**Kodi播放BD文件详解**

前面界面部分本文不关注，直接从点击ISO文件开始：

**一、选中ISO文件后开始解析**

判断出是一个文件，

**1、执行PlayBackRet CApplication::PlayFile(CFileItem item, const std::string& player, bool bRestart)**

从下面进入BD文件解析：

// a disc image might be Blu-Ray disc

if (!(options.startpercent > 0.0f || options.starttime > 0.0f) && (item.IsBDFile() || item.IsDiscImage()))

{

//check if we must show the simplified bd menu

if (!CGUIDialogSimpleMenu::ShowPlaySelection(const\_cast<CFileItem&>(item)))

return PLAYBACK\_CANCELED;

}

item.IsDiscImage()中执行URIUtils::HasExtension(m\_strPath, ".img|.iso|.nrg")。通过后缀名iso，满足if条件。

**2、进入CGUIDialogSimpleMenu::ShowPlaySelection：**

if (item.IsDiscImage())

{

CURL url2("udf://");

url2.SetHostName(item.GetPath());

url2.SetFileName("BDMV/index.bdmv");

if (XFILE::CFile::Exists(url2.Get()))

{

url2.SetFileName("");

CURL url("bluray://");

url.SetHostName(url2.Get());

url.SetFileName("root");

return ShowPlaySelection(item, url.Get());

}

}

第一次进来，给url重新命名后得到：

url.Get() = bluray://udf%3a%2f%2f%252fmnt%252fusb%252fsdb1%252fPR41HLE2.bluray.iso%2f/root

含有root。

进入ShowPlaySelection中获取iso中目录结构，显示出iso文件列表，while循环等待用户选择。

**3、进入CDirectory::GetDirectory(directory, items, XFILE::CDirectory::CHints(), true)**

realURL是bluray://udf%3a%2f%2f%252fmnt%252fusb%252fsda1%252fPR41HLE2.bluray.iso%2f/root

**4、CDirectoryFactory::Create(realURL)**

CFileItem item(url.Get(), true);这一步会设置Protocol为"bluray"

这个判断成立if (url.IsProtocol("bluray")) return new CBlurayDirectory();

返回第3步、CDirectory::GetDirectory进入while循环

if (g\_application.IsCurrentThread() && allowThreads && !URIUtils::IsSpecial(pathToUrl))条件满足allowThreads是true

**CGetDirectory get(pDirectory, realURL, url);这里新开一个线程，执行读取工作：**

**if (!CGUIDialogBusy::WaitOnEvent(get.GetEvent(), TIME\_TO\_BUSY\_DIALOG))当前线程进入等待，block。**

m\_id = CJobManager::GetInstance().AddJob(new CGetJob(imp, m\_result), NULL, CJob::PRIORITY\_HIGH);

CGetJob::DoWork()中执行m\_imp->GetDirectory走到CBlurayDirectory.GetDirectory (m\_result->m\_dir, m\_result->m\_list)

**注意：下面再一个新的线程里跑**

**5、进入CBlurayDirectory.GetDirectory**

std::string root = m\_url.GetHostName(); // udf://%2fmnt%2fusb%2fsda1%2fPR41HLE2.bluray.iso

std::string file = m\_url.GetFileName(); //root

m\_dll = new DllLibbluray();

if (!m\_dll->Load())

**6、class DllLibbluray : public DllDynamic, DllLibblurayInterface**

**A、DllLibbluray.h：**

1、DECLARE\_DLL\_WRAPPER(DllLibbluray, DLL\_PATH\_LIBBLURAY)

定义DllLibbluray构造函数，给基类DllDynamic填充so库的name

2、接着是DEFINE\_METHOD1、2、3、4等等，看看这些宏的实现：

以DEFINE\_METHOD2(BLURAY\*, bd\_open, (const char\* p1, const char\* p2))为例。

#define DEFINE\_METHOD2(result, name, args) DEFINE\_METHOD\_LINKAGE2(result, \_\_cdecl, name, args)

#define DEFINE\_METHOD\_LINKAGE2(result, linkage, name, args) \

DEFINE\_METHOD\_LINKAGE\_BASE(result, linkage, name, args, (p1, p2))

#define DEFINE\_METHOD\_LINKAGE\_BASE(result, linkage, name, args, args2) \

protected: \

typedef result (linkage \* name##\_METHOD) args; \

union { \

name##\_METHOD m\_##name; \

void\* m\_##name##\_ptr; \

}; \

public: \

virtual result name args \

{ \

return m\_##name args2; \

}

替换以后得到：

typedef BLURAY\* (\_\_cdecl \* bd\_open\_METHOD) (const char\* p1, const char\* p2);

union {

bd\_open \_METHOD m\_ bd\_open;

void\* m\_ bd\_open \_ptr;

};

public:

virtual BLURAY \* bd\_open (const char\* p1, const char\* p2)

{

return m\_ bd\_open (p1, p2);

}

这里是对基类DllLibblurayInterface中纯虚函数的覆盖定义，注意里面的union。

3、BEGIN\_METHOD\_RESOLVE()和END\_METHOD\_RESOLVE()加上中间的内容，定义了一个函数如下：

virtual bool ResolveExports()

{

…

RESOLVE\_METHOD\_RENAME(bd\_open, bd\_open)

RESOLVE\_METHOD\_RENAME定义{

#define RESOLVE\_METHOD\_RENAME(dllmethod, method) \

if (!m\_dll->ResolveExport( #dllmethod , & m\_##method##\_ptr )) \

return false;

}

所以调用RESOLVE\_METHOD\_RENAME相当于：

if (!m\_dll->ResolveExport(“bd\_open” , & m\_ bd\_open \_ptr ))

return false;

…

return true;

}

DllLibbluray还有一些公有的静态方法：file\_close、bluray\_logger等。

**B、根据A**的分析，new DllLibbluray()后DllDynamic中的m\_strDllName是libbluray.so

**C、m\_dll->Load()**

这里这个m\_dll是DllLibbluray，DllLibbluray没有实现load，调用基类DllDynamic::Load()，DllDynamic中有一个LibraryLoader\* m\_dll

1、CSectionLoader::LoadDLL

层层调用：CSectionLoader::LoadDLL、DllLoaderContainer::LoadModule、DllLoaderContainer::LoadDll，LoadDll中有三个loader类：

new SoLoader(sName, bLoadSymbols); //ANDROID走这里。

new Win32DllLoader(sName, false);

new DllLoader(sName, m\_bTrack, false, bLoadSymbols);

LibraryLoader\* m\_dll的值是SoLoader，SoLoader::Load()完成对libbluray.so的加载。

2、if (!ResolveExports())

ResolveExports，就是刚才在DllLibbluray.h中用宏定义实现的方法，遍历所有定义过的方法，执行m\_dll->ResolveExport。以bd\_open为例：

if (!m\_dll->ResolveExport(“bd\_open” , & m\_ bd\_open \_ptr ))

return false;

3、SoLoader::ResolveExport

SoLoader::ResolveExport(bd\_open, &m\_ bd\_open \_ptr, true)

{

if (!m\_bLoaded && !Load())，检查有没有load过。

void\* s = dlsym(m\_soHandle, bd\_open);

{

FARPROC symbol = GetProcAddress( myhandle, bd\_open); //得到动态库libbluray.so中bd\_open方法的函数地址

return (void\*)symbol

}

m\_ bd\_open \_ptr = s; //把得到的函数地址赋值给m\_ bd\_open \_ptr

}

而m\_bd\_open\_ptr在宏DEFINE\_METHOD2(BLURAY\*,bd\_open,(const char\* p1, const char\* p2))中定义。

union {

bd\_open \_METHOD m\_ bd\_open;

void\* m\_ bd\_open \_ptr;

};

m\_bd\_open\_ptr既是m\_ bd\_open，所以下面的方法，如果被调用到，实际上调用的是libbluray.so中的bd\_open。

virtual BLURAY\* bd\_open (const char\* p1, const char\* p2)

{

return m\_ bd\_open(p1, p2);

}

4、总结。

Class DllLibbluray完成与libbluray.so中的方法对接，扩充的方法有file\_close、file\_seek、bluray\_logger等。

回到**CBlurayDirectory.GetDirectory**

**7、CBlurayDirectory.GetDirectory**

m\_dll->bd\_register\_dir(DllLibbluray::dir\_open); //替换libbluray中的dir\_open

m\_dll->bd\_register\_file(DllLibbluray::file\_open); //替换libbluray中的file\_open

m\_dll->bd\_set\_debug\_handler(DllLibbluray::bluray\_logger); //替换libbluray中的log\_func，log输出转移到kodi中

m\_dll->bd\_set\_debug\_mask(DBG\_CRIT | DBG\_BLURAY | DBG\_NAV); //debug信息输出级别设置

m\_bd = m\_dll->bd\_open(root.c\_str(), NULL);

这里这个m\_dll是DllLibbluray

根据之前的分析，会直接调用到libbluray中的bd\_register\_dir等。

**8、m\_dll->bd\_open中回调DllLibbluray::dir\_open**

dirname是udf://%2fmnt%2fusb%2fsda1%2fPR41HLE2.bluray.iso/

if(!CDirectory::GetDirectory(strDirname, st->list))又回到了CDirectory::GetDirectory

这次的路径变为：udf://%2fmnt%2fusb%2fsda1%2fPR41HLE2.bluray.iso/，与之前相比少了root

if (g\_application.IsCurrentThread() && allowThreads && !URIUtils::IsSpecial(pathToUrl))条件不满足allowThreads是false

走else

result = pDirectory->GetDirectory(realURL, items);

再次回到**CBlurayDirectory.GetDirectory：关闭之前的bd\_close，注意这一次打开失败了，之后返回dir\_open**

**Libbluray中再次调用dir\_open**

**路径是：udf://%2fmnt%2fusb%2fsda1%2fPR41HLE2.bluray.iso/BDMV/PLAYLIST**

这次**bd\_open打开成功之后，通过GetRoot(items)、GetTitles、bd\_get\_titles、bd\_get\_title\_info获取BLURAY\_TITLE\_INFO \*t信息，根据t->duration找到最长的一个title就是主title，我们要播放的主m2ts文件**

**把main title信息存入CFileItemList &items中。剩下的清除掉bd\_free\_title\_info**

层层返回，又回到CDirectory::GetDirectory

9、CDirectory::GetDirectory

if (!CGUIDialogBusy::WaitOnEvent(get.GetEvent(), TIME\_TO\_BUSY\_DIALOG))

主线程这里唤醒后，继续执行

result = get.GetDirectory(items);

这里只是一个copy动作，把上面7处获取到的包含BD main titles信息的**CFileItemList &items**信息复制到函数参数items中。While循环退出，函数返回

10、回到CGUIDialogSimpleMenu::ShowPlaySelection

进入while (true)，显示获取到的播放信息，用户选中main title播放。

11、播放main title

选择main title后，从ShowPlaySelection返回到CApplication::PlayFile

newPlayer = CPlayerCoreFactory::GetInstance().GetDefaultPlayer(item);//得到默认player：VideoPlayer

m\_pPlayer->CreatePlayer(newPlayer, \*this);// IPlayerCallback参数是this，CApplication对象 public IPlayerCallback

pPlayer = new CVideoPlayer(callback);pPlayer->Initialize(m\_config)，结果是true，CreatePlayer直接返回CVideoPlayer对象，保存在std::shared\_ptr<IPlayer> m\_pPlayer中。

m\_pPlayer->OpenFile(item, options)；内部调用player->OpenFile(item, options)，这里使用CVideoPlayer打开选中的main title，

VideoPlayer: Opening: bluray://udf%3a%2f%2f%252fmnt%252fusb%252fsda1%252fPR41HLE2.bluray.iso%2f/BDMV/PLAYLIST/00003.mpls