



## 一.Mplayer支持的格式

MPlayer是一个LINUX下的视频播放器，它支持相当多的媒体格式，无论在音频播放还是在视频播放方面，可以说它支持的格式是相当全面的。

视频格式支持：MPEG、AVI、ASF 与WMV、QuickTime 与 OGG/OGM、SDP、PVA、GIF。

音频格式支持：MP3、WAV、OGG/OGM 文件(Vorbis)、WMA 与 ASF、MP4、CD音频、XMMS。

## 二. Mplayer 中头文件的功能分析



```
[cpp]
1.  config.h // 各种本地配置宏定义头
2.  version.h // 版本定义头 #define VERSION "1.0pre7try2-3.4.2"
3.  mp_msg.h // 消息处理头
4.  help_mp.h // 根据配置自动生成的帮助头 #include "help/help_mpen.h"
5.  cfg-mplayer-def.h // Mplayer 运行时的选项缺省值头文件 char*
6.  default_config =
7.  sub_reader.h // 拥有格式自动发现功能的字幕(subtitle)阅读器
8.  libvo/video_out.h // 该文件包含 libvo 视频输出的公共函数、变量
9.  libvo/font_load.h // 有关字体装载的例程
10. libao2/audio_out.h // 音频输出驱动程序相关结构定义和全局数据
11. libmpcodecs/dec_audio.h // 音频解码
12. libmpcodecs/dec_video.h // 视频解码
13. libmpdemux/matroska.h // 多路解复用，媒体容器格式 matroska 处理头
14. libmpdemux/stream.h // 流处理
15. libmpdemux/demuxer.h // 多路解复用头文件
16. libmpdemux/sheader.h // 媒体流头处理
17. get_path.c // 路径获取头文件
18. spudec.h // SPU 子画面单元头，DVD 字幕流
19. edl.h // 剪辑控制清单
20. m_option.h // 选项类型处理头
21. m_config.h // 配置处理头文件
```

## 三. MPlayer.main 主流程简要说明



```
[cpp]
1.  int main() {
2.  1) 变量声明，电影信息 movie info:
3.  2) 初始化，消息系统.....
4.  play_next_file:
5.  3)播放文件 filename 的循环 goto play_next_file 开始
6.  main:
7.  4) 主处理 main
8.  5) 播放真正主循环 2010 ~3541 while (!eof)
9.  while (!eof) {
10.  5.1) 播放音频 PLAY AUDIO 2017 ~ 2064 decode_audio(sh_audio, ...);
11.  5.2) 播放视频 PLAY VIDEO, 2068 ~ 2300 decode_video(sh_video, ...);
12.  5.3) 处理暂停 PAUSE
13.  5.4) 处理 EDL
14.  5.5) 键盘事件处理，搜索2400~3216 while (!brk_cmd &&
15.  (cmd=mp_input_get_cmd(0,0,0))!=NULL)
16.  5.6) 时间寻道(秒) if (seek_to_sec)
17.  5.7) 寻道 3243 ~ 3306, if (rel_seek_secs || abs_seek_pos)
18.  5.8) 处理 GUI
19.  5.9) 变更 Update OSD
20.  5.10) 找到字幕 find sub
21.  5.11) 处理 X11 窗口
22.  5.12) DVD 字幕 sub:
23.  }
24.  goto_next_file:
25.  6) 播放结束，转到下个文件 goto_next_file:
26.  }
```

## 四.Mplayer源码分析

从Mplayer.c的main开始处理参数

```
[cpp]    
1. mconfig = m_config_new();  
2. m_config_register_options(mconfig,mplayer_opts);  
3. // TODO : add something to let modules register their options  
4. mp_input_register_options(mconfig);  
5. parse_cfgfiles(mconfig);
```

初始化mpctx结构体，mpctx应该是mplayer context的意思，顾名思义是一个统筹全局的变量。

```
[cpp]    
1. static MPContext *mpctx = &mpctx_s;  
2. // Not all functions in mplayer.c take the context as an argument yet  
3. static MPContext mpctx_s = {  
4. .osd_function = OSD_PLAY,  
5. .begin_skip = MP_NOPTS_VALUE,  
6. .play_tree_step = 1,  
7. .global_sub_pos = -1,  
8. .set_of_sub_pos = -1,  
9. .file_format = DEMUXER_TYPE_UNKNOWN,  
10. .loop_times = -1,  
11. #ifdef HAS_DVBIN_SUPPORT  
12. .last_dvb_step = 1,  
13. #endif  
14. };
```

原型

```

1. //真正统筹全局的结构
2. typedef struct MPContext {
3.     int osd_show_percentage;
4.     int osd_function;
5.     const ao_functions_t *audio_out;
6.     play_tree_t *playtree;
7.     play_tree_iter_t *playtree_iter;
8.     int eof;
9.     int play_tree_step;
10.    int loop_times;
11.
12.    stream_t *stream;
13.    demuxer_t *demuxer;
14.    sh_audio_t *sh_audio;
15.    sh_video_t *sh_video;
16.    demux_stream_t *d_audio;
17.    demux_stream_t *d_video;
18.    demux_stream_t *d_sub;
19.    mixer_t mixer;
20.    const vo_functions_t *video_out;
21.    // Frames buffered in the vo ready to flip. Currently always 0 or 1.
22.    // This is really a vo variable but currently there's no suitable vo
23.    // struct.
24.    int num_buffered_frames;
25.
26.    // used to retry decoding after startup/seeking to compensate for codec delay
27.    int startup_decode_retry;
28.    // how long until we need to display the "current" frame
29.    float time_frame;
30.
31.    // AV sync: the next frame should be shown when the audio out has this
32.    // much (in seconds) buffered data left. Increased when more data is
33.    // written to the ao, decreased when moving to the next frame.
34.    // In the audio-only case used as a timer since the last seek
35.    // by the audio CPU usage meter.
36.    double delay;
37.
38.    float begin_skip; ///< start time of the current skip while on edlout mode
39.    // audio is muted if either EDL or user activates mute
40.    short edl_muted; ///< Stores whether EDL is currently in muted mode.
41.    short user_muted; ///< Stores whether user wanted muted mode.
42.
43.    int global_sub_size; // this encompasses all subtitle sources
44.    int global_sub_pos; // this encompasses all subtitle sources
45.    int set_of_sub_pos;
46.    int set_of_sub_size;
47.    int sub_counts[SUB_SOURCES];
48. #ifdef CONFIG_ASS
49.    // set_of_ass_tracks[i] contains subtitles from set_of_subtitles[i]
50.    // parsed by libass or NULL if format unsupported
51.    ASS_Track* set_of_ass_tracks[MAX_SUBTITLE_FILES];
52. #endif
53.    sub_data* set_of_subtitles[MAX_SUBTITLE_FILES];
54.
55.    int file_format;
56.
57. #ifdef CONFIG_DVBIN
58.    int last_dvb_step;
59.    int dvbin_reopen;
60. #endif
61.
62.    int was_paused;
63.
64. #ifdef CONFIG_DVDNAV
65.    struct mp_image *nav_smpi; ///< last decoded dvdnav video image
66.    unsigned char *nav_buffer; ///< last read dvdnav video frame
67.    unsigned char *nav_start;  ///< pointer to last read video buffer
68.    int nav_in_size;          ///< last read size
69. #endif
70. } MPContext;

```

一些GUI相关的操作

打开字幕流

打开音视频流

```

1. mpctx->stream=open_stream(filename,0,&mpctx->file_format);
2. fileformat 文件还是TV 流DEMUXER_TYPE_PLAYLIST 或DEMUXER_TYPE_UNKNOWN
3. DEMUXER_TYPE_TV
4. current_module记录状态vobsub open_stream handle_playlist dumpstream
5. stream_reset(mpctx->stream);
6. stream_seek(mpctx->stream,mpctx->stream->start_pos);
7. f=fopen(stream_dump_name,"wb"); dump文件流
8. stream->type==STREAMTYPE_DVD

```

//===== Open DEMUXERS — DETECT file type =====

Demux. 分离视频流和音频流

```
[cpp]
1. mpctx->demuxer=demux_open(mpctx->stream,mpctx-
2. >file_format,audio_id,video_id,dvdsb_id,filename);
3. Demux过程
4. demux_open
5. get_demuxer_type_from_name
6. ....
7. mpctx->d_audio=mpctx->demuxer->audio;
8. mpctx->d_video=mpctx->demuxer->video;
9. mpctx->d_sub=mpctx->demuxer->sub;
10. mpctx->sh_audio=mpctx->d_audio->sh;
11. mpctx->sh_video=mpctx->d_video->sh;
```

分离了之后就开始分别Play audio和video

这里只关心play video

```
[cpp]
1. /*===== PLAY VIDEO =====*/
2. vo_pts=mpctx->sh_video->timer*90000.0;
3. vo_fps=mpctx->sh_video->fps;
4. if (!mpctx->num_buffered_frames) {
5.     double frame_time = update_video(&blit_frame);
6.     mp_dbg(MSGT_AVSYN,MSGT_DBG2,"*** ftime=%5.3f ***\n",frame_time);
7.     if (mpctx->sh_video->vf_initd < 0) {
8.         mp_msg(MSGT_CPLAYER,MSGT_FATAL, MSGTR_NotInitializeVOPorV0);
9.         mpctx->eof = 1; goto goto_next_file;
10.    }
11.    if (frame_time < 0)
12.        mpctx->eof = 1;
13.    else {
14.        // might return with !eof && !blit_frame if !correct_pts
15.        mpctx->num_buffered_frames += blit_frame;
16.        time_frame += frame_time / playback_speed; // for nosound
17.    }
18. }
```

关键的函数是update\_video根据pts是否正确调整一下同步并在必要的时候丢帧处理。最终调用decode\_video开始解码（包括generate\_video\_frame里）。mpi = mpvdec  
c->decode(sh\_video, start, in\_size, drop\_frame);mpvdec是在main里通过reinit\_video\_chain的一系列调用动态选定的解码程序。其实就一结构体。它的原型是

```
[cpp]
1. typedef struct vd_functions_s
2. {
3.     vd_info_t *info;
4.     int (*init)(sh_video_t *sh);
5.     void (*uninit)(sh_video_t *sh);
6.     int (*control)(sh_video_t *sh,int cmd,void* arg, ...);
7.     mp_image_t* (*decode)(sh_video_t *sh,void* data,int len,int flags);
8. } vd_functions_t;
```

这是所有解码器必须实现的接口。

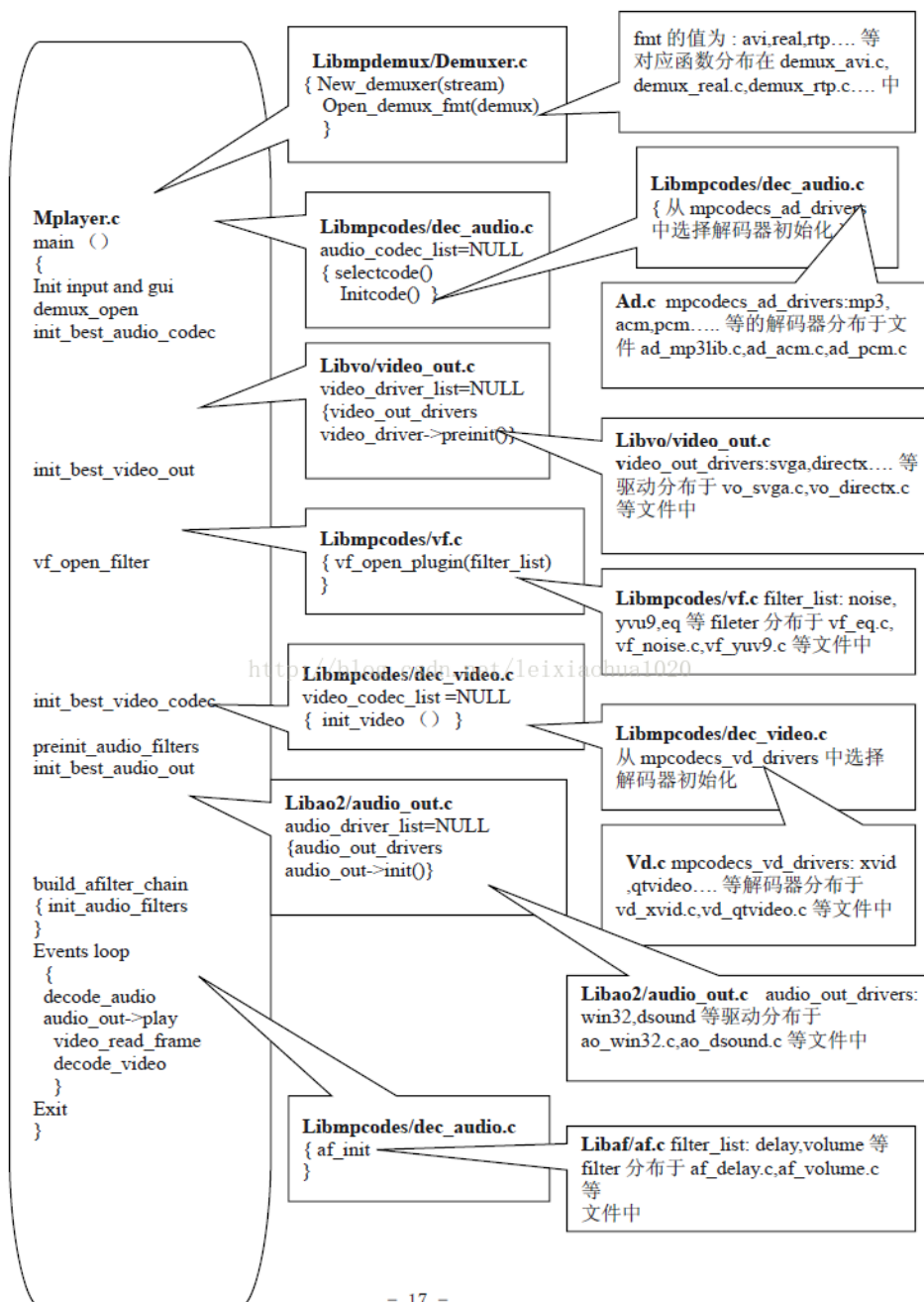
int (\*init)(sh\_video\_t \*sh);是一个名为init的指针，指向一个接受sh\_video\_t \*类型参数，并返回int类型值的函数地址。那些vd\_开头的文件都是解码相关的。随便打开一个vd文件以上几个函数和info变量肯定都包含了。mpi被mplayer用来存储解码后的图像。在mp\_image.h里定义。

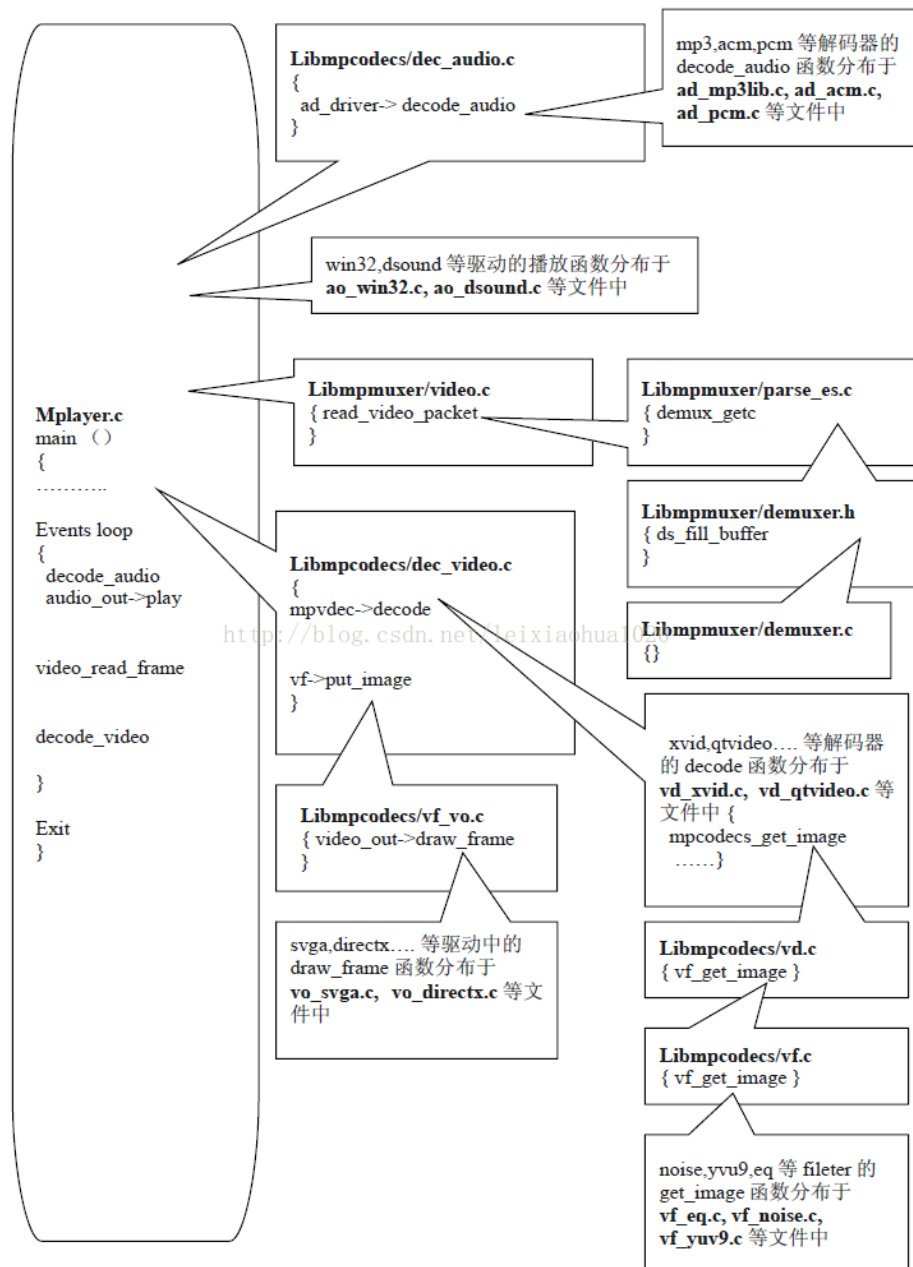
```
[cpp]
1. typedef struct mp_image_s {
2.     unsigned short flags;
3.     unsigned char type;
4.     unsigned char bpp; // bits/pixel. NOT depth! for RGB it will be n*8
5.     unsigned int imgfmt;
6.     int width,height; // stored dimensions
7.     int x,y,w,h; // visible dimensions
8.     unsigned char* planes[MP_MAX_PLANES];
9.     int stride[MP_MAX_PLANES];
10.    char * qscale;
11.    int qstride;
12.    int pict_type; // 0->unknown, 1->I, 2->P, 3->B
13.    int fields;
14.    int qscale_type; // 0->mpeg1/4/h263, 1->mpeg2
15.    int num_planes;
16.    /* these are only used by planar formats Y,U(Cb),V(Cr) */
17.    int chroma_width;
18.    int chroma_height;
19.    int chroma_x_shift; // horizontal
20.    int chroma_y_shift; // vertical
21.    /* for private use by filter or vo driver (to store buffer id or dmpi) */
22.    void* priv;
23. } mp_image_t;
```

图像在解码以后会输出到显示器，mplayer本来就是一个视频播放器么。但也有可能作为输入提供给编码器进行二次编码，MP附带的mencoder.ex

e就是专门用来编码的。在这之前可以定义filter对图像进行处理，以实现各种效果。所有以vf\_开头的文件，都是这样的filter。图像的显示是通过vo，即video\_out来实现的。解码器只负责把解码完成的帧传给vo，怎样显示就不用管了。这也是平台相关性最大的部分，单独分出来的好处是不言而喻的，像在Windows下有通过directx实现的vo，Linux下有输出到X的vo。vo\_\*文件是各种不同的vo实现，只是他们不都是以显示为目的，像vo\_md5sum.c只是计算一下图像的md5值。在解码完成以后，即得到mpi以后，filter\_video被调用，其结果是整个filter链上的所有filter都被调用了一遍，包括最后的VO，在vo的put\_image里把图像输出到显示器。这个时候需要考虑的是图像存储的方法即用哪种色彩空间。

附上两张MPlayer结构图：





MPLayer源代码下载地址：<http://download.csdn.net/detail/leixiaohua1020/6374337>

文章标签：[mplayer](#) [源代码](#) [分析](#)

个人分类：[Mplayer](#)

所属专栏：[开源多媒体项目源代码分析](#)

此PDF由spyyg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com