FLV格式分析

大体的解析框架

FLV header

FLV Body

- FLV Tag
  - tag header
  - Script Tag Data结构(脚本类型、帧类型)
  - Audio Tag Data结构(音频类型)
    - AAC AUDIO DATA
  - Video Tag Data结构(视频类型)
    - AVCVIDEOPACKET

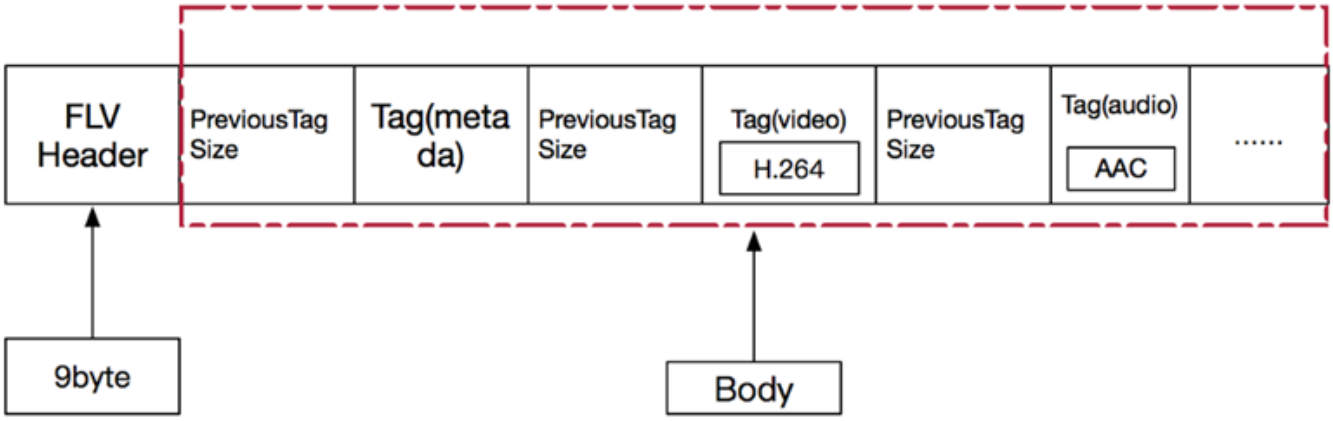
附录

- major_brand && minor_version && compatible_brands
- FLV时间戳计算

FLV(Flash Video)是Adobe公司推出的一种流媒体格式，由于其封装后的音视频文件体积小、封装简单等特点，非常适合于互联网上使用。目前主流的视频网站基本都支持FLV。采用FLV格式封装的文件后缀为.flv。

FLV封装格式是由一个**文件头(file header)**和 **文件体(file Body)**组成。其中，FLV body由一对对的(Previous Tag Size字段 + tag)组成。Previous Tag Size字段 排列在Tag之前，占用4个字节。Previous Tag Size记录了前面一个Tag的大小，用于逆向读取处理。FLV header后的第一个Pervious Tag Size的值为0。

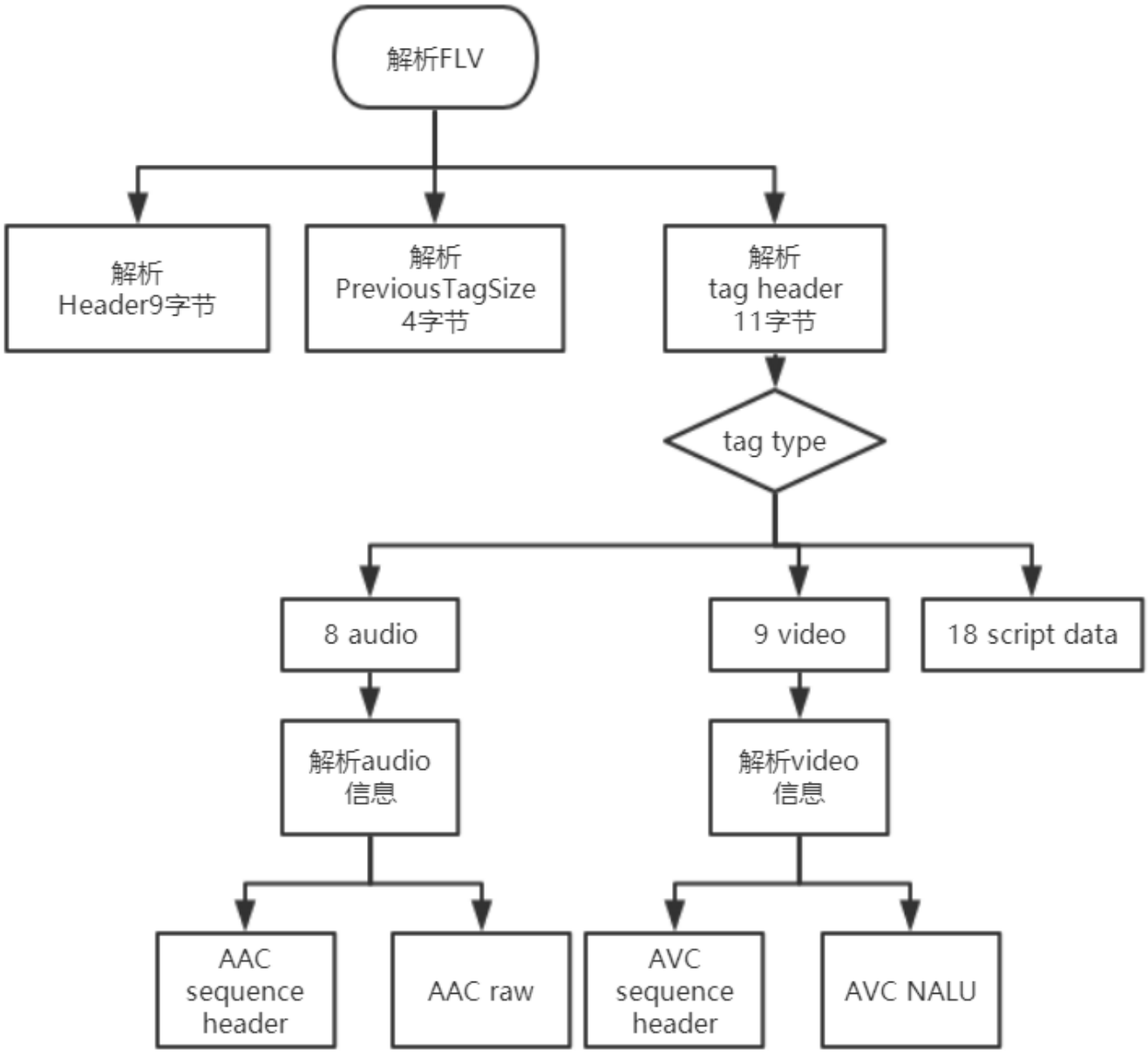
Tag一般可以分为3种类型：脚本(帧)数据类型、音频数据类型、视频数据。FLV数据以大端序进行存储，在解析时需要注意。一个标准FLV文件结构如下图：



FLV文件的详细内容结构如下图：

FLV Header	Signature：3 个字节的文件标识，总为"F","L","V"（0x46,0x4C,0x56）		
	Version：1 个字节的版本标识，目前为 0x01		
	TypeFlags：1 个字节，前 5 位为保留位，必须为 0，第 6 位表示是否存在音频 tag，第 7 位为保留位，必须为 0，第 8 位表示是否存在视频 tag		
	DataOffset：4 个字节，表示从 File Header 起始位置到 File Body 起始位置的字节数（即 File Header 的大小），版本 1 中为 9		
FLV Body	PreviousTagSize0：4 字节，表示前一个 tag 的长度		
	Tag1	Tag Header	TagType：1 个字节，表示 tag 的类型，包括音频（0x08）、视频（0x09）和 script data（0x12），其他的类型值被保留
			DataSize：3 个字节，表示该 tag 的 data 部分的大小
			Timestamp：3 个字节，表示该 tag 的时间戳
			TimestampExtended：1 个字节，表示时间戳的扩展字节，当 24 位数值不够时，以该字节为最高位将时间戳扩展为 32 位数值
			StreamID：3 个字节，总为 0
		Tag Data	Data：不同类型的 tag 的 data 部分的结构各不相同，但 header 部分的结构是相同的
	PreviousTagSize1：前一个 tag（即 tag1）的大小		
	Tag2		
	...		
	PreviousTagSizeN-1		
	TagN		
	PreviousTagSizeN		

大体的解析框架



FLV header

注：在下面的数据type中，UI表示无符号整形，后面跟的数字表示其长度是多少位。比如UI8，表示无符号整形，长度一个字节。UI24是三个字节,UI[8*n]表示多个字节。UB表示位域，UB5表示**一个字节的5位**。可以参考c中的位域结构体。

FLV头占9个字节，用来标识文件为FLV类型，以及后续存储的音视频流。一个FLV文件，每种类型的tag都属于一个流，也就是一个flv文件最多只有一个音频流，一个视频流，不存在多个独立的音视频流在一个文件的情况。

00000 1 0 1

FLV头的结构如下：

Field	Type	Comment

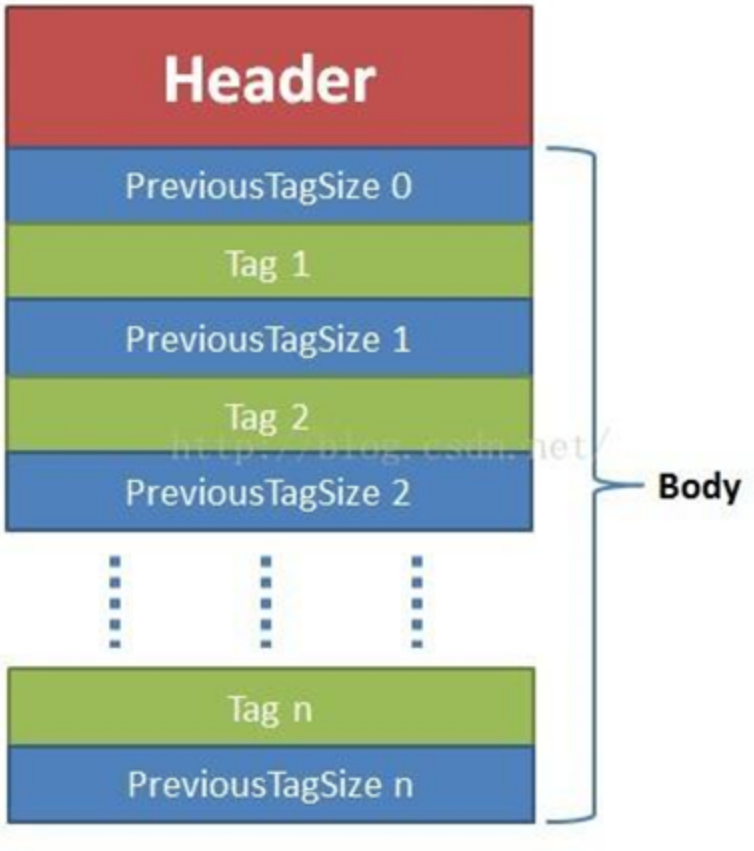
签名	UI8	'F'(0x46)
签名	UI8	'L'(0x4C)
签名	UI8	'V'(0x56)
版本	UI8	FLV的版本。0x01表示FLV版本为1
保留字段	UB5	前五位都为0
音频流标识	UB1	是否存在音频流
保留字段	UB1	为0
视频流标识	UB1	是否存在视频流
文件头大小	UI32	FLV版本1时填写9，表明的是FLV头的大小，为后期的FLV版本扩展使用。包括这四个字节。数据的起始位置就是从文件开头偏移这么多的大小。

FLV Body

FLV Header之后，就是FLV File Body。FLV File Body是由一连串的back-pointers + tags构成。Back-pointer表示Previous Tag Size(前一个tag的字节数据长度)，占4个字节。

```
00 00 01 7F
```

计算出大小 383= 0x0000017F



FLV Tag

每一个Tag也是由两部分组成:**tag header**和**tag data**。Tag Header里存放的是当前tag的类型、数据区(tag data)的长度等信息。

tag header

tag header一般占11个字节的内存空间。FLV tag结构如下：

Field	Type	Comment
Tag类型 Type	UI8	**8:audeo 9:video 18:Script data(脚本数据)** all Others:reserved 其他所有值未使用
数据区大小	UI24	当前tag的数据域的大小，不包含tag header。 Length of the data in the Data field
时间戳Timestamp	UI24	当前帧时戳，单位是毫秒。相对值，第一个tag的时戳总是为0
时戳扩展字段 TimestampExtended	UI8	如果时戳大于0xFFFFFF，将会使用这个字节。这个字节是时戳的高8位，上面的三个字节是低24位。
StreamID	UI24	总是为0

数据域	UI[8*n]	数据域数据

注意：

1. flv文件中Timestamp和TimestampExtended拼出来的是dts。也就是解码时间。Timestamp和TimestampExtended拼出来dts单位为ms。(如果不存在B帧，当然dts等于pts)

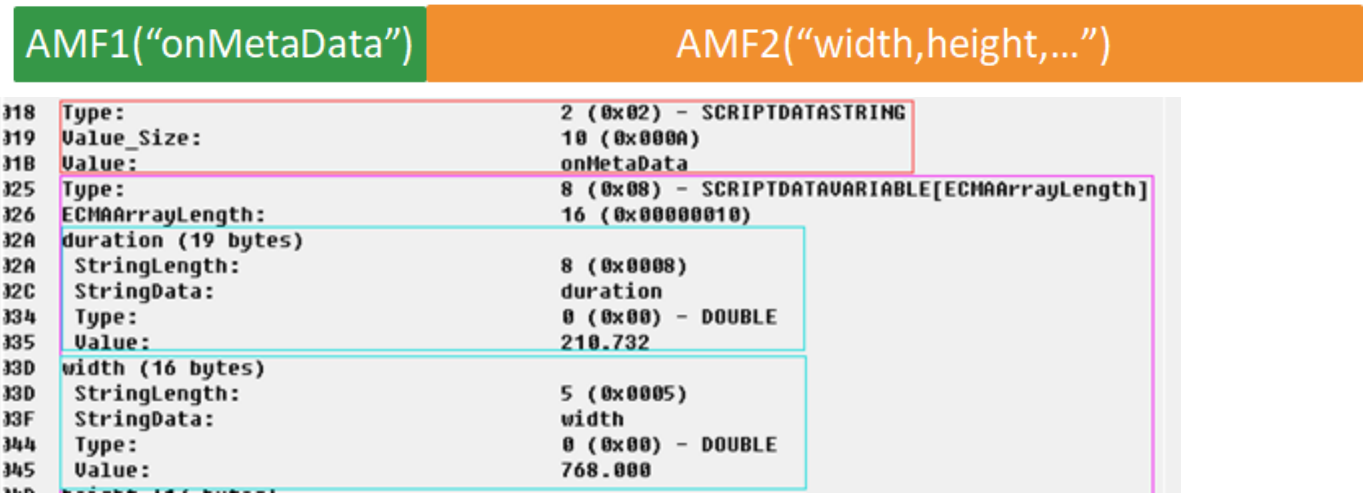
2. CompositionTime 表示PTS相对于DTS的偏移值， 在每个视频tag的第14~16字节， 。显示时间(pts) = 解码时间（tag的第5~8字节） + CompositionTime

CompositionTime的单位也是ms

Script data脚本数据就是描述视频或音频的信息的数据，如宽度、高度、时间等等，一个文件中通常只有一个元数据，音频tag和视频tag就是音视频信息了，采样、声道、频率，编码等信息。

Script Tag Data结构(脚本类型、帧类型)

该类型Tag又被称为**MetaData Tag**,存放一些关于FLV视频和音频的元信息，比如：**duration**、**width**、**height**等。通常该类型Tag会作为FLV文件的第一个tag，并且只有一个，跟在File Header后。该类型Tag DaTa的结构如下所示（source.200kbps.768x320.flv文件为例）：

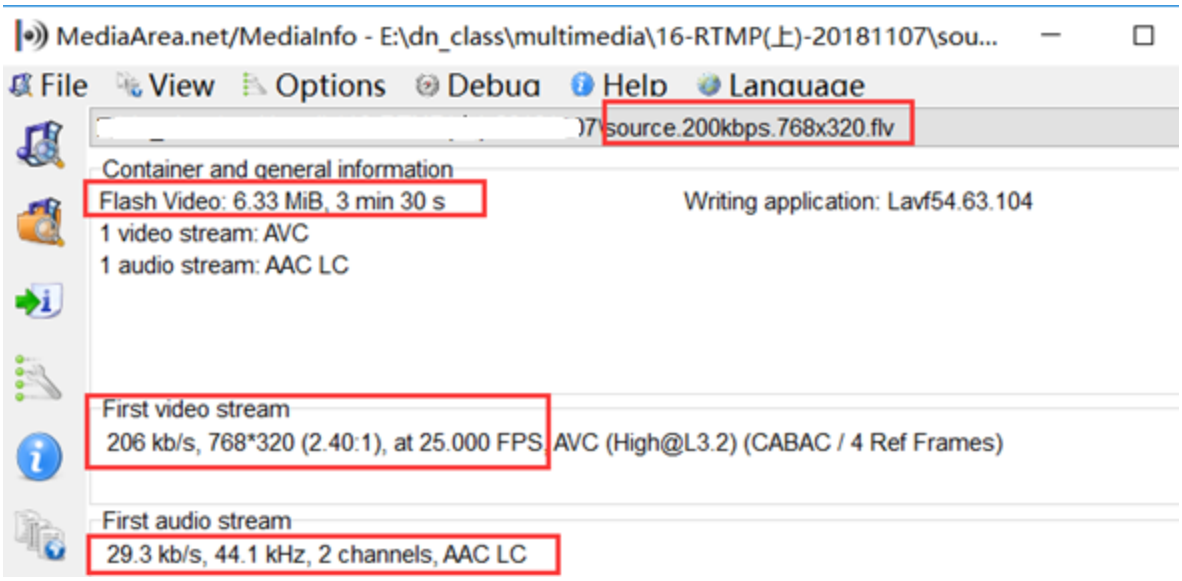


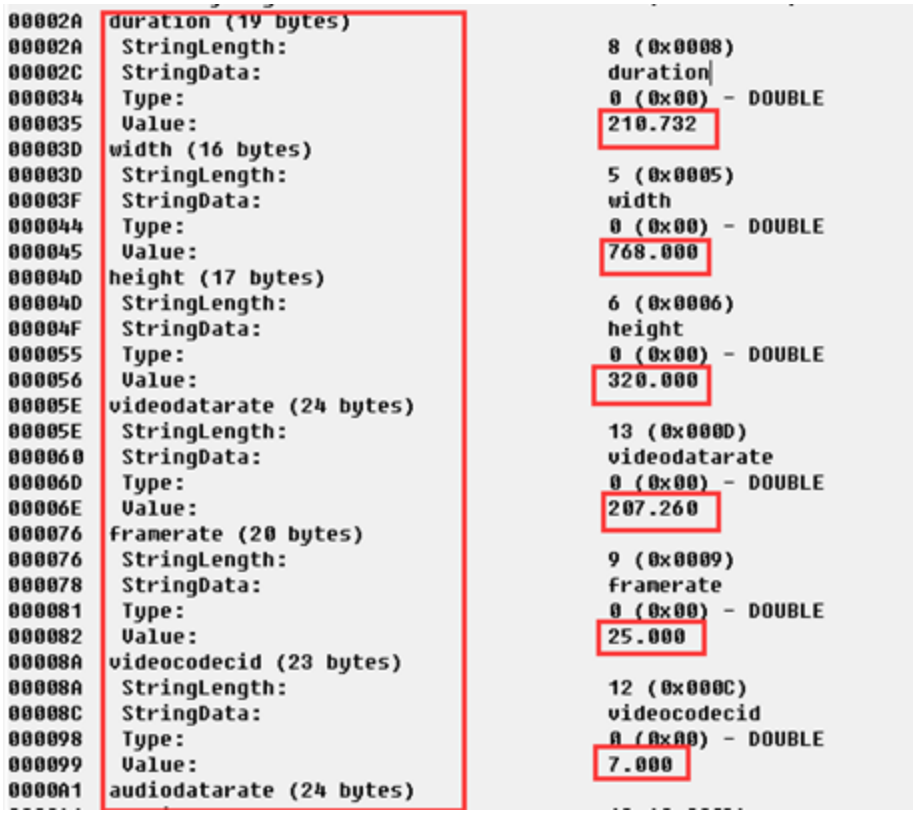
第一个AMF包： 第1个字节表示AMF包类型，一般总是0x02，表示字符串。第2-3个字节为UI16类型值，标识字符串的长度，一般总是0x000A（“onMetaData”长度）。后面字节为具体的字符串，一般总为“onMetaData”（6F,6E,4D,65,74,61,44,61,74,61）。

第二个AMF包： 第1个字节表示AMF包类型，一般总是0x08，表示数组。第2-5个字节为UI32类型值，表示数组元素的个数。后面即为各数组元素的封装，数组元素为元素名称和值组成的对。常见的数组元素如下表所示。

值	Comment	例如
duration	时长(秒)	210.732
width	视频宽度	768.000
height	视频高度	320.000
videodatarate	视频码率	207.260
framerate	视频帧率	25.000
videocodecid	视频编码ID	7.000 (H264为7)
audiodatarate	音频码率	29.329
audiosamplerate	音频采样率	44100.000
stereo	是否立体声	1
audiocodecid	音频编码ID	10.000 (AAC为10)
major_brand	格式规范相关	isom
minor_version	格式规范相关	512
compatible_brands	格式规范相关	isomiso2avc1mp41
encoder	封装工具名称	Lavf54.63.104
filesize	文件大小（字节）	6636853.000

注：Lavf54.63.104即是 *L*ib*avf*ormat version 54.63.104. 即是ffmpeg对于库的版本





Audio Tag Data结构(音频类型)

音频Tag Data区域开始的：

- 第一个字节包含了音频数据的参数信息，
- 第二个字节开始为音频流数据。

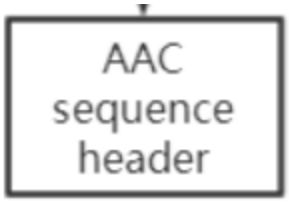
（这两个字节属于tag的data部分，不是header部分）

第一个字节为音频的信息（仔细看spec发现对于AAC而言，比较有用的字段是SoundFormat），格式如下：

Field	Type	Comment
音频格式 SoundFormat	UB4	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
采样率 SoundRate	UB2	0 = 5.5kHz 1 = 11kHz 2 = 22.05kHz 3 = 44.1kHz 对于AAC总是3。但实际上AAC是可以支持到48khz以上的频率(这个参数对于AAC意义不大)。
采样精度 SoundSize	UB1	0 = snd8Bit 1 = snd16Bit 此参数仅适用于未压缩的格式，压缩后的格式都是将其设为1
音频声道 SoundType	UB1	0 = sndMono 单声道 1 = sndStereo 立体声，双声道 对于AAC总是1

If the SoundFormat indicates AAC, the SoundType should be set to 1 (stereo) and the SoundRate should be set to 3 (44 kHz). However, this does not mean that AAC audio in FLV is always stereo, 44 kHz data. Instead, the Flash Player ignores these values and **extracts the channel and sample rate data is encoded in the AAC bitstream.**



第二个字节开始为音频数据（需要判断该数据是真正的音频数据，还是音频config信息）。

Field	Type	Comment
音频数据	UI[8*n]	if SoundFormat == 10 （AAC类型） AACAUDIODATA else Sound data—varies by format

AAC AUDIO DATA

AACAUDIODATA		
Field	**Type**	**Comment**
AACPacketType	UI8	0: AAC sequence header 1: AAC raw
Data	UI8[n]	if AACPacketType == 0 AudioSpecificConfig else if AACPacketType == 1 Raw AAC frame data

The AudioSpecificConfig is explained in ISO 14496-3. AAC sequence header存放的是AudioSpecificConfig结构，该结构则在“ISO-14496-3 Audio”中描述。

《完整版ISO-14496-3(2009-09).pdf 完整版ISO-14496-3(2009-09).pdf》

如果是AAC数据，如果他是AAC RAW，tag data[3] 开始才是真正的AAC frame data。

Table 1.15 – Syntax of AudioSpecificConfig()

Syntax	No. of bits	Mnemonic
AudioSpecificConfig () {		

audioObjectType = GetAudioObjectType();
samplingFrequencyIndex;
if (samplingFrequencyIndex == 0xf) {
samplingFrequency;
}
channelConfiguration;
	
	4
	
	24
	
	4
		
		bslbf
		
		uimsbf
		
		bslbf

sbrPresentFlag = -1; psPresentFlag = -1;		

Table 1.16 – Syntax of GetAudioObjectType()

Syntax	No. of bits	Mnemonic
GetAudioObjectType() {		

audioObjectType;
if (audioObjectType == 31) {
audioObjectType = 32 + **audioObjectTypeExt**;
}
return audioObjectType;
	5
	
	6
		uimsbf
		
		uimsbf

}		

Video Tag Data结构(视频类型)

视频Tag Data开始的：

- 第一个字节包含视频数据的参数信息，
- 第二个字节开始为视频流数据。

第一个字节包含视频信息，格式如下：

Field	Type	Comment
帧类型	UB4	1: keyframe (for AVC, a seekable frame)——h264的IDR，关键帧 2: inter frame (for AVC, a non- seekable frame)——h264的普通帧 3: disposable inter frame (H.263 only) 4: generated keyframe (reserved for server use only) 5: video info/command frame
编码ID	UB4	使用哪种编码类型： 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

第二个字节开始为视频数据

Field	Type	Comment
视频数据	UI[8*n]	If CodecID == 2 H263VIDEOPACKET If CodecID == 3 SCREENVIDEOPACKET If CodecID == 4 VP6FLVVIDEOPACKET If CodecID == 5 VP6FLVALPHAVIDEOPACKET If CodecID == 6 SCREENV2VIDEOPACKET if CodecID == 7 （AVC格式） AVCVIDEOPACKET

AVCVIDEOPACKET

AVCVIDEOPACKET

Field	Type	Comment
AVCPacketType	UI8	0: AVC sequence header 1: AVC NALU 2: AVC end of sequence (lower level NALU sequence ender is not required or supported)
CompositionTime	SI24	if AVCPacketType == 1 Composition time offset else 0
Data	UI8[n]	if AVCPacketType == 0 AVCDecoderConfigurationRecord else if AVCPacketType == 1 One or more NALUs (can be individual slices per FLV packets; that is, full frames are not strictly required) else if AVCPacketType == 2 Empty

（1）CompositionTime 单位毫秒

CompositionTime 每个视频tag（整个tag）的第14~16字节（如果是tag data偏移[3]~[5], [0],[1][2:AVCPackettype] ）（表示PTS相对于DTS的偏移值 ）。

CompositionTime 单位为ms： **显示时间** = 解码时间（tag的第5~8字节,位置索引[4]~[7]） + CompositionTime

（2）AVCDecoderConfigurationRecord

AVC sequence header就是**AVCDecoderConfigurationRecord**结构，该结构在标准文档“**ISO-14496-15 AVC file format**”

📎 ISOIEC 14496-15 Advanced Video Coding (AVC) file format.pdf中有详细说明。

```
aligned(8) class AVCDecoderConfigurationRecord {
   unsigned int(8) configurationVersion = 1;
   unsigned int(8) AVCProfileIndication;
   unsigned int(8) profile_compatibility;
   unsigned int(8) AVCLevelIndication;
   bit(6) reserved = '111111'b;
   unsigned int(2) lengthSizeMinusOne;
   bit(3) reserved = '111'b;
   unsigned int(5) numOfSequenceParameterSets;
   for (i=0; i< numOfSequenceParameterSets;  i++) {
      unsigned int(16) sequenceParameterSetLength ;
      bit(8*sequenceParameterSetLength) sequenceParameterSetNALUnit;
   }
   unsigned int(8) numOfPictureParameterSets;
   for (i=0; i< numOfPictureParameterSets;  i++) {
      unsigned int(16) pictureParameterSetLength;
      bit(8*pictureParameterSetLength) pictureParameterSetNALUnit;
   }
}
```

附录

major_brand && minor_version && compatible_brands

关于上述三个术语的解释，在wiki上摘抄了一段英文放在这儿供大家自己分析（自己的理解可能不对，就不误导大家了），wiki地址为File Type Box。所有从iso base media file format中衍生出来的文件都在www.mp4ra.org上注册，已注册列表为http://mp4ra.org/#/brands。

In order to identify the specifications to which a file based on ISO base media file format complies, brands are used as identifiers in the file format. They are set in a box named File Type Box ('ftyp'), which must be placed in the beginning of the file. It is somewhat analogous to the so-called fourcc code, used for a similar purpose for media embedded in AVI container format.[50] A brand might indicate the type of encoding used, how the data of each encoding is stored, constraints and extensions that are applied to the file, the compatibility, or the intended usage of the file. Brands are a printable four-character codes. A File Type Box contains two kinds of brands. One is "major_brand" which identifies the specification of the best use for the file. It is followed by "minor_version," an informative 4 bytes integer for the minor version of the major brand. The second kind of brand is "compatible_brands," which identifies multiple specifications to which the file complies. All files shall contain a File Type Box, but for compatibility reasons with an earlier version of the specification, files may be conformant to ISO base media file format and not contain a File Type Box. In that case they should be read as if they contained an ftyp with major and compatible brand "mp41" (MP4 v1 —

ISO 14496-1, Chapter 13).[1] Many in-use brands (ftyps) are not registered and can be found on some webpages.[26]

A multimedia file structured upon ISO base media file format may be compatible with more than one concrete specification, and it is therefore not always possible to speak of a single "type" or "brand" for the file. In this regard, the utility of the Multipurpose Internet Mail Extension type and file name extension is somewhat reduced. In spite of that, when a derived specification is written, a new file extension will be used, a new MIME type and a new Macintosh file type.[1]

FLV时间戳计算

题记：时间戳将每一秒分成90000份，即将每一毫秒分成90份　在flv中直接存储的都是毫秒级　在TS存储的是时间戳级
其中TS、flv一般按照编码顺序排列
一个视频tag一般只包含一帧视频的码流
其中视频tag的时间戳对应的是解码时间戳（DTS/90）
当前序列：
编码顺序 I　P　P　B　B　B.....
对应帧号 0　1　5　3　2　4.....
flv对每一个tag都规定了它将要播放的时间戳
每个时间戳都可以对应转换特性的时间
其中script（脚本）、video（视频）、audio（音频）的第一个tag的时间戳值都为0
时间戳占4个字节 其中第四个字节是高位 前三个字节是低位(每个tag的5~8字节)
如6E 8D A8 01 = 0x 01 6E 8D A8 = 24022440

CompositionTime 每个视频tag的第14~16字节（表示PTS相对于DTS的偏移值 ）
CompositionTime 单位为ms　显示时间 = 解码时间（tag的第5~8字节）　+　CompositionTime
例如（注意显示时间最后一个字节是高位）
tag0 （脚本）：时间戳为0
tag1 （视频）：第一个视频时间戳　值为0 无CompositionTime (头信息)
tag2 （音频）：第一个音频时间戳　值为0
tag3 （视频）：00 00 00 00　值：0　00:00:00:00 (解码时间) CompositionTime：0x 00 00 50　值：80　00:00:00:80 I帧 显示时间：　00:00:00: 80　poc=0
tag4 （视频）：00 00 28 00　值：40　00:00:00:40 (解码时间) CompositionTime：0x 00 00 50　值：80　00:00:00:80　P帧　显示时间：　00:00:00: 120　poc=1
tag5 （视频）：00 00 50 00　值：80　00:00:00:80 (显示时间) CompositionTime：0x 00 00 C8　值：200　00:00:00:200　P帧　显示时间：　00:00:00: 280　poc=5

tag6 （音频） ： 00 00 50 00 值：80 00:00:00:80(显示时间)
tag7 （音频） ： 00 00 67 00 值：103 00:00:00:103(显示时间)
tag8 （视频） ： 00 00 78 00 值：120 00:00:00:120 (解码时间) CompositionTime：0x 00 00 50 值：80 00:00:00:80 B帧 显示时间： 00:00:00: 200 poc=3
tag9 （音频） ： 00 00 7E 00 值：126 00:00:00:126(显示时间)
tag10 （音频） ： 00 00 96 00 值：150 00:00:00:150(显示时间)
tag11 （视频） ： 00 00 A0 00 值：160 00:00:00:160(解码时间) CompositionTime：0x 00 00 00 值：00 00:00:00:00 b帧 显示时间： 00:00:00: 160 poc=2
tag12 （音频） ： 00 00 AD 00 值：173 00:00:00:173(显示时间)
tag13 （音频） ： 00 00 C4 00 值：196 00:00:00:196(显示时间)
tag14（视频） ： 00 00 C8 00 值：200 00:00:00:200(解码时间) CompositionTime：0x 00 00 28 值：40 00:00:00:40 b帧 显示时间： 00:00:00: 240 poc=4我们可以看到 每个视频tag相差约 40ms 刚好是25fps视频 每帧视频的播放时长
在上例中，我们会看到按照解码时间排列
编码顺序 I P P B B B.....
对应帧号 0 1 5 3 2 4.....