转 ffmpeg源码分析：transcode_init()函数

2013年09月20日 17:36:06　阅读数：6761

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
//为转换过程做准备
static int transcode_init(OutputFile *output_files,
        int nb_output_files,
        InputFile *input_files,
        int nb_input_files)
{
    int ret = 0, i, j, k;
    AVFormatContext *oc;
    AVCodecContext *codec, *icodec;
    OutputStream *ost;
    InputStream *ist;
    char error[1024];
    int want_sdp = 1;

    /* init framerate emulation */
    //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)
    for (i = 0; i < nb_input_files; i++)
    {
        InputFile *ifile = &input_files[i];
        //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),
        //则为这个文件中所有流记录下开始时间
        if (ifile->rate_emu)
            for (j = 0; j < ifile->nb_streams; j++)
                input_streams[j + ifile->ist_index].start = av_gettime();
    }

    /* output stream init */
    for (i = 0; i < nb_output_files; i++)
    {
        //什么也没做,只是做了个判断而已
        oc = output_files[i].ctx;
        if (!oc->nb_streams && !(oc->oformat->flags & AVFMT_NOSTREAMS))
        {
            av_dump_format(oc, i, oc->filename, 1);
            av_log(NULL, AV_LOG_ERROR,
                   "Output file #%d does not contain any stream\n", i);
            return AVERROR(EINVAL);
        }
    }

    //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
    for (i = 0; i < nb_output_streams; i++)
    {
        //轮循所有的输出流
        ost = &output_streams[i];
        //输出流对应的FormatContext
        oc = output_files[ost->file_index].ctx;
        //取得输出流对应的输入流
        ist = &input_streams[ost->source_index];

        //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
        if (ost->attachment_filename)
            continue;

        codec = ost->st->codec;//输出流的编解码器结构
        icodec = ist->st->codec;//输入流的编解码器结构

        //先把能复制的复制一下
        ost->st->disposition = ist->st->disposition;
        codec->bits_per_raw_sample = icodec->bits_per_raw_sample;
        codec->chroma_sample_location = icodec->chroma_sample_location;

        //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
        //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
        if (ost->stream_copy)
        {
            //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
            //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
            uint64_t extra_size = (uint64_t) icodec->extradata_size
                    + FF_INPUT_BUFFER_PADDING_SIZE;

            if (extra_size > INT_MAX)    {
                return AVERROR(EINVAL);
            }

            /* if stream_copy is selected, no need to decode or encode */
            codec->codec_id = icodec->codec_id;
            codec->codec_type = icodec->codec_type;
```

```
            if (!codec->codec_tag){
                if (!oc->oformat->codec_tag
                    ||av_codec_get_id(oc->oformat->codec_tag,icodec->codec_tag) == codec->codec_id
                    ||av_codec_get_tag(oc->oformat->codec_tag,icodec->codec_id) <= 0)
                    codec->codec_tag = icodec->codec_tag;
            }

            codec->bit_rate = icodec->bit_rate;
            codec->rc_max_rate = icodec->rc_max_rate;
            codec->rc_buffer_size = icodec->rc_buffer_size;
            codec->extradata = av_mallocz(extra_size);
            if (!codec->extradata){
                return AVERROR(ENOMEM);
            }
            memcpy(codec->extradata, icodec->extradata, icodec->extradata_size);
            codec->extradata_size = icodec->extradata_size;

            //重新鼓捣一下time base(这家伙就是帧率)
            codec->time_base = ist->st->time_base;
            //如果输出文件是avi,做一点特殊处理
            if (!strcmp(oc->oformat->name, "avi"))    {
                if (copy_tb < 0
                        && av_q2d(icodec->time_base) * icodec->ticks_per_frame    >
                            2 * av_q2d(ist->st->time_base)
                        && av_q2d(ist->st->time_base) < 1.0 / 500
                        || copy_tb == 0)
                {
                    codec->time_base = icodec->time_base;
                    codec->time_base.num *= icodec->ticks_per_frame;
                    codec->time_base.den *= 2;
                }
            }
            else if (!(oc->oformat->flags & AVFMT_VARIABLE_FPS))
            {
                if (copy_tb < 0
                        && av_q2d(icodec->time_base) * icodec->ticks_per_frame
                                > av_q2d(ist->st->time_base)
                        && av_q2d(ist->st->time_base) < 1.0 / 500
                        || copy_tb == 0)
                {
                    codec->time_base = icodec->time_base;
                    codec->time_base.num *= icodec->ticks_per_frame;
                }
            }

            //再修正一下帧率
            av_reduce(&codec->time_base.num, &codec->time_base.den,
                    codec->time_base.num, codec->time_base.den, INT_MAX);

            //单独复制各不同媒体自己的编码参数
            switch (codec->codec_type)
            {
            case AVMEDIA_TYPE_AUDIO:
                //音频的
                if (audio_volume != 256){
                    av_log( NULL,AV_LOG_FATAL,
                            "-acodec copy and -vol are incompatible (frames are not decoded)\n");
                    exit_program(1);
                }
                codec->channel_layout = icodec->channel_layout;
                codec->sample_rate = icodec->sample_rate;
                codec->channels = icodec->channels;
                codec->frame_size = icodec->frame_size;
                codec->audio_service_type = icodec->audio_service_type;
                codec->block_align = icodec->block_align;
                break;
            case AVMEDIA_TYPE_VIDEO:
                //视频的
                codec->pix_fmt = icodec->pix_fmt;
                codec->width = icodec->width;
                codec->height = icodec->height;
                codec->has_b_frames = icodec->has_b_frames;
                if (!codec->sample_aspect_ratio.num){
                    codec->sample_aspect_ratio = ost->st->sample_aspect_ratio =
                            ist->st->sample_aspect_ratio.num ?ist->st->sample_aspect_ratio :
                                    ist->st->codec->sample_aspect_ratio.num ?ist->st->codec->sample_aspect_ratio :(AVRational){0, 1}
;
                }
                ost->st->avg_frame_rate = ist->st->avg_frame_rate;
                break;
            case AVMEDIA_TYPE_SUBTITLE:
                //字幕的
                codec->width  = icodec->width;
                codec->height = icodec->height;
                break;
            case AVMEDIA_TYPE_DATA:
            case AVMEDIA_TYPE_ATTACHMENT:
                //??的
                break;
            default:
                abort();
```

```
            }
    }
    else
    {
        //如果不是复制,就麻烦多了

        //获取编码器
        if (!ost->enc)
            ost->enc = avcodec_find_encoder(ost->st->codec->codec_id);

        //因为需要转换,所以既需解码又需编码
        ist->decoding_needed = 1;
        ost->encoding_needed = 1;

        switch(codec->codec_type)
        {
        case AVMEDIA_TYPE_AUDIO:
            //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
            ost->fifo = av_fifo_alloc(1024);//音频数据所在的缓冲
            if (!ost->fifo)  {
                return AVERROR(ENOMEM);
            }

            //采样率
            if (!codec->sample_rate)
                codec->sample_rate = icodec->sample_rate;
            choose_sample_rate(ost->st, ost->enc);
            codec->time_base = (AVRational){1, codec->sample_rate};

            //样点格式
            if (codec->sample_fmt == AV_SAMPLE_FMT_NONE)
                codec->sample_fmt = icodec->sample_fmt;
            choose_sample_fmt(ost->st, ost->enc);

            //声道
            if (ost->audio_channels_mapped)  {
                /* the requested output channel is set to the number of
                 * -map_channel only if no -ac are specified */
                if (!codec->channels)    {
                    codec->channels = ost->audio_channels_mapped;
                    codec->channel_layout = av_get_default_channel_layout(codec->channels);
                    if (!codec->channel_layout)  {
                        av_log(NULL, AV_LOG_FATAL, "Unable to find an appropriate channel layout for requested number of channel\n");

                        exit_program(1);
                    }
                }
                /* fill unused channel mapping with -1 (which means a muted
                 * channel in case the number of output channels is bigger
                 * than the number of mapped channel) */
                for (j = ost->audio_channels_mapped; j < FF_ARRAY_ELEMS(ost->audio_channels_map); j++)
                <span>  </span>ost->audio_channels_map[j] = -1;
            }else if (!codec->channels){
                codec->channels = icodec->channels;
                codec->channel_layout = icodec->channel_layout;
            }
            if (av_get_channel_layout_nb_channels(codec->channel_layout) != codec->channels)
                codec->channel_layout = 0;

            //是否需要重采样
            ost->audio_resample = codec->sample_rate != icodec->sample_rate || audio_sync_method > 1;
            ost->audio_resample |= codec->sample_fmt != icodec->sample_fmt ||
                    codec->channel_layout != icodec->channel_layout;
            icodec->request_channels = codec->channels;
            ost->resample_sample_fmt = icodec->sample_fmt;
            ost->resample_sample_rate = icodec->sample_rate;
            ost->resample_channels = icodec->channels;
            break;
        case AVMEDIA_TYPE_VIDEO:
            //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
            if (codec->pix_fmt == PIX_FMT_NONE)
                codec->pix_fmt = icodec->pix_fmt;
            choose_pixel_fmt(ost->st, ost->enc);
            if (ost->st->codec->pix_fmt == PIX_FMT_NONE){
                av_log(NULL, AV_LOG_FATAL, "Video pixel format is unknown, stream cannot be encoded\n");
                exit_program(1);
            }

            //宽高
            if (!codec->width || !codec->height){
                codec->width = icodec->width;
                codec->height = icodec->height;
            }

            //视频是否需要重采样
            ost->video_resample = codec->width != icodec->width ||
                    codec->height != icodec->height ||
                    codec->pix_fmt != icodec->pix_fmt;
            if (ost->video_resample){
                codec->bits_per_raw_sample= frame_bits_per_raw_sample;
            }
```

```
            ost->resample_height = icodec->height;
            ost->resample_width = icodec->width;
            ost->resample_pix_fmt = icodec->pix_fmt;

            //计算帧率
            if (!ost->frame_rate.num)
                ost->frame_rate = ist->st->r_frame_rate.num ?
                        ist->st->r_frame_rate : (AVRational){25,1};
            if (ost->enc && ost->enc->supported_framerates && !ost->force_fps)  {
                int idx = av_find_nearest_q_idx(ost->frame_rate,ost->enc->supported_framerates);
                ost->frame_rate = ost->enc->supported_framerates[idx];
            }
            codec->time_base = (AVRational)  {ost->frame_rate.den, ost->frame_rate.num};
            if( av_q2d(codec->time_base) < 0.001 &&
                    video_sync_method &&
                    (video_sync_method==1 ||
                        (video_sync_method<0 &&  !
                            (oc->oformat->flags & AVFMT_VARIABLE_FPS))))
            {
                av_log(oc, AV_LOG_WARNING, "Frame rate very high for a muxer not effciciently supporting it.\n"
                        "Please consider specifiying a lower framerate, a different muxer or -vsync 2\n");
            }
        <span>  </span>for (j = 0; j < ost->forced_kf_count; j++)
                ost->forced_kf_pts[j] = av_rescale_q(ost->forced_kf_pts[j],
                        AV_TIME_BASE_Q, codec->time_base);
                break;
            case AVMEDIA_TYPE_SUBTITLE:
                break;
            default:
                abort();
                break;
            }
            /* two pass mode */
            if (codec->codec_id != CODEC_ID_H264 &&
                    (codec->flags & (CODEC_FLAG_PASS1 | CODEC_FLAG_PASS2)))
            {
                char logfilename[1024];
                FILE *f;

                snprintf(logfilename, sizeof(logfilename), "%s-%d.log",
                        pass_logfilename_prefix ? pass_logfilename_prefix : DEFAULT_PASS_LOGFILENAME_PREFIX,
                        i);
                if (codec->flags & CODEC_FLAG_PASS2){
                    char *logbuffer;
                    size_t logbuffer_size;
                    if (cmdutils_read_file(logfilename, &logbuffer, &logbuffer_size) < 0){
                        av_log(NULL, AV_LOG_FATAL,
                                "Error reading log file '%s' for pass-2 encoding\n",
                                logfilename);
                        exit_program(1);
                    }
                    codec->stats_in = logbuffer;
                }
                if (codec->flags & CODEC_FLAG_PASS1){
                    f = fopen(logfilename, "wb");
                    if (!f) {
                        av_log(NULL, AV_LOG_FATAL, "Cannot write log file '%s' for pass-1 encoding: %s\n",
                                logfilename, strerror(errno));
                        exit_program(1);
                    }
                    ost->logfile = f;
                }
            }
        }
        if (codec->codec_type == AVMEDIA_TYPE_VIDEO){
            /* maximum video buffer size is 6-bytes per pixel, plus DPX header size (1664)*/
            //计算编码输出缓冲的大小,计算一个最大值
            int size = codec->width * codec->height;
            bit_buffer_size = FFMAX(bit_buffer_size, 7 * size + 10000);
        }
    }

    //分配编码后数据所在的缓冲
    if (!bit_buffer)
        bit_buffer = av_malloc(bit_buffer_size);
    if (!bit_buffer){
        av_log(NULL, AV_LOG_ERROR,
                "Cannot allocate %d bytes output buffer\n",
                bit_buffer_size);
        return AVERROR(ENOMEM);
    }

    //轮循所有输出流,打开每个输出流的编码器
    for (i = 0; i < nb_output_streams; i++)
    {
        ost = &output_streams[i];
        if (ost->encoding_needed){
            //当然,只有在需要编码时才打开编码器
            AVCodec *codec = ost->enc;
            AVCodecContext *dec = input_streams[ost->source_index].st->codec;
```

```
            if (!codec) {
                snprintf(error, sizeof(error),
                        "Encoder (codec %s) not found for output stream #%d:%d",
                        avcodec_get_name(ost->st->codec->codec_id),
                        ost->file_index, ost->index);
                ret = AVERROR(EINVAL);
                goto dump_format;
            }
            if (dec->subtitle_header){
                ost->st->codec->subtitle_header = av_malloc(dec->subtitle_header_size);
                if (!ost->st->codec->subtitle_header){
                    ret = AVERROR(ENOMEM);
                    goto dump_format;
                }
                memcpy(ost->st->codec->subtitle_header,
                        dec->subtitle_header,dec->subtitle_header_size);
                ost->st->codec->subtitle_header_size = dec->subtitle_header_size;
            }
            //打开啦
            if (avcodec_open2(ost->st->codec, codec, &ost->opts) < 0)   {
                snprintf(error, sizeof(error),
                        "Error while opening encoder for output stream #%d:%d - maybe incorrect parameters such as bit_rate, rate, wi
h or height",
                        ost->file_index, ost->index);
                ret = AVERROR(EINVAL);
                goto dump_format;
            }
            assert_codec_experimental(ost->st->codec, 1);
            assert_avoptions(ost->opts);
            if (ost->st->codec->bit_rate && ost->st->codec->bit_rate < 1000)
                av_log(NULL, AV_LOG_WARNING,
                        "The bitrate parameter is set too low."
                        " It takes bits/s as argument, not kbits/s\n");
            extra_size += ost->st->codec->extradata_size;

            if (ost->st->codec->me_threshold)
                input_streams[ost->source_index].st->codec->debug |= FF_DEBUG_MV;
        }
    }

    //初始化所有的输入流(主要做的就是在需要时打开解码器)
    for (i = 0; i < nb_input_streams; i++)
        if ((ret = init_input_stream(i, output_streams, nb_output_streams,
                error, sizeof(error))) < 0)
            goto dump_format;

    /* discard unused programs */
    for (i = 0; i < nb_input_files; i++){
        InputFile *ifile = &input_files[i];
        for (j = 0; j < ifile->ctx->nb_programs; j++){
            AVProgram *p = ifile->ctx->programs[j];
            int discard = AVDISCARD_ALL;

            for (k = 0; k < p->nb_stream_indexes; k++){
                if (!input_streams[ifile->ist_index + p->stream_index[k]].discard){
                    discard = AVDISCARD_DEFAULT;
                    break;
                }
            }
            p->discard = discard;
        }
    }

    //打开所有输出文件，写入媒体文件头
    for (i = 0; i < nb_output_files; i++){
        oc = output_files[i].ctx;
        oc->interrupt_callback = int_cb;
        if (avformat_write_header(oc, &output_files[i].opts) < 0){
            snprintf(error, sizeof(error),
                    "Could not write header for output file #%d (incorrect codec parameters ?)",
                    i);
            ret = AVERROR(EINVAL);
            goto dump_format;
        }
//        assert_avoptions(output_files[i].opts);
        if (strcmp(oc->oformat->name, "rtp")){
            want_sdp = 0;
        }
    }

    return 0;
}
```

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

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