

## ffds show 源代码分析 7：libavcodec 视频解码器类（TVideoCodecLibavcodec）

2013年11月12日 00:36:32 阅读数：6552

=====

ffds show 源代码分析系列文章列表：

[ffds show 源代码分析 1：整体结构](#)

[ffds show 源代码分析 2：位图覆盖滤镜（对话框部分Dialog）](#)

[ffds show 源代码分析 3：位图覆盖滤镜（设置部分Settings）](#)

[ffds show 源代码分析 4：位图覆盖滤镜（滤镜部分Filter）](#)

[ffds show 源代码分析 5：位图覆盖滤镜（总结）](#)

[ffds show 源代码分析 6：对解码器的dll的封装（libavcodec）](#)

[ffds show 源代码分析 7：libavcodec 视频解码器类（TVideoCodecLibavcodec）](#)

[ffds show 源代码分析 8：视频解码器类（TVideoCodecDec）](#)

[ffds show 源代码分析 9：编解码器有关类的总结](#)

=====



前文已经介绍了ffds show中对libavcodec封装的类Tlibavcodec：

[ffds show 源代码分析 6：对解码器的dll的封装（libavcodec）](#)

在这里我们进一步介绍一下其libavcodec解码器类。注意前一篇文章介绍的类Tlibavcodec仅仅是对libavcodec所在的“ffmpeg.dll”的函数进行封装的类。但Tlibavcodec并不是一个解码器类，其没有继承任何类，还不能为ffds show所用。本文介绍的TVideoCodecLibavcodec才是libavcodec解码器类，其继承了TVideoCodecDec。

先来看一看TVideoCodecLibavcodec的定义吧，位于codecs->TVideoCodecLibavcodec.h中。

```
[cpp]
1.  /*
2.   *雷霄骅
3.   *leixiaohua1020@126.com
4.   *中国传媒大学/数字电视技术
5.   */
6.  #ifndef _TVIDEOCODECLIBAVCODEC_H_
7.  #define _TVIDEOCODECLIBAVCODEC_H_
8.
9.  #include "TVideoCodec.h"
10. #include "ffmpeg/Tlibavcodec.h"
11. #include "ffmpeg/libavcodec/avcodec.h"
12.
13. #define MAX_THREADS 8 // FIXME: This is defined in mpegvideo.h.
14.
15. struct Textradata;
16. class TccDecoder;
17. //libavcodec解码器（视频）
18. struct TlibavcodecExt {
19. private:
20.     static int get_buffer(AVCodecContext *s, AVFrame *pic);
21.     int (*default_get_buffer)(AVCodecContext *s, AVFrame *pic);
22.     static void release_buffer(AVCodecContext *s, AVFrame *pic);
23.     void (*default_release_buffer)(AVCodecContext *s, AVFrame *pic);
24.     static int reget_buffer(AVCodecContext *s, AVFrame *pic);
25.     int (*default_reget_buffer)(AVCodecContext *s, AVFrame *pic);
26.     static void handle_user_data0(AVCodecContext *c, const uint8_t *buf, int buf_len);
27. public:
28.     virtual ~TlibavcodecExt() {}
29.     void connectTo(AVCodecContext *ctx, Tlibavcodec *libavcodec);
30.     virtual void onGetBuffer(AVFrame *pic) {}
31.     virtual void onRegetBuffer(AVFrame *pic) {}
32.     virtual void onReleaseBuffer(AVFrame *pic) {}
33.     virtual void handle_user_data(const uint8_t *buf, int buf_len) {}
```

```

34. };
35. //libavcodec解码, 不算是Filter?
36. class TvideoCodecLibavcodec : public TvideoCodecDec, public TvideoCodecEnc, public TlibavcodecExt
37. {
38.     friend class TDXVADecoderVC1;
39.     friend class TDXVADecoderH264;
40. protected:
41.     //各种信息 (源自AVCodecContext中)
42.     Tlibavcodec *libavcodec;
43.     void create(void);
44.     AVCodec *avcodec;
45.     mutable char_t codecName[100];
46.     AVCodecContext *avctx;
47.     uint32_t palette[AVPALETTE_COUNT];
48.     int palette_size;
49.     AVFrame *frame;
50.     FOURCC fcc;
51.     FILE *statsfile;
52.     int cfgcomode;
53.     int psnr;
54.     bool isAdaptive;
55.     int threadcount;
56.     bool dont_use_rtStop_from_upper_stream; // and reordering of timestamps is justified.
57.     bool theorart;
58.     bool codecinited, ownmatrices;
59.     REFERENCE_TIME rtStart, rtStop, avgTimePerFrame, segmentTimeStart;
60.     REFERENCE_TIME prior_in_rtStart, prior_in_rtStop;
61.     REFERENCE_TIME prior_out_rtStart, prior_out_rtStop;
62.
63.     struct {
64.         REFERENCE_TIME rtStart, rtStop;
65.         unsigned int srcSize;
66.     } b[MAX_THREADS + 1];
67.     int inPosB;
68.
69.     Textradata *extradata;
70.     bool sendextradata;
71.     unsigned int mb_width, mb_height, mb_count;
72.     static void line(unsigned char *dst, unsigned int _x0, unsigned int _y0, unsigned int _x1, unsigned int _y1, stride_t strideY);
73.     static void draw_arrow(uint8_t *buf, int sx, int sy, int ex, int ey, stride_t stride, int mulx, int muly, int dstdx, int dstdy);
74.
75.     unsigned char *ffbuf;
76.     unsigned int ffbuflen;
77.     bool wasKey;
78.     virtual void handle_user_data(const uint8_t *buf, int buf_len);
79.     TccDecoder *ccDecoder;
80.     bool autoSkippingLoopFilter;
81.     enum AVDiscard initialSkipLoopFilter;
82.     int got_picture;
83.     bool firstSeek; // firstSeek means start of palyback.
84.     bool mpeg2_in_doubt;
85.     bool mpeg2_new_sequence;
86.     bool bReorderBFrame;
87.     //时长 (AVCodecContext中)
88.     REFERENCE_TIME getDuration();
89.     int isReallyMPEG2(const unsigned char *src, size_t srcLen);
90. protected:
91.     virtual LRESULT beginCompress(int cfgcomode, uint64_t csp, const Trect &r);
92.     virtual bool beginDecompress(TffPictBase &pict, FOURCC infcc, const CMediaType &mt, int sourceFlags);
93.     virtual HRESULT flushDec(void);
94.     AVCodecParserContext *parser;
95. public:
96.     TvideoCodecLibavcodec(IffdshowBase *Ideci, IdecVideoSink *IsinkD);
97.     TvideoCodecLibavcodec(IffdshowBase *Ideci, IencVideoSink *IsinkE);
98.     virtual ~TvideoCodecLibavcodec();
99.     virtual int getType(void) const {
100.         return IDFF_MOVIE_LAVC;
101.     }
102.     virtual const char_t* getName(void) const;
103.     virtual int caps(void) const {
104.         return CAPS::VIS_MV | CAPS::VIS_QUANTS;
105.     }
106.
107.     virtual void end(void);
108.
109.     virtual void getCompressColorspaces(Tcsps &csp, unsigned int outDx, unsigned int outDy);
110.     virtual bool supExtradata(void);
111.     //获得ExtraData (AVCodecContext中)
112.     virtual bool getExtradata(const void* *ptr, size_t *len);
113.     virtual HRESULT compress(const TffPict &pict, TencFrameParams ¶ms);
114.     virtual HRESULT flushEnc(const TffPict &pict, TencFrameParams ¶ms) {
115.         return compress(pict, ¶ms);
116.     }
117.
118.     virtual HRESULT decompress(const unsigned char *src, size_t srcLen, IMediaSample *pIn);
119.     virtual void onGetBuffer(AVFrame *pic);
120.     virtual bool onSeek(REFERENCE_TIME segmentStart);
121.     virtual bool onDiscontinuity(void);
122.     //画出运动矢量 (AVCodecContext中)
123.     virtual bool drawMV(unsigned char *dst, unsigned int dx, stride_t stride, unsigned int dy) const;
124.     //编码器信息 (AVCodecContext中)

```

```

124.     virtual void getEncoderInfo(char_t *buf, size_t buflen) const;
125.     virtual const char* get_current_idct(void);
126.     virtual HRESULT BeginFlush();
127.     bool isReorderBFrame() {
128.         return bReorderBFrame;
129.     };
130.     virtual void reorderBFrames(REFERENCE_TIME& rtStart, REFERENCE_TIME& rtStop);
131.
132.     class Th264RandomAccess
133.     {
134.     friend class TvideoCodecLibavcodec;
135.     private:
136.         TvideoCodecLibavcodec* parent;
137.         int recovery_mode; // 0:OK, 1:searching 2: found, 3:waiting for frame_num decoded, 4:waiting for POC outputed
138.         int recovery_frame_cnt;
139.         int recovery_poc;
140.         int thread_delay;
141.
142.     public:
143.         Th264RandomAccess(TvideoCodecLibavcodec* Iparent);
144.         int search(uint8_t* buf, int buf_size);
145.         void onSeek(void);
146.         void judgeUsability(int *got_picture_ptr);
147.     } h264RandomAccess;
148. };
149.
150. #endif

```

这里有一个类TlibavcodecExt，我觉得应该是扩展了Tlibavcodec的一些功能，在这里我们先不管它，直接看看TvideoCodecLibavcodec都包含了什么变量：

Tlibavcodec \*libavcodec：该类封装了libavcodec的各种函数，在前一篇文章中已经做过介绍，在此不再重复叙述了。可以认为该变量是TvideoCodecLibavcodec类的灵魂，所有libavcodec中的函数都是通过该类调用的。

AVCodec \*avcodec：FFMPEG中的结构体，解码器

AVCodecContext \*avctx：FFMPEG中的结构体，解码器上下文

AVFrame \*frame FFMPEG中的结构体，视频帧

mutable char\_t codecName[100]：解码器名称

FOURCC fcc：FourCC

Textadata \*extradata：附加数据

...

再来看一下TvideoCodecLibavcodec都包含什么方法：

create()：创建解码器的时候调用

getDuration()：获得时长

getExtradata()：获得附加数据

drawMV()：画运动矢量

getEncoderInfo()：获得编码器信息

此外还包括一些有关解码的方法【这个是最关键的】：

beginDecompress()：解码初始化

decompress()：解码

下面我们来详细看看这些函数的实现吧：

先来看一下TvideoCodecLibavcodec的构造函数：

```

1. //libavcodec解码器（视频）
2. //内容大部分都很熟悉，因为是FFmpeg的API
3. TvideoCodecLibavcodec::TvideoCodecLibavcodec(IffdshowBase *Ideci, IdecVideoSink *IsinkD):
4.     Tcodec(Ideci), TcodecDec(Ideci, IsinkD),
5.     TvideoCodec(Ideci),
6.     TvideoCodecDec(Ideci, IsinkD),
7.     TvideoCodecEnc(Ideci, NULL),
8.     h264RandomAccess(this),
9.     bReorderBFrame(true)
10. {
11.     create();
12. }

```

可见构造函数调用了Create()，我们再来看看Create()：

```
[cpp]
1. void TvideoCodecLibavcodec::create(void)
2. {
3.     ownmatrices = false;
4.     deci->getLibavcodec(&libavcodec);
5.     ok = libavcodec ? libavcodec->ok : false;
6.     avctx = NULL;
7.     avcodec = NULL;
8.     frame = NULL;
9.     quantBytes = 1;
10.    statsfile = NULL;
11.    threadcount = 0;
12.    codecinited = false;
13.    extradata = NULL;
14.    theorart = false;
15.    ffbuff = NULL;
16.    ffbufflen = 0;
17.    codecName[0] = '\0';
18.    ccDecoder = NULL;
19.    autoSkippingLoopFilter = false;
20.    inPosB = 1;
21.    firstSeek = true;
22.    mpeg2_new_sequence = true;
23.    parser = NULL;
24. }
```

从Create()函数我们可以看出，其完成了各种变量的初始化工作。其中有一行代码：

```
[cpp]
1. deci->getLibavcodec(&libavcodec);
```

完成了Tlibavcodec\*libavcodec的初始化工作。

再来看几个函数。

getDuration()，用于从AVCodecContext中获取时长：

```
[cpp]
1. REFERENCE_TIME TvideoCodecLibavcodec::getDuration()
2. {
3.     REFERENCE_TIME duration = REF_SECOND_MULT / 100;
4.     if (avctx && avctx->time_base.num && avctx->time_base.den) {
5.         duration = REF_SECOND_MULT * avctx->time_base.num / avctx->time_base.den;
6.         if (codecId == AV_CODEC_ID_H264) {
7.             duration *= 2;
8.         }
9.     }
10.    if (duration == 0) {
11.        return REF_SECOND_MULT / 100;
12.    }
13.    return duration;
14. }
```

getExtradata()用于从AVCodecContext中获取附加信息：

```
[cpp]
1. bool TvideoCodecLibavcodec::getExtradata(const void* *ptr, size_t *len)
2. {
3.     if (!avctx || !len) {
4.         return false;
5.     }
6.     *len = avctx->extradata_size;
7.     if (ptr) {
8.         *ptr = avctx->extradata;
9.     }
10.    return true;
11. }
```

drawMV()用于从AVFrame中获取运动矢量信息，并画出来（这个函数用于一个名为“可视化”的滤镜里面，用于显示视频的运动矢量信息）。

```

1. //画出运动矢量
2. bool TvideoCodecLibavcodec::drawMV(unsigned char *dst, unsigned int dstdx, stride_t stride, unsigned int dstdy) const
3. {
4.     if (!frame->motion_val || !frame->mb_type || !frame->motion_val[0]) {
5.         return false;
6.     }
7.
8.     #define IS_8X8(a) ((a)&MB_TYPE_8x8)
9.     #define IS_16X8(a) ((a)&MB_TYPE_16x8)
10.    #define IS_8X16(a) ((a)&MB_TYPE_8x16)
11.    #define IS_INTERLACED(a) ((a)&MB_TYPE_INTERLACED)
12.    #define USES_LIST(a, list) ((a) & ((MB_TYPE_P0L0|MB_TYPE_P1L0)<<(2*(list))))
13.
14.    const int shift = 1 + ((frame->play_flags & CODEC_FLAG_QPEL) ? 1 : 0);
15.    const int mv_sample_log2 = 4 - frame->motion_subsample_log2;
16.    const int mv_stride = (frame->mb_width << mv_sample_log2) + (avctx->codec_id == AV_CODEC_ID_H264 ? 0 : 1);
17.    int direction = 0;
18.
19.    int mulx = (dstdx << 12) / avctx->width;
20.    int muly = (dstdy << 12) / avctx->height;
21.    //提取两个方向上的运动矢量信息 (根据不同的宏块划分, 可以分成几种情况)
22.    //在AVCodecContext的motion_val中
23.    for (int mb_y = 0; mb_y < frame->mb_height; mb_y++)
24.        for (int mb_x = 0; mb_x < frame->mb_width; mb_x++) {
25.            const int mb_index = mb_x + mb_y * frame->mb_stride;
26.            if (!USES_LIST(frame->mb_type[mb_index], direction)) {
27.                continue;
28.            }
29.            ...此处代码太长, 略
30.        }
31.    #undef IS_8X8
32.    #undef IS_16X8
33.    #undef IS_8X16
34.    #undef IS_INTERLACED
35.    #undef USES_LIST
36.    return true;
37. }

```

下面来看几个很重要的函数，这几个函数继承自TvideoCodecDec类。

**beginDecompress()**用于解码器的初始化。注：这个函数的代码太长了，因此只选择一点关键的代码。

[cpp]  

```
1. //----- decompression -----
2. bool TvideoCodecLibavcodec::beginDecompress(TffPictBase &pict, FOURCC fcc, const CMediaType &mt, int sourceFlags)
3. {
4.     palette_size = 0;
5.     prior_out_rtStart = REFTIME_INVALID;
6.     prior_out_rtStop = 0;
7.     rtStart = rtStop = REFTIME_INVALID;
8.     prior_in_rtStart = prior_in_rtStop = REFTIME_INVALID;
9.     mpeg2_in_doubt = codecId == AV_CODEC_ID_MPEG2VIDEO;
10.
11.     int using_dxva = 0;
12.
13.     int numthreads = deci->getParam2(IDFF_numLAVCdecThreads);
14.     int thread_type = 0;
15.     if (numthreads > 1 && sup_threads_dec_frame(codecId)) {
16.         thread_type = FF_THREAD_FRAME;
17.     } else if (numthreads > 1 && sup_threads_dec_slice(codecId)) {
18.         thread_type = FF_THREAD_SLICE;
19.     }
20.
21.     if (numthreads > 1 && thread_type != 0) {
22.         threadcount = numthreads;
23.     } else {
24.         threadcount = 1;
25.     }
26.
27.     if (codecId == CODEC_ID_H264_DXVA) {
28.         codecId = AV_CODEC_ID_H264;
29.         using_dxva = 1;
30.     } else if (codecId == CODEC_ID_VC1_DXVA) {
31.         codecId = AV_CODEC_ID_VC1;
32.         using_dxva = 1;
33.     }
34.
35.     avcodec = libavcodec->avcodec_find_decoder(codecId);
36.     if (!avcodec) {
37.         return false;
38.     }
39.     avctx = libavcodec->avcodec_alloc_context(avcodec, this);
40.     avctx->thread_type = thread_type;
41.     avctx->thread_count = threadcount;
42.     avctx->h264_using_dxva = using_dxva;
43.     if (codecId == AV_CODEC_ID_H264) {
44.         // If we do not set this, first B-frames before the IDR pictures are dropped.
45.         avctx->has_b_frames = 1;
46.     }
47.
48.     frame = libavcodec->avcodec_alloc_frame();
49.     avctx->width = pict.rectFull.dx;
50.     avctx->height = pict.rectFull.dy;
51.     intra_matrix = avctx->intra_matrix = (uint16_t*)calloc(sizeof(uint16_t), 64);
52.     inter_matrix = avctx->inter_matrix = (uint16_t*)calloc(sizeof(uint16_t), 64);
53.     ownmatrices = true;
54.
55.
56.     // Fix for new Haali custom media type and fourcc. ffmpeg does not understand it, we have to change it to FOURCC_AVC1
57.     if (fcc == FOURCC_CCV1) {
58.         fcc = FOURCC_AVC1;
59.     }
60.
61.     avctx->codec_tag = fcc;
62.     avctx->workaround_bugs = deci->getParam2(IDFF_workaroundBugs);
63. #if 0
64.     avctx->error_concealment = FF_EC_GUESS_MVS | FF_EC_DEBLOCK;
65.     avctx->err_recognition = AV_EF_CRC_CHECK | AV_EF_BITSTREAM | AV_EF_BUFFER | AV_EF_COMPLIANT | AV_EF_AGGRESSIVE;
66. #endif
67.     if (codecId == AV_CODEC_ID_MJPEG) {
68.         avctx->flags |= CODEC_FLAG_TRUNCATED;
69.     }
70.     if (mpeg12_codec(codecId) && deci->getParam2(IDFF_fastMpeg2)) {
71.         avctx->flags2 = CODEC_FLAG2_FAST;
72.     }
73.     if (codecId == AV_CODEC_ID_H264)
74.         if (int skip = deci->getParam2(IDFF_fastH264)) {
75.             avctx->skip_loop_filter = skip & 2 ? AVDISCARD_ALL : AVDISCARD_NONREF;
76.         }
77.     initialSkipLoopFilter = avctx->skip_loop_filter;
78.
79.     avctx->debug_mv = !using_dxva; //(deci->getParam2(IDFF_isVis) & deci->getParam2(IDFF_visMV));
80.
81.     avctx->idct_algo = limit(deci->getParam2(IDFF_idct), 0, 6);
82.     if (extradata) {
83.         delete extradata;
84.     }
85.     extradata = new Textradata(mt, FF_INPUT_BUFFER_PADDING_SIZE);
86.     此处代码太长，略...
87. }
```

从代码中可以看出这个函数的流程是：

```
1.avcodec_find_decoder();
2.avcodec_alloc_context();
3.avcodec_alloc_frame();
4.avcodec_open();
```

主要做了libavcodec初始化工作。

**begin decompress()用于解码器的初始化。** 注：这个函数的代码太长了，因此只选择一点关键的代码。

```
[cpp]
1. HRESULT TvideoCodecLibavcodec::decompress(const unsigned char *src, size_t srcLen, IMediaSample *pIn)
2. {
3.     代码太长，略...
4.     AVPacket avpkt;
5.     libavcodec->av_init_packet(&avpkt);
6.     if (palette_size) {
7.         uint32_t *pal = (uint32_t *)libavcodec->av_packet_new_side_data(&avpkt, AV_PKT_DATA_PALETTE, AVPALETTE_SIZE);
8.         for (int i = 0; i < palette_size / 4; i++) {
9.             pal[i] = 0xFF << 24 | AV_RL32(palette + i);
10.        }
11.    }
12.
13.    while (!src || size > 0) {
14.        int used_bytes;
15.
16.        avctx->reordered_opaque = rtStart;
17.        avctx->reordered_opaque2 = rtStop;
18.        avctx->reordered_opaque3 = size;
19.
20.        if (sendextradata && extradata->data && extradata->size > 0) {
21.            avpkt.data = (uint8_t *)extradata->data;
22.            avpkt.size = (int)extradata->size;
23.            used_bytes = libavcodec->avcodec_decode_video2(avctx, frame, &got_picture, &avpkt);
24.            sendextradata = false;
25.            if (used_bytes > 0) {
26.                used_bytes = 0;
27.            }
28.            if (mpeg12_codec(codecId)) {
29.                avctx->extradata = NULL;
30.                avctx->extradata_size = 0;
31.            }
32.        } else {
33.            unsigned int neededsize = size + FF_INPUT_BUFFER_PADDING_SIZE;
34.
35.            if (ffbuflen < neededsize) {
36.                ffbuf = (unsigned char*)realloc(ffbuf, ffbuflen = neededsize);
37.            }
38.
39.            if (src) {
40.                memcpy(ffbuf, src, size);
41.                memset(ffbuf + size, 0, FF_INPUT_BUFFER_PADDING_SIZE);
42.            }
43.            if (parser) {
44.                uint8_t *outBuf = NULL;
45.                int out_size = 0;
46.                used_bytes = libavcodec-
>av_parser_parse2(parser, avctx, &outBuf, &out_size, src ? ffbuf : NULL, size, AV_NOPTS_VALUE, AV_NOPTS_VALUE, 0);
47.                if (prior_in_rtStart == REFTIME_INVALID) {
48.                    prior_in_rtStart = rtStart;
49.                    prior_in_rtStop = rtStop;
50.                }
51.                if (out_size > 0 || !src) {
52.                    mpeg2_in_doubt = false;
53.                    avpkt.data = out_size > 0 ? outBuf : NULL;
54.                    avpkt.size = out_size;
55.                    if (out_size > used_bytes) {
56.                        avctx->reordered_opaque = prior_in_rtStart;
57.                        avctx->reordered_opaque2 = prior_in_rtStop;
58.                    } else {
59.                        avctx->reordered_opaque = rtStart;
60.                        avctx->reordered_opaque2 = rtStop;
61.                    }
62.                    prior_in_rtStart = rtStart;
63.                    prior_in_rtStop = rtStop;
64.                    avctx->reordered_opaque3 = out_size;
65.                    if (h264RandomAccess.search(avpkt.data, avpkt.size)) {
66.                        libavcodec->avcodec_decode_video2(avctx, frame, &got_picture, &avpkt);
67.                        h264RandomAccess.judgeUsability(&got_picture);
68.                    } else {
69.                        got_picture = 0;
70.                    }
71.                } else {
72.                    got_picture = 0;
73.                }
74.            } else {
75.                avpkt.data = src ? ffbuf : NULL;
```

```
76.         avpkt.size = size;
77.         if (codecId == AV_CODEC_ID_H264) {
78.             if (h264RandomAccess.search(avpkt.data, avpkt.size)) {
79.                 used_bytes = libavcodec->avcodec_decode_video2(avctx, &got_picture, &avpkt);
80.                 if (used_bytes < 0) {
81.                     return S_OK;
82.                 }
83.                 h264RandomAccess.judgeUsability(&got_picture);
84.             } else {
85.                 got_picture = 0;
86.                 return S_OK;
87.             }
88.         } else {
89.             used_bytes = libavcodec->avcodec_decode_video2(avctx, frame, &got_picture, &avpkt);
90.         }
91.     }
92. }
93.     代码太长，略...
94. }
```

从代码中可以看出这个函数的流程是：

- 1.AVPacket avpkt;
- 2.av\_init\_packet();
- 3.avcodec\_decode\_video2();

和ffmpeg的解码流程相差不大。

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/15493521>

文章标签：[ffdshow](#) [libavcodec](#) [解码器](#) [源代码](#) [ffmpeg](#)

个人分类：[ffdshow](#)

所属专栏：[开源多媒体项目源代码分析](#)

此PDF由spygg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com