目录

[1. H264编码框架 2](#_Toc1443)

[2. H264编码原理 2](#_Toc24674)

[3. H264码流分析 3](#_Toc24284)

[3.1 H264编码格式 3](#_Toc19963)

[3.2 NAL Header 3](#_Toc15588)

[3.3 H264的传输 4](#_Toc1930)

[3.4 H264码流结构 5](#_Toc15592)

[3.5 H264的Level和Profile说明 6](#_Toc2703)

[4. 发送视频数据fireVideo流程解析 7](#_Toc3912)

[5. 添加H264 header信息解析 8](#_Toc21317)

[6.添加H264 body信息解析 10](#_Toc32580)

[7. AAC音频格式 11](#_Toc2070)

[8. 发送视频数据fireAudio流程解析 13](#_Toc10528)

[9. 添加AAC Header解析 13](#_Toc11697)

[10. 添加AAC Body解析 14](#_Toc23140)

[11. RTMP协议介绍 15](#_Toc4292)

[12. 利用RTMP传输音视频数据解析 17](#_Toc13605)

## 1. H264编码框架

H264码流文件分为两层：

(1) VCL(Video Coding Layer)视频编码层：负责高效的视频内容表示，VCL 数据即编码处理的输出，它表示被压缩编码后的视频数据序列。

(2) NAL(Network Abstraction Layer)网络提取层：负责以网络所要求的恰当的方式对数据进行打包和传送，是传输层，不管是在本地播放还是在网络播放的传输，都要通过这一层来传输。

## 2. H264编码原理

在H264协议里定义了三种帧，完整编码的帧叫I帧，参考之前的I帧生成的只包含差异部分编码的帧叫P帧，还有一种参考前后的帧编码的帧叫B帧。

在H264中图像以序列为单位进行组织，一个序列是一段图像编码后的数据流，以I帧开始，到下一个I帧结束。一个序列的第一个图像叫做 IDR 图像（立即刷新图像），IDR 图像都是 I 帧图像。

以下是三种帧的说明：

(1) I帧

I帧:帧内编码帧 ，I帧表示关键帧，你可以理解为这一帧画面的完整保留；解码时只需要本帧数据就可以完成（因为包含完整画面）

I帧特点:

1)它是一个全帧压缩编码帧。它将全帧图像信息进行JPEG压缩编码及传输;

2)解码时仅用I帧的数据就可重构完整图像;

3)I帧描述了图像背景和运动主体的详情;

4)I帧不需要参考其他画面而生成;

5)I帧是P帧和B帧的参考帧(其质量直接影响到同组中以后各帧的质量);

6)I帧是帧组GOP的基础帧(第一帧),在一组中只有一个I帧;

7)I帧不需要考虑运动矢量;

8)I帧所占数据的信息量比较大。

(2) P帧

P帧:前向预测编码帧。P帧表示的是这一帧跟之前的一个关键帧（或P帧）的差别，解码时需要用之前缓存的画面叠加上本帧定义的差别，生成最终画面。（也就是差别帧，P帧没有完整画面数据，只有与前一帧的画面差别的数据）

P帧的预测与重构:P帧是以I帧为参考帧,在I帧中找出P帧“某点”的预测值和运动矢量,取预测差值和运动矢量一起传送。在接收端根据运动矢量从I帧中找出P帧“某点”的预测值并与差值相加以得到P帧“某点”样值,从而可得到完整的P帧。

P帧特点:

1)P帧是I帧后面相隔1~2帧的编码帧;

2)P帧采用运动补偿的方法传送它与前面的I或P帧的差值及运动矢量(预测误差);

3)解码时必须将I帧中的预测值与预测误差求和后才能重构完整的P帧图像;

4)P帧属于前向预测的帧间编码。它只参考前面最靠近它的I帧或P帧;

5)P帧可以是其后面P帧的参考帧,也可以是其前后的B帧的参考帧;

6)由于P帧是参考帧,它可能造成解码错误的扩散;

7)由于是差值传送,P帧的压缩比较高。

(3) B帧

B帧:双向预测内插编码帧。B帧是双向差别帧，也就是B帧记录的是本帧与前后帧的差别（具体比较复杂，有4种情况，但我这样说简单些），换言之，要解码B帧，不仅要取得之前的缓存画面，还要解码之后的画面，通过前后画面的与本帧数据的叠加取得最终的画面。B帧压缩率高，但是解码时CPU会比较累。

B帧的预测与重构

B帧以前面的I或P帧和后面的P帧为参考帧,“找出”B帧“某点”的预测值和两个运动矢量,并取预测差值和运动矢量传送。接收端根据运动矢量在两个参考帧中“找出(算出)”预测值并与差值求和,得到B帧“某点”样值,从而可得到完整的B帧。

B帧特点

1）B帧是由前面的I或P帧和后面的P帧来进行预测的;

2）B帧传送的是它与前面的I或P帧和后面的P帧之间的预测误差及运动矢量;

3）B帧是双向预测编码帧;

4）B帧压缩比最高,因为它只反映两参考帧间运动主体的变化情况,预测比较准确;

5）B帧不是参考帧,不会造成解码错误的扩散。

## 3. H264码流分析

### 3.1 H264编码格式

在 VCL 数据传输或存储之前，这些编码的 VCL 数据，先被映射或封装进NAL 单元中。每个 NAL 单元包括一个原始字节序列负荷（ RBSP, Raw Byte Sequence Payload）和一组对应于视频编码的 NAL 头信息。RBSP 的基本结构是：在原始编码数据的后面填加了结尾比特。一个 bit“1”若干比特“0”，以便字节对齐。



图1 NAL单元序列

### 3.2 NAL Header

NAL头由一个字节组成 ，如图2所示。

语法：禁止位(1bit)、重要性位(2bit)、NALU类型(5bit)。

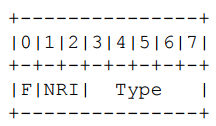


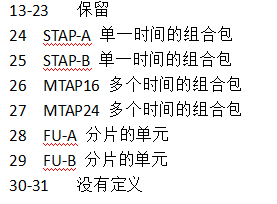
图2 NAL Header

NAL头信息的每一位说明如表1所示。

表1 NAL Header说明

|  |  |  |  |
| --- | --- | --- | --- |
| 位 | 0 | 1-2 | 3-7 |
| 简称 | F | NRI | TYPE |
| 全称 | forbidden\_zero\_bit | nal\_ref\_idc | nal\_unit\_type |
| 中文 | 禁止位 | 重要性指示位 | NALU类型 |
| 作用 | 网络发现NAL单元有比特错误时可设置该比特为1，以便接收方丢掉该单元 | 标志该NAL单元用于重建时的重要性，值越大，越重要。取 00 ~ 11 | 1 ~ 23表示单个NAL包，24 ~ 31需要分包或者组合发送，具体含义需要参考下面的表格 |

nal\_unit\_type取值的含义如下：

0 没有定义

1-23 NAL单元，单个 NAL 单元包

1 不分区，非IDR图像的片

2 片分区A

3 片分区B

4 片分区C

5 IDR图像中的片

6 补充增强信息单元（SEI）

7 SPS（Sequence Parameter Set序列参数集，作用于一串连续的视频图像，即视频序列）

8 PPS（Picture Parameter Set图像参数集，作用于视频序列中的一个或多个图像）

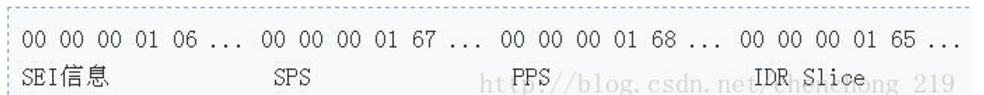
9 序列结束

10 序列结束

11 码流结束

12 填充

举例如下：



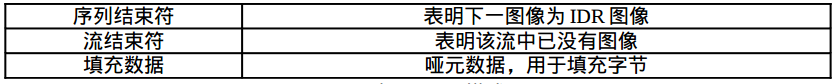
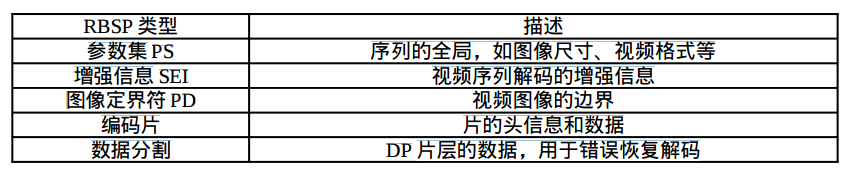
### 3.3 H264的传输

H.264 的编码视频序列包括一系列的NAL单元，每个NAL单元包含一个 RBSP。典型的 RBSP 单元序列如图3所示。每个单元都按独立的 NAL 单元传送。单元的信息头（一个字节）定义了 RBSP 单元的类型，NAL 单元的其余部分为 RBSP 数据。



图3 RBSP序列举例

RBSP各部分的解释如表2所示。

表2 RBSP描述

### 3.4 H264码流结构

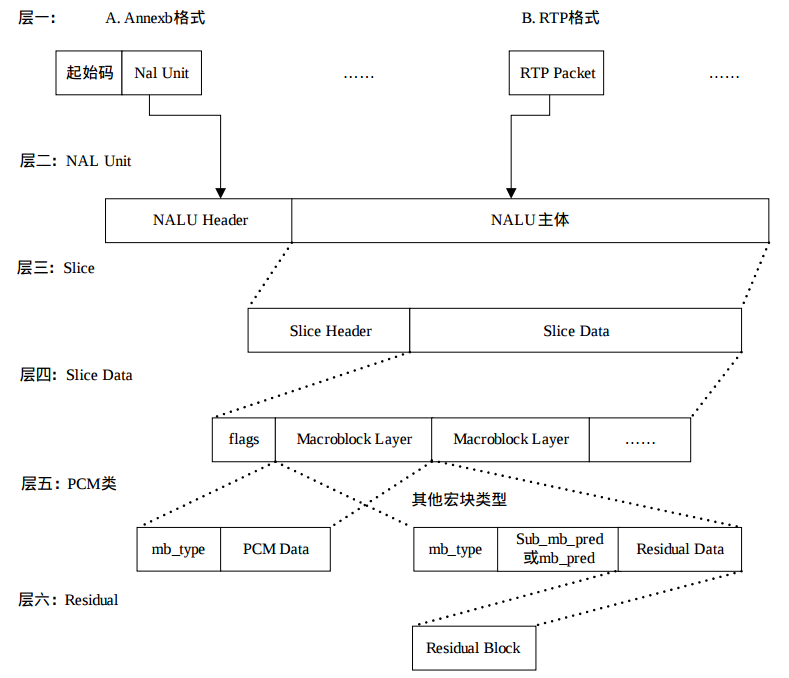


图4 H264码流分层结构

图4为H264码流分层结构，解释如下。

第一层：比特流。该层有两种格式：附录B格式和RTP格式。

第二层：NAL Unit层。包含了NAL Header和NAL Body信息。

第三层：Slice层。

一帧视频图像可编码成一个或者多个片，每片包含整数个宏块，即每片至少 一个宏块，最多时包含整个图像的宏块。

片的目的：为了限制误码的扩散和传输，使编码片相互间保持独立。片共有 5 种类型： I 片（只包含 I 宏块）、P 片（P 和 I 宏块）、B 片（B 和 I 宏块）、SP 片（用于不同编码流之 间的切换）和 SI 片（特殊类型的编码宏块）。

片的语法结构（如图5所示）：片头规定了片的类型、属于哪个图像、有关的参考图像等；片的数据包含了一系列宏块和不编码数据。

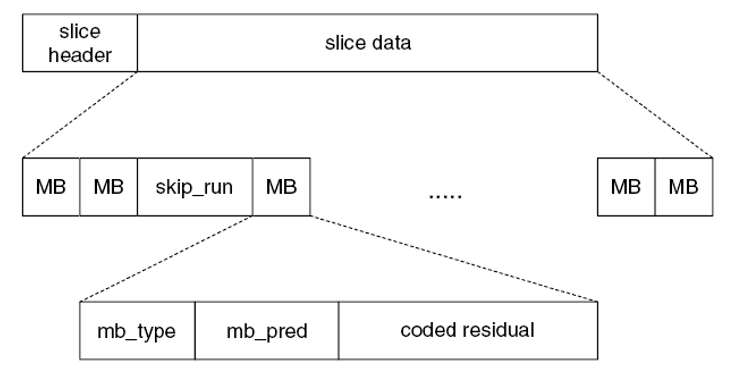


图5 Slice的语法结构

第四层：Slice data层。Slice由宏块（macro block, MB）组成。宏块是编码处理的基本单元。

第五层：PCM类。

第六层：残差层。

### 3.5 H264的Level（级别）和Profile（档次）说明

H264的Level（级别）是用来约束分辨率、帧率和码率的。具体请参照表3。

表3 Level的简要说明

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Level | Max macroblocks | | Max video bit rate (kbit/s) | | | | Examples for high resolution @ frame rate (max stored frames) |
| per second | per frame | BP, XP, MP | HiP | Hi10P | Hi422P, Hi444PP |
| 1 | 1,485 | 99 | 64 | 80 | 192 | 256 | 128×96@30.9 (8) 176×144@15.0 (4) |
| 1b | 1,485 | 99 | 128 | 160 | 384 | 512 | 128×96@30.9 (8) 176×144@15.0 (4) |
| 1.1 | 3,000 | 396 | 192 | 240 | 576 | 768 | 176×144@30.3 (9) 320×240@10.0 (3) 352×288@7.5 (2) |
| 1.2 | 6,000 | 396 | 384 | 480 | 1,152 | 1,536 | 320×240@20.0 (7) 352×288@15.2 (6) |
| 1.3 | 11,880 | 396 | 768 | 960 | 2,304 | 3,072 | 320×240@36.0 (7) 352×288@30.0 (6) |
| 2 | 11,880 | 396 | 2,000 | 2,500 | 6,000 | 8,000 | 320×240@36.0 (7) 352×288@30.0 (6) |
| 2.1 | 19,800 | 792 | 4,000 | 5,000 | 12,000 | 16,000 | 352×480@30.0 (7) 352×576@25.0 (6) |
| 2.2 | 20,250 | 1,620 | 4,000 | 5,000 | 12,000 | 16,000 | 352×480@30.7(10) 352×576@25.6 (7) 720×480@15.0 (6) 720×576@12.5 (5) |
| 3 | 40,500 | 1,620 | 10,000 | 12,500 | 30,000 | 40,000 | 352×480@61.4 (12) 352×576@51.1 (10) 720×480@30.0 (6) 720×576@25.0 (5) |
| 3.1 | 108,000 | 3,600 | 14,000 | 17,500 | 42,000 | 56,000 | 720×480@80.0 (13) 720×576@66.7 (11) 1280×720@30.0 (5) |
| 3.2 | 216,000 | 5,120 | 20,000 | 25,000 | 60,000 | 80,000 | 1,280×720@60.0 (5) 1,280×1,024@42.2 (4) |
| 4 | 245,760 | 8,192 | 20,000 | 25,000 | 60,000 | 80,000 | 1,280×720@68.3 (9) 1,920×1,080@30.1 (4) 2,048×1,024@30.0 (4) |
| 4.1 | 245,760 | 8,192 | 50,000 | 62,500 | 150,000 | 200,000 | 1,280×720@68.3 (9) 1,920×1,080@30.1 (4) 2,048×1,024@30.0 (4) |
| 4.2 | 522,240 | 8,704 | 50,000 | 62,500 | 150,000 | 200,000 | 1,920×1,080@64.0 (4) 2,048×1,080@60.0 (4) |
| 5 | 589,824 | 22,080 | 135,000 | 168,750 | 405,000 | 540,000 | 1,920×1,080@72.3 (13) 2,048×1,024@72.0 (13) 2,048×1,080@67.8 (12) 2,560×1,920@30.7 (5) 3,680×1,536@26.7 (5) |
| 5.1 | 983,040 | 36,864 | 240,000 | 300,000 | 720,000 | 960,000 | 1,920×1,080@120.5 (16) 4,096×2,048@30.0 (5) 4,096×2,304@26.7 (5) |

Max macroblocks：最大宏块数。注：宏块尺寸是16x16的。

　　per second：每秒（的最大宏块数）。可用于约束帧率。

　　per frame：每帧（的最大宏块数）。可用于约束分辨率。

Max video bit rate (kbit/s)：最大视频码率。不同档次（Profile）下会有区别。

　　BP：Baseline Profile，基线档次。

　　XP：Extended Profile，进阶档次。

　　MP：Main Profile，主要档次。

　　HiP：High Profile，高级档次。

　　Hi10P：High 10 Profile，高级10位档次。

　　Hi422P：High 4:2:2 Profile，高级4:2:2档次。

　　Hi444PP：High 4:4:4 Predictive Profile，高级4:4:4（实验性？）档次。

H.264 主要Profile 说明：

* BP(Baseline profile)：提供I/P帧，仅支持Progressive(逐行扫描)和CAVLC。多应用于“视频会话”，如可视电话、会议电视、远程教学、视频监控等实时通信领域；
* XP(Extended profile)：提供I/P/B/SP/SI帧，仅支持Progressive和CAVLC。多应用于流媒体领域，如视频点播、基于网络的视频监控等；
* MP(Main profile)：提供I/P/B帧，支持Progressive和Interlaced(隔行扫描），提供CAVLC和CABAC。多应用于数字电视广播、数字视频存储等领域；
* HiP(High profile)：（Fidelity Range Extensions,FRExt)在Main profile基础上新增8\*8帧内预测，Custom Quant，Lossless Video Coding,更多YUV格式(4:2:2,4:4:4)，像素精度提高到10位或14位。多应用于对高分辨率和高清晰度有特别要求的领域。

## 4. 发送视频数据fireVideo流程解析

作用：将相机采集到的视频数据使用x264编码，并加入到队列中

1) 首先将安卓Camera采集到的视频NV21数据转码成YUV420给到H264进行编码。

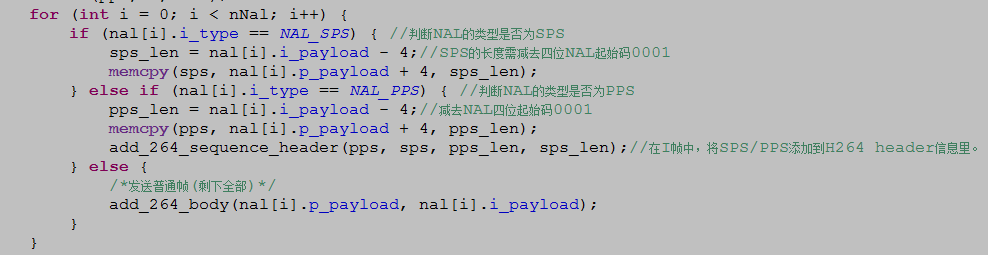
NV21与YUV420p都属于YUV420格式,每四个Y共用一组UV分量。区别是UV分量的空间排列不同。

NV21的颜色空间排列 ：YYYYYYYY VUVU

YUV420p的颜色空间排列：YYYYYYYY UVUV  
NV21转YUV420p的公式：(Y不变)Y=Y，U=Y+1+1，V=Y+1



1. 使用x264\_encoder\_encode函数对视频进行H264编码，编码成功后，复制SPS和PPS放在每个关键帧的前面，让每个关键帧(I帧)都附带SPS或PPS。



## 5. 添加H264 header信息解析

如果packet头中的TagType==9时，就表示这个TAG是video。那么StreamID之后的数据就表示是VideoTagHeader。VideoTagHeader只有一个字节，也就是接跟着StreamID的1个字节包含着视频帧类型及视频CodecID最基本信息。

(1) FrameType，4bit，帧类型

1 = key frame (for AVC, a seekable frame)

2 = inter frame (for AVC, a non-seekable frame)

3 = disposable inter frame (H.263 only)

4 = generated key frame (reserved for server use only)

5 = video info/command frame

H264的一般为1或者2.

(2)CodecID ，4bit，编码类型

1 = JPEG(currently unused)

2 = Sorenson H.263

3 = Screen video

4 = On2 VP6

5 = On2 VP6 with alpha channel

6 = Screen video version 2

7 = AVC

VideoTagHeader之后跟着的就是VIDEODATA数据了，也就是video payload。如果视频的格式是AVC（H.264）的话，VideoTagHeader会多出4个字节的信息，全部置为0。

接下来设置AVCDecoderConfigurationRecord，它包含的是H.264解码相关比较重要的SPS和PPS信息，在给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在FLV文件中一般情况也是出现1次，也就是第一个 video tag。

(3) AVCPacketType 8bit

IF AVCPacketType == 0 AVCDecoderConfigurationRecord（AVC sequence header）(此时FrameType必为1)

IF AVCPacketType == 1 One or more NALUs (Full frames are required)

IF AVCPacketType == 2 AVC end of sequence (lower level NALU sequence ender is not required or supported)

Composition Time，24bit

(4) AVCDecoderConfigurationRecord（AVCPacketType == 0，FrameType==1）

1.configurationVersion，8bit

2.AVCProfileIndication，8bit

3.profile\_compatibility，8bit

4.AVCLevelIndication，8bit

5.lengthSizeMinusOne，8bit

H.264 视频中 NALU 的长度，计算方法是 1 + (lengthSizeMinusOne & 3),实际测试时发现总为ff，计算结果为4.

6.numOfSequenceParameterSets，8bit

SPS 的个数，计算方法是 numOfSequenceParameterSets & 0x1F,实际测试时发现总为E1，计算结果为1

7.sequenceParameterSetLength，16bit

SPS 的长度

8.sequenceParameterSetNALUnits ，sps。

长度为sequenceParameterSetLength。

9.numOfPictureParameterSets，8bit

PPS 的个数,计算方法是 numOfPictureParameterSets & 0x1F,实际测试时发现总为E1，计算结果为1。

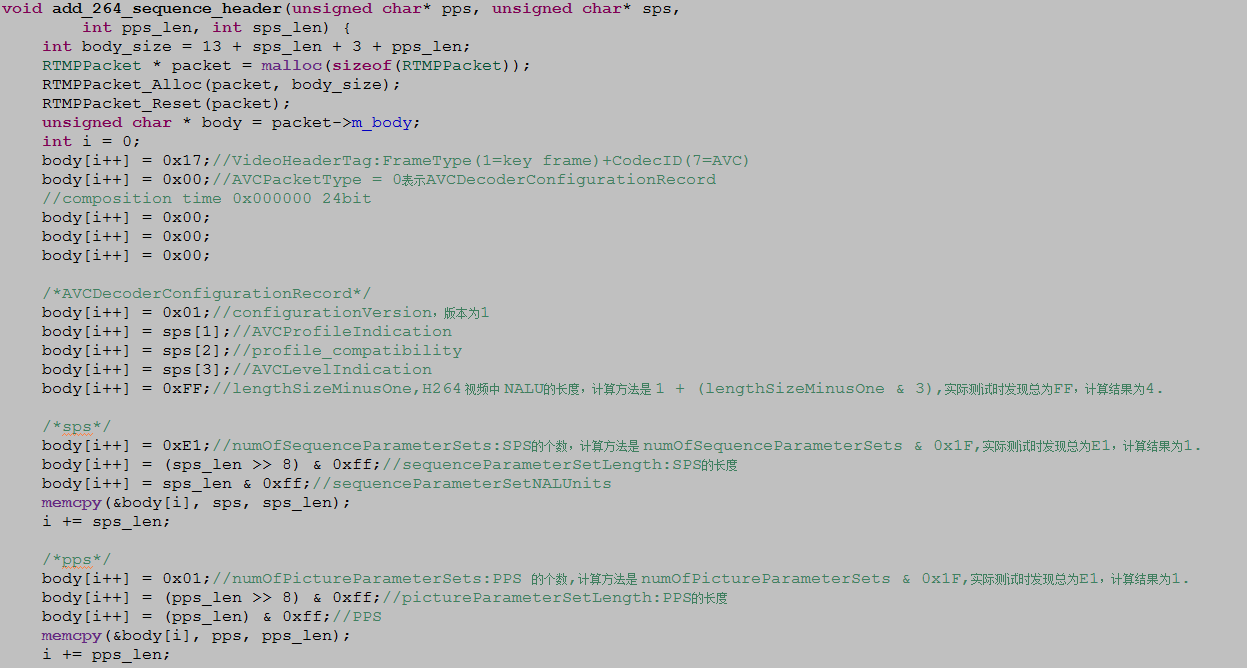
10.pictureParameterSetLength，16bit。

PPS的长度。

11.PPS

长度为pictureParameterSetLength。

具体说明如下所示：



## 6.添加H264 body信息解析

NALUs（AVCPacketType == 1，**FrameType==1或者2**）

一个packet里面可能包含多个NALUs，每个NALUs前的4个字节都是标示这个NALU长度的4个字节，然后是这个NALU包。一个NALU包结束之后，接着时下一个NALU包的长度4字节，然后是下一个NALU包。以此类推，直到这个packet结束。

1.nal\_length，32bit。每个nal包长度

每个NALU包前面都有（lengthSizeMinusOne & 3）+1个字节的NAL包长度描述,前面计算结果为4个字节。

2.nal包

这里插入一点NALU的小知识，每个NALU第一个字节的前5位标明的是该NAL包的类型，即NAL nal\_unit\_type

具体说明如下所示：



### AAC音频格式

AAC音频格式有ADIF和ADTS两种：

ADIF：Audio Data Interchange Format 音频数据交换格式。这种格式的特征是可以确定的找到这个音频数据的开始，不需进行在音频数据流中间开始的解码，即它的解码必须在明确定义的开始处进行。故这种格式常用在磁盘文件中。

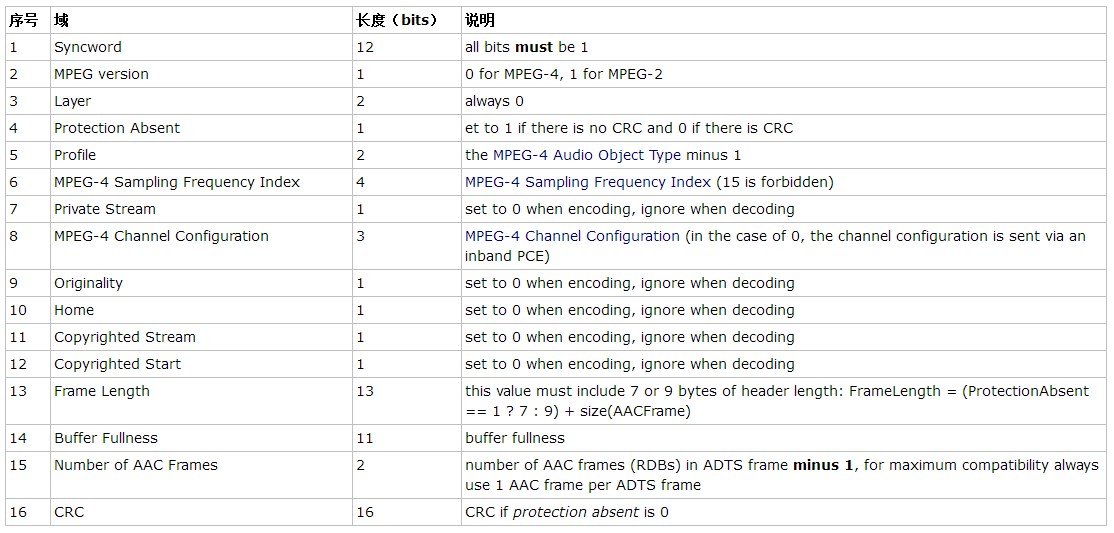
ADTS：Audio Data Transport Stream 音频数据传输流。这种格式的特征是它是一个有同步字的比特流，解码可以在这个流中任何位置开始。它的特征类似于mp3数据流格式。

两者的区别：ADTS可以在任意帧解码，也就是说它每一帧都有头信息。ADIF只有一个统一的头，所以必须得到所有的数据后解码。且这两种的header的格式也是不同的，目前一般编码后的和抽取出的都是ADTS格式的音频流。

ADTS是帧序列，本身具备流特征，在音频流的传输与处理方面更加合适。ADST帧结构如下：



ADTS帧Header结构如下：

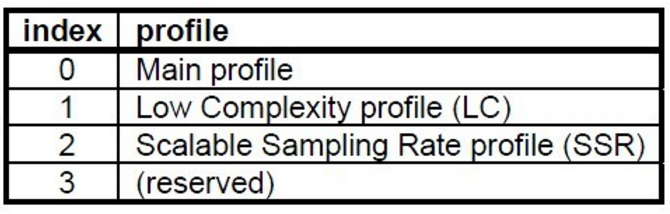


**syncword**：同步头 总是0xFFF, all bits must be 1，代表着一个ADTS帧的开始

**ID**：MPEG Version: 0 for MPEG-4, 1 for MPEG-2

**Layer**：always: '00'

**profile**：表示使用哪个级别的AAC，有些芯片只支持AAC LC 。在MPEG-2 AAC中定义了3种：



**sampling\_frequency\_index**：表示使用的采样率下标，通过这个下标在 **Sampling Frequencies[ ]**数组中查找得知采样率的值。

* 0: 96000 Hz
* 1: 88200 Hz
* 2: 64000 Hz
* 3: 48000 Hz
* 4: 44100 Hz
* 5: 32000 Hz

**channel\_configuration**: 表示声道数

* 0: Defined in AOT Specifc Config
* 1: 1 channel: front-center
* 2: 2 channels: front-left, front-right
* 3: 3 channels: front-center, front-left, front-right
* 4: 4 channels: front-center, front-left, front-right, back-center
* 5: 5 channels: front-center, front-left, front-right, back-left, back-right

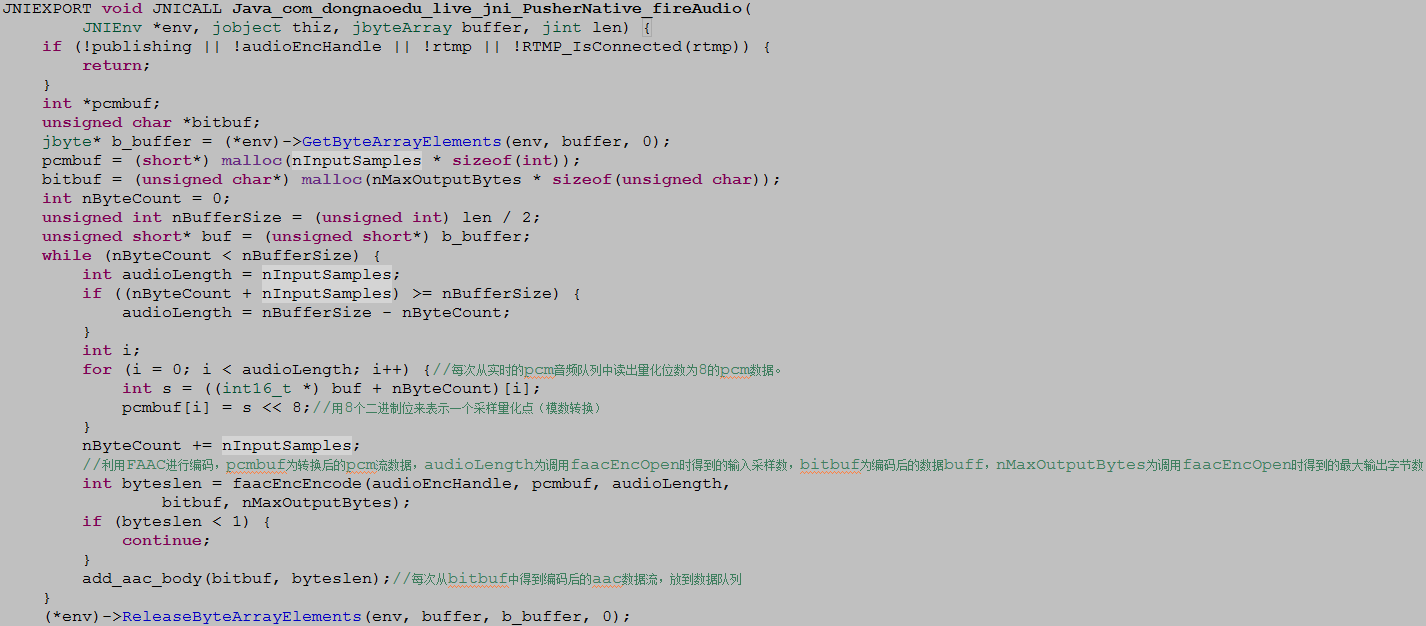
**frame\_length** : 一个ADTS帧的长度包括ADTS头和AAC原始流.

**adts\_buffer\_fullness：**0x7FF 说明是码率可变的码流

### 8. 发送视频数据fireAudio流程解析

fireAudio将麦克风采集到的音频数据使用faac编码，并加入到队列中。

每次从实时的pcm音频队列中读出量化位数为8的pcm数据，用8个二进制位来表示一个采样量化点（模数转换）。然后利用FAAC进行编码，pcmbuf为转换后的pcm流数据，audioLength为调用faacEncOpen时得到的输入采样数，bitbuf为编码后的数据buff，nMaxOutputBytes为调用faacEncOpen时得到的最大输出字节数。从bitbuf中得到编码后的aac数据流，放到数据队列。



### 9. 添加AAC Header解析

AAC header结构（2个字节）

1、SoundFormat，4bit

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

2、SoundRate，2bit，抽样频率

0 = 5.5 kHz

1 = 11 kHz

2 = 22 kHz

3 = 44 kHz

对于AAC音频来说，总是0x11，即44khz.

3、SoundSize，1bit，音频的位数。

0 = 8-bit samples

1 = 16-bit samples

AAC总是为0x01,16位。

4、SoundType，1bit，声道

0 = Mono sound

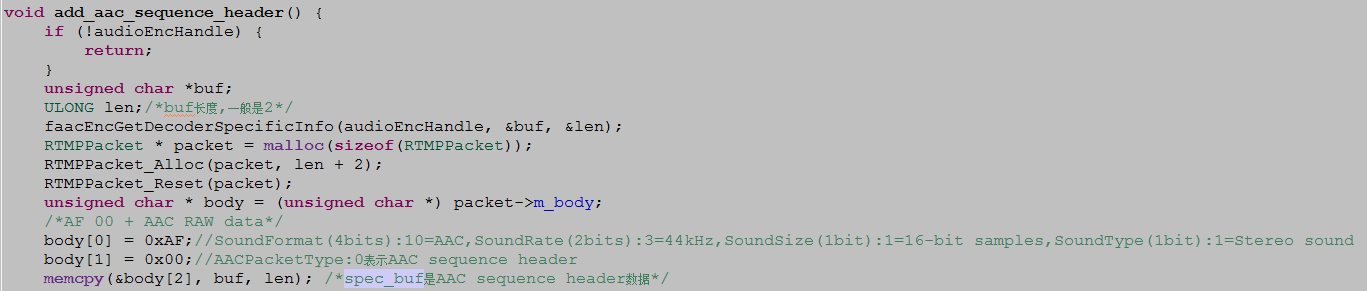
1 = Stereo sound

5、AACPacketType，8bit。

这个字段来表示AACAUDIODATA的类型：0 = AAC sequence header，1 = AAC raw。第一个音频包用0，后面的都用1。

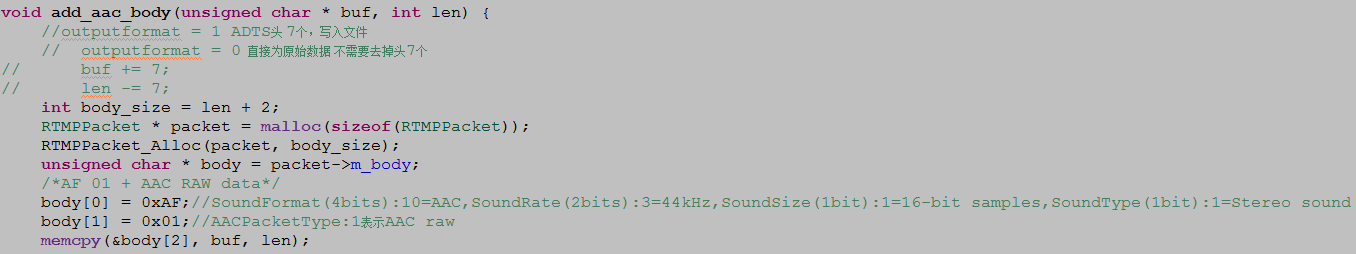
body[0] = 0xAF表示采用44100HZ 16bit stereo的AAC，指每秒钟有 44100 次采样, 采样数据用16位(2字节)记录，双声道(立体声)。

音频文件的格式决定了其声音的品质，日常生活中电话、收音机等均为模拟音频信号，即不存在采样频率和采样位数的概念，可以这样比较一下：  
          44KHz，16bit的声音称作：CD音质；  
          22KHz、16Bit的声音效果近似于立体声（FM Stereo）广播，称作：广播音质；  
          11kHz、8Bit的声音，称作：电话音质。



### 10. 添加AAC Body解析

AAC主体部分的设置与AAC Header部分的设置大体相同，只有最后一位的AACPacketType为1，表示AAC raw。



### RTMP协议介绍

Real Time Messaging Protocol（实时消息传送协议协议)是Adobe Systems公司为Flash播放器和服务器之间音频、视频和数据传输开发的私有协议。RTMP协议是TCP/IP五层体系结构中应用层的协议。

RTMP协议中基本的数据单元称为消息（Message）。当RTMP协议在互联网中传输数据的时候，消息会被拆分成更小的单元，称为消息块（Chunk）。

1. 消息是RTMP协议中基本的数据单元。不同种类的消息包含不同的Message Type ID，代表不同的功能。



图6 消息的报文结构

**Message Type**：1字节

0x04表示Ping包，0x08为audio，0x09为video。

**Payload Length**：3字节

这三个字节表述了tag中数据段的大小。

**Time Stamp**：4字节

记录了每一个tag相对于第一个tag（File Header）的相对时间。以毫秒为单位。而File Header的time stamp永远为0。

**Stream ID**：3字节

标识消息所属媒体流，永远为0。

**Message Body**：大小是Datasize

存储音视频信息。

1. 消息块

RTMP协议中规定，消息在网络上传输时被拆分成消息块（Chunk）。消息块首部（Chunk Header）有三部分组成：用于标识本块的Chunk Basic Header，用于标识本块负载所属消息的Chunk Message Header，以及当时间戳溢出时才出现的Extended Timestamp。



图7 消息块的报文结构

**Chunk Basic Header**：1-3字节

由Header Type + Channel ID组成，其中Channel ID的大小决定了整个Chunk Basic Header的大小。而HeaderType (fmt):决定了Chunk Message Header的编码方式和大小，在第一个字节的高两位。

|  |  |
| --- | --- |
| Bits | Chunk Message Header Length |
| 00 | 12 bytes |
| 01 | 8 bytes |
| 01 | 4 bytes |
| 11 | 1 byte |

|  |  |
| --- | --- |
| Channel ID | 用途 |
| 02 | Ping 和ByteRoad通道 |
| 03 | nvoke通道 我们的connect() publish()和自字写的NetConnection.Call() 数据都是在这个通道的 |
| 04 | Audio和Vidio通道 |
| 05 06 07 | 服务器保留,经观察FMS2用这些Channel也用来发送音频或视频数据 |

**Chunk Message Header**：该字段包含了将要发送的消息的信息（或者是一部分，一个消息拆成多个chunk的情况下是一部分）该字段的长度由chunk basic header中的type决定。

**Extend Timestamp**：该字段发送的时候必须是正常的时间戳设置成0xffffff时，当正常时间戳不为0xffffff时，该字段不发送。

**Chunk Data**：实际数据（Payload），可以是信令，也可以是媒体数据。

1. 消息分块

在消息被分割成几个消息块的过程中，消息负载部分（Message Body）被分割成大小固定的数据块（默认是128字节，最后一个数据块可以小于该固定长度），并在其首部加上消息块首部（Chunk Header），就组成了相应的消息块。消息分块过程如下图所示，一个大小为307字节的消息被分割成128字节的消息块（除了最后一个），如图8所示。

RTMP传输媒体数据的过程中，发送端首先把媒体数据封装成消息，然后把消息分割成消息块，最后将分割后的消息块通过TCP协议发送出去。接收端在通过TCP协议收到数据后，首先把消息块重新组合成消息，然后通过对消息进行解封装处理就可以恢复出媒体数据。

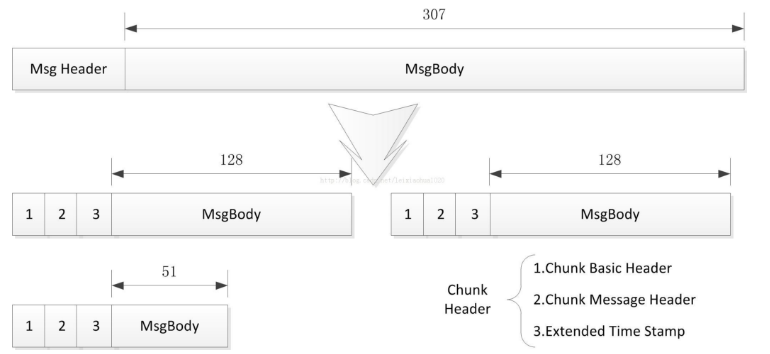
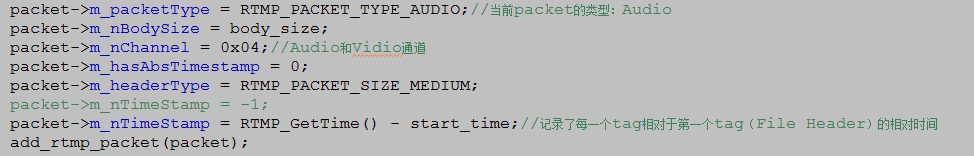


图8 消息分块的过程

### 利用RTMP传输音视频数据解析

以传输AAC body为例，分别设置packetType、BodySize、Channel、headerType、TimeStamp等信息。