





转 ffmpeg源码分析：transcode()函数

2013年09月20日 17:34:05 阅读数：6239

还是先看一下主函数吧(省略了很多无关大雅的代码)

```
[cpp]    
1. int main(int argc, char **argv)  
2. {  
3.     OptionsContext o = { 0 };  
4.     int64_t ti;  
5.  
6.     //与命令行分析有关的结构的初始化,下面不再罗嗦  
7.     reset_options(&o, 0);  
8.  
9.     //设置日志级别  
10.    av_log_set_flags(AV_LOG_SKIP_REPEATED);  
11.    parse_loglevel(argc, argv, options);  
12.  
13.    if (argc > 1 && !strcmp(argv[1], "-d")) {  
14.        run_as_daemon = 1;  
15.        av_log_set_callback(log_callback_null);  
16.        argc--;  
17.        argv++;  
18.    }  
19.  
20.    //注册组件们  
21.    avcodec_register_all();  
22.    #if CONFIG_AVDEVICE  
23.    avdevice_register_all();  
24.    #endif  
25.    #if CONFIG_AVFILTER  
26.    avfilter_register_all();  
27.    #endif  
28.    av_register_all();  
29.    //初始化网络,windows下需要  
30.    avformat_network_init();  
31.  
32.    show_banner();  
33.  
34.    term_init();  
35.  
36.    //分析命令行输入的参数们  
37.    parse_options(&o, argc, argv, options, opt_output_file);  
38.  
39.    //文件的转换就在此函数中发生  
40.    if (transcode(output_files, nb_output_files, input_files, nb_input_files) < 0)  
41.        exit_program(1);  
42.  
43.    exit_program(0);  
44.    return 0;  
45. }
```

下面是transcode()函数,转换就发生在它里面.不废话,看注释吧,应很详细了

```
[cpp]    
1. static int transcode(  
2.     OutputFile *output_files, //输出文件数组  
3.     int nb_output_files, //输出文件的数量  
4.     InputFile *input_files, //输入文件数组  
5.     int nb_input_files) //输入文件的数量  
6. {  
7.     int ret, i;  
8.     AVFormatContext *is, *os;  
9.     OutputStream *ost;  
10.    InputStream *ist;  
11.    uint8_t *no_packet;  
12.    int no_packet_count = 0;  
13.    int64_t timer_start;  
14.    int key;  
15.  
16.    if (!(no_packet = av_mallocz(nb_input_files)))  
17.        exit_program(1);  
18.  
19.    //设置编码参数,打开所有输出流的编码器,打开所有输入流的解码器,写入所有输出文件的文件头,于是准备好了  
20.    ret = transcode_init(output_files, nb_output_files, input_files, nb_input_files);  
21.    if (ret < 0)  
22.        goto fail;  
23.  
24.    if (!using_stdin){  
25.        av_log(NULL, AV_LOG_INFO, "Press [q] to stop, [?] for help\n");  
26.    }  
27. }
```

```

27.
28.     timer_start = av_gettime();
29.
30.     //循环,直到收到系统信号才退出
31.     for (; received_sigterm == 0;)
32.     {
33.         int file_index, ist_index;
34.         AVPacket pkt;
35.         int64_t ipts_min;
36.         double opts_min;
37.         int64_t cur_time = av_gettime();
38.
39.         ipts_min = INT64_MAX;
40.         opts_min = 1e100;
41.         /* if 'q' pressed, exits */
42.         if (!using_stdin)
43.         {
44.             //先查看用户按下了什么键,跟按键做出相应的反应
45.             static int64_t last_time;
46.             if (received_nb_signals)
47.                 break;
48.             /* read_key() returns 0 on EOF */
49.             if (cur_time - last_time >= 100000 && !run_as_daemon){
50.                 key = read_key();
51.                 last_time = cur_time;
52.             }else{
53. <span>
54.             }
55.
56.             /* select the stream that we must read now by looking at the
57.              * smallest output pts */
58.             //下面这个循环的目的是找一个最小的输出pts(也就是离当前最近的)的输出流
59.             file_index = -1;
60.             for (i = 0; i < nb_output_streams; i++){
61.                 OutputFile *of;
62.                 int64_t ipts;
63.                 double opts;
64.                 ost = &output_streams[i]; //循环每一个输出流
65.                 of = &output_files[ost->file_index]; //输出流对应的输出文件
66.                 os = output_files[ost->file_index].ctx; //输出流对应的FormatContext
67.                 ist = &input_streams[ost->source_index]; //输出流对应的输入流
68.
69.                 if (ost->is_past_recording_time || //是否过了录制时间?(可能用户指定了一个录制时间段)
70.                     no_packet[ist->file_index] || //对应的输入流这个时间内没有数据?
71.                     (os->pb && avio_tell(os->pb) >= of->limit_filesize)) //是否超出了录制范围(也是用户指定的)
72.                     continue; //是的,符合上面某一条,那么再看下一个输出流吧
73.
74.                 //判断当前输入流所在的文件是否可以被使用(我也不很明白)
75.                 opts = ost->st->pts.val * av_q2d(ost->st->time_base);
76.                 ipts = ist->pts;
77.                 if (!input_files[ist->file_index].eof_reached) {
78.                     if (ipts < ipts_min){
79.                         //每找到一个pts更小的输入流就记录下来,这样循环完所有的输出流时就找到了
80.                         //pts最小的输入流,及输入文件的序号
81.                         ipts_min = ipts;
82.                         if (input_sync)
83.                             file_index = ist->file_index;
84.                     }
85.                     if (opts < opts_min){
86.                         opts_min = opts;
87.                         if (!input_sync)
88.                             file_index = ist->file_index;
89.                     }
90.                 }
91.
92.                 //难道下面这句话的意思是:如果当前的输出流已接收的帧数,超出用户指定的输出最大帧数时,
93.                 //则当前输出流所属的输出文件对应的所有输出流,都算超过了录像时间?
94.                 if (ost->frame_number >= ost->max_frames){
95.                     int j;
96.                     for (j = 0; j < of->ctx->nb_streams; j++)
97.                         output_streams[of->ost_index + j].is_past_recording_time = 1;
98.                     continue;
99.                 }
100.            }
101.            /* if none, if is finished */
102.            if (file_index < 0) {
103.                //如果没有找到合适的输入文件
104.                if (no_packet_count){
105.                    //如果是因为有的输入文件暂时得不到数据,则还不算是结束
106.                    no_packet_count = 0;
107.                    memset(no_packet, 0, nb_input_files);
108.                    usleep(10000);
109.                    continue;
110.                }
111.                //全部转换完成了,跳出大循环
112.                break;
113.            }
114.
115.            //从找到的输入文件中读出一帧(可能是音频也可能是视频),并放到fifo队列中
116.            is = input_files[file_index].ctx;
117.            ret = av_read_frame(is, &pkt);
118.            if (ret == AVERRROR(EAGAIN)) {

```

```

110.         //此时发生了暂时没数据的情况
111.         no_packet[file_index] = 1;
112.         no_packet_count++;
113.         continue;
114.     }
115.
116.     //下文判断是否有输入文件到最后了
117.     if (ret < 0){
118.         input_files[file_index].eof_reached = 1;
119.         if (opt_shorttest)
120.             break;
121.         else
122.             continue;
123.     }
124.
125.     no_packet_count = 0;
126.     memset(no_packet, 0, nb_input_files);
127.
128.     if (do_pkt_dump){
129.         av_pkt_dump_log2(NULL, AV_LOG_DEBUG, &pkt, do_hex_dump,
130.             is->streams[pkt.stream_index]);
131.     }
132.     /* the following test is needed in case new streams appear
133.        dynamically in stream : we ignore them */
134.     //如果在输入文件中遇到一个忽然冒出的流,那么我们不鸟它
135.     if (pkt.stream_index >= input_files[file_index].nb_streams)
136.         goto discard_packet;
137.
138.     //取得当前获得的帧对应的输入流
139.     ist_index = input_files[file_index].ist_index + pkt.stream_index;
140.     ist = &input_streams[ist_index];
141.     if (ist->discard)
142.         goto discard_packet;
143.
144.     //重新鼓捣一下帧的时间戳
145.     if (pkt.dts != AV_NOPTS_VALUE)
146.         pkt.dts += av_rescale_q(input_files[ist->file_index].ts_offset,
147.             AV_TIME_BASE_Q, ist->st->time_base);
148.     if (pkt.pts != AV_NOPTS_VALUE)
149.         pkt.pts += av_rescale_q(input_files[ist->file_index].ts_offset,
150.             AV_TIME_BASE_Q, ist->st->time_base);
151.
152.     if (pkt.pts != AV_NOPTS_VALUE)
153.         pkt.pts *= ist->ts_scale;
154.     if (pkt.dts != AV_NOPTS_VALUE)
155.         pkt.dts *= ist->ts_scale;
156.
157.     if (pkt.dts != AV_NOPTS_VALUE && ist->next_pts != AV_NOPTS_VALUE
158.         && (is->iformat->flags & AVFMT_TS_DISCONT))
159.     {
160.         int64_t pkt_dts = av_rescale_q(pkt.dts, ist->st->time_base,
161.             AV_TIME_BASE_Q);
162.         int64_t delta = pkt_dts - ist->next_pts;
163.         if ((delta < -1LL * dts_delta_threshold * AV_TIME_BASE
164.             || (delta > 1LL * dts_delta_threshold * AV_TIME_BASE
165.                 && ist->st->codec->codec_type
166.                     != AVMEDIA_TYPE_SUBTITLE)
167.             || pkt_dts + 1 < ist->pts) && !copy_ts)
168.         {
169.             input_files[ist->file_index].ts_offset -= delta;
170.             av_log(NULL, AV_LOG_DEBUG,
171.                 "timestamp discontinuity %"PRIu64", new offset= %"PRIu64"\n",
172.                 delta, input_files[ist->file_index].ts_offset);
173.             pkt.dts -= av_rescale_q(delta, AV_TIME_BASE_Q, ist->st->time_base);
174.             if (pkt.pts != AV_NOPTS_VALUE)
175.                 pkt.pts -= av_rescale_q(delta, AV_TIME_BASE_Q, ist->st->time_base);
176.         }
177.     }
178.
179.     //把这一帧转换并写入到输出文件中
180.     if (output_packet(ist, output_streams, nb_output_streams, &pkt) < 0){
181.         av_log(NULL, AV_LOG_ERROR,
182.             "Error while decoding stream #d:%d\n",
183.             ist->file_index, ist->st->index);
184.         if (exit_on_error)
185.             exit_program(1);
186.         av_free_packet(&pkt);
187.         continue;
188.     }
189.
190.     discard_packet:
191.         av_free_packet(&pkt);
192.
193.     /* dump report by using the output first video and audio streams */
194.     print_report(output_files, output_streams, nb_output_streams, 0,
195.         timer_start, cur_time);
196. }
197.
198. //文件处理完了,把缓冲中剩余的数据写到输出文件中
199. for (i = 0; i < nb_input_streams; i++){
200.     ist = &input_streams[i];

```

```

210.         if (ist->decoding_needed){
211.             output_packet(ist, output_streams, nb_output_streams, NULL);
212.         }
213.     }
214.     flush_encoders(output_streams, nb_output_streams);
215.
216.     term_exit();
217.
218.     //为输出文件写文件尾(有的不需要).
219.     for (i = 0; i < nb_output_files; i++){
220.         os = output_files[i].ctx;
221.         av_write_trailer(os);
222.     }
223.
224.     /* dump report by using the first video and audio streams */
225.     print_report(output_files, output_streams, nb_output_streams, 1,
226.         timer_start, av_gettime());
227.
228.     //关闭所有的编码器
229.     for (i = 0; i < nb_output_streams; i++){
230.         ost = &output_streams[i];
231.         if (ost->encoding_needed){
232.             av_freep(&ost->st->codec->stats_in);
233.             avcodec_close(ost->st->codec);
234.         }
235. #if CONFIG_AVFILTER
236.         avfilter_graph_free(&ost->graph);
237. #endif
238.     }
239.
240.     //关闭所有的解码器
241.     for (i = 0; i < nb_input_streams; i++){
242.         ist = &input_streams[i];
243.         if (ist->decoding_needed){
244.             avcodec_close(ist->st->codec);
245.         }
246.     }
247.
248.     /* finished ! */
249.     ret = 0;
250.
251.     fail: av_freep(&bit_buffer);
252.     av_freep(&no_packet);
253.
254.     if (output_streams) {
255.         for (i = 0; i < nb_output_streams; i++) {
256.             ost = &output_streams[i];
257.             if (ost) {
258.                 if (ost->stream_copy)
259.                     av_freep(&ost->st->codec->extradata);
260.                 if (ost->logfile){
261.                     fclose(ost->logfile);
262.                     ost->logfile = NULL;
263.                 }
264.                 av_fifo_free(ost->fifo); /* works even if fifo is not
265.                     initialized but set to zero */
266.                 av_freep(&ost->st->codec->subtitle_header);
267.                 av_free(ost->resample_frame.data[0]);
268.                 av_free(ost->forced_kf_pts);
269.                 if (ost->video_resample)
270.                     sws_freeContext(ost->img_resample_ctx);
271.                 swr_free(&ost->swr);
272.                 av_dict_free(&ost->opts);
273.             }
274.         }
275.     }
276.     return ret;
277. }

```

原文地址：http://blog.csdn.net/niu_gao/article/details/7175421

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

个人分类：[FFMPEG](#)

所属专栏：[开源多媒体项目源代码分析](#) [FFmpeg](#)