原 ffplay for mfc 代码备忘

2014年07月10日 00:49:22　阅读数：12009

之前上传了一个开源播放器工程ffplay for mfc。它将ffmpeg项目中的ffplay播放器(ffplay.c)移植到了VC的环境下，并且使用MFC做了一套界面。它可以完成一个播放器播放视频的基本流程：解协议，解封装，视频/音频解码，视音频同步，视音频输出。此外还包含一些控制功能：播放，暂停/继续，前进，后退，停止，逐帧播放，全屏等；以及一些码流分析功能：视频解码分析和音频解码分析。

详细的软件使用就不仔细介绍了，本文简单介绍其中比较重要的模块的流程。以防长时间不看的话忘了~

软件信息：

ffplay播放器移植VC的工程：ffplay for MFC

SourceForge项目主页：
https://sourceforge.net/projects/ffplayformfc/

1. 软件结构

软件结构如图1所示，包含如下模块：控制，视频播放，参数提取，码流分析。其中，视频播放模块用于视频的解码和播放；控制模块用于控制视频的播放；参数提取模块用于提取显示视频的各种参数；码流分析模块伴随着视频的播放分析视音频流中的参数。

图1.软件结构

2.模块说明

2.1. 视频播放模块

视频播放模块的作用就是将网络上（或者是本地）的视音频数据接收下来经过一系列处理后最终输出到视音频设备上。根据处理的顺序不同，它可以分为以下几个子模块：

1) 解协议模块

2) 解封装模块

3) 视频解码模块

4) 音频解码模块

5) 视音频同步模块

视频播放模块的流程图如图2所示。按照处理的顺序分为解协议，解封装，视频解码，音频解码，视音频同步。

详细的原理在文章 [总结]视音频编解码技术零基础学习方法 中有详细的说明，在这里不再重复，示意图如下所示。

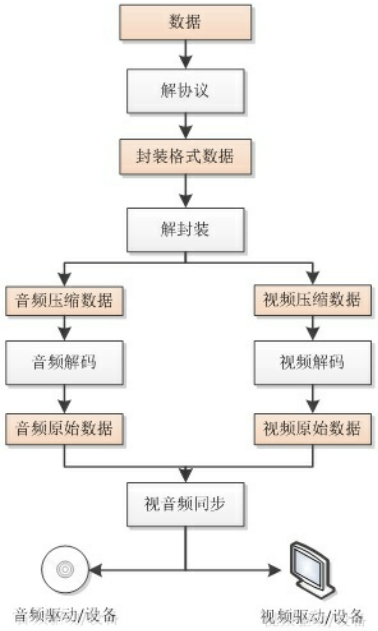

图2.视频播放模块流程

这一模块主要是通过对ffplay.c改写而得到的。改写完成后为ffplaycore.cpp。

简单的代码方面的流程可以参考：

100行代码实现最简单的基于FFMPEG+SDL的视频播放器

比较完整的代码方面的流程可以参考：

FFplay源代码分析：整体流程图

2.2. 控制模块

控制模块的作用就是控制视频的播放。包含以下几种功能：

1) 开始

2) 暂停/继续

3) 快进/快退

4) 逐帧播放

5) 调整窗口大小

6) 全屏

7) 调整播放进度

软件开始解码视频数据之后，会进入一个函数event_loop()。该函数内部不停地循环，使用SDL_WaitEvent()等待着响应系统的消息。接收到消息之后，根据消息类型的不同，做出不同的响应。如图3所示例举了6种不同的消息对应的不同的响应：

1) SDLK_ESCAPE。对应键盘上“Esc”键的响应。功能是退出程序。

2) SDLK_SPACE。对应键盘上“空格”键的响应。功能是暂停播放。

3) SDL_MOUSEBUTTONDOWN。对应鼠标单击的响应。功能是调整视频播放进度。

4) SDL_VIDEORESIZE。对应“VideoResize”消息的响应。功能是调整播放窗口的大小。

5) FF_REFRESH_EVENT。对应自定义消息“FF_REFRESH_EVENT”的响应。功能是刷新视频画面。

6) FFMFC_SEEK_BAR_EVENT。对应自定义消息“FFMFC_SEEK_BAR_EVENT”的响应。功能是调整视频播放进度条。

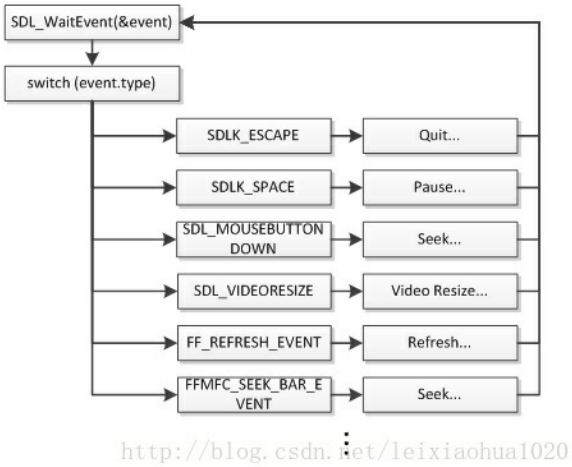

图3.控制模块流程（消息循环）

event_loop()函数代码如下：

```cpp
/* handle an event sent by the GUI */
//处理各种鼠标键盘命令,包括各种事件
static void event_loop(VideoState *cur_stream)
{
    SDL_Event event;
    double incr, pos, frac;

    for (;;) {

        double x;
        //判断退出-------
        if(exit_remark==1)
            break;
        //---------------
        if (cur_stream->abort_request)
            break;

        SDL_WaitEvent(&event);
        switch (event.type) {
        case SDL_KEYDOWN:
            if (exit_on_keydown)
            {
                do_exit(cur_stream);
                break;
            }
            switch (event.key.keysym.sym) {
            case SDLK_ESCAPE:
            case SDLK_q:
                do_exit(cur_stream);
                break;
            case SDLK_f:
                //全屏
                toggle_full_screen(cur_stream);
                cur_stream->force_refresh = 1;
                break;
            case SDLK_p:
                //暂停
            case SDLK_SPACE:
                toggle_pause(cur_stream);
                break;
            case SDLK_s: // S: Step to next frame
                step_to_next_frame(cur_stream);
                break;
            case SDLK_a:
                stream_cycle_channel(cur_stream, AVMEDIA_TYPE_AUDIO);
                break;
            case SDLK_v:
                stream_cycle_channel(cur_stream, AVMEDIA_TYPE_VIDEO);
                break;
            case SDLK_t:
                stream_cycle_channel(cur_stream, AVMEDIA_TYPE_SUBTITLE);
                break;
                //修改了一下，三中显示模式分成了三个键
            case SDLK_w:
                toggle_audio_display(cur_stream,SHOW_MODE_VIDEO);
                cur_stream->force_refresh = 1;
                break;
            case SDLK_e:
                toggle_audio_display(cur_stream,SHOW_MODE_WAVES);
                cur_stream->force_refresh = 1;
                break;
            case SDLK_r:
                toggle_audio_display(cur_stream,SHOW_MODE_RDFT);
```

```
                cur_stream->force_refresh = 1;
                break;
            case SDLK_y:
                cur_stream->v_show_mode=SHOW_MODE_Y;
                break;
            case SDLK_PAGEUP:
                incr = 600.0;
                goto do_seek;
            case SDLK_PAGEDOWN:
                incr = -600.0;
                goto do_seek;
                //左方向键
            case SDLK_LEFT:
                incr = -10.0;
                goto do_seek;
            case SDLK_RIGHT:
                incr = 10.0;
                goto do_seek;
            case SDLK_UP:
                incr = 60.0;
                goto do_seek;
            case SDLK_DOWN:
                incr = -60.0;
do_seek:
                if (seek_by_bytes) {
                    if (cur_stream->video_stream >= 0 && cur_stream->video_current_pos >= 0) {
                        pos = cur_stream->video_current_pos;
                    } else if (cur_stream->audio_stream >= 0 && cur_stream->audio_pkt.pos >= 0) {
                        pos = cur_stream->audio_pkt.pos;
                    } else
                        pos = avio_tell(cur_stream->ic->pb);
                    if (cur_stream->ic->bit_rate)
                        incr *= cur_stream->ic->bit_rate / 8.0;
                    else
                        incr *= 180000.0;
                    pos += incr;
                    stream_seek(cur_stream, pos, incr, 1);
                } else {
                    pos = get_master_clock(cur_stream);
                    pos += incr;
                    stream_seek(cur_stream, (int64_t)(pos * AV_TIME_BASE), (int64_t)(incr * AV_TIME_BASE), 0);
                }
                break;
            default:
                break;
            }
            break;
        case SDL_VIDEOEXPOSE:
            cur_stream->force_refresh = 1;
            break;
            //鼠标单击
        case SDL_MOUSEBUTTONDOWN:
            if (exit_on_mousedown) {
                do_exit(cur_stream);
                break;
            }
        case SDL_MOUSEMOTION:
            if (event.type == SDL_MOUSEBUTTONDOWN) {
                x = event.button.x;
            } else {
                if (event.motion.state != SDL_PRESSED)
                    break;
                x = event.motion.x;
            }
            if (seek_by_bytes || cur_stream->ic->duration <= 0) {
                uint64_t size =  avio_size(cur_stream->ic->pb);
                stream_seek(cur_stream, size*x/cur_stream->width, 0, 1);
            } else {
                int64_t ts;
                int ns, hh, mm, ss;
                int tns, thh, tmm, tss;
                tns  = cur_stream->ic->duration / 1000000LL;
                thh  = tns / 3600;
                tmm  = (tns % 3600) / 60;
                tss  = (tns % 60);
                frac = x / cur_stream->width;
                ns   = frac * tns;
                hh   = ns / 3600;
                mm   = (ns % 3600) / 60;
                ss   = (ns % 60);
                fprintf(stderr, "Seek to %2.0f%% (%2d:%02d:%02d) of total duration (%2d:%02d:%02d)       \n", frac*100,
                    hh, mm, ss, thh, tmm, tss);
                ts = frac * cur_stream->ic->duration;
                if (cur_stream->ic->start_time != AV_NOPTS_VALUE)
                    ts += cur_stream->ic->start_time;
                stream_seek(cur_stream, ts, 0, 0);
            }
            break;
        case SDL_VIDEORESIZE:
            screen = SDL_SetVideoMode(event.resize.w, event.resize.h, 0,
                SDL_HWSURFACE|SDL_RESIZABLE|SDL_ASYNCBLIT|SDL_HWACCEL);
```

```
            screen_width  = cur_stream->width  = event.resize.w;
            screen_height = cur_stream->height = event.resize.h;
            cur_stream->force_refresh = 1;
            break;
        case SDL_QUIT:
        case FF_QUIT_EVENT:
            do_exit(cur_stream);
            break;
        case FF_ALLOC_EVENT:
            alloc_picture((VideoState *)(event.user.data1));
            break;
        case FF_REFRESH_EVENT:
            video_refresh(event.user.data1);
            cur_stream->refresh = 0;
            break;
        case FFMFC_SEEK_BAR_EVENT:{
            if (seek_by_bytes || cur_stream->ic->duration <= 0) {
                uint64_t size =  avio_size(cur_stream->ic->pb);
                stream_seek(cur_stream, size*seek_bar_pos/1000, 0, 1);
            } else {
                int64_t ts;
                frac=(double)seek_bar_pos/1000;
                ts = frac * cur_stream->ic->duration;
                if (cur_stream->ic->start_time != AV_NOPTS_VALUE)
                    ts += cur_stream->ic->start_time;
                stream_seek(cur_stream, ts, 0, 0);
            }
            break;
                            }

        default:
            break;
        }
    }
}
```

控制模块的各个功能函数，只需要设置一定内容的消息，再发送出去，就可以完成相应的控制功能。如图4所示，分别例举了3种控制功能的完成方式。

1) “暂停”功能，发送SDLK_SPACE消息。

2) “调整窗口大小”功能。发送VIDEORESIZE消息，并附带窗口的大小。

3) “调整视频播放进度条”功能。发送FFMFC_SEEK_BAR_EVENT消息。

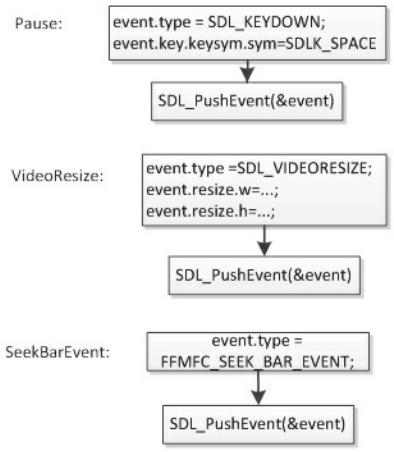

图4.控制模块流程（发送消息）

各个功能函数的代码如下：

```cpp
//发送“全屏”命令
//Send Command "FullScreen"
void ffmfc_play_fullcreen(){
    SDL_Event event;
    event.type = SDL_KEYDOWN;
    event.key.keysym.sym=SDLK_f;
    SDL_PushEvent(&event);
}

//发送“暂停”命令
//Send Command "Pause"
void ffmfc_play_pause(){
    SDL_Event event;
    event.type = SDL_KEYDOWN;
    event.key.keysym.sym=SDLK_p;
```

```
    SDL_PushEvent(&event);
}

//发送“逐帧”命令
//Send Command "Step"
void ffmfc_seek_step(){
    SDL_Event event;
    event.type = SDL_KEYDOWN;
    event.key.keysym.sym=SDLK_s;
    SDL_PushEvent(&event);
}

//发送“宽高比”命令
//Send Command "AspectRatio"
void ffmfc_aspectratio(int num,int den){
    int w=g_is->width;
    int h=g_is->height;
    int w_re=h*num/den;
    SDL_Event event;
    event.type = SDL_VIDEORESIZE;
    event.resize.w=w_re;
    event.resize.h=h;
    SDL_PushEvent(&event);
}

//发送“大小”命令
//Send Command "WindowSize"
void ffmfc_size(int percentage){
    int w=g_is->ic->streams[g_is->video_stream]->codec->width;
    int h=g_is->ic->streams[g_is->video_stream]->codec->height;
    SDL_Event event;
    event.type = SDL_VIDEORESIZE;
    event.resize.w=w*percentage/100;
    event.resize.h=h*percentage/100;
    SDL_PushEvent(&event);
}

//发送“窗口画面内容”命令
//Send Command "Audio Display Mode"
void ffmfc_audio_display(int mode){

    SDL_Event event;
    event.type = SDL_KEYDOWN;
    switch(mode){
    case 0:event.key.keysym.sym=SDLK_w;break;
    case 1:event.key.keysym.sym=SDLK_e;break;
    case 2:event.key.keysym.sym=SDLK_r;break;
    }
    SDL_PushEvent(&event);
}

//发送“前进/后退”命令
//Send Command "Seek"
void ffmfc_seek(int time){
    SDL_Event event;
    event.type = SDL_KEYDOWN;
    switch (time){
    case -10 :event.key.keysym.sym=SDLK_LEFT;break;
    case 10 :event.key.keysym.sym=SDLK_RIGHT;break;
    case -60 :event.key.keysym.sym=SDLK_DOWN;break;
    case 60 :event.key.keysym.sym=SDLK_UP;break;
    case -600 :event.key.keysym.sym=SDLK_PAGEDOWN;break;
    case 600 :event.key.keysym.sym=SDLK_PAGEUP;break;
    default :event.key.keysym.sym=SDLK_RIGHT;break;
    }
    SDL_PushEvent(&event);
}

//播放进度
//Seek Bar
void ffmfc_seek_bar(int pos){
    SDL_Event event;
    event.type = FFMFC_SEEK_BAR_EVENT;
    seek_bar_pos=pos;
    SDL_PushEvent(&event);
}
```

MFC中按钮，进度条控件的消息响应函数只要调用以上功能函数就可以实现相应的功能：

```cpp
void CffplaymfcDlg::OnBnClickedSeekB()
{
    ffmfc_seek(-60);
}


void CffplaymfcDlg::OnBnClickedPause()
{
    ffmfc_play_pause();
}


void CffplaymfcDlg::OnBnClickedSeekF()
{
    ffmfc_seek(60);
}


void CffplaymfcDlg::OnBnClickedStop()
{
    ffmfc_quit();
    SystemClear();
    ResetBtn();
}


void CffplaymfcDlg::OnBnClickedSeekStep()
{
    ffmfc_seek_step();
}


void CffplaymfcDlg::OnBnClickedFullscreen()
{
    ffmfc_play_fullcreen();
}
```

2.3.参数提取模块

参数提取模块的作用就是提取视频码流中的一部分参数。按照参数种类的不同，分为封装格式参数，视频编码参数，音频编码参数。

（1） 封装格式参数

封装格式参数指的是封装格式中包含的参数。包括：

1) 输入协议

2) 封装格式

3) 比特率

4) 时长

5) 元数据

（2）视频编码参数

视频编码参数指的是视频码流中的参数。包括：

1) 输出像素格式

2) 编码方式

3) 帧率

4) 画面大小

（3）音频编码参数

音频编码参数指的是音频码流中的参数。包括：

1) 采样率

2) 编码方式

3) 声道数

参数提取模块的流程图如图5所示。参数提取的功能在函数ffmfc_param_global()中实现。系统通过调用av_register_all()、avformat_open_input()等

一系列函数直到avcodec_open()函数完成初始化工作。初始化完成之后，系统调用ffmfc_param_global()完成参数提取功能。参数提取功能完成之后，系统循环调用函数av_read_frame()获取每帧压缩码流数据。

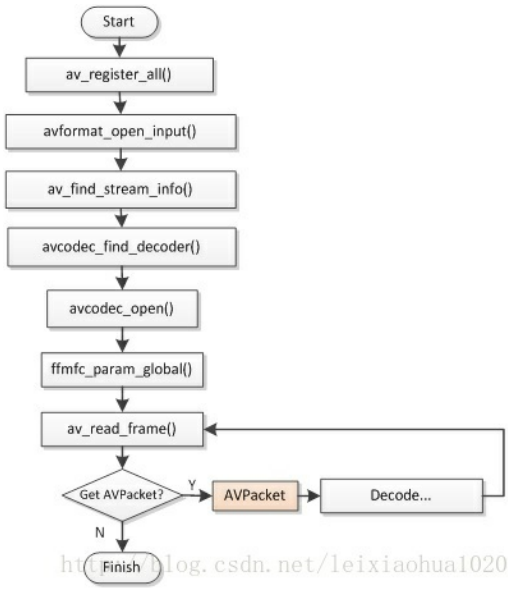

图5.参数提取模块流程

参数提取函数ffmfc_param_global()代码如下：

```cpp
//全局的，只设置一次
int ffmfc_param_global(VideoState *is){
    //初始化
    CString input_protocol,input_format,wxh,decoder_name,
        decoder_type,bitrate,extention,pix_fmt,framerate,timelong,decoder_name_au,sample_rate_au,channels_au;
    float framerate_temp,timelong_temp,bitrate_temp;
    //注意：把int等类型转换成LPCTSTR
    //CString可以直接赋值给LPCTSTR
    AVFormatContext *pFormatCtx = is->ic;
    int video_stream=is->video_stream;
    int audio_stream=is->audio_stream;
    AVCodecContext *pCodecCtx = pFormatCtx->streams[video_stream]->codec;
    AVCodecContext *pCodecCtx_au = pFormatCtx->streams[audio_stream]->codec;

    URLContext *uc=(URLContext *)pFormatCtx->pb->opaque;
    URLProtocol *up=(URLProtocol *)uc->prot;
    //输入文件的协议----------
    input_protocol.Format("%s",up->name);
    dlg->m_formatprotocol.SetWindowText(input_protocol);

    //视频解码参数，有视频的时候设置
    if(video_stream!=-1){
        wxh.Format("%d x %d",pCodecCtx->width,pCodecCtx->height);
        dlg->m_codecvresolution.SetWindowText(wxh);

        decoder_name.Format("%s",pCodecCtx->codec->long_name);
        dlg->m_codecvname.SetWindowText(decoder_name);
        //帧率显示还有问题
        framerate_temp=(pFormatCtx->streams[video_stream]->r_frame_rate.num)/(pFormatCtx->streams[video_stream]->r_frame_rate.den);

        framerate.Format("%5.2ffps",framerate_temp);
        dlg->m_codecvframerate.SetWindowText(framerate);

        switch(pCodecCtx->pix_fmt){
        case 0:
            pix_fmt.Format("YUV420P");break;
        case 1:
            pix_fmt.Format("YUYV422");break;
        case 2:
            pix_fmt.Format("RGB24");break;
        case 3:
            pix_fmt.Format("BGR24");break;
        case 12:
            pix_fmt.Format("PIX_FMT_YUVJ420P");break;
        default:
            pix_fmt.Format("UNKNOWN");
        }
        dlg->m_codecvpixfmt.SetWindowText(pix_fmt);
    }
    //音频解码参数，有音频的时候设置
    if(audio_stream!=-1){
        decoder_name_au.Format("%s",pCodecCtx_au->codec->long_name);
        dlg->m_codecaname.SetWindowText(decoder_name_au);
        sample_rate_au.Format("%d",pCodecCtx_au->sample_rate);
```

```
        dlg->m_codecasamplerate.SetWindowText(sample_rate_au);
        channels_au.Format("%d",pCodecCtx_au->channels);
        dlg->m_codecachannels.SetWindowText(channels_au);
    }
    //显示成以k为单位
    bitrate_temp=((float)(pFormatCtx->bit_rate))/1000;
    bitrate.Format("%5.2fkbps",bitrate_temp);
    dlg->m_formatbitrate.SetWindowText(bitrate);
    //duration是以微秒为单位
    timelong_temp=(pFormatCtx->duration)/1000000;
    //转换成hh:mm:ss形式
    int tns, thh, tmm, tss;
    tns  = (pFormatCtx->duration)/1000000;
    thh  = tns / 3600;
    tmm  = (tns % 3600) / 60;
    tss  = (tns % 60);
    timelong.Format("%02d:%02d:%02d",thh,tmm,tss);
    dlg->m_formatduration.SetWindowText(timelong);
    dlg->m_duration.SetWindowText(timelong);
    //输入文件的封装格式------
    input_format.Format("%s",pFormatCtx->iformat->long_name);
    dlg->m_formatinputformat.SetWindowText(input_format);
    //------------------------


    //bitrate.Format("%d",pCodecCtx->bit_rate);
    //dlg->m_bitrate.SetWindowText(bitrate);

    //MetaData---------------------------------------------------------
    //从AVDictionary获得
    //需要用到AVDictionaryEntry对象
    //CString author,copyright,description;
    CString meta=NULL,key,value;
    AVDictionaryEntry *m = NULL;
    //不用一个一个找出来
    /*  m=av_dict_get(pFormatCtx->metadata,"author",m,0);
    author.Format("作者：%s",m->value);
    m=av_dict_get(pFormatCtx->metadata,"copyright",m,0);
    copyright.Format("版权：%s",m->value);
    m=av_dict_get(pFormatCtx->metadata,"description",m,0);
    description.Format("描述：%s",m->value);
    */
    //使用循环读出
    //(需要读取的数据，字段名称，前一条字段（循环时使用），参数)
    while(m=av_dict_get(pFormatCtx->metadata,"",m,AV_DICT_IGNORE_SUFFIX)){
        key.Format(m->key);
        value.Format(m->value);
        meta+=key+"\t:"+value+"\r\n" ;
    }

    //EditControl换行用\n不行，需要使用\r\n
    //除了要用\r\n外，还要都CEdit 的属性进行设置：
    //Auto HScroll 设置为 False
    //MultiLine  设置为 True

    //dlg->m_metadata.SetWindowText(author+"\r\n"+copyright+"\r\n"+description);
    dlg->m_formatmetadata.SetWindowText(meta);
    //-----------------------------------------------------------------
    return 0;
}
```

2.4. 码流分析模块

码流分析模块在视频播放过程中，伴随着视频的解码，分析其中的视音频参数。可以分为视频码流分析模块和音频码流分析模块。

（1）视频码流分析模块

视频码流分析模块伴随着视频的解码，分析每一个视频帧的参数。包括：

1) 序号

2) 帧类型

3) 关键帧

4) 码流序号

5) PTS

（2） 音频码流分析模块

音频码流分析模块伴随着音频的解码，分析音频帧的参数。包括：

1) 序号

2)　大小

3)　PTS

码流分析模块的流程图如图6所示。视频码流分析功能在函数ffmfc_param_vframe()中实现。音频码流分析功能在函数ffmfc_param_aframe()中实现。这两个函数在系统一帧一帧解码视频/音频的过程中循环调用。系统在初始化完成之后，调用av_read_frame()获取一帧一帧的视频/音频压缩编码数据（存储在结构体AVPacket中）。获取一帧压缩编码数据之后，首先判断它的类型。如果该帧数据是视频，则调用avcodec_decode_video2()对该帧视频进行解码，随后调用ffmfc_param_vframe()分析该帧视频的参数（主要存储在结构体AVFrame中）。如果该帧数据是音频，则调用avcodec_decode_audio4()对该帧音频进行解码，随后调用ffmfc_param_aframe()分析该帧音频的参数（主要也是存储在结构体AVFrame中）。

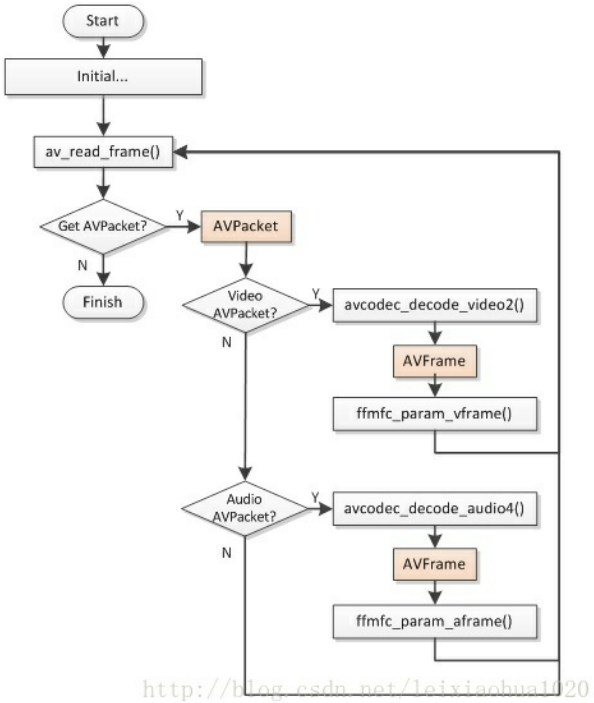

图6.码流分析模块流程

视频码流分析的函数ffmfc_param_vframe()代码如下：

```cpp
//视频帧参数提取
int ffmfc_param_vframe(VideoState *is,AVFrame *pFrame,AVPacket *packet){
    //-------------------------------------------------------------------
    CString key_frame,pict_type,reference,f_index,pts,dts,codednum;
    AVFormatContext *pFormatCtx = is->ic;
    int video_stream=is->video_stream;
    AVCodecContext *pCodecCtx = pFormatCtx->streams[video_stream]->codec;
    //避免数据太多，超过一定量之后，就会清零------------------------

    if(vframe_index>=MAX_FRAME_NUM){
        dlg->SystemClear();
    }

    //----------------------------
    f_index.Format("%d",vframe_index);
    //获取当前记录条数
    int nIndex=dlg->vddlg->m_videodecodelist.GetItemCount();
    //“行”数据结构
    LV_ITEM lvitem;
    lvitem.mask=LVIF_TEXT;
    lvitem.iItem=nIndex;
    lvitem.iSubItem=0;
    //注：vframe_index不可以直接赋值！
    //务必使用f_index执行Format!再赋值！
    lvitem.pszText=(char *)(LPCTSTR)f_index;
    //-----------------------


    switch(pFrame->key_frame){
    case 0:
        key_frame.Format("No");break;
    case 1:
        key_frame.Format("Yes");break;
    default:
        key_frame.Format("Unknown");
    }

    switch(pFrame->pict_type){
    case 0:
        pict_type.Format("Unknown");break;
    case 1:
        pict_type.Format("I");break;
    case 2:
        pict_type.Format("P");break;
    case 3:
        pict_type.Format("B");break;
    case 4:
        pict_type.Format("S");break;
    case 5:
        pict_type.Format("SI");break;
    case 6:
        pict_type.Format("SP");break;
    case 7:
        pict_type.Format("BI");break;
    default:
        pict_type.Format("Unknown");
    }

    reference.Format("%d",pFrame->reference);
    pts.Format("%d",pFrame->pkt_pts);
    dts.Format("%d",pFrame->pkt_dts);
    codednum.Format("%d",pFrame->coded_picture_number);

    //插入表格----------------------
    dlg->vddlg->m_videodecodelist.InsertItem(&lvitem);
    dlg->vddlg->m_videodecodelist.SetItemText(nIndex,1,pict_type);
    dlg->vddlg->m_videodecodelist.SetItemText(nIndex,2,key_frame);
    dlg->vddlg->m_videodecodelist.SetItemText(nIndex,3,codednum);
    dlg->vddlg->m_videodecodelist.SetItemText(nIndex,4,pts);
    dlg->vddlg->m_videodecodelist.SendMessage(WM_VSCROLL, SB_BOTTOM, NULL);
    vframe_index++;
    return 0;
}
```

音频码流分析的函数ffmfc_param_aframe()代码如下：

```cpp
//音频帧参数提取
int ffmfc_param_aframe(VideoState *is,AVFrame *pFrame,AVPacket *packet){
    //-----------------------------------------------------------------
    AVFormatContext *pFormatCtx = is->ic;
    int audio_stream=is->audio_stream;
    AVCodecContext *pCodecCtx = pFormatCtx->streams[audio_stream]->codec;
    //避免数据太多，超过一定量之后，就会清零------------------------

    if(aframe_index>=MAX_FRAME_NUM){
        dlg->SystemClear();
    }
    //----------------------------
    CString number,packet_size,dts,pts;
    //---------------
    number.Format("%d",aframe_index);
    //获取当前记录条数
    int nIndex=dlg->addlg->m_audiodecodelist.GetItemCount();
    //“行”数据结构
    LV_ITEM lvitem;
    lvitem.mask=LVIF_TEXT;
    lvitem.iItem=nIndex;
    lvitem.iSubItem=0;
    //注：frame_index不可以直接赋值！
    //务必使用f_index执行Format!再赋值！
    lvitem.pszText=(char *)(LPCTSTR)number;
    //------------------------
    packet_size.Format("%d",packet->size);
    pts.Format("%d",packet->pts);
    dts.Format("%d",packet->dts);
    //---------------
    dlg->addlg->m_audiodecodelist.InsertItem(&lvitem);
    dlg->addlg->m_audiodecodelist.SetItemText(nIndex,1,packet_size);
    dlg->addlg->m_audiodecodelist.SetItemText(nIndex,2,pts);
    dlg->addlg->m_audiodecodelist.SetItemText(nIndex,3,dts);
    dlg->addlg->m_audiodecodelist.SendMessage(WM_VSCROLL, SB_BOTTOM, NULL);
    aframe_index++;
    return 0;
}
```


文章标签： mfc 代码简介 ffplay 播放器 解码

个人分类： FFMPEG

所属专栏： FFmpeg