原 XBMC源代码简析 5：视频播放器（dvdplayer）-解复用器（以ffmpeg为例）

2014年01月08日 00:03:28 阅读数：6720

XBMC分析系列文章：

XBMC源代码分析 1：整体结构以及编译方法

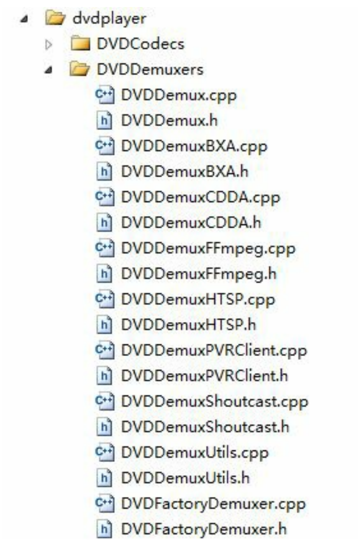
XBMC源代码分析 2：Addons（皮肤Skin）

XBMC源代码分析 3：核心部分（core）-综述

XBMC源代码分析 4：视频播放器（dvdplayer）-解码器（以ffmpeg为例）

本文我们分析XBMC中视频播放器（dvdplayer）中的解复用器部分。由于解复用器种类很多，不可能一一分析，因此以ffmpeg解复用器为例进行分析。

XBMC解复用器部分文件目录如下图所示：



```
dvdplayer
  DVDCodecs
  DVDDemuxers
    DVDDemux.cpp
    DVDDemux.h
    DVDDemuxBXA.cpp
    DVDDemuxBXA.h
    DVDDemuxCDDA.cpp
    DVDDemuxCDDA.h
    DVDDemuxFFmpeg.cpp
    DVDDemuxFFmpeg.h
    DVDDemuxHTSP.cpp
    DVDDemuxHTSP.h
    DVDDemuxPVRClient.cpp
    DVDDemuxPVRClient.h
    DVDDemuxShoutcast.cpp
    DVDDemuxShoutcast.h
    DVDDemuxUtils.cpp
    DVDDemuxUtils.h
    DVDFactoryDemuxer.cpp
    DVDFactoryDemuxer.h
```

在这里我们看一下解复用器中的FFMPEG解复用器。对应DVDDemuxFFmpeg.h和DVDDemuxFFmpeg.cpp

之前的分析类文章在解复用器这方面已经做过详细的分析了。在此就不多叙述了，代码很清晰。重点的地方已经标上了注释。

DVDDemuxFFmpeg.h源代码如下所示：

[cpp]

```cpp
/*
 * 雷霄骅
 * leixiaohua1020@126.com
 * 中国传媒大学/数字电视技术
 *
 */
#include "DVDDemux.h"
#include "DllAvFormat.h"
#include "DllAvCodec.h"
#include "DllAvUtil.h"

#include "threads/CriticalSection.h"
#include "threads/SystemClock.h"

#include <map>

class CDVDDemuxFFmpeg;
class CURL;

class CDemuxStreamVideoFFmpeg
  : public CDemuxStreamVideo
{
  CDVDDemuxFFmpeg *m_parent;
  AVStream*        m_stream;
public:
  CDemuxStreamVideoFFmpeg(CDVDDemuxFFmpeg *parent, AVStream* stream)
    : m_parent(parent)
    , m_stream(stream)
  {}
  virtual void GetStreamInfo(std::string& strInfo);
};


```

```cpp
class CDemuxStreamAudioFFmpeg
  : public CDemuxStreamAudio
{
  CDVDDemuxFFmpeg *m_parent;
  AVStream*        m_stream;
public:
  CDemuxStreamAudioFFmpeg(CDVDDemuxFFmpeg *parent, AVStream* stream)
    : m_parent(parent)
    , m_stream(stream)
  {}
  std::string m_description;

  virtual void GetStreamInfo(std::string& strInfo);
  virtual void GetStreamName(std::string& strInfo);
};

class CDemuxStreamSubtitleFFmpeg
  : public CDemuxStreamSubtitle
{
  CDVDDemuxFFmpeg *m_parent;
  AVStream*        m_stream;
public:
  CDemuxStreamSubtitleFFmpeg(CDVDDemuxFFmpeg *parent, AVStream* stream)
    : m_parent(parent)
    , m_stream(stream)
  {}
  std::string m_description;

  virtual void GetStreamInfo(std::string& strInfo);
  virtual void GetStreamName(std::string& strInfo);

};

#define FFMPEG_FILE_BUFFER_SIZE   32768 // default reading size for ffmpeg
#define FFMPEG_DVDNAV_BUFFER_SIZE 2048  // for dvd's
//FFMPEG解复用
class CDVDDemuxFFmpeg : public CDVDDemux
{
public:
  CDVDDemuxFFmpeg();
  virtual ~CDVDDemuxFFmpeg();
  //打开一个流
  bool Open(CDVDInputStream* pInput);
  void Dispose();//关闭
  void Reset();//复位
  void Flush();
  void Abort();
  void SetSpeed(int iSpeed);
  virtual std::string GetFileName();

  DemuxPacket* Read();

  bool SeekTime(int time, bool backwords = false, double* startpts = NULL);
  bool SeekByte(int64_t pos);
  int GetStreamLength();
  CDemuxStream* GetStream(int iStreamId);
  int GetNrOfStreams();

  bool SeekChapter(int chapter, double* startpts = NULL);
  int GetChapterCount();
  int GetChapter();
  void GetChapterName(std::string& strChapterName);
  virtual void GetStreamCodecName(int iStreamId, CStdString &strName);

  bool Aborted();

  AVFormatContext* m_pFormatContext;
  CDVDInputStream* m_pInput;

protected:
  friend class CDemuxStreamAudioFFmpeg;
  friend class CDemuxStreamVideoFFmpeg;
  friend class CDemuxStreamSubtitleFFmpeg;

  int ReadFrame(AVPacket *packet);
  CDemuxStream* AddStream(int iId);
  void AddStream(int iId, CDemuxStream* stream);
  CDemuxStream* GetStreamInternal(int iStreamId);
  void CreateStreams(unsigned int program = UINT_MAX);
  void DisposeStreams();

  AVDictionary *GetFFMpegOptionsFromURL(const CURL &url);
  double ConvertTimestamp(int64_t pts, int den, int num);
  void UpdateCurrentPTS();
  bool IsProgramChange();

  CCriticalSection m_critSection;
  std::map<int, CDemuxStream*> m_streams;
  std::vector<std::map<int, CDemuxStream*>::iterator> m_stream_index;

  AVIOContext* m_ioContext;
  //各种封装的Dll
  DllAvFormat m_dllAvFormat;
  DllAvCodec  m_dllAvCodec;
  DllAvUtil   m_dllAvUtil;

  double   m_iCurrentPts; // used for stream length estimation
  bool     m_bMatroska;
  bool     m_bAVI;
  int      m_speed;
  unsigned m_program;
  XbmcThreads::EndTime  m_timeout;

  // Due to limitations of ffmpeg, we only can detect a program change
  // with a packet. This struct saves the packet for the next read and
  // signals STREAMCHANGE to player
  struct
  {
    AVPacket pkt;       // packet ffmpeg returned
    int      result;    // result from av_read_packet
  }m_pkt;
};
```

该类中以下几个函数包含了解复用器的几个功能。

bool Open(CDVDInputStream* pInput);//打开

void Dispose();//关闭

void Reset();//复位

void Flush();

我们查看一下这几个函数的源代码。

Open()

[cpp]

```cpp
//打开一个流
bool CDVDDemuxFFmpeg::Open(CDVDInputStream* pInput)
{
  AVInputFormat* iformat = NULL;
  std::string strFile;
  m_iCurrentPts = DVD_NOPTS_VALUE;
  m_speed = DVD_PLAYSPEED_NORMAL;
  m_program = UINT_MAX;
  const AVIOInterruptCB int_cb = { interrupt_cb, this };

  if (!pInput) return false;

  if (!m_dllAvUtil.Load() || !m_dllAvCodec.Load() || !m_dllAvFormat.Load())  {
    CLog::Log(LOGERROR,"CDVDDemuxFFmpeg::Open - failed to load ffmpeg libraries");
    return false;
  }
  //注册解复用器
  // register codecs
  m_dllAvFormat.av_register_all();

  m_pInput = pInput;
  strFile = m_pInput->GetFileName();

  bool streaminfo = true; /* set to true if we want to look for streams before playback*/

  if( m_pInput->GetContent().length() > 0 )
  {
    std::string content = m_pInput->GetContent();

    /* check if we can get a hint from content */
    if      ( content.compare("video/x-vobsub") == 0 )
      iformat = m_dllAvFormat.av_find_input_format("mpeg");
    else if( content.compare("video/x-dvd-mpeg") == 0 )
      iformat = m_dllAvFormat.av_find_input_format("mpeg");
    else if( content.compare("video/x-mpegts") == 0 )
      iformat = m_dllAvFormat.av_find_input_format("mpegts");
    else if( content.compare("multipart/x-mixed-replace") == 0 )
      iformat = m_dllAvFormat.av_find_input_format("mjpeg");
  }

  // open the demuxer
  m_pFormatContext  = m_dllAvFormat.avformat_alloc_context();
  m_pFormatContext->interrupt_callback = int_cb;

  // try to abort after 30 seconds
  m_timeout.Set(30000);

  if( m_pInput->IsStreamType(DVDSTREAM_TYPE_FFMPEG) )
  {
    // special stream type that makes avformat handle file opening
    // allows internal ffmpeg protocols to be used
    CURL url = m_pInput->GetURL();
    CStdString protocol = url.GetProtocol();
    AVDictionary *options = GetFFMpegOptionsFromURL(url);

    int result=-1;
    if (protocol.Equals("mms"))
    {
      // try mmsh, then mmst
      url.SetProtocol("mmsh");
      url.SetProtocolOptions("");
      //真正地打开
      result = m_dllAvFormat.avformat_open_input(&m_pFormatContext, url.Get().c_str(), iformat, &options);
      if (result < 0)
      {
        url.SetProtocol("mmst");
        strFile = url.Get();
      }
    }
    //真正地打开
    if (result < 0 && m_dllAvFormat.avformat_open_input(&m_pFormatContext, strFile.c_str(), iformat, &options) < 0 )
    {
      CLog::Log(LOGDEBUG, "Error, could not open file %s", CURL::GetRedacted(strFile).c_str());
      Dispose();
      m_dllAvUtil.av_dict_free(&options);
      return false;
    }
    m_dllAvUtil.av_dict_free(&options);
  }
  else
  {
    unsigned char* buffer = (unsigned char*)m_dllAvUtil.av_malloc(FFMPEG_FILE_BUFFER_SIZE);
    m_ioContext = m_dllAvFormat.avio_alloc_context(buffer, FFMPEG_FILE_BUFFER_SIZE, 0, this, dvd_file_read, NULL, dvd_file_seek);
    m_ioContext->max_packet_size = m_pInput->GetBlockSize();
    if(m_ioContext->max_packet_size)
      m_ioContext->max_packet_size *= FFMPEG_FILE_BUFFER_SIZE / m_ioContext->max_packet_size;

    if(m_pInput->Seek(0, SEEK_POSSIBLE) == 0)
      m_ioContext->seekable = 0;

    if( iformat == NULL )
    {
      // let ffmpeg decide which demuxer we have to open

      bool trySPDIFonly = (m_pInput->GetContent() == "audio/x-spdif-compressed");

      if (!trySPDIFonly)
        m_dllAvFormat.av_probe_input_buffer(m_ioContext, &iformat, strFile.c_str(), NULL, 0, 0);

      // Use the more low-level code in case we have been built against an old
      // FFmpeg without the above av_probe_input_buffer(), or in case we only
      // want to probe for spdif (DTS or IEC 61937) compressed audio
      // specifically, or in case the file is a wav which may contain DTS or
      // IEC 61937 (e.g. ac3-in-wav) and we want to check for those formats.
      if (trySPDIFonly || (iformat && strcmp(iformat->name, "wav") == 0))
      {
        AVProbeData pd;
        uint8_t probe_buffer[FFMPEG_FILE_BUFFER_SIZE + AVPROBE_PADDING_SIZE];

        // init probe data
        pd.buf = probe_buffer;
        pd.filename = strFile.c_str();

        // read data using avformat's buffers
        pd.buf_size = m_dllAvFormat.avio_read(m_ioContext, pd.buf, m_ioContext->max_packet_size ? m_ioContext->max_packet_size : m_ioContext->buffer_size);
        if (pd.buf_size <= 0)
        {
          CLog::Log(LOGERROR, "%s - error reading from input stream, %s", __FUNCTION__, CURL::GetRedacted(strFile).c_str());
          return false;
        }
        memset(pd.buf+pd.buf_size, 0, AVPROBE_PADDING_SIZE);

        // restore position again
        m_dllAvFormat.avio_seek(m_ioContext , 0, SEEK_SET);

        // the advancedsetting is for allowing the user to force outputting the
        // 44.1 kHz DTS wav file as PCM, so that an A/V receiver can decode
        // it (this is temporary until we handle 44.1 kHz passthrough properly)
        if (trySPDIFonly || (iformat && strcmp(iformat->name, "wav") == 0 && !g_advancedSettings.m_dvdplayerIgnoreDTSinWAV))
        {
          // check for spdif and dts
          // This is used with wav files and audio CDs that may contain
          // a DTS or AC3 track padded for S/PDIF playback. If neither of those
          // is present, we assume it is PCM audio.
          // AC3 is always wrapped in iec61937 (ffmpeg "spdif"), while DTS
          // may be just padded.
          AVInputFormat *iformat2;
          iformat2 = m_dllAvFormat.av_find_input_format("spdif");

          if (iformat2 && iformat2->read_probe(&pd) > AVPROBE_SCORE_MAX / 4)
          {
            iformat = iformat2;
          }
          }
          else
          {
            // not spdif or no spdif demuxer, try dts
            iformat2 = m_dllAvFormat.av_find_input_format("dts");

            if (iformat2 && iformat2->read_probe(&pd) > AVPROBE_SCORE_MAX / 4)
            {
              iformat = iformat2;
            }
            else if (trySPDIFonly)
            {
              // not dts either, return false in case we were explicitely
              // requested to only check for S/PDIF padded compressed audio
              CLog::Log(LOGDEBUG, "%s - not spdif or dts file, fallbacking", __FUNCTION__);
              return false;
            }
          }
        }
      }

      if(!iformat)
      {
        std::string content = m_pInput->GetContent();

        /* check if we can get a hint from content */
        if( content.compare("audio/aacp") == 0 )
          iformat = m_dllAvFormat.av_find_input_format("aac");
        else if( content.compare("audio/aac") == 0 )
          iformat = m_dllAvFormat.av_find_input_format("aac");
        else if( content.compare("video/flv") == 0 )
          iformat = m_dllAvFormat.av_find_input_format("flv");
        else if( content.compare("video/x-flv") == 0 )
          iformat = m_dllAvFormat.av_find_input_format("flv");
      }

      if (!iformat)
      {
        CLog::Log(LOGERROR, "%s - error probing input format, %s", __FUNCTION__, CURL::GetRedacted(strFile).c_str());
        return false;
      }
      else
      {
        if (iformat->name)
          CLog::Log(LOGDEBUG, "%s - probing detected format [%s]", __FUNCTION__, iformat->name);
        else
          CLog::Log(LOGDEBUG, "%s - probing detected unnamed format", __FUNCTION__);
      }
    }


    m_pFormatContext->pb = m_ioContext;

    if (m_dllAvFormat.avformat_open_input(&m_pFormatContext, strFile.c_str(), iformat, NULL) < 0)
    {
      CLog::Log(LOGERROR, "%s - Error, could not open file %s", __FUNCTION__, CURL::GetRedacted(strFile).c_str());
      Dispose();
      return false;
    }
  }

  // Avoid detecting framerate if advancedsettings.xml says so
  if (g_advancedSettings.m_videoFpsDetect == 0)
      m_pFormatContext->fps_probe_size = 0;

  // analyse very short to speed up mjpeg playback start
  if (iformat && (strcmp(iformat->name, "mjpeg") == 0) && m_ioContext->seekable == 0)
    m_pFormatContext->max_analyze_duration = 500000;

  // we need to know if this is matroska or avi later
  m_bMatroska = strncmp(m_pFormatContext->iformat->name, "matroska", 8) == 0; // for "matroska.webm"
  m_bAVI = strcmp(m_pFormatContext->iformat->name, "avi") == 0;

  if (streaminfo)
  {
    /* too speed up dvd switches, only analyse very short */
    if(m_pInput->IsStreamType(DVDSTREAM_TYPE_DVD))
      m_pFormatContext->max_analyze_duration = 500000;


    CLog::Log(LOGDEBUG, "%s - avformat_find_stream_info starting", __FUNCTION__);
    int iErr = m_dllAvFormat.avformat_find_stream_info(m_pFormatContext, NULL);
    if (iErr < 0)
    {
      CLog::Log(LOGWARNING,"could not find codec parameters for %s", CURL::GetRedacted(strFile).c_str());
      if (m_pInput->IsStreamType(DVDSTREAM_TYPE_DVD)
      ||  m_pInput->IsStreamType(DVDSTREAM_TYPE_BLURAY)
      || (m_pFormatContext->nb_streams == 1 && m_pFormatContext->streams[0]->codec->codec_id == AV_CODEC_ID_AC3))
      {
        // special case, our codecs can still handle it.
      }
      else
```

```cpp
      {
        Dispose();
        return false;
      }
    }
    CLog::Log(LOGDEBUG, "%s - av_find_stream_info finished", __FUNCTION__);
  }
  // reset any timeout
  m_timeout.SetInfinite();

  // if format can be nonblocking, let's use that
  m_pFormatContext->flags |= AVFMT_FLAG_NONBLOCK;

  // print some extra information
  m_dllAvFormat.av_dump_format(m_pFormatContext, 0, strFile.c_str(), 0);

  UpdateCurrentPTS();

  CreateStreams();

  return true;
}
```

Dispose()

[cpp]

```cpp
//关闭
void CDVDDemuxFFmpeg::Dispose()
{
  m_pkt.result = -1;
  m_dllAvCodec.av_free_packet(&m_pkt.pkt);

  if (m_pFormatContext)
  {
    if (m_ioContext && m_pFormatContext->pb && m_pFormatContext->pb != m_ioContext)
    {
      CLog::Log(LOGWARNING, "CDVDDemuxFFmpeg::Dispose - demuxer changed our byte context behind our back, possible memleak");
      m_ioContext = m_pFormatContext->pb;
    }
    m_dllAvFormat.avformat_close_input(&m_pFormatContext);
  }

  if(m_ioContext)
  {
    m_dllAvUtil.av_free(m_ioContext->buffer);
    m_dllAvUtil.av_free(m_ioContext);
  }

  m_ioContext = NULL;
  m_pFormatContext = NULL;
  m_speed = DVD_PLAYSPEED_NORMAL;

  DisposeStreams();

  m_pInput = NULL;

  m_dllAvFormat.Unload();
  m_dllAvCodec.Unload();
  m_dllAvUtil.Unload();
}
```

Reset()

[cpp]

```cpp
//复位
void CDVDDemuxFFmpeg::Reset()
{
  CDVDInputStream* pInputStream = m_pInput;
  Dispose();
  Open(pInputStream);
}
```

Flush()

[cpp]

```cpp
void CDVDDemuxFFmpeg::Flush()
{
  // naughty usage of an internal ffmpeg function
  if (m_pFormatContext)
    m_dllAvFormat.av_read_frame_flush(m_pFormatContext);

  m_iCurrentPts = DVD_NOPTS_VALUE;

  m_pkt.result = -1;
  m_dllAvCodec.av_free_packet(&m_pkt.pkt);
}
```

文章标签： xbmc ffmpeg 源代码 解复用 播放器

个人分类： FFMPEG XBMC

所属专栏： 开源多媒体项目源代码分析