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

2014年01月09日 00:28:26　阅读数：7123

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）

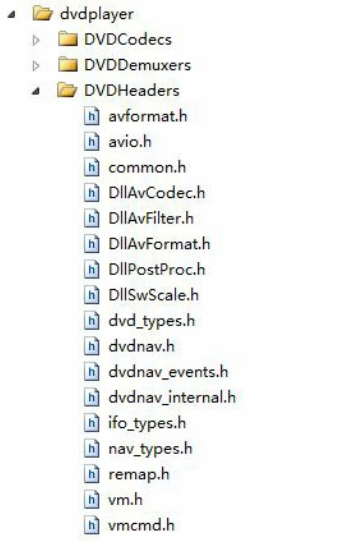
XBMC源代码分析 3：核心部分（core）-综述

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;
```

```
  virtual AVInputFormat *av_probe_input_format(AVProbeData *pd, int is_opened)=0;
  virtual AVInputFormat *av_probe_input_format2(AVProbeData *pd, int is_opened, int *score_max)=0;
  virtual int av_probe_input_buffer(AVIOContext *pb, AVInputFormat **fmt, const char *filename, void *logctx, unsigned int offset, unsigned int max_probe_size)=0;
  virtual void av_dump_format(AVFormatContext *ic, int index, const char *url, int is_output)=0;
  virtual int avio_open(AVIOContext **s, const char *filename, int flags)=0;
  virtual int avio_close(AVIOContext *s)=0;
  virtual int avio_open_dyn_buf(AVIOContext **s)=0;
  virtual int avio_close_dyn_buf(AVIOContext *s, uint8_t **pbuffer)=0;
  virtual offset_t avio_seek(AVIOContext *s, offset_t offset, int whence)=0;
  virtual int avio_read(AVIOContext *s, unsigned char *buf, int size)=0;
  virtual void avio_w8(AVIOContext *s, int b)=0;
  virtual void avio_write(AVIOContext *s, const unsigned char *buf, int size)=0;
  virtual void avio_wb24(AVIOContext *s, unsigned int val)=0;
  virtual void avio_wb32(AVIOContext *s, unsigned int val)=0;
  virtual void avio_wb16(AVIOContext *s, unsigned int val)=0;
  virtual AVFormatContext *avformat_alloc_context(void)=0;
  virtual int avformat_alloc_output_context2(AVFormatContext **ctx, AVOutputFormat *oformat, const char *format_name, const char *filename) = 0;
  virtual AVStream *avformat_new_stream(AVFormatContext *s, AVCodec *c)=0;
  virtual AVOutputFormat *av_guess_format(const char *short_name, const char *filename, const char *mime_type)=0;
  virtual int avformat_write_header (AVFormatContext *s, AVDictionary **options)=0;
  virtual int av_write_trailer(AVFormatContext *s)=0;
  virtual int av_write_frame  (AVFormatContext *s, AVPacket *pkt)=0;
#if defined(AVFORMAT_HAS_STREAM_GET_R_FRAME_RATE)
  virtual AVRational av_stream_get_r_frame_rate(const AVStream *s)=0;
#endif
};

//封装的Dll，继承了DllDynamic，以及接口
class DllAvFormat : public DllDynamic, DllAvFormatInterface
{
  DECLARE_DLL_WRAPPER(DllAvFormat, DLL_PATH_LIBAVFORMAT)

  LOAD_SYMBOLS()

  DEFINE_METHOD0(void, av_register_all_dont_call)
  DEFINE_METHOD0(void, avformat_network_init_dont_call)
  DEFINE_METHOD0(void, avformat_network_deinit_dont_call)
  DEFINE_METHOD1(AVInputFormat*, av_find_input_format, (const char *p1))
  DEFINE_METHOD1(void, avformat_close_input, (AVFormatContext **p1))
  DEFINE_METHOD1(int, av_read_play, (AVFormatContext *p1))
  DEFINE_METHOD1(int, av_read_pause, (AVFormatContext *p1))
  DEFINE_METHOD1(void, av_read_frame_flush, (AVFormatContext *p1))
  DEFINE_FUNC_ALIGNED2(int, __cdecl, av_read_frame, AVFormatContext *, AVPacket *)
  DEFINE_FUNC_ALIGNED4(int, __cdecl, av_seek_frame, AVFormatContext*, int, int64_t, int)
  DEFINE_FUNC_ALIGNED2(int, __cdecl, avformat_find_stream_info_dont_call, AVFormatContext*, AVDictionary **)
  DEFINE_FUNC_ALIGNED4(int, __cdecl, avformat_open_input, AVFormatContext **, const char *, AVInputFormat *, AVDictionary **)
  DEFINE_FUNC_ALIGNED2(AVInputFormat*, __cdecl, av_probe_input_format, AVProbeData*, int)
  DEFINE_FUNC_ALIGNED3(AVInputFormat*, __cdecl, av_probe_input_format2, AVProbeData*, int, int*)
  DEFINE_FUNC_ALIGNED6(int, __cdecl, av_probe_input_buffer, AVIOContext *, AVInputFormat **, const char *, void *, unsigned int, unsigned int)
  DEFINE_FUNC_ALIGNED3(int, __cdecl, avio_read, AVIOContext*, unsigned char *, int)
  DEFINE_FUNC_ALIGNED2(void, __cdecl, avio_w8, AVIOContext*, int)
  DEFINE_FUNC_ALIGNED3(void, __cdecl, avio_write, AVIOContext*, const unsigned char *, int)
  DEFINE_FUNC_ALIGNED2(void, __cdecl, avio_wb24, AVIOContext*, unsigned int)
  DEFINE_FUNC_ALIGNED2(void, __cdecl, avio_wb32, AVIOContext*, unsigned int)
  DEFINE_FUNC_ALIGNED2(void, __cdecl, avio_wb16, AVIOContext*, unsigned int)
  DEFINE_METHOD7(AVIOContext *, avio_alloc_context, (unsigned char *p1, int p2, int p3, void *p4,
                  int (*p5)(void *opaque, uint8_t *buf, int buf_size),
                  int (*p6)(void *opaque, uint8_t *buf, int buf_size),
                  offset_t (*p7)(void *opaque, offset_t offset, int whence)))
  DEFINE_METHOD4(void, av_dump_format, (AVFormatContext *p1, int p2, const char *p3, int p4))
  DEFINE_METHOD3(int, avio_open, (AVIOContext **p1, const char *p2, int p3))
  DEFINE_METHOD1(int, avio_close, (AVIOContext *p1))
  DEFINE_METHOD1(int, avio_open_dyn_buf, (AVIOContext **p1))
  DEFINE_METHOD2(int, avio_close_dyn_buf, (AVIOContext *p1, uint8_t **p2))
  DEFINE_METHOD3(offset_t, avio_seek, (AVIOContext *p1, offset_t p2, int p3))
  DEFINE_METHOD0(AVFormatContext *, avformat_alloc_context)
  DEFINE_METHOD4(int, avformat_alloc_output_context2, (AVFormatContext **p1, AVOutputFormat *p2, const char *p3, const char *p4))
  DEFINE_METHOD2(AVStream *, avformat_new_stream, (AVFormatContext *p1, AVCodec *p2))
  DEFINE_METHOD3(AVOutputFormat *, av_guess_format, (const char *p1, const char *p2, const char *p3))
  DEFINE_METHOD2(int, avformat_write_header , (AVFormatContext *p1, AVDictionary **p2))
  DEFINE_METHOD1(int, av_write_trailer, (AVFormatContext *p1))
  DEFINE_METHOD2(int, av_write_frame  , (AVFormatContext *p1, AVPacket *p2))
#if defined(AVFORMAT_HAS_STREAM_GET_R_FRAME_RATE)
  DEFINE_METHOD1(AVRational, av_stream_get_r_frame_rate, (const AVStream *p1))
#endif
  BEGIN_METHOD_RESOLVE()
    RESOLVE_METHOD_RENAME(av_register_all, av_register_all_dont_call)
    RESOLVE_METHOD_RENAME(avformat_network_init,   avformat_network_init_dont_call)
    RESOLVE_METHOD_RENAME(avformat_network_deinit, avformat_network_deinit_dont_call)
    RESOLVE_METHOD(av_find_input_format)
    RESOLVE_METHOD(avformat_close_input)
    RESOLVE_METHOD(av_read_frame)
    RESOLVE_METHOD(av_read_play)
    RESOLVE_METHOD(av_read_pause)
    RESOLVE_METHOD(av_read_frame_flush)
    RESOLVE_METHOD(av_seek_frame)
    RESOLVE_METHOD_RENAME(avformat_find_stream_info, avformat_find_stream_info_dont_call)
    RESOLVE_METHOD(avformat_open_input)
```

```
    RESOLVE_METHOD(avio_alloc_context)
    RESOLVE_METHOD(av_probe_input_format)
    RESOLVE_METHOD(av_probe_input_format2)
    RESOLVE_METHOD(av_probe_input_buffer)
    RESOLVE_METHOD(av_dump_format)
    RESOLVE_METHOD(avio_open)
    RESOLVE_METHOD(avio_close)
    RESOLVE_METHOD(avio_open_dyn_buf)
    RESOLVE_METHOD(avio_close_dyn_buf)
    RESOLVE_METHOD(avio_seek)
    RESOLVE_METHOD(avio_read)
    RESOLVE_METHOD(avio_w8)
    RESOLVE_METHOD(avio_write)
    RESOLVE_METHOD(avio_wb24)
    RESOLVE_METHOD(avio_wb32)
    RESOLVE_METHOD(avio_wb16)
    RESOLVE_METHOD(avformat_alloc_context)
    RESOLVE_METHOD(avformat_alloc_output_context2)
    RESOLVE_METHOD(avformat_new_stream)
    RESOLVE_METHOD(av_guess_format)
    RESOLVE_METHOD(avformat_write_header)
    RESOLVE_METHOD(av_write_trailer)
    RESOLVE_METHOD(av_write_frame)
#if defined(AVFORMAT_HAS_STREAM_GET_R_FRAME_RATE)
    RESOLVE_METHOD(av_stream_get_r_frame_rate)
#endif
  END_METHOD_RESOLVE()

  /* dependencies of libavformat */
  DllAvCodec m_dllAvCodec;
  // DllAvUtil loaded implicitely by m_dllAvCodec

public:
  void av_register_all()
  {
    CSingleLock lock(DllAvCodec::m_critSection);
    av_register_all_dont_call();
  }
  int avformat_find_stream_info(AVFormatContext *ic, AVDictionary **options)
  {
    CSingleLock lock(DllAvCodec::m_critSection);
    return avformat_find_stream_info_dont_call(ic, options);
  }

  virtual bool Load()
  {
    if (!m_dllAvCodec.Load())
      return false;
    bool loaded = DllDynamic::Load();

    CSingleLock lock(DllAvCodec::m_critSection);
    if (++m_avformat_refcnt == 1 && loaded)
      avformat_network_init_dont_call();
    return loaded;
  }

  virtual void Unload()
  {
    CSingleLock lock(DllAvCodec::m_critSection);
    if (--m_avformat_refcnt == 0 && DllDynamic::IsLoaded())
      avformat_network_deinit_dont_call();

    DllDynamic::Unload();
  }

protected:
  static int m_avformat_refcnt;
};

#endif
```

这些宏的含义如下：

```
[plain]
DEFINE_METHOD0(result, name)        定义一个方法（不包含参数）
DEFINE_METHOD1(result, name, args)  定义一个方法（1个参数）
DEFINE_METHOD2(result, name, args)  定义一个方法（2个参数）
DEFINE_METHOD3(result, name, args)  定义一个方法（3个参数）
DEFINE_METHOD4(result, name, args)  定义一个方法（4个参数）
以此类推...

DEFINE_FUNC_ALIGNED0(result, linkage, name)         定义一个方法（不包含参数）
DEFINE_FUNC_ALIGNED1(result, linkage, name, t1)     定义一个方法（1个参数）
DEFINE_FUNC_ALIGNED2(result, linkage, name, t1, t2)     定义一个方法（2个参数）
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
#define RESOLVE_METHOD(method) \
  if (!m_dll->ResolveExport( #method , & m_##method##_ptr )) \
    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
//Dll动态加载类
class DllDynamic
{
public:
  DllDynamic();
  DllDynamic(const CStdString& strDllName);
  virtual ~DllDynamic();
  virtual bool Load();//加载
  virtual void Unload();//卸载
  virtual bool IsLoaded() const { return m_dll!=NULL; }//是否加载
  bool CanLoad();
  bool EnableDelayedUnload(bool bOnOff);
  bool SetFile(const CStdString& strDllName);//设置文件
  const CStdString &GetFile() const { return m_strDllName; }

protected:
  virtual bool ResolveExports()=0;
  virtual bool LoadSymbols() { return false; }
  bool  m_DelayUnload;
  LibraryLoader* m_dll;
  CStdString m_strDllName;
};
```

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
//加载
bool DllDynamic::Load()
{
  if (m_dll)
    return true;

  if (!(m_dll=CSectionLoader::LoadDLL(m_strDllName, m_DelayUnload, LoadSymbols())))
    return false;

  if (!ResolveExports())
  {
    CLog::Log(LOGERROR, "Unable to resolve exports from dll %s", m_strDllName.c_str());
    Unload();
    return false;
  }

  return true;
}
```

```cpp
//卸载
void DllDynamic::Unload()
{
  if(m_dll)
    CSectionLoader::UnloadDLL(m_strDllName);
  m_dll=NULL;
}
```

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
//加载
bool Win32DllLoader::Load()
{
  if (m_dllHandle != NULL)
    return true;
  //文件路径
  CStdString strFileName = GetFileName();

  CStdStringW strDllW;
  g_charsetConverter.utf8ToW(CSpecialProtocol::TranslatePath(strFileName), strDllW, false, false, false);
  //加载库
  m_dllHandle = LoadLibraryExW(strDllW.c_str(), NULL, LOAD_WITH_ALTERED_SEARCH_PATH);
  if (!m_dllHandle)
  {
    LPVOID lpMsgBuf;
    DWORD dw = GetLastError();

    FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS, NULL, dw, 0, (LPTSTR) &lpMsgBuf, 0, NULL );
    CLog::Log(LOGERROR, "%s: Failed to load %s with error %d:%s", __FUNCTION__, CSpecialProtocol::TranslatePath(strFileName).c_str(), dw, lpMsgBuf);
    LocalFree(lpMsgBuf);
    return false;
  }

  // handle functions that the dll imports
  if (NeedsHooking(strFileName.c_str()))
    OverrideImports(strFileName);
  else
    bIsSystemDll = true;

  return true;
}
//卸载
void Win32DllLoader::Unload()
{
  // restore our imports
  RestoreImports();
  //卸载库
  if (m_dllHandle)
  {
    if (!FreeLibrary(m_dllHandle))
      CLog::Log(LOGERROR, "%s Unable to unload %s", __FUNCTION__, GetName());
  }

  m_dllHandle = NULL;
}
```

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/17512653

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

此PDF由spygg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com