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
/* 雷霄骅
 * 中国传媒大学/数字电视技术
 * leixiaohua1020@126.com
 *
 */
 /**
 * AVCodec.
 */
typedef struct AVCodec {
    /**
     * Name of the codec implementation.
     * The name is globally unique among encoders and among decoders (but an
     * encoder and a decoder can share the same name).
     * This is the primary way to find a codec from the user perspective.
     */
    const char *name;
    /**
     * Descriptive name for the codec, meant to be more human readable than name.
     * You should use the NULL_IF_CONFIG_SMALL() macro to define it.
     */
    const char *long_name;
    enum AVMediaType type;
    enum CodecID id;
    /**
     * Codec capabilities.
     * see CODEC_CAP_*
     */
    int capabilities;
    const AVRational *supported_framerates; ///< array of supported framerates, or NULL if any, array is terminated by {0,0}
    const enum PixelFormat *pix_fmts;       ///< array of supported pixel formats, or NULL if unknown, array is terminated by -1
    const int *supported_samplerates;       ///< array of supported audio samplerates, or NULL if unknown, array is terminated by 0
    const enum AVSampleFormat *sample_fmts; ///< array of supported sample formats, or NULL if unknown, array is terminated by -1
    const uint64_t *channel_layouts;         ///< array of support channel layouts, or NULL if unknown. array is terminated by 0
    uint8_t max_lowres;                     ///< maximum value for lowres supported by the decoder
    const AVClass *priv_class;              ///< AVClass for the private context
    const AVProfile *profiles;              ///< array of recognized profiles, or NULL if unknown, array is terminated by {FF_PROFILE_UNKNOWN}

    /*****************************************************************
     * No fields below this line are part of the public API. They
     * may not be used outside of libavcodec and can be changed and
     * removed at will.
     * New public fields should be added right above.
     *****************************************************************
     */
    int priv_data_size;
    struct AVCodec *next;
    /**
     * @name Frame-level threading support functions
     * @{
     */
    /**
     * If defined, called on thread contexts when they are created.
     * If the codec allocates writable tables in init(), re-allocate them here.
     * priv_data will be set to a copy of the original.
```

```
     */
    int (*init_thread_copy)(AVCodecContext *);
    /**
     * Copy necessary context variables from a previous thread context to the current one.
     * If not defined, the next thread will start automatically; otherwise, the codec
     * must call ff_thread_finish_setup().
     *
     * dst and src will (rarely) point to the same context, in which case memcpy should be skipped.
     */
    int (*update_thread_context)(AVCodecContext *dst, const AVCodecContext *src);
    /** @} */

    /**
     * Private codec-specific defaults.
     */
    const AVCodecDefault *defaults;

    /**
     * Initialize codec static data, called from avcodec_register().
     */
    void (*init_static_data)(struct AVCodec *codec);

    int (*init)(AVCodecContext *);
    int (*encode)(AVCodecContext *, uint8_t *buf, int buf_size, void *data);
    /**
     * Encode data to an AVPacket.
     *
     * @param      avctx          codec context
     * @param      avpkt          output AVPacket (may contain a user-provided buffer)
     * @param[in]  frame          AVFrame containing the raw data to be encoded
     * @param[out] got_packet_ptr encoder sets to 0 or 1 to indicate that a
     *                            non-empty packet was returned in avpkt.
     * @return 0 on success, negative error code on failure
     */
    int (*encode2)(AVCodecContext *avctx, AVPacket *avpkt, const AVFrame *frame,
                   int *got_packet_ptr);
    int (*decode)(AVCodecContext *, void *outdata, int *outdata_size, AVPacket *avpkt);
    int (*close)(AVCodecContext *);
    /**
     * Flush buffers.
     * Will be called when seeking
     */
    void (*flush)(AVCodecContext *);
} 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：

```cpp
enum AVMediaType {
    AVMEDIA_TYPE_UNKNOWN = -1,  ///< Usually treated as AVMEDIA_TYPE_DATA
    AVMEDIA_TYPE_VIDEO,
    AVMEDIA_TYPE_AUDIO,
    AVMEDIA_TYPE_DATA,          ///< Opaque data information usually continuous
    AVMEDIA_TYPE_SUBTITLE,
    AVMEDIA_TYPE_ATTACHMENT,    ///< Opaque data information usually sparse
    AVMEDIA_TYPE_NB
};
```

2.enum AVCodecID id

AVCodecID定义如下：

```cpp
enum AVCodecID {
    AV_CODEC_ID_NONE,

    /* video codecs */
    AV_CODEC_ID_MPEG1VIDEO,
    AV_CODEC_ID_MPEG2VIDEO, ///< preferred ID for MPEG-1/2 video decoding
    AV_CODEC_ID_MPEG2VIDEO_XVMC,
    AV_CODEC_ID_H261,
    AV_CODEC_ID_H263,
    AV_CODEC_ID_RV10,
    AV_CODEC_ID_RV20,
    AV_CODEC_ID_MJPEG,
    AV_CODEC_ID_MJPEGB,
    AV_CODEC_ID_LJPEG,
    AV_CODEC_ID_SP5X,
    AV_CODEC_ID_JPEGLS,
    AV_CODEC_ID_MPEG4,
    AV_CODEC_ID_RAWVIDEO,
    AV_CODEC_ID_MSMPEG4V1,
    AV_CODEC_ID_MSMPEG4V2,
    AV_CODEC_ID_MSMPEG4V3,
    AV_CODEC_ID_WMV1,
    AV_CODEC_ID_WMV2,
    AV_CODEC_ID_H263P,
    AV_CODEC_ID_H263I,
    AV_CODEC_ID_FLV1,
    AV_CODEC_ID_SVQ1,
    AV_CODEC_ID_SVQ3,
    AV_CODEC_ID_DVVIDEO,
    AV_CODEC_ID_HUFFYUV,
    AV_CODEC_ID_CYUV,
    AV_CODEC_ID_H264,
    ...（代码太长，略）
}
```

3.const enum AVPixelFormat *pix_fmts

AVPixelFormat定义如下：

```cpp
enum AVPixelFormat {
    AV_PIX_FMT_NONE = -1,
    AV_PIX_FMT_YUV420P,   ///< planar YUV 4:2:0, 12bpp, (1 Cr & Cb sample per 2x2 Y samples)
    AV_PIX_FMT_YUYV422,   ///< packed YUV 4:2:2, 16bpp, Y0 Cb Y1 Cr
    AV_PIX_FMT_RGB24,     ///< packed RGB 8:8:8, 24bpp, RGBRGB...
    AV_PIX_FMT_BGR24,     ///< packed RGB 8:8:8, 24bpp, BGRBGR...
    AV_PIX_FMT_YUV422P,   ///< planar YUV 4:2:2, 16bpp, (1 Cr & Cb sample per 2x1 Y samples)
    AV_PIX_FMT_YUV444P,   ///< planar YUV 4:4:4, 24bpp, (1 Cr & Cb sample per 1x1 Y samples)
    AV_PIX_FMT_YUV410P,   ///< planar YUV 4:1:0,  9bpp, (1 Cr & Cb sample per 4x4 Y samples)
    AV_PIX_FMT_YUV411P,   ///< planar YUV 4:1:1, 12bpp, (1 Cr & Cb sample per 4x1 Y samples)
    AV_PIX_FMT_GRAY8,     ///<        Y        ,  8bpp
    AV_PIX_FMT_MONOWHITE, ///<        Y        ,  1bpp, 0 is white, 1 is black, in each byte pixels are ordered from the msb to the l

    AV_PIX_FMT_MONOBLACK, ///<        Y        ,  1bpp, 0 is black, 1 is white, in each byte pixels are ordered from the msb to the l

    AV_PIX_FMT_PAL8,      ///< 8 bit with PIX_FMT_RGB32 palette
    AV_PIX_FMT_YUVJ420P,  ///< planar YUV 4:2:0, 12bpp, full scale (JPEG), deprecated in favor of PIX_FMT_YUV420P and setting color_r
ge
    AV_PIX_FMT_YUVJ422P,  ///< planar YUV 4:2:2, 16bpp, full scale (JPEG), deprecated in favor of PIX_FMT_YUV422P and setting color_r
ge
    AV_PIX_FMT_YUVJ444P,  ///< planar YUV 4:4:4, 24bpp, full scale (JPEG), deprecated in favor of PIX_FMT_YUV444P and setting color_r
ge
    AV_PIX_FMT_XVMC_MPEG2_MC,///< XVideo Motion Acceleration via common packet passing
    AV_PIX_FMT_XVMC_MPEG2_IDCT,
    ...（代码太长，略）
}
```

4.const enum AVSampleFormat *sample_fmts

```cpp
enum AVSampleFormat {
    AV_SAMPLE_FMT_NONE = -1,
    AV_SAMPLE_FMT_U8,          ///< unsigned 8 bits
    AV_SAMPLE_FMT_S16,         ///< signed 16 bits
    AV_SAMPLE_FMT_S32,         ///< signed 32 bits
    AV_SAMPLE_FMT_FLT,         ///< float
    AV_SAMPLE_FMT_DBL,         ///< double

    AV_SAMPLE_FMT_U8P,         ///< unsigned 8 bits, planar
    AV_SAMPLE_FMT_S16P,        ///< signed 16 bits, planar
    AV_SAMPLE_FMT_S32P,        ///< signed 32 bits, planar
    AV_SAMPLE_FMT_FLTP,        ///< float, planar
    AV_SAMPLE_FMT_DBLP,        ///< double, planar

    AV_SAMPLE_FMT_NB           ///< Number of sample formats. DO NOT USE if linking dynamically
};
```

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
AVCodec ff_h264_decoder = {
    .name           = "h264",
    .type           = AVMEDIA_TYPE_VIDEO,
    .id             = CODEC_ID_H264,
    .priv_data_size = sizeof(H264Context),
    .init           = ff_h264_decode_init,
    .close          = ff_h264_decode_end,
    .decode         = decode_frame,
    .capabilities   = /*CODEC_CAP_DRAW_HORIZ_BAND |*/ CODEC_CAP_DR1 | CODEC_CAP_DELAY |
                      CODEC_CAP_SLICE_THREADS | CODEC_CAP_FRAME_THREADS,
    .flush= flush_dpb,
    .long_name = NULL_IF_CONFIG_SMALL("H.264 / AVC / MPEG-4 AVC / MPEG-4 part 10"),
    .init_thread_copy      = ONLY_IF_THREADS_ENABLED(decode_init_thread_copy),
    .update_thread_context = ONLY_IF_THREADS_ENABLED(decode_update_thread_context),
    .profiles = NULL_IF_CONFIG_SMALL(profiles),
    .priv_class     = &h264_class,
};
```

JPEG2000解码器结构体（j2kdec.c）

```cpp
AVCodec ff_jpeg2000_decoder = {
    .name           = "j2k",
    .type           = AVMEDIA_TYPE_VIDEO,
    .id             = CODEC_ID_JPEG2000,
    .priv_data_size = sizeof(J2kDecoderContext),
    .init           = j2kdec_init,
    .close          = decode_end,
    .decode         = decode_frame,
    .capabilities = CODEC_CAP_EXPERIMENTAL,
    .long_name = NULL_IF_CONFIG_SMALL("JPEG 2000"),
    .pix_fmts =
        (const enum PixelFormat[]) {PIX_FMT_GRAY8, PIX_FMT_RGB24, PIX_FMT_NONE}
};
```

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。

文章标签： ffmpeg avcodec 视频 解码器 结构体

个人分类： FFMPEG

所属专栏： FFmpeg