原 ffplay.c函数结构简单分析（画图）

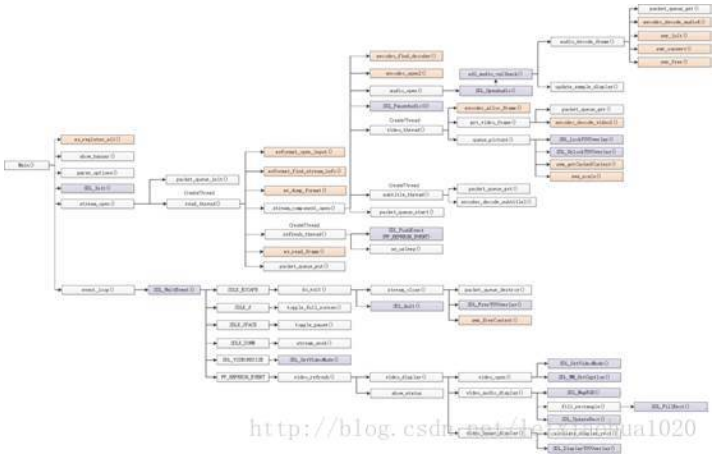
2014年10月04日 12:48:54　阅读数：23967

最近重温了一下FFplay的源代码。FFplay是FFmpeg项目提供的播放器示例。尽管FFplay只是一个简单的播放器示例，它的源代码的量也是不少的。之前看代码，主要是集中于某一个“点”进行研究，而没有从总体结构上进行分析。本文就打算弥补之前学习的不足，从总体结构上分析一下FFplay的源代码，画图理一下它的结构。其中还有诸多不足，以后有机会慢慢完善。
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。紫色的函数是SDL的API。其他不算很重要的函数就不再列出了。
在看ffplay.c的代码之前，最好先看一下简单的代码了解FFmpeg播放一个视频的核心代码：
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

总体结构图

FFplay的总体函数调用结构图如下图所示。



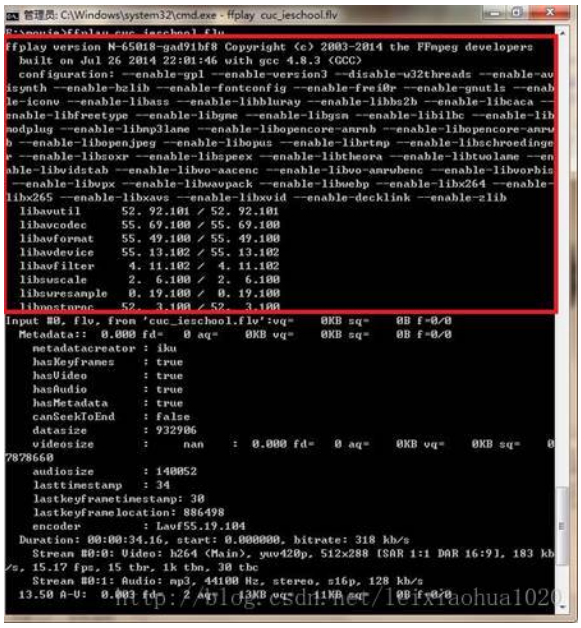
上图所示本是一张高清大图。但是页面显示不下。因此上传了一份：
http://my.csdn.net/leixiaohua1020/album/detail/1788077
上面地址的那张图保存下来的话就是一张清晰的图片了。

下文对主要函数分别解析。

main()

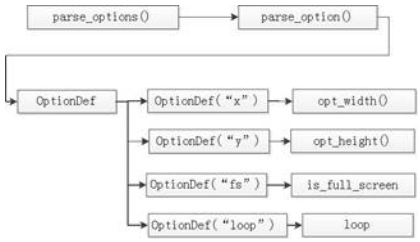
main()是FFplay的主函数。
调用了如下函数
av_register_all()：注册所有编码器和解码器。
show_banner()：打印输出FFmpeg版本信息（编译时间，编译选项，类库信息等）。
parse_options()：解析输入的命令。
SDL_Init()：SDL初始化。
stream_open ()：打开输入媒体。
event_loop()：处理各种消息，不停地循环下去。

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“ffplay -f h264 test.264”中的“-f”这样的命令解析出来。其函数调用结构如下图所示。需要注意的是，FFplay（ffplay.c）的parse_options()和FFmpeg（ffmpeg.c）中的parse_options()实际上是一样的。因此本部分的内容和《 ffmpeg.c函数结构简单分析（画图） 》中的parse_options()有很多重复的地方。



parse_options()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。这些选项对于FFmpeg，FFplay以及FFprobe都试用。
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ffplay.c中，如下所示：

```cpp
static const OptionDef options[] = {
#include "cmdutils_common_opts.h"//包含进来
    { "x", HAS_ARG, { (void*) opt_width }, "force displayed width", "width" },
    { "y", HAS_ARG, { (void*) opt_height }, "force displayed height", "height" },
    { "s", HAS_ARG | OPT_VIDEO, { (void*) opt_frame_size }, "set frame size (WxH or abbreviation)", "size" },
    { "fs", OPT_BOOL, { &is_full_screen }, "force full screen" },
    { "an", OPT_BOOL, { &audio_disable }, "disable audio" },
    { "vn", OPT_BOOL, { &video_disable }, "disable video" },
    { "ast", OPT_INT | HAS_ARG | OPT_EXPERT, { &wanted_stream[AVMEDIA_TYPE_AUDIO] }, "select desired audio stream", "stream_number"
},
    { "vst", OPT_INT | HAS_ARG | OPT_EXPERT, { &wanted_stream[AVMEDIA_TYPE_VIDEO] }, "select desired video stream", "stream_number"
},
    { "sst", OPT_INT | HAS_ARG | OPT_EXPERT, { &wanted_stream[AVMEDIA_TYPE_SUBTITLE] }, "select desired subtitle stream", "stream_nu
mber" },
    { "ss", HAS_ARG, { (void*) opt_seek }, "seek to a given position in seconds", "pos" },
    { "t", HAS_ARG, { (void*) opt_duration }, "play  \"duration\" seconds of audio/video", "duration" },
    //选项众多，不再一一列出…
};
```

选项众多，简单举几个例子：
强行设置设置屏幕的宽度选项（“-x”选项）：

```cpp
{ "x", HAS_ARG, { (void*) opt_width }, "force displayed width", "width" }
```

从代码中可以看出，“-x”选项包含选项值（HAS_ARG），选项处理函数是opt_width()。选项说明是"force displayed width"。opt_width()的内容如下：

```cpp
static int opt_width(void *optctx, const char *opt, const char *arg)
{
    screen_width = parse_number_or_die(opt, arg, OPT_INT64, 1, INT_MAX);
    return 0;
}
```

可以看出其作用是解析输入的字符串为整数并赋值给全局变量screen_width。

全屏（“-fs”选项）

```cpp
{ "fs", OPT_BOOL, { &is_full_screen }, "force full screen" }
```

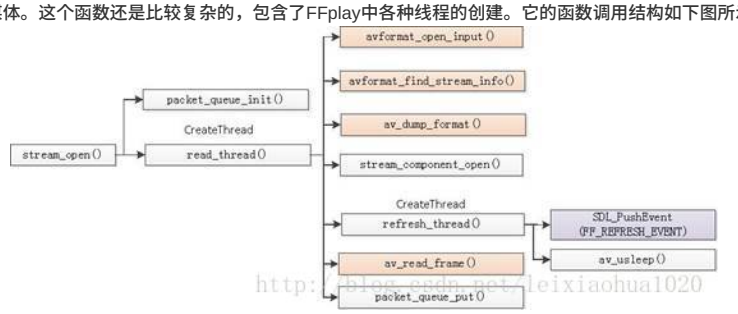
从代码中可以看出，“-fs”选项包含布尔型选项值（OPT_BOOL），并绑定了全局变量is_full_screen。选项说明是"force full screen"。

SDL_Init()

SDL_Init()用于初始化SDL。FFplay中视频的显示和声音的播放都用到了SDL。

stream_open()

stream_open()的作用是打开输入的媒体。这个函数还是比较复杂的，包含了FFplay中各种线程的创建。它的函数调用结构如下图所示。



stream_open()调用了如下函数：
packet_queue_init()：初始化各个PacketQueue（视频/音频/字幕）
read_thread()：读取媒体信息线程。

read_thread()

read_thread()调用了如下函数：

avformat_open_input()：打开媒体。
avformat_find_stream_info()：获得媒体信息。
av_dump_format()：输出媒体信息到控制台。
stream_component_open()：分别打开视频/音频/字幕解码线程。
refresh_thread()：视频刷新线程。
av_read_frame()：获取一帧压缩编码数据（即一个AVPacket）。
packet_queue_put()：根据压缩编码数据类型的不同（视频/音频/字幕），放到不同的PacketQueue中。

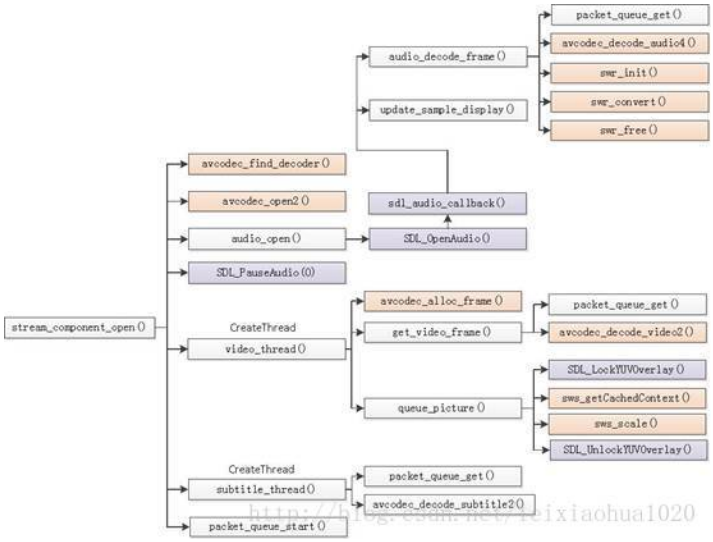
refresh_thread()

refresh_thread()调用了如下函数：

SDL_PushEvent(FF_REFRESH_EVENT)：发送FF_REFRESH_EVENT的SDL_Event
av_usleep()：每两次发送之间，间隔一段时间。

stream_component_open()

stream_component_open()用于打开视频/音频/字幕解码的线程。其函数调用关系如下图所示。



stream_component_open()调用了如下函数：
avcodec_find_decoder()：获得解码器。
avcodec_open2()：打开解码器。
audio_open()：打开音频解码。

SDL_PauseAudio(0)：SDL中播放音频的函数。
video_thread()：创建视频解码线程。
subtitle_thread()：创建字幕解码线程。
packet_queue_start()：初始化PacketQueue。

audio_open()调用了如下函数
SDL_OpenAudio()：SDL中打开音频设备的函数。注意它是根据SDL_AudioSpec参数打开音频设备。SDL_AudioSpec中的callback字段指定了音频播放的回调函数sdl_audio_callback()。当音频设备需要更多数据的时候，会调用该回调函数。因此该函数是会被反复调用的。

下面来看一下SDL_AudioSpec中指定的回调函数sdl_audio_callback()。
sdl_audio_callback()调用了如下函数
audio_decode_frame()：解码音频数据。
update_sample_display()：当不显示视频图像，而是显示音频波形的时候，调用此函数。

audio_decode_frame()调用了如下函数
packet_queue_get()：获取音频压缩编码数据（一个AVPacket）。
avcodec_decode_audio4()：解码音频压缩编码数据（得到一个AVFrame）。
swr_init()：初始化libswresample中的SwrContext。libswresample用于音频采样采样数据（PCM）的转换。
swr_convert()：转换音频采样率到适合系统播放的格式。
swr_free()：释放SwrContext。

video_thread()调用了如下函数
avcodec_alloc_frame()：初始化一个AVFrame。
get_video_frame()：获取一个存储解码后数据的AVFrame。
queue_picture()：

get_video_frame()调用了如下函数
packet_queue_get()：获取视频压缩编码数据（一个AVPacket）。
avcodec_decode_video2()：解码视频压缩编码数据（得到一个AVFrame）。

queue_picture()调用了如下函数
SDL_LockYUVOverlay()：锁定一个SDL_Overlay。
sws_getCachedContext()：初始化libswscale中的SwsContext。Libswscale用于图像的Raw格式数据（YUV，RGB）之间的转换。注意sws_getCachedContext()和sws_getContext()功能是一致的。
sws_scale()：转换图像数据到适合系统播放的格式。
SDL_UnlockYUVOverlay()：解锁一个SDL_Overlay。

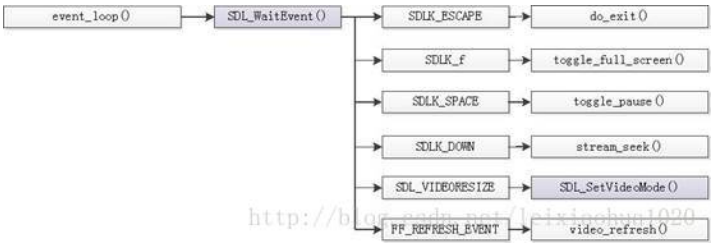
subtitle_thread()调用了如下函数
packet_queue_get()：获取字幕压缩编码数据（一个AVPacket）。
avcodec_decode_subtitle2()：解码字幕压缩编码数据。

event_loop()

FFplay再打开媒体之后，便会进入event_loop()函数，永远不停的循环下去。该函数用于接收并处理各种各样的消息。有点像Windows的消息循环机制。
PS：该循环确实是无止尽的，其形式为如下

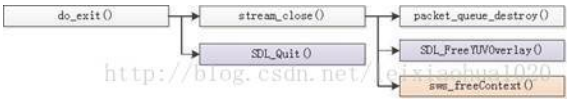
```cpp
SDL_Event event;
for (;;) {
    SDL_WaitEvent(&event);
        switch (event.type) {
        case SDLK_ESCAPE:
        case SDLK_q:
                do_exit(cur_stream);
                break;
        case SDLK_f:
        …
        …
        }
}
```

event_loop()函数调用关系如下所示。



根据event_loop()中SDL_WaitEvent()接收到的SDL_Event类型的不同，会调用不同的函数进行处理（从编程的角度来说就是一个switch()语法）。图中仅仅列举了几个例子：
SDLK_ESCAPE（按下“ESC”键）：do_exit()。退出程序。
SDLK_f（按下“f”键）：toggle_full_screen()。切换全屏显示。
SDLK_SPACE（按下“空格”键）：toggle_pause()。切换“暂停”。
SDLK_DOWN（按下鼠标键）：stream_seek()。跳转到指定的时间点播放。
SDL_VIDEORESIZE（窗口大小发生变化）：SDL_SetVideoMode()。重新设置宽高。
FF_REFRESH_EVENT（视频刷新事件（自定义事件））：video_refresh()。刷新视频。

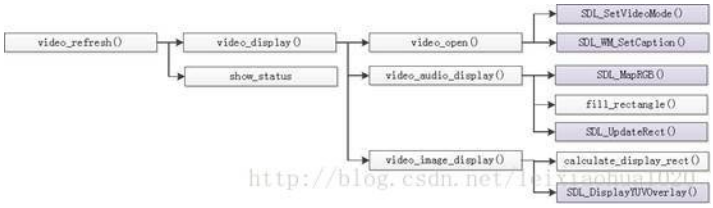
下面分析一下do_exit()函数。该函数用于退出程序。函数的调用关系如下图所示。



do_exit()函数调用了以下函数
stream_close()：关闭打开的媒体。
SDL_Quit()：关闭SDL。

stream_close()函数调用了以下函数
packet_queue_destroy()：释放PacketQueue。
SDL_FreeYUVOverlay()：释放SDL_Overlay。
sws_freeContext()：释放SwsContext。

下面重点分析video_refresh()函数。该函数用于将图像显示到显示器上。函数的调用关系如下图所示。



video_refresh()函数调用了以下函数
video_display()：显示像素数据到屏幕上。
show_status：这算不上是一个函数，但是是一个独立的功能模块，因此列了出来。该部分打印输出播放的状态至屏幕上。如下图所示。

video_display()函数调用了以下函数
video_open()：初始化的时候调用，打开播放窗口。
video_audio_display()：显示音频波形图（或者频谱图）的时候调用。里面包含了不少画图操作。
video_image_display()：显示视频画面的时候调用。

video_open()函数调用了以下函数
SDL_SetVideoMode()：设置SDL_Surface（即SDL最基础的黑色的框）的大小等信息。
SDL_WM_SetCaption()：设置SDL_Surface对应窗口的标题文字。

video_audio_display()函数调用了以下函数
SDL_MapRGB()：获得指定（R，G，B）以及SDL_PixelFormat的颜色数值。例如获得黑色的值，作为背景。（R，G，B）为（0x00，0x00，0x00）。
fill_rectangle()：将指定颜色显示到屏幕上。
SDL_UpdateRect()：更新屏幕。

video_image_display()函数调用了以下函数
calculate_display_rect()：计算显示画面的位置。当拉伸了SDL的窗口的时候，可以让其中的视频保持纵横比。
SDL_DisplayYUVOverlay()：显示画面至屏幕。

文章标签： ffplay 源代码 函数 ffmpeg

个人分类： FFMPEG

所属专栏： FFmpeg