原 最简单的基于FFMPEG的封装格式转换器（无编解码）

2014年05月10日 00:25:43　阅读数：51743

==========

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

本文介绍一个基于FFMPEG的封装格式转换器。所谓的封装格式转换，就是在AVI，FLV，MKV，MP4这些格式之间转换（对应.avi，.flv，.mkv，.mp4文件）。需要注意的是，本程序并不进行视音频的编码和解码工作。而是直接将视音频压缩码流从一种封装格式文件中获取出来然后打包成另外一种封装格式的文件。传统的转码程序工作原理如下图所示：

上图例举了一个举例：FLV（视频：H.264，音频：AAC）转码为AVI（视频：MPEG2，音频MP3）的例子。可见视频转码的过程通俗地讲相当于把视频和音频重新“录”了一遍。
本程序的工作原理如下图所示：

由图可见，本程序并不进行视频和音频的编解码工作，因此本程序和普通的转码软件相比，有以下两个特点：
处理速度极快。视音频编解码算法十分复杂，占据了转码的绝大部分时间。因为不需要进行视音频的编码和解码，所以节约了大量的时间。
视音频质量无损。因为不需要进行视音频的编码和解码，所以不会有视音频的压缩损伤。

流程（2014.9.29更新）

下面附上基于FFmpeg的Remuxer的流程图。图中使用浅红色标出了关键的数据结构，浅蓝色标出了输出视频数据的函数。可见成个程序包含了对两个文件的处理：读取输入文件（位于左边）和写入输出文件（位于右边）。中间使用了一个avcodec_copy_context()拷贝输入的AVCodecContext到输出的AVCodecContext。

简单介绍一下流程中关键函数的意义：

输入文件操作：

avformat_open_input()：打开输入文件，初始化输入视频码流的AVFormatContext。

av_read_frame()：从输入文件中读取一个AVPacket。

输出文件操作：

avformat_alloc_output_context2()：初始化输出视频码流的AVFormatContext。

avformat_new_stream()：创建输出码流的AVStream。

avcodec_copy_context()：拷贝输入视频码流的AVCodecContex的数值t到输出视频的AVCodecContext。

avio_open()：打开输出文件。

avformat_write_header()：写文件头（对于某些没有文件头的封装格式，不需要此函数。比如说MPEG2TS）。

av_interleaved_write_frame()：将AVPacket（存储视频压缩码流数据）写入文件。

av_write_trailer()：写文件尾（对于某些没有文件头的封装格式，不需要此函数。比如说MPEG2TS）。

代码

贴上代码，代码是从FFmpeg的例子改编的，平台是VC2010。

```cpp
/*
 *最简单的基于FFmpeg的封装格式转换器
 *Simplest FFmpeg Remuxer
 *
 *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
 *
 *本程序实现了视频封装格式之间的转换。
 *需要注意的是本程序并不改变视音频的编码格式。
 *
 * This software converts a media file from one container format
 * to another container format without encoding/decoding video files.
 */

#include "stdafx.h"

extern "C"
{
#include "libavformat/avformat.h"
};


int _tmain(int argc, _TCHAR* argv[])
{
    AVOutputFormat *ofmt = NULL;
    //输入对应一个AVFormatContext，输出对应一个AVFormatContext
    // (Input AVFormatContext and Output AVFormatContext)
    AVFormatContext *ifmt_ctx = NULL, *ofmt_ctx = NULL;
    AVPacket pkt;
    const char *in_filename, *out_filename;
    int ret, i;
    if (argc < 3) {
        printf("usage: %s input output\n"
            "Remux a media file with libavformat and libavcodec.\n"
            "The output format is guessed according to the file extension.\n"
            "Modified by Lei Xiaohua, leixiaohua1020@126.com\n"
            "Communication University of China / Digital TV Technology\n"
            "http://blog.csdn.net/leixiaohua1020", argv[0]);
        return 1;
    }
    in_filename  = argv[1];//输入文件名 (Input file URL)
    out_filename = argv[2];//输出文件名 (Output file URL)
    av_register_all();
    //输入 (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;
    }
    av_dump_format(ifmt_ctx, 0, in_filename, 0);
    //输出 (Output)
    avformat_alloc_output_context2(&ofmt_ctx, NULL, NULL, out_filename);
    if (!ofmt_ctx) {
        printf( "Could not create output context\n");
        ret = AVERROR_UNKNOWN;
        goto end;
    }
    ofmt = ofmt_ctx->oformat;
    for (i = 0; i < ifmt_ctx->nb_streams; i++) {
        //根据输入流创建输出流 (Create output AVStream according to input AVStream)
        AVStream *in_stream = ifmt_ctx->streams[i];
        AVStream *out_stream = avformat_new_stream(ofmt_ctx, in_stream->codec->codec);
        if (!out_stream) {
            printf( "Failed allocating output stream\n");
            ret = AVERROR_UNKNOWN;
            goto end;
        }
        //复制AVCodecContext的设置 (Copy the settings of AVCodecContext)
        ret = avcodec_copy_context(out_stream->codec, in_stream->codec);
        if (ret < 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;
    }
    //输出一下格式------------------
    av_dump_format(ofmt_ctx, 0, out_filename, 1);
    //打开输出文件 (Open output file)
    if (!(ofmt->flags & AVFMT_NOFILE)) {
```

```
        ret = avio_open(&ofmt_ctx->pb, out_filename, AVIO_FLAG_WRITE);
        if (ret < 0) {
            printf( "Could not open output file '%s'", out_filename);
            goto end;
        }
    }
    //写文件头（Write file header）
    ret = avformat_write_header(ofmt_ctx, NULL);
    if (ret < 0) {
        printf( "Error occurred when opening output file\n");
        goto end;
    }
    int frame_index=0;
    while (1) {
        AVStream *in_stream, *out_stream;
        //获取一个AVPacket（Get an AVPacket）
        ret = av_read_frame(ifmt_ctx, &pkt);
        if (ret < 0)
            break;
        in_stream  = ifmt_ctx->streams[pkt.stream_index];
        out_stream = ofmt_ctx->streams[pkt.stream_index];
        /* copy packet */
        //转换PTS/DTS（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;
        //写入（Write）
        ret = av_interleaved_write_frame(ofmt_ctx, &pkt);
        if (ret < 0) {
            printf( "Error muxing packet\n");
            break;
        }
        printf("Write %8d frames to output file\n",frame_index);
        av_free_packet(&pkt);
        frame_index++;
    }
    //写文件尾（Write file trailer）
    av_write_trailer(ofmt_ctx);
end:
    avformat_close_input(&ifmt_ctx);
    /* close output */
    if (ofmt_ctx && !(ofmt->flags & AVFMT_NOFILE))
        avio_close(ofmt_ctx->pb);
    avformat_free_context(ofmt_ctx);
    if (ret < 0 && ret != AVERROR_EOF) {
        printf( "Error occurred.\n");
        return -1;
    }
    return 0;
}
```

调试的时候，只需要“右键工程->调试->命令行参数”里面设置输入的文件名和输出的文件名就可以了。

结果

下图显示了一个测试的输入文件的视音频参数。

下图显示了输出文件的视音频参数。可以看出除了视频的封装格式从flv转换成了mp4，其他有关视音频编码的参数没有任何变化。

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：
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封装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]
1. ::VS2010 Environment
2. call "D:\Program Files\Microsoft Visual Studio 10.0\VC\vcvarsall.bat"
3. ::include
4. @set INCLUDE=include;%INCLUDE%
5. ::lib
6. @set LIB=lib;%LIB%
7. ::compile and link
8. cl simplest_ffmpeg_remuxer.cpp /link avcodec.lib avformat.lib avutil.lib ^
9. 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]
1. g++ simplest_ffmpeg_remuxer.cpp -g -o simplest_ffmpeg_remuxer.exe \
2. -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]
1. gcc simplest_ffmpeg_remuxer.cpp -g -o simplest_ffmpeg_remuxer.out -I /usr/local/include -L /usr/local/lib \
2. -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 封装格式 转换 转码
个人分类： FFMPEG 我的开源项目
所属专栏： FFmpeg