原 ffmpeg.c函数结构简单分析（画图）

2014年10月04日 00:12:40　阅读数：26785

前一阵子研究转码的时候看了FFmpeg的源代码。由于ffmpeg.c的代码相对比较长，而且其中有相当一部分是AVFilter有关的代码（这一部分一直不太熟），因此之前学习FFmpeg的时候一直也没有好好看一下其源代码。最近正好看了看AVFilter的知识，顺便就看了下FFmpeg的源代码，在这里画图理一下它的结构。目前好多地方还没有弄明白，等到以后慢慢完善了。
先说明一下自己画的结构图的规则：图中仅画出了比较重要的函数之间的调用关系。粉红色的函数是FFmpeg编解码类库（libavcodec，libavformat等）的API。绿色的函数是FFmpeg的libavfilter的API。其他不算很重要的函数就不再列出了。

PS：有一部分代码可能和ffmpeg.c有一些出入。因为本文使用的ffmpeg.c的代码是移植到VC之后的代码。

在看ffmpeg.c的代码之前，最好先看一下简单的代码了解FFmpeg解码，编码的关键API：

100行代码实现最简单的基于FFMPEG+SDL的视频播放器

最简单的基于FFmpeg+SDL的音频播放器

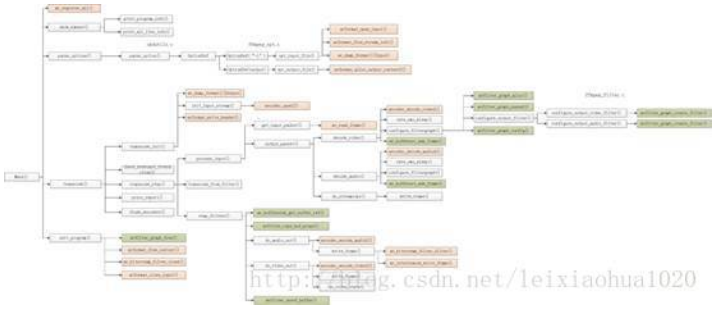
最简单的基于FFMPEG的视频编码器（YUV编码为H.264）

最简单的基于FFMPEG的音频编码器（PCM编码为AAC）

最简单的基于FFmpeg的转码程序

函数调用结构图

FFmpeg的总体函数调用结构图如下图所示



上图所示本是一张高清大图。但是页面显示不下。因此上传了一份：

http://my.csdn.net/leixiaohua1020/album/detail/1788075

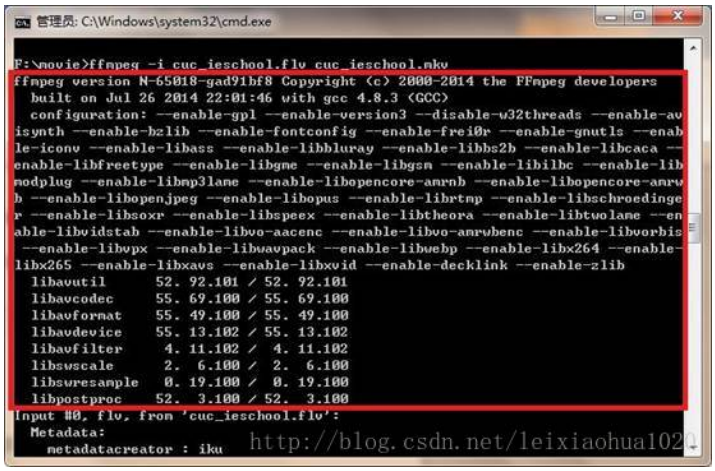
上面地址的那张图保存下来的话就是一张清晰的图片了。

下文将会对主要函数分别解析。

main()

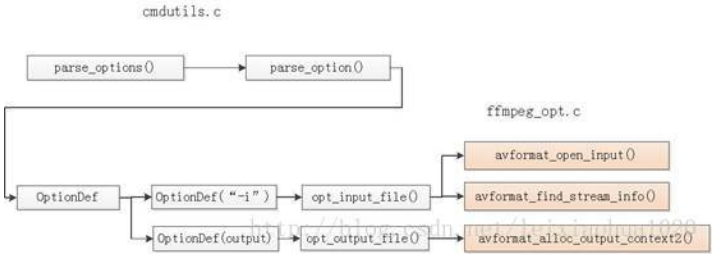
main()是FFmpeg的主函数。
调用了如下函数
av_register_all()：注册所有编码器和解码器。
show_banner()：打印输出FFmpeg版本信息（编译时间，编译选项，类库信息等）。
parse_options()：解析输入的命令。
transcode()：转码。
exit_progam()：退出和清理。

下图红框中的内容即为show_banner()的输出结果。



parse_options()

parse_options()解析全部输入选项。即将输入命令“ffmpeg -i xxx.mpg -vcodec libx264 yyy.mkv”中的“-i”，“-vcodec”这样的命令解析出来。其函数调用结构如下图所示。
注：定义位于cmdutils.c中。



调用了如下函数：
parse_option()：解析一个输入选项。具体的解析步骤不再赘述。parse_options()会循环调用parse_option()直到所有选项解析完毕。FFmpeg的每一个选项信息存储在一个OptionDef结构体中。定义如下：

```cpp
typedef struct OptionDef {
    const char *name;
    int flags;
#define HAS_ARG    0x0001
#define OPT_BOOL   0x0002
#define OPT_EXPERT 0x0004
#define OPT_STRING 0x0008
#define OPT_VIDEO  0x0010
#define OPT_AUDIO  0x0020
#define OPT_INT    0x0080
#define OPT_FLOAT  0x0100
#define OPT_SUBTITLE 0x0200
#define OPT_INT64  0x0400
#define OPT_EXIT   0x0800
#define OPT_DATA   0x1000
#define OPT_PERFILE  0x2000     /* the option is per-file (currently ffmpeg-only).
         implied by OPT_OFFSET or OPT_SPEC */
#define OPT_OFFSET 0x4000       /* option is specified as an offset in a passed optctx */
#define OPT_SPEC   0x8000       /* option is to be stored in an array of SpecifierOpt.
         Implies OPT_OFFSET. Next element after the offset is
         an int containing element count in the array. */
#define OPT_TIME  0x10000
#define OPT_DOUBLE 0x20000
     union {
        void *dst_ptr;
        int (*func_arg)(void *, const char *, const char *);
        size_t off;
    } u;
    const char *help;
    const char *argname;
} OptionDef;
```

其中的重要字段：
name：用于存储选项的名称。例如“i”，“f”，“codec”等等。
flags：存储选项值的类型。例如：HAS_ARG（包含选项值），OPT_STRING（选项值为字符串类型），OPT_TIME（选项值为时间类型。
u：存储该选项的处理函数。

help：选项的说明信息。
FFmpeg使用一个名称为options，类型为OptionDef的数组存储所有的选项。有一部分通用选项存储在cmdutils_common_opts.h中。cmdutils_common_opts.h内容如下：

```cpp
{ "L"           , OPT_EXIT, {(void*)show_license},      "show license" },
{ "h"           , OPT_EXIT, {(void*) show_help},         "show help", "topic" },
{ "?"           , OPT_EXIT, {(void*)show_help},         "show help", "topic" },
{ "help"        , OPT_EXIT, {(void*)show_help},         "show help", "topic" },
{ "-help"       , OPT_EXIT, {(void*)show_help},         "show help", "topic" },
{ "version"     , OPT_EXIT, {(void*)show_version},      "show version" },
{ "formats"     , OPT_EXIT, {(void*)show_formats   },   "show available formats" },
{ "codecs"      , OPT_EXIT, {(void*)show_codecs    },   "show available codecs" },
{ "decoders"    , OPT_EXIT, {(void*)show_decoders  },   "show available decoders" },
{ "encoders"    , OPT_EXIT, {(void*)show_encoders  },   "show available encoders" },
{ "bsfs"        , OPT_EXIT, {(void*)show_bsfs      },   "show available bit stream filters" },
{ "protocols"   , OPT_EXIT, {(void*)show_protocols},    "show available protocols" },
{ "filters"     , OPT_EXIT, {(void*)show_filters   },   "show available filters" },
{ "pix_fmts"    , OPT_EXIT, {(void*)show_pix_fmts  },   "show available pixel formats" },
{ "layouts"     , OPT_EXIT, {(void*)show_layouts   },   "show standard channel layouts" },
{ "sample_fmts", OPT_EXIT, {(void*)show_sample_fmts }, "show available audio sample formats" },
{ "loglevel"    , HAS_ARG,  {(void*)opt_loglevel},      "set libav* logging level", "loglevel" },
{ "v",            HAS_ARG,  {(void*)opt_loglevel},      "set libav* logging level", "loglevel" },
{ "debug"       , HAS_ARG,  {(void*)opt_codec_debug},   "set debug flags", "flags" },
{ "fdebug"      , HAS_ARG,  {(void*)opt_codec_debug},   "set debug flags", "flags" },
{ "report"      , 0,        {(void*)opt_report}, "generate a report" },
{ "max_alloc"   , HAS_ARG,  {(void*) opt_max_alloc},     "set maximum size of a single allocated block", "bytes" },
{ "cpuflags"    , HAS_ARG | OPT_EXPERT, {(void*) opt_cpuflags}, "force specific cpu flags", "flags" },
```

options数组的定义位于ffmpeg_opt.c中：

```cpp
const OptionDef options[] = {
    /* main options */
#include "cmdutils_common_opts.h"//包含了cmdutils_common_opts.h中的选项
    { "f", HAS_ARG | OPT_STRING | OPT_OFFSET,           { (void*)OFFSET(format) },
        "force format", "fmt" },
    { "i", HAS_ARG | OPT_PERFILE,          { (void*) opt_input_file },
        "input file name", "filename" },
    { "y", OPT_BOOL,          { &file_overwrite },
        "overwrite output files" },
    { "n", OPT_BOOL,          { &no_file_overwrite },
        "do not overwrite output files" },
    { "c", HAS_ARG | OPT_STRING | OPT_SPEC,{ (void*) OFFSET(codec_names) },
        "codec name", "codec" },
    { "codec",          HAS_ARG | OPT_STRING | OPT_SPEC,{(void*) OFFSET(codec_names) },
        "codec name", "codec" },
    { "pre",            HAS_ARG | OPT_STRING | OPT_SPEC,{ (void*) OFFSET(presets) },
        "preset name", "preset" },
    { "map",            HAS_ARG | OPT_EXPERT | OPT_PERFILE,          { (void*) opt_map },
        "set input stream mapping",
        "[-]input_file_id[:stream_specifier][,sync_file_id[:stream_specifier]]" },
    { "map_channel",    HAS_ARG | OPT_EXPERT | OPT_PERFILE,          {(void*)opt_map_channel },
        "map an audio channel from one stream to another", "file.stream.channel[:syncfile.syncstream]" },
    { "map_metadata",   HAS_ARG | OPT_STRING | OPT_SPEC,{ (void*)OFFSET(metadata_map) },
        "set metadata information of outfile from infile",
        "outfile[,metadata]:infile[,metadata]" },
    { "map_chapters",   HAS_ARG | OPT_INT | OPT_EXPERT | OPT_OFFSET, { (void*) OFFSET(chapters_input_file) },
        "set chapters mapping", "input_file_index" },
    { "t", HAS_ARG | OPT_TIME | OPT_OFFSET,{(void*) OFFSET(recording_time) },
        "record or transcode \"duration\" seconds of audio/video",
        "duration" },
    { "fs",HAS_ARG | OPT_INT64 | OPT_OFFSET,            { (void*) OFFSET(limit_filesize) },
        "set the limit file size in bytes", "limit_size" },
    { "ss",HAS_ARG | OPT_TIME | OPT_OFFSET,{ (void*) OFFSET(start_time) },
        "set the start time offset", "time_off" },
   …//选项太多，不一一列出
};
```

在这里，例举一个选项的OptionDef结构体：输入

```cpp
{ "i",HAS_ARG | OPT_PERFILE, { (void*) opt_input_file }, "input file name", "filename" }
```

在这个结构体中，可以看出选项的名称为“i”，选项包含选项值（HAS_ARG），选项的处理函数是opt_input_file()，选项的说明是“input file name”。下面可以详细看一下选项的处理函数opt_input_file()。该函数的定义位于ffmpeg_opt.c文件中。可以看出，调用了avformat_alloc_context()初始化了AVFormatContext结构体，调用了avformat_open_input()函数打开了“-i”选项指定的文件。此外，调用了avformat_find_stream_info()等完成了一些初始化操作。此外，调用了av_dump_format()打印输出输入文件信息。

```cpp
static int opt_input_file(void *optctx, const char *opt, const char *filename)
{
    //略…
    /* open the input file with generic avformat function */
    err = avformat_open_input(&ic, filename, file_iformat, &format_opts);
    if (err < 0) {
        print_error(filename, err);
        exit(1);
    }

    //略…
    /* Set AVCodecContext options for avformat_find_stream_info */
    opts = setup_find_stream_info_opts(ic, codec_opts);
    orig_nb_streams = ic->nb_streams;

    /* If not enough info to get the stream parameters, we decode the
       first frames to get it. (used in mpeg case for example) */
    ret = avformat_find_stream_info(ic, opts);
    if (ret < 0) {
        av_log(NULL, AV_LOG_FATAL, "%s: could not find codec parameters\n", filename);
        avformat_close_input(&ic);
        exit(1);
    }

    //略…
    /* dump the file content */
    av_dump_format(ic, nb_input_files, filename, 0);

    //略…
    return 0;
}
```

再例举一个输出文件处理函数opt_output_file()。这里需要注意，输出文件的处理并不包含在OptionDef类型的数组options中。因为FFmpeg中指定输出文件时并不包含选项名称，这是一个比较特殊的地方。一般的选项格式是“-名称 值”，例如指定输入文件的时候，选项格式是“-i xxx.flv”。而指定输出文件的时候，直接指定“值”即可，这是新手可能容易搞混的地方。
例如，最简单的转码命令如下（输出文件前面不包含选项）：

```plain
ffmpeg -i xxx.mpg xxx.mkv
```

而不是

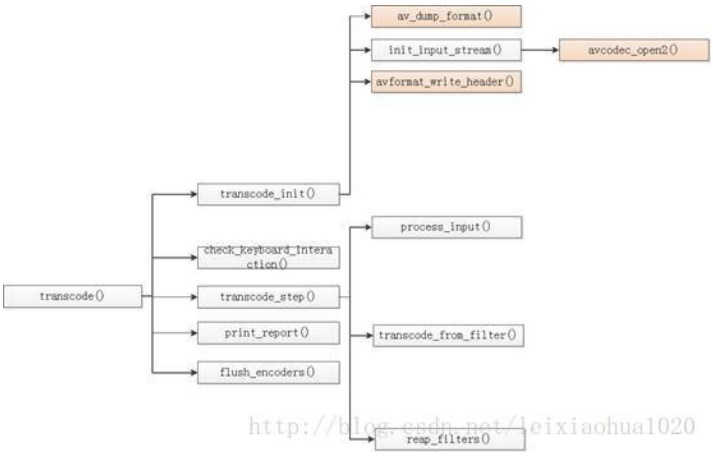
```plain
ffmpeg -i xxx.mpeg -o xxx.mkv
```

下面简单看一下opt_output_file()函数的定义。该函数的定义同样位于ffmpeg_opt.c文件中。这个函数的定义特别长，完成了输出视频的初始化工作。在这里就不列出代码了。该函数首先调用avformat_alloc_output_context2()初始化AVFormatContext结构体。而后根据媒体类型的不同，分别调用new_video_stream()，new_audio_stream()，new_subtitle_stream()等创建不同的AVStream。实际上上述的几个创建AVStream的函数调用了new_output_stream()。而new_output_stream()又调用了FFmpeg类库的API函数avformat_new_stream()。

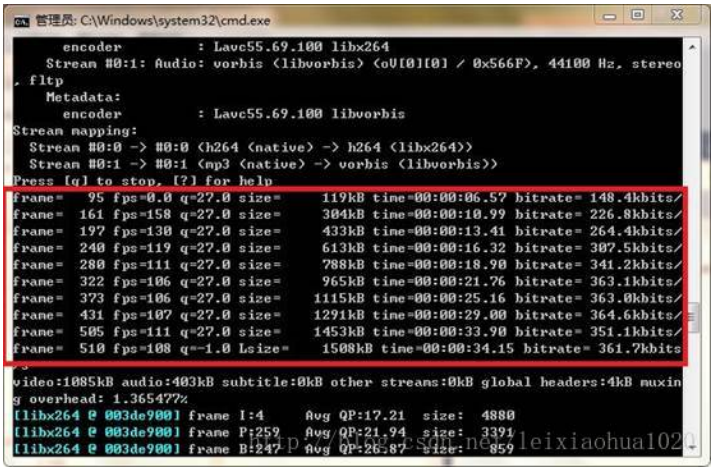
```cpp
void opt_output_file(void *optctx, const char *filename)
{
    //略…
    err = avformat_alloc_output_context2(&oc, NULL, o->format, filename);

    if (!oc) {
        print_error(filename, err);
        exit(1);
    }
    //略…
   new_video_stream();
   …
   new_audio_stream();
   …
   new_subtitle_stream ();
    //略…

}
```

transcode()

transcode()的功能是转码。其函数调用结构如下图所示。



调用了如下函数
transcode_init()：转码的初始化工作。
check_keyboard_interaction()：检测键盘操作。例如转码的过程中按下“Q”键之后，会退出转码。
transcode_step()：进行转码。
print_report()：打印转码信息，输出到屏幕上。
flush_encoder()：输出编码器中剩余的帧。
其中check_keyboard_interaction()，transcode_step()，print_report()三个函数位于一个循环之中会不断地执行。
下图红框所示即为print_report()打印输出到屏幕上的信息。



下面简单介绍两个重要的函数transcode_init()和transcode_step()。

transcode_init()

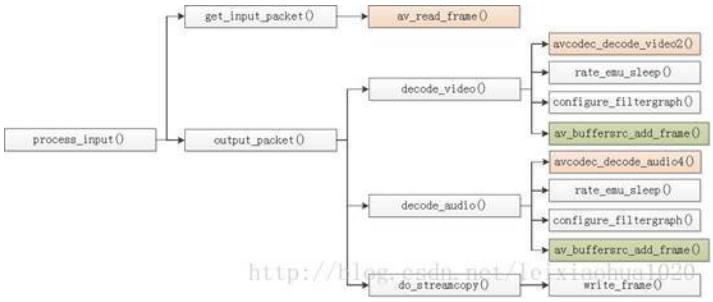
transcode_init()调用了以下几个重要的函数：
av_dump_format()：在屏幕上打印输出格式信息。注意是输出格式的信息，输入格式的信息的打印是在parse_options()函数执行过程中调用opt_input_file()的时候打印到屏幕上的。
init_input_stream()：其中调用了avcodec_open2()打开编码器。
avformat_write_header()：写输出文件的文件头。

transcode_step()

transcode_step()调用了如下函数：
process_input()：完成解码工作。
transcode_from_filter()：未分析。
reap_filters()：完成编码工作。

process_input()

process_input()主要完成了解码的工作。其函数调用结构如下图所示。



process_input()调用了如下函数：
get_input_packet()：获取一帧压缩编码数据，即一个AVPacket。其中调用了av_read_frame()。
output_packet()：解码压缩编码的数据并将之送至AVFilterContext。

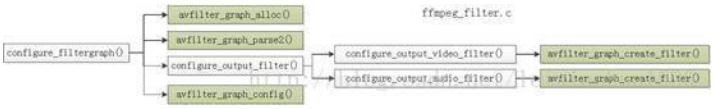
output_packet()调用了如下函数：
decode_video()：解码一帧视频（一个AVPacket）。
decode_audio()：解码音频（并不一定是一帧，是一个AVPacket）。
do_streamcopy()：如果不需要重新编码的话，则调用此函数，一般用于封装格式之间的转换。速度比转码快很多。

decode_video()调用了如下函数：
avcodec_decode_video2()：解码一帧视频。
rate_emu_sleep()：要求按照帧率处理数据的时候调用，可以避免FFmpeg处理速度过快。常用于网络实时流的处理（RTP/RTMP流的推送）。
configure_filtergraph()：设置AVFilterGraph。
av_buffersrc_add_frame()：将解码后的数据（一个AVFrame）送至AVFilterContext。

decode_audio()调用的函数和decode_video()基本一样。唯一的不同在于其解码音频的函数是avcodec_decode_audio4()

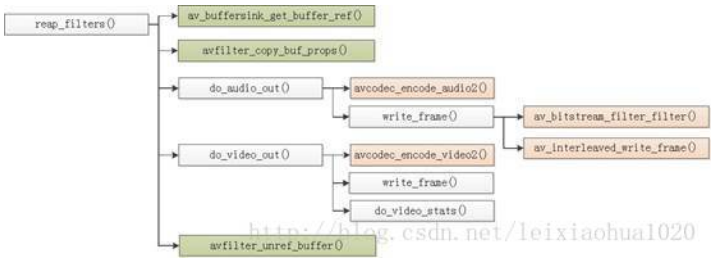
configure_filtergraph()

未分析。



reap_filters()

reap_filters()主要完成了编码的工作。其函数调用结构如下图所示。



reap_filters()调用了如下函数
av_buffersink_get_buffer_ref()：从AVFilterContext中取出一帧解码后的数据（结构为AVFilterBufferRef，可以转换为AVFrame）。
avfilter_copy_buf_props()：AVFilterBufferRef转换为AVFrame。
do_audio_out()：编码音频。
do_video_out()：编码视频。
avfilter_unref_buffer()：释放资源。

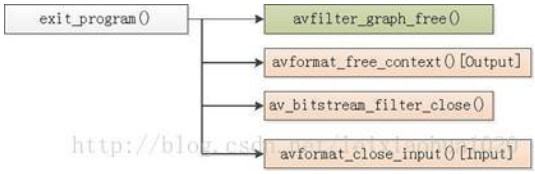
do_video_out()调用了如下函数
avcodec_encode_video2()：编码一帧视频。
write_frame()：写入编码后的视频压缩数据。

write_frame()调用了如下函数：
av_bitstream_filter_filter()：使用AVBitStreamFilter的时候，会调用此函数进行处理。
av_interleaved_write_frame()：写入压缩编码数据。

do_audio_out()调用的函数与do_video_out()基本上一样。唯一的不同在于视频编码函数avcodec_encode_video2()变成了音频编码函数avcodec_encode_audio2()。

exit_program()

exit_program()主要完成了清理工作。调用关系如下图所示。



调用了如下函数：
avfilter_graph_free()：释放AVFilterGraph。
avformat_free_context()：释放输出文件的AVFormatContext。
av_bitstream_filter_close()：关闭AVBitStreamFilter。
avformat_close_input()：关闭输入文件。

附录

FFmpeg转码时在屏幕上的输出。

（转码命令为ffmpeg -i cuc_ieschool.flv cuc_ieschool.mkv）

F:\movie>ffmpeg -i cuc_ieschool.flv cuc_ieschool.mkv
//版本信息（main()->show_banner()）

```
ffmpeg version N-65018-gad91bf8 Copyright (c) 2000-2014 the FFmpeg developers
built on Jul 26 2014 22:01:46 with gcc 4.8.3 (GCC)
configuration: --enable-gpl --enable-version3 --disable-w32threads --enable-av
isynth --enable-bzlib --enable-fontconfig --enable-frei0r --enable-gnutls --enab
le-iconv --enable-libass --enable-libbluray --enable-libbs2b --enable-libcaca --
enable-libfreetype --enable-libgme --enable-libgsm --enable-libilbc --enable-lib
modplug --enable-libmp3lame --enable-libopencore-amrnb --enable-libopencore-amrw
b --enable-libopenjpeg --enable-libopus --enable-librtmp --enable-libschroedinge
r --enable-libsoxr --enable-libspeex --enable-libtheora --enable-libtwolame --en
able-libvidstab --enable-libvo-aacenc --enable-libvo-amrwbenc --enable-libvorbis
--enable-libvpx --enable-libwavpack --enable-libwebp --enable-libx264 --enable-
libx265 --enable-libxavs --enable-libxvid --enable-decklink --enable-zlib
libavutil      52. 92.101 / 52. 92.101
libavcodec     55. 69.100 / 55. 69.100
libavformat    55. 49.100 / 55. 49.100
libavdevice    55. 13.102 / 55. 13.102
libavfilter     4. 11.102 /  4. 11.102
libswscale      2.  6.100 /  2.  6.100
libswresample   0. 19.100 /  0. 19.100
libpostproc    52.  3.100 / 52.  3.100
```

//输入信息（main()->parse_options()->opt_input_file()->av_dump_format()）

```
Input #0, flv, from 'cuc_ieschool.flv':
Metadata:
metadatacreator : iku
hasKeyframes    : true
hasVideo        : true
hasAudio        : true
hasMetadata     : true
canSeekToEnd    : false
datasize        : 932906
videosize       : 787866
audiosize       : 140052
lasttimestamp   : 34
lastkeyframetimestamp: 30
lastkeyframelocation: 886498
encoder         : Lavf55.19.104
Duration: 00:00:34.16, start: 0.000000, bitrate: 318 kb/s
Stream #0:0: Video: h264 (Main), yuv420p, 512x288 [SAR 1:1 DAR 16:9], 183 kb
/s, 15.17 fps, 15 tbr, 1k tbn, 30 tbc
Stream #0:1: Audio: mp3, 44100 Hz, stereo, s16p, 128 kb/s
[libx264 @ 003de900] using SAR=1/1
[libx264 @ 003de900] using cpu capabilities: MMX2 SSE2Fast SSSE3 SSE4.2 AVX
[libx264 @ 003de900] profile High, level 2.1
[libx264 @ 003de900] 264 - core 142 r2431 ac76440 - H.264/MPEG-4 AVC codec - Cop
yleft 2003-2014 - http://www.videolan.org/x264.html - options: cabac=1 ref=3 deb
lock=1:0:0 analyse=0x3:0x113 me=hex subme=7 psy=1 psy_rd=1.00:0.00 mixed_ref=1 m
e_range=16 chroma_me=1 trellis=1 8x8dct=1 cqm=0 deadzone=21,11 fast_pskip=1 chro
ma_qp_offset=-2 threads=6 lookahead_threads=1 sliced_threads=0 nr=0 decimate=1 i
nterlaced=0 bluray_compat=0 constrained_intra=0 bframes=3 b_pyramid=2 b_adapt=1
b_bias=0 direct=1 weightb=1 open_gop=0 weightp=2 keyint=250 keyint_min=15 scenec
ut=40 intra_refresh=0 rc_lookahead=40 rc=crf mbtree=1 crf=23.0 qcomp=0.60 qpmin=
0 qpmax=69 qpstep=4 ip_ratio=1.40 aq=1:1.00
```

//输出信息（main()->transcode()->transcode_init()->av_dump_format()）

```
Output #0, matroska, to 'cuc_ieschool.mkv':
Metadata:
metadatacreator : iku
hasKeyframes    : true
hasVideo        : true
hasAudio        : true
hasMetadata     : true
canSeekToEnd    : false
datasize        : 932906
videosize       : 787866
audiosize       : 140052
lasttimestamp   : 34
lastkeyframetimestamp: 30
lastkeyframelocation: 886498
encoder         : Lavf55.49.100
Stream #0:0: Video: h264 (libx264) (H264 / 0x34363248), yuv420p, 512x288 [SA
R 1:1 DAR 16:9], q=-1--1, 15 fps, 1k tbn, 15 tbc
Metadata:
encoder         : Lavc55.69.100 libx264
Stream #0:1: Audio: vorbis (libvorbis) (oV[0][0] / 0x566F), 44100 Hz, stereo
, fltp
Metadata:
encoder         : Lavc55.69.100 libvorbis
```

//输出Stream Maping 信息（main()->transcode()->transcode_init()）

```
Stream mapping:
Stream #0:0 -> #0:0 (h264 (native) -> h264 (libx264))
Stream #0:1 -> #0:1 (mp3 (native) -> vorbis (libvorbis))
```

//一行字（main()->transcode()）

```
Press [q] to stop, [?] for help
```

//输出信息（main()->transcode()->print_report()）

```
frame=   95 fps=0.0 q=27.0 size=     119kB time=00:00:06.57 bitrate= 148.4kbits/
frame=  161 fps=158 q=27.0 size=     304kB time=00:00:10.99 bitrate= 226.8kbits/
frame=  197 fps=130 q=27.0 size=     433kB time=00:00:13.41 bitrate= 264.4kbits/
frame=  240 fps=119 q=27.0 size=     613kB time=00:00:16.32 bitrate= 307.5kbits/
frame=  280 fps=111 q=27.0 size=     788kB time=00:00:18.90 bitrate= 341.2kbits/
frame=  322 fps=106 q=27.0 size=     965kB time=00:00:21.76 bitrate= 363.1kbits/
frame=  373 fps=106 q=27.0 size=    1115kB time=00:00:25.16 bitrate= 363.0kbits/
frame=  431 fps=107 q=27.0 size=    1291kB time=00:00:29.00 bitrate= 364.6kbits/
frame=  505 fps=111 q=27.0 size=    1453kB time=00:00:33.90 bitrate= 351.1kbits/
frame=  510 fps=108 q=-1.0 Lsize=    1508kB time=00:00:34.15 bitrate= 361.7kbits/s
```

//最后一次输出

```
video:1085kB audio:403kB subtitle:0kB other streams:0kB global headers:4kB muxin
g overhead: 1.365477%
```

//avcodec_close()的时候输出（libx264专有的输出信息）

```
[libx264 @ 003de900] frame I:4     Avg QP:17.21  size:  4880
[libx264 @ 003de900] frame P:259   Avg QP:21.94  size:  3391
[libx264 @ 003de900] frame B:247   Avg QP:26.87  size:   859
[libx264 @ 003de900] consecutive B-frames: 18.6% 46.3% 12.4% 22.7%
[libx264 @ 003de900] mb I  I16..4: 23.0% 57.5% 19.4%
[libx264 @ 003de900] mb P  I16..4:  4.2%  7.5%  4.2%  P16..4: 30.1% 12.2%  5.7%
0.0%  0.0%    skip:36.2%
[libx264 @ 003de900] mb B  I16..4:  0.3%  0.5%  0.4%  B16..8: 28.8%  4.4%  1.0%
direct: 1.4%  skip:63.2%  L0:40.4% L1:49.9% BI: 9.7%
[libx264 @ 003de900] 8x8 transform intra:47.7% inter:41.7%
[libx264 @ 003de900] coded y,uvDC,uvAC intra: 47.4% 27.8% 5.0% inter: 13.2% 4.4%
0.3%
[libx264 @ 003de900] i16 v,h,dc,p: 22% 36%  9% 33%
[libx264 @ 003de900] i8 v,h,dc,ddl,ddr,vr,hd,vl,hu: 23% 26% 28%  3%  3%  3%  4%
3%  4%
[libx264 @ 003de900] i4 v,h,dc,ddl,ddr,vr,hd,vl,hu: 22% 25% 20%  5%  6%  5%  6%
5%  6%
[libx264 @ 003de900] i8c dc,h,v,p: 71% 18% 10%  1%
[libx264 @ 003de900] Weighted P-Frames: Y:0.8% UV:0.8%
[libx264 @ 003de900] ref P L0: 69.3% 12.3% 14.1%  4.3%  0.0%
[libx264 @ 003de900] ref B L0: 83.9% 15.3%  0.7%
[libx264 @ 003de900] ref B L1: 96.0%  4.0%
[libx264 @ 003de900] kb/s:261.17
```


文章标签： ffmpeg 函数 分析

个人分类： FFMPEG

所属专栏： FFmpeg