

## 转 ffmpeg源码分析：transcode\_init()函数

2013年09月20日 17:36:06 阅读数：6776

transcode\_init()函数是在转换前做准备工作的.此处看一下它的真面目,不废话,看注释吧:

```
[cpp]  
1. //为转换过程做准备
2. static int transcode_init(OutputFile *output_files,
3.     int nb_output_files,
4.     InputFile *input_files,
5.     int nb_input_files)
6. {
7.     int ret = 0, i, j, k;
8.     AVFormatContext *oc;
9.     AVCodecContext *codec, *icodec;
10.    OutputStream *ost;
11.    InputStream *ist;
12.    char error[1024];
13.    int want_sdp = 1;
14.
15.    /* init framerate emulation */
16.    //初始化帧率仿真 (转换时是不按帧率来的,但如果要求帧率仿真,就可以做到)
17.    for (i = 0; i < nb_input_files; i++)
18.    {
19.        InputFile *ifile = &input_files[i];
20.        //如果一个输入文件被要求帧率仿真 (指的是即使是转换也像播放那样按照帧率来进行),
21.        //则为这个文件中所有流记录下开始时间
22.        if (ifile->rate_emu)
23.            for (j = 0; j < ifile->nb_streams; j++)
24.                input_streams[j + ifile->ist_index].start = av_gettime();
25.    }
26.
27.    /* output stream init */
28.    for (i = 0; i < nb_output_files; i++)
29.    {
30.        //什么也没做,只是做了个判断而已
31.        oc = output_files[i].ctx;
32.        if (!oc->nb_streams && !(oc->oformat->flags & AVFMT_NOSTREAMS))
33.        {
34.            av_dump_format(oc, i, oc->filename, 1);
35.            av_log(NULL, AV_LOG_ERROR,
36.                "Output file #%d does not contain any stream\n", i);
37.            return AVERROR(EINVAL);
38.        }
39.    }
40.
41.    //轮循所有的输出流,跟据对应的输入流,设置其编解码器的参数
42.    for (i = 0; i < nb_output_streams; i++)
43.    {
44.        //轮循所有的输出流
45.        ost = &output_streams[i];
46.        //输出流对应的FormatContext
47.        oc = output_files[ost->file_index].ctx;
48.        //取得输出流对应的输入流
49.        ist = &input_streams[ost->source_index];
50.
51.        //attachment_filename是不是这样的东西:-一个文件,它单独容纳一个输出流?此处不懂
52.        if (ost->attachment_filename)
53.            continue;
54.
55.        codec = ost->st->codec; //输出流的编解码器结构
56.        icodec = ist->st->codec; //输入流的编解码器结构
57.
58.        //先把能复制的复制一下
59.        ost->st->disposition = ist->st->disposition;
60.        codec->bits_per_raw_sample = icodec->bits_per_raw_sample;
61.        codec->chroma_sample_location = icodec->chroma_sample_location;
62.
63.        //如果只是复制一个流(不用解码后再编码),则把输入流的编码参数直接复制给输出流
64.        //此时是不需要解码也不需要编码的,所以不需打开解码器和编码器
65.        if (ost->stream_copy)
66.        {
67.            //计算输出流的编解码器的extradata的大小,然后分配容纳extradata的缓冲
68.            //然后把输入流的编解码器的extradata复制到输出流的编解码器中
69.            uint64_t extra_size = (uint64_t) icodec->extradata_size
70.                + FF_INPUT_BUFFER_PADDING_SIZE;
71.
72.            if (extra_size > INT_MAX) {
73.                return AVERROR(EINVAL);
74.            }
75.
76.            /* if stream_copy is selected, no need to decode or encode */
77.            codec->codec_id = icodec->codec_id;
78.            codec->codec_type = icodec->codec_type;
79.        }
```

```

80.     if (!codec->codec_tag){
81.         if (!oc->oformat->codec_tag
82.             || av_codec_get_id(oc->oformat->codec_tag, codec->codec_tag) == codec->codec_id
83.             || av_codec_get_tag(oc->oformat->codec_tag, codec->codec_id) <= 0)
84.             codec->codec_tag = codec->codec_tag;
85.     }
86.
87.     codec->bit_rate = icodec->bit_rate;
88.     codec->rc_max_rate = icodec->rc_max_rate;
89.     codec->rc_buffer_size = icodec->rc_buffer_size;
90.     codec->extradata = av_mallocz(extra_size);
91.     if (!codec->extradata){
92.         return AVERROR(ENOMEM);
93.     }
94.     memcpy(codec->extradata, icodec->extradata, icodec->extradata_size);
95.     codec->extradata_size = icodec->extradata_size;
96.
97.     //重新鼓捣一下time base(这家伙就是帧率)
98.     codec->time_base = ist->st->time_base;
99.     //如果输出文件是avi,做一点特殊处理
100.    if (!strcmp(oc->oformat->name, "avi")) {
101.        if (copy_tb < 0
102.            && av_q2d(icodec->time_base) * icodec->ticks_per_frame >
103.                2 * av_q2d(ist->st->time_base)
104.            && av_q2d(ist->st->time_base) < 1.0 / 500
105.            || copy_tb == 0)
106.        {
107.            codec->time_base = icodec->time_base;
108.            codec->time_base.num *= icodec->ticks_per_frame;
109.            codec->time_base.den *= 2;
110.        }
111.    }
112.    else if (!(oc->oformat->flags & AVFMT_VARIABLE_FPS))
113.    {
114.        if (copy_tb < 0
115.            && av_q2d(icodec->time_base) * icodec->ticks_per_frame
116.                > av_q2d(ist->st->time_base)
117.            && av_q2d(ist->st->time_base) < 1.0 / 500
118.            || copy_tb == 0)
119.        {
120.            codec->time_base = icodec->time_base;
121.            codec->time_base.num *= icodec->ticks_per_frame;
122.        }
123.    }
124.
125.    //再修正一下帧率
126.    av_reduce(&codec->time_base.num, &codec->time_base.den,
127.        codec->time_base.num, codec->time_base.den, INT_MAX);
128.
129.    //单独复制各不同媒体自己的编码参数
130.    switch (codec->codec_type)
131.    {
132.    case AVMEDIA_TYPE_AUDIO:
133.        //音频的
134.        if (audio_volume != 256){
135.            av_log( NULL, AV_LOG_FATAL,
136.                "-acodec copy and -vol are incompatible (frames are not decoded)\n");
137.            exit_program(1);
138.        }
139.        codec->channel_layout = icodec->channel_layout;
140.        codec->sample_rate = icodec->sample_rate;
141.        codec->channels = icodec->channels;
142.        codec->frame_size = icodec->frame_size;
143.        codec->audio_service_type = icodec->audio_service_type;
144.        codec->block_align = icodec->block_align;
145.        break;
146.    case AVMEDIA_TYPE_VIDEO:
147.        //视频的
148.        codec->pix_fmt = icodec->pix_fmt;
149.        codec->width = icodec->width;
150.        codec->height = icodec->height;
151.        codec->has_b_frames = icodec->has_b_frames;
152.        if (!codec->sample_aspect_ratio.num){
153.            codec->sample_aspect_ratio = ost->st->sample_aspect_ratio =
154.                ist->st->sample_aspect_ratio.num ? ist->st->sample_aspect_ratio :
155.                ist->st->codec->sample_aspect_ratio.num ? ist->st->codec->sample_aspect_ratio : (AVRational){0, 1}
156.        }
157.        ost->st->avg_frame_rate = ist->st->avg_frame_rate;
158.        break;
159.    case AVMEDIA_TYPE_SUBTITLE:
160.        //字幕的
161.        codec->width = icodec->width;
162.        codec->height = icodec->height;
163.        break;
164.    case AVMEDIA_TYPE_DATA:
165.    case AVMEDIA_TYPE_ATTACHMENT:
166.        //??的
167.        break;
168.    default:
169.        abort();

```

```

170.     }
171. }
172. else
173. {
174.     //如果不是复制,就麻烦多了
175.
176.     //获取编码器
177.     if (!ost->enc)
178.         ost->enc = avcodec_find_encoder(ost->st->codec->codec_id);
179.
180.     //因为需要转换,所以既需解码又需编码
181.     ist->decoding_needed = 1;
182.     ost->encoding_needed = 1;
183.
184.     switch(codec->codec_type)
185.     {
186.     case AVMEDIA_TYPE_AUDIO:
187.         //鼓捣音频编码器的参数,基本上是把一些不合适的参数替换掉
188.         ost->fifo = av_fifo_alloc(1024); //音频数据所在的缓冲
189.         if (!ost->fifo) {
190.             return AVERROR(ENOMEM);
191.         }
192.
193.         //采样率
194.         if (!codec->sample_rate)
195.             codec->sample_rate = icodec->sample_rate;
196.         choose_sample_rate(ost->st, ost->enc);
197.         codec->time_base = (AVRational){1, codec->sample_rate};
198.
199.         //样点格式
200.         if (codec->sample_fmt == AV_SAMPLE_FMT_NONE)
201.             codec->sample_fmt = icodec->sample_fmt;
202.         choose_sample_fmt(ost->st, ost->enc);
203.
204.         //声道
205.         if (ost->audio_channels_mapped) {
206.             /* the requested output channel is set to the number of
207.              * -map_channel only if no -ac are specified */
208.             if (!codec->channels) {
209.                 codec->channels = ost->audio_channels_mapped;
210.                 codec->channel_layout = av_get_default_channel_layout(codec->channels);
211.                 if (!codec->channel_layout) {
212.                     av_log(NULL, AV_LOG_FATAL, "Unable to find an appropriate channel layout for requested number of channel\n");
213.                     exit_program(1);
214.                 }
215.             }
216.             /* fill unused channel mapping with -1 (which means a muted
217.              * channel in case the number of output channels is bigger
218.              * than the number of mapped channel) */
219.             for (j = ost->audio_channels_mapped; j < FF_ARRAY_ELEMS(ost->audio_channels_map); j++)
220.                 ost->audio_channels_map[j] = -1;
221.         } else if (!codec->channels) {
222.             codec->channels = icodec->channels;
223.             codec->channel_layout = icodec->channel_layout;
224.         }
225.         if (av_get_channel_layout_nb_channels(codec->channel_layout) != codec->channels)
226.             codec->channel_layout = 0;
227.
228.         //是否需要重采样
229.         ost->audio_resample = codec->sample_rate != icodec->sample_rate || audio_sync_method > 1;
230.         ost->audio_resample |= codec->sample_fmt != icodec->sample_fmt ||
231.             codec->channel_layout != icodec->channel_layout;
232.         icodec->request_channels = codec->channels;
233.         ost->resample_sample_fmt = icodec->sample_fmt;
234.         ost->resample_sample_rate = icodec->sample_rate;
235.         ost->resample_channels = icodec->channels;
236.         break;
237.     case AVMEDIA_TYPE_VIDEO:
238.         //鼓捣视频编码器的参数,基本上是把一些不合适的参数替换掉
239.         if (codec->pix_fmt == PIX_FMT_NONE)
240.             codec->pix_fmt = icodec->pix_fmt;
241.         choose_pixel_fmt(ost->st, ost->enc);
242.         if (ost->st->codec->pix_fmt == PIX_FMT_NONE) {
243.             av_log(NULL, AV_LOG_FATAL, "Video pixel format is unknown, stream cannot be encoded\n");
244.             exit_program(1);
245.         }
246.
247.         //宽高
248.         if (!codec->width || !codec->height) {
249.             codec->width = icodec->width;
250.             codec->height = icodec->height;
251.         }
252.
253.         //视频是否需要重采样
254.         ost->video_resample = codec->width != icodec->width ||
255.             codec->height != icodec->height ||
256.             codec->pix_fmt != icodec->pix_fmt;
257.         if (ost->video_resample) {
258.             codec->bits_per_raw_sample = frame_bits_per_raw_sample;
259.         }

```

```

260.
261.     ost->resample_height = icodec->height;
262.     ost->resample_width = icodec->width;
263.     ost->resample_pix_fmt = icodec->pix_fmt;
264.
265.     //计算帧率
266.     if (!ost->frame_rate.num)
267.         ost->frame_rate = ist->st->r_frame_rate.num ?
268.             ist->st->r_frame_rate : (AVRational){25,1};
269.     if (ost->enc && ost->enc->supported_framerates && !ost->force_fps) {
270.         int idx = av_find_nearest_q_idx(ost->frame_rate, ost->enc->supported_framerates);
271.         ost->frame_rate = ost->enc->supported_framerates[idx];
272.     }
273.     codec->time_base = (AVRational) {ost->frame_rate.den, ost->frame_rate.num};
274.     if( av_q2d(codec->time_base) < 0.001 &&
275.         video_sync_method &&
276.         (video_sync_method==1 ||
277.          (video_sync_method<0 && !
278.           (oc->oformat->flags & AVFMT_VARIABLE_FPS))))
279.     {
280.         av_log(oc, AV_LOG_WARNING, "Frame rate very high for a muxer not efficiently supporting it.\n"
281.             "Please consider specifying a lower framerate, a different muxer or -vsync 2\n");
282.     }
283.     for (j = 0; j < ost->forced_kf_count; j++)
284.         ost->forced_kf_pts[j] = av_rescale_q(ost->forced_kf_pts[j],
285.             AV_TIME_BASE_Q, codec->time_base);
286.     break;
287. case AVMEDIA_TYPE_SUBTITLE:
288.     break;
289. default:
290.     abort();
291.     break;
292. }
293. /* two pass mode */
294. if (codec->codec_id != CODEC_ID_H264 &&
295.     (codec->flags & (CODEC_FLAG_PASS1 | CODEC_FLAG_PASS2)))
296. {
297.     char logfilename[1024];
298.     FILE *f;
299.
300.     snprintf(logfilename, sizeof(logfilename), "%s-%d.log",
301.         pass_logfilename_prefix ? pass_logfilename_prefix : DEFAULT_PASS_LOGFILENAME_PREFIX,
302.         i);
303.     if (codec->flags & CODEC_FLAG_PASS2){
304.         char *logbuffer;
305.         size_t logbuffer_size;
306.         if (cmdutils_read_file(logfilename, &logbuffer, &logbuffer_size) < 0){
307.             av_log(NULL, AV_LOG_FATAL,
308.                 "Error reading log file '%s' for pass-2 encoding\n",
309.                 logfilename);
310.             exit_program(1);
311.         }
312.         codec->stats_in = logbuffer;
313.     }
314.     if (codec->flags & CODEC_FLAG_PASS1){
315.         f = fopen(logfilename, "wb");
316.         if (!f) {
317.             av_log(NULL, AV_LOG_FATAL, "Cannot write log file '%s' for pass-1 encoding: %s\n",
318.                 logfilename, strerror(errno));
319.             exit_program(1);
320.         }
321.         ost->logfile = f;
322.     }
323. }
324. }
325. if (codec->codec_type == AVMEDIA_TYPE_VIDEO){
326.     /* maximum video buffer size is 6-bytes per pixel, plus DPX header size (1664)*/
327.     //计算编码输出缓冲的大小,计算一个最大值
328.     int size = codec->width * codec->height;
329.     bit_buffer_size = FFMAX(bit_buffer_size, 7 * size + 10000);
330. }
331. }
332.
333. //分配编码后数据所在的缓冲
334. if (!bit_buffer)
335.     bit_buffer = av_malloc(bit_buffer_size);
336. if (!bit_buffer){
337.     av_log(NULL, AV_LOG_ERROR,
338.         "Cannot allocate %d bytes output buffer\n",
339.         bit_buffer_size);
340.     return AVERROR(ENOMEM);
341. }
342.
343. //轮询所有输出流, 打开每个输出流的编码器
344. for (i = 0; i < nb_output_streams; i++)
345. {
346.     ost = &output_streams[i];
347.     if (ost->encoding_needed){
348.         //当然,只有在需要编码时才打开编码器
349.         AVCodec *codec = ost->enc;
350.         AVCodecContext *dec = input_streams[ost->source_index].st->codec;

```

```

351.         if (!codec) {
352.             snprintf(error, sizeof(error),
353.                 "Encoder (codec %s) not found for output stream #%d:%d",
354.                 avcodec_get_name(ost->st->codec->codec_id),
355.                 ost->file_index, ost->index);
356.             ret = AERROR(EINVAL);
357.             goto dump_format;
358.         }
359.         if (dec->subtitle_header){
360.             ost->st->codec->subtitle_header = av_malloc(dec->subtitle_header_size);
361.             if (!ost->st->codec->subtitle_header){
362.                 ret = AERROR(ENOMEM);
363.                 goto dump_format;
364.             }
365.             memcpy(ost->st->codec->subtitle_header,
366.                 dec->subtitle_header, dec->subtitle_header_size);
367.             ost->st->codec->subtitle_header_size = dec->subtitle_header_size;
368.         }
369.         //打开啦
370.         if (avcodec_open2(ost->st->codec, codec, &ost->opts) < 0) {
371.             snprintf(error, sizeof(error),
372.                 "Error while opening encoder for output stream #%d:%d - maybe incorrect parameters such as bit_rate, rate, wi
h or height",
373.                 ost->file_index, ost->index);
374.             ret = AERROR(EINVAL);
375.             goto dump_format;
376.         }
377.         assert_codec_experimental(ost->st->codec, 1);
378.         assert_avoptions(ost->opts);
379.         if (ost->st->codec->bit_rate && ost->st->codec->bit_rate < 1000)
380.             av_log(NULL, AV_LOG_WARNING,
381.                 "The bitrate parameter is set too low."
382.                 " It takes bits/s as argument, not kbits/s\n");
383.         extra_size += ost->st->codec->extradata_size;
384.
385.         if (ost->st->codec->me_threshold)
386.             input_streams[ost->source_index].st->codec->debug |= FF_DEBUG_MV;
387.     }
388. }
389.
390. //初始化所有的输入流(主要做的就是在需要时打开解码器)
391. for (i = 0; i < nb_input_streams; i++)
392.     if ((ret = init_input_stream(i, output_streams, nb_output_streams,
393.         error, sizeof(error))) < 0)
394.         goto dump_format;
395.
396. /* discard unused programs */
397. for (i = 0; i < nb_input_files; i++){
398.     InputFile *ifile = &input_files[i];
399.     for (j = 0; j < ifile->ctx->nb_programs; j++){
400.         AVProgram *p = ifile->ctx->programs[j];
401.         int discard = AVDISCARD_ALL;
402.
403.         for (k = 0; k < p->nb_stream_indexes; k++){
404.             if (!input_streams[ifile->ist_index + p->stream_index[k]].discard){
405.                 discard = AVDISCARD_DEFAULT;
406.                 break;
407.             }
408.         }
409.         p->discard = discard;
410.     }
411. }
412.
413. //打开所有输出文件, 写入媒体文件头
414. for (i = 0; i < nb_output_files; i++){
415.     oc = output_files[i].ctx;
416.     oc->interrupt_callback = int_cb;
417.     if (avformat_write_header(oc, &output_files[i].opts) < 0){
418.         snprintf(error, sizeof(error),
419.             "Could not write header for output file #%d (incorrect codec parameters ?)",
420.             i);
421.         ret = AERROR(EINVAL);
422.         goto dump_format;
423.     }
424.     // assert_avoptions(output_files[i].opts);
425.     if (strcmp(oc->oformat->name, "rtp")){
426.         want_sdp = 0;
427.     }
428. }
429.
430. return 0;
431. }

```

文章标签：[ffmpeg](#) [transcode\\_init](#) [源代码](#)

个人分类：[FFMPEG](#)

所属专栏：[开源多媒体项目源代码分析](#) [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)