

原 FFMpeg源代码简单分析：avformat_write_header()

2015年03月08日 18:45:57 阅读数：24428

=====

FFmpeg的库函数源代码分析文章列表：

【架构图】

[FFmpeg 源代码结构图 - 解码](#)

[FFmpeg 源代码结构图 - 编码](#)

【通用】

[FFmpeg 源代码简单分析：av_register_all\(\)](#)

[FFmpeg 源代码简单分析：avcodec_register_all\(\)](#)

[FFmpeg 源代码简单分析：内存的分配和释放（av_malloc\(\)、av_free\(\)等）](#)

[FFmpeg 源代码简单分析：常见结构体的初始化和销毁（AVFormatContext，AVFrame等）](#)

[FFmpeg 源代码简单分析：avio_open2\(\)](#)

[FFmpeg 源代码简单分析：av_find_decoder\(\)和av_find_encoder\(\)](#)

[FFmpeg 源代码简单分析：avcodec_open2\(\)](#)

[FFmpeg 源代码简单分析：avcodec_close\(\)](#)

【解码】

[图解 FFMPEG 打开媒体的函数 avformat_open_input](#)

[FFmpeg 源代码简单分析：avformat_open_input\(\)](#)

[FFmpeg 源代码简单分析：avformat_find_stream_info\(\)](#)

[FFmpeg 源代码简单分析：av_read_frame\(\)](#)

[FFmpeg 源代码简单分析：avcodec_decode_video2\(\)](#)

[FFmpeg 源代码简单分析：avformat_close_input\(\)](#)

【编码】

[FFmpeg 源代码简单分析：avformat_alloc_output_context2\(\)](#)

[FFmpeg 源代码简单分析：avformat_write_header\(\)](#)

[FFmpeg 源代码简单分析：avcodec_encode_video\(\)](#)

[FFmpeg 源代码简单分析：av_write_frame\(\)](#)

[FFmpeg 源代码简单分析：av_write_trailer\(\)](#)

【其它】

[FFmpeg 源代码简单分析：日志输出系统（av_log\(\)等）](#)

[FFmpeg 源代码简单分析：结构体成员管理系统 -AVClass](#)

[FFmpeg 源代码简单分析：结构体成员管理系统 -AVOption](#)

[FFmpeg 源代码简单分析：libswscale 的 sws_getContext\(\)](#)

[FFmpeg 源代码简单分析：libswscale 的 sws_scale\(\)](#)

[FFmpeg 源代码简单分析：libavdevice 的 avdevice_register_all\(\)](#)

[FFmpeg 源代码简单分析：libavdevice 的 gdigrab](#)

【脚本】

[FFmpeg 源代码简单分析：makefile](#)

[FFmpeg 源代码简单分析：configure](#)

【H.264】

[FFmpeg 的 H.264 解码器源代码简单分析：概述](#)

打算写两篇文章简单分析FFmpeg的写文件用到的3个函数：avformat_write_header(), av_write_frame()以及av_write_trailer()。其中av_write_frame()用于写视频数据，avformat_write_header()用于写视频文件头，而av_write_trailer()用于写视频文件尾。

本文首先分析avformat_write_header()。

PS：

需要注意的是，尽管这3个函数功能是配套的，但是它们的前缀却不一样，写文件头Header的函数前缀是“avformat_”，其他两个函数前缀是“av_”（不太明白其中的原因）。

avformat_write_header()的声明位于libavformat\avformat.h，如下所示。

```
[cpp]
1.  /**
2.   * Allocate the stream private data and write the stream header to
3.   * an output media file.
4.   *
5.   * @param s Media file handle, must be allocated with avformat_alloc_context().
6.   *         Its oformat field must be set to the desired output format;
7.   *         Its pb field must be set to an already opened AVIOContext.
8.   * @param options An AVDictionary filled with AVFormatContext and muxer-private options.
9.   *               On return this parameter will be destroyed and replaced with a dict containing
10.  *               options that were not found. May be NULL.
11.  *
12.  * @return 0 on success, negative AVERROR on failure.
13.  *
14.  * @see av_opt_find, av_dict_set, avio_open, av_oformat_next.
15.  */
16.  int avformat_write_header(AVFormatContext *s, AVDictionary **options);
```

简单解释一下它的参数的含义：

s：用于输出的AVFormatContext。

options：额外的选项，目前没有深入研究过，一般为NULL。

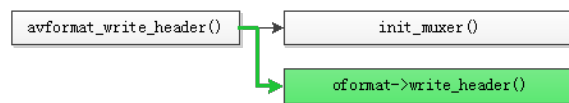
函数正常执行后返回值等于0。

该函数最典型的例子可以参考：

[最简单的基于FFMPEG的视频编码器（YUV编码为H.264）](#)

函数调用关系图

avformat_write_header()的调用关系如下图所示。



struct AVOutputFormat *oformat;

雷霄骅 (Lei Xiaohua)
leixiaohua1020@126.com
<http://blog.csdn.net/leixiaohua1020>

avformat_write_header()

avformat_write_header()的定义位于libavformat\mux.c，如下所示。

```
[cpp]
1. int avformat_write_header(AVFormatContext *s, AVDictionary **options)
2. {
3.     int ret = 0;
4.
5.     if (ret = init_muxer(s, options))
6.         return ret;
7.
8.     if (s->oformat->write_header) {
9.         ret = s->oformat->write_header(s);
10.        if (ret >= 0 && s->pb && s->pb->error < 0)
11.            ret = s->pb->error;
12.        if (ret < 0)
13.            return ret;
14.        if (s->flush_packets && s->pb && s->pb->error >= 0 && s->flags & AVFMT_FLAG_FLUSH_PACKETS)
15.            avio_flush(s->pb);
16.    }
17.
18.    if ((ret = init_pts(s)) < 0)
19.        return ret;
20.
21.    if (s->avoid_negative_ts < 0) {
22.        av_assert2(s->avoid_negative_ts == AVFMT_AVOID_NEG_TS_AUTO);
23.        if (s->oformat->flags & (AVFMT_TS_NEGATIVE | AVFMT_NOTIMESTAMPS)) {
24.            s->avoid_negative_ts = 0;
25.        } else
26.            s->avoid_negative_ts = AVFMT_AVOID_NEG_TS_MAKE_NON_NEGATIVE;
27.    }
28.
29.    return 0;
30. }
```

从源代码可以看出，avformat_write_header()完成了以下工作：

- (1) 调用init_muxer()初始化复用器
- (2) 调用AVOutputFormat的write_header()

下面看一下这两个函数。

init_muxer()

init_muxer()用于初始化复用器，它的定义如下所示。

```
[cpp]
1. static int init_muxer(AVFormatContext *s, AVDictionary **options)
2. {
3.     int ret = 0, i;
4.     AVStream *st;
5.     AVDictionary *tmp = NULL;
6.     AVCodecContext *codec = NULL;
7.     AVOutputFormat *of = s->oformat;
8.     AVDictionaryEntry *e;
9.
10.    if (options)
11.        av_dict_copy(&tmp, *options, 0);
12.
13.    if ((ret = av_opt_set_dict(s, &tmp)) < 0)
14.        goto fail;
15.    if (s->priv_data && s->oformat->priv_class && *(const AVClass**)s->priv_data==s->oformat->priv_class &&
16.        (ret = av_opt_set_dict2(s->priv_data, &tmp, AV_OPT_SEARCH_CHILDREN)) < 0)
17.        goto fail;
18.
19.    #if FF_API_LAVF_BITEXACT
20.        if (s->nb_streams && s->streams[0]->codec->flags & CODEC_FLAG_BITEXACT)
21.            s->flags |= AVFMT_FLAG_BITEXACT;
22.    #endif
23.
24.    // some sanity checks
25.    if (s->nb_streams == 0 && !(of->flags & AVFMT_NOSTREAMS)) {
26.        av_log(s, AV_LOG_ERROR, "No streams to mux were specified\n");
27.        ret = AVERROR(EINVAL);
28.        goto fail;
29.    }
30.
31.    for (i = 0; i < s->nb_streams; i++) {
32.        st = s->streams[i];
33.        codec = st->codec;
34.
35.        #if FF_API_LAVF_CODEC_TB
36.        FF_DISABLE_DEPRECATED_WARNINGS
37.        if (!st->time_base.num && codec->time_base.num) {
38.            av_log(s, AV_LOG_WARNING, "Using AVStream.codec.time_base as a "
39.                "timebase hint to the muxer is deprecated. Set "
40.                "AVStream.time_base instead.\n");
41.            avpriv_set_pts_info(st, 64, codec->time_base.num, codec->time_base.den);
42.        }
43.        FF_ENABLE_DEPRECATED_WARNINGS
44.    #endif
```

```

45.
46.     if (!st->time_base.num) {
47.         /* fall back on the default timebase values */
48.         if (codec->codec_type == AVMEDIA_TYPE_AUDIO && codec->sample_rate)
49.             avpriv_set_pts_info(st, 64, 1, codec->sample_rate);
50.         else
51.             avpriv_set_pts_info(st, 33, 1, 90000);
52.     }
53.
54.     switch (codec->codec_type) {
55.     case AVMEDIA_TYPE_AUDIO:
56.         if (codec->sample_rate <= 0) {
57.             av_log(s, AV_LOG_ERROR, "sample rate not set\n");
58.             ret = AERROR(EINVAL);
59.             goto fail;
60.         }
61.         if (!codec->block_align)
62.             codec->block_align = codec->channels *
63.                 av_get_bits_per_sample(codec->codec_id) >> 3;
64.         break;
65.     case AVMEDIA_TYPE_VIDEO:
66.         if ((codec->width <= 0 || codec->height <= 0) &&
67.             !(of->flags & AVFMT_NODIMENSIONS)) {
68.             av_log(s, AV_LOG_ERROR, "dimensions not set\n");
69.             ret = AERROR(EINVAL);
70.             goto fail;
71.         }
72.         if (av_cmp_q(st->sample_aspect_ratio, codec->sample_aspect_ratio)
73.             && FFABS(av_q2d(st->sample_aspect_ratio) - av_q2d(codec->sample_aspect_ratio)) > 0.004*av_q2d(st->sample_aspect_rati
o)
74.         ) {
75.             if (st->sample_aspect_ratio.num != 0 &&
76.                 st->sample_aspect_ratio.den != 0 &&
77.                 codec->sample_aspect_ratio.num != 0 &&
78.                 codec->sample_aspect_ratio.den != 0) {
79.                 av_log(s, AV_LOG_ERROR, "Aspect ratio mismatch between muxer "
80.                     "(%d/%d) and encoder layer (%d/%d)\n",
81.                     st->sample_aspect_ratio.num, st->sample_aspect_ratio.den,
82.                     codec->sample_aspect_ratio.num,
83.                     codec->sample_aspect_ratio.den);
84.                 ret = AERROR(EINVAL);
85.                 goto fail;
86.             }
87.         }
88.         break;
89.     }
90.
91.     if (of->codec_tag) {
92.         if ( codec->codec_tag
93.             && codec->codec_id == AV_CODEC_ID_RAWVIDEO
94.             && ( av_codec_get_tag(of->codec_tag, codec->codec_id) == 0
95.                 || av_codec_get_tag(of->codec_tag, codec->codec_id) == MKTAG('r', 'a', 'w', ' ') )
96.             && !validate_codec_tag(s, st)) {
97.             // the current rawvideo encoding system ends up setting
98.             // the wrong codec_tag for avi/mov, we override it here
99.             codec->codec_tag = 0;
100.        }
101.        if (codec->codec_tag) {
102.            if (!validate_codec_tag(s, st)) {
103.                char tagbuf[32], tagbuf2[32];
104.                av_get_codec_tag_string(tagbuf, sizeof(tagbuf), codec->codec_tag);
105.                av_get_codec_tag_string(tagbuf2, sizeof(tagbuf2), av_codec_get_tag(s->oformat->codec_tag, codec->codec_id));
106.                av_log(s, AV_LOG_ERROR,
107.                    "Tag %s/0x%08x incompatible with output codec id '%d' (%s)\n",
108.                    tagbuf, codec->codec_tag, codec->codec_id, tagbuf2);
109.                ret = AERROR_INVALIDDATA;
110.                goto fail;
111.            }
112.        } else
113.            codec->codec_tag = av_codec_get_tag(of->codec_tag, codec->codec_id);
114.    }
115.
116.    if (of->flags & AVFMT_GLOBALHEADER &&
117.        !(codec->flags & CODEC_FLAG_GLOBAL_HEADER))
118.        av_log(s, AV_LOG_WARNING,
119.            "Codec for stream %d does not use global headers "
120.            "but container format requires global headers\n", i);
121.
122.    if (codec->codec_type != AVMEDIA_TYPE_ATTACHMENT)
123.        s->internal->nb_interleaved_streams++;
124. }
125.
126. if (!s->priv_data && of->priv_data_size > 0) {
127.     s->priv_data = av_malloc(of->priv_data_size);
128.     if (!s->priv_data) {
129.         ret = AERROR(ENOMEM);
130.         goto fail;
131.     }
132.     if (of->priv_class) {
133.         *(const AVClass **)s->priv_data = of->priv_class;
134.         av_opt_set_defaults(s->priv_data);

```

```

135.         if ((ret = av_opt_set_dict2(s->priv_data, &tmp, AV_OPT_SEARCH_CHILDREN)) < 0)
136.             goto fail;
137.     }
138. }
139.
140. /* set muxer identification string */
141. if (!(s->flags & AVFMT_FLAG_BITEACT)) {
142.     av_dict_set(&s->metadata, "encoder", LIBAVFORMAT_IDENT, 0);
143. } else {
144.     av_dict_set(&s->metadata, "encoder", NULL, 0);
145. }
146.
147. for (e = NULL; e = av_dict_get(s->metadata, "encoder-", e, AV_DICT_IGNORE_SUFFIX); ) {
148.     av_dict_set(&s->metadata, e->key, NULL, 0);
149. }
150.
151. if (options) {
152.     av_dict_free(options);
153.     *options = tmp;
154. }
155.
156. return 0;
157.
158. fail:
159.     av_dict_free(&tmp);
160.     return ret;
161. }

```

init_muxer()代码很长，但是它所做的工作比较简单，可以概括成两个字：检查。函数的流程可以概括成以下几步：

- (1) 将传入的AVDictionary形式的选项设置到AVFormatContext
- (2) 遍历AVFormatContext中的每个AVStream，并作如下检查：
 - a) AVStream的time_base是否正确设置。如果发现AVStream的time_base没有设置，则会调用avpriv_set_pts_info()进行设置。
 - b) 对于音频，检查采样率设置是否正确；对于视频，检查宽、高、宽高比。
 - c) 其他一些检查，不再详述。

AVOutputFormat->write_header()

avformat_write_header()中最关键的地方就是调用了AVOutputFormat的write_header()。write_header()是AVOutputFormat中的一个函数指针，指向写文件头的函数。不同的AVOutputFormat有不同的write_header()的实现方法。在这里我们举例子看一下FLV封装格式对应的AVOutputFormat，它的定义位于libavformat/flvenc.c，如下所示。

```

1. AVOutputFormat ff_flv_muxer = {
2.     .name           = "flv",
3.     .long_name      = NULL_IF_CONFIG_SMALL("FLV (Flash Video)"),
4.     .mime_type       = "video/x-flv",
5.     .extensions     = "flv",
6.     .priv_data_size = sizeof(FLVContext),
7.     .audio_codec     = CONFIG_LIBMP3LAME ? AV_CODEC_ID_MP3 : AV_CODEC_ID_ADPCM_SWF,
8.     .video_codec     = AV_CODEC_ID_FLV1,
9.     .write_header    = flv_write_header,
10.    .write_packet     = flv_write_packet,
11.    .write_trailer    = flv_write_trailer,
12.    .codec_tag       = (const AVCodecTag* const []) {
13.        flv_video_codec_ids, flv_audio_codec_ids, 0
14.    },
15.    .flags           = AVFMT_GLOBALHEADER | AVFMT_VARIABLE_FPS |
16.        AVFMT_TS_NONSTRICT,
17. };

```

从ff_flv_muxer的定义中可以看出，write_header()指向的函数为flv_write_header()。我们继续看一下flv_write_header()函数。flv_write_header()的定义同样位于libavformat/flvenc.c，如下所示。

```

1. static int flv_write_header(AVFormatContext *s)
2. {
3.     int i;
4.     AVIOContext *pb = s->pb;
5.     FLVContext *flv = s->priv_data;
6.     int64_t data_size;
7.     //设置参数
8.     for (i = 0; i < s->nb_streams; i++) {
9.         AVCodecContext *enc = s->streams[i]->codec;
10.        FLVStreamContext *sc;
11.        switch (enc->codec_type) {
12.            case AVMEDIA_TYPE_VIDEO:
13.                if (s->streams[i]->avg_frame_rate.den &&
14.                    s->streams[i]->avg_frame_rate.num) {
15.                    //设置帧率，从AVStream拷贝过来
16.                    flv->framerate = av_q2d(s->streams[i]->avg_frame_rate);
17.                }
18.                if (flv->video_enc) {

```

```

19.         av_log(s, AV_LOG_ERROR,
20.             "at most one video stream is supported in flv\n");
21.         return AVERROR(EINVAL);
22.     }
23.     //视频编码的AVCodecContext
24.     flv->video_enc = enc;
25.     if (enc->codec_tag == 0) {
26.         av_log(s, AV_LOG_ERROR, "Video codec '%s' for stream %d is not compatible with FLV\n",
27.             avcodec_get_name(enc->codec_id), i);
28.         return AVERROR(EINVAL);
29.     }
30.     if (enc->codec_id == AV_CODEC_ID_MPEG4 ||
31.         enc->codec_id == AV_CODEC_ID_H263) {
32.         int error = s->strict_std_compliance > FF_COMPLIANCE_UNOFFICIAL;
33.         av_log(s, error ? AV_LOG_ERROR : AV_LOG_WARNING,
34.             "Codec %s is not supported in the official FLV specification,\n", avcodec_get_name(enc->codec_id));
35.
36.         if (error) {
37.             av_log(s, AV_LOG_ERROR,
38.                 "use vstrict=-1 / -strict -1 to use it anyway.\n");
39.             return AVERROR(EINVAL);
40.         }
41.     } else if (enc->codec_id == AV_CODEC_ID_VP6) {
42.         av_log(s, AV_LOG_WARNING,
43.             "Muxing VP6 in flv will produce flipped video on playback.\n");
44.     }
45.     break;
46. case AVMEDIA_TYPE_AUDIO:
47.     if (flv->audio_enc) {
48.         av_log(s, AV_LOG_ERROR,
49.             "at most one audio stream is supported in flv\n");
50.         return AVERROR(EINVAL);
51.     }
52.     //音频编码的AVCodecContext
53.     flv->audio_enc = enc;
54.     if (get_audio_flags(s, enc) < 0)
55.         return AVERROR_INVALIDDATA;
56.     if (enc->codec_id == AV_CODEC_ID_PCM_S16BE)
57.         av_log(s, AV_LOG_WARNING,
58.             "16-bit big-endian audio in flv is valid but most likely unplayable (hardware dependent); use s16le\n");
59.     break;
60. case AVMEDIA_TYPE_DATA:
61.     if (enc->codec_id != AV_CODEC_ID_TEXT && enc->codec_id != AV_CODEC_ID_NONE) {
62.         av_log(s, AV_LOG_ERROR, "Data codec '%s' for stream %d is not compatible with FLV\n",
63.             avcodec_get_name(enc->codec_id), i);
64.         return AVERROR_INVALIDDATA;
65.     }
66.     flv->data_enc = enc;
67.     break;
68. default:
69.     av_log(s, AV_LOG_ERROR, "Codec type '%s' for stream %d is not compatible with FLV\n",
70.         av_get_media_type_string(enc->codec_type), i);
71.     return AVERROR(EINVAL);
72. }
73. avpriv_set_pts_info(s->streams[i], 32, 1, 1000); /* 32 bit pts in ms */
74.
75. sc = av_mallocz(sizeof(FLVStreamContext));
76. if (!sc)
77.     return AVERROR(ENOMEM);
78. s->streams[i]->priv_data = sc;
79. sc->last_ts = -1;
80. }
81.
82. flv->delay = AV_NOPTS_VALUE;
83. //开始写入
84. //Signature
85. avio_write(pb, "FLV", 3);
86. //Version
87. avio_w8(pb, 1);
88. //""!!"意思是把非0转换成1
89. //Flags
90. avio_w8(pb, FLV_HEADER_FLAG_HASAUDIO * !!flv->audio_enc +
91.     FLV_HEADER_FLAG_HASVIDEO * !!flv->video_enc);
92. //Header size
93. avio_wb32(pb, 9);
94. //Header结束
95. //Previous Tag Size
96. avio_wb32(pb, 0);
97.
98. for (i = 0; i < s->nb_streams; i++)
99.     if (s->streams[i]->codec->codec_tag == 5) {
100.         avio_w8(pb, 8); // message type
101.         avio_wb24(pb, 0); // include flags
102.         avio_wb24(pb, 0); // time stamp
103.         avio_wb32(pb, 0); // reserved
104.         avio_wb32(pb, 11); // size
105.         flv->reserved = 5;
106.     }
107.
108. write_metadata(s, 0);
109.
110.

```

```

110.     for (i = 0; i < s->nb_streams; i++) {
111.         AVCodecContext *enc = s->streams[i]->codec;
112.         if (enc->codec_id == AV_CODEC_ID_AAC || enc->codec_id == AV_CODEC_ID_H264 || enc->codec_id == AV_CODEC_ID_MPEG4) {
113.             int64_t pos;
114.             avio_w8(pb, enc->codec_type == AVMEDIA_TYPE_VIDEO ?
115.                 FLV_TAG_TYPE_VIDEO : FLV_TAG_TYPE_AUDIO);
116.             avio_wb24(pb, 0); // size patched later
117.             avio_wb24(pb, 0); // ts
118.             avio_w8(pb, 0); // ts ext
119.             avio_wb24(pb, 0); // streamid
120.             pos = avio_tell(pb);
121.             if (enc->codec_id == AV_CODEC_ID_AAC) {
122.                 avio_w8(pb, get_audio_flags(s, enc));
123.                 avio_w8(pb, 0); // AAC sequence header
124.                 avio_write(pb, enc->extradata, enc->extradata_size);
125.             } else {
126.                 avio_w8(pb, enc->codec_tag | FLV_FRAME_KEY); // flags
127.                 avio_w8(pb, 0); // AVC sequence header
128.                 avio_wb24(pb, 0); // composition time
129.                 ff_isom_write_avcc(pb, enc->extradata, enc->extradata_size);
130.             }
131.             data_size = avio_tell(pb) - pos;
132.             avio_seek(pb, -data_size - 10, SEEK_CUR);
133.             avio_wb24(pb, data_size);
134.             avio_skip(pb, data_size + 10 - 3);
135.             avio_wb32(pb, data_size + 11); // previous tag size
136.         }
137.     }
138.
139.     return 0;
140. }

```

从源代码可以看出，flv_write_header()完成了FLV文件头的写入工作。该函数的工作可以大体分为以下两部分：

- (1) 给FLVContext设置参数
- (2) 写文件头，以及相关的Tag

写文件头的代码很短，如下所示。

```

1. avio_write(pb, "FLV", 3);
2. avio_w8(pb, 1);
3. avio_w8(pb, FLV_HEADER_FLAG_HASAUDIO * !!flv->audio_enc +
4.     FLV_HEADER_FLAG_HASVIDEO * !!flv->video_enc);
5. avio_wb32(pb, 9);

```

可以参考下图中FLV文件头的定义比对一下上面的代码。

Flv Header	Signature (3 字节) 为文件标识，总为“FLV”，(0x46, 0x4c, 0x66)
	Version (1 字节) 为版本，目前为 0x01
	Flags (1 字节) 前 5 位保留，必须为 0。第 6 位表示是否存在音频 Tag。第 7 位保留，必须为 0。第 8 位表示是否存在视频 Tag。
	Headersize (4 字节) 为从 File Header 开始到 File Body 开始的字节数，版本 1 中总为 9。

雷霄骅

leixiaohua1020@126.com

<http://blog.csdn.net/leixiaohua1020>

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/44116215>

文章标签： [FFmpeg](#) [源代码](#) [文件头](#) [封装格式](#)

个人分类： [FFMPEG](#)

所属专栏： [FFmpeg](#)

此PDF由spyyg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com