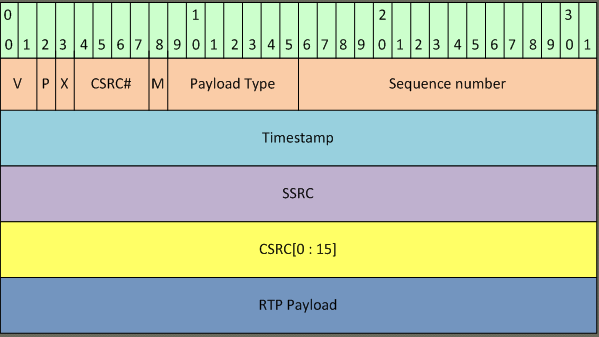
## RTP包格式



RTP 报头中各字段的含义如下: 版本(V):2个比特,表示RTP的版本号。

填充(P):1 个比特,置“1”表示用户数据最后加有填充位,用户数据中 最后一个字节是填充位计数,它表示一共加了多少个填充位。在两种情况下可能 需要填充,一是某些加密算法要求数据块大小固定;二是在一个低层协议数据包 中装载多个 RTP 分组。

扩展(X):1个比特,置“1”表示RTP报头后紧随一个扩展报头。

CSRC计数(CC):4个比特,表示在定长的RTP报头后的CSRC标识符的 数量。

标记(M):1 个比特,其具体解释由应用文档来定义。例如,对于视频流, 它表示一帧的结束,而对于音频,则表示一次谈话的开始。

载荷类别(PT):7 个比特,它指示在用户数据字段中承载数据的载荷类别。

序号(SN):2 个字节,每发送一个 RTP 数据包该序号增加 1。该序号在接 收方可用来发现丢失的数据包和对到来的数据包进行排序。

　时间戳(TS):4个字节,它用来表示RTP包中用户数据段的第一个字节的 采样时刻。时间戳的时间表示应为线性单调递增的,以便完成同步实现和抖动的 计算。

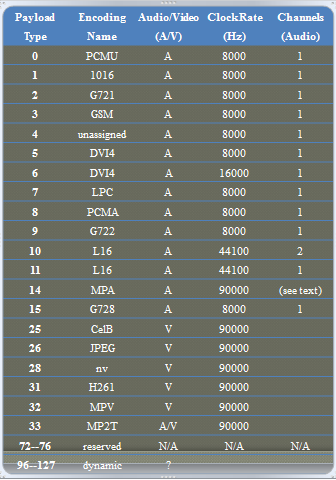
　同步源标识符(SSRC):4 个字节,用来标识一个同步源。此标识符是随机 选择的,但要保证同一 RTP 会话中的任意两个 SSRC 各不相同,RTP 必须检测 并解决冲突。

提供源标识符(CSRC):它可有 0~15 项标识符,每一项长度为 32 比特,其 项数由 CC 字段来确定。如果提供源多于 15 个,则只有 15 个被标识。

　为了能满足各种应用的需要,RTP 报头可进一步扩充,其时 X 比特将置“1”, 扩充的 RTP 报头部分则紧随在 SCRC 清单之内。

　　如果扩展位被置为 1,意味着 RTP 固定头后紧跟着一个头扩展,其格式如图 3-2 所示。前 16 位的内容由轮廓文件决定,主要用来标识不同的头扩展类型。这 种扩展方式主要用来传递独立于具体格式的载荷(payload-format-independent) 的应用信息。

其中的RTP Payload ID，举例来说Payload Type = 0对应的Audio编码格式为PCMU，采样率为8000Hz



## h.264基本知识

先介绍一些基本术语

**大B和小b**：B for Byte，b for bit，1字节＝8位

码流/码率：单位是bps(bit per second),一秒钟有多少数据

2/10/16进制：二进制 0000 1111 =>0x0F=>十进制的15

帧率：单位是fps(frame per second),一秒钟有多少幅画面

I、P、B帧：关键帧、参考帧、双向参考帧。其中I 帧是完整的帧也就适合关键帧，基本的压缩率很低；P帧是向前预测帧,B帧是双向预测帧。压缩率:I<P<B.B帧压缩率最大,但是在网络流里是不能用B帧的，原因很简单，因为B帧需要参考前后帧才能生成一张完整的图,在互联网传输的时候，是很难保证前后帧都能够接收到。下面的图片是例子:

第一帧



第二帧



I帧



P帧



h264又称为MPEG4-10或者AVC。压缩率和品质优于MPE-2/4。内建VCL（视频编码层）有更好的编码压缩率，NAL（网络提取层）有更好的纠错率，适合多媒体流和移动电视。有三种编码规模：

Baseline profile 低码率应用，使用在视频通信，个人随身播放

Main profile 交互式电影

Extension profile 高抗错编码工具使用IPTV

用NAL作为基本封包单位，也作为VCL运算单位。当网络拥堵，收到out of order包的时候，UDP设置Reference Flag =1，解压可以做纠错

NAL单元的形式：12种2大类

第一类 VCL NAL-Unit纯粹是视频压缩的内容

第二类 non VCL NAL-Unit 描述压缩所需要的参数：长宽比，显示时间，版权等

视频编码层 VCL，压缩的目的是去除冗余。h264的编码机制，以图块Block-base为基本单元

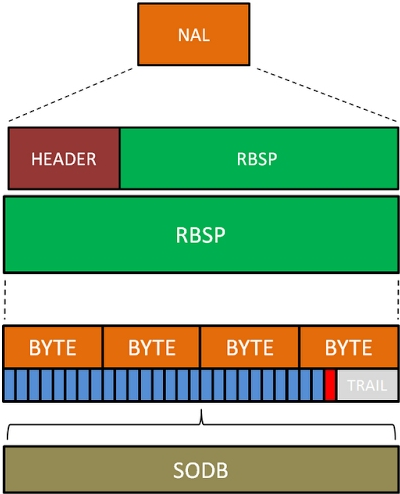
一 Macro-Block 一幅图像被分割成矩形小块

二 预测技术intra-prediction和inter-prediction得到图像之间的Residual

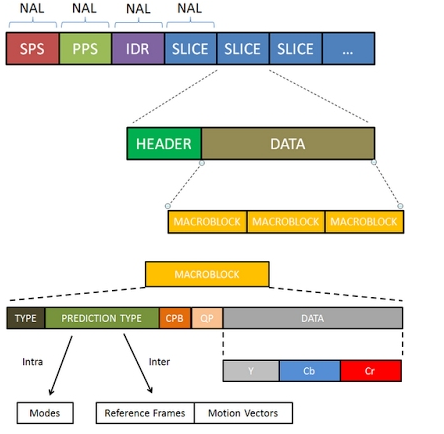
三 将Residual用空间transform和Quantize方法压缩

经过上面三步使用压缩编码得到bit-stream，加上header得到NAL-Unit

形象的看，可以用下面的图表示NAL内部的层级关系：

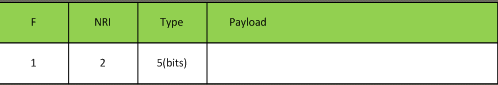


H264流的直观表示如图：



这里I和IDR帧都是使用帧内预测的。它们都是同一个东西而已,在编码和解码中为了方便，要首个I帧和其他I帧区别开，所以才把第一个首个I帧叫IDR，这样就方便控制编码和解码流程。

NAL



比如第一个byte值为60，对应二进制为00111100 也就是 F=0，NRI=01，Type=11100//28

具体见下面的程序

byte0 = (byte)(0x7f & (0xff & abyte0[1]));//获取payload 类型，在第二字节

short word0 = (short)(abyte0[2] << 8 | 0xff & abyte0[3]);//获取sequence number

i = (0xff & abyte0[4]) << 24 | (0xff & abyte0[5]) << 16 | (0xff & abyte0[6]) << 8 | 0xff & abyte0[7]; //获取time stamp

long l = (0xff & abyte0[8]) << 24 | (0xff & abyte0[9]) << 16 | (0xff & abyte0[10]) << 8 | 0xff & abyte0[11]; //获取SSRC

rtpPkt = new RTPPacket();

rtpPkt.setSeqNumber(word0);

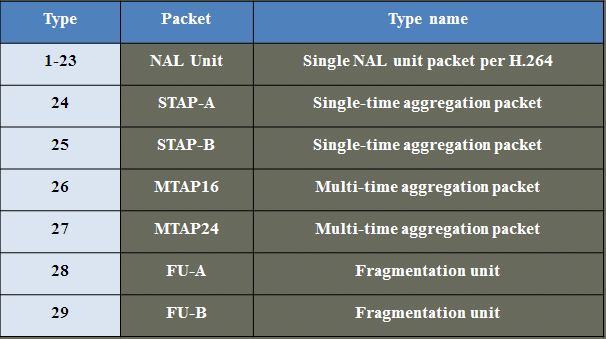
j = 0xffff & word0;

rtpPkt.setTimeStamp(i);

rtpPkt.setSsrc(l);

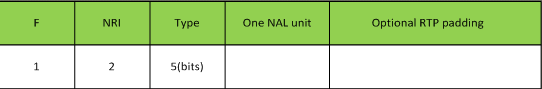
rtpPkt.setPT(byte0);

NAL Type对照表

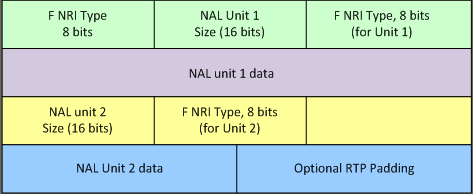


NAL Unit的三种格式

Single



STAP-A



FU-A，Fragmentation Units (FUs).

而当 NALU 的长度超过 MTU 时, 就必须对 NALU 单元进行分片封包. 也称为 Fragmentation Units (FUs).



The FU header has the following format:

+---------------+

|0|1|2|3|4|5|6|7|

+-+-+-+-+-+-+-+-+

|S|E|R| Type |

+---------------+

具体说明：

S：开始位(1bit)，当设置为1，开始位指示分片NAL单元的开始。第一个分片包设为1，其他的分片设置为0。

E：结束位(1bit)，当设置为1，结束位指示分片NAL单元的结束，即，FU荷载是最后分片时设置为1，其他时候设置为0。

R：保留位(1bit)，必须设置为0。

Type：5bit

RTP解包时应该这样分析：

byte startBit=(byte)(recbuf[13]&0x80);

byte endBit=(byte)(recbuf[13]&0x40);

1 如果，startBit==-128，这包是分片的首包。

NalBuf[4]=(byte) ((recbuf[12]&0xE0)+(recbuf[13]&0x1F)); 这句用于重建组合NAL单元类型

2如果(startBit==0)&&(endBit==0)，这包是分片的中间部分。

3如果 endBit==64 ，这包是分片尾部。

关于Audio代码的分析

payload = new byte[7 + (-16 + datagrampacket.getLength())];

//新生成的audio信息的长度总长度是是rtp静负荷的长度减去16然后加上7，意味自己要生成7个字节的audio头部信息

//获取的rtp包信息从第16个字节开始

for (int i1 = 0; i1 < -7 + payload.length; i1++) {

payload[i1 + 7] = abyte0[i1 + 16];

}

bIsFirstAudioStream = false;

desAudioPos = 0;

int j1 = 7 + (-16 + datagrampacket.getLength()); //audio 总长度

byte[] abyte3 = new byte[7];//audio header长度为7

abyte3[0] = -1;

abyte3[1] = -15;

abyte3[6] = -4;

int k1 = j1 / 2047;

byte byte1 = 4;

switch(48000) //采样率设定为48000

byte1 = 3; //byte1与采样率有关

abyte3[0] = -1;

abyte3[1] = -15;

abyte3[2] = 64;

abyte3[2] = (byte)(abyte3[2] | byte1 << 2);

abyte3[2] = (byte)(0 | abyte3[2]); ////与采样率有关信息写在audio header的3字节

abyte3[3] = (byte)128;

abyte3[3] = (byte)(abyte3[3] | (j1 & 0x1800) >> 11); //audio包总长度信息写入4字节

abyte3[4] = (byte)((j1 & 0x1ff8) >> 3); ///audio包总长度信息写入

abyte3[5] = (byte)((j1 & 7) << 5); ///audio包总长度信息写入

abyte3[5] = (byte)(0x1f | abyte3[5]); ///audio包总长度信息写入

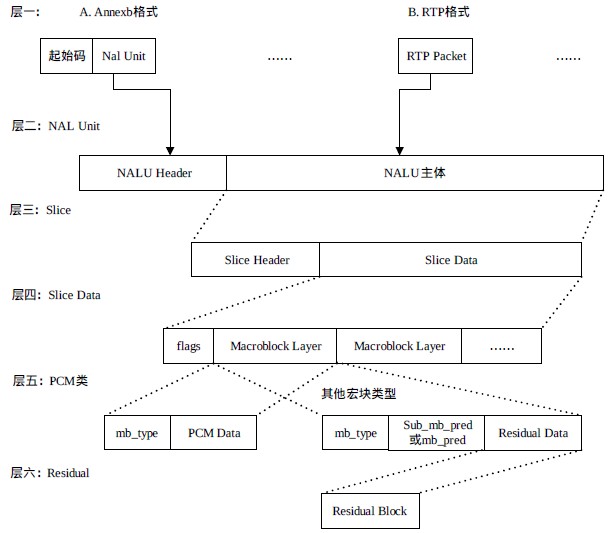
abyte3[6] = -4;

abyte3[6] = (byte)(abyte3[6] | k1 & 3); // k1 = j1 / 2047也是总长度有关信息写入

System.arraycopy(abyte3, 0, payload, desAudioPos, abyte3.length);

// 构造的audio header信息写入，该函数第一个参数为源地址，二个参数为源起始位置，第三个参数为目标地址，第四个参数为目标位置，第五个参数为复制长度

关于video代码的分析



注意到NAL的封装有两种格式：

Annexb格式：NALU数据+起始码，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 字节表示,0x000001。

RTP格式：NALU数据+RTP协议的RTP头数据。

NAL Header：forbidden\_bit,nal\_reference\_bit(优先级),nal\_unit\_type(类型)。

所以程序的输入应该是

脱壳操作：为了使 NALU 主体不包括起始码,在编码时每遇到两个字节(连续)的0，就插入一字节 0x03，以和起始码相区别。解码时,则将相应的 0x03 删除掉。

首先，在接收端根据rtp包的seqnumber来判断是否丢包，如果丢包就标记一下。

0x17 intra-frame; 0x27 non-intraframe

byte[] abyte1 = new byte[4];

abyte1[3] = 1;

//这里构造了4字节的0x00000001，作为NAL头

payload = new byte[-12 + datagrampacket.getLength()]; //NAL负载需要从RTP负载中去掉12个字节的

if (rtpPkt.getSeqNumber() == seq) //包按照顺序到达

if (payload[0] == 60)

if (payload[0] == 60)

//这里60二进制为00111100 也就是 F=0，NRI=01，Type=11100//28，也就是FU-A类型unit

{

if ((0xf0 & payload[1]) == 128)

{

// 11110000 & FU-A信息 == 10000000，这包是分片的首包

payload[1] = (byte)(32 + (0xf & payload[1]));

//重建组合NAL单元类型

System.arraycopy(abyte1, 0, totalByte, desPos, abyte1.length);

desPos = desPos + abyte1.length;

System.arraycopy(payload, 1, totalByte, desPos, -1 + payload.length);

//上面先复制4字节的0x00000001，然后复制RTP包payload

desPos = -1 + (desPos + payload.length);

Log.i("RTPLog", "VIDEOTYPE, if condition 0x3C8X");

} else

{

//这里2字节的NAL头不需要复制

System.arraycopy(payload, 2, totalByte, desPos, -2 + payload.length);

desPos = -2 + (desPos + payload.length);

Log.i("RTPLog", "VIDEOTYPE, else condition 0x3CXX");

}

} else

//如果不是，则先复制4字节的0x00000001，然后复制RTP包payload

{

System.arraycopy(abyte1, 0, totalByte, desPos, abyte1.length);

desPos = desPos + abyte1.length;

System.arraycopy(payload, 0, totalByte, desPos, payload.length);

desPos = desPos + payload.length;

}

if (bWaitingForIFrame && abyte2[4] == 39)

{

bWaitingForIFrame = false;

}

这里39的十六进制为0x27非关键帧iframe

关于位操作

Read: 00000001

Write: 00000010

则表达式 (READ | WRITE) 的值为00000011。

左移操作可以把4个byte组合成一个int

比如

A = 01000000

B = 00000101

C = 00101011

D = 11100011

int val = (A << 24) | (B << 16) | (C << 8) | D; 则

val = 01000000 00000101 00101011 11100011

颜色通常用这样的方法来处理

A = 255 = 11111111

R = 21 = 00010101

G = 255 = 11111111

B = 0 = 00000000

Color = 11111111 00010101 11111111 00000000

如果我们要从Color中读取各颜色值，只需要右移操作直到该颜色值到达最右边，然后对剩余的高位bits取掩码，

Int Alpha = Color >> 24

Int Red = Color >> 16 & 0xFF

Int Green = Color >> 8 & 0xFF

Int Blue = Color & 0xFF

0xFF 这里为 11111111. 例如对Red来说,详细的操作如下：

Color = 11111111 00010101 11111111 00000000

Color >> 16 = (filled in 00000000 00000000)11111111 00010101 (removed 11111111 00000000)

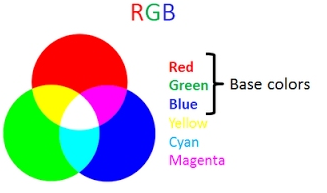
00000000 00000000 11111111 00010101 &

00000000 00000000 00000000 11111111 =

00000000 00000000 00000000 00010101

颜色空间

1 RGB模型，需要三个数字来表示



2 YCbCr模型，是更有效的模型

    Y = krR + kgG + kbB, 亮度  Y – luminance (luma),有了Y可以计算色差chroma

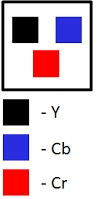
Cr = R – Y,  
Cg = G – Y,  
Cb = B – Y

    例子

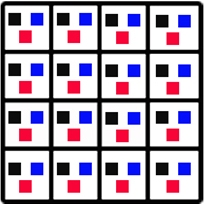
Y = 0.299R + 0.587G + 0.114B  
  
Cb = 0.564(B - Y),   
Cr = 0.713(R - Y)  
  
R = Y + 1.402Cr,   
B = Y + 1.772Cb

G = Y - 0.344Cb - 0.714Cr

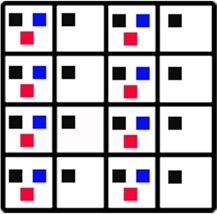
图像像素的直观表示为



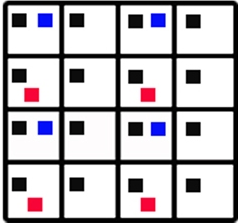
图像是像素的几何，表示为



为了压缩空间，我们可以采用一个技巧：就是牺牲某些chroma成分，图片表示为，注意其中绿色和蓝色部分的消失



进一步压缩，可以表示成



发送RTMP视频流

先介绍固定的文件格式

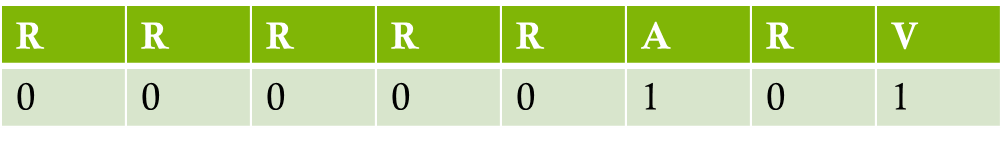
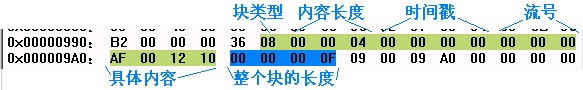
FLV Adobe公司设计开发的流媒体封装格式，我们可以将其数据看为二进制字节流。总体上看，FLV包括文件头（File Header）和文件体（File Body）两部分，其中文件体由一系列的Tag及Tag Size对组成。因此一个FLV文件看上去是下面的结构：



每个FLV文件开头都是这13个字节，属性05表示文件中既有视频，也有音频。

块头是固定的11字节，因此内容长度＋11＝块长度

类型只有08音频 09视频 12文本数据（metadata，cuepoint）



其中视频块为



帧类型：

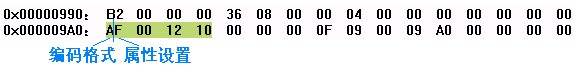
0x1 P帧 0x2 I帧 0x3 B帧

编码格式：

0x1 JPG 0x2 H263 0x4 On2 VP6

0x7 AVC／H264

音频块



编码格式：

0x2、0xE MP3 0x4~6 Nellymoser 0x7~8 G711 0xA AAC

属性设置：



采样率： 0x0 5.5kHz 0x1 11kHz 0x2 22kHz 0x3 44kHz

位长： 0x0 8bit 0x1 16bit

声道：0x0 mono 0x1 stereo

视频的H264的nalu



由于H264编解码时需要设置很多基本参数，因此flv格式封装的h264特别空出第一个字节（图中红字）来描述数据内容是什么。

其中00 后面的数据是NALU单元

01 后面的是普通视频数据

NALU里装着H264必须的SPS和PPS（序列、图像参数集）,可参照ISO 14496-15。

标签的时间序列只依赖 flv 的时间戳，任何基于负载数据的时间机制被忽略。在音频类型的标签中，数据的编码方式有两种，分别是 MP3 和 ADPCM。视频数据的编码方式有五种，分别是 Sorenson H.263，Screen video， On2 VP6，On2 VP6 with alpha channel，Screen video version 2。

并发送"describe"命令,开始猎取sdp.回调函数就是continueAfterDESCRIBE.在这里创建MediaSession,以后再发送"setup"命令,等,这些都能够在live555的例子和网上的说明中看到.

在unieye中的具体实现

基本文件在rtspclient

初始化在CameraServiceImpl中

public void startStreaming()

throws ResponseException, ConnectionException, InvalidNetworkException

{

int i = DataUtil.randomPort();

client = new RTSPClient(new InetSocketAddress(ApiConstant.SP\_IP, 554), null, "rtsp://192.168.1.1:554/av1", i);

client.init(application);

}

public void stopStreaming()

throws ResponseException, ConnectionException, InvalidNetworkException

{

if (client != null)

{

client.setReceiveRTP(false);

client.doTeardown();

}

}

}

使用rtp数据的类，例如PreviewActivity要实现

implements Runnable, com.Unieye.smartphone.rtsp.RTSPClient.OnRTPReceiveDataListener

public void onRTPReceiveData(DatagramPacket datagrampacket)

{

if (isShowingActivity)

{

if (datagrampacket.getLength() > 10)

{

receiveRTPData = true;

} else

{

receiveRTPData = false;

}

mCamerView.post(this);

itemCamerView.setDatagramPacket(datagrampacket);

}

}

public void setDatagramPacket(DatagramPacket datagrampacket)

{

if (datagrampacket != null)

{

getRTPPacket(datagrampacket, datagrampacket.getData());

}

}

在里面的getRTPPacket实现具体的机制

CameraServiceImpl的startstream是在PreviewActivity.java里面开始的

private void startStream()

{

mStartStream = new SmartPhoneAsyncTask(this, false) {

final PreviewActivity this$0;

protected BaseResponse doInBackground(Void void1)

throws ConnectionException, ResponseException, InvalidNetworkException

{

try

{

mCameraService.startStreaming();

mCameraService.setOnRTPReceiverDataListener(PreviewActivity.this);

}

catch (ResponseException responseexception)

{

responseexception.printStackTrace();

}

catch (ConnectionException connectionexception)

{

connectionexception.printStackTrace();

}

catch (InvalidNetworkException invalidnetworkexception)

{

invalidnetworkexception.printStackTrace();

}

return null;

}

mStartStream.execute(new Void[0]);

}

0x17 00010111

0x27 00100111

这里60二进制为00111100 也就是 F=0，NRI=01，Type=11100//28

播放方法

我们将解码后的视频数据用bitmap显示，draw到surfaceView的方法显示到手机屏上，由于有些手机不支持rgb24但几乎所有手机都支持rgb565，所以解码后返回的是rgb565数据。