XBMC源代码分析 6：视频播放器（dvdplayer）-文件头（以ffmpeg为例）

XBMC分析系列文章：

[XBMC源代码分析 1：整体结构以及编译方法](#)

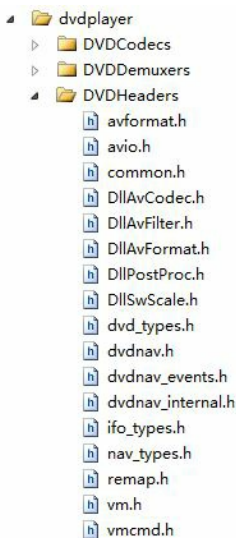[XBMC源代码分析 2：Addons（皮肤Skin）](#)

[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）](#)

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

本文我们分析XBMC中视频播放器（dvdplayer）中的文件头部分。文件头部分里包含的是封装Dll用到的头文件。由于文件头种类很多，不可能一一分析，因此还是以ffmpeg文件头为例进行分析。

XBMC中文件头部分文件目录结构如下图所示。

在这里我们看一下封装AVCodec和AVFormat结构体的头文件，分别是DllAvCodec.h和DllAvFormat.h。

DllAvFormat.h内容如下。其中包含了2个主要的类：DllAvFormatInterface和DllAvFormat。

其中DllAvFormatInterface是一个纯虚类，里面全是纯虚函数。

DllAvFormat中包含很多已经定义过的宏，稍后我们分析一下这些宏的含义。

```cpp
/*
 * 雷霄骅
 * leixiaohua1020@126.com
 * 中国传媒大学/数字电视技术
 *
 */
//接口的作用
class DllAvFormatInterface
{
public:
  virtual ~DllAvFormatInterface() {}
  virtual void av_register_all_dont_call(void)=0;
  virtual void avformat_network_init_dont_call(void)=0;
  virtual void avformat_network_deinit_dont_call(void)=0;
  virtual AVInputFormat *av_find_input_format(const char *short_name)=0;
  virtual void avformat_close_input(AVFormatContext **s)=0;
  virtual int av_read_frame(AVFormatContext *s, AVPacket *pkt)=0;
  virtual void av_read_frame_flush(AVFormatContext *s)=0;
  virtual int av_read_play(AVFormatContext *s)=0;
  virtual int av_read_pause(AVFormatContext *s)=0;
  virtual int av_seek_frame(AVFormatContext *s, int stream_index, int64_t timestamp, int flags)=0;
#if (!defined USE_EXTERNAL_FFMPEG) && (!defined TARGET_DARWIN)
  virtual int avformat_find_stream_info_dont_call(AVFormatContext *ic, AVDictionary **options)=0;
#endif
  virtual int avformat_open_input(AVFormatContext **ps, const char *filename, AVInputFormat *fmt, AVDictionary **options)=0;
  virtual AVIOContext *avio_alloc_context(unsigned char *buffer, int buffer_size, int write_flag, void *opaque,
                     int (*read_packet)(void *opaque, uint8_t *buf, int buf_size),
                     int (*write_packet)(void *opaque, uint8_t *buf, int buf_size),
                     offset_t (*seek)(void *opaque, offset_t offset, int whence))=0;
```

```cpp
30.     virtual AVInputFormat *av_probe_input_format(AVProbeData *pd, int is_opened)=0;
31.     virtual AVInputFormat *av_probe_input_format2(AVProbeData *pd, int is_opened, int *score_max)=0;
32.     virtual int av_probe_input_buffer(AVIOContext *pb, AVInputFormat **fmt, const char *filename, void *logctx, unsigned int offset, u
nsigned int max_probe_size)=0;
33.     virtual void av_dump_format(AVFormatContext *ic, int index, const char *url, int is_output)=0;
34.     virtual int avio_open(AVIOContext **s, const char *filename, int flags)=0;
35.     virtual int avio_close(AVIOContext *s)=0;
36.     virtual int avio_open_dyn_buf(AVIOContext **s)=0;
37.     virtual int avio_close_dyn_buf(AVIOContext *s, uint8_t **pbuffer)=0;
38.     virtual offset_t avio_seek(AVIOContext *s, offset_t offset, int whence)=0;
39.     virtual int avio_read(AVIOContext *s, unsigned char *buf, int size)=0;
40.     virtual void avio_w8(AVIOContext *s, int b)=0;
41.     virtual void avio_write(AVIOContext *s, const unsigned char *buf, int size)=0;
42.     virtual void avio_wb24(AVIOContext *s, unsigned int val)=0;
43.     virtual void avio_wb32(AVIOContext *s, unsigned int val)=0;
44.     virtual void avio_wb16(AVIOContext *s, unsigned int val)=0;
45.     virtual AVFormatContext *avformat_alloc_context(void)=0;
46.     virtual int avformat_alloc_output_context2(AVFormatContext **ctx, AVOutputFormat *oformat, const char *format_name, const char *fi
lename) = 0;
47.     virtual AVStream *avformat_new_stream(AVFormatContext *s, AVCodec *c)=0;
48.     virtual AVOutputFormat *av_guess_format(const char *short_name, const char *filename, const char *mime_type)=0;
49.     virtual int avformat_write_header (AVFormatContext *s, AVDictionary **options)=0;
50.     virtual int av_write_trailer(AVFormatContext *s)=0;
51.     virtual int av_write_frame  (AVFormatContext *s, AVPacket *pkt)=0;
52. #if defined(AVFORMAT_HAS_STREAM_GET_R_FRAME_RATE)
53.     virtual AVRational av_stream_get_r_frame_rate(const AVStream *s)=0;
54. #endif
55. };
56.
57. //封装的Dll，继承了DllDynamic，以及接口
58. class DllAvFormat : public DllDynamic, DllAvFormatInterface
59. {
60.   DECLARE_DLL_WRAPPER(DllAvFormat, DLL_PATH_LIBAVFORMAT)
61.
62.   LOAD_SYMBOLS()
63.
64.   DEFINE_METHOD0(void, av_register_all_dont_call)
65.   DEFINE_METHOD0(void, avformat_network_init_dont_call)
66.   DEFINE_METHOD0(void, avformat_network_deinit_dont_call)
67.   DEFINE_METHOD1(AVInputFormat*, av_find_input_format, (const char *p1))
68.   DEFINE_METHOD1(void, avformat_close_input, (AVFormatContext **p1))
69.   DEFINE_METHOD1(int, av_read_play, (AVFormatContext *p1))
70.   DEFINE_METHOD1(int, av_read_pause, (AVFormatContext *p1))
71.   DEFINE_METHOD1(void, av_read_frame_flush, (AVFormatContext *p1))
72.   DEFINE_FUNC_ALIGNED2(int, __cdecl, av_read_frame, AVFormatContext *, AVPacket *)
73.   DEFINE_FUNC_ALIGNED4(int, __cdecl, av_seek_frame, AVFormatContext*, int, int64_t, int)
74.   DEFINE_FUNC_ALIGNED2(int, __cdecl, avformat_find_stream_info_dont_call, AVFormatContext*, AVDictionary **)
75.   DEFINE_FUNC_ALIGNED4(int, __cdecl, avformat_open_input, AVFormatContext **, const char *, AVInputFormat *, AVDictionary **)
76.   DEFINE_FUNC_ALIGNED2(AVInputFormat*, __cdecl, av_probe_input_format, AVProbeData*, int)
77.   DEFINE_FUNC_ALIGNED3(AVInputFormat*, __cdecl, av_probe_input_format2, AVProbeData*, int, int*)
78.   DEFINE_FUNC_ALIGNED6(int, __cdecl, av_probe_input_buffer, AVIOContext *, AVInputFormat **, const char *, void *, unsigned int, uns
igned int)
79.   DEFINE_FUNC_ALIGNED3(int, __cdecl, avio_read, AVIOContext*, unsigned char *, int)
80.   DEFINE_FUNC_ALIGNED2(void, __cdecl, avio_w8, AVIOContext*, int)
81.   DEFINE_FUNC_ALIGNED3(void, __cdecl, avio_write, AVIOContext*, const unsigned char *, int)
82.   DEFINE_FUNC_ALIGNED2(void, __cdecl, avio_wb24, AVIOContext*, unsigned int)
83.   DEFINE_FUNC_ALIGNED2(void, __cdecl, avio_wb32, AVIOContext*, unsigned int)
84.   DEFINE_FUNC_ALIGNED2(void, __cdecl, avio_wb16, AVIOContext*, unsigned int)
85.   DEFINE_METHOD7(AVIOContext *, avio_alloc_context, (unsigned char *p1, int p2, int p3, void *p4,
86.                   int (*p5)(void *opaque, uint8_t *buf, int buf_size),
87.                   int (*p6)(void *opaque, uint8_t *buf, int buf_size),
88.                   offset_t (*p7)(void *opaque, offset_t offset, int whence)))
89.   DEFINE_METHOD4(void, av_dump_format, (AVFormatContext *p1, int p2, const char *p3, int p4))
90.   DEFINE_METHOD3(int, avio_open, (AVIOContext **p1, const char *p2, int p3))
91.   DEFINE_METHOD1(int, avio_close, (AVIOContext *p1))
92.   DEFINE_METHOD1(int, avio_open_dyn_buf, (AVIOContext **p1))
93.   DEFINE_METHOD2(int, avio_close_dyn_buf, (AVIOContext *p1, uint8_t **p2))
94.   DEFINE_METHOD3(offset_t, avio_seek, (AVIOContext *p1, offset_t p2, int p3))
95.   DEFINE_METHOD0(AVFormatContext *, avformat_alloc_context)
96.   DEFINE_METHOD4(int, avformat_alloc_output_context2, (AVFormatContext **p1, AVOutputFormat *p2, const char *p3, const char *p4))
97.   DEFINE_METHOD2(AVStream *, avformat_new_stream, (AVFormatContext *p1, AVCodec *p2))
98.   DEFINE_METHOD3(AVOutputFormat *, av_guess_format, (const char *p1, const char *p2, const char *p3))
99.   DEFINE_METHOD2(int, avformat_write_header , (AVFormatContext *p1, AVDictionary **p2))
100.  DEFINE_METHOD1(int, av_write_trailer, (AVFormatContext *p1))
101.  DEFINE_METHOD2(int, av_write_frame  , (AVFormatContext *p1, AVPacket *p2))
102. #if defined(AVFORMAT_HAS_STREAM_GET_R_FRAME_RATE)
103.  DEFINE_METHOD1(AVRational, av_stream_get_r_frame_rate, (const AVStream *p1))
104. #endif
105.  BEGIN_METHOD_RESOLVE()
106.    RESOLVE_METHOD_RENAME(av_register_all, av_register_all_dont_call)
107.    RESOLVE_METHOD_RENAME(avformat_network_init,   avformat_network_init_dont_call)
108.    RESOLVE_METHOD_RENAME(avformat_network_deinit, avformat_network_deinit_dont_call)
109.    RESOLVE_METHOD(av_find_input_format)
110.    RESOLVE_METHOD(avformat_close_input)
111.    RESOLVE_METHOD(av_read_frame)
112.    RESOLVE_METHOD(av_read_play)
113.    RESOLVE_METHOD(av_read_pause)
114.    RESOLVE_METHOD(av_read_frame_flush)
115.    RESOLVE_METHOD(av_seek_frame)
116.    RESOLVE_METHOD_RENAME(avformat_find_stream_info, avformat_find_stream_info_dont_call)
117.    RESOLVE_METHOD(avformat_open_input)
```

```
118.        RESOLVE_METHOD(avio_alloc_context)
119.        RESOLVE_METHOD(av_probe_input_format)
120.        RESOLVE_METHOD(av_probe_input_format2)
121.        RESOLVE_METHOD(av_probe_input_buffer)
122.        RESOLVE_METHOD(av_dump_format)
123.      RESOLVE_METHOD(avio_open)
124.      RESOLVE_METHOD(avio_close)
125.        RESOLVE_METHOD(avio_open_dyn_buf)
126.        RESOLVE_METHOD(avio_close_dyn_buf)
127.      RESOLVE_METHOD(avio_seek)
128.      RESOLVE_METHOD(avio_read)
129.      RESOLVE_METHOD(avio_w8)
130.      RESOLVE_METHOD(avio_write)
131.      RESOLVE_METHOD(avio_wb24)
132.      RESOLVE_METHOD(avio_wb32)
133.      RESOLVE_METHOD(avio_wb16)
134.      RESOLVE_METHOD(avformat_alloc_context)
135.        RESOLVE_METHOD(avformat_alloc_output_context2)
136.      RESOLVE_METHOD(avformat_new_stream)
137.      RESOLVE_METHOD(av_guess_format)
138.      RESOLVE_METHOD(avformat_write_header)
139.      RESOLVE_METHOD(av_write_trailer)
140.      RESOLVE_METHOD(av_write_frame)
141.  #if defined(AVFORMAT_HAS_STREAM_GET_R_FRAME_RATE)
142.      RESOLVE_METHOD(av_stream_get_r_frame_rate)
143.  #endif
144.    END_METHOD_RESOLVE()
145.
146.    /* dependencies of libavformat */
147.    DllAvCodec m_dllAvCodec;
148.    // DllAvUtil loaded implicitly by m_dllAvCodec
149.
150.  public:
151.    void av_register_all()
152.    {
153.      CSingleLock lock(DllAvCodec::m_critSection);
154.      av_register_all_dont_call();
155.    }
156.    int avformat_find_stream_info(AVFormatContext *ic, AVDictionary **options)
157.    {
158.      CSingleLock lock(DllAvCodec::m_critSection);
159.      return avformat_find_stream_info_dont_call(ic, options);
160.    }
161.
162.    virtual bool Load()
163.    {
164.      if (!m_dllAvCodec.Load())
165.        return false;
166.      bool loaded = DllDynamic::Load();
167.
168.      CSingleLock lock(DllAvCodec::m_critSection);
169.      if (++m_avformat_refcnt == 1 && loaded)
170.        avformat_network_init_dont_call();
171.      return loaded;
172.    }
173.
174.    virtual void Unload()
175.    {
176.      CSingleLock lock(DllAvCodec::m_critSection);
177.      if (--m_avformat_refcnt == 0 && DllDynamic::IsLoaded())
178.        avformat_network_deinit_dont_call();
179.
180.      DllDynamic::Unload();
181.    }
182.
183.  protected:
184.    static int m_avformat_refcnt;
185.  };
186.
187.  #endif
```

这些宏的含义如下：

```
1.   DEFINE_METHOD0(result, name)         定义一个方法（不包含参数）
2.   DEFINE_METHOD1(result, name, args)   定义一个方法（1个参数）
3.   DEFINE_METHOD2(result, name, args)   定义一个方法（2个参数）
4.   DEFINE_METHOD3(result, name, args)   定义一个方法（3个参数）
5.   DEFINE_METHOD4(result, name, args)   定义一个方法（4个参数）
6.   以此类推...
7.
8.   DEFINE_FUNC_ALIGNED0(result, linkage, name)          定义一个方法（不包含参数）
9.   DEFINE_FUNC_ALIGNED1(result, linkage, name, t1)      定义一个方法（1个参数）
10.  DEFINE_FUNC_ALIGNED2(result, linkage, name, t1, t2)     定义一个方法（2个参数）
11.  以此类推...
```

可以看一下这些宏的定义。看了一会，感觉宏的定义太多了，好乱。在这里仅举一个例子：RESOLVE_METHOD

```cpp
1.  #define RESOLVE_METHOD(method) \
2.    if (!m_dll->ResolveExport( #method , & m_##method##_ptr )) \
3.      return false;
```

从定义中可以看出，调用了m_dll的方法ResolveExport()。但是在DllAvFormat中并没有m_dll变量。实际上m_dll位于DllAvFormat的父类DllDynamic里面。

DllAvFormat继承了DllDynamic。DllDynamic是用于加载Dll的类。我们可以看一下它的定义：

```cpp
1.  //Dll动态加载类
2.  class DllDynamic
3.  {
4.  public:
5.    DllDynamic();
6.    DllDynamic(const CStdString& strDllName);
7.    virtual ~DllDynamic();
8.    virtual bool Load();//加载
9.    virtual void Unload();//卸载
10.   virtual bool IsLoaded() const { return m_dll!=NULL; }//是否加载
11.   bool CanLoad();
12.   bool EnableDelayedUnload(bool bOnOff);
13.   bool SetFile(const CStdString& strDllName);//设置文件
14.   const CStdString &GetFile() const { return m_strDllName; }
15.
16.  protected:
17.    virtual bool ResolveExports()=0;
18.    virtual bool LoadSymbols() { return false; }
19.    bool  m_DelayUnload;
20.    LibraryLoader* m_dll;
21.    CStdString m_strDllName;
22.  };
```

其中有一个变量LibraryLoader* m_dll。是用于加载Dll的。

可以看一DllDynamic中主要的几个函数，就能明白这个类的作用了。

```cpp
1.  //加载
2.  bool DllDynamic::Load()
3.  {
4.    if (m_dll)
5.      return true;
6.
7.    if (!(m_dll=CSectionLoader::LoadDLL(m_strDllName, m_DelayUnload, LoadSymbols())))
8.      return false;
9.
10.   if (!ResolveExports())
11.   {
12.     CLog::Log(LOGERROR, "Unable to resolve exports from dll %s", m_strDllName.c_str());
13.     Unload();
14.     return false;
15.   }
16.
17.   return true;
18. }
```

```cpp
1.  //卸载
2.  void DllDynamic::Unload()
3.  {
4.    if(m_dll)
5.      CSectionLoader::UnloadDLL(m_strDllName);
6.    m_dll=NULL;
7.  }
```

可以看看LibraryLoader的定义。LibraryLoader本身是一个纯虚类，具体方法的实现在其子类里面。

```cpp
//Dll加载类
class LibraryLoader
{
public:
  LibraryLoader(const char* libraryFile);
  virtual ~LibraryLoader();

  virtual bool Load() = 0;
  virtual void Unload() = 0;

  virtual int ResolveExport(const char* symbol, void** ptr, bool logging = true) = 0;
  virtual int ResolveOrdinal(unsigned long ordinal, void** ptr);
  virtual bool IsSystemDll() = 0;
  virtual HMODULE GetHModule() = 0;
  virtual bool HasSymbols() = 0;

  char* GetName(); // eg "mplayer.dll"
  char* GetFileName(); // "special://xbmcbin/system/mplayer/players/mplayer.dll"
  char* GetPath(); // "special://xbmcbin/system/mplayer/players/"

  int IncrRef();
  int DecrRef();
  int GetRef();

private:
  LibraryLoader(const LibraryLoader&);
  LibraryLoader& operator=(const LibraryLoader&);
  char* m_sFileName;
  char* m_sPath;
  int   m_iRefCount;
};
```

LibraryLoader的继承关系如下图所示。

由于自己的操作系统是Windows下的，因此可以看看Win32DllLoader的定义。

```cpp
//Windows下的Dll加载类
class Win32DllLoader : public LibraryLoader
{
public:
  class Import
  {
  public:
    void *table;
    DWORD function;
  };

  Win32DllLoader(const char *dll);
  ~Win32DllLoader();

  virtual bool Load();//加载
  virtual void Unload();//卸载

  virtual int ResolveExport(const char* symbol, void** ptr, bool logging = true);
  virtual bool IsSystemDll();
  virtual HMODULE GetHModule();
  virtual bool HasSymbols();

private:
  void OverrideImports(const CStdString &dll);
  void RestoreImports();
  static bool ResolveImport(const char *dllName, const char *functionName, void **fixup);
  static bool ResolveOrdinal(const char *dllName, unsigned long ordinal, void **fixup);
  bool NeedsHooking(const char *dllName);

  HMODULE m_dllHandle;
  bool bIsSystemDll;

  std::vector<Import> m_overriddenImports;
  std::vector<HMODULE> m_referencedDlls;
};
```

其中加载Dll使用Load()，卸载Dll使用Unload()。可以看看这两个函数具体的代码。

```cpp
1.   //加载
2.   bool Win32DllLoader::Load()
3.   {
4.     if (m_dllHandle != NULL)
5.       return true;
6.     //文件路径
7.     CStdString strFileName = GetFileName();
8.
9.     CStdStringW strDllW;
10.    g_charsetConverter.utf8ToW(CSpecialProtocol::TranslatePath(strFileName), strDllW, false, false, false);
11.    //加载库
12.    m_dllHandle = LoadLibraryExW(strDllW.c_str(), NULL, LOAD_WITH_ALTERED_SEARCH_PATH);
13.    if (!m_dllHandle)
14.    {
15.      LPVOID lpMsgBuf;
16.      DWORD dw = GetLastError();
17.
18.      FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS, NULL, dw, 0, (LPTSTR) &lpMsgBuf, 0, NULL );
19.      CLog::Log(LOGERROR, "%s: Failed to load %s with error %d:%s", __FUNCTION__, CSpecialProtocol::TranslatePath(strFileName).c_str(), dw, lpMsgBuf);
20.      LocalFree(lpMsgBuf);
21.      return false;
22.    }
23.
24.    // handle functions that the dll imports
25.    if (NeedsHooking(strFileName.c_str()))
26.      OverrideImports(strFileName);
27.    else
28.      bIsSystemDll = true;
29.
30.    return true;
31.  }
32.  //卸载
33.  void Win32DllLoader::Unload()
34.  {
35.    // restore our imports
36.    RestoreImports();
37.    //卸载库
38.    if (m_dllHandle)
39.    {
40.      if (!FreeLibrary(m_dllHandle))
41.        CLog::Log(LOGERROR, "%s Unable to unload %s", __FUNCTION__, GetName());
42.    }
43.
44.    m_dllHandle = NULL;
45.  }
```

文章标签： xbmc   ffmpeg   源代码   dll   播放器

个人分类： XBMC   FFMPEG

所属专栏： 开源多媒体项目源代码分析