ffmpeg 音频转码

作者 WillToSky (/u/a042ebae0eb2) + 关注
2016.09.24 20:18 字数 578 阅读 138 评论 0 喜欢 1
(/u/a042ebae0eb2)

转码一般流程

1. 获取音频数据(AAC,MP3等)
2. 解码（获取音频原始采样数据pcm）
3. 编码（对pcm进行编码）

使用ffmpeg的函数表示的大概流程

```
//初始化输入
avformat_open_input()               -- 打开对应音频文件
avformat_find_stream_info()      -- 从输入文件中获取到流的相关信息，例如：文件中流的数量

//初始化解码器
avcodec_find_decoder()              -- 根据ffmpeg提供的解码器id，找到对应的解码器
avcodec_open2()                      -- 打开解码器

//初始化输出
avformat_alloc_context()          -- 创建输出信息上下文
avcodec_find_encoder()             -- 找到编码器
avcodec_alloc_context3()             -- 根据编码器，初始化编码上下文
avcodec_open2()                     -- 打开编码器

//初始化一个FIFO(先进先出)
av_audio_fifo_alloc()              -- 初始化一个先进先出的缓存，用了存储解码后的pcm数据

//初始化音频重采用器
swr_alloc_set_opts()             -- 设置转化器参数
swr_init()                           -- 初始化转换器

//开始音频转换

while(finished) {
// 解码
av_read_frame()                      -- 读取要进行转码的数据
avcodec_decode_audio4()            -- 进行解码
av_samples_alloc()                 -- 创建样本空间
swr_convert()                          -- 数据重采样
av_audio_fifo_write()             -- 将数据存储到fifo缓存中

// 编码
av_audio_fifo_read()             -- 从fifo缓存中读取pcm数据
avcodec_encode_audio2()            -- 将数据进行编码

}
```

知识点

AVSampleFormat(样本数据格式)

在说明该格式之前，先说一个样本数据存储的2种方式**packet**和**planar**。

packed - 将所有声道的数据，交替的存储成一维数组

planar - 每一个声道单独存放,一个二维数组表示，每一行代表一个声道。

例如：

现在有一段音频采用，左声道用L表示，右声道用R表示

`LRLRLR.....LR` 表示使用了**packed**方式存储

`LLLLL..L` 和 `RRRRR..R` 表示使用了**planar**方式存储

在ffmpeg中，采样样本的数据格式有如下几种类型

```
enum AVSampleFormat {
    AV_SAMPLE_FMT_NONE = -1,
    AV_SAMPLE_FMT_U8,          ///< unsigned 8 bits
    AV_SAMPLE_FMT_S16,         ///< signed 16 bits
    AV_SAMPLE_FMT_S32,         ///< signed 32 bits
    AV_SAMPLE_FMT_FLT,         ///< float
    AV_SAMPLE_FMT_DBL,         ///< double

    AV_SAMPLE_FMT_U8P,         ///< unsigned 8 bits, planar
    AV_SAMPLE_FMT_S16P,        ///< signed 16 bits, planar
    AV_SAMPLE_FMT_S32P,        ///< signed 32 bits, planar
    AV_SAMPLE_FMT_FLTP,        ///< float, planar
    AV_SAMPLE_FMT_DBLP,        ///< double, planar

    AV_SAMPLE_FMT_NB           ///< Number of sample formats. DO NOT USE if linking dynamica
};
```

AVCodecContext的属性

channel_layout

`channel_layout` 表示声道布局，代表左右声道形成的立体声，还是单声道。

在ffmpeg中，支持的声道布局有

```
/**
 * @defgroup channel_masks Audio channel masks
 *
 * A channel layout is a 64-bits integer with a bit set for every channel.
 * The number of bits set must be equal to the number of channels.
 * The value 0 means that the channel layout is not known.
 * @note this data structure is not powerful enough to handle channels
 * combinations that have the same channel multiple times, such as
 * dual-mono.
 *
 * @{
 */
#define AV_CH_FRONT_LEFT             0x00000001
#define AV_CH_FRONT_RIGHT            0x00000002
#define AV_CH_FRONT_CENTER           0x00000004
#define AV_CH_LOW_FREQUENCY          0x00000008
#define AV_CH_BACK_LEFT              0x00000010
#define AV_CH_BACK_RIGHT             0x00000020
#define AV_CH_FRONT_LEFT_OF_CENTER   0x00000040
#define AV_CH_FRONT_RIGHT_OF_CENTER  0x00000080
#define AV_CH_BACK_CENTER            0x00000100
#define AV_CH_SIDE_LEFT              0x00000200
#define AV_CH_SIDE_RIGHT             0x00000400
#define AV_CH_TOP_CENTER             0x00000800
#define AV_CH_TOP_FRONT_LEFT         0x00001000
#define AV_CH_TOP_FRONT_CENTER       0x00002000
#define AV_CH_TOP_FRONT_RIGHT        0x00004000
#define AV_CH_TOP_BACK_LEFT          0x00008000
#define AV_CH_TOP_BACK_CENTER        0x00010000
#define AV_CH_TOP_BACK_RIGHT         0x00020000
#define AV_CH_STEREO_LEFT            0x20000000  ///< Stereo downmix.
#define AV_CH_STEREO_RIGHT           0x40000000  ///< See AV_CH_STEREO_LEFT.
#define AV_CH_WIDE_LEFT              0x0000000080000000ULL
#define AV_CH_WIDE_RIGHT             0x0000000100000000ULL
#define AV_CH_SURROUND_DIRECT_LEFT   0x0000000200000000ULL
#define AV_CH_SURROUND_DIRECT_RIGHT  0x0000000400000000ULL
#define AV_CH_LOW_FREQUENCY_2        0x0000000800000000ULL

/** Channel mask value used for AVCodecContext.request_channel_layout
    to indicate that the user requests the channel order of the decoder output
    to be the native codec channel order. */
#define AV_CH_LAYOUT_NATIVE          0x8000000000000000ULL

/**
 * @}
 * @defgroup channel_mask_c Audio channel layouts
 * @{
 * */
#define AV_CH_LAYOUT_MONO              (AV_CH_FRONT_CENTER)  // 单声道
#define AV_CH_LAYOUT_STEREO            (AV_CH_FRONT_LEFT|AV_CH_FRONT_RIGHT) // 立体声道
#define AV_CH_LAYOUT_2POINT1           (AV_CH_LAYOUT_STEREO|AV_CH_LOW_FREQUENCY)
#define AV_CH_LAYOUT_2_1               (AV_CH_LAYOUT_STEREO|AV_CH_BACK_CENTER)
#define AV_CH_LAYOUT_SURROUND          (AV_CH_LAYOUT_STEREO|AV_CH_FRONT_CENTER)
#define AV_CH_LAYOUT_3POINT1           (AV_CH_LAYOUT_SURROUND|AV_CH_LOW_FREQUENCY)
#define AV_CH_LAYOUT_4POINT0           (AV_CH_LAYOUT_SURROUND|AV_CH_BACK_CENTER)
#define AV_CH_LAYOUT_4POINT1           (AV_CH_LAYOUT_4POINT0|AV_CH_LOW_FREQUENCY)
#define AV_CH_LAYOUT_2_2               (AV_CH_LAYOUT_STEREO|AV_CH_SIDE_LEFT|AV_CH_SIDE_RIGHT
#define AV_CH_LAYOUT_QUAD              (AV_CH_LAYOUT_STEREO|AV_CH_BACK_LEFT|AV_CH_BACK_RIGHT
#define AV_CH_LAYOUT_5POINT0           (AV_CH_LAYOUT_SURROUND|AV_CH_SIDE_LEFT|AV_CH_SIDE_RIG
#define AV_CH_LAYOUT_5POINT1           (AV_CH_LAYOUT_5POINT0|AV_CH_LOW_FREQUENCY)
#define AV_CH_LAYOUT_5POINT0_BACK      (AV_CH_LAYOUT_SURROUND|AV_CH_BACK_LEFT|AV_CH_BACK_RIG
#define AV_CH_LAYOUT_5POINT1_BACK      (AV_CH_LAYOUT_5POINT0_BACK|AV_CH_LOW_FREQUENCY)
#define AV_CH_LAYOUT_6POINT0           (AV_CH_LAYOUT_5POINT0|AV_CH_BACK_CENTER)
#define AV_CH_LAYOUT_6POINT0_FRONT     (AV_CH_LAYOUT_2_2|AV_CH_FRONT_LEFT_OF_CENTER|AV_CH_FR
#define AV_CH_LAYOUT_HEXAGONAL         (AV_CH_LAYOUT_5POINT0_BACK|AV_CH_BACK_CENTER)
#define AV_CH_LAYOUT_6POINT1           (AV_CH_LAYOUT_5POINT1|AV_CH_BACK_CENTER)
#define AV_CH_LAYOUT_6POINT1_BACK      (AV_CH_LAYOUT_5POINT1_BACK|AV_CH_BACK_CENTER)
#define AV_CH_LAYOUT_6POINT1_FRONT     (AV_CH_LAYOUT_6POINT0_FRONT|AV_CH_LOW_FREQUENCY)
#define AV_CH_LAYOUT_7POINT0           (AV_CH_LAYOUT_5POINT0|AV_CH_BACK_LEFT|AV_CH_BACK_RIGH
#define AV_CH_LAYOUT_7POINT0_FRONT     (AV_CH_LAYOUT_5POINT0|AV_CH_FRONT_LEFT_OF_CENTER|AV_C
#define AV_CH_LAYOUT_7POINT1           (AV_CH_LAYOUT_5POINT1|AV_CH_BACK_LEFT|AV_CH_BACK_RIGH
#define AV_CH_LAYOUT_7POINT1_WIDE      (AV_CH_LAYOUT_5POINT1|AV_CH_FRONT_LEFT_OF_CENTER|AV_C
#define AV_CH_LAYOUT_7POINT1_WIDE_BACK (AV_CH_LAYOUT_5POINT1_BACK|AV_CH_FRONT_LEFT_OF_CENTER
#define AV_CH_LAYOUT_OCTAGONAL         (AV_CH_LAYOUT_5POINT0|AV_CH_BACK_LEFT|AV_CH_BACK_CENT
#define AV_CH_LAYOUT_HEXADECAGONAL     (AV_CH_LAYOUT_OCTAGONAL|AV_CH_WIDE_LEFT|AV_CH_WIDE_RI
#define AV_CH_LAYOUT_STEREO_DOWNMIX    (AV_CH_STEREO_LEFT|AV_CH_STEREO_RIGHT)
```

通过 `channels` , `channel_layout` 和 `sample_fmt` ,可以很便捷的设置声道的相关信息

例如：

```
channels = 2;
channel_layout = AV_CH_LAYOUT_STEREO
sample_fmt = AV_SAMPLE_FMT_FLTP
```

表示 这个音频数据有2个声道，分别是左右声道，声音数据信息采用浮点型的 `planar` 格式进行存储，即左右声道分开存储

sample_rate

sample_rate表示声道的采样率，表示1s内采集的声音样本个数。

例如：44100表示1s内采集了44100个声音的样本数据

AVFrame

AVFrame 存储的数据，是解码后的数据。即音频中的PCM数据

data

存储了音频声道或图片信息

linesize

对于视频，存储了每一个图片平面的长度

对于音频，存储了每一个声道中数据的长度

对于音频，只有 `linesize[0]` 被使用，因为音频中，每一个声道的大小应该相等

extended_data

对于视频，只是简单的指向 `data[]`

对于 `planar` 格式的音频，每一个声道有一个独立的数据指针，并且 `linesize[0]` 包含了每一个声道存储数据的大小

nb_samples

表示这一帧中，每个声道中有多少个采样点

详细代码讲解

```
- (void)tranformateToAAC {

    // 创建文件夹
    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMa
    NSString *docDir = [paths objectAtIndex:0];

    NSFileManager *fileManager = [NSFileManager defaultManager];

    NSString *h264FileName = @"testMp4ToAAC.aac";
    NSString *pcmFileName = @"testMp4ToPcm.pcm";


    NSString *h264Path = [docDir stringByAppendingPathComponent:h264FileName];
    NSString *pcmPath = [docDir stringByAppendingPathComponent:pcmFileName];


    [fileManager removeItemAtPath:h264Path error:nil];
    [fileManager removeItemAtPath:pcmPath error:nil];


    [fileManager createFileAtPath:h264Path contents:nil attributes:nil];
    [fileManager createFileAtPath:pcmPath contents:nil attributes:nil];


    fileHandle = [NSFileHandle fileHandleForWritingAtPath:h264Path];
    pcmfileHandle = [NSFileHandle fileHandleForWritingAtPath:pcmPath];


    NSLog(@"dic = %@",docDir);
    NSString *fileName = [docDir stringByAppendingPathComponent:@"video.mp4"];


    av_register_all();
    avcodec_register_all();

    AVFormatContext *inputFormatCtx = NULL;

    // 打开输入音频文件
    int ret = avformat_open_input(&inputFormatCtx, [fileName UTF8String], NULL, 0);

    if (ret != 0) {
        NSLog(@"打开文件失败");
        return;
    }

    //获取音频中流的相关信息
    ret = avformat_find_stream_info(inputFormatCtx, 0);

    if (ret != 0) {
        NSLog(@"不能获取流信息");
        return;
    }

    // 获取数据中音频流的序列号，这是一个标识符
    int  index = 0,audioStream = -1;
    AVCodecContext *inputCodecCtx;

    for (index = 0; index <inputFormatCtx->nb_streams; index++) {

        AVStream *stream = inputFormatCtx->streams[index];
        AVCodecContext *code = stream->codec;
        if (code->codec_type == AVMEDIA_TYPE_AUDIO){
            audioStream = index;
            break;
        }
    }

    //从音频流中获取输入编解码相关的上下文
    inputCodecCtx = inputFormatCtx->streams[audioStream]->codec;
    //查找解码器
    AVCodec *pCodec = avcodec_find_decoder(inputCodecCtx->codec_id);
    // 打开解码器
    int result =  avcodec_open2(inputCodecCtx, pCodec, nil);
    if (result < 0) {
        NSLog(@"打开音频解码器失败");
        return;
    }

    // 创建aac编码器
    AVCodec *aacCodec = avcodec_find_encoder(AV_CODEC_ID_AAC);

    if (!aacCodec){
        printf("Can not find encoder!\n");
        return ;
    }
```

```
//常见aac编码相关上下文信息
AVCodecContext *aacCodeContex = avcodec_alloc_context3(aacCodec);
// 设置编码相关信息
aacCodeContex->sample_fmt = aacCodec->sample_fmts[0];
aacCodeContex->sample_rate= inputCodecCtx->sample_rate;                // 音频的采样率
aacCodeContex->channel_layout = av_get_default_channel_layout(2);
aacCodeContex->channels = inputCodecCtx->channels;
aacCodeContex->strict_std_compliance = FF_COMPLIANCE_EXPERIMENTAL;

//打开编码器
AVDictionary *opts = NULL;
result = avcodec_open2(aacCodeContex, aacCodec, &opts);

if (result < 0) {
    NSLog(@"failure open code");
    return;
}

//初始化先进先出缓存队列
AVAudioFifo *fifo = av_audio_fifo_alloc(AV_SAMPLE_FMT_FLTP,aacCodeContex->channels, aacC

//获取编码每帧的最大取样数
int output_frame_size = aacCodeContex->frame_size;

// 初始化重采样上下文
SwrContext *resample_context = NULL;
if (init_resampler(inputCodecCtx, aacCodeContex,
                   &resample_context)){
}

BOOL finished  = NO;
while (1) {

    if (finished){
        break;
    }

    // 查看fifo队列中的大小是否超过可以编码的一帧的大小
    while (av_audio_fifo_size(fifo) < output_frame_size) {

        // 如果没超过，则继续进行解码

        if (finished)
        {
            break;
        }

        AVFrame *audioFrame = av_frame_alloc();
        AVPacket packet;
        packet.data = NULL;
        packet.size = 0;
        int data_present;

        // 读取出一帧未解码数据
        finished =  (av_read_frame(inputFormatCtx, &packet) == AVERROR_EOF);

        // 判断该帧数据是否为音频数据
        if (packet.stream_index != audioStream) {
            continue;
        }

        // 开始进行解码
        if ( avcodec_decode_audio4(inputCodecCtx, audioFrame, &data_present, &packet) <
            NSLog(@"音频解码失败");
            return ;
        }

        if (data_present)
        {
            //将pcm数据写入文件
            for(int i = 0 ; i <audioFrame->channels;i++)
            {
                NSData *data = [NSData dataWithBytes:audioFrame->data[i] length:audioFra
                [pcmfileHandle writeData:data];

            }
        }

        // 初始化进行重采样的存储空间
        uint8_t **converted_input_samples = NULL;
        if (init_converted_samples(&converted_input_samples, aacCodeContex,
                                   audioFrame->nb_samples))
        {
            return;
        }

        // 进行重采样
```

```
            if (convert_samples((const uint8_t**)audioFrame->extended_data, converted_input_
                                audioFrame->nb_samples, resample_context))
            {
                return;
            }

            //将采样结果加入进fifo中
            add_samples_to_fifo(fifo, converted_input_samples,audioFrame->nb_samples);


            // 释放重采样存储空间
            if (converted_input_samples)
            {
                av_freep(&converted_input_samples[0]);
                free(converted_input_samples);
            }
        }

        // 从fifo队列中读入数据
        while (av_audio_fifo_size(fifo) >= output_frame_size || finished) {

            AVFrame *frame;

            frame = av_frame_alloc();

            const int frame_size = FFMIN(av_audio_fifo_size(fifo),aacCodeContex->frame_size)

            // 设置输入帧的相关参数
            (frame)->nb_samples     = frame_size;
            (frame)->channel_layout = aacCodeContex->channel_layout;
            (frame)->format         = aacCodeContex->sample_fmt;
            (frame)->sample_rate    = aacCodeContex->sample_rate;

            int error;

            //根据帧的相关参数，获取数据存储空间
            if ((error = av_frame_get_buffer(frame, 0)) < 0)
            {
                av_frame_free(&frame);
                return ;
            }

            // 从fifo中读取frame_size个样本数据
            if (av_audio_fifo_read(fifo, (void **)frame->data, frame_size) < frame_size)
            {
                av_frame_free(&frame);
                return ;
            }

            AVPacket pkt;
            av_init_packet(&pkt);
            pkt.data = NULL;
            pkt.size = 0;

            int data_present = 0;

            frame->pts = av_frame_get_best_effort_timestamp(frame);
            frame->pict_type=AV_PICTURE_TYPE_NONE;

            // 将pcm数据进行编码
            if ((error = avcodec_encode_audio2(aacCodeContex, &pkt,frame, &data_present)) <
            {
                av_free_packet(&pkt);
                return ;
            }
            av_frame_free(&frame);

            // 如果编码成功，写入文件
            if (data_present) {
                NSData *data = [NSData dataWithBytes:pkt.data length:pkt.size];
                NSLog(@"pkt length = %d",pkt.size);
                [fileHandle writeData:[self adtsDataForPacketLength:pkt.size]];
                [fileHandle writeData:data];
            }

            av_free_packet(&pkt);
        }

    }

    NSLog(@"*************************************end");
}
```

参考资料

ffmpeg官方例子 (http://ffmpeg.org/doxygen/trunk/examples.html)
代码下载 (https://github.com/zhshijie/AudioTransform_AAC)