FLV 细节解析

FLV 文件格式解析：FLV 文件由 FLV header 和 FLV body 组成，FLV body 又由很多 tag 组成，tag 又可以分成三类：audio、video、script，分别代表音频流、视频流、脚本流（保存文件信息，如帧率、分辨率等）。

FLV header 一般为 9 个字节，格式如下：

文件类型	3bytes	总是 FLV（0x460x4C0x56）
版本	1byte	一般为 0x01，表示 FLV version 1
流信息	1byte	第 6bit 为音频标记，第 8bit 为视频标记
header 长度	4bytes	FLV header 的长度，一般为 9

每个 tag 包含两个部分，一是 tag 类型信息部分，15 个字节；二是 tag data。

tag 类型信息格式如下：

previoustagsize	4bytes	前一个 tag 的长度，第一个 tag 为 0
tag 类型	1byte	三类：0x08--audio tag，0x09--video tag，0x12--script tag
数据区长度	3bytes	
时间戳	3bytes	单位毫秒，如果是 script tag 为 0
扩展时间戳	1byte	作为时间戳的高位
streamsID	3bytes	总为 0

tag data 分为 audio、video、script 部分：

audio tag：由 1 个字节的 audio 信息部分和 audio 数据区部分组成，格式如下：

audio 信息	1byte	前 4bits 表示类型： 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 U-law logarithmic PCM 9--reserved 10-AAC 11-Speex 14-MP3 8-kHz 15-Device-specific sound 第 5、6bit 表示采样率，对于 AAC 总为 3： 0--5.5kHz 1--11kHz 2--22kHz 3--44kHz

		第 7bit 表示采样大小： 0--snd8bit 1--snd16bit 第 8bit 表示音频声道数，对于 AAC 总为 1： 0--sndMono 1--sndStereo
audio 数据区	不定	if SoundFormat == 10 AACAUDIODATA else Sound data -- varies by format

针对 AAC 编码部分，audio 数据区定义如下：

AAC packet type	1byte	0--AAC sequence header 1--AAC raw
data	N bytes	if AAC packet type == 0 AudioSpecificConfig else if AAC packet type == 1 Raw AAC frame data

文件中 AudioSpecificConfig 只有一个，出现在第一个 audio tag，结构如下：

```
AudioSpecificConfig()
{
    audioObjectType;                                    5bits
    samplingFrequencyIndex;                             4bits
    if ( samplingFrequencyIndex == 0xf )
        samplingFrequency;                              24bits
    channelConfiguration;                               4bits

    sbrPresentFlag = -1;
    if ( audioObjectType == 5 ) {
        extensionAudioObjectType = audioObjectType;
        sbrPresentFlag = 1;
        extensionSamplingFrequencyIndex;                4bits
        if ( extensionSamplingFrequencyIndex == 0xf )
            extensionSamplingFrequency;                 24bits
        audioObjectType;                                5bits
    }
    else {
        extensionAudioObjectType = 0;
    }

    if ( audioObjectType == 1 || audioObjectType == 2 ||
        audioObjectType == 3 || audioObjectType == 4 ||
        audioObjectType == 6 || audioObjectType == 7 )
```

```
    GASpecificConfig();
if ( audioObjectType == 8 )
    CelpSpecificConfig();
if ( audioObjectType == 9 )
    HvxcSpecificConfig();
if ( audioObjectType == 12 )
    TTSSpecificConfig();
if ( audioObjectType == 13 || audioObjectType == 14 ||
    audioObjectType == 15 || audioObjectType == 16 )
    StructureAudioSpecificConfig();

if ( audioObjectType == 17 || audioObjectType == 19 ||
    audioObjectType == 20 || audioObjectType == 21 ||
    audioObjectType == 22 || audioObjectType == 23 )
    GASpecificConfig();
if ( audioObjectType == 24 )
    ErrorResilientCelpSpecificConfig();
if ( audioObjectType == 25 )
    ErrorResilientHvxcSpecificConfig();
if ( audioObjectType == 26 || audioObjectType == 27 )
    ParametricSpecificConfig();
if ( audioObjectType == 17 || audioObjectType == 19 ||
    audioObjectType == 20 || audioObjectType == 21 ||
    audioObjectType == 22 || audioObjectType == 23 ||
    audioObjectType == 24 || audioObjectType == 25 ||
    audioObjectType == 26 || audioObjectType == 27 ) {
    epConfig;                                              2bits
    if ( epConfig == 2 || epConfig == 3 ) {
        ErrorProtectionSpecificConfig();
    }
    if ( epConfig == 3 ) {
        directMapping;                                     1bit
        if ( ! directMapping ) {
            /* tbd */
        }
    }
}
if ( audioObjectType == 28 )
    SSCSpecificConfig();

if ( extensionAudioObjectType != 5 && bits_to_decode() >= 16 ) {
    syncExtensionType;                                     11bits
    if ( syncExtensionType == 0x2b7 ) {
        extensionAudioObjectType;                          5bits
```

```
            if ( extensionAudioObjectType == 5 ) {
                sbrPresentFlag;                                    1bit
                if ( sbrPresentFlag == 1 ) {
                    extensionSamplingFrequencyIndex;               4bits
                    if ( extensionSamplingFrequencyIndex == 0xf )
                        extensionSamplingFrequency;                24bits
                }
            }
        }
    }
}
```

AudioSpecificConfig 简化格式如下：

audioObjectType	5bits	编码结构类型，AAC-LC 为 2
samplingFrequencyIndex	4bits	音频采样率索引值
channelConfiguration	4bits	音频声道数
GASpecificConfig		该结构包含一下三项
frameLengthFlag	1bit	标志位，用于表明 IMDCT 窗口长度，为 0
dependsOnCoreCoder	1bit	标志位，表明是否依赖 corecoder，为 0
extensionFlag	1bit	扩展标志位，选择了 AAC-LC，这里必须为 0

其中 **samplingFrequencyIndex** 对应关系如下：

0--96000 1--88200 2--64000 3--48000 4--44100 5--32000 6--24000
7--22050 8--16000 9--12000 10-11025 11-8000 12-7350 13-Reserved
14-Reserved 15-frequency is written explictly

文件中 Raw AAC frame data 是 AAC 音频原始数据，不包含 AAC 头数据 ADTS（包含采样率、帧长度等信息，总共 7 字节，分为两部分：adts_fixed_header()和 adts_variable_header()）

adts_fixed_header()

{

syncword	12bits	always 0xFFF
ID	1bit	0--MPEG-4 1--MPEG-2
layer	2bits	always '00'
protection_absent	1bit	
profile	2bits	0--Main profile 1--Low Complexity profile(LC) 2--Scalable Sampling Rate profile(SSR) 3--reserved
sampling_frequency_index	4bits	音频采样率索引值
private_bit	1bit	
channel_configuration	3bits	音频声道数
original_copy	1bit	
home	1bit	

}

channel_configuration 对应关系如下：

0--Defined in AOT Specific Config

1--1 channel：front-center
2--2 channels：front-left，front-right
3--3 channels：front-center，front-left，front-right
4--4 channels：front-center，front-left，front-right，back-center
5--5 channels：front-center，front-left，front-right，back-left，back-right
6--6 channels：front-center，front-left，front-right，back-left，back-right，LFE-channel
7--8 channels：front-center，front-left，front-right，side-left，side-right，back-left，back-right，LFE-channel
8-15：Reserved

```
adts_variable_header()
{
    copyright_identification_bit                1bit
    copyright_identification_start               1bit
    aac_frame_length                            13bits    ADTS 头+AAC 原始流
    adts_buffer_fullness                        11bits    0x7FF 表示码率可变的码流
    number_of_raw_data_blocks_in_frame    2bits
}
```

针对 MP3 编码部分，audio 数据区直接为 MP3 头+MP3 原始数据，MP3 头数据格式如下：

```
MP3FrameHeader
{
    sync                    11bits    同步信息 always 0xFFF
    version                 2bits     版本 00--MPEG 2.5    01--未定义  10--MPEG 2    11--MPEG 1
    layer                   2bits     层 00--未定义  01--Layer 3    10--Layer 2    11--Layer 1
    error_protection        1bit      CRC 校验 0--校验    1--不校验
    bitrate_index           4bits     位率
    sampling_frequency      2bits     采样率索引值
    padding                 1bit      帧长调节
    private                 1bit      保留字
    mode                    2bits     声道模式
    mode_extension          2bits     扩充模式
    copyright               1bit      版权
    original                 1bit      原版标志
    emphasis                2bits     强调模式
}
```

sampling_frequency 对应关系如下：

version	00--MPEG 2.5	01--未定义	10--MPEG 2	11--MPEG 1
MPEG 1：	00--44.1kHz	01--48kHz	10--32kHz	11--未定义
MPEG 2：	00--22.05kHz	01--24kHz	10--16kHz	11--未定义
MPEG 2.5：	00--11.025kHz	01--12kHz	10--8kHz	11--未定义

mode 对应关系如下：
00--立体声 Stereo　01--Joint Stereo　10--双声道　11--单声道

video tag：由 1 个字节的 video 信息部分和 video 数据区部分组成，格式如下：

		前 4bits 表示类型： 1--keyframe(for AVC, a seekable frame) 2--interframe(for AVC, a non-seekable frame) 3--disposable inter frame(H.263 only) 4--generated keyframe(reserved for server use only) 5--video info/command frame
video 信息	1byte	后 4bits 表示编码器 id： 2--Seronson H.263 3--Screen video 4--On2 VP6 5--On2 VP6 without channel 6--Screen video version 2 7--H264/AVC
video 数据区	不定	

针对 H264/AVC 编码部分，video 数据区定义如下：

AVC packet type	1byte	0--AVC sequence header 1--AVC NALU 2--AVC end of sequence
Composition time	3bytes	if AVC packet type == 1 Composition time offset else 0
data	N bytes	if AVC packet type == 0 AVCDecoderConfigurationRecord else if AVC packet type == 1 One or more NALUs else Empty

文件中 AVCsequenceheader 只有一个，出现在第一个 video tag。为了能够从 FLV 文件中获取 NALU，必须要知道前面的 NALU 长度字段所占的字节数（通常是 1、2 或 4 个字节），这个内容必须要从 AVCDecoderConfigurationRecord 中获取，这个遵从标准 ISO/IEC 14496-15 中的 5.2.4 小节。AVCDecoderConfigurationRecord 结构如下：

```
aligned(8) class AVCDecoderConfigurationRecord {
   unsigned int(8) configurationVersion = 1;   //版本号
   unsigned int(8) AVCProfileIndication;       //sps[1]，即 0x67 后面那个字节
   unsigned int(8) profile_compatibility;      //sps[2]
```

```
unsigned int(8) AVCLevelIndication;          //sps[3]
bit(6) reserved = '111111'b;
unsigned int(2) lengthSizeMinusOne;          //NALUnitLength 的长度减 1，一般为 3
bit(3) reserved = '111'b;
unsigned int(5) numOfSequenceParameterSets;       //sps 个数，一般为 1
for ( i=0; i<numOfSequenceParameterSets; i++ ) {
   unsigned int(16) sequenceParameterSetLength;    //sps 的长度
   bit(8*sequenceParameterSetLength) sequenceParameterSetNALUnit;
}
unsigned int(8) numOfPictureParameterSets;          //pps 个数，一般为 1
for ( i=0; i<numOfPictureParameterSets; i++) {
   unsigned int(16) pictureParameterSetLength;     //pps 长度
   bit(8*pictureParameterSetLength) pictureParameterSetNALUnit;
}
```

lengthSizeMinusOne 加 1 就是 NALU 长度字段所占字节数。

script tag 一般只有一个，是 FLV 文件的第一个 tag，用于存放 FLV 文件信息，比如时长、分辨率、音频采样率等。所有的数据都是以数据类型+(数据长度)+数据的格式出现，数据类型占 1byte，数据长度看数据类型是否存在，后面才是数据。格式如下：

objects	SCRIPTDATAOBJECT[]	不定长
End	3bytes	always 0x000009，作为 SCRIPTDATAOBJECT[]的结尾

SCRIPTDATAOBJECT[]数据类型如下：

TYPE	1byte	0--Number type 1--Boolean type 2--String type 3--Object type 4--MovieClip type 5--Null type 6--Undefined type 7--Reference type 8--ECMA array type 10--Strict array type 11--Date type 12--Long string type

```
if Type == 0
   Double
else if Type == 1
   UI8
else if Type == 2
   SCRIPTDATASTRING
else if Type == 3
```

	SCRIPTDATAOBJECT[n]
	else if Type == 4
ScriptDataValue	SCRIPTDATASTRING defining the MovieClip path
	else if Type == 7
	UI16
	else if Type == 8
	SCRIPTDATAVARIABLE[ECMAArrayLength]
	else if Type == 10
	SCRIPTDATAVARIABLE[n]
	else if Type == 11
	SCRIPTDATADATE
	else if Type == 12
	SCRIPTDATALONGSTRING
	If Type == 3
ScriptDataValueTerminator	SCRIPTDATAOBJECTEND
	else if Type == 8
	SCRIPTDATAVARIABLEEND

其中 SCRIPTDATAOBJECTEND 和 SCRIPTDATAVARIABLEEND 为 0x000009，用于标记结尾
SCRIPTDATASTRING 结构为：StringLength--2bytes StringData--STRING
SCRIPTDATALONGSTRING 结构为：StringLength--4bytes StringData--STRING
ECMA array type 结构为：ECMAArrayLength--4bytes StringLength--2bytes StringData--STRING DataType--1byte Data--不定长 SCRIPTDATAVARIABLEEND
Object type 结构为：StringLength--2bytes StringData--STRING DataType--1byte DataVale--不定长 SCRIPTDATAOBJECTEND
Strict array type 结构为：ArrayNum--4bytes DataType-byte DataValue--不定长