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
/* 雷霄骅
 * 中国传媒大学/数字电视技术
 * leixiaohua1020@126.com
 *
 */
/**
 * Format I/O context.
 * New fields can be added to the end with minor version bumps.
 * Removal, reordering and changes to existing fields require a major
 * version bump.
 * sizeof(AVFormatContext) must not be used outside libav*, use
 * avformat_alloc_context() to create an AVFormatContext.
 */
typedef struct AVFormatContext {
    /**
     * A class for logging and AVOptions. Set by avformat_alloc_context().
     * Exports (de)muxer private options if they exist.
     */
    const AVClass *av_class;

    /**
     * Can only be iformat or oformat, not both at the same time.
     *
     * decoding: set by avformat_open_input().
     * encoding: set by the user.
     */
    struct AVInputFormat *iformat;
    struct AVOutputFormat *oformat;

    /**
     * Format private data. This is an AVOptions-enabled struct
     * if and only if iformat/oformat.priv_class is not NULL.
     */
    void *priv_data;

    /*
     * I/O context.
     *
     * decoding: either set by the user before avformat_open_input() (then
     * the user must close it manually) or set by avformat_open_input().
     * encoding: set by the user.
     *
     * Do NOT set this field if AVFMT_NOFILE flag is set in
     * iformat/oformat.flags. In such a case, the (de)muxer will handle
     * I/O in some other way and this field will be NULL.
     */
    AVIOContext *pb;

    /* stream info */
    int ctx_flags; /**< Format-specific flags, see AVFMTCTX_xx */

    /**
     * A list of all streams in the file. New streams are created with
     * avformat_new_stream().
     *
     * decoding: streams are created by libavformat in avformat_open_input().
     * If AVFMTCTX_NOHEADER is set in ctx_flags, then new streams may also
```

```
     * If AVFMTCTX_NOHEADER is set in ctx_flags, then new streams may also
     * appear in av_read_frame().
     * encoding: streams are created by the user before avformat_write_header().
     */
    unsigned int nb_streams;
    AVStream **streams;

    char filename[1024]; /**< input or output filename */

    /**
     * Decoding: position of the first frame of the component, in
     * AV_TIME_BASE fractional seconds. NEVER set this value directly:
     * It is deduced from the AVStream values.
     */
    int64_t start_time;

    /**
     * Decoding: duration of the stream, in AV_TIME_BASE fractional
     * seconds. Only set this value if you know none of the individual stream
     * durations and also do not set any of them. This is deduced from the
     * AVStream values if not set.
     */
    int64_t duration;

    /**
     * Decoding: total stream bitrate in bit/s, 0 if not
     * available. Never set it directly if the file_size and the
     * duration are known as FFmpeg can compute it automatically.
     */
    int bit_rate;

    unsigned int packet_size;
    int max_delay;

    int flags;
#define AVFMT_FLAG_GENPTS       0x0001 ///< Generate missing pts even if it requires parsing future frames.
#define AVFMT_FLAG_IGNIDX       0x0002 ///< Ignore index.
#define AVFMT_FLAG_NONBLOCK     0x0004 ///< Do not block when reading packets from input.
#define AVFMT_FLAG_IGNDTS       0x0008 ///< Ignore DTS on frames that contain both DTS & PTS
#define AVFMT_FLAG_NOFILLIN     0x0010 ///< Do not infer any values from other values, just return what is stored in the container
#define AVFMT_FLAG_NOPARSE      0x0020 ///< Do not use AVParsers, you also must set AVFMT_FLAG_NOFILLIN as the fillin code works on f
es and no parsing -> no frames. Also seeking to frames can not work if parsing to find frame boundaries has been disabled
#define AVFMT_FLAG_CUSTOM_IO    0x0080 ///< The caller has supplied a custom AVIOContext, don't avio_close() it.
#define AVFMT_FLAG_DISCARD_CORRUPT  0x0100 ///< Discard frames marked corrupted
#define AVFMT_FLAG_MP4A_LATM    0x8000 ///< Enable RTP MP4A-LATM payload
#define AVFMT_FLAG_SORT_DTS    0x10000 ///< try to interleave outputted packets by dts (using this flag can slow demuxing down)
#define AVFMT_FLAG_PRIV_OPT    0x20000 ///< Enable use of private options by delaying codec open (this could be made default once all
de is converted)
#define AVFMT_FLAG_KEEP_SIDE_DATA 0x40000 ///< Dont merge side data but keep it separate.

    /**
     * decoding: size of data to probe; encoding: unused.
     */
    unsigned int probesize;

    /**
     * decoding: maximum time (in AV_TIME_BASE units) during which the input should
     * be analyzed in avformat_find_stream_info().
     */
    int max_analyze_duration;

    const uint8_t *key;
    int keylen;

    unsigned int nb_programs;
    AVProgram **programs;

    /**
     * Forced video codec_id.
     * Demuxing: Set by user.
     */
    enum CodecID video_codec_id;

    /**
     * Forced audio codec_id.
     * Demuxing: Set by user.
     */
    enum CodecID audio_codec_id;

    /**
     * Forced subtitle codec_id.
     * Demuxing: Set by user.
     */
    enum CodecID subtitle_codec_id;

    /**
     * Maximum amount of memory in bytes to use for the index of each stream.
     * If the index exceeds this size, entries will be discarded as
     * needed to maintain a smaller size. This can lead to slower or less
     * accurate seeking (depends on demuxer).
     * Demuxers for which a full in-memory index is mandatory will ignore
     * this.
```

```
     * muxing  : unused
     * demuxing: set by user
     */
    unsigned int max_index_size;

    /**
     * Maximum amount of memory in bytes to use for buffering frames
     * obtained from realtime capture devices.
     */
    unsigned int max_picture_buffer;

    unsigned int nb_chapters;
    AVChapter **chapters;

    AVDictionary *metadata;

    /**
     * Start time of the stream in real world time, in microseconds
     * since the unix epoch (00:00 1st January 1970). That is, pts=0
     * in the stream was captured at this real world time.
     * - encoding: Set by user.
     * - decoding: Unused.
     */
    int64_t start_time_realtime;

    /**
     * decoding: number of frames used to probe fps
     */
    int fps_probe_size;

    /**
     * Error recognition; higher values will detect more errors but may
     * misdetect some more or less valid parts as errors.
     * - encoding: unused
     * - decoding: Set by user.
     */
    int error_recognition;

    /**
     * Custom interrupt callbacks for the I/O layer.
     *
     * decoding: set by the user before avformat_open_input().
     * encoding: set by the user before avformat_write_header()
     * (mainly useful for AVFMT_NOFILE formats). The callback
     * should also be passed to avio_open2() if it's used to
     * open the file.
     */
    AVIOInterruptCB interrupt_callback;

    /**
     * Flags to enable debugging.
     */
    int debug;
#define FF_FDEBUG_TS        0x0001

    /**
     * Transport stream id.
     * This will be moved into demuxer private options. Thus no API/ABI compatibility
     */
    int ts_id;

    /**
     * Audio preload in microseconds.
     * Note, not all formats support this and unpredictable things may happen if it is used when not supported.
     * - encoding: Set by user via AVOptions (NO direct access)
     * - decoding: unused
     */
    int audio_preload;

    /**
     * Max chunk time in microseconds.
     * Note, not all formats support this and unpredictable things may happen if it is used when not supported.
     * - encoding: Set by user via AVOptions (NO direct access)
     * - decoding: unused
     */
    int max_chunk_duration;

    /**
     * Max chunk size in bytes
     * Note, not all formats support this and unpredictable things may happen if it is used when not supported.
     * - encoding: Set by user via AVOptions (NO direct access)
     * - decoding: unused
     */
    int max_chunk_size;

    /*****************************************************************
     * All fields below this line are not part of the public API. They
     * may not be used outside of libavformat and can be changed and
     * removed at will.
     * New public fields should be added right above.
     *****************************************************************
```

```
     */

    /**
     * This buffer is only needed when packets were already buffered but
     * not decoded, for example to get the codec parameters in MPEG
     * streams.
     */
    struct AVPacketList *packet_buffer;
    struct AVPacketList *packet_buffer_end;

    /* av_seek_frame() support */
    int64_t data_offset; /**< offset of the first packet */

    /**
     * Raw packets from the demuxer, prior to parsing and decoding.
     * This buffer is used for buffering packets until the codec can
     * be identified, as parsing cannot be done without knowing the
     * codec.
     */
    struct AVPacketList *raw_packet_buffer;
    struct AVPacketList *raw_packet_buffer_end;
    /**
     * Packets split by the parser get queued here.
     */
    struct AVPacketList *parse_queue;
    struct AVPacketList *parse_queue_end;
    /**
     * Remaining size available for raw_packet_buffer, in bytes.
     */
#define RAW_PACKET_BUFFER_SIZE 2500000
    int raw_packet_buffer_remaining_size;
} 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：

```cpp
AVFormatContext *pFormatCtx;
CString timelong;
...
//duration是以微秒为单位
//转换成hh:mm:ss形式
int tns, thh, tmm, tss;
tns  = (pFormatCtx->duration)/1000000;
thh  = tns / 3600;
tmm  = (tns % 3600) / 60;
tss  = (tns % 60);
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

```cpp
typedef struct AVDictionaryEntry {
    char *key;
    char *value;
} 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
//MetaData---------------------------------------------------------
//从AVDictionary获得
//需要用到AVDictionaryEntry对象
//CString author,copyright,description;
CString meta=NULL,key,value;
AVDictionaryEntry *m = NULL;
//不用一个一个找出来
/*  m=av_dict_get(pFormatCtx->metadata,"author",m,0);
author.Format("作者：%s",m->value);
m=av_dict_get(pFormatCtx->metadata,"copyright",m,0);
copyright.Format("版权：%s",m->value);
m=av_dict_get(pFormatCtx->metadata,"description",m,0);
description.Format("描述：%s",m->value);
*/
//使用循环读出
//(需要读取的数据，字段名称，前一条字段（循环时使用），参数)
while(m=av_dict_get(pFormatCtx->metadata,"",m,AV_DICT_IGNORE_SUFFIX)){
    key.Format(m->key);
    value.Format(m->value);
    meta+=key+"\t:"+value+"\r\n" ;
}
```

文章标签： ffmpeg AVFormatContext 源代码 视频 元数据

个人分类： FFMPEG

所属专栏： FFmpeg