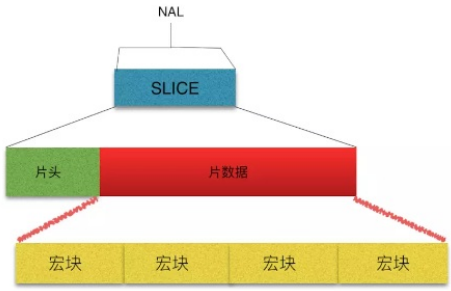
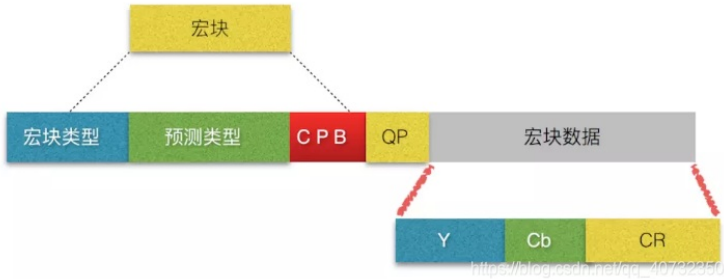
H.264(九)Slice数据和宏块结构

1 Slice的组成

每一个Slice总体来看都由两部分组成，一部分作为Slice header，用于保存Slice的总体信息（如当前Slice的类型等），另一部分为Slice body，通常是一组连续的宏块结构（或者宏块跳过信息），如下图所示：





2 Slice Data结构的定义

在已经实现了一个slice的header部分之后，下面的工作将是研究如何解析一个slice的主体，即Slice Body部分。一个Slice的body部分主要是一个个的宏块结构Macroblock组成，此外还存在一些辅助的信息。标准文档中规定的slice_data()结构如下图：

Slice Data语法表：

slice_data( ) {	C	Descriptor
if(entropy_coding_mode_flag)		
while(!byte_aligned( ))		
cabac_alignment_one_bit	2	f(1)
CurrMbAddr = first_mb_in_slice * (1 + MbaffFrameFlag)		
moreDataFlag = 1		
prevMbSkipped = 0		
do {		
if(slice_type != I && slice_type != SI)		
if(!entropy_coding_mode_flag) {		
mb_skip_run	2	ue(v)
prevMbSkipped = (mb_skip_run > 0)		
for(i=0; i<mb_skip_run; i++)		
CurrMbAddr = NextMbAddress(CurrMbAddr)		
if(mb_skip_run > 0)		
moreDataFlag = more_rbsp_data( )		
} else {		
mb_skip_flag	2	ae(v)
moreDataFlag = !mb_skip_flag		
}		
if(moreDataFlag) {		
if(MbaffFrameFlag && (CurrMbAddr % 2 == 0 || (CurrMbAddr % 2 == 1 && prevMbSkipped)))		
mb_field_decoding_flag	2	u(1) | ae(v)
macroblock_layer( ) 这里有绝大多数数据	2 | 3 | 4	
}		
if(!entropy_coding_mode_flag)		
moreDataFlag = more_rbsp_data( )		
else {		
if(slice_type != I && slice_type != SI)		
prevMbSkipped = mb_skip_flag		
if(MbaffFrameFlag && CurrMbAddr % 2 == 0)		
moreDataFlag = 1		
else {		
end_of_slice_flag	2	ae(v)
moreDataFlag = !end_of_slice_flag		
}		
}		
CurrMbAddr = NextMbAddress(CurrMbAddr)		
} while(moreDataFlag)		
}		

> **cabac_alignment_one_bit** 当熵编码模式是 CABAC 时,此时要求数据字节对齐,即数据从下一个字节的第一个比特开始,如果还没有字节对齐将出现若干个 cabac_alignment_one_bit 作为填充。

> **mb_skip_run** 当图像采用帧间预测编码时， H.264 允许在图像平坦的区域使用“跳跃”块， “跳跃”块本身不携带任何数据，解码器通过周围已重建的宏块的数据来恢复“跳跃”块。
>
> 在表 我们可以看到，当熵编码为 CAVLC 或 CABAC 时， “跳跃”块的表示方法不同。
>
> - 当entropy_coding_mode_flag 为 1，即熵编码为 CABAC 时，是每个“跳跃”块都会有句法元素 mb_skip_flag 指明。
> - 当entropy_coding_mode_flag 等于 0，即熵编码为 CAVLC 时，用一种行程的方法给出紧连着的“跳跃”块的数目，即句法元素mb_skip_run。 mb_skip_run 值的范围 0 to PicSizeInMbs – CurrMbAddr 。
>
> 这两个语法元素都用于表示宏块结构是否可以被跳过。“跳过”的宏块指的是，在帧间预测的slice中，当图像区域平坦时，码流中跳过这个宏块的所有数据，不进行传输，只通过这两个语法元素进行标记。在解码端，跳过的宏块通过周围已经重建的宏块来进行恢复。mb_skip_run用于熵编码使用CAVLC时，用一个语法元素表示连续跳过的宏块的个数；mb_skip_flag用于熵编码使用CABAC时，表示每一个宏块是否被跳过。
>
> **mb_skip_flag** 见上一条， 指明当前宏块是否是跳跃编码模式的宏块。

> **mb_field_decoding_flag** 在帧场自适应图像中，指明当前宏块所属的宏块对是帧模式还是场模式。 0 帧模式； 1 场模式。如果一个宏块对的两个宏块句法结构中都没有出现这个句法元素，即它们都是“跳跃”块时，本句法元素由以下决定：
>
> - 如果这个宏块对与相邻的、左边的宏块对属于同一个片时，这个宏块对的 mb_field_decoding_flag的值等于左边的宏块对的 mb_field_decoding_flag 的值。
> - 否则，这个宏块对的mb_field_decoding_flag的值等于上边同属于一个片的宏块对的 mb_field_decoding_flag 的值。
> - 如果这个宏块对既没有相邻的、上边同属于一个片的宏块对；也没有相邻的、左边同属于一个片的宏块对，这个宏块对的 mb_field_decoding_flag 的值等于 0，即帧模式。
>
> **end_of_slice_flag** 在CABAC模式下的一个标识位，表示是否到了slice的末尾。

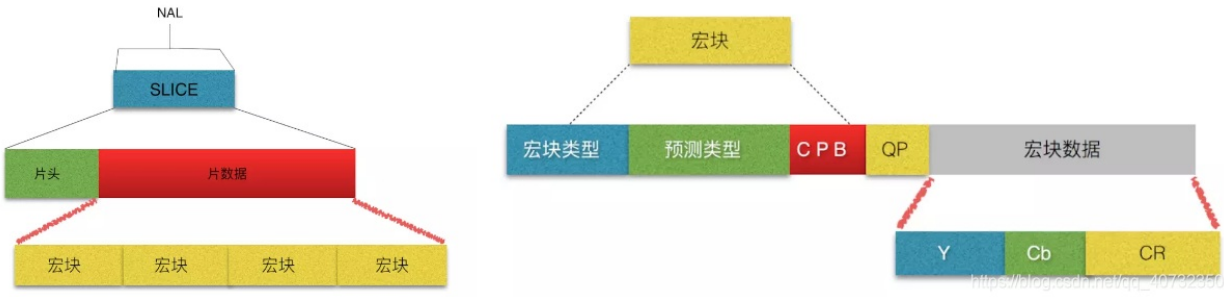
3 H.264的宏块Macroblock

宏块(Macroblock)：

- 编码视频信息的基本单元；
- 在编码过程中提供了较强的灵活性；

一帧图像划分为多个宏块，每个宏块包含：

- 1个16×16像素的亮度像素块
- 两个8×8像素的色度像素块；



常用宏块类型：

- I宏块：采用帧内预测宏块，可能位于I/B/P帧；
- P宏块：采用单向帧间预测，只存在于P帧；
- B宏块：采用双向帧间预测，只存在于B帧；

根据宏块类型的不同，宏块在码流中采用不同结构的语法元素表示

mb_pred(mb_type) {	C	Descriptor
if(MbPartPredMode(mb_type, 0) == Intra_4x4 || MbPartPredMode(mb_type, 0) == Intra_8x8 || MbPartPredMode(mb_type, 0) == Intra_16x16) {		
if(MbPartPredMode(mb_type, 0) == Intra_4x4)		
for(luma4x4BlkIdx=0; luma4x4BlkIdx<16; luma4x4BlkIdx++) {		
prev_intra4x4_pred_mode_flag[luma4x4BlkIdx **]**	2	u(1) | ae(v)
if(!prev_intra4x4_pred_mode_flag[luma4x4BlkIdx])		
rem_intra4x4_pred_mode[luma4x4BlkIdx **]**	2	u(3) | ae(v)
}		
if(MbPartPredMode(mb_type, 0) == Intra_8x8)		
for(luma8x8BlkIdx=0; luma8x8BlkIdx<4; luma8x8BlkIdx++) {		
prev_intra8x8_pred_mode_flag[luma8x8BlkIdx **]**	2	u(1) | ae(v)
if(!prev_intra8x8_pred_mode_flag[luma8x8BlkIdx])		
rem_intra8x8_pred_mode[luma8x8BlkIdx **]**	2	u(3) | ae(v)
}		
if(ChromaArrayType == 1 || ChromaArrayType == 2)		
intra_chroma_pred_mode	2	ue(v) | ae(v)
} else if(MbPartPredMode(mb_type, 0) != Direct) {		
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if((num_ref_idx_l0_active_minus1 > 0 || mb_field_decoding_flag != field_pic_flag) && MbPartPredMode(mb_type, mbPartIdx) != Pred_L1)		
ref_idx_l0[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if((num_ref_idx_l1_active_minus1 > 0 || mb_field_decoding_flag != field_pic_flag) && MbPartPredMode(mb_type, mbPartIdx) != Pred_L0)		
ref_idx_l1[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode (mb_type, mbPartIdx) != Pred_L1)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l0[mbPartIdx **][** 0 **][** compIdx **]**	2	se(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode(mb_type, mbPartIdx) != Pred_L0)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l1[mbPartIdx **][** 0 **][** compIdx **]**	2	se(v) | ae(v)
}		
}		

mb_type 指明当前宏块的类型。 H.264规定，不同的片中允许出现的宏块类型也不同。下表指明在各种片类型中允许出现的宏块种类。

各种片中允许出现的宏块类型

片类型	允许出现的宏块种类
I (slice)	I 宏块
P (slice)	P 宏块、 I 宏块
B (slice)	B 宏块、 I 宏块
SI (slice)	SI 宏块、 I 宏块
SP (slice)	P 宏块、 I 宏块

可以看到， I 片中只允许出现 I 宏块，而 P 片中即可以出现 P 宏块也可以出现 I 宏块，也就是说，在帧间预测的图像中也可以包括帧内预测的图像。其它片也有类似情况。
每一种宏块包含许多的类型。比起以往的视频编码标准， H.264 定义了更多的宏块的类型。

在帧间预测模式下，宏块可以有七种运动矢量的划分方法。
在帧内预测模式下，可以是帧内 16x16 预测，这时可以宏块有四种预测方法，即四种类型；也可以是 4x4 预测，这时每个 4x4 块可以有九种预测方法，整个宏块共有 144 种类型。

mb_type 并不能描述以上所有有关宏块类型的信息。事实上可以体会到， mb_tye 是出现在宏块层的第一个句法元素，它描述跟整个宏块有关的基本的类型信息。在不同的片中 mb_type 的定义是不同的，下面我们分别讨论 I、 P、 B 片中这个句法元素的意义。

a) I 片中的 mb_type

mb_type	类型名称	预测方式	帧内 16x16 的预测模式	CodedBlockPatternChroma	CodedBlockPatternLuma
0	I_4x4	Intra_4x4	无	无	无
1	I_16x16_0_0_0	Intra_16x16	0	0	0
2	I_16x16_1_0_0	Intra_16x16	1	0	0
3	I_16x16_2_0_0	Intra_16x16	2	0	0
4	I_16x16_3_0_0	Intra_16x16	3	0	0
5	I_16x16_0_1_0	Intra_16x16	0	1	0
6	I_16x16_1_1_0	Intra_16x16	1	1	0
7	I_16x16_2_1_0	Intra_16x16	2	1	0
8	I_16x16_3_1_0	Intra_16x16	3	1	0
9	I_16x16_0_2_0	Intra_16x16	0	2	0
10	I_16x16_1_2_0	Intra_16x16	1	2	0
11	I_16x16_2_2_0	Intra_16x16	2	2	0
12	I_16x16_3_2_0	Intra_16x16	3	2	0
13	I_16x16_0_0_1	Intra_16x16	0	0	15
14	I_16x16_1_0_1	Intra_16x16	1	0	15
15	I_16x16_2_0_1	Intra_16x16	2	0	15
16	I_16x16_3_0_1	Intra_16x16	3	0	15
17	I_16x16_0_1_1	Intra_16x16	0	1	15
18	I_16x16_1_1_1	Intra_16x16	1	1	15
19	I_16x16_2_1_1	Intra_16x16	2	1	15
20	I_16x16_3_1_1	Intra_16x16	3	1	15
21	I_16x16_0_2_1	Intra_16x16	0	2	15
22	I_16x16_1_2_1	Intra_16x16	1	2	15
23	I_16x16_2_2_1	Intra_16x16	2	2	15
24	I_16x16_3_2_1	Intra_16x16	3	2	15
25	I_PCM	无	无	无	无

表中 ， Intra_4x4 表示使用帧内 4x4 预测， Intra_16x16 表示使用帧内 16x16 预测。 当使用帧内 16x16时，类型名称由了如下的结构组成： **I_16x16_x_y_z**

其中， x 对应于表中“帧内 16x16 的预测模式”字段的值， y 对应于表中“色度 CBP”字段的值， z 对应于表中“亮度 CBP”的值。

- 帧内 16x16 的预测模式：当使用帧内 16x16 预测时，指定使用何种预测方式，帧内 16x16 共有四种预测模式，第八章中会详细介绍这些预测模式的算法。
- CodedBlockPatternLuma：指定当前宏块色度分量的 CBP， CBP（CodedBlockPattern）是指子宏块残差的编码方案。该变量详细语义见 coded_block_pattern 条目。
- 亮度 CBP：指定当前宏块亮度分量的 CBP，详细语义见 coded_block_pattern 条目。

我们看到，帧内 16x16 宏块类型的 mb_type 语义原比其它宏块类型的复杂，这是因为当使用帧内 16x16 时，整个宏块是一个统一的整体，宏块中各子宏块、 4x4 小块的预测模式信息都是相同的，所以可以把这些信息放入 mb_type ,以减少码流。其它宏块类型的这些信息必须在各子块中另外用句法元素指明。

b) P 片中的 mb_type 。

mb_type	类型名称	宏块分区数目	预测模式 (mb_type,0)	预测模式 (mb_type, 1)	宏块分区宽度 (mb_type)	宏块分区高度 (mb_type)
0	P_L0_16x16	1	Pred_L0	无	16	16
1	P_L0_L0_16x8	2	Pred_L0	Pred_L0	16	8
2	P_L0_L0_8x16	2	Pred_L0	Pred_L0	8	16
3	P_8x8	4	无	无	8	8
4	P_8x8ref0	4	无	无	8	8
无	P_Skip	1	Pred_L0	无	16	16

在表 7.26 中,Pred_Lo 表示用 Lo,即前向预测。如果当前宏块的 mb_type 等于 0 到 4， mb_type 的含义见 表 7.26；当 mb_type 等于 5 到 30 时， mb_type 的含义见 表 7.25, 用 mb_type-5 所得到的值来进行查找。预测模式（mb_type,n）预测模式是 mb_type 的函数， n 是宏块的第 n 个分区。

c) B 片中的 mb_type

如果当前宏块是属于 B 片且 mb_type 等于 0 到 22， mb_type 的含义见 表 7-11；当 mb_type 等于23 到 48 时， mb_type 的含义见 表 7-8, 用 mb_type-23 所得到的值来进行查找。

mb_type	类型名称	宏块分区数目 (mb_type)	预测模式 (mb_type, 0)	预测模式 (mb_type, 1)	宏块分区宽度 (mb_type)	宏块分区高度 (mb_type)
0	B_Direct_16x16	无	Direct	无	8	8
1	B_L0_16x16	1	Pred_L0	无	16	16
2	B_L1_16x16	1	Pred_L1	无	16	16
3	B_Bi_16x16	1	BiPred	无	16	16
4	B_L0_L0_16x8	2	Pred_L0	Pred_L0	16	8
5	B_L0_L0_8x16	2	Pred_L0	Pred_L0	8	16
6	B_L1_L1_16x8	2	Pred_L1	Pred_L1	16	8
7	B_L1_L1_8x16	2	Pred_L1	Pred_L1	8	16
8	B_L0_L1_16x8	2	Pred_L0	Pred_L1	16	8

9	B_L0_L1_8x16	2	Pred_L0	Pred_L1	8	16
10	B_L1_L0_16x8	2	Pred_L1	Pred_L0	16	8
11	B_L1_L0_8x16	2	Pred_L1	Pred_L0	8	16
12	B_L0_Bi_16x8	2	Pred_L0	BiPred	16	8
13	B_L0_Bi_8x16	2	Pred_L0	BiPred	8	16
14	B_L1_Bi_16x8	2	Pred_L1	BiPred	16	8
15	B_L1_Bi_8x16	2	Pred_L1	BiPred	8	16
16	B_Bi_L0_16x8	2	BiPred	Pred_L0	16	8
17	B_Bi_L0_8x16	2	BiPred	Pred_L0	8	16
18	B_Bi_L1_16x8	2	BiPred	Pred_L1	16	8
19	B_Bi_L1_8x16	2	BiPred	Pred_L1	8	16
20	B_Bi_Bi_16x8	2	BiPred	BiPred	16	8
21	B_Bi_Bi_8x16	2	BiPred	BiPred	8	16
22	B_8x8	4	无	无	8	8
无	B_Skip	无	Direct	无	8	8

表中， Pred_Lo 表示使用 Lo，即前向预测， Pred_L1 表示使用 L1，即后向预测， Bipred 表示双向预测，Direct 表示直接预测模式。预测模式（mb_type,n）预测模式是 mb_type 的函数， n 是宏块的第n 个分区。

pcm_alignment_zero_bit 等于 0。
pcm_byte[i] 像素值。前 256 pcm_byte[i] 的值代表亮度像素的值，下一个
(256 * (ChromaFormatFactor - 1)) / 2 个 pcm_byte[i] 的值代表 Cb 分量的值 . 最后一个
(256 * (ChromaFormatFactor - 1)) / 2 个 pcm_byte[i]的值代表 Cr 分量的值。

coded_block_pattern 即 CBP，指亮度和色度分量的各小块的残差的编码方案，所谓编码方案有以下几种：
a) 所有残差（包括 DC、 AC）都编码。
b) 只对 DC 系数编码。
c) 所有残差（包括 DC、 AC）都不编码。
这个句法元素同时隐含了一个宏块中亮度、色度分量的 CBP，所以第一步必须先分别解算出各分量各自 CBP 的值。其中，两个色度分量的 CBP 是相同的。变量 CodedBlockPatternLuma 是亮度分量的 CBP，变量 CodedBlocPatternChroma 是色度分量的 CBP：
对于非 Intra_16x16 的宏块类型：

```
1. CodedBlockPatternLuma = coded_block_pattern % 16;
2. CodedBlockPatternChroma = coded_block_pattern / 16;
```

对于Intra_16x16宏块类型， CodedBlockPatternLuma 和CodedBlockPatternChroma 的值不是由本句法元素给出，而是通过mb_type得到。

- CodedBlockPatternLuma：是一个16位的变量，其中只有最低四位有定义。由于非Intra_16x16的宏块不单独编码DC系数，所以这个变量只指明两种编码方案：残差全部编码或全部不编码。变量的最低位比特从最低位开始，每一位对应一个子宏块，该位等于1时表明对应子宏块残差系数被传送；该位等于0时表明对应子宏块残差全部不被传送，解码器把这些残差系数赋为0。
- CodedBlockPatternChroma：当值为0、 1、 2时有定义，见下表。

CodedBlockPatternChroma 的定义

CodedBlockPatternChroma	定义
0	所有残差都不被传送，解码器把所有残差系数赋为0。
1	只有DC系数被传送，解码器把所有AC系数赋为0。
2	所有残差系数（包括DC、 AC）都被传送。解码器用接收到的残差系 数重建图像。

mb_qp_delta 在宏块层中的量化参数的偏移值。 mb_qp_delta 值的范围是 -26 to +25。 量化参数是在图像参数集、片头、宏块分三层给出的，最终用于解码的量化参数由以下公式得到：

```
QPY = ( QPY,PREV + mb_qp_delta + 52 ) % 52;
```

QPY,PREV 是当前宏块按照解码顺序的前一个宏块的量化参数，我们可以看到， mb_qp_delta 所指示的偏移是前后两个宏块之间的偏移。而对于片中第一个宏块的 QPY,PREV 是由 7-16 式给出

```
QPY,PREV = 26 + pic_init_qp_minus26 + slice_qp_delta;
```