

## 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](#)

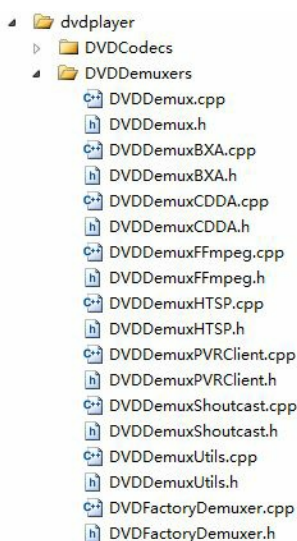
[XBMC源代码分析 2：Addons（皮肤Skin）](#)

[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：



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]
1.  /*
2.   * 雷霄骅
3.   *  leixiaohua1020@126.com
4.   *  中国传媒大学/数字电视技术
5.   *
6.   */
7.  #include "DVDDemux.h"
8.  #include "DllAvFormat.h"
9.  #include "DllAvCodec.h"
10. #include "DllAvUtil.h"
11.
12. #include "threads/CriticalSection.h"
13. #include "threads/SystemClock.h"
14.
15. #include <map>
16.
17. class CDVDDemuxFFmpeg;
18. class CURL;
19.
20. class CDemuxStreamVideoFFmpeg
21. : public CDemuxStreamVideo
22. {
23.     CDVDDemuxFFmpeg *m_parent;
24.     AVStream*      m_stream;
25. public:
26.     CDemuxStreamVideoFFmpeg(CDVDDemuxFFmpeg *parent, AVStream* stream)
27.         : m_parent(parent)
28.         , m_stream(stream)
29.     {}
30.     virtual void GetStreamInfo(std::string& strInfo);
31. };
32.
33.
```

```

34. class CDemuxStreamAudioFFmpeg
35. : public CDemuxStreamAudio
36. {
37.     CDVDDemuxFFmpeg *m_parent;
38.     AVStream*      m_stream;
39. public:
40.     CDemuxStreamAudioFFmpeg(CDVDDemuxFFmpeg *parent, AVStream* stream)
41.         : m_parent(parent)
42.         , m_stream(stream)
43.     {}
44.     std::string m_description;
45.
46.     virtual void GetStreamInfo(std::string& strInfo);
47.     virtual void GetStreamName(std::string& strInfo);
48. };
49.
50. class CDemuxStreamSubtitleFFmpeg
51. : public CDemuxStreamSubtitle
52. {
53.     CDVDDemuxFFmpeg *m_parent;
54.     AVStream*      m_stream;
55. public:
56.     CDemuxStreamSubtitleFFmpeg(CDVDDemuxFFmpeg *parent, AVStream* stream)
57.         : m_parent(parent)
58.         , m_stream(stream)
59.     {}
60.     std::string m_description;
61.
62.     virtual void GetStreamInfo(std::string& strInfo);
63.     virtual void GetStreamName(std::string& strInfo);
64. };
65.
66.
67. #define FFMPEG_FILE_BUFFER_SIZE 32768 // default reading size for ffmpeg
68. #define FFMPEG_DVDNAV_BUFFER_SIZE 2048 // for dvd's
69. //FFMPEG解复用
70. class CDVDDemuxFFmpeg : public CDVDDemux
71. {
72. public:
73.     CDVDDemuxFFmpeg();
74.     virtual ~CDVDDemuxFFmpeg();
75.     //打开一个流
76.     bool Open(CDVDInputStream* pInput);
77.     void Dispose(); //关闭
78.     void Reset(); //复位
79.     void Flush();
80.     void Abort();
81.     void SetSpeed(int iSpeed);
82.     virtual std::string GetFileName();
83.
84.     DemuxPacket* Read();
85.
86.     bool SeekTime(int time, bool backwards = false, double* startpts = NULL);
87.     bool SeekByte(int64_t pos);
88.     int GetStreamLength();
89.     CDemuxStream* GetStream(int iStreamId);
90.     int GetNrOfStreams();
91.
92.     bool SeekChapter(int chapter, double* startpts = NULL);
93.     int GetChapterCount();
94.     int GetChapter();
95.     void GetChapterName(std::string& strChapterName);
96.     virtual void GetStreamCodecName(int iStreamId, CStdString &strName);
97.
98.     bool Aborted();
99.
100.     AVFormatContext* m_pFormatContext;
101.     CDVDInputStream* m_pInput;
102.
103. protected:
104.     friend class CDemuxStreamAudioFFmpeg;
105.     friend class CDemuxStreamVideoFFmpeg;
106.     friend class CDemuxStreamSubtitleFFmpeg;
107.
108.     int ReadFrame(AVPacket *packet);
109.     CDemuxStream* AddStream(int iId);
110.     void AddStream(int iId, CDemuxStream* stream);
111.     CDemuxStream* GetStreamInternal(int iStreamId);
112.     void CreateStreams(unsigned int program = UINT_MAX);
113.     void DisposeStreams();
114.
115.     AVDictionary *GetFFmpegOptionsFromURL(const CURL &url);
116.     double ConvertTimestamp(int64_t pts, int den, int num);
117.     void UpdateCurrentPTS();
118.     bool IsProgramChange();
119.
120.     CCriticalSection m_critSection;
121.     std::map<int, CDemuxStream*> m_streams;
122.     std::vector<std::map<int, CDemuxStream*>::iterator> m_stream_index;
123.
124.     AVIOContext* m_ioContext;

```

```

125. //各种封装的Dll
126. DllAvFormat m_dllAvFormat;
127. DllAvCodec m_dllAvCodec;
128. DllAvUtil m_dllAvUtil;
129.
130. double m_iCurrentPts; // used for stream length estimation
131. bool m_bMatroska;
132. bool m_bAVI;
133. int m_speed;
134. unsigned m_program;
135. XbmcThreads::EndTime m_timeout;
136.
137. // Due to limitations of ffmpeg, we only can detect a program change
138. // with a packet. This struct saves the packet for the next read and
139. // signals STREAMCHANGE to player
140. struct
141. {
142.     AVPacket pkt; // packet ffmpeg returned
143.     int result; // result from av_read_packet
144. }m_pkt;
145. };

```

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();

```

我们看一下这几个函数的源代码。

Open()

```

1. //打开一个流
2. bool CDVDDemuxFFmpeg::Open(CDVDInputStream* pInput)
3. {
4.     AVInputFormat* iformat = NULL;
5.     std::string strFile;
6.     m_iCurrentPts = DVD_NOPTS_VALUE;
7.     m_speed = DVD_PLAYSPEED_NORMAL;
8.     m_program = UINT_MAX;
9.     const AVIOInterruptCB int_cb = { interrupt_cb, this };
10.
11.     if (!pInput) return false;
12.
13.     if (!m_dllAvUtil.Load() || !m_dllAvCodec.Load() || !m_dllAvFormat.Load()) {
14.         CLog::Log(LOGERROR, "CDVDDemuxFFmpeg::Open - failed to load ffmpeg libraries");
15.         return false;
16.     }
17.     //注册解复用器
18.     // register codecs
19.     m_dllAvFormat.av_register_all();
20.
21.     m_pInput = pInput;
22.     strFile = m_pInput->GetFileName();
23.
24.     bool streaminfo = true; /* set to true if we want to look for streams before playback*/
25.
26.     if( m_pInput->GetContent().length() > 0 )
27.     {
28.         std::string content = m_pInput->GetContent();
29.
30.         /* check if we can get a hint from content */
31.         if ( content.compare("video/x-vobsub") == 0 )
32.             iformat = m_dllAvFormat.av_find_input_format("mpeg");
33.         else if( content.compare("video/x-dvd-mpeg") == 0 )
34.             iformat = m_dllAvFormat.av_find_input_format("mpeg");
35.         else if( content.compare("video/x-mpegts") == 0 )
36.             iformat = m_dllAvFormat.av_find_input_format("mpegts");
37.         else if( content.compare("multipart/x-mixed-replace") == 0 )
38.             iformat = m_dllAvFormat.av_find_input_format("mjpeg");
39.     }
40.
41.     // open the demuxer
42.     m_pFormatContext = m_dllAvFormat.avformat_alloc_context();
43.     m_pFormatContext->interrupt_callback = int_cb;
44.
45.     // try to abort after 30 seconds
46.     m_timeout.Set(30000);
47.
48.     if( m_pInput->IsStreamType(DVDSTREAM_TYPE_FFMPEG) )
49.     {
50.         // special stream type that makes avformat handle file opening
51.         // allows internal ffmpeg protocols to be used
52.         CURL url = m_pInput->GetURL();
53.         CString protocol = url.GetProtocol();
54.

```

```

54.
55.     AVDictionary *options = GetFFmpegOptionsFromURL(url);
56.
57.     int result=-1;
58.     if (protocol.Equals("mms"))
59.     {
60.         // try mmsh, then mmst
61.         url.SetProtocol("mmsh");
62.         url.SetProtocolOptions("");
63.         //真正地打开
64.         result = m_dllAvFormat.avformat_open_input(&m_pFormatContext, url.Get().c_str(), iformat, &options);
65.         if (result < 0)
66.         {
67.             url.SetProtocol("mmst");
68.             strFile = url.Get();
69.         }
70.     }
71.     //真正地打开
72.     if (result < 0 && m_dllAvFormat.avformat_open_input(&m_pFormatContext, strFile.c_str(), iformat, &options) < 0 )
73.     {
74.         CLog::Log(LOGDEBUG, "Error, could not open file %s", CURL::GetRedacted(strFile).c_str());
75.         Dispose();
76.         m_dllAvUtil.av_dict_free(&options);
77.         return false;
78.     }
79.     m_dllAvUtil.av_dict_free(&options);
80. }
81. else
82. {
83.     unsigned char* buffer = (unsigned char*)m_dllAvUtil.av_malloc(FFMPEG_FILE_BUFFER_SIZE);
84.     m_ioContext = m_dllAvFormat.avio_alloc_context(buffer, FFMPEG_FILE_BUFFER_SIZE, 0, this, dvd_file_read, NULL, dvd_file_seek);
85.     m_ioContext->max_packet_size = m_pInput->GetBlockSize();
86.     if(m_ioContext->max_packet_size)
87.         m_ioContext->max_packet_size *= FFMPEG_FILE_BUFFER_SIZE / m_ioContext->max_packet_size;
88.
89.     if(m_pInput->Seek(0, SEEK_POSSIBLE) == 0)
90.         m_ioContext->seekable = 0;
91.
92.     if( iformat == NULL )
93.     {
94.         // let ffmpeg decide which demuxer we have to open
95.
96.         bool trySPDIFonly = (m_pInput->GetContent() == "audio/x-spdif-compressed");
97.
98.         if (!trySPDIFonly)
99.             m_dllAvFormat.av_probe_input_buffer(m_ioContext, &iformat, strFile.c_str(), NULL, 0, 0);
100.
101.         // Use the more low-level code in case we have been built against an old
102.         // Ffmpeg without the above av_probe_input_buffer(), or in case we only
103.         // want to probe for spdif (DTS or IEC 61937) compressed audio
104.         // specifically, or in case the file is a wav which may contain DTS or
105.         // IEC 61937 (e.g. ac3-in-wav) and we want to check for those formats.
106.         if (trySPDIFonly || (iformat && strcmp(iformat->name, "wav") == 0))
107.         {
108.             AVProbeData pd;
109.             uint8_t probe_buffer[FFMPEG_FILE_BUFFER_SIZE + AVPROBE_PADDING_SIZE];
110.
111.             // init probe data
112.             pd.buf = probe_buffer;
113.             pd.filename = strFile.c_str();
114.
115.             // read data using avformat's buffers
116.             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);
117.             if (pd.buf_size <= 0)
118.             {
119.                 CLog::Log(LOGERROR, "%s - error reading from input stream, %s", __FUNCTION__, CURL::GetRedacted(strFile).c_str());
120.                 return false;
121.             }
122.             memset(pd.buf+pd.buf_size, 0, AVPROBE_PADDING_SIZE);
123.
124.             // restore position again
125.             m_dllAvFormat.avio_seek(m_ioContext, 0, SEEK_SET);
126.
127.             // the advancedsetting is for allowing the user to force outputting the
128.             // 44.1 kHz DTS wav file as PCM, so that an A/V receiver can decode
129.             // it (this is temporary until we handle 44.1 kHz passthrough properly)
130.             if (trySPDIFonly || (iformat && strcmp(iformat->name, "wav") == 0 && !g_advancedSettings.m_dvdplayerIgnoreDTSinWAV))
131.             {
132.                 // check for spdif and dts
133.                 // This is used with wav files and audio CDs that may contain
134.                 // a DTS or AC3 track padded for S/PDIF playback. If neither of those
135.                 // is present, we assume it is PCM audio.
136.                 // AC3 is always wrapped in iec61937 (ffmpeg "spdif"), while DTS
137.                 // may be just padded.
138.                 AVInputFormat *iformat2;
139.                 iformat2 = m_dllAvFormat.av_find_input_format("spdif");
140.
141.                 if (iformat2 && iformat2->read_probe(&pd) > AVPROBE_SCORE_MAX / 4)
142.                 {
143.                     iformat = iformat2;
144.                 }

```

```

145.         else
146.         {
147.             // not spdif or no spdif demuxer, try dts
148.             iformat2 = m_dllAvFormat.av_find_input_format("dts");
149.
150.             if (iformat2 && iformat2->read_probe(&pd) > AVPROBE_SCORE_MAX / 4)
151.             {
152.                 iformat = iformat2;
153.             }
154.             else if (trySPDIFonly)
155.             {
156.                 // not dts either, return false in case we were explicitly
157.                 // requested to only check for SPDIF padded compressed audio
158.                 CLog::Log(LOGDEBUG, "%s - not spdif or dts file, fallbacking", __FUNCTION__);
159.                 return false;
160.             }
161.         }
162.     }
163. }
164.
165. if(!iformat)
166. {
167.     std::string content = m_pInput->GetContent();
168.
169.     /* check if we can get a hint from content */
170.     if( content.compare("audio/aac") == 0 )
171.         iformat = m_dllAvFormat.av_find_input_format("aac");
172.     else if( content.compare("audio/aac") == 0 )
173.         iformat = m_dllAvFormat.av_find_input_format("aac");
174.     else if( content.compare("video/flv") == 0 )
175.         iformat = m_dllAvFormat.av_find_input_format("flv");
176.     else if( content.compare("video/x-flv") == 0 )
177.         iformat = m_dllAvFormat.av_find_input_format("flv");
178. }
179.
180. if (!iformat)
181. {
182.     CLog::Log(LOGERROR, "%s - error probing input format, %s", __FUNCTION__, CURL::GetRedacted(strFile).c_str());
183.     return false;
184. }
185. else
186. {
187.     if (iformat->name)
188.         CLog::Log(LOGDEBUG, "%s - probing detected format [%s]", __FUNCTION__, iformat->name);
189.     else
190.         CLog::Log(LOGDEBUG, "%s - probing detected unnamed format", __FUNCTION__);
191. }
192. }
193.
194.
195. m_pFormatContext->pb = m_ioContext;
196.
197. if (m_dllAvFormat.avformat_open_input(&m_pFormatContext, strFile.c_str(), iformat, NULL) < 0)
198. {
199.     CLog::Log(LOGERROR, "%s - Error, could not open file %s", __FUNCTION__, CURL::GetRedacted(strFile).c_str());
200.     Dispose();
201.     return false;
202. }
203. }
204.
205. // Avoid detecting framerate if advancedSettings.xml says so
206. if (g_advancedSettings.m_videoFpsDetect == 0)
207.     m_pFormatContext->fps_probe_size = 0;
208.
209. // analyse very short to speed up mjpeg playback start
210. if (iformat && (strcmp(iformat->name, "mjpeg") == 0) && m_ioContext->seekable == 0)
211.     m_pFormatContext->max_analyze_duration = 500000;
212.
213. // we need to know if this is matroska or avi later
214. m_bMatroska = strncmp(m_pFormatContext->iformat->name, "matroska", 8) == 0; // for "matroska.webm"
215. m_bAVI = strcmp(m_pFormatContext->iformat->name, "avi") == 0;
216.
217. if (streaminfo)
218. {
219.     /* too speed up dvd switches, only analyse very short */
220.     if(m_pInput->IsStreamType(DVDSTREAM_TYPE_DVD))
221.         m_pFormatContext->max_analyze_duration = 500000;
222.
223.
224.     CLog::Log(LOGDEBUG, "%s - avformat_find_stream_info starting", __FUNCTION__);
225.     int iErr = m_dllAvFormat.avformat_find_stream_info(m_pFormatContext, NULL);
226.     if (iErr < 0)
227.     {
228.         CLog::Log(LOGWARNING, "could not find codec parameters for %s", CURL::GetRedacted(strFile).c_str());
229.         if (m_pInput->IsStreamType(DVDSTREAM_TYPE_DVD)
230.             || m_pInput->IsStreamType(DVDSTREAM_TYPE_BLURAY)
231.             || (m_pFormatContext->nb_streams == 1 && m_pFormatContext->streams[0]->codec->codec_id == AV_CODEC_ID_AC3))
232.         {
233.             // special case, our codecs can still handle it.
234.         }
235.     }
236. }

```

```

236.     {
237.         Dispose();
238.         return false;
239.     }
240. }
241. CLog::Log(LOGDEBUG, "%s - av_find_stream_info finished", __FUNCTION__);
242. }
243. // reset any timeout
244. m_timeout.SetInfinite();
245.
246. // if format can be nonblocking, let's use that
247. m_pFormatContext->flags |= AVFMT_FLAG_NONBLOCK;
248.
249. // print some extra information
250. m_dllAvFormat.av_dump_format(m_pFormatContext, 0, strFile.c_str(), 0);
251.
252. UpdateCurrentPTS();
253.
254. CreateStreams();
255.
256. return true;
257. }

```

## Dispose()

```

1. //关闭
2. void CDVDDemuxFFmpeg::Dispose()
3. {
4.     m_pkt.result = -1;
5.     m_dllAvCodec.av_free_packet(&m_pkt.pkt);
6.
7.     if (m_pFormatContext)
8.     {
9.         if (m_ioContext && m_pFormatContext->pb && m_pFormatContext->pb != m_ioContext)
10.        {
11.            CLog::Log(LOGWARNING, "CDVDDemuxFFmpeg::Dispose - demuxer changed our byte context behind our back, possible memleak");
12.            m_ioContext = m_pFormatContext->pb;
13.        }
14.        m_dllAvFormat.avformat_close_input(&m_pFormatContext);
15.    }
16.
17.    if(m_ioContext)
18.    {
19.        m_dllAvUtil.av_free(m_ioContext->buffer);
20.        m_dllAvUtil.av_free(m_ioContext);
21.    }
22.
23.    m_ioContext = NULL;
24.    m_pFormatContext = NULL;
25.    m_speed = DVD_PLAYSPEED_NORMAL;
26.
27.    DisposeStreams();
28.
29.    m_pInput = NULL;
30.
31.    m_dllAvFormat.Unload();
32.    m_dllAvCodec.Unload();
33.    m_dllAvUtil.Unload();
34. }

```

## Reset()

```

1. //复位
2. void CDVDDemuxFFmpeg::Reset()
3. {
4.     CDVDInputStream* pInputStream = m_pInput;
5.     Dispose();
6.     Open(pInputStream);
7. }

```

## Flush()

[cpp]  

```
1. void CDVDDemuxFFmpeg::Flush()  
2. {  
3.     // naughty usage of an internal ffmpeg function  
4.     if (m_pFormatContext)  
5.         m_dllAvFormat.av_read_frame_flush(m_pFormatContext);  
6.  
7.     m_iCurrentPts = DVD_NOPTS_VALUE;  
8.  
9.     m_pkt.result = -1;  
10.    m_dllAvCodec.av_free_packet(&m_pkt.pkt);  
11. }
```

版权声明：本文为博主原创文章，未经博主允许不得转载。 <https://blog.csdn.net/leixiaohua1020/article/details/17512633>

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](#)

此PDF由[spygg](#)生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com