2013年9月27日星期五

亚星回复的邮件:

1. 挂接事件时候，根据上下文来看，是要求栈上第三个参数是函数，但是传递给DataReadyListener时候，没有明确说明是使用luastate上第几个参数，这里依赖了隐式的规则，容易出错

另外luaL\_ref只是对栈顶进行操作，并且会弹出栈顶，而第三个参数一定是栈顶？这种情况下要求先使用lua\_pushvalue把指定位置的参数拷贝到栈顶，然后再lua\_ref

if(!lua\_isfunction(luaState, 3))//在模板中是2, 用模板的时候lua端没有加类名参数, 用接口的时候加了

                   {

                            return 0;

                   }

                   //long functionRef =  luaL\_ref(luaState, LUA\_REGISTRYINDEX);

                   pDataModelClass->SetDataBatchReadyListener(new DataReadyListener(LuaDataModelClass::LuaDataReadyListener, luaState));

……

m\_functionRef = luaL\_ref(m\_luaState, LUA\_REGISTRYINDEX);

疑问: 如果在new DataReadyListener之前有如下调用:

lua\_toString(luaState, 2);

luaL\_checkudata(luaState, 1, className);

lua\_isfunction(luaState, 3);

当前栈顶元素是什么? 为什么?

以上三个调用都不会改变lua栈的结构, 正索引在栈底, 类型检查和取值不会从栈中弹出, 这和我之前假设的不一样.

函数在不在栈顶和上面的调用都没有关系, 是因为lua中的调用是:

userdata.class:SetDataBatchReadyListener(userdata.className , dataBatchReadyListener)

这个调用压栈的顺序是: dataModelClass, userdata.className, dataBatchReadyListener, 如果调用时在后面增加一个参数ARG, 从栈底开始的元素是dataModelClass, userdata.className, dataBatchReadyListener, ARG, 栈顶就不是函数了.

1. XL\_LRT\_ENV\_HANDLE hEnv = XLLRT\_GetEnv(NULL);

luaruntime库的类型都是基于引用计数，一个原则是所有返回值都会增持引用计数，使用完毕要释放

已释放

1. void LuaDataModelClassFactory::RegisterObj(const char \*dataModelClassName, XL\_LRT\_ENV\_HANDLE hEnv)

{

    if(hEnv == NULL)

         {

        return ;

         }

    XLLRTObject factoryObject;

         char factoryClassName[100];

         strcpy(factoryClassName, dataModelClassName);

         strcat(factoryClassName, ".Factory.Class");

典型的缓冲区溢出的风险，需要修正为更安全的代码

已改为strcpy\_s

疑问: 是否使用xl\_lib提供的string\_utility?

1. std::wstring wplaylistName;

         xl::text::transcode::UTF8\_to\_Unicode(fileName, MAX\_PATH, wplaylistName);

MAX\_PATH应为fileName字符串的长度

已修改

void XmlParser::OnStartElement(const XML\_Char \*pszName, const XML\_Char \*\*papszAttrs)

{

         if (strcmp(pszName, this->attrNamesOfColumn[MEDIA\_TAG]) == 0)

         {

                   StrSongInfo \*info = new StrSongInfo();

……

m\_playlist->push\_back(\*info);

这里先new了StrSongInfo，但是最后只拷贝了一份到playlist里面，存在内存泄露

另外StrSongInfo内部存在字符串等复杂对象，所以std::vector<StrSongInfo> m\_playlist;  最好改为使用指针形式存储std::vector<StrSongInfo\*> m\_playlist;  效率上会更好

改用指针形式存储playlist.

1. 后台窗口的实现问题，在DataModel这个模块里面，只需要创建一个后台窗口即可

MyMain函数里面使用了atl应用程序的流程，还有appmodule，这些都没必要，消息循环也没有托管到DataModel里面，所以都是无意义的，去掉即可

去掉了

疑问: 是不是DataModel实现CMessageFilter就可以加到Module.GetMessageLoop()里了?是否可以不实现隐藏窗口呢, 只是先CMessageFilter?

1. C/C++代码规范的一些问题

下面提到的问题都改了. 代码规范1.0中的其他部分正在修改中.

virtual void onDataReady(const int & arg1, const int & arg2)   函数名首字母须大写

funcDataReadyCallback  函数指针类型，须全用大写字母

if (!dataModelClassName)

                   dataModelClassName = "XmlDataModel";

if (from <= 0) from = 1;

if (to > pDataModelClass->GetCount()) to =  pDataModelClass->GetCount();

单行语句的块也需要用花括号，不能省略

if (strcmp(types[column-1], "string") == 0) //这一段把值压栈

                                               {

                                                        char\* cData = (char\*)itemData;

                                                        lua\_pushstring(L, cData);

                                               } else if (strcmp(types[column-1], "bitmap") == 0)

else须在下一行开始

static XLLRTGlobalAPI mLuaDataModelClassMemberFunctions[];

静态成员变量须用s\_xxx来命名

char \* attrNamesOfColumn[MAX\_COL];  类成员函数，须用m\_xxx来命名

enum {

                   MEDIA\_TAG = 0,

                   COVER\_COL = 1,

                   NAME\_COL = 2,

                   SOURCE\_COL = 3,

                   MAX\_COL

         } COL\_NAMES;

第一个括号须使用悬垂式对齐的括号风格

if (m\_tOLogFile.is\_open() == false)

布尔类型不要和true和flase比较，直接 if (m\_tOLogFile.is\_open())即可

const char\* lhs = static\_cast<const char\*> (lua\_tostring(luaState, 2));

lua\_tostring本身返回的就是const char\*类型，无须再转换类型

if(PostMessageToUIThread && m\_callbackToDataModelOnSingleDataReady)

PostMessageToUIThread是全局函数不可能为空，冗余判断

LuaDir::DeleteSelf(lua\_State \*luaState)

对象泄露

#include ".\xl\_lib\text\transcode.h"

路径分隔符须使用斜杠 /

2013年9月26日星期四

亚星指出的:

ItemView改名为ListView // 遵循MFC等主流UI框架的命名

DataModelInterface 的纯虚析构方法改成虚方法 // 纯虚析构方法在场景”这个类必须是抽象类, 但是我没有合适的虚方法”时是有用的, 但是此处没有必要.

宽字符和utf8互相转换时不必要的操作 // 清理一下

SetListener 的参数从含义不明的UserData改成接口类 // 覆盖listener时可以通过销毁接口类指针销毁listener对象, 如果只是DWORD类型的UserData会造成内存泄露

实现类要保留虚函数关键字 // 为了代码阅读方便, 不引起混淆

lua封装类接口~~const~~ // 这样做会限制实现类的行为, 封装类不应如此

XmlDataModel :: m\_playlist 管理 // loader持有playlist的引用是为了方便加卸载, playlist的创建和销毁应该留在XmlDataModel里处理

lua function ref/unref // function的ref和unref要成对出现

用playlist.clear() 代替while循环销毁playlist中的对象 // 容器中的对象有指针成员且指针指向对象需要被销毁时, 还是要用while循环销毁, 容器中的对象只有对象成员时可以用clear()

GetItemAtIndex(void \*\*itemData) 的内存泄露 // 对于string, push到lua环境中后要释放掉为此开辟的新内存

春哥指出的:

TreeView在scroll range发生变化时scrollbar的thumb button大小不会发生变化, 要动一下才会变化 // 在setScrollRange时更新一下