## 一、码流的组织形式

在 H264 中完全没有I帧、P帧、B帧、IDR帧的概念，之所以沿用这些说法是为了表明数据的编码模式。I、B、P各帧是根据压缩算法的需要，是人为定义的,它们都是实实在在的物理帧。一般来说，I帧的压缩率是7（跟JPG差不多），P帧是20，B帧可以达到50。可见使用B帧能节省大量空间，节省出来的空间可以用来保存多一些I帧，这样在相同码率下，可以提供更好的画质。

H264压缩方式

H264采用的核心算法是帧内压缩和帧间压缩，帧内压缩是生成I帧的算法，帧间压缩是生成B帧和P帧的算法。

帧内（Intraframe）压缩也称为空间压缩（Spatialcompression）。当压缩一帧图像时，仅考虑本帧的数据而不考虑相邻帧之间的冗余信息，这实际上与静态图像压缩类似。帧内一般采用有损压缩算法，由于帧内压缩是编码一个完整的图像，所以可以独立的解码、显示。帧内压缩一般达不到很高的压缩，跟编码jpeg差不多。

帧间（Interframe）压缩的原理是：相邻几帧的数据有很大的相关性，或者说前后两帧信息变化很小的特点。也即连续的视频其相邻帧之间具有冗余信息,根据这一特性，压缩相邻帧之间的冗余量就可以进一步提高压缩量，减小压缩比。帧间压缩也称为时间压缩（Temporalcompression），它通过比较时间轴上不同帧之间的数据进行压缩。帧间压缩一般是无损的。帧差值（Framedifferencing）算法是一种典型的时间压缩法，它通过比较本帧与相邻帧之间的差异，仅记录本帧与其相邻帧的差值，这样可以大大减少数据量。

压缩方式说明

* 分组，也就是将一系列变换不大的图像归为一个组，也就是一个序列，也可以叫GOP（画面组）；
* 定义帧，将每组的图像帧归分为I帧、P帧和B帧三种类型；
* 预测帧， 以I帧做为基础帧,以I帧预测P帧,再由I帧和P帧预测B帧;
* 数据传输， 最后将I帧数据与预测的差值信息进行存储和传输。

I帧:即Intra-coded picture（帧内编码图像帧），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）。又称为内部画面 (intra picture)，I 帧通常是每个 GOP（MPEG 所使用的一种视频压缩技术）的第一个帧，经过适度地压缩，做为随机访问的参考点，可以当成图象。在MPEG编码的过程中，部分视频帧序列压缩成为I帧；部分压缩成P帧；还有部分压缩成B帧。I帧法是帧内压缩法，也称为“关键帧”压缩法。I帧法是基于离散余弦变换DCT（Discrete Cosine Transform）的压缩技术，这种算法与JPEG压缩算法类似。采用I帧压缩可达到1/6的压缩比而无明显的压缩痕迹。

## 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。

## I帧编码流程：

1.进行帧内预测，决定所采用的帧内预测模式。

2.像素值减去预测值，得到残差。

3.对残差进行变换和量化。

4.变长编码和算术编码。

5.重构图像并滤波，得到的图像作为其它帧的参考帧。

例如：在视频会议系统中，终端发送给MCU（或者MCU发送给终端）的图像，并不是每次都把完整的一幅幅图片发送到远端，而只是发送后一幅画面在前一幅画面基础上发生变化的部分。如果在网络状况不好的情况下，终端的接收远端或者发送给远程的画面就会有丢包而出现图像花屏、图像卡顿的现象，在这种情况下如果没有I帧机制来让远端重新发一幅新的完整的图像到本地（或者本地重新发一幅新的完整的图像给远端），终端的输出图像的花屏、卡顿现象会越来越严重，从而造成会议无法正常进行。  
在视频画面播放过程中，若I帧丢失了，则后面的P帧也就随着解不出来，就会出现视频画面黑屏的现象；若P帧丢失了，则视频画面会出现花屏、马赛克等现象。  
在视频会议系统中I帧只会在会议限定的带宽内发生，不会超越会议带宽而生效。I帧机制不仅存在于MCU中，电视墙服务器、录播服务器中也存在。就是为了解决在网络状况不好的情况下，出现的丢包而造成的如图像花屏、卡顿，而影响会议会正常进行。

P帧:即Predictive-coded Picture（前向预测编码图像帧）。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。

B帧:即Bidirectionally predicted picture（双向预测编码图像帧)。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，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。采用运动预测的方式进行帧间双向预测编码

## 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帧或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,因为它只反映丙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

## 为什么需要B帧：

从上面的看，我们知道I和P的解码算法比较简单，资源占用也比较少，I只要自己完成就行了，P呢，也只需要解码器把前一个画面缓存一下，遇到P时就使用之前缓存的画面就好了，如果视频流只有I和P，解码器可以不管后面的数据，边读边解码，线性前进，大家很舒服。那么为什么还要引入B帧？

网络上的电影很多都采用了B帧，因为B帧记录的是前后帧的差别，比P帧能节约更多的空间，但这样一来，文件小了，解码器就麻烦了，因为在解码时，不仅要用之前缓存的画面，还要知道下一个I或者P的画面（也就是说要预读预解码），而且，B帧不能简单地丢掉，因为B帧其实也包含了画面信息，如果简单丢掉，并用之前的画面简单重复，就会造成画面卡（其实就是丢帧了），并且由于网络上的电影为了节约空间，往往使用相当多的B帧，B帧用的多，对不支持B帧的播放器就造成更大的困扰，画面也就越卡。

## GOP(序列)和IDR：

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。  
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。H.264 引入 IDR 图像是为了解码的重同步，当解码器解码到 IDR 图像时，立即将参考帧队列清空，将已解码的数据全部输出或抛弃，重新查找参数集，开始一个新的序列。这样，如果前一个序列出现重大错误，在这里可以获得重新同步的机会。IDR图像之后的图像永远不会使用IDR之前的图像的数据来解码。  
一个序列就是一段内容差异不太大的图像编码后生成的一串数据流。当运动变化比较少时，一个序列可以很长，因为运动变化少就代表图像画面的内容变动很小，所以就可以编一个I帧，然后一直P帧、B帧了。当运动变化多时，可能一个序列就比较短了，比如就包含一个I帧和3、4个P帧。  
在视频编码序列中，GOP即Group of picture（图像组），指两个I帧之间的距离，Reference（参考周期）指两个P帧之间的距离。两个I帧之间形成一组图片，就是GOP（Group Of Picture）。

在码率不变的前提下，GOP值越大，P、B帧的数量会越多，平均每个I、P、B帧所占用的字节数就越多，也就更容易获取较好的图像质量；Reference越大，B帧的数量越多，同理也更容易获得较好的图像质量。

GOP结构一般有两个数字，如M=3，N=12。M指定I帧和P帧之间的距离，N指定两个I帧之间的距离。上面的M=3，N=12，GOP结构为：IBBPBBPBBPBBI。在一个GOP内I frame解码不依赖任何的其它帧，p frame解码则依赖前面的I frame或P frame，B frame解码依赖前最近的一个I frame或P frame 及其后最近的一个P frame。

IDR（Instantaneous Decoding Refresh）即时解码刷新。 在编码解码中为了方便，将GOP中首个I帧要和其他I帧区别开，把第一个I帧叫IDR，这样方便控制编码和解码流程，所以IDR帧一定是I帧，但I帧不一定是IDR帧；IDR帧的作用是立刻刷新,使错误不致传播,从IDR帧开始算新的序列开始编码。I帧有被跨帧参考的可能,IDR不会。

I帧不用参考任何帧，但是之后的P帧和B帧是有可能参考这个I帧之前的帧的。

每个GOP一定是以一个I帧开始的，但是却不一定指代的是两个I帧之间的距离。

因为一个GOP内可能包含几个I帧，只有第一个I帧（也就是**第一帧**）才是关键帧。

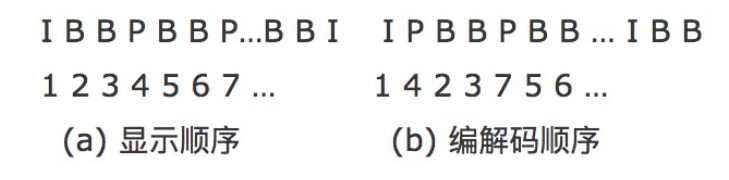
在H.264中，I帧分为普通I帧和IDR帧(特殊I帧); 在H.264中，是IDR帧阻断了误差的积累, IDR帧后面的帧都不能参考该IDR帧前面的帧, 普通的I帧并没有阻断误差的积累，普通I帧后面的帧可以参考该I帧之前的帧. 在MPEG2中，I帧阻断了误差的积累，I帧后面的帧不可以参考该I帧之前的帧。从这个意义上说，H.264中的IDR帧颇有MPEG2中I帧的味道。

# PTS和DTS：

DTS（Decoding Time Stamp）：即解码时间戳，这个时间戳的意义在于告诉播放器该在什么时候解码这一帧的数据。  
PTS（Presentation Time Stamp）：即显示时间戳，这个时间戳用来告诉播放器该在什么时候显示这一帧的数据。

虽然 DTS、PTS 是用于指导播放端的行为，但它们是在编码的时候由编码器生成的。

在视频采集的时候是录制一帧就编码一帧发送一帧的，在编码的时候会生成 PTS，这里需要特别注意的是 frame（帧）的编码方式，在通常的场景中，编解码器编码一个 I 帧，然后向后跳过几个帧，用编码 I 帧作为基准帧对一个未来 P 帧进行编码，然后跳回到 I 帧之后的下一个帧。编码的 I 帧和 P 帧之间的帧被编码为 B 帧。之后，编码器会再次跳过几个帧，使用第一个 P 帧作为基准帧编码另外一个 P 帧，然后再次跳回，用 B 帧填充显示序列中的空隙。这个过程不断继续，每 12 到 15 个 P 帧和 B 帧内插入一个新的 I 帧。P 帧由前一个 I 帧或 P 帧图像来预测，而 B 帧由前后的两个 P 帧或一个 I 帧和一个 P 帧来预测，因而编解码和帧的显示顺序有所不同。

编码器采集到的帧是这个样子的： I B B P B B P

它的显示顺序，也就是PTS应该是这样：1 2 3 4 5 6 7

编码器的编码顺序是： 1 4 2 3 7 5 6

推流顺序也是按照编码顺序去推的，即： I P B B P B B

那么接收断收到的视频流也就是： I P B B P B B

这时候去解码，也是按照收到的视频流一帧一帧去解的了，接收一帧解码一帧，因为在编码的时候已经按照 I、B、P的依赖关系编好了，接收到数据直接解码就好了。那么解码顺序是：

I P B B P B B

DTS：1 2 3 4 5 6 7

PTS：1 4 2 3 7 5 6

可以看到解码出来对应的 PTS 不是顺序的，为了正确显示视频流，这时候我们就必须按照 PTS 重新调整解码后的 frame(帧)，即：

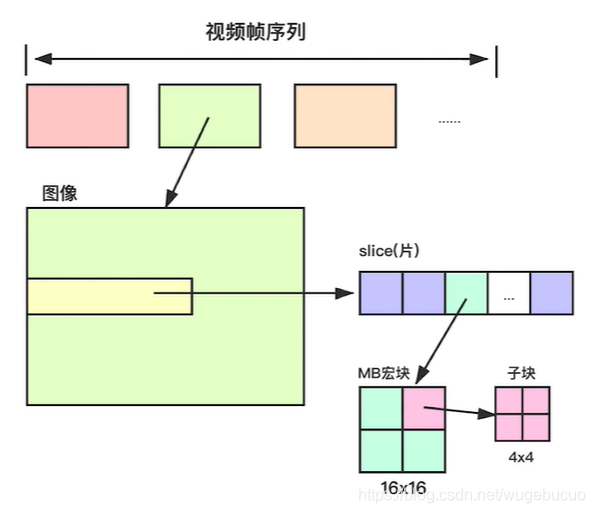
I B B P B B P

DTS：1 3 4 2 6 7 5

PTS：1 2 3 4 5 6 7

另外，并不是一定要使用B帧。在实时互动直播系统中，很少使用B帧。主要的原因是压缩和解码B帧时，由于要双向参考，所以它需要缓冲更多的数据，且使用的CPU也会更高。由于实时性的要求，所以一般不使用它。不过对于播放器来说，遇到带有B帧的H264数据是常有的事儿。在没有B帧的情况下，存放帧的顺序和显示帧的顺序就是一样的，PTS和DTS的值也是一样的。

H264 码流的组织形式从大到小排序是：视频序列(video sequence)、图像(frame/field-picture)、片组(slice group)、片(slice)、宏块(macroblock)、子块(sub-block)、像素(pixel)。

上图中我们可以看到视频帧序列每一帧图像是由slice构成的，每一个slice是由多个宏块构成的，在实际传输的过程中，一般一帧图像就是一个slice，没有分那么细。

## 二、码流功能

1.H264的主要目标是为了有高的视频压缩比和良好的网络亲和性，为了达成这两个目标，H264的解决方案是将系统框架分为两个层面，分别是视频编码层面（VCL：Video Coding Layer）和网络抽象层面（NAL：Network Coding Layer），H.264原始码流(裸流)是由⼀个接⼀个NALU组成。

2.VLC层是对核心算法引擎、块、宏块及片的语法级别的定义，负责有效表示视频数据的内容，最终输出编码完的数据SODB；

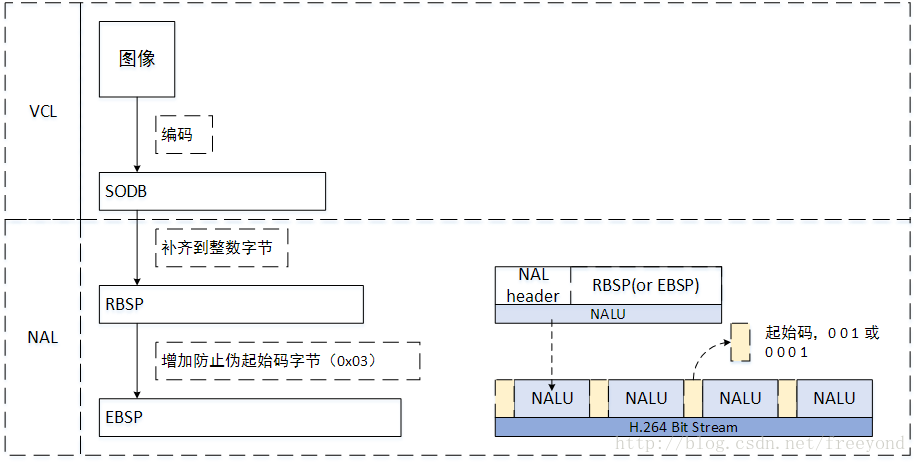
3.NAL层定义了片级以上的语法级别（如序列参数集和图像参数集），负责以网络所要求的恰当方式去格式化数据并提供头信息，以保证数据适合各种信道和存储介质上的传输。NAL层将SODB打包成RBSP然后加上NAL头组成一个NALU单元。

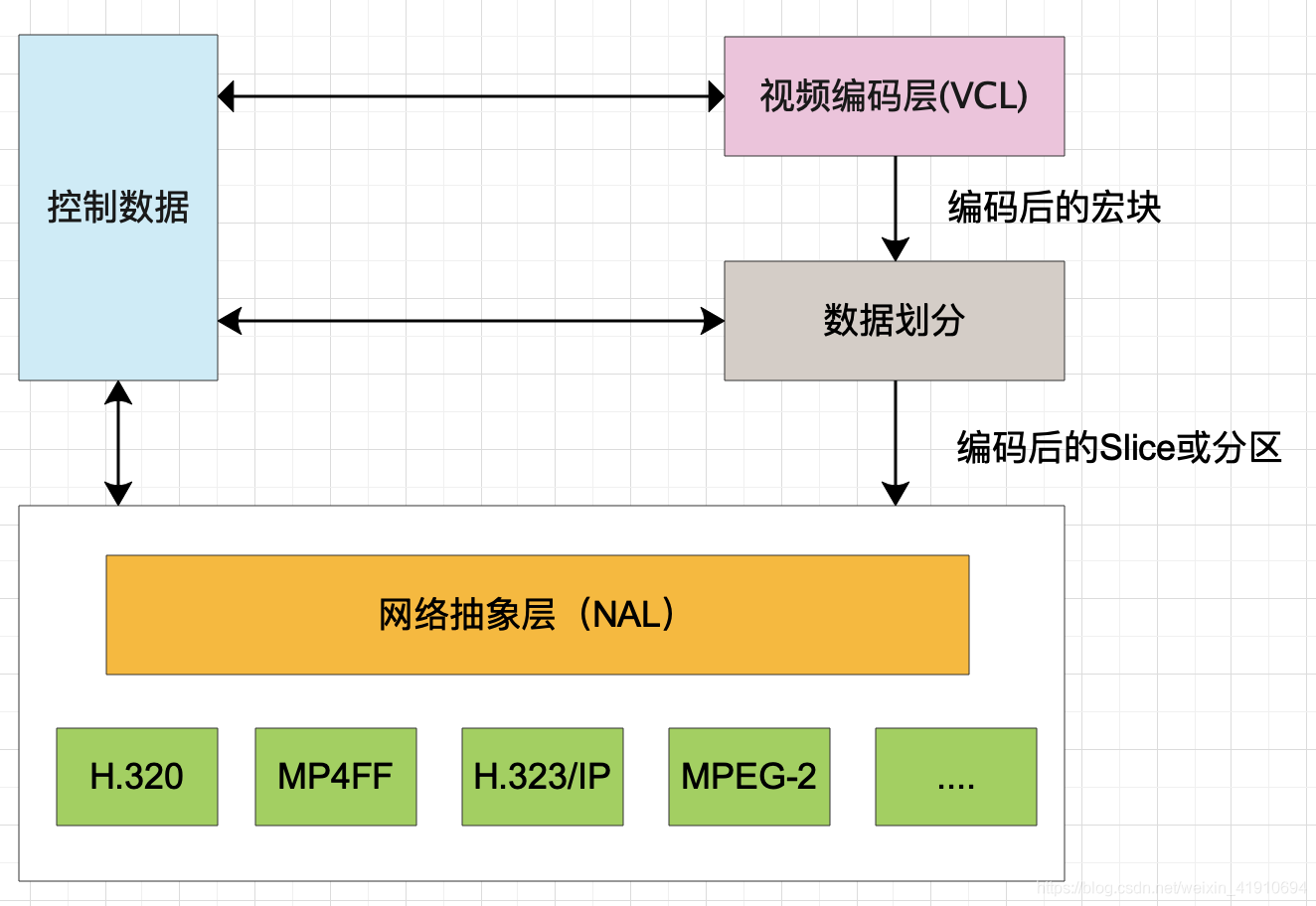
从码流功能的角度可以分为两层：视频编码层（VCL）和网络提取层（NAL）

* VCL：进行视频编解码，包括预测（帧内预测和帧间预测），DCT 变化和量化，熵编码和切分数据等功能，是为了实现更高的视频压缩比。
* NAL：负责以网络所要求的恰当的方式对 VCL 数据进行打包和传送。 在传输的过程每个包以太网是1500字节，而H264的帧往往会大于1500字节，所以要进行拆包，将一个帧拆成多个包进行传输，所有的拆包或者组包都是通过NAL层去处理的。

VCL 是管理 H264 的视频数据层，是为了实现更高的视频压缩比，那 VCL 究竟是怎么管理 H264 视频数据的呢？抛开 H264 压缩算法细节来看就 3 步：

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. 压缩切分后的 VCL 数据会包装成为 NAL 中的一部分。



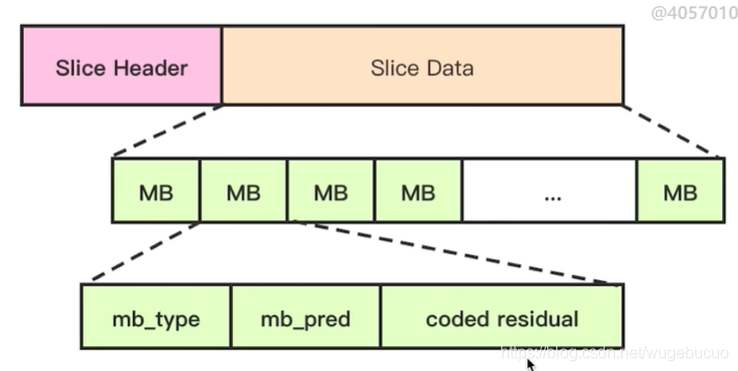


## 

## 三、视频编解码（VCL）

在VCL层中，切片与宏块划分的具体情况如下，slice是由header和data组成：

header中保存的信息较少，主要包括帧类型，GOP中的解码帧序号，预测权重，滤波相关信息。如果PPS中没有开启预测权重和滤波，这里就没有对应的信息。

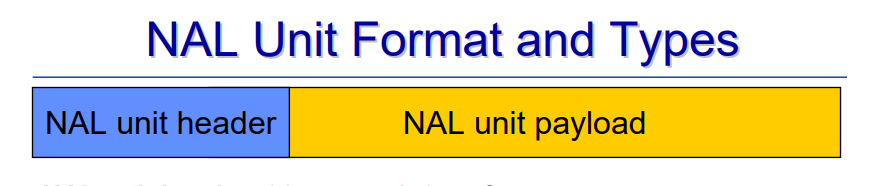
data中由很多的宏块(MacroBlock)组成，在宏块中存储的包括宏块的类型 mb\_type，宏块的预测值mb\_pred 和残差值 codec residual  


## 四、网络提取层（NAL）

NAL，英文全称为 Network Abstraction Layer，这块和 H264 压缩算法无关，涉设计出 NAL 的目的就是为了获得 “network-friendly”，即为了实现良好的网络亲和性，即可适用于各种传输网络。

### 3.1 NAL单元 - NALU

NALU 的格式如下图所示：

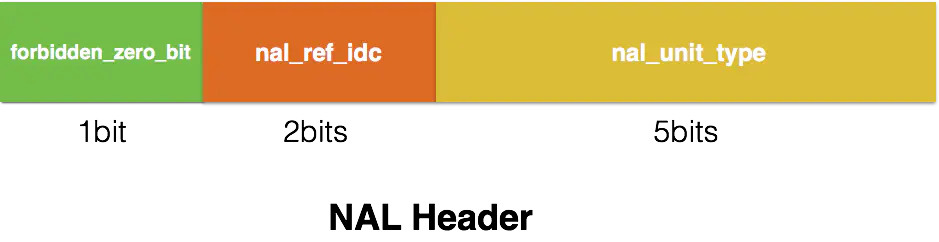


很明显，NALU 由头和身体两个部分组成：

* 头：一般存储标志信息，譬如 NALU 的类型。前面讲过 NAL 会打包 VCL 数据，但是这并不意味着所有的 NALU 负载的都是 VCL，也有一些 NALU 仅仅存储了和编解码信息相关的数据；
* 身体：存储了真正的数据。但实际上，这块也会相对比较复杂，前面其实也提到过 H264 的一个目的是“网络友好性”，说白了就是能够很好地适配各种传输格式。所以根据实际采用数据传输流格式，也会对这部分数据格式再进行处理。

### 3.2 NALU Header

首先，NALU Header只占 1 个字节，即 8 位，其组成如下图所示：



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

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

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

|F|NRI| Type |

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

* forbidden\_zero\_bit

在网络传输中发生错误时，会被置为 1，告诉接收方丢掉该单元；否则为 0。

在 H.264 规范中规定了这一位必须为 0。

* nal\_ref\_idc  
  用于表示当前NALU的重要性，取00~11，值越大，越重要。

如00的NALU解码器可以丢弃它而不影响图像的回放，11表示其可能有SPS、PPS和作为参考帧的slice等等比较重要的数据。  
比如在网传的时候，如果网络卡，那么就需要适当丢掉一些数据，丢掉哪些数据就是根据重要性来进行选择。

* nal\_unit\_type  
  表示 NALU 数据的类型，有以下几种：



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

25 STAP-B 单一时间的组包

26 MTAP16 多个时间的合包

27 MTPA24 多个事件组合包

28 FU-A 一帧单包传输

29 FU-B 一帧分多包传输

30-31 没有定义

h264仅用1-23,24以后的用在RTP H264负载类型头中

其中比较注意的应该是以下几个：

* 1-4：I/P/B帧，如果 nal\_ref\_idc 为 0，则表示 I 帧，不为 0 则为 P/B 帧。
* 5：IDR帧，I 帧的一种，告诉解码器，之前依赖的解码参数集合（接下来要出现的 SPS\PPS 等）可以被刷新了。
* 6：SEI，英文全称 Supplemental Enhancement Information，翻译为“补充增强信息”，提供了向视频码流中加入额外信息的方法。
* 7：SPS，全称 Sequence Paramater Set，翻译为“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。因此该类型保存的是和编码序列相关的参数。
* 8: PPS，全称 Picture Paramater Set，翻译为“图像参数集”。该类型保存了整体图像相关的参数。
* 9：AU 分隔符，AU 全称 Access Unit，它是一个或者多个 NALU 的集合，代表了一个完整的帧，有时候用于解码中的帧边界识别。

NAL分为VCL的NAL单元和非VCL的NAL单元：

nal\_unit\_type为1，2，3，4，5及12的NAL单元称为VCL的NAL单元，其他类型的NAL单元为非VCL的NAL单元。

1. 非VCL的NAL数据类型：

* SPS（Sequence Parameter Set：序列参数集）包含一些通用的参数，比如Profile和Level，比如视频帧的尺寸，参考帧的最大数量等，这些参数对整个Video Sequence或者Programme都是通用的。
* PPS（Picture Parameter Set：图像参数集）包含一些通用的参数，比如熵编码类型，有效的参考图像的数目和初始化参数等，这些参数可以应用到一个Video Sequence或者一部分编码帧。
* SEI（补充增强信息）：这部分参数可作为H264的比特流数据而被传输，每一个SEI信息被封装成一个NAL单元。SEI对于解码器来说可能是有用的，但是对于基本的解码过程来说，并不是必须的。

一个Parameter Set在开始的时候是不活跃的，直到被激活。一个PPS被预先传到解码器，当在一个Slice Header中涉及到的时候，就会被激活，而且直到一个不同的PPS被激活。对于SPS，当一个PPS涉及到它的时候，就会被激活。对于一个以IDR Access Unit开始的Coded Video Sequence，在整个过程中，一个SPS会一直处于活跃状态。因此，一个SPS可以有效的被IDR Slice激活。

2. VCL的NAL数据类型

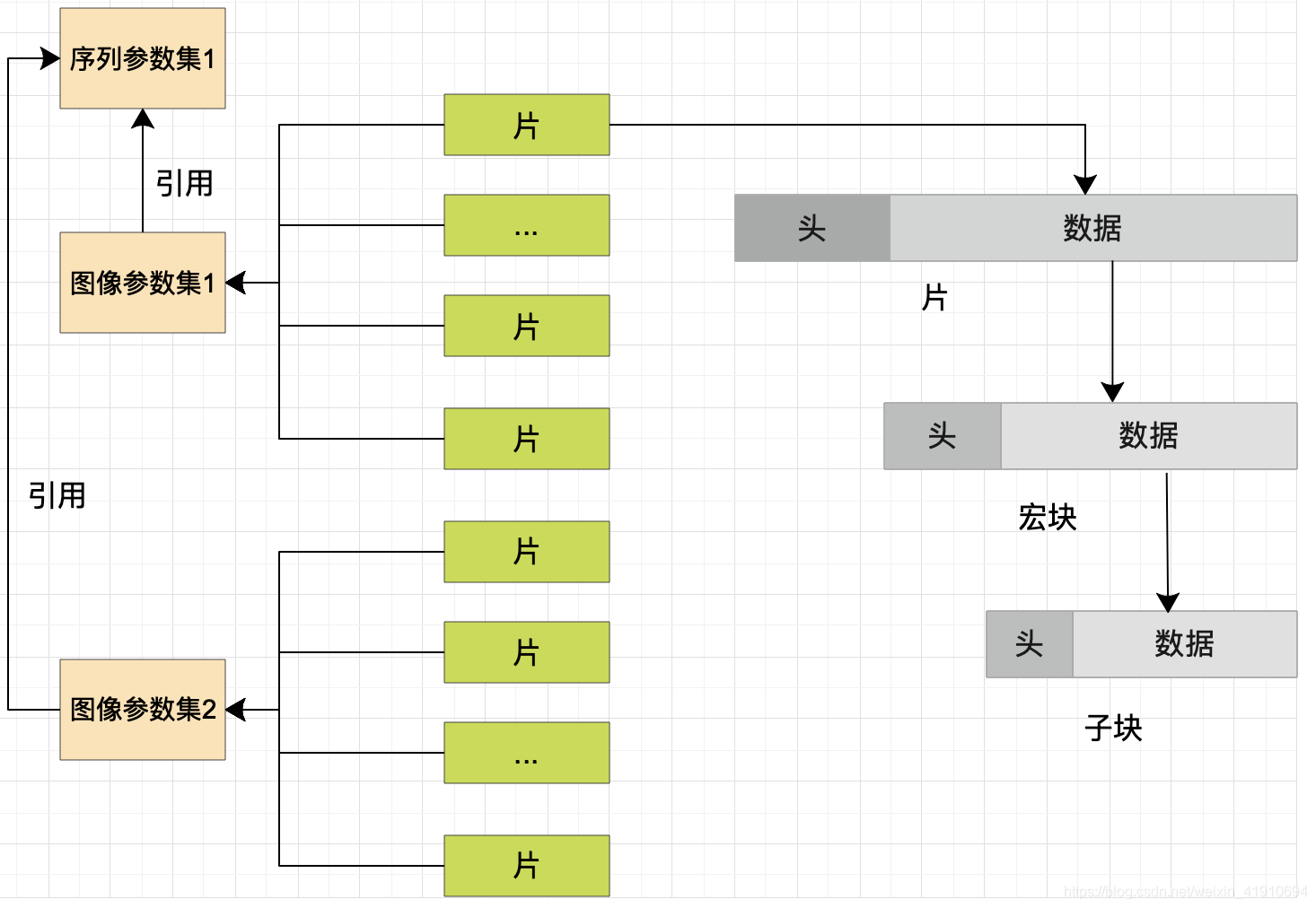
* 头信息块，包括宏块类型，量化参数，运动矢量。这些信息是最重要的，因为离开他们，被的数据块种的码元都无法使用。该数据分块称为A类数据分块。
* 帧内编码信息数据块，称为B类数据分块。它包含帧内编码宏块类型，帧内编码系数。对应的slice来说，B类数据分块的可用性依赖于A类数据分块。和帧间编码信息数据块不同的是，帧内编码信息能防止进一步的偏差，因此比帧间编码信息更重要。
* 帧间编码信息数据块，称为C类数据分块。它包含帧间编码宏块类型，帧间编码系数。它通常是slice种最大的一部分。帧间编码信息数据块是不重要的一部分。它所包含的信息并不提供编解码器之间的同步。C类数据分块的可用性也依赖于A类数据分块，但与B类数据分块无关。

以上三种数据块每种分割被单独的存放在一个NAL单元中，因此可以被单独传输。

参数集是携带解码参数的NAL单元，参数集对于正确解码是非常重要的，在一个有损耗的传输场景中，传输过程中比特列或包可能丢失或损坏，在这种网络环境下，参数集可以通过高质量的服务来发送，比如向前纠错机制或优先级机制。

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，SPS,PPS 需要在 I 帧前出现，不然解码器没法解码。而 SPS,PPS 出现的频率也跟不同应用场景有关，对于一个本地 h264 流，可能只要在第一个 I 帧前面出现一次就可以，但对于直播流，每个 I 帧前面都应该插入 sps 或 pps，因为直播时客户端进入的时间是不确定的。



#### **3.3 H264的NAL单元与片，宏之间的联系**

1. 为什么数据NAL单元中有这么多数据类型，这个SLICE又是什么东西，为什么不直接是编码后出来的原始字节序列载荷，所以我觉得在这里再讲述帧所细分的一些片和宏的概念应该是比较合适的，也是能够参照上下文更能理解这些概念的位置，又能给这些困惑做一个合理一点的解释，所以在此做一个描述：

1帧（一幅图像） = 1~N个片（slice） //也可以说1到多个片为一个片组

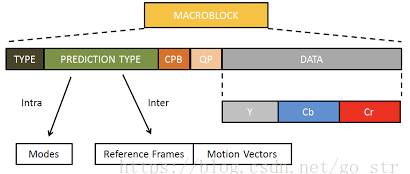
1个片 = 1~N个宏块（Marcroblock）

1个宏块 = 16\*16的YUV数据（原始视频采集数据）

1. 从数据层次角度来说，一幅原始的图片可以算作广义上的一帧，帧包含片组和片，片组由片来组成，片由宏块来组成，每个宏块可以是4\*4、8\*8、16\*16像素规模的大小，每个片都是一个独立的编码单位。
2. 如果不采用 FMO（灵活宏块排序） 机制，则一幅图像只有一个片组；
3. 如果不使用多个片，则一个片组只有一个片；
4. 如果不采用 DP（数据分割）机制，则一个片就是一个 NALU，一个 NALU 也就是一个片。
5. 否则，一个片的组成需要由 三个 NALU 组成，也就是上面说到的A、B、C类数据块。

每个分片也包含着头和数据两部分，分片头中包含着分片类型、分片中的宏块类型、分片帧的数量以及对应的帧的设置和参数等信息，而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。

宏块中包含了宏块类型、预测类型、Coded Block Pattern、Quantization Parameter、像素的亮度和色度数据集等等信息，如下图：



### 需要注意的几点：

H.264/AVC标准对送到解码器的NAL单元顺序是有严格要求的，如果NAL单元的顺序是混乱的，必须将其重新依照规范组织后送入解码器，否则解码器不能够正确解码。

序列参数集NAL单元必须在传送所有以此参数集为参考的其他NAL单元之前传送，不过允许这些NAL单元中间出现重复的序列参数集NAL单元。所谓重复的详细解释为：序列参数集NAL单元都有其专门的标识，如果两个序列参数集NAL单元的标识相同，就可以认为后一个只不过是前一个的拷贝，而非新的序列参数集。

图像参数集NAL单元必须在所有以此参数集为参考的其他NAL单元之前传送，不过允许这些NAL单元中间出现重复的图像参数集NAL单元，这一点与上述的序列参数集NAL单元是相同。

一幅图像由若干片组成，每片包含一系列的宏块（MB）。MB的排列可按光栅扫描顺序，也可不按扫描顺序。每个片独立解码，不同片的宏块不能用于自身片中作预测参考。因此，片的设置不会造成误码扩散。灵活的宏块排序FMO是H.264的一大特色，适用于H.264的基本档次和扩展档次的应用。 图像内部预测机制，例如帧内预测或运动矢量预测，仅允许用同一片组里的空间相邻的宏块。FMO通过宏块分配映射技术，把每个宏块分配到不按扫描顺序的片 中。FMO模式划分图像的模式各种各样，重要的有棋盘模式、矩形模式等。当然FMO模式也可以使一帧中的宏块顺序分割，使得分割后的片的大小小于无线网络 的MTU尺寸，经过FMO模式分割后的图像数据分开进行传输。使用FMO的代价是稍微降低了编码效率（因为它打破了原先非邻居MB之间的预测），而且在高度优化的环境中，有较高的时延。

通常情况下，一个宏块的数据是存放在一起而组成片的，数据划分使得一个片中的宏块数据重新组合，把宏块语义相关的数据组成一个划分，由划分来组装片。H.264视频编码标准使用了三种不同类型的数据分割：  
1． A型分割 ：A型分割是头信息划分，包括宏块类型、量化参数和运动矢量，这个信息是最重要的。   
2． B型分割 ：B型分割是帧内信息划分，包括帧内CBPs和帧内系数。帧内信息可以阻止错误的传播，该型数据分割要求给定分片的A型分割有效，相对于帧间信息，帧内信息能更好地阻止漂移效应，因此它比帧间分割更为重要。  
3． C型分割：C型分割是帧间信息划分，包括帧间CBPs和帧间系数，一般情况下它是编码分片的最大分区。帧间分割是最不重要的，它的使用要求A型分割有效。

当使用数据分割时，源编码器把不同类型的分割安排在3个不同的缓冲器中，同时分片的尺寸必须进行调整以保证小于MTU长度，因此是编码器而 不是NAL来实现数据分割。在解码器上，所有分割用于信息重建。这样，如果帧内或帧间信息丢失了，有效的帧头信息仍然能用来提高错误隐藏效率，即有效的宏 块类型和运动矢量，保留了宏块的基本特征，从而仍可获得一个相当高的信息重构质量，而仅仅丢失了细节信息。

如果profile/level中规定允许乱序slice(FMO)，则I帧可以随意顺序(棋盘模式、矩形模式等等)传输，而B、P帧中的coded slice以及A分割slice data可以随意顺序传输。相同slice\_id的A分割slice data优先于B分割，而B分割又优先于C分割。

如果profile/level中规定不允许乱序slice(FMO)，则I帧的VCL nalu顺序应该按照MB地址递增（光栅扫描）的顺序进行传输。B、P帧的coded以及A分割slice data按照MB地址递增（光栅扫描）顺序传输。同样的，相同slice\_id的A分割slice data优先于B分割，而B分割又优先于C分割。

### 3.4 NALU Payload

很少有资料会称身体部分为 Payload，绝大部分资料对 NALU 组成的定义是这样子的：

NALU = NALU Header + SODB // 定义1

NALU = NALU Header + RBSP // 定义2

NALU = NALU Header + EBSP // 定义3

于是新的问题来了：SODB，RBSP和EBSP都是什么东西呢？

**SODB**

英文全称 String Of Data Bits，称原始数据比特流，就是最原始的编码/压缩得到的数据。

由VCL层产生，数据长度不一定是8的倍数，所以处理起来比较麻烦。

**RBSP**

英文全称 Raw Byte Sequence Payload，又称原始字节序列载荷，是编码后的数据流。

和 SODB 关系如下：

RBSP = SODB + RBSP Trailing Bits（RBSP尾部补齐字节）

算法是在SODB最后一位补1，不按字节对齐补0，如果补齐0，不知道在哪里结束，所以补1，如果不够8位则按位补0。

**EBSP**

英文全称 Encapsulated Byte Sequence Payload，称为扩展字节序列载荷。

和 RBSP 关系如下：

EBSP ：RBSP插入防竞争字节（`0x03`）

这里说明下防止竞争字节（0x03）：读者可以先认为 H264 会插入一个叫做 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。

若内部的连续4个字节数据出现了以下情况时

0x 00 00 00

0x 00 00 01

0x 00 00 02

0x 00 00 03

在两个0字节之后会插入值为3的一个字节：

0x 00 00 03 00

0x 00 00 03 01

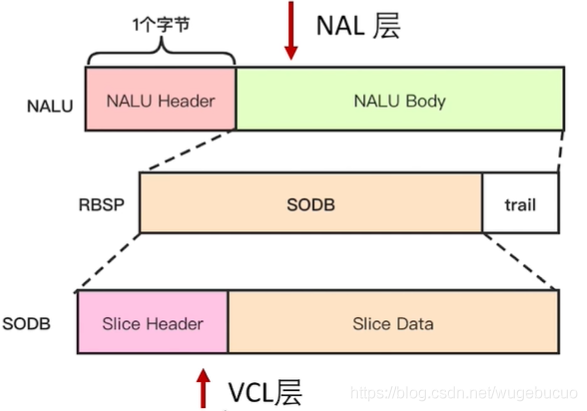
0x 00 00 03 02

0x 00 00 03 03

防止NAL内部出现和起始码一样的冲突

在00后面插入一个03，形成的就是Extended Byte String Payload

如果需要对其解析，需要将03去除后进行解析。



## 

## 五、码流解析

H264 码流实际可以理解为由一个一个的 NALU 单元组成。（下图中的 RBSP 类似 NALU Payload）



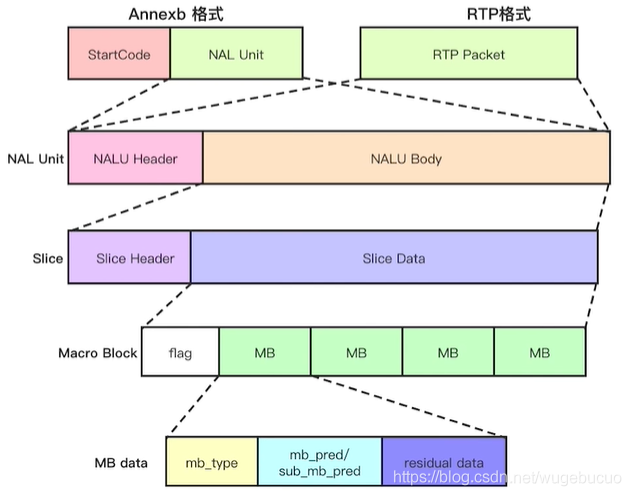
前面提到的一帧图像（I 帧, P 帧, B 帧）就是一个 NALU 单元，NALU 单元除了代表图像外还能包含其他类型的数据，如 PPS 和 SPS。

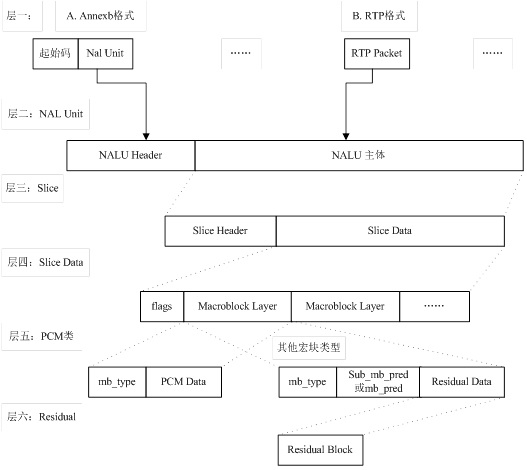
## 六、H264详细的分层结构

实际上H264码流包含了两种格式，Annexb和RTP格式的。

在文件中保存的，每一个NAL单元前面都有一个startcode，00开头的起始码，这样由Startcode和NAL单元构成的就是Annexb格式。

如果只是在网上传输，不包含startcode，直接传输NAL单元叫RTP码流。





* 第一层：比特流。该层有两种格式：Annexb 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。
* 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。

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）。

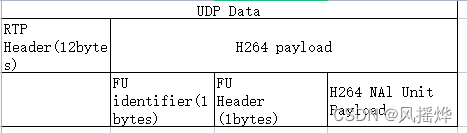
### H264在网络中传输

在UDP或者TCP单次发包过程中，每次MTU为1500字节，所以当单次传输NALU时，有可能数据包大小大于MTU（1500字节），所以要进行切包传输。分为以下两种情况：

当NALU Size小于MTU时，整个NALU包可以发送

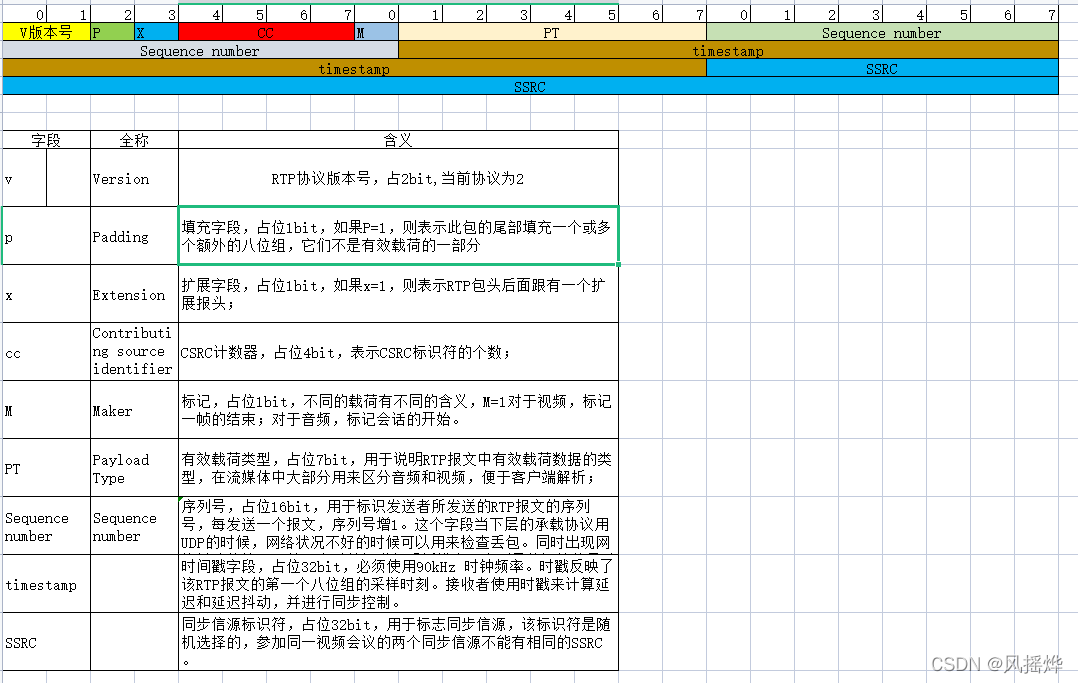
当NALU Size 大于MTU时，需要切包发送。

H264裸码流数据在网络中传输会转成RTP传输格式，格式如下图所示



RTP = RTP header + H264payload

RTP header是由12个字节组成，具体数据格式如下图所示



## FU identifier和FU header



## 七、扩展：怎么区分 NALU 的边界？

了解了 NALU 之后，关于 H264 格式，还有一个问题：解码器怎么知道一个 NALU 要结束了？或者说它怎么区分 NALU 的边界？

要回答这个问题，就必须了解 H264 的打包方式，通俗来说是H264 如何组织一连串的 NALU 为完整的 H264 码流。目前 H264 主流的两种格式：

* Annex-B：本文关于 NALU 的很多细节介绍都是 Annex-B，它依靠前文提到的 Start Code 来分隔 NALU，打包方式如下：

[start code]--[NALU]--[start code]--[NALU]...

* AVCC：笔者对这个格式了解的不多，从网上找到很多资料知道以下几点：
  + 由 NALU 和 extradata/sequence header 组成，由于在 extradata/sequence header 中存储了 NALU 的长度，因此 NALU Payload 不需要做字节对齐，不过防竞争字节还是有的；
  + SPS 和 PPS 被放在了 extradata/sequence header。
  + 打包方式如下：

[SIZE (4 bytes)]--[NAL]--[SIZE (4 bytes)]--[NAL]... // 请注意，SIZE一般为4字节，但是具体以实际为准

## 八、SPS和PPS

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。

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。

除了序列参数集SPS之外，H.264中另一重要的参数集合为图像参数集Picture Paramater Set(PPS)。通常情况下，PPS类似于SPS，在H.264的裸码流中单独保存在一个NAL Unit中，只是PPS NAL Unit的nal\_unit\_type值为8；

而在封装格式中，PPS通常与SPS一起，保存在视频文件的文件头中。

SPS及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。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。

通常情况下：

H.264码流第一个 NALU是 SPS（序列参数集Sequence Parameter Set）

H.264码流第二个 NALU是 PPS（图像参数集Picture Parameter Set）

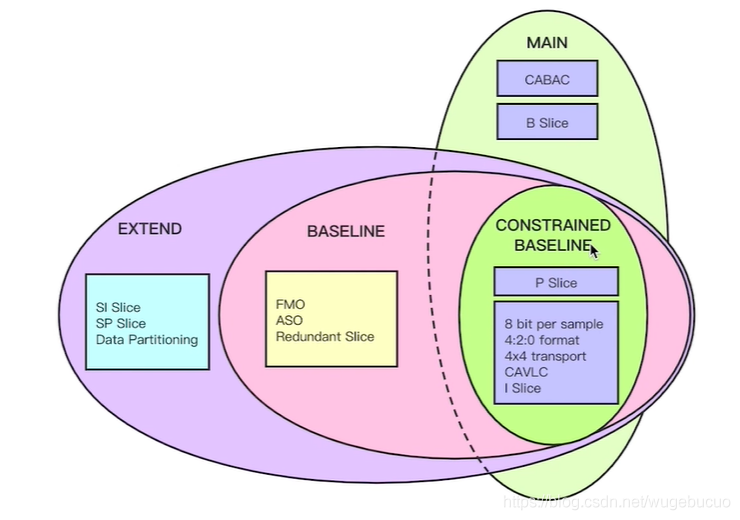
H.264码流第三个 NALU 是 IDR（即时解码器刷新）

在对SPS PPS有基本了解以后我们来具体看一下SPS和PPS保存了哪些参数。

SPS中两个重要的参数

* H264 Profile：对视频压缩特性的描述，Profile越高，说明采用了越高级的压缩特性
* H264 Level：Level是对视频的描述，Level越高，视频的码率、分辨率、fps越高。

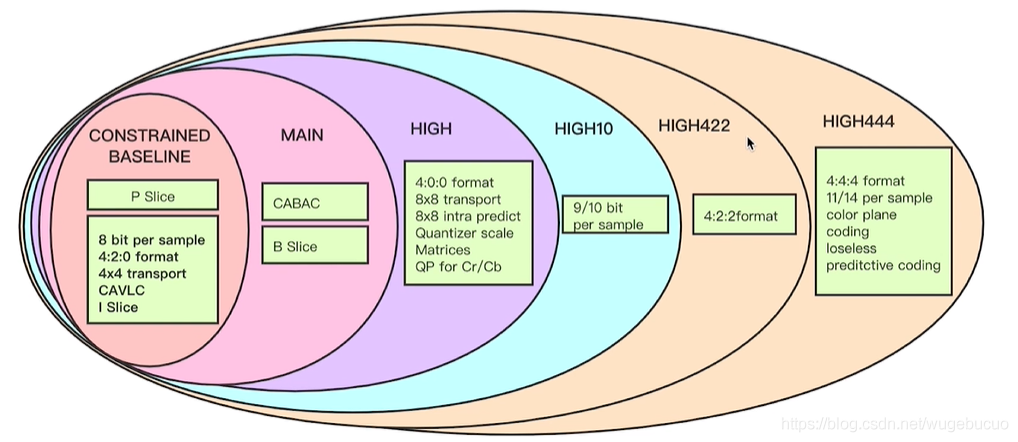
下图展示了H264 Profile的发展过程



Profile主要分成了两极，第一极以CONSTRAINTS BASELINE为核心发展而来的是MAIN PROFILE，第二极是以CONSTARINT BASELINE发展而来的BASELINE和EXTEND。

核心CONSTARINT BASELINE中包含一些核心的P帧，I帧的压缩技术，以及比较老的CAVLC的无损压缩技术，发展到MAINPROFILE出现了B帧，和新的无损压缩技术CABAC,也使得压缩率进一步提高。

第二极不常见，所以我们具体阐释常见的第一极，在第一极发展的过程中，出现了更多的更高压缩比的压缩特性。



SPS中与分辨率有关的参数



SPS中与帧有关的参数

* 最大帧数：log2\_max\_frame\_num\_minus4，通过此参数可以了解到一个GOP中有多少帧
* 最大参考帧数：max\_num\_ref\_frames，表示了解码器中参考帧的缓冲队列的大小，如果该值是5，解码器缓冲队列参考帧就是5帧的大小。
* 显示帧序号：pic\_order\_cnt\_type， 可以通过不同的type计算具体的帧是哪一号显示

PPS相较于SPS要简单很多，下面列出PPS中较为重要的参数：

