

# FFMPEG结构体分析：AVFormatContext

2013年11月08日 00:16:47 阅读数：59414

注：写了一系列的结构体的分析的文章，在这里列一个列表：

[FFMPEG结构体分析：AVFrame](#)

[FFMPEG结构体分析：AVFormatContext](#)

[FFMPEG结构体分析：AVCodecContext](#)

[FFMPEG结构体分析：AVIOContext](#)

[FFMPEG结构体分析：AVCodec](#)

[FFMPEG结构体分析：AVStream](#)

[FFMPEG结构体分析：AVPacket](#)

FFMPEG有几个最重要的结构体，包含了解协议，解封装，解码操作，此前已经进行过分析：

[FFMPEG中最关键的结构体之间的关系](#)

在此不再详述，其中AVFormatContext是包含码流参数较多的结构体。本文将会详细分析一下该结构体里每个变量的含义和作用。

首先看一下结构体的定义（位于avformat.h）：

```
[cpp]
1.  /*  雷霄骅
2.   *  中国传媒大学/数字电视技术
3.   *  leixiaohua1020@126.com
4.   *
5.   */
6.  /**
7.   * Format I/O context.
8.   * New fields can be added to the end with minor version bumps.
9.   * Removal, reordering and changes to existing fields require a major
10.  * version bump.
11.  * sizeof(AVFormatContext) must not be used outside libav*, use
12.  * avformat_alloc_context() to create an AVFormatContext.
13.  */
14.  typedef struct AVFormatContext {
15.      /**
16.       * A class for logging and AVOptions. Set by avformat_alloc_context().
17.       * Exports (de)muxer private options if they exist.
18.       */
19.      const AVClass *av_class;
20.
21.      /**
22.       * Can only be iformat or oformat, not both at the same time.
23.       *
24.       * decoding: set by avformat_open_input().
25.       * encoding: set by the user.
26.       */
27.      struct AVInputFormat *iformat;
28.      struct AVOutputFormat *oformat;
29.
30.      /**
31.       * Format private data. This is an AVOptions-enabled struct
32.       * if and only if iformat/oformat.priv_class is not NULL.
33.       */
34.      void *priv_data;
35.
36.      /*
37.       * I/O context.
38.       *
39.       * decoding: either set by the user before avformat_open_input() (then
40.       * the user must close it manually) or set by avformat_open_input().
41.       * encoding: set by the user.
42.       *
43.       * Do NOT set this field if AVFMT_NOFILE flag is set in
44.       * iformat/oformat.flags. In such a case, the (de)muxer will handle
45.       * I/O in some other way and this field will be NULL.
46.       */
47.      AVIOContext *pb;
48.
49.      /* stream info */
50.      int ctx_flags; /**< Format-specific flags, see AVFMTCTX_xx */
51.
52.      /**
53.       * A list of all streams in the file. New streams are created with
54.       * avformat_new_stream().
55.       *
56.       * decoding: streams are created by libavformat in avformat_open_input().
57.       * If AVFMTCTX_NOHEADER is set in ctx_flags, then new streams may also
```

```

57.  * If AV_FMT_FLAG_NONBLOCK is set in OPA_flags, then new streams may also
58.  * appear in av_read_frame().
59.  * encoding: streams are created by the user before avformat_write_header().
60.  */
61.  unsigned int nb_streams;
62.  AVStream **streams;
63.
64.  char filename[1024]; /**< input or output filename */
65.
66.  /**
67.   * Decoding: position of the first frame of the component, in
68.   * AV_TIME_BASE fractional seconds. NEVER set this value directly:
69.   * It is deduced from the AVStream values.
70.   */
71.  int64_t start_time;
72.
73.  /**
74.   * Decoding: duration of the stream, in AV_TIME_BASE fractional
75.   * seconds. Only set this value if you know none of the individual stream
76.   * durations and also do not set any of them. This is deduced from the
77.   * AVStream values if not set.
78.   */
79.  int64_t duration;
80.
81.  /**
82.   * Decoding: total stream bitrate in bit/s, 0 if not
83.   * available. Never set it directly if the file_size and the
84.   * duration are known as FFmpeg can compute it automatically.
85.   */
86.  int bit_rate;
87.
88.  unsigned int packet_size;
89.  int max_delay;
90.
91.  int flags;
92.  #define AVFMT_FLAG_GENPTS      0x0001 /**< Generate missing pts even if it requires parsing future frames.
93.  #define AVFMT_FLAG_IGNIDX      0x0002 /**< Ignore index.
94.  #define AVFMT_FLAG_NONBLOCK    0x0004 /**< Do not block when reading packets from input.
95.  #define AVFMT_FLAG_IGNDTS      0x0008 /**< Ignore DTS on frames that contain both DTS & PTS
96.  #define AVFMT_FLAG_NOFILLIN    0x0010 /**< Do not infer any values from other values, just return what is stored in the container
97.  #define AVFMT_FLAG_NOPARSE     0x0020 /**< Do not use AVParsers, you also must set AVFMT_FLAG_NOFILLIN as the fillin code works on f
98.  #define AVFMT_FLAG_CUSTOM_IO   0x0080 /**< The caller has supplied a custom AVIOContext, don't avio_close() it.
99.  #define AVFMT_FLAG_DISCARD_CORRUPT 0x0100 /**< Discard frames marked corrupted
100. #define AVFMT_FLAG_MP4A_LATM    0x8000 /**< Enable RTP MP4A-LATM payload
101. #define AVFMT_FLAG_SORT_DTS     0x10000 /**< try to interleave outputted packets by dts (using this flag can slow demuxing down)
102. #define AVFMT_FLAG_PRIV_OPT     0x20000 /**< Enable use of private options by delaying codec open (this could be made default once all
103. #define AVFMT_FLAG_KEEP_SIDE_DATA 0x40000 /**< Dont merge side data but keep it separate.
104.
105.  /**
106.   * decoding: size of data to probe; encoding: unused.
107.   */
108.  unsigned int probesize;
109.
110.  /**
111.   * decoding: maximum time (in AV_TIME_BASE units) during which the input should
112.   * be analyzed in avformat_find_stream_info().
113.   */
114.  int max_analyze_duration;
115.
116.  const uint8_t *key;
117.  int keylen;
118.
119.  unsigned int nb_programs;
120.  AVProgram **programs;
121.
122.  /**
123.   * Forced video codec_id.
124.   * Demuxing: Set by user.
125.   */
126.  enum CodecID video_codec_id;
127.
128.  /**
129.   * Forced audio codec_id.
130.   * Demuxing: Set by user.
131.   */
132.  enum CodecID audio_codec_id;
133.
134.  /**
135.   * Forced subtitle codec_id.
136.   * Demuxing: Set by user.
137.   */
138.  enum CodecID subtitle_codec_id;
139.
140.  /**
141.   * Maximum amount of memory in bytes to use for the index of each stream.
142.   * If the index exceeds this size, entries will be discarded as
143.   * needed to maintain a smaller size. This can lead to slower or less
144.   * accurate seeking (depends on demuxer).
145.   * Demuxers for which a full in-memory index is mandatory will ignore
146.   * this.

```

```

147.     * muxing : unused
148.     * demuxing: set by user
149.     */
150.     unsigned int max_index_size;
151.
152.     /**
153.      * Maximum amount of memory in bytes to use for buffering frames
154.      * obtained from realtime capture devices.
155.      */
156.     unsigned int max_picture_buffer;
157.
158.     unsigned int nb_chapters;
159.     AVChapter **chapters;
160.
161.     AVDictionary *metadata;
162.
163.     /**
164.      * Start time of the stream in real world time, in microseconds
165.      * since the unix epoch (00:00 1st January 1970). That is, pts=0
166.      * in the stream was captured at this real world time.
167.      * - encoding: Set by user.
168.      * - decoding: Unused.
169.      */
170.     int64_t start_time_realtime;
171.
172.     /**
173.      * decoding: number of frames used to probe fps
174.      */
175.     int fps_probe_size;
176.
177.     /**
178.      * Error recognition; higher values will detect more errors but may
179.      * misdetect some more or less valid parts as errors.
180.      * - encoding: unused
181.      * - decoding: Set by user.
182.      */
183.     int error_recognition;
184.
185.     /**
186.      * Custom interrupt callbacks for the I/O layer.
187.      *
188.      * decoding: set by the user before avformat_open_input().
189.      * encoding: set by the user before avformat_write_header()
190.      * (mainly useful for AVFMT_NOFILE formats). The callback
191.      * should also be passed to avio_open2() if it's used to
192.      * open the file.
193.      */
194.     AVIOInterruptCB interrupt_callback;
195.
196.     /**
197.      * Flags to enable debugging.
198.      */
199.     int debug;
200.     #define FF_FDEBUG_TS          0x0001
201.
202.     /**
203.      * Transport stream id.
204.      * This will be moved into demuxer private options. Thus no API/ABI compatibility
205.      */
206.     int ts_id;
207.
208.     /**
209.      * Audio preload in microseconds.
210.      * Note, not all formats support this and unpredictable things may happen if it is used when not supported.
211.      * - encoding: Set by user via AVOptions (NO direct access)
212.      * - decoding: unused
213.      */
214.     int audio_preload;
215.
216.     /**
217.      * Max chunk time in microseconds.
218.      * Note, not all formats support this and unpredictable things may happen if it is used when not supported.
219.      * - encoding: Set by user via AVOptions (NO direct access)
220.      * - decoding: unused
221.      */
222.     int max_chunk_duration;
223.
224.     /**
225.      * Max chunk size in bytes
226.      * Note, not all formats support this and unpredictable things may happen if it is used when not supported.
227.      * - encoding: Set by user via AVOptions (NO direct access)
228.      * - decoding: unused
229.      */
230.     int max_chunk_size;
231.
232.     /**
233.      * All fields below this line are not part of the public API. They
234.      * may not be used outside of libavformat and can be changed and
235.      * removed at will.
236.      * New public fields should be added right above.
237.      */

```

```

238.     */
239.
240.     /**
241.      * This buffer is only needed when packets were already buffered but
242.      * not decoded, for example to get the codec parameters in MPEG
243.      * streams.
244.      */
245.     struct AVPacketList *packet_buffer;
246.     struct AVPacketList *packet_buffer_end;
247.
248.     /* av_seek_frame() support */
249.     int64_t data_offset; /**< offset of the first packet */
250.
251.     /**
252.      * Raw packets from the demuxer, prior to parsing and decoding.
253.      * This buffer is used for buffering packets until the codec can
254.      * be identified, as parsing cannot be done without knowing the
255.      * codec.
256.      */
257.     struct AVPacketList *raw_packet_buffer;
258.     struct AVPacketList *raw_packet_buffer_end;
259.     /**
260.      * Packets split by the parser get queued here.
261.      */
262.     struct AVPacketList *parse_queue;
263.     struct AVPacketList *parse_queue_end;
264.     /**
265.      * Remaining size available for raw_packet_buffer, in bytes.
266.      */
267. #define RAW_PACKET_BUFFER_SIZE 2500000
268.     int raw_packet_buffer_remaining_size;
269. } AVFormatContext;

```

在使用FFMPEG进行开发的时候，AVFormatContext是一个贯穿始终的数据结构，很多函数都要用到它作为参数。它是FFMPEG解封装（flv，mp4，rmvb，avi）功能的结构体。下面看几个主要变量的作用（在这里考虑解码的情况）：

struct AVInputFormat \*iformat：输入数据的封装格式

AVIOContext \*pb：输入数据的缓存

unsigned int nb\_streams：视音频流的个数

AVStream \*\*streams：视音频流

char filename[1024]：文件名

int64\_t duration：时长（单位：微秒us，转换为秒需要除以1000000）

int bit\_rate：比特率（单位bps，转换为kbps需要除以1000）

AVDictionary \*metadata：元数据

视频的时长可以转换成HH:MM:SS的形式，示例代码如下：

```

1.  AVFormatContext *pFormatCtx;
2.  CString timelong;
3.  ...
4.  //duration是以微秒为单位
5.  //转换成hh:mm:ss形式
6.  int tns, thh, tmm, tss;
7.  tns = (pFormatCtx->duration)/1000000;
8.  thh = tns / 3600;
9.  tmm = (tns % 3600) / 60;
10. tss = (tns % 60);
11. timelong.Format("%02d:%02d:%02d", thh, tmm, tss);

```

视频的原数据（metadata）信息可以通过AVDictionary获取。元数据存储在AVDictionaryEntry结构体中，如下所示

```

1.  typedef struct AVDictionaryEntry {
2.      char *key;
3.      char *value;
4.  } AVDictionaryEntry;

```

每一条元数据分为key和value两个属性。

在ffmpeg中通过av\_dict\_get()函数获得视频的原数据。

下列代码显示了获取元数据并存入meta字符串变量的过程，注意每一条key和value之间有一个"\t:"，value之后有一个"\r\n"

```
[cpp]    
1. //MetaData-----  
2. //从AVDictionary获得  
3. //需要用到AVDictionaryEntry对象  
4. //CString author,copyright,description;  
5. CString meta=NULL,key,value;  
6. AVDictionaryEntry *m = NULL;  
7. //不用一个一个找出来  
8. /* m=av_dict_get(pFormatCtx->metadata,"author",m,0);  
9. author.Format("作者：%s",m->value);  
10. m=av_dict_get(pFormatCtx->metadata,"copyright",m,0);  
11. copyright.Format("版权：%s",m->value);  
12. m=av_dict_get(pFormatCtx->metadata,"description",m,0);  
13. description.Format("描述：%s",m->value);  
14. */  
15. //使用循环读出  
16. //(需要读取的数据，字段名称，前一条字段（循环时使用），参数)  
17. while(m=av_dict_get(pFormatCtx->metadata,"",m,AV_DICT_IGNORE_SUFFIX)){  
18.     key.Format(m->key);  
19.     value.Format(m->value);  
20.     meta+=key+"\t:"+value+"\r\n" ;  
21. }
```

文章标签：[ffmpeg](#) [AVFormatContext](#) [源代码](#) [视频](#) [元数据](#)

个人分类：[FFMPEG](#)

所属专栏：[FFmpeg](#)

此PDF由spygg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com