## H264码流

SPS：Sequence Parameter Set 序列参数集 H.264码流的第一个NALU

保存了一组编码视频序列（Coded video sequence）的全局参数，即原始视频的一帧一帧的像素数据经过编码之后的结构组成的序列

每一帧的编码后数据所依赖的参数保存于图像参数集中

一般情况SPS和PPS的NALU通常位于整个码流的起始位置。

特殊情况下，码流中间也可能出现这两种结构，主要原因如下：

1、解码器需要在码流中间开始解码

2、编码器在编码过程中改变了码流参数（如分辨率）

语法元素含义：

（1）profile\_idc：

标识当前H.264码流的profile（H.264的SPS中，第一个字节标识profile\_idc）

H.264中三种常用的profile：baseline profile = 66、main profile = 77、extended profile = 88

（2）level\_idc：

标识当前码流的level。编码的level定义了某种条件下的最大视频分辨率、最大视频帧率等参数

（3）seq\_parameter\_set\_id：

标识当前的序列参数及的id，通过该值，图像参数集PPS中可以引用其代表的SPS中的参数

（4）log2\_max\_frame\_num\_minus4

用于计算MaxFrameNum的值，公式为MaxFrameNum = 2^(log2\_max\_frame\_num\_minus4 + 4)

MaxFrameNum是frame\_num的上限值，frame\_num是图像序号的一种表示方法，在帧间编码中常用作一种参考帧标记的手段

（5）pic\_order\_cnt\_type

表示解码picture order count（POC）的方法。POC是另一种计量图像序号的方式，与frame\_num计算方法不同。该语法元素取值为0、1、2

（6）log2\_max\_pic\_order\_cnt\_lsb\_minus4

用于计算MaxPicOrderCntLsb的值，该值表示POC的上限。MaxPicOrderCntLsb = 2^(log2\_max\_pic\_order\_cnt\_lsb\_minus4 + 4)

（7）max\_num\_ref\_frames

表示参考帧的最大数目

（8）gaps\_in\_frame\_num\_value\_allowed\_flag

标志位，说明frame\_num中是否允许不连续的值

（9）pic\_width\_in\_mbs\_minus1

用于计算图像的宽度。单位为宏块个数，因此图像的实际宽度为frame\_width = 16 \* (pic\_width\_in\_mbs\_minux1 + 1)

（10）pic\_height\_in\_map\_units\_minus1

使用PicHeightInMapUnits来度量视频中一帧图像的高度，该值并非图像明确的以像素或宏块为单位的高度，而需要考虑该宏块是帧编码或场编码

PicHeightInMapUnits = pic\_height\_in\_map\_units\_minus1 + 1

（11）frame\_mbs\_only\_flag

标志位，说明宏块的编码方式

0：宏块可能为帧编码或场编码。PicHeightInMapUnits表示一场数据按宏块计算的高度

1：所有宏块采用帧编码。PicHeightInMapUnits表示一帧数据按宏块计算的高度。FrameheightInMbs = ( 2 − frame\_mbs\_only\_flag ) \* PicHeightInMapUnits

（12）mb\_adaptive\_frame\_field\_flag

标志位，说明是否采用了宏块级的帧、场自适应编码

0：不存在帧编码和场编码之间的切换

1：宏块可能在帧编码和场编码模式之间进行选择

（13）direct\_8x8\_inference\_flag

标志位，用于B\_Skip、B\_Direct模式运动矢量的推导计算

（14）frame\_cropping\_flag

标志位，说明是否需要对输出的图像帧进行裁剪

（15）vui\_parameters\_present\_flag

标志位，说明SPS中是否存在VUI信息

PPS：Picture Parameter Set 图像参数集 H.264码流的第二个NALU nal\_unit\_type为8

语法元素含义：

（1）pic\_parameter\_set\_id：

表示当前PPS的id，某个PPS在码流中会被相应的slice引用，引用PPS的方式就是在Slice header中保存PPS的id值，取值的范围[0, 255]

（2）seq\_parameter\_set\_id：

表示当前PPS所引用的激活的SPS的id，通过这种方式，PPS中也可以取到对应SPS中的参数，取值范围[0, 31]

（3）entropy\_coding\_mode\_flag

熵编码模式标志，该标志位标识码流中熵编码/解码选择的算法。对于部分语法元素，在不同的编码配置下，选择的熵编码方式不同

如：在一个宏块语法元素中，宏块类型mb\_type的语法元素描述符为“ue(v) | ae(v)”，在baseline profile等设置下采用指数哥伦布编码，在main profile等设置下采用CABAC编码

标志位entropy\_coding\_mode\_flag的作用就是控制这种算法选择

0：选择左边的算法，通常为指数哥伦布编码或者CAVLC

1：选择右边的算法，通常为CABAC

（4）bottom\_field\_pic\_order\_in\_frame\_present\_flag

标志位，用于标识另外条带头中两个语法元素delta\_pic\_ordercnt\_bottom和delta\_pic\_order\_cn是否存在的标志

这两个语法元素表示了某一帧的底场的POC的计算方法

（5）num\_slice\_groups\_minus1

表示某一帧中slice group的个数。

0：一帧中所有的slice都属于一个slice group。slice group是一帧中宏块的组合方法

（6）num\_ref\_idx\_I0\_default\_active\_minux1、num\_ref\_idx\_I0\_default\_active\_minus1

表示当Slice Header中的num\_ref\_idx\_active\_override\_flag标志位为0时，P/SP/B slice的语法元素

num\_ref\_idx\_I0\_active\_minus1和num\_ref\_idx\_l1\_active\_minus1的默认值

（7）weighted\_pred\_flag

标志位，表示在P/SP slice中是否开启加权预测

（8）weighted\_bipred\_idc

表示在B slice中加权预测的方法，取值范围[0, 2]

0：默认加权预测

1：显示加权预测

2：隐式加权预测

（9）pic\_init\_qp\_minus26、pic\_init\_qs\_minus26

表示初始的量化参数。实际的量化参数由该参数、slice header中的slice\_qp\_delta/slice\_qs\_delta计算得到

（10）chroma\_qp\_index\_offset

用于计算色度分量的量化参数，取值范围为[-12,12]。

（11）deblocking\_filter\_control\_present\_flag

标识位，用于表示Slice header中是否存在用于去块滤波器控制的信息。

0：slice header中没有相应的信息。

1：slice header中包含去块滤波相应的信息

（12）constrained\_intra\_pred\_flag

0：表示I宏块可以使用来自Inter类型宏块的信息。

1：表示I宏块在进行帧内预测时只能使用来自I和SI类型宏块的信息

（13）redundant\_pic\_cnt\_present\_flag

标识位，用于表示Slice header中是否存在redundant\_pic\_cnt语法元素。

0：slice header中没有相应的信息。

1：slice header中包含redundant\_pic\_cnt

## 2 SPS PPS详解

**2.1 SPS语法元素及其含义**

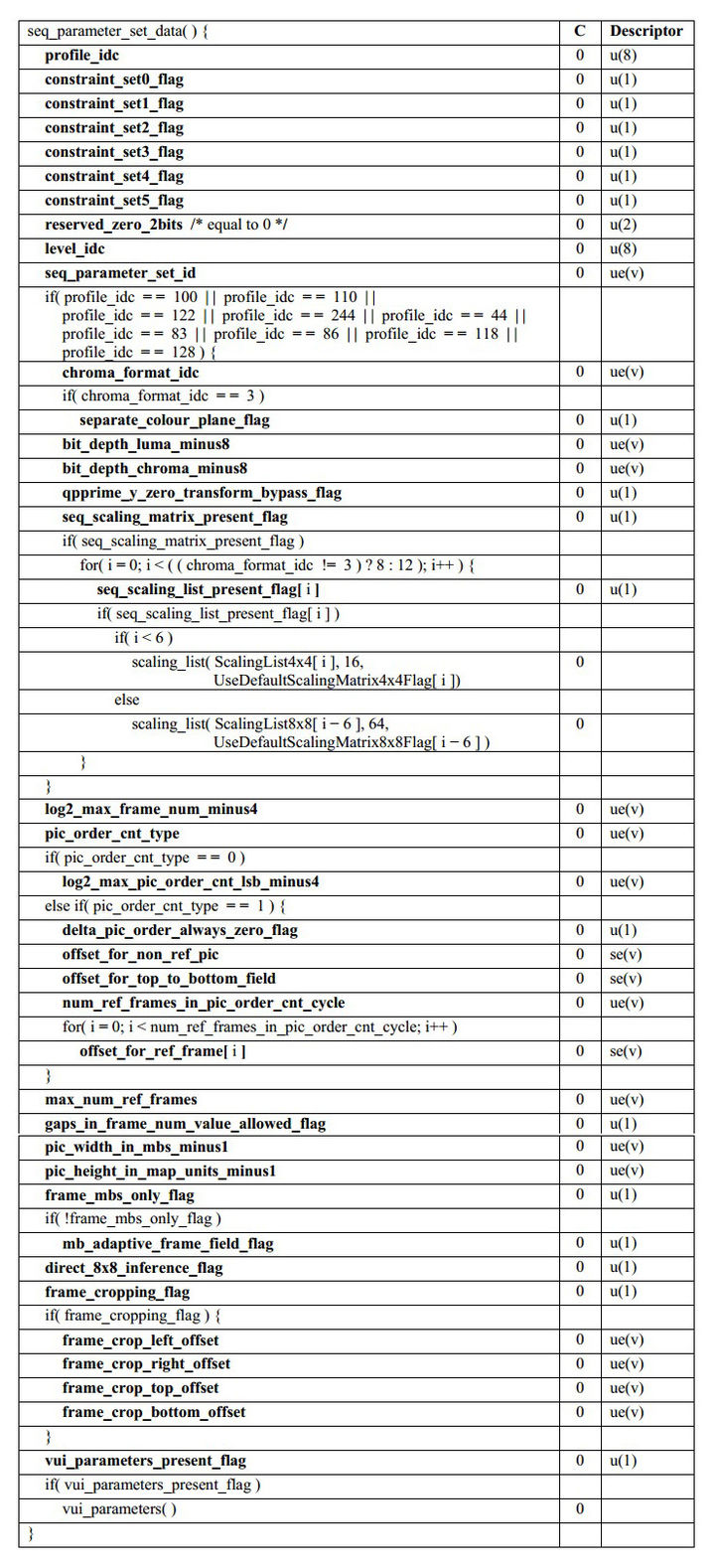
在H.264标准协议中规定了多种不同的NAL Unit类型，其中类型7表示该NAL Unit内保存的数据为Sequence Paramater Set。在H.264的各种语法元素中，SPS中的信息至关重要。如果其中的数据丢失或出现错误，那么解码过程很可能会失败。SPS及后续将要讲述的图像参数集PPS在某些平台的视频处理框架（比如iOS的VideoToolBox等）还通常作为解码器实例的初始化信息使用。

SPS即Sequence Paramater Set，又称作序列参数集。**SPS中保存了一组编码视频序列(Coded video sequence)的全局参数**。所谓的编码视频序列即原始视频的一帧一帧的像素数据经过编码之后的结构组成的序列。而每一帧的编码后数据所依赖的参数保存于图像参数集中。**一般情况SPS和PPS的NAL Unit通常位于整个码流的起始位置**。但在某些特殊情况下，在码流中间也可能出现这两种结构，主要原因可能为：

解码器需要在码流中间开始解码；

编码器在编码的过程中改变了码流的参数（如图像分辨率等）；

在做视频播放器时，为了让后续的解码过程可以使用SPS中包含的参数，必须对其中的数据进行解析。其中H.264标准协议中规定的SPS格式位于文档的7.3.2.1.1部分，如下图所示：



**IDR：Instantaneous Decoding Refresh 即时解码刷新 H.264码流的第三个NALU**

### 注意：RTP在网络传输中不需要前面的start code起始码，但是中间的防竞争码是一直存在的。

### 一般来说，在H．264中，RTP封装应该遵循几个设计原则：

1、较低的开销，因此MTU的尺寸应该限制在100—64K字节范围内。

2、易于区分分组的重要性，而不必对分组内的数据解码。

3、应能检测到数据的类型，而不需解码整个数据流，并能根据编码流之间的相关性丢弃无用数据，如网关应能检测A型分割的丢失，并能丢弃相应的B型和C型分割。

4、应支持将一个NALU拆分为若干个RTP包：不同大小的输入图片决定了NALU的长度可能会大于MTU，只有拆分后才会避免IP层在传输时出现分片。

5、支持将多个NALU汇集在一个RTP分组中，即在一个RTP包中传输超过一个NALU，当多个图片的编码输出小于MTU时就考虑此模式，以提高网络传输效率。

### 网络传输时为什么要将 H264 数据封装成 RTP 包？

更方便分包和进行丢包处理，因为 H264 中 NALU 的存在，封装成 RTP 包在 NALU 前加个包头即可（最理想情况下）

RTP打包模式

主要分为三种模式：单一NALU模式、分片模式、组合模式，实际中前两种用的比较多。

(1)单一NALU模式

一个RTP包仅由一个完整的NALU组成。这种情况下RTP NAL头类型字段和原始的H.264的NALU头类型字段是一样的。

适合条件是当NALU的长度小于RTP包长减去12时。

特别NALU type 值为 7 和 8 的NALU分别为序列参数集（sps）和图像参数集（pps）。

(2)组合封包模式

即可能是由多个 NAL 单元组成一个 RTP 包. 分别有4种组合方式: STAP-A, STAP-B, MTAP16, MTAP24.

那么这里的类型值分别是 24, 25, 26 以及 27.适合条件当 NALU 的长度特别小时, 可以把几个 NALU 单元封在一个 RTP 包中.

(3)分片封包模式Fragmentation Units (FUs)

用于把一个 NALU 单元封装成多个 RTP 包. 存在两种类型 FU-A 和 FU-B. 类型值分别是 28 和 29。

适合条件当 NALU 的长度超过 MTU 时, 就必须对 NALU 单元进行分片封包.

其中，VCL层是对核心算法引擎，宏块及片的语法级别的定义，他最终输出编码完的数据 SODB；

1.压缩：预测（帧内预测和帧间预测）-> DCT变化和量化 -> 比特流编码；

2.切分数据，主要为了第三步。这里一点，网上看到的“切片（slice）”、“宏块（macroblock）”是在VCL中的概念，

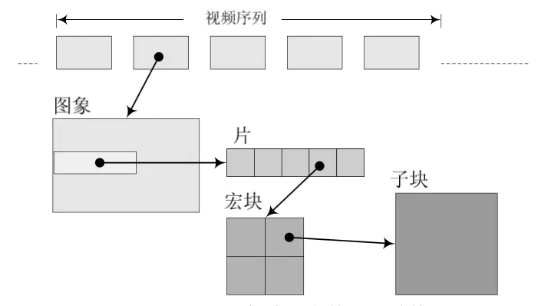
一方面提高编码效率和降低误码率、另一方面提高网络传输的灵活性。

3.输出编码完的数据 SODB（SODB：String Of Data Bits （原始数据比特流, 长度不一定是8的倍数，故需要补齐））

### H264 码流

**H.264 中，句法元素共被组织成 序列、图像、片、宏块、子宏块五个层次。**

H.264 的码流结构并没有大家想的那么复杂，编码后视频的每一组图像（GOP，图像组）都给予了传输中的序列（PPS）和本身这个帧的图像参数（SPS），所以，我们的整体结构，应该如此：



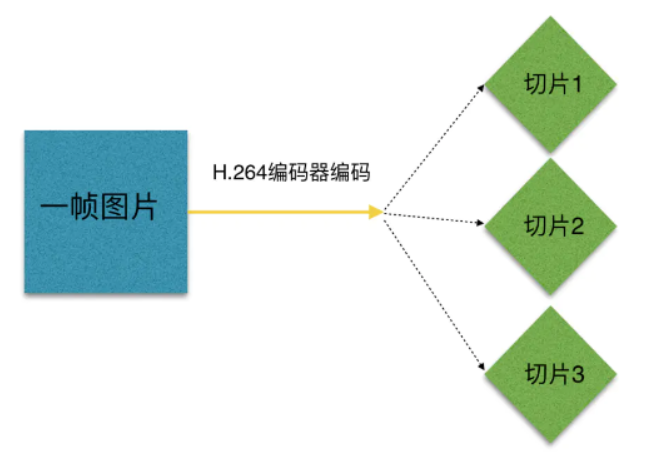
1. GOP （图像组）主要用作形容一个 i 帧 到下一个 i 帧之间的间隔了多少个帧，增大图片组能有效的减少编码后的视频体积，但是也会降低视频质量，至于怎么取舍，得看需求了。**GOP不包括 SPS、PPS 等类型的 NALU**
2. GOP 就是一段连续图像帧且画面之间变化不大。//当某个帧与前面的帧图像变化很大，无法参考前面帧生成时，就要开始一个新序列。

NAL层定义片级以上的语法级别（如序列参数集和图像参数集，针对网络传输），

同时支持以下功能：独立片解码，起始码唯一保证，SEI以及流格式编码数据传送，

NAL层将SODB打包成RBSP然后加上NAL头，组成一个NALU（NAL单元）；

一帧图片经过 H.264 编码器之后，就被编码为一个或多个片（slice）。<编码阶段,由VCL实现>



而装载着这些片（slice）的载体，就是 NALU

NALU 跟片的关系（slice）<由NAL实现>

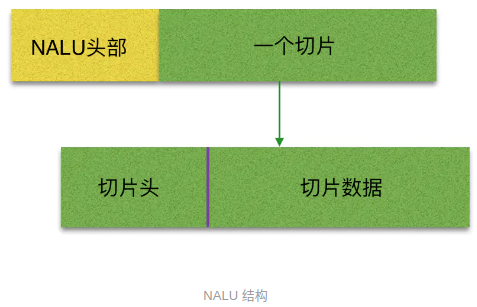
注意：实际中NAL还要对切片进行处理，然后形成NALU。

| NALU 头部 | 一个切片 |

||

V

| 切片头 | 切片数据 |



片（slice）的概念不同与帧（frame），帧（frame）是用作描述一张图片的，一帧（frame）对应一张图片，

而片（slice），是 H.264 中提出的新概念，是通过编码图片后切分通过高效的方式整合出来的概念，**一张图片至少有一个或多个片（slice）。**

片（slice）都是由 NALU 装载并进行网络传输的，但是这并不代表 NALU 内就一定是切片，

这是充分不必要条件，因为 NALU 还有可能装载着其他用作描述视频的信息，比如SPS、PPS。

片的主要作用是用作宏块（Macroblock）的载体。片之所以被创造出来，主要目的是为限制误码的扩散和传输。

如何限制误码的扩散和传输？

每个片（slice）都应该是互相独立被传输的，某片的预测（片（slice）内预测和片（slice）间预测）不能以其它片中的宏块（Macroblock）为参考图像。

| Slice |

||

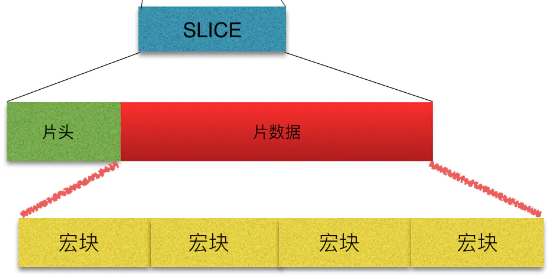
V

| 片头 | 片数据 |

||

V

| 宏块 | 宏块 | 宏块| 。。。|



每个分片也包含着头和数据两部分：

1、分片头中包含着分片类型、分片中的宏块类型、分片帧的数量、分片属于那个图像以及对应的帧的设置和参数等信息。

2、分片数据中则是宏块，这里就是我们要找的存储像素数据的地方。

宏块是视频信息的主要承载者，因为它包含着每一个像素的亮度和色度信息。

视频解码最主要的工作则是提供高效的方式从码流中获得宏块中的像素阵列。

组成部分：一个宏块由一个16×16亮度像素和附加的一个8×8 Cb和一个 8×8 Cr 彩色像素块组成。

每个图象中，若干宏块被排列成片的形式。

| 宏块 |

||

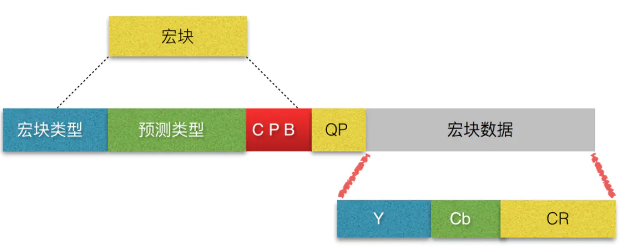
V

| 宏块类型 | 预测类型 | C P B | QP | 宏块数据 |

||

V

| Y | Cb | Cr |



宏块中包含了宏块类型、预测类型、​ ​Coded Block Pattern 编码的块模式​​​、​​Quantization Parameter 量化参数​​、像素的亮度和色度数据集等等信息。

切片（slice）类型跟宏块类型的关系

I片：只包 I宏块，I 宏块利用从当前片中已解码的像素作为参考进行帧内预测(不能取其它片中的已解码像素作为参考进行帧内预测)。

P片：可包 P 和 I 宏块，P 宏块利用前面已编码图象作为参考图象进行帧内预测，一个帧内编码的宏块可进一步作宏块的分割:

即 16×16、16×8、8×16 或 8×8 亮度像素块(以及附带的彩色像素);

如果选了 8×8 的子宏块，则可再分成各种子宏块的分割，其尺寸为 8×8、8×4、4×8 或 4×4 亮度像素块(以及附带的彩色像素)。

B片：可包 B 和 I 宏块，B 宏块则利用双向的参考图象(当前和 来的已编码图象帧)进行帧内预测。

SP片(切换P)：用于不同编码流之间的切换，包含 P 和/或 I 宏块

SI片：扩展档次中必须具有的切换，它包 了一种特殊类型的编码宏块，叫做 SI 宏块，SI 也是扩展档次中的必备功能。

H.264的两种打包/封装方法：字节流 AnnexB 格式 和 AVCC 格式 （只有这两种）

在H264用于网络发送时，要封装成RTP格式！！！

# H.264格式

##### 一.AnnexB流结构 --- 用于实时播放：

使用start code分隔NAL(start code为三字节或四字节，0x000001或0x00000001，一般是四字节)；

SPS和PPS按流的方式写在头部。

开始前缀（00000001或000001）＋ NALU数据 绝大部分编码器的默认输出格式

NALU 就是 h264的实际数据部分。

NALU = NALU Header + EBSP 组成; EBSP = 防止竞争码+RBSP; RBSP = SODB + RBSP尾部 。

EBSP为扩展字节序列载荷（Encapsulated Byte Sequence Payload） EBSP = RBSP插入防竞争字节（0x03）

RBSP为原始字节序列载荷（Raw Byte Sequence Payload）-------- RBSP = SODB + RBSP Trailing Bits（RBSP尾部补齐字节）;引入RBSP Trailing Bits做8位字节补齐。

SODB为原始数据比特流 （String Of Data Bits）　　　-------　　　就是最原始的编码/压缩得到的数据。

H264 码流结构

H.264 Annex B 格式

||

V

| 起始码 | NALU | | 起始码 | NALU | | 起始码 | NALU |

||

V

| NALU Header | EBSP |

|| 去除0x03

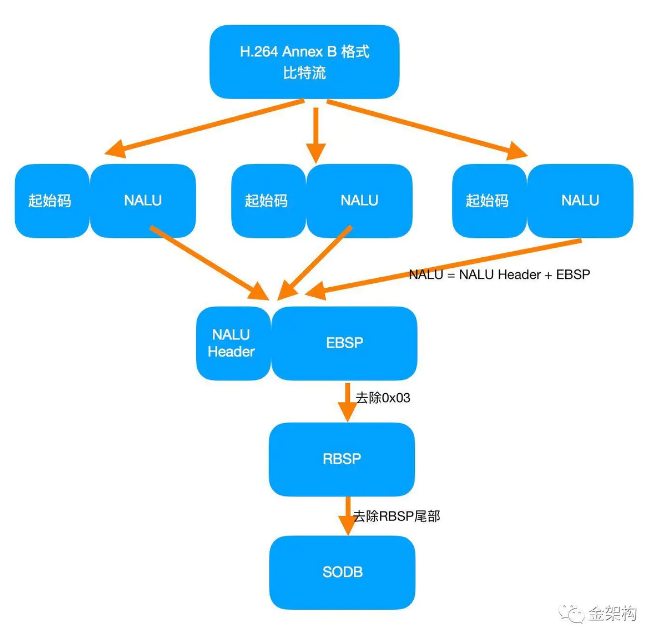
V

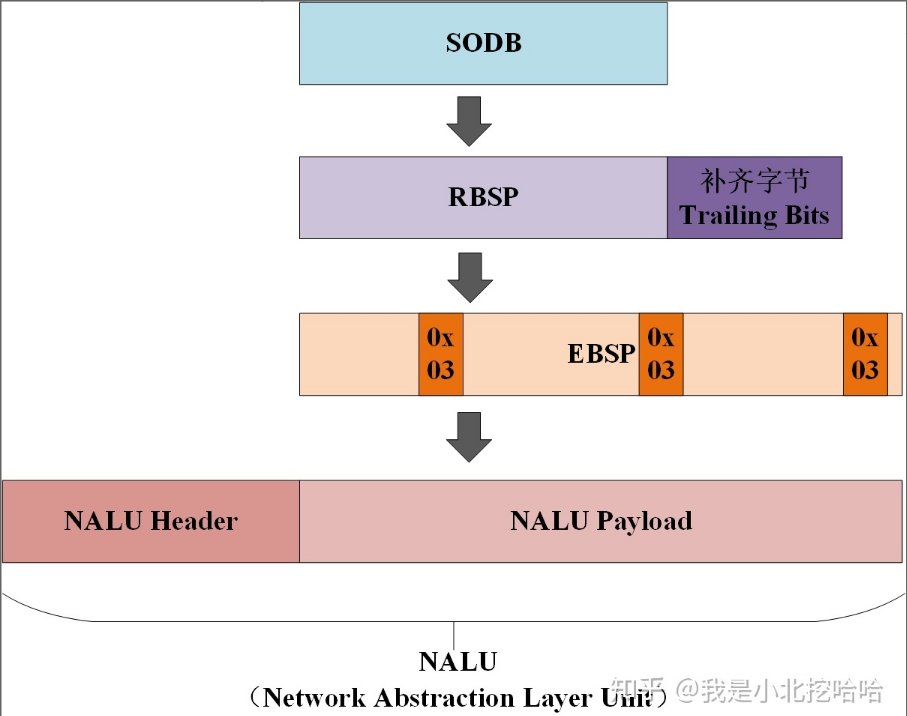
| RBSP |

|| 去除 RBSP 尾部

V

| SODB |





一共有两种起始码start\_code：

1、3字节0x000001 单帧多slice（即单帧多个NALU）之间间隔

2、4字节0x00000001 帧之间，或者SPS、PPS等之前

4字节类型的起始码在连续的数据传输中非常有用，因为用字节来对齐、分割流数据，

比如：用连续的31个bit 0 后接一个bit 1 来分割流数据，是很容易的。

如果接下来的bit是0(因为每个NALU都以bit0开始)，那么这就是一个NALU包数据的起始位置了。

4字节类型的开始码通常只用于标识流中的随机访问点，如SPS PPS AUD和IDR，然后其他地方都用3字节类型的开始码以减少数据量。

防止竞争字节（​​0x03​​）:

前面讲到用StartCode的字节串来分割NALU，于是问题来了，

如果RBSP中也包括了StartCode（0x000001或0x00000001）怎么办呢？

所以，就有了防止竞争字节（0x03）：

编码时，扫描RBSP，如果遇到连续两个0x00字节，就在后面添加防止竞争字节（0x03）；

解码时，同样扫描EBSP，进行逆向操作即可。

编码示例如下：

0x000000 ===> 0x00000300

0x000001 ===> 0x00000301

0x000002 ===> 0x00000302

0x000003 ===> 0x00000303

解码操作：

在解码的时候如果在内部遇到0x000003序列时，就可以将其抛弃即可以恢复原始数据。

EBSP 去除防止竞争码后就可以得到 RBSP。

AU分隔：

一个单独的NALU包、或者甚至一个VCL NALU包都不意味着是一个独立的帧，

一帧数据可以被分割成几个NALU，一个或多个NALU组成了一个Access Units(AU)，

AU包含了一个完整的帧。把帧分割成几个独立的NALU需要耗费许多CPU资源，所以分割帧数据并不经常使用。

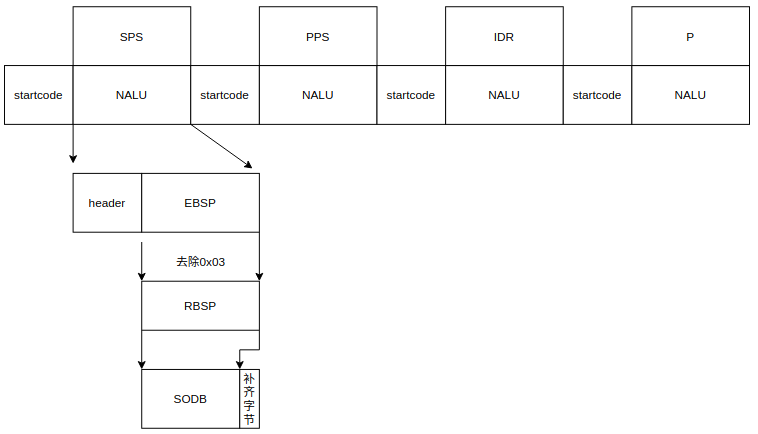
特殊的NALU类型：SPS和PPS

SPS和PPS存储了编解码需要一些图像参数。

AnnexB格式每个NALU都包含起始码，且通常会周期性的在关键帧之前重复SPS和PPS （在I帧之前）

例：

| startcode | NALU （SPS） | startcode | NALU （PPS）| startcode | IDR帧 | P 帧 | 。。。。。。



所以解码器可以从视频流随机点开始进行解码，实时的流格式

H.264 帧数据解析：

0000 0001 （start code) 67 (SPS) 64 000a ac72 8444 2684 0000 03 (防竞争字节) 00 0400 0003 (防竞争字节) 00ca 3c48

9611 8000 0000 01 （start code) 68 （PPS） e843 8f13 2130 0000 01 （start code） 65 （I帧） 8881 0005 4e7f 87df

61a5 8b95 eea4 e938 b76a 306a 71b9 5560 0b76 2eb5 0ee4 8059 27b8 67a9 6337 5e82 2055 fbe4 6ae9 3735 72e2

2291 9e4d ff60 86ce 7e42 b795 ce2a e126 be87 7384 26ba 1636 f4e6 9f17 dad8 6475 54b1 f345 0c0b 3c74 b39d

bceb 5373 87c3 0e62 4748 62ca 59eb 863f 3afa 86b5 bfa8 6d06 1650 82c4 ce62 9e4e e64c c730 3ede a10b d883

0bb6 b828 bca9 eb77 43fc 7a17 9485 21ca 376b 3095 b546 7730 60b7 12d6 8cc5 5485 29d8 69a9 6f12 4e71 dfe3

e2b1 6b6b bf9f fb2e 5730 a969 76c4 46a2 dffa 91d9 5074 551d 4904 5a1c d686 687c b661 486c 96e6 124c 27ad

bac7 5199 8ed0 f0ed 8ef6 6579 79a6 12a1 95db c8ae e3b6 35e6 8dbc 48a3 7faf 4a28 8a53 e27e 6808 9f67 7798

52db 5084 d65e 25e1 4a99 5834 c711 d643 ffc4 fd9a 4416 d1b2 fb02 dba1 8969 34c2 3255 98f9 9bb2 313f 4959

0c06 8cdb a5b2 9d7e 122f d087 9444 e40a 76ef 992d 9118 3950 3b29 3bf5 2c97 7348 9183 b0a6 f34b 702f 1c8f

3b78 23c6 aa86 4643 1dd7 2a23 5e2c d948 0af5 f52c d1fb 3ff0 4b78 37e9 45dd 72cf 8035 c395 07f3 d906 e54a

5876 036c 8120 6245 6544 73bc fec1 9f31 e5db 895c 6b79 d868 90d7 26a8 a188 8681 dc9a 4f40 a523 c7de be6f

76ab 7916 5121 6783 2ef3 d627 1a42 c294 d15d 6cdb 4a7a e2cb 0bb0 680b be19 5900 50fc c0bd 9df5 f5f8 a817

19d6 b3e9 74ba 50e5 2c45 7bf9 93ea 5af9 a930 b16f 5b36 241e 8d55 57f4 cc67 b265 6aa9 3626 d006 b8e2 e373

8bd1 c01c 5215 cab5 ac60 3e36 42f1 2cbd 9977 aba8 a9a4 8e9c 8b84 de73 f091 2997 aedb afd6 f85e 9b86 b3b3

03b3 ac75 6fa6 1169 2f3d 3ace fa53 8660 956c bbc5 4ef3

##### 二、AVCC --- 用于存储

另一个存储H.264流的方式是AVCC格式，在这种格式中， 每一个NALU包都加上了一个指定其长度(NALU包大小)的前缀(in big endian format大端格式)，

这种格式的包非常容易解析，但是这种格式去掉了Annex B格式中的字节对齐特性，

而且前缀可以是1、2或4字节，这让AVCC格式变得更复杂了，

指定前缀字节数(1、2或4字节)的值保存在一个头部对象中(流开始的部分)，这个头通常称为’extradata’或者’sequence header’。（重点）

1.AVCC格式了解：

使用NALU长度（固定字节，通常为4字节，取决于头部的 NALU LengthSizeMinusOne 字段）分隔 NAL；

在头部包含extradata(或sequence header)的结构体。（extradata 包含分隔的字节数、SPS 和 PPS）

解码器配置参数在一开始就配置好了（所以我们不能像视频网站中的实时播放一样可以在中间修改参数，比如：帧率，画面），

系统可以很容易的识别NALU的边界，不需要额外的起始码，减少了资源的浪费，同时可以在播放时调到视频的中间位置。

这种格式通常被用于可以被随机访问的多媒体数据。如存储在硬盘的文件：MP4、MKV 通常用 AVCC 格式来存储。

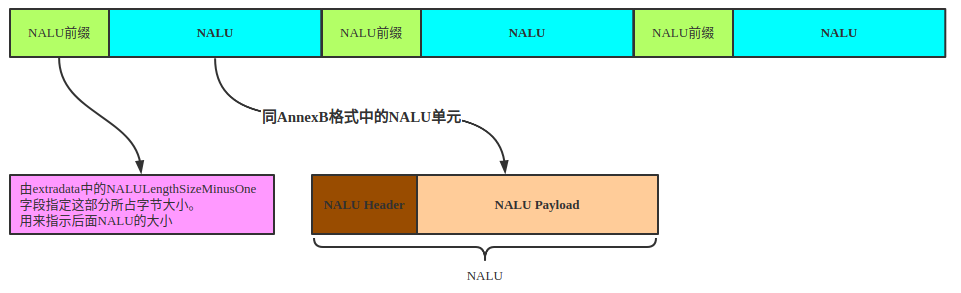
AVCC 格式不使用起始码(start code)作为NALU的分界，这种格式在每个 NALU 前都加上一个大端格式的前缀（1、2、4字节，代表NALU长度）

所以在解析 AVCC 格式的时候需要将指定的前缀字节数的值保存在一个头部对象中，

这个都通常称为 extradata 或者 sequence header。同时，SPS 和 PPS 数据也需要保存在 extradata 或者叫’sequence header’中。

AVCC 码流结构

| NALU 前缀 | NALU | NALU 前缀 | NALU |。。。。。。



AVCC中的NALU格式，与AnnexB格式一致。当我们将AVCC转AnnexB时，如果检测到NALU Type = 5关键帧，那么在关键帧前面加上SPS NALU和PPS NALU即可。

虽然AVCC格式不使用起始码，防竞争字节还是有的。所以我们在转换AVCC与AnnexB格式的时候，不用考虑防竞争字节。因为NALU内部是一致的。

##### 三、RTP格式 --- 用于网络发送

RTP封装= 12字节固定RTP包头 + 载荷（NALU）

补充：针对IP网络的RTP打包方式。为原始的NAL打包格式，就是开始的若干字节（1，2，4字节）是NAL的长度， 而不是start\_code,此时必须借助某个全局的数据来获得编码器的profile,level,PPS,SPS等信息才可以解码。

RTP 协议实际上是由实时传输协议RTP（Real-time Transport Protocol）和实时传输控制协议RTCP（Real-time Transport Control Protocol）两部分组成。

RTP 协议基于多播或单播网络为用户提供连续媒体数据的实时传输服务；

RTCP 协议是 RTP 协议的控制部分，用于实时监控数据传输质量，为系统提供拥塞控制和流控制。

NALU header结构图

NALU header由一个字节组成, 它的语法如下:

F: 1 个比特.forbidden\_zero\_bit. 在 H.264 规范中规定了这一位必须为 0.

NRI: 2 个比特.nal\_ref\_idc. 取 00 ~ 11, 似乎指示这个 NALU 的重要性, 如00的NALU解码器可以丢弃它而不影响图像的回放. 不过一般情况下不太关心这个属性.

Type: 5 个比特.nal\_unit\_type. 这个 NALU 单元的类型. 但是在h264中只有 1~23 是有效的值.而其他的24~29在RTP封包采用”组合封包模式“和”分片封包模式“时所用的type类型，而非“单一NAL单元模式”时。

简述如下:

0 没有定义

1-23 NAL单元 单个 NAL 单元包.

24 STAP-A 单一时间的组合包

25 STAP-B 单一时间的组合包

26 MTAP16 多个时间的组合包

27 MTAP24 多个时间的组合包

28 FU-A 分片的单元

29 FU-B 分片的单元

30-31 没有定义

RTP单次发送有上限????2种RTP打包：拆包or不拆包

在IP网络中，当要传输的IP报文大小超过【最大传输单元MTU】时就会产生IP分片情况。

（若交给底层协议拆包容易出问题→→→主动拆分NALU再打包成RTP包后发送）

**H264视频流的RTP封包**

1.RTP打包原则

RTP的包长度必须要小于MTU(最大传输单元)，IP协议中MTU的最大长度为1500字节。

除去IP报头（20字节）、UDP报头（8字节）、RTP头（12字节），

所有RTP有效载荷（即NALU内容）的长度不得超过1460字节。

2.RTP协议的报文结构

开始12个八进制出现在每个RTP包中，而CSRC标识列表仅出现在混合器插入时。各段含义如下：

①版本（V）

version (V): 2 bits 2位，标识RTP版本,协议初始版本为0，RFC3550中规定的版本号为2。。

②填充标识（P）

padding (P): 1 bit 1位，如设置填充位，在包末尾包含了额外的附加信息，它不属于有效载荷。

附加信息的最后一个字节表示额外附加信息的长度（包含该字节本身）。

该字段之所以 存在是因为某些加密算法需要固定大小的填充字，或为在底层协议数据单元中携带几个RTP包。

③扩展（X）

extension (X): 1 bit 1位，如果该位被设置，则在固定的头部后存在一个扩展头部，格式定义在RFC3550 5.3.1节。

④CSRC计数（CC）

CSRC count (CC): 4 bits 4位，CSRC计数包括紧接在固定头后标识CSRC个数。

⑤标记（M）

marker (M): 1 bit 1位，标记解释由设置定义，目的在于允许重要事件在包流中标记出来。

设置可定义其他标示位，或通过改变位数量来指定没有标记位，该位的功能依赖于 profile的定义。

profile可以改变该位的长度，但是要保持marker和payload type总长度不变（一共是8 bit）。。

或M：标示位，1 位。如果当前 NALU为一个接入单元最后的那个NALU，那么将M位置 1；

或者当前RTP 数据包为一个NALU 的最后的那个分片时（NALU 的分片在后面讲述），M位置 1。

其余情况下M 位保持为 0。

⑥载荷类型（PT）

payload type (PT): 7 bits 7位，记录后面资料使用哪种 Codec ， receiver 端找出相应的 decoder 解碼出來，

该位标记着RTP packet所携带信息的类型，标准类型列出在RFC3551中。

如果接收方不能识别该类型，必须忽略该packet。

⑦系列号

sequence number:16 bits 16位，系列号随每个RTP数据包发送后而增加1，

接收方可以根据该序列号重新排列数据包顺序，或者探测包损失。

系列号初值是随机的，使对加密的文本攻击更加困难。

⑧时间戳

timestamp: 32 bits 32位，时标反映RTP数据包中第一个八进制数的采样时刻，

采样时刻必须从单调、线性增加的时钟导出，以允许同步与抖动计算。

时标可以让receiver端知道在正确的时间将资料播放出来。

实际中当采用”分片封包模式“打包RTP时，当一个NALU打包完毕时，时间戳更一次。

1、单一NALU的RTP包

对于 NALU 的长度小于 MTU 大小的包, 一般采用单一 NAL 单元模式.

对于一个原始的 H.264 NALU 单元常由 [Start Code] [NALU Header] [NALU Payload] 三部分组成,

其中 Start Code 用于标示这是一个NALU 单元的开始, 必须是 "00 00 00 01" 或 "00 00 01", NALU 头仅一个字节,

其后都是 NALU 单元内容. 打包时去除 "00 00 01" 或 "00 00 00 01" 的开始码, 把其他数据封包的 RTP 包即可.

如有一个 H.264 的 NALU 是这样的:

[00 00 00 01 67 42 A0 1E 23 56 0E 2F ... ] 这是一个序列参数集 NAL 单元. [00 00 00 01] 是四个字节的开始码,

67 是 NALU 头, 42 开始的数据是 NALU 内容.

封装成 RTP 包将如下:

[ RTP Header ] [ 67 42 A0 1E 23 56 0E 2F ]

即只要去掉 4 个字节的开始码就可以了.

2、组合NALU的RTP包

当 NALU 的长度特别小时, 可以把几个 NALU 单元封在一个 RTP 包中.

例：如有一个 H.264 的 NALU 是这样的:

[00 00 00 01 67 42 A0 1E 23 56 0E 2F ... ] [00 00 00 01 68 42 B0 12 58 6A D4 FF ... ]

封装成 RTP 包将如下:

[ RTP Header ] [78 (STAP-A头，占用1个字节)] [第一个NALU长度 (占用两个字节)] [ 67 42 A0 1E 23 56 0E 2F ] [第二个NALU长度 (占用两个字节)] [68 42 B0 12 58 6A D4 FF ... ]

3、分片NALU的RTP包：FU\_indicator和FU\_head（RTP分包时的包头）

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).

FU-A的分片格式：数据比较大的H264视频包，被RTP分片发送。12字节的RTP头后面跟随的就是FU-A分片

FU\_indicator： F 禁止位 NRI 重要标识位????即拆包的nalu自身的NRI ---- F与NRI 保存了 NALU的前3位 type RTP打包头类型，FU-A时type=28

FU\_header：S　　开始位　1表示分片NAL单元的开始，反之=0E　　结束位　1表示分片NAL单元的结束，反之=0。

R　　保留位　必须为0，接收者必须忽略该位。type　NALU数据类型 ????NALU\_header　　----　　保存了 NALU 类型（即NALU的后5位）

如有一个 H.264 的 NALU 是这样的:

[00 00 00 01 65 42 A0 1E 23 56 0E 2F ... 02 17 C8 FD F1 B9 C7 53 59 72 ... CB FF FF F4 1A D5 C4 18 A8 ... F1 B9 C7 1D A5 FA 13 0B ...]

封装成 RTP 包将如下（注意：下面去掉了开始码和NALU头部，但是在FU\_header的type存放了NALU数据类型）

[ RTP Header ] [ 7C 85 42 A0 1E 23 56 0E 2F ...] [ RTP Header ] [ 7C 05 02 17 C8 FD F1 B9 C7 53 59 72 ...] [ RTP Header ] [ 7C 05 CB FF FF F4 1A D5 C4 18 A8 ...] [ RTP Header ] [ 7C 45 F1 B9 C7 1D A5 FA 13 0B ...]

**拆包和解包：**

发送端—拆包：NAL\_header与分片后的FU的单元头有如下关系： NAL\_header前三位为FU\_indicator的前三位 NAL\_header后五位为FU\_header的后五位 接收端—解包：将所有的分片包组合还原成原始的NAl包 nal\_unit\_type = (fu\_indicator & 0xe0) | (fu\_header & 0x1f)