原 最简单的基于FFmpeg的封装格式处理：视音频分离器（demuxer）

2014年10月08日 00:58:56　阅读数：23377

==========

最简单的基于FFmpeg的封装格式处理系列文章列表：

最简单的基于FFmpeg的封装格式处理：视音频分离器简化版（demuxer-simple）

最简单的基于FFmpeg的封装格式处理：视音频分离器（demuxer）

最简单的基于FFmpeg的封装格式处理：视音频复用器（muxer）

最简单的基于FFMPEG的封装格式处理：封装格式转换（remuxer）

==========

简介

打算记录一下基于FFmpeg的封装格式处理方面的例子。包括了视音频分离，复用，封装格式转换。这是第2篇。

本文记录一个基于FFmpeg的视音频分离器(Simplest FFmpeg demuxer)。视音频分离器(Demuxer)即是将封装格式数据(例如MKV)中的视频压缩数据（例如H.264）和音频压缩数据（例如AAC）分离开。如图所示。在这个过程中并不涉及到编码和解码。

本文记录的程序可以将一个MPEG2TS封装的视频文件（其中视频编码为H.264，音频编码为AAC）分离成为两个文件：一个H.264编码的视频码流文件，一个AAC编码的音频码流文件。
前一篇文章中，记录一个简单版的视音频分离器。相比于前一篇文中的分离器，本篇文章记录的分离器复杂了很多。相比于简单版的分离器，学习的难度大了一些。但是该分离器可以很好地处理FFmpeg支持的各种格式（例如分离AAC音频流），拥有更好的实用性。

流程图

程序的流程如下图所示。从流程图中可以看出，一共初始化了3个AVFormatContext，其中1个用于输入，另外2个分别用于视频输出和音频输出。3个AVFormatContext初始化之后，通过avcodec_copy_context()函数可以将输入视频/音频的参数拷贝至输出视频/音频的AVCodecContext结构体。最后，通过av_read_frame()获取AVPacket，根据AVPacket类型的不同，分别使用av_interleaved_write_frame()写入不同的输出文件中即可。

单击查看更清晰的图片

PS：对于某些封装格式（例如MP4/FLV/MKV等）中的H.264，需要用到名称为“h264_mp4toannexb”的bitstream filter。这一点在前一篇文章《最简单的基于FFmpeg的封装格式处理：视音频分离器简化版（demuxer-simple）》中，已经有过详细叙述，这里不再重复。

简单介绍一下流程中各个重要函数的意义：

avformat_open_input()：打开输入文件。
avcodec_copy_context()：赋值AVCodecContext的参数。
avformat_alloc_output_context2()：初始化输出文件。
avio_open()：打开输出文件。
avformat_write_header()：写入文件头。
av_read_frame()：从输入文件读取一个AVPacket。
av_interleaved_write_frame()：写入一个AVPacket到输出文件。
av_write_trailer()：写入文件尾。

代码

下面贴上代码：

```cpp
/**
 * 最简单的基于FFmpeg的视音频分离器
 * Simplest FFmpeg Demuxer
 *
 * 雷霄骅 Lei Xiaohua
 * leixiaohua1020@126.com
 * 中国传媒大学/数字电视技术
 * Communication University of China / Digital TV Technology
 * http://blog.csdn.net/leixiaohua1020
 *
 * 本程序可以将封装格式中的视频码流数据和音频码流数据分离出来。
```

```
 * 在该例子中， 将MPEG2TS的文件分离得到H.264视频码流文件和AAC
 * 音频码流文件。
 *
 * This software split a media file (in Container such as
 * MKV, FLV, AVI...) to video and audio bitstream.
 * In this example, it demux a MPEG2TS file to H.264 bitstream
 * and AAC bitstream.
 */

#include <stdio.h>

#define __STDC_CONSTANT_MACROS

#ifdef _WIN32
//Windows
extern "C"
{
#include "libavformat/avformat.h"
};
#else
//Linux...
#ifdef __cplusplus
extern "C"
{
#endif
#include <libavformat/avformat.h>
#ifdef __cplusplus
};
#endif
#endif

/*
FIX: H.264 in some container format (FLV, MP4, MKV etc.) need
"h264_mp4toannexb" bitstream filter (BSF)
  *Add SPS,PPS in front of IDR frame
  *Add start code ("0,0,0,1") in front of NALU
H.264 in some container (MPEG2TS) don't need this BSF.
*/
//'1': Use H.264 Bitstream Filter
#define USE_H264BSF 0

int main(int argc, char* argv[])
{
    AVOutputFormat *ofmt_a = NULL,*ofmt_v = NULL;
    // (Input AVFormatContext and Output AVFormatContext)
    AVFormatContext *ifmt_ctx = NULL, *ofmt_ctx_a = NULL, *ofmt_ctx_v = NULL;
    AVPacket pkt;
    int ret, i;
    int videoindex=-1,audioindex=-1;
    int frame_index=0;

    const char *in_filename  = "cuc_ieschool.ts";//Input file URL
    //char *in_filename  = "cuc_ieschool.mkv";
    const char *out_filename_v = "cuc_ieschool.h264";//Output file URL
    //char *out_filename_a = "cuc_ieschool.mp3";
    const char *out_filename_a = "cuc_ieschool.aac";

    av_register_all();
    //Input
    if ((ret = avformat_open_input(&ifmt_ctx, in_filename, 0, 0)) < 0) {
        printf( "Could not open input file.");
        goto end;
    }
    if ((ret = avformat_find_stream_info(ifmt_ctx, 0)) < 0) {
        printf( "Failed to retrieve input stream information");
        goto end;
    }

    //Output
    avformat_alloc_output_context2(&ofmt_ctx_v, NULL, NULL, out_filename_v);
    if (!ofmt_ctx_v) {
        printf( "Could not create output context\n");
        ret = AVERROR_UNKNOWN;
        goto end;
    }
    ofmt_v = ofmt_ctx_v->oformat;

    avformat_alloc_output_context2(&ofmt_ctx_a, NULL, NULL, out_filename_a);
    if (!ofmt_ctx_a) {
        printf( "Could not create output context\n");
        ret = AVERROR_UNKNOWN;
        goto end;
    }
    ofmt_a = ofmt_ctx_a->oformat;

    for (i = 0; i < ifmt_ctx->nb_streams; i++) {
            //Create output AVStream according to input AVStream
            AVFormatContext *ofmt_ctx;
            AVStream *in_stream = ifmt_ctx->streams[i];
            AVStream *out_stream = NULL;

```

```
            if(ifmt_ctx->streams[i]->codec->codec_type==AVMEDIA_TYPE_VIDEO){
                videoindex=i;
                out_stream=avformat_new_stream(ofmt_ctx_v, in_stream->codec->codec);
                ofmt_ctx=ofmt_ctx_v;
            }else if(ifmt_ctx->streams[i]->codec->codec_type==AVMEDIA_TYPE_AUDIO){
                audioindex=i;
                out_stream=avformat_new_stream(ofmt_ctx_a, in_stream->codec->codec);
                ofmt_ctx=ofmt_ctx_a;
            }else{
                break;
            }

            if (!out_stream) {
                printf( "Failed allocating output stream\n");
                ret = AVERROR_UNKNOWN;
                goto end;
            }
            //Copy the settings of AVCodecContext
            if (avcodec_copy_context(out_stream->codec, in_stream->codec) < 0) {
                printf( "Failed to copy context from input to output stream codec context\n");
                goto end;
            }
            out_stream->codec->codec_tag = 0;

            if (ofmt_ctx->oformat->flags & AVFMT_GLOBALHEADER)
                out_stream->codec->flags |= CODEC_FLAG_GLOBAL_HEADER;
    }

    //Dump Format------------------
    printf("\n==============Input Video=============\n");
    av_dump_format(ifmt_ctx, 0, in_filename, 0);
    printf("\n==============Output Video============\n");
    av_dump_format(ofmt_ctx_v, 0, out_filename_v, 1);
    printf("\n==============Output Audio============\n");
    av_dump_format(ofmt_ctx_a, 0, out_filename_a, 1);
    printf("\n======================================\n");
    //Open output file
    if (!(ofmt_v->flags & AVFMT_NOFILE)) {
        if (avio_open(&ofmt_ctx_v->pb, out_filename_v, AVIO_FLAG_WRITE) < 0) {
            printf( "Could not open output file '%s'", out_filename_v);
            goto end;
        }
    }

    if (!(ofmt_a->flags & AVFMT_NOFILE)) {
        if (avio_open(&ofmt_ctx_a->pb, out_filename_a, AVIO_FLAG_WRITE) < 0) {
            printf( "Could not open output file '%s'", out_filename_a);
            goto end;
        }
    }

    //Write file header
    if (avformat_write_header(ofmt_ctx_v, NULL) < 0) {
        printf( "Error occurred when opening video output file\n");
        goto end;
    }
    if (avformat_write_header(ofmt_ctx_a, NULL) < 0) {
        printf( "Error occurred when opening audio output file\n");
        goto end;
    }

#if USE_H264BSF
    AVBitStreamFilterContext* h264bsfc =  av_bitstream_filter_init("h264_mp4toannexb");
#endif

    while (1) {
        AVFormatContext *ofmt_ctx;
        AVStream *in_stream, *out_stream;
        //Get an AVPacket
        if (av_read_frame(ifmt_ctx, &pkt) < 0)
            break;
        in_stream  = ifmt_ctx->streams[pkt.stream_index];


        if(pkt.stream_index==videoindex){
            out_stream = ofmt_ctx_v->streams[0];
            ofmt_ctx=ofmt_ctx_v;
            printf("Write Video Packet. size:%d\tpts:%lld\n",pkt.size,pkt.pts);
#if USE_H264BSF
            av_bitstream_filter_filter(h264bsfc, in_stream->codec, NULL, &pkt.data, &pkt.size, pkt.data, pkt.size, 0);
#endif
        }else if(pkt.stream_index==audioindex){
            out_stream = ofmt_ctx_a->streams[0];
            ofmt_ctx=ofmt_ctx_a;
            printf("Write Audio Packet. size:%d\tpts:%lld\n",pkt.size,pkt.pts);
        }else{
            continue;
        }


        //Convert PTS/DTS
```

```
        pkt.pts = av_rescale_q_rnd(pkt.pts, in_stream->time_base, out_stream->time_base, (AVRounding)
(AV_ROUND_NEAR_INF|AV_ROUND_PASS_MINMAX));
        pkt.dts = av_rescale_q_rnd(pkt.dts, in_stream->time_base, out_stream->time_base, (AVRounding)
(AV_ROUND_NEAR_INF|AV_ROUND_PASS_MINMAX));
        pkt.duration = av_rescale_q(pkt.duration, in_stream->time_base, out_stream->time_base);
        pkt.pos = -1;
        pkt.stream_index=0;
        //Write
        if (av_interleaved_write_frame(ofmt_ctx, &pkt) < 0) {
            printf( "Error muxing packet\n");
            break;
        }
        //printf("Write %8d frames to output file\n",frame_index);
        av_free_packet(&pkt);
        frame_index++;
    }

#if USE_H264BSF
    av_bitstream_filter_close(h264bsfc);
#endif

    //Write file trailer
    av_write_trailer(ofmt_ctx_a);
    av_write_trailer(ofmt_ctx_v);
end:
    avformat_close_input(&ifmt_ctx);
    /* close output */
    if (ofmt_ctx_a && !(ofmt_a->flags & AVFMT_NOFILE))
        avio_close(ofmt_ctx_a->pb);

    if (ofmt_ctx_v && !(ofmt_v->flags & AVFMT_NOFILE))
        avio_close(ofmt_ctx_v->pb);

    avformat_free_context(ofmt_ctx_a);
    avformat_free_context(ofmt_ctx_v);


    if (ret < 0 && ret != AVERROR_EOF) {
        printf( "Error occurred.\n");
        return -1;
    }
    return 0;
}
```

结果

输入文件为：
cuc_ieschool.ts：MPEG2TS封装格式数据。

输出文件为：
cuc_ieschool.h264：H.264视频码流数据。
cuc_ieschool.aac：AAC音频码流数据。

下载

simplest ffmpeg format

项目主页

SourceForge： https://sourceforge.net/projects/simplestffmpegformat/

Github： https://github.com/leixiaohua1020/simplest_ffmpeg_format

开源中国： http://git.oschina.net/leixiaohua1020/simplest_ffmpeg_format

CSDN下载地址：
http://download.csdn.net/detail/leixiaohua1020/8005317

工程中包含4个例子：

simplest_ffmpeg_demuxer_simple：视音频分离器（简化版）。

simplest_ffmpeg_demuxer：视音频分离器。

simplest_ffmpeg_muxer：视音频复用器。

simplest_ffmpeg_remuxer：封装格式转换器。

更新-1.1==========

修复了以下问题：
(1)Release版本下的运行问题
(2)simplest_ffmpeg_muxer分装H.264裸流的时候丢失声音的错误

CSDN下载地址：

http://download.csdn.net/detail/leixiaohua1020/8284309

更新-1.2 (2015.2.13)==========

这次考虑到了跨平台的要求，调整了源代码。经过这次调整之后，源代码可以在以下平台编译通过：

VC++：打开sln文件即可编译，无需配置。

cl.exe：打开compile_cl.bat即可命令行下使用cl.exe进行编译，注意可能需要按照VC的安装路径调整脚本里面的参数。编译命令如下。

```plain
::VS2010 Environment
call "D:\Program Files\Microsoft Visual Studio 10.0\VC\vcvarsall.bat"
::include
@set INCLUDE=include;%INCLUDE%
::lib
@set LIB=lib;%LIB%
::compile and link
cl simplest_ffmpeg_demuxer.cpp /link avcodec.lib avformat.lib avutil.lib ^
avdevice.lib avfilter.lib postproc.lib swresample.lib swscale.lib /OPT:NOREF
```

MinGW：MinGW命令行下运行compile_mingw.sh即可使用MinGW的g++进行编译。编译命令如下。

```plain
g++ simplest_ffmpeg_demuxer.cpp -g -o simplest_ffmpeg_demuxer.exe \
-I /usr/local/include -L /usr/local/lib -lavformat -lavcodec -lavutil
```

GCC：Linux或者MacOS命令行下运行compile_gcc.sh即可使用GCC进行编译。编译命令如下。

```plain
gcc simplest_ffmpeg_demuxer.cpp -g -o simplest_ffmpeg_demuxer.out \
-I /usr/local/include -L /usr/local/lib -lavformat -lavcodec -lavutil
```

PS：相关的编译命令已经保存到了工程文件夹中

CSDN下载地址： http://download.csdn.net/detail/leixiaohua1020/8445303
SourceForge上已经更新。

文章标签： ffmpeg demux 分离 AAC 封装格式
个人分类： 我的开源项目 FFMPEG
所属专栏： FFmpeg