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
int main(int argc, char **argv)
{
    OptionsContext o = { 0 };
    int64_t ti;

    //与命令行分析有关的结构的初始化,下面不再罗嗦
    reset_options(&o, 0);

    //设置日志级别
    av_log_set_flags(AV_LOG_SKIP_REPEATED);
    parse_loglevel(argc, argv, options);

    if (argc > 1 && !strcmp(argv[1], "-d"))  {
        run_as_daemon = 1;
        av_log_set_callback(log_callback_null);
        argc--;
        argv++;
    }

    //注册组件们
    avcodec_register_all();
#if CONFIG_AVDEVICE
    avdevice_register_all();
#endif
#if CONFIG_AVFILTER
    avfilter_register_all();
#endif
    av_register_all();
    //初始化网络,windows下需要
    avformat_network_init();

    show_banner();

    term_init();

    //分析命令行输入的参数们
    parse_options(&o, argc, argv, options, opt_output_file);

    //文件的转换就在此函数中发生
    if (transcode(output_files, nb_output_files, input_files, nb_input_files)< 0)
        exit_program(1);

    exit_program(0);
    return 0;
}
```

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
static int transcode(
        OutputFile *output_files,//输出文件数组
        int nb_output_files,//输出文件的数量
        InputFile *input_files,//输入文件数组
        int nb_input_files)//输入文件的数量
{
    int ret, i;
    AVFormatContext *is, *os;
    OutputStream *ost;
    InputStream *ist;
    uint8_t *no_packet;
    int no_packet_count = 0;
    int64_t timer_start;
    int key;

    if (!(no_packet = av_mallocz(nb_input_files)))
        exit_program(1);

    //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
    ret = transcode_init(output_files, nb_output_files, input_files,nb_input_files);
    if (ret < 0)
        goto fail;

    if (!using_stdin){
        av_log(NULL, AV_LOG_INFO, "Press [q] to stop, [?] for help\n");
    }
```

```
    timer_start = av_gettime();

    //循环,直到收到系统信号才退出
    for (; received_sigterm == 0;)
    {
        int file_index, ist_index;
        AVPacket pkt;
        int64_t ipts_min;
        double opts_min;
        int64_t cur_time = av_gettime();

        ipts_min = INT64_MAX;
        opts_min = 1e100;
        /* if 'q' pressed, exits */
        if (!using_stdin)
        {
            //先查看用户按下了什么键,跟据键做出相应的反应
            static int64_t last_time;
            if (received_nb_signals)
                break;
            /* read_key() returns 0 on EOF */
            if (cur_time - last_time >= 100000 && !run_as_daemon){
                key = read_key();
                last_time = cur_time;
            }else{
<span>            </span>.............................
        }

        /* select the stream that we must read now by looking at the
         smallest output pts */
        //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
        file_index = -1;
        for (i = 0; i < nb_output_streams; i++){
            OutputFile *of;
            int64_t ipts;
            double opts;
            ost = &output_streams[i];//循环每一个输出流
            of = &output_files[ost->file_index];//输出流对应的输出文件
            os = output_files[ost->file_index].ctx;//输出流对应的FormatContext
            ist = &input_streams[ost->source_index];//输出流对应的输入流

            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)
                    no_packet[ist->file_index]|| //对应的输入流这个时间内没有数据?
                    (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)
                continue;//是的,符合上面某一条,那么再看下一个输出流吧

            //判断当前输入流所在的文件是否可以使用(我也不很明白)
            opts = ost->st->pts.val * av_q2d(ost->st->time_base);
            ipts = ist->pts;
            if (!input_files[ist->file_index].eof_reached)   {
                if (ipts < ipts_min){
                    //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
                    //pts最小的输入流,及输入文件的序号
                    ipts_min = ipts;
                    if (input_sync)
                        file_index = ist->file_index;
                }
                if (opts < opts_min){
                    opts_min = opts;
                    if (!input_sync)
                        file_index = ist->file_index;
                }
            }

            //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,
            //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?
            if (ost->frame_number >= ost->max_frames){
                int j;
                for (j = 0; j < of->ctx->nb_streams; j++)
                    output_streams[of->ost_index + j].is_past_recording_time =   1;
                continue;
            }
        }
        /* if none, if is finished */
        if (file_index < 0)  {
            //如果没有找到合适的输入文件
            if (no_packet_count){
                //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
                no_packet_count = 0;
                memset(no_packet, 0, nb_input_files);
                usleep(10000);
                continue;
            }
            //全部转换完成了,跳出大循环
            break;
        }

        //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
        is = input_files[file_index].ctx;
        ret = av_read_frame(is, &pkt);
        if (ret == AVERROR(EAGAIN)) {
```

```
        if (ret == AVERROR(EAGAIN)) {
            //此时发生了暂时没数据的情况
            no_packet[file_index] = 1;
            no_packet_count++;
            continue;
        }

        //下文判断是否有输入文件到最后了
        if (ret < 0){
            input_files[file_index].eof_reached = 1;
            if (opt_shortest)
                break;
            else
                continue;
        }

        no_packet_count = 0;
        memset(no_packet, 0, nb_input_files);

        if (do_pkt_dump){
            av_pkt_dump_log2(NULL, AV_LOG_DEBUG, &pkt, do_hex_dump,
                    is->streams[pkt.stream_index]);
        }
        /* the following test is needed in case new streams appear
         dynamically in stream : we ignore them */
        //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
        if (pkt.stream_index >= input_files[file_index].nb_streams)
            goto discard_packet;

        //取得当前获得的帧对应的输入流
        ist_index = input_files[file_index].ist_index + pkt.stream_index;
        ist = &input_streams[ist_index];
        if (ist->discard)
            goto discard_packet;

        //重新鼓捣一下帧的时间戳
        if (pkt.dts != AV_NOPTS_VALUE)
            pkt.dts += av_rescale_q(input_files[ist->file_index].ts_offset,
                    AV_TIME_BASE_Q, ist->st->time_base);
        if (pkt.pts != AV_NOPTS_VALUE)
            pkt.pts += av_rescale_q(input_files[ist->file_index].ts_offset,
                    AV_TIME_BASE_Q, ist->st->time_base);

        if (pkt.pts != AV_NOPTS_VALUE)
            pkt.pts *= ist->ts_scale;
        if (pkt.dts != AV_NOPTS_VALUE)
            pkt.dts *= ist->ts_scale;

        if (pkt.dts != AV_NOPTS_VALUE && ist->next_pts != AV_NOPTS_VALUE
                && (is->iformat->flags & AVFMT_TS_DISCONT))
        {
            int64_t pkt_dts = av_rescale_q(pkt.dts, ist->st->time_base,
                    AV_TIME_BASE_Q);
            int64_t delta = pkt_dts - ist->next_pts;
            if ((delta < -1LL * dts_delta_threshold * AV_TIME_BASE
                    || (delta > 1LL * dts_delta_threshold * AV_TIME_BASE
                            && ist->st->codec->codec_type
                                    != AVMEDIA_TYPE_SUBTITLE)
                    || pkt_dts + 1 < ist->pts) && !copy_ts)
            {
                input_files[ist->file_index].ts_offset -= delta;
                av_log( NULL,   AV_LOG_DEBUG,
                        "timestamp discontinuity %"PRId64", new offset= %"PRId64"\n",
                        delta, input_files[ist->file_index].ts_offset);
                pkt.dts -= av_rescale_q(delta, AV_TIME_BASE_Q,  ist->st->time_base);
                if (pkt.pts != AV_NOPTS_VALUE)
                    pkt.pts -= av_rescale_q(delta, AV_TIME_BASE_Q,  ist->st->time_base);
            }
        }

        //把这一帧转换并写入到输出文件中
        if (output_packet(ist, output_streams, nb_output_streams, &pkt) < 0){
            av_log(NULL, AV_LOG_ERROR,
                    "Error while decoding stream #%d:%d\n",
                    ist->file_index, ist->st->index);
            if (exit_on_error)
                exit_program(1);
            av_free_packet(&pkt);
            continue;
        }

discard_packet:
        av_free_packet(&pkt);

        /* dump report by using the output first video and audio streams */
        print_report(output_files, output_streams, nb_output_streams, 0,
                timer_start, cur_time);
    }

    //文件处理完了,把缓冲中剩余的数据写到输出文件中
    for (i = 0; i < nb_input_streams; i++){
        ist = &input_streams[i];
```

```
        if (ist->decoding_needed){
            output_packet(ist, output_streams, nb_output_streams, NULL);
        }
    }
    flush_encoders(output_streams, nb_output_streams);

    term_exit();

    //为输出文件写文件尾(有的不需要).
    for (i = 0; i < nb_output_files; i++){
        os = output_files[i].ctx;
        av_write_trailer(os);
    }

    /* dump report by using the first video and audio streams */
    print_report(output_files, output_streams, nb_output_streams, 1,
            timer_start, av_gettime());

    //关闭所有的编码器
    for (i = 0; i < nb_output_streams; i++){
        ost = &output_streams[i];
        if (ost->encoding_needed){
            av_freep(&ost->st->codec->stats_in);
            avcodec_close(ost->st->codec);
        }
#if CONFIG_AVFILTER
        avfilter_graph_free(&ost->graph);
#endif
    }

    //关闭所有的解码器
    for (i = 0; i < nb_input_streams; i++){
        ist = &input_streams[i];
        if (ist->decoding_needed){
            avcodec_close(ist->st->codec);
        }
    }

    /* finished ! */
    ret = 0;

    fail: av_freep(&bit_buffer);
    av_freep(&no_packet);

    if (output_streams) {
        for (i = 0; i < nb_output_streams; i++)  {
            ost = &output_streams[i];
            if (ost)    {
                if (ost->stream_copy)
                    av_freep(&ost->st->codec->extradata);
                if (ost->logfile){
                    fclose(ost->logfile);
                    ost->logfile = NULL;
                }
                av_fifo_free(ost->fifo); /* works even if fifo is not
                 initialized but set to zero */
                av_freep(&ost->st->codec->subtitle_header);
                av_free(ost->resample_frame.data[0]);
                av_free(ost->forced_kf_pts);
                if (ost->video_resample)
                    sws_freeContext(ost->img_resample_ctx);
                swr_free(&ost->swr);
                av_dict_free(&ost->opts);
            }
        }
    }
    return ret;
}
```

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