

## 原 FFMPEG结构体分析：AVCodec

2013年11月09日 00:17:28 阅读数：34680

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AVCodec是存储编解码器信息的结构体。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avcodec.h文件中）：

```
[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.   * AVCodec.
8.   */
9.  typedef struct AVCodec {
10.     /**
11.      * Name of the codec implementation.
12.      * The name is globally unique among encoders and among decoders (but an
13.      * encoder and a decoder can share the same name).
14.      * This is the primary way to find a codec from the user perspective.
15.      */
16.     const char *name;
17.     /**
18.      * Descriptive name for the codec, meant to be more human readable than name.
19.      * You should use the NULL_IF_CONFIG_SMALL() macro to define it.
20.      */
21.     const char *long_name;
22.     enum AVMediaType type;
23.     enum CodecID id;
24.     /**
25.      * Codec capabilities.
26.      * see CODEC_CAP_*
27.      */
28.     int capabilities;
29.     const AVRational *supported_framerates; ///< array of supported framerates, or NULL if any, array is terminated by {0,0}
30.     const enum PixelFormat *pix_fmts;      ///< array of supported pixel formats, or NULL if unknown, array is terminated by -1
31.     const int *supported_samplerates;      ///< array of supported audio samplerates, or NULL if unknown, array is terminated by 0
32.
33.     const enum AVSampleFormat *sample_fmts; ///< array of supported sample formats, or NULL if unknown, array is terminated by -1
34.     const uint64_t *channel_layouts;        ///< array of support channel layouts, or NULL if unknown. array is terminated by 0
35.     uint8_t max_lowres;                    ///< maximum value for lowres supported by the decoder
36.     const AVClass *priv_class;             ///< AVClass for the private context
37.     const AVProfile *profiles;             ///< array of recognized profiles, or NULL if unknown, array is terminated by {FF_PROFILE_UNKNOWN}
38.
39.     /*****
40.      * No fields below this line are part of the public API. They
41.      * may not be used outside of libavcodec and can be changed and
42.      * removed at will.
43.      * New public fields should be added right above.
44.      *****/
45.     int priv_data_size;
46.     struct AVCodec *next;
47.     /**
48.      * @name Frame-level threading support functions
49.      * @{
50.      */
51.     /**
52.      * If defined, called on thread contexts when they are created.
53.      * If the codec allocates writable tables in init(), re-allocate them here.
54.      * priv_data will be set to a copy of the original.
```

```

55.     /*
56.     int (*init_thread_copy)(AVCodecContext *);
57.     /**
58.     * Copy necessary context variables from a previous thread context to the current one.
59.     * If not defined, the next thread will start automatically; otherwise, the codec
60.     * must call ff_thread_finish_setup().
61.     *
62.     * dst and src will (rarely) point to the same context, in which case memcpy should be skipped.
63.     */
64.     int (*update_thread_context)(AVCodecContext *dst, const AVCodecContext *src);
65.     /** @} */
66.
67.     /**
68.     * Private codec-specific defaults.
69.     */
70.     const AVCodecDefault *defaults;
71.
72.     /**
73.     * Initialize codec static data, called from avcodec_register().
74.     */
75.     void (*init_static_data)(struct AVCodec *codec);
76.
77.     int (*init)(AVCodecContext *);
78.     int (*encode)(AVCodecContext *, uint8_t *buf, int buf_size, void *data);
79.     /**
80.     * Encode data to an AVPacket.
81.     *
82.     * @param avctx codec context
83.     * @param avpkt output AVPacket (may contain a user-provided buffer)
84.     * @param[in] frame AVFrame containing the raw data to be encoded
85.     * @param[out] got_packet_ptr encoder sets to 0 or 1 to indicate that a
86.     * non-empty packet was returned in avpkt.
87.     * @return 0 on success, negative error code on failure
88.     */
89.     int (*encode2)(AVCodecContext *avctx, AVPacket *avpkt, const AVFrame *frame,
90.                    int *got_packet_ptr);
91.     int (*decode)(AVCodecContext *, void *outdata, int *outdata_size, AVPacket *avpkt);
92.     int (*close)(AVCodecContext *);
93.     /**
94.     * Flush buffers.
95.     * Will be called when seeking
96.     */
97.     void (*flush)(AVCodecContext *);
98. } AVCodec;

```

下面说一下最主要的几个变量：

const char \*name：编解码器的名字，比较短

const char \*long\_name：编解码器的名字，全称，比较长

enum AVMediaType type：指明了类型，是视频，音频，还是字幕

enum AVCodecID id：ID，不重复

const AVRational \*supported\_framerates：支持的帧率（仅视频）

const enum AVPixelFormat \*pix\_fmts：支持的像素格式（仅视频）

const int \*supported\_samplerates：支持的采样率（仅音频）

const enum AVSampleFormat \*sample\_fmts：支持的采样格式（仅音频）

const uint64\_t \*channel\_layouts：支持的声道数（仅音频）

int priv\_data\_size：私有数据的大小

详细介绍几个变量：

1.enum AVMediaType type

AVMediaType定义如下：

```

1. enum AVMediaType {
2.     AVMEDIA_TYPE_UNKNOWN = -1, ///< Usually treated as AVMEDIA_TYPE_DATA
3.     AVMEDIA_TYPE_VIDEO,
4.     AVMEDIA_TYPE_AUDIO,
5.     AVMEDIA_TYPE_DATA,          ///< Opaque data information usually continuous
6.     AVMEDIA_TYPE_SUBTITLE,
7.     AVMEDIA_TYPE_ATTACHMENT,    ///< Opaque data information usually sparse
8.     AVMEDIA_TYPE_NB
9. };

```

2.enum AVCodecID id

AVCodecID定义如下：

```

1. enum AVCodecID {
2.     AV_CODEC_ID_NONE,
3.
4.     /* video codecs */
5.     AV_CODEC_ID_MPEG1VIDEO,
6.     AV_CODEC_ID_MPEG2VIDEO, ///< preferred ID for MPEG-1/2 video decoding
7.     AV_CODEC_ID_MPEG2VIDEO_XVMC,
8.     AV_CODEC_ID_H261,
9.     AV_CODEC_ID_H263,
10.    AV_CODEC_ID_RV10,
11.    AV_CODEC_ID_RV20,
12.    AV_CODEC_ID_MJPEG,
13.    AV_CODEC_ID_MJPEGB,
14.    AV_CODEC_ID_LJPEG,
15.    AV_CODEC_ID_SPSX,
16.    AV_CODEC_ID_JPEGLS,
17.    AV_CODEC_ID_MPEG4,
18.    AV_CODEC_ID_RAWVIDEO,
19.    AV_CODEC_ID_MSMPEG4V1,
20.    AV_CODEC_ID_MSMPEG4V2,
21.    AV_CODEC_ID_MSMPEG4V3,
22.    AV_CODEC_ID_WMV1,
23.    AV_CODEC_ID_WMV2,
24.    AV_CODEC_ID_H263P,
25.    AV_CODEC_ID_H263I,
26.    AV_CODEC_ID_FLV1,
27.    AV_CODEC_ID_SVQ1,
28.    AV_CODEC_ID_SVQ3,
29.    AV_CODEC_ID_DVVIDEO,
30.    AV_CODEC_ID_HUFFYUV,
31.    AV_CODEC_ID_CYUV,
32.    AV_CODEC_ID_H264,
33.    ... (代码太长, 略)
34. }

```

3.const enum AVPixelFormat \*pix\_fmts

AVPixelFormat定义如下：

```

1. enum AVPixelFormat {
2.     AV_PIX_FMT_NONE = -1,
3.     AV_PIX_FMT_YUV420P, ///< planar YUV 4:2:0, 12bpp, (1 Cr & Cb sample per 2x2 Y samples)
4.     AV_PIX_FMT_YUV422,  ///< packed YUV 4:2:2, 16bpp, Y0 Cb Y1 Cr
5.     AV_PIX_FMT_RGB24,    ///< packed RGB 8:8:8, 24bpp, RGBRGB...
6.     AV_PIX_FMT_BGR24,    ///< packed RGB 8:8:8, 24bpp, BGRBGR...
7.     AV_PIX_FMT_YUV422P,  ///< planar YUV 4:2:2, 16bpp, (1 Cr & Cb sample per 2x1 Y samples)
8.     AV_PIX_FMT_YUV444P,  ///< planar YUV 4:4:4, 24bpp, (1 Cr & Cb sample per 1x1 Y samples)
9.     AV_PIX_FMT_YUV410P,  ///< planar YUV 4:1:0, 9bpp, (1 Cr & Cb sample per 4x4 Y samples)
10.    AV_PIX_FMT_YUV411P,  ///< planar YUV 4:1:1, 12bpp, (1 Cr & Cb sample per 4x1 Y samples)
11.    AV_PIX_FMT_GRAY8,     ///<      Y      , 8bpp
12.    AV_PIX_FMT_MONOWHITE, ///<      Y      , 1bpp, 0 is white, 1 is black, in each byte pixels are ordered from the msb to the lsb
13.
14.    AV_PIX_FMT_MONOBLACK, ///<      Y      , 1bpp, 0 is black, 1 is white, in each byte pixels are ordered from the msb to the lsb
15.
16.    AV_PIX_FMT_PAL8,      ///< 8 bit with PIX_FMT_RGB32 palette
17.    AV_PIX_FMT_YUVJ420P,  ///< planar YUV 4:2:0, 12bpp, full scale (JPEG), deprecated in favor of PIX_FMT_YUV420P and setting color_range
18.    AV_PIX_FMT_YUVJ422P,  ///< planar YUV 4:2:2, 16bpp, full scale (JPEG), deprecated in favor of PIX_FMT_YUV422P and setting color_range
19.    AV_PIX_FMT_YUVJ444P,  ///< planar YUV 4:4:4, 24bpp, full scale (JPEG), deprecated in favor of PIX_FMT_YUV444P and setting color_range
20.
21.    AV_PIX_FMT_XVMC_MPEG2_MC, ///< XVideo Motion Acceleration via common packet passing
22.    AV_PIX_FMT_XVMC_MPEG2_IDCT,
23.    ... (代码太长, 略)
24. }

```

4.const enum AVSampleFormat \*sample\_fmts

```
[cpp]
1. enum AVSampleFormat {
2.     AV_SAMPLE_FMT_NONE = -1,
3.     AV_SAMPLE_FMT_U8,      ///< unsigned 8 bits
4.     AV_SAMPLE_FMT_S16,     ///< signed 16 bits
5.     AV_SAMPLE_FMT_S32,     ///< signed 32 bits
6.     AV_SAMPLE_FMT_FLT,     ///< float
7.     AV_SAMPLE_FMT_DBL,     ///< double
8.
9.     AV_SAMPLE_FMT_U8P,     ///< unsigned 8 bits, planar
10.    AV_SAMPLE_FMT_S16P,     ///< signed 16 bits, planar
11.    AV_SAMPLE_FMT_S32P,     ///< signed 32 bits, planar
12.    AV_SAMPLE_FMT_FLTP,     ///< float, planar
13.    AV_SAMPLE_FMT_DBLP,     ///< double, planar
14.
15.    AV_SAMPLE_FMT_NB        ///< Number of sample formats. DO NOT USE if linking dynamically
16. };
```

每一个编解码器对应一个该结构体，查看一下ffmpeg的源代码，我们可以看一下H.264解码器的结构体如下所示（h264.c）：

```
[cpp]
1. AVCodec ff_h264_decoder = {
2.     .name      = "h264",
3.     .type      = AVMEDIA_TYPE_VIDEO,
4.     .id        = CODEC_ID_H264,
5.     .priv_data_size = sizeof(H264Context),
6.     .init       = ff_h264_decode_init,
7.     .close      = ff_h264_decode_end,
8.     .decode     = decode_frame,
9.     .capabilities = /*CODEC_CAP_DRAW_HORIZ_BAND |*/ CODEC_CAP_DR1 | CODEC_CAP_DELAY |
10.                    CODEC_CAP_SLICE_THREADS | CODEC_CAP_FRAME_THREADS,
11.     .flush      = flush_dpb,
12.     .long_name = NULL_IF_CONFIG_SMALL("H.264 / AVC / MPEG-4 AVC / MPEG-4 part 10"),
13.     .init_thread_copy = ONLY_IF_THREADS_ENABLED(decode_init_thread_copy),
14.     .update_thread_context = ONLY_IF_THREADS_ENABLED(decode_update_thread_context),
15.     .profiles = NULL_IF_CONFIG_SMALL(profiles),
16.     .priv_class = &h264_class,
17. };
```

JPEG2000解码器结构体（j2kdec.c）

```
[cpp]
1. AVCodec ff_jpeg2000_decoder = {
2.     .name      = "j2k",
3.     .type      = AVMEDIA_TYPE_VIDEO,
4.     .id        = CODEC_ID_JPEG2000,
5.     .priv_data_size = sizeof(J2kDecoderContext),
6.     .init       = j2kdec_init,
7.     .close      = decode_end,
8.     .decode     = decode_frame,
9.     .capabilities = CODEC_CAP_EXPERIMENTAL,
10.    .long_name = NULL_IF_CONFIG_SMALL("JPEG 2000"),
11.    .pix_fmts =
12.        (const enum PixelFormat[]) {PIX_FMT_GRAY8, PIX_FMT_RGB24, PIX_FMT_NONE}
13. };
```

下面简单介绍一下遍历ffmpeg中的解码器信息的方法（这些解码器以一个链表的形式存储）：

1.注册所有编解码器：av\_register\_all();

2.声明一个AVCodec类型的指针，比如说AVCodec\* first\_c;

3.调用av\_codec\_next()函数，即可获得指向链表下一个解码器的指针，循环往复可以获得所有解码器的信息。注意，如果想要获得指向第一个解码器的指针，则需要将该函数的参数设置为NULL。

版权声明：本文为博主原创文章，未经博主允许不得转载。<https://blog.csdn.net/leixiaohua1020/article/details/14215833>

文章标签：[ffmpeg](#) [avcodec](#) [视频](#) [解码器](#) [结构体](#)

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](mailto:liushidc@163.com)