS长度	2	整个sps的长度
sps的内容	n	整个sps
pps个数	1	0x01，1个
pps长度	2	整个pps长度
pps内容	n	整个pps内容

04-RTMP音频数据格式

> RTMP音频流数据的格式
>
> 音频区域
>
> 音频其实类似于推送视频，第一个包总是包含sps和pps的音频序列包。
>
> 一般情况下，组装的音频AAC的RTMPPacket(RTMPDump中的结构体)为：

解码信息	0xAF	0x00	解码数据
数据	0xAF	0x01	音频数据

05-Audio Tag Data

```
AF双声道：
十六进制：0xAF ===>  二进制对比下图：1010 1111
1010 ==> 十进制等于10 ==> AAC
11 ==> 十进制等于3 ===> 3 = 44-kHz(对于AAC来说，该字段总是3)
1 ==> 十进制等于1 ===> 1 = snd16Bit(对于压缩过的音频来说，一般都是16bit)
1 ==> 十进制等于1 ===> 1 = sndStereo(对于AAC，总是1)双声道
```

```
AE单声道：
十六进制：0xAE ===>  二进制对比下图：1010 1110
1010 ==> 十进制等于10 ==> AAC
11 ==> 十进制等于3 ===> 3 = 44-kHz(对于AAC来说，该字段总是3)
1 ==> 十进制等于1 ===> 1 = snd16Bit(对于压缩过的音频来说，一般都是16bit)
0 ==> 十进制等于0 ===> 0 = sndMono单声道
```

与Video Tag类似。

字段	占位	描述
SoundFormat	4	音频数据格式。 值： 0 = Linear PCM, platform endian 1 = ADPCM 2 = MP3 3 = Linear PCM, little endian 4 = Nellymoser 16-kHz mono 5 = Nellymoser 8-kHz mono 6 = Nellymoser 7 = G.711 A-law logarithmic PCM 8 = G.711 mu-law logarithmic PCM 9 = reserved 10 = AAC 11 = Speex 14 = MP3 8-kHz 15 = Device-specific sound (7，8，14，15是内部
SoundRate	2	音频采样率。值： 0 = 5.5-kHz 1 = 11-kHz 2 = 22-kHz 3 = 44-kHz(对于AAC来说，该字段总是3)
SoundSize	1	采样长度。值： 0 = snd9Bit 1 = snd16Bit(对于压缩过的音频来说，一般都是16
SoundType	1	音频类型(单声道还是双声道)。值： 0 = sndMono 单声道 1 = sndStereo(对于AAC，总是1) 双声道
SoundData	n	音频数据部分（AAC则需要参考下面AACAUDIODA分）

AACAUDIODATA

字段	字节	描述
AACPacketType	1	0：AAC 序列头 1：AAC 数据
Data	n	如果AACPacketType==0参考下面的AudioSpecificConfig，如果AACPacketType==1，AAC原始音频数据

AudioSpecificConfig

字段	占位
audioObjectType	5
samplingFrequencyIndex	4
channelConfiguration	4
frameLengthFlag	1
dependsOnCoreCoder	1
extensionFlag	1

1.FAAC交叉编译

faac维基百科的详细解释：

https://zh.wikipedia.org/wiki/%E9%80%B2%E9%9A%8E%E9%9F%B3%E8%A8%8A%E7%B7%A8%E7%A2%BC

高级音频编码(Advanced Audio Coding)，出现于1997年，基于MPEG-2的音频编码技术，目的是取代MP3格式。2000年，MPEG-4标准出现后，AAC重新集成了其特性，为了区别于传统的MPEG-2 AAC又称为MPEG-4 AAC。相对于mp3，AAC格式的音质更佳，文件更小。

FAAC https://www.audiocoding.com/ (Freeware Advanced Audio Coder)。 这个网站现在无法访问了

下载页面 https://www.audiocoding.com/downloads.html（FAAD2 是解码库）。 这个网站现在无法访问了

http://faac.sourceforge.net/ 。 这个网站可以访问

下载FAAC编码库源码：

```
wget https://ayera.dl.sourceforge.net/project/faac/faac-src/faac-1.29/faac-1.
如果上面的wget，下载失败，我还有办法，如下：
https://sourceforge.net/projects/faac/files/faac-src/faac-1.29/faac-1.29.9.2.
上面的链接，会在Windows自动下载中，然后你只需要 ”复制下载链接“
wget https://udomain.dl.sourceforge.net/project/faac/faac-src/faac-1.29/faac-
```

解压：

```
tar zxvf faac-1.29.9.2.tar.gz
cd faac-1.29.9.2
```

编译阶段：

```
ls
cat README 【有兴趣可以看看】
sh configure --help 【这个是帮助信息，可以看看】
```

编写脚本：

```
chmod 777 build_faac.sh
sh build_faac.sh
cd android/armeabi-v7a/lib/  【检查编译后成果】
android]# zip -r faac.zip * 【打成zip包】
sz faac.zip 【导出到Windows】
```

build_faac.sh脚本如下：

```
#!/bin/bash
```

```
2
3  # 同学们：必须是自己服务器上的 NDK路径哦
4  NDK_ROOT=/root/DerryAll/Tools/android-ndk-r17c
5
6  # 这个是最终输出成功的路径
7  PREFIX=`pwd`/android/armeabi-v7a
8
9  TOOLCHAIN=$NDK_ROOT/toolchains/arm-linux-androideabi-4.9/prebuilt/linux-x86_6
10 CROSS_COMPILE=$TOOLCHAIN/bin/arm-linux-androideabi
11
12 FLAGS="-isysroot $NDK_ROOT/sysroot -isystem $NDK_ROOT/sysroot/usr/include/arm
13
14 export CC="$CROSS_COMPILE-gcc --sysroot=$NDK_ROOT/platforms/android-17/arch-a
15 export CFLAGS="$FLAGS"
16
17
18 ./configure \
19 --prefix=$PREFIX \
20 --host=arm-linux \
21 --with-pic \
22 --enable-shared=no
23
24 make clean
25 make install
26
```