# 原 最简单的基于FFmpeg的AVUtil例子 (AVLog, AVOption等)

2015年07月18日 15:53:43　阅读数：11216

本文的示例程序记录了FFmpeg的libavutil中几种工具函数的使用方法：

AVLog：日志输出
AVOption (AVClass)：选项设置
AVDictionary：键值对存储
ParseUtil：字符串解析

## 几个libavutil的工具

### AVLog

AVLog是FFmpeg的日志输出工具。在FFmpeg中所有的日志输出不是通过printf()函数而是通过av_log()函数。av_log()会最终调用fprintf(stderr,…)函数将日志内容输出到命令行界面上。但是在一些非命令行程序（MFC程序，Android程序等）中，av_log()调用的fprintf(stderr,…)就无法将日志内容显示出来了。对于这种情况，FFmpeg提供了日志回调函数av_log_set_callback()。该函数可以指定一个自定义的日志输出函数，将日志输出到指定的位置。
下面的自定义函数custom_output()将日志输出到了“simplest_ffmpeg_log.txt”文本中。

```cpp
void custom_output(void* ptr, int level, const char* fmt,va_list vl){
    FILE *fp = fopen("simplest_ffmpeg_log.txt","a+");
    if(fp){
        vfprintf(fp,fmt,vl);
        fflush(fp);
        fclose(fp);
    }
}
```

在主函数中调用av_log_set_callback()设置一下该函数就可以了，如下所示。

```cpp
int main(int argc, char* argv[])
{
    av_log_set_callback(custom_output);
    return 0;
}
```

此外，日志信息从重到轻分为Panic、Fatal、Error、Warning、Info、Verbose、Debug几个级别。下面的函数输出了几种不同级别的日志。

```cpp
void test_log(){
    av_register_all();
    AVFormatContext *obj=NULL;
    obj=avformat_alloc_context();
    printf("====================================\n");
    av_log(obj,AV_LOG_PANIC,"Panic: Something went really wrong and we will crash now.\n");
    av_log(obj,AV_LOG_FATAL,"Fatal: Something went wrong and recovery is not possible.\n");
    av_log(obj,AV_LOG_ERROR,"Error: Something went wrong and cannot losslessly be recovered.\n");
    av_log(obj,AV_LOG_WARNING,"Warning: This may or may not lead to problems.\n");
    av_log(obj,AV_LOG_INFO,"Info: Standard information.\n");
    av_log(obj,AV_LOG_VERBOSE,"Verbose: Detailed information.\n");
    av_log(obj,AV_LOG_DEBUG,"Debug: Stuff which is only useful for libav* developers.\n");
    printf("====================================\n");
    avformat_free_context(obj);
}
```

PS：该部分源代码的解析可以参考文章《 FFmpeg源代码简单分析：日志输出系统（av_log()等） 》

### AVOption (AVClass)

AVOption是FFmpeg的选项设置工具。与AVOption最相关的选项设置函数就是av_opt_set()了。AVOption的核心概念就是“根据字符串操作结构体的属性值”。例如下面代码中“#if”和“#else”之间代码的作用和“#else”和“#endif”之间代码的作用是一样的。

```cpp
#if TEST_OPT
    av_opt_set(pCodecCtx,"b","400000",0);       //bitrate
    //Another method
    //av_opt_set_int(pCodecCtx,"b",400000,0);   //bitrate

    av_opt_set(pCodecCtx,"time_base","1/25",0); //time_base
    av_opt_set(pCodecCtx,"bf","5",0);           //max b frame
    av_opt_set(pCodecCtx,"g","25",0);           //gop
    av_opt_set(pCodecCtx,"qmin","10",0);        //qmin/qmax
    av_opt_set(pCodecCtx,"qmax","51",0);
#else
    pCodecCtx->time_base.num = 1;
    pCodecCtx->time_base.den = 25;
    pCodecCtx->max_b_frames=5;
    pCodecCtx->bit_rate = 400000;
    pCodecCtx->gop_size=25;
    pCodecCtx->qmin = 10;
    pCodecCtx->qmax = 51;
#endif
```

同理，av_opt_get()可以将结构体的属性值以字符串的形式返回回来。例如下面这段代码就验证了av_opt_get()的作用：

```cpp
char *val_str=(char *)av_malloc(50);

//preset: ultrafast, superfast, veryfast, faster, fast,
//medium, slow, slower, veryslow, placebo
av_opt_set(pCodecCtx->priv_data,"preset","slow",0);
//tune: film, animation, grain, stillimage, psnr,
//ssim, fastdecode, zerolatency
av_opt_set(pCodecCtx->priv_data,"tune","zerolatency",0);
//profile: baseline, main, high, high10, high422, high444
av_opt_set(pCodecCtx->priv_data,"profile","main",0);

//print
av_opt_get(pCodecCtx->priv_data,"preset",0,(uint8_t **)&val_str);
printf("preset val: %s\n",val_str);
av_opt_get(pCodecCtx->priv_data,"tune",0,(uint8_t **)&val_str);
printf("tune val: %s\n",val_str);
av_opt_get(pCodecCtx->priv_data,"profile",0,(uint8_t **)&val_str);
printf("profile val: %s\n",val_str);
av_free(val_str);
```

可以通过av_opt_find()获取结构体中任意选项的AVOption结构体。写了一个简单的函数读取该结构体中一些字段的值。

```cpp
void print_opt(const AVOption *opt_test){

    printf("====================================\n");
    printf("Option Information:\n");
    printf("[name]%s\n",opt_test->name);
    printf("[help]%s\n",opt_test->help);
    printf("[offset]%d\n",opt_test->offset);

    switch(opt_test->type){
    case AV_OPT_TYPE_INT:{
        printf("[type]int\n[default]%d\n",opt_test->default_val.i64);
        break;
                        }
    case AV_OPT_TYPE_INT64:{
        printf("[type]int64\n[default]%lld\n",opt_test->default_val.i64);
        break;
                          }
    case AV_OPT_TYPE_FLOAT:{
        printf("[type]float\n[default]%f\n",opt_test->default_val.dbl);
        break;
                          }
    case AV_OPT_TYPE_STRING:{
        printf("[type]string\n[default]%s\n",opt_test->default_val.str);
        break;
                           }
    case AV_OPT_TYPE_RATIONAL:{
        printf("[type]rational\n[default]%d/%d\n",opt_test->default_val.q.num,opt_test->default_val.q.den);
        break;
                             }
    default:{
        printf("[type]others\n");
        break;
            }
    }

    printf("[max val]%f\n",opt_test->max);
    printf("[min val]%f\n",opt_test->min);

    if(opt_test->flags&AV_OPT_FLAG_ENCODING_PARAM){
        printf("Encoding param.\n");
    }
    if(opt_test->flags&AV_OPT_FLAG_DECODING_PARAM){
        printf("Decoding param.\n");
    }
    if(opt_test->flags&AV_OPT_FLAG_AUDIO_PARAM){
        printf("Audio param.\n");
    }
    if(opt_test->flags&AV_OPT_FLAG_VIDEO_PARAM){
        printf("Video param.\n");
    }
    if(opt_test->unit!=NULL)
        printf("Unit belong to:%s\n",opt_test->unit);

    printf("====================================\n");
}
```

使用下列代码调用上面的函数就可以打印出AVOption结构体每个字段的值。

```cpp
const AVOption *opt=NULL;
opt=av_opt_find(pCodecCtx, "b", NULL, 0, 0);
print_opt(opt);
opt=av_opt_find(pCodecCtx, "g", NULL, 0, 0);
print_opt(opt);
opt=av_opt_find(pCodecCtx, "time_base", NULL, 0, 0);
print_opt(opt);
```

下面代码可以打印出支持AVOption（即包含AVClass）的结构体的所有选项：

```cpp
void list_obj_opt(void *obj){
    printf("Output some option info about object:\n");
    printf("Object name:%s\n",(*(AVClass **) obj)->class_name);
    printf("====================================\n");
    printf("Video param:\n");
    av_opt_show2(obj,stderr,AV_OPT_FLAG_VIDEO_PARAM,NULL);
    printf("Audio param:\n");
    av_opt_show2(obj,stderr,AV_OPT_FLAG_AUDIO_PARAM,NULL);
    printf("Decoding param:\n");
    av_opt_show2(obj,stderr,AV_OPT_FLAG_DECODING_PARAM,NULL);
    printf("Encoding param:\n");
    av_opt_show2(obj,stderr,AV_OPT_FLAG_ENCODING_PARAM,NULL);
    printf("====================================\n");
}
```

下面代码调用上面的函数就可以打印出AVFormatContext中的所有选项。

```cpp
void test_opt(){
    av_register_all();
    AVFormatContext *obj=NULL;
    obj=avformat_alloc_context();
    list_obj_opt(obj);
    avformat_free_context(obj);
}
```

PS：该部分源代码的解析可以参考文章《 FFmpeg源代码简单分析：结构体成员管理系统-AVClass 》、《 FFmpeg源代码简单分析：结构体成员管理系统-AVOption 》。

## AVDictionary

AVDictionary是FFmpeg的键值对存储工具，FFmpeg经常使用AVDictionary设置/读取内部参数。下面这段代码记录了AVDictionary的使用方法。

```cpp
void test_avdictionary(){

    AVDictionary *d = NULL;
    AVDictionaryEntry *t = NULL;

    av_dict_set(&d, "name", "lei xiaohua", 0);
    av_dict_set(&d, "email", "leixiaohua1020@126.com", 0);
    av_dict_set(&d, "school", "cuc", 0);
    av_dict_set(&d, "gender", "man", 0);
    av_dict_set(&d, "website", "http://blog.csdn.net/leixiaohua1020", 0);
    //av_strdup()
    char *k = av_strdup("location");
    char *v = av_strdup("Beijing-China");
    av_dict_set(&d, k, v, AV_DICT_DONT_STRDUP_KEY | AV_DICT_DONT_STRDUP_VAL);
    printf("====================================\n");
    int dict_cnt= av_dict_count(d);
    printf("dict_count:%d\n",dict_cnt);
    printf("dict_element:\n");
    while (t = av_dict_get(d, "", t, AV_DICT_IGNORE_SUFFIX)) {
        printf("key:%10s  |  value:%s\n",t->key,t->value);
    }

    t = av_dict_get(d, "email", t, AV_DICT_IGNORE_SUFFIX);
    printf("email is %s\n",t->value);
    printf("====================================\n");
    av_dict_free(&d);
}
```

## ParseUtil

ParseUtil是FFmpeg的字符串解析工具。它的分辨率解析函数av_parse_video_size()可以从形如“1920x1080”的字符串中解析出图像宽为1920，高为1080；它的帧率函数av_parse_video_rate()可以解析出帧率信息；它的时间解析函数则可以从形如“00:01:01”的字符串解析出时间的毫秒数。下面这段代码记录了ParseUtil的使用方法。

```cpp
void test_parseutil(){
    char input_str[100]={0};
    printf("========= Parse Video Size =========\n");
    int output_w=0;
    int output_h=0;
    strcpy(input_str,"1920x1080");
    av_parse_video_size(&output_w,&output_h,input_str);
    printf("w:%4d | h:%4d\n",output_w,output_h);
    strcpy(input_str,"vga");
    //strcpy(input_str,"hd1080");
    //strcpy(input_str,"ntsc");
    av_parse_video_size(&output_w,&output_h,input_str);
    printf("w:%4d | h:%4d\n",output_w,output_h);
    printf("========= Parse Frame Rate =========\n");
    AVRational output_rational={0,0};
    strcpy(input_str,"15/1");
    av_parse_video_rate(&output_rational,input_str);
    printf("framerate:%d/%d\n",output_rational.num,output_rational.den);
    strcpy(input_str,"pal");
    av_parse_video_rate(&output_rational,input_str);
    printf("framerate:%d/%d\n",output_rational.num,output_rational.den);
    printf("========== Parse Time ============\n");
    int64_t output_timeval;
    strcpy(input_str,"00:01:01");
    av_parse_time(&output_timeval,input_str,1);
    printf("microseconds:%lld\n",output_timeval);
    printf("====================================\n");
}
```

## 源代码

```cpp
/**
 * 最简单的FFmpeg的AVUtil示例
 * Simplest FFmpeg AVUtil
 *
 * 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
 *
 * 本程序是FFmpeg中的libavutil的示例，目前包含：
 * AVLog
 * AVOption (AVClass)
 * AVDictionary
 * ParseUtil
 *
 * This software is the example about FFmpeg's libavutil.
 * It contains:
 * AVLog
 * AVOption (AVClass)
 * AVDictionary
 * ParseUtil
 *
 */

#include <stdio.h>

#define __STDC_CONSTANT_MACROS

#ifdef _WIN32
//Windows
extern "C"
{
#include "libavcodec/avcodec.h"
#include "libavformat/avformat.h"
#include "libavutil/opt.h"
#include "libavutil/parseutils.h"
#include "libavutil/avutil.h"
};
#else
//Linux...
#ifdef __cplusplus
extern "C"
{
#endif
#include <libavcodec/avcodec.h>
#include <libavformat/avformat.h>
#include <libavutil/opt.h>
#include <libavutil/parseutils.h>
#include <libavutil/avutil.h>
#ifdef __cplusplus
};
#endif
#endif


#define TEST_OPT    1
#define TEST_LOG    1
#define TEST_DIC    0


void list_obj_opt(void *obj){
    printf("Output some option info about object:\n");
    printf("Object name:%s\n",(*(AVClass **) obj)->class_name);
    printf("====================================\n");
    printf("Video param:\n");
    av_opt_show2(obj,stderr,AV_OPT_FLAG_VIDEO_PARAM,NULL);
    printf("Audio param:\n");
    av_opt_show2(obj,stderr,AV_OPT_FLAG_AUDIO_PARAM,NULL);
    printf("Decoding param:\n");
    av_opt_show2(obj,stderr,AV_OPT_FLAG_DECODING_PARAM,NULL);
    printf("Encoding param:\n");
    av_opt_show2(obj,stderr,AV_OPT_FLAG_ENCODING_PARAM,NULL);
    printf("====================================\n");
}

void test_opt(){
    av_register_all();
    AVFormatContext *obj=NULL;
    obj=avformat_alloc_context();
    list_obj_opt(obj);
    avformat_free_context(obj);
```

```
}


void test_log(){
    av_register_all();
    AVFormatContext *obj=NULL;
    obj=avformat_alloc_context();
    printf("====================================\n");
    av_log(obj,AV_LOG_PANIC,"Panic: Something went really wrong and we will crash now.\n");
    av_log(obj,AV_LOG_FATAL,"Fatal: Something went wrong and recovery is not possible.\n");
    av_log(obj,AV_LOG_ERROR,"Error: Something went wrong and cannot losslessly be recovered.\n");
    av_log(obj,AV_LOG_WARNING,"Warning: This may or may not lead to problems.\n");
    av_log(obj,AV_LOG_INFO,"Info: Standard information.\n");
    av_log(obj,AV_LOG_VERBOSE,"Verbose: Detailed information.\n");
    av_log(obj,AV_LOG_DEBUG,"Debug: Stuff which is only useful for libav* developers.\n");
    printf("====================================\n");
    avformat_free_context(obj);
}

void print_opt(const AVOption *opt_test){

    printf("====================================\n");
    printf("Option Information:\n");
    printf("[name]%s\n",opt_test->name);
    printf("[help]%s\n",opt_test->help);
    printf("[offset]%d\n",opt_test->offset);

    switch(opt_test->type){
    case AV_OPT_TYPE_INT:{
        printf("[type]int\n[default]%d\n",opt_test->default_val.i64);
        break;
                         }
    case AV_OPT_TYPE_INT64:{
        printf("[type]int64\n[default]%lld\n",opt_test->default_val.i64);
        break;
                           }
    case AV_OPT_TYPE_FLOAT:{
        printf("[type]float\n[default]%f\n",opt_test->default_val.dbl);
        break;
                           }
    case AV_OPT_TYPE_STRING:{
        printf("[type]string\n[default]%s\n",opt_test->default_val.str);
        break;
                            }
    case AV_OPT_TYPE_RATIONAL:{
        printf("[type]rational\n[default]%d/%d\n",opt_test->default_val.q.num,opt_test->default_val.q.den);
        break;
                              }
    default:{
        printf("[type]others\n");
        break;
            }
    }

    printf("[max val]%f\n",opt_test->max);
    printf("[min val]%f\n",opt_test->min);

    if(opt_test->flags&AV_OPT_FLAG_ENCODING_PARAM){
        printf("Encoding param.\n");
    }
    if(opt_test->flags&AV_OPT_FLAG_DECODING_PARAM){
        printf("Decoding param.\n");
    }
    if(opt_test->flags&AV_OPT_FLAG_AUDIO_PARAM){
        printf("Audio param.\n");
    }
    if(opt_test->flags&AV_OPT_FLAG_VIDEO_PARAM){
        printf("Video param.\n");
    }
    if(opt_test->unit!=NULL)
        printf("Unit belong to:%s\n",opt_test->unit);

    printf("====================================\n");
}

int flush_encoder(AVFormatContext *fmt_ctx,unsigned int stream_index)
{
    int ret;
    int got_frame;
    AVPacket enc_pkt;
    if (!(fmt_ctx->streams[stream_index]->codec->codec->capabilities &
        CODEC_CAP_DELAY))
        return 0;
    while (1) {
        printf("Flushing stream #%u encoder\n", stream_index);
        enc_pkt.data = NULL;
        enc_pkt.size = 0;
        av_init_packet(&enc_pkt);
        ret = avcodec_encode_video2 (fmt_ctx->streams[stream_index]->codec, &enc_pkt,
            NULL, &got_frame);
        av_frame_free(NULL);
```

```
        if (ret < 0)
            break;
        if (!got_frame){
            ret=0;
            break;
        }
        printf("Succeed to encode 1 frame!\n");
        /* mux encoded frame */
        ret = av_write_frame(fmt_ctx, &enc_pkt);
        if (ret < 0)
            break;
    }
    return ret;
}

int encoder(){
    AVFormatContext* pFormatCtx;
    AVStream* video_st;
    AVCodecContext* pCodecCtx;
    AVCodec* pCodec;

    uint8_t* picture_buf;
    AVFrame* picture;
    int size;
    int ret;
    AVPacket pkt;
    int y_size;

    FILE *in_file = fopen("ds_480x272.yuv", "rb");  //Input YUV data
    int in_w=480,in_h=272;                          //Input width and height
    //Frames to encode
    int framenum=100;
    const char* out_file = "ds.h264";               //Output Filepath
    //const char* out_file = "ds.ts";
    //const char* out_file = "ds.hevc";
    char temp_str[250]={0};

    av_register_all();

    avformat_alloc_output_context2(&pFormatCtx, NULL, NULL, out_file);

    if (avio_open(&pFormatCtx->pb,out_file, AVIO_FLAG_READ_WRITE) < 0){
        printf("Failed to open output file!\n");
        return -1;
    }

    pCodec = avcodec_find_encoder(pFormatCtx->oformat->video_codec);
    if (!pCodec) {
        fprintf(stderr, "Codec not found.\n");
        return -1;
    }
    video_st = avformat_new_stream(pFormatCtx, pCodec);
    video_st->time_base.num = 1;
    video_st->time_base.den = 25;

    if (video_st==NULL){
        return -1;
    }
    //Param that must set
    pCodecCtx = video_st->codec;
    pCodecCtx->codec_type = AVMEDIA_TYPE_VIDEO;
    pCodecCtx->pix_fmt = PIX_FMT_YUV420P;
    pCodecCtx->width = in_w;
    pCodecCtx->height = in_h;

#if TEST_OPT
    av_opt_set(pCodecCtx,"b","400000",0);       //bitrate
    //Another method
    //av_opt_set_int(pCodecCtx,"b",400000,0);   //bitrate

    av_opt_set(pCodecCtx,"time_base","1/25",0); //time_base
    av_opt_set(pCodecCtx,"bf","5",0);           //max b frame
    av_opt_set(pCodecCtx,"g","25",0);           //gop
    av_opt_set(pCodecCtx,"qmin","10",0);        //qmin/qmax
    av_opt_set(pCodecCtx,"qmax","51",0);
#else
    pCodecCtx->time_base.num = 1;
    pCodecCtx->time_base.den = 25;
    pCodecCtx->max_b_frames=5;
    pCodecCtx->bit_rate = 400000;
    pCodecCtx->gop_size=25;
    pCodecCtx->qmin = 10;
    pCodecCtx->qmax = 51;
#endif

#if TEST_OPT
    //list_obj_opt(pFormatCtx);
    //list_obj_opt(pCodecCtx);
    const AVOption *opt=NULL;
    opt=av_opt_find(pCodecCtx, "b", NULL, 0, 0);
    print_opt(opt);
```

```
    opt=av_opt_find(pCodecCtx, "g", NULL, 0, 0);
    print_opt(opt);
    opt=av_opt_find(pCodecCtx, "time_base", NULL, 0, 0);
    print_opt(opt);
    //Get Option
    //Get String
    int64_t *val_str=(int64_t *)av_malloc(1*sizeof(int64_t));
    av_opt_get(pCodecCtx,"b",0,(uint8_t **)&val_str);
    printf("get bitrate(str):%s\n",val_str);
    av_free(val_str);
    //Get int
    int64_t val_int=0;
    av_opt_get_int(pCodecCtx,"b",0,&val_int);
    printf("get bitrate(int):%lld\n",val_int);
#endif

    AVDictionary *param = 0;

    //H.264
    if(pCodecCtx->codec_id == AV_CODEC_ID_H264) {
        char *val_str=(char *)av_malloc(50);
        //List it
        //list_obj_opt(pCodecCtx->priv_data);

        //preset: ultrafast, superfast, veryfast, faster, fast,
        //medium, slow, slower, veryslow, placebo
        av_opt_set(pCodecCtx->priv_data,"preset","slow",0);
        //tune: film, animation, grain, stillimage, psnr,
        //ssim, fastdecode, zerolatency
        av_opt_set(pCodecCtx->priv_data,"tune","zerolatency",0);
        //profile: baseline, main, high, high10, high422, high444
        av_opt_set(pCodecCtx->priv_data,"profile","main",0);

        //print
        av_opt_get(pCodecCtx->priv_data,"preset",0,(uint8_t **)&val_str);
        printf("preset val: %s\n",val_str);
        av_opt_get(pCodecCtx->priv_data,"tune",0,(uint8_t **)&val_str);
        printf("tune val: %s\n",val_str);
        av_opt_get(pCodecCtx->priv_data,"profile",0,(uint8_t **)&val_str);
        printf("profile val: %s\n",val_str);
        av_free(val_str);

#if TEST_DIC
        av_dict_set(¶m, "preset", "slow", 0);
        av_dict_set(¶m, "tune", "zerolatency", 0);
        //av_dict_set(¶m, "profile", "main", 0);
#endif
    }
    //H.265
    if(pCodecCtx->codec_id == AV_CODEC_ID_H265){
        //list_obj_opt(pCodecCtx->priv_data);

        //preset: ultrafast, superfast, veryfast, faster, fast,
        //medium, slow, slower, veryslow, placebo
        av_opt_set(pCodecCtx->priv_data, "preset", "ultrafast", 0);
        //tune: psnr, ssim, zerolatency, fastdecode
        av_opt_set(pCodecCtx->priv_data, "tune", "zero-latency", 0);
        //profile: main, main10, mainstillpicture
        av_opt_set(pCodecCtx->priv_data,"profile","main",0);
    }

    if (avcodec_open2(pCodecCtx, pCodec,¶m) < 0){
        printf("Failed to open encoder!\n");
        return -1;
    }

    picture = avcodec_alloc_frame();
    size = avpicture_get_size(pCodecCtx->pix_fmt, pCodecCtx->width, pCodecCtx->height);
    picture_buf = (uint8_t *)av_malloc(size);
    avpicture_fill((AVPicture *)picture, picture_buf, pCodecCtx->pix_fmt, pCodecCtx->width, pCodecCtx->height);

    //Write File Header
    avformat_write_header(pFormatCtx,NULL);

    y_size = pCodecCtx->width * pCodecCtx->height;
    av_new_packet(&pkt,y_size*3);

    for (int i=0; i<framenum; i++){
        //Read YUV
        if (fread(picture_buf, 1, y_size*3/2, in_file) < 0){
            printf("Failed to read YUV data!\n");
            return -1;
        }else if(feof(in_file)){
            break;
        }
        picture->data[0] = picture_buf;              // Y
        picture->data[1] = picture_buf+ y_size;      // U
        picture->data[2] = picture_buf+ y_size*5/4;  // V
        //PTS
        picture->pts=i;
        int got_picture=0;
```

```
        //Encode
        ret = avcodec_encode_video2(pCodecCtx, &pkt,picture, &got_picture);
        if(ret < 0){
            printf("Failed to encode!\n");
            return -1;
        }
        if (got_picture==1){
            //printf("Succeed to encode 1 frame!\n");
            pkt.stream_index = video_st->index;
            ret = av_write_frame(pFormatCtx, &pkt);
            av_free_packet(&pkt);
        }
    }
    //Flush Encoder
    ret = flush_encoder(pFormatCtx,0);
    if (ret < 0) {
        printf("Flushing encoder failed\n");
        return -1;
    }

    //Write file trailer
    av_write_trailer(pFormatCtx);

    //Clean
    if (video_st){
        avcodec_close(video_st->codec);
        av_free(picture);
        av_free(picture_buf);
    }
    avio_close(pFormatCtx->pb);
    avformat_free_context(pFormatCtx);

    fclose(in_file);
    return 0;
}

void custom_output(void* ptr, int level, const char* fmt,va_list vl){
    FILE *fp = fopen("simplest_ffmpeg_log.txt","a+");
    if(fp){
        vfprintf(fp,fmt,vl);
        fflush(fp);
        fclose(fp);
    }
}

void test_parseutil(){
    char input_str[100]={0};
    printf("========= Parse Video Size =========\n");
    int output_w=0;
    int output_h=0;
    strcpy(input_str,"1920x1080");
    av_parse_video_size(&output_w,&output_h,input_str);
    printf("w:%4d | h:%4d\n",output_w,output_h);
    strcpy(input_str,"vga");
    //strcpy(input_str,"hd1080");
    //strcpy(input_str,"ntsc");
    av_parse_video_size(&output_w,&output_h,input_str);
    printf("w:%4d | h:%4d\n",output_w,output_h);
    printf("========= Parse Frame Rate =========\n");
    AVRational output_rational={0,0};
    strcpy(input_str,"15/1");
    av_parse_video_rate(&output_rational,input_str);
    printf("framerate:%d/%d\n",output_rational.num,output_rational.den);
    strcpy(input_str,"pal");
    av_parse_video_rate(&output_rational,input_str);
    printf("framerate:%d/%d\n",output_rational.num,output_rational.den);
    printf("========== Parse Time =============\n");
    int64_t output_timeval;
    strcpy(input_str,"00:01:01");
    av_parse_time(&output_timeval,input_str,1);
    printf("microseconds:%lld\n",output_timeval);
    printf("=====================================\n");
}

void test_avdictionary(){

    AVDictionary *d = NULL;
    AVDictionaryEntry *t = NULL;

    av_dict_set(&d, "name", "lei xiaohua", 0);
    av_dict_set(&d, "email", "leixiaohua1020@126.com", 0);
    av_dict_set(&d, "school", "cuc", 0);
    av_dict_set(&d, "gender", "man", 0);
    av_dict_set(&d, "website", "http://blog.csdn.net/leixiaohua1020", 0);
    //av_strdup()
    char *k = av_strdup("location");
    char *v = av_strdup("Beijing-China");
    av_dict_set(&d, k, v, AV_DICT_DONT_STRDUP_KEY | AV_DICT_DONT_STRDUP_VAL);
    printf("====================================\n");
    int dict_cnt= av_dict_count(d);
    printf("dict_count:%d\n",dict_cnt);
```

```
    printf("dict element:\n");
    while (t = av_dict_get(d, "", t, AV_DICT_IGNORE_SUFFIX)) {
        printf("key:%10s  |  value:%s\n",t->key,t->value);
    }

    t = av_dict_get(d, "email", t, AV_DICT_IGNORE_SUFFIX);
    printf("email is %s\n",t->value);
    printf("===================================\n");
    av_dict_free(&d);
}

int main(int argc, char* argv[])
{
    int loglevel=av_log_get_level();
    av_log_set_level(AV_LOG_DEBUG);
    //av_log_set_flags(AV_LOG_PRINT_LEVEL);
    //av_log_set_callback(custom_output);
    test_log();

    test_avdictionary();
    test_parseutil();

    //test_opt();

    encoder();

    return 0;
}
```

## 运行结果

不同级别的AVLog日志输出后的结果：

结构体中所有AVOption信息输出后的结果：

AVDictionary示例输出的结果：

ParseUtil示例输出的结果：

编码的时候设置回调函数后输出到文本中的日志：

## 下载

**Simplest FFmpeg AVUtil**

**项目主页**
SourceForge：https://sourceforge.net/projects/simplestffmpegavutil/
Github：https://github.com/leixiaohua1020/simplest_ffmpeg_avutil
开源中国：http://git.oschina.net/leixiaohua1020/simplest_ffmpeg_avutil

CSDN下载地址：http://download.csdn.net/detail/leixiaohua1020/8924311

本程序是FFmpeg中的libavutil的示例，目前包含：

- AVLog
- AVOption (AVClass)
- AVDictionary
- ParseUtil



文章标签：FFmpeg AVlog AVOption AVDictionary ParseUtil
个人分类：FFMPEG
所属专栏：FFmpeg