0 基础

▲关于char和WCHAR

typedef wchar\_t WCHAR; // 宽字符, 16-bit UNICODE character

比如，fopen函数有2个版本，一个是char的，一个wchar\_t的版本；

这个是宽字符版本：

FILE \*fp = NULL;

fp = \_wfopen(pPath, \_T("r")); // pPath的类型为WCHAR

这个是普通char的版本：

char pPath[1000];

strcpy(pPath, "C:\\mschosencity");

FILE \*fp = fopen(pPath, "w");

▲字符串的类型

△LPCTSTR aa = L"abcd";//这是一个Unicode字符串，L表示long[指针](https://baike.baidu.com/item/%E6%8C%87%E9%92%88) P表示这是一个指针C表示是一个常量T表示在Win32环境中， 有一个\_T宏STR表示这个变量是一个字符串，CString 和 LPCTSTR 可以说通用：

CString 转LPCTSTR:

CString cStr;

const char \*lpctStr=(LPCTSTR)cStr;

LPCTSTR转CString:

LPCTSTR lpctStr;

CString cStr=lpctStr;

[常量](https://baike.baidu.com/item/%E5%B8%B8%E9%87%8F)字符串ansi和unicode的区分是由宏\_T来决定的。但是用\_T("abcd")时， 字符串"abcd"就会根据编译时是否定是\_UNICODE来决定是char\* 还是 wchar\_t\*。 同样，TCHAR 也是相同目的[字符](https://baike.baidu.com/item/%E5%AD%97%E7%AC%A6)宏。

ansi情况下，LPCTSTR 就是 const char\*, 是[常量](https://baike.baidu.com/item/%E5%B8%B8%E9%87%8F)字符串（不能修改的）。

而LPTSTR 就是 char\*, 即普通字符串（非常量，可修改的）

这两种都是基本类型， 而CString 是 C++类， 兼容这两种基本类型是最起码的任务了。

由于const char\* 最简单（常量，不涉及内存变更，操作迅速）

△char sttemp[10] = { '0' }; char \* pchar

△WCHAR WCHAR \* pwchar

△string

△CString

▲typedef为现有类型创建一个新的名字，用途，有4种

△定义一种类型的别名

typedef char\* PCHAR

CHAR pa, pb;

△用在旧的C的代码中帮助struct，enum

typedef struct tagPOINT

{

int x;

int y;

}POINT;

POINT p1;

△用typedef来定义与平台无关的类型  
比如定义一个叫 REAL 的浮点类型，在目标平台一上，让它表示最高精度的类型为：  
typedef long double REAL;   
在不支持 long double 的平台二上，改为：  
typedef double REAL;   
在连 double 都不支持的平台三上，改为：  
typedef float REAL;   
也就是说，当跨平台时，只要改下 typedef 本身就行，不用对其他源码做任何修改。  
标准库就广泛使用了这个技巧，比如size\_t。  
另外，因为typedef是定义了一种类型的新别名，不是简单的字符串替换，所以它比宏来得稳健

△为复杂的声明定义一个新的简单的别名。

afxinet.h的系统目录：

D:\Program Files (x86)\Microsoft Visual Studio\2017\Professional\VC\Tools\MSVC\14.13.26128\atlmfc\include

Windows.h的系统目录：

C:\Program Files (x86)\Windows Kits\10\Include\10.0.16299.0\um

Windowsx.h的系统目录：（#include <windowsx.h>）

C:\Program Files (x86)\Windows Kits\10\Include\10.0.16299.0\shared

String的系统目录：（#include <string>）

D:\Program Files (x86)\Microsoft Visual Studio\2017\Professional\VC\Tools\MSVC\14.13.26128\include

1. 基本概念和专有名词  
   GDI ：Windows的图形设备接口  
   API：Application Programming Interface -- 应用程序编程接口  
   SDK：Software Development Kit -- 软件开发工具包  
   MFC：Microsoft Foundations Classes -- 微软基础类
2. 变量或者叫标识符\_常量（在windows头中定义，通常是数字）  
   CS 窗口类别样式  
   CW\_ 建立窗口  
   DT \_ 绘制文字 DT\_CENTER  
   IDI\_ 图示ID  
   IDC\_ 游标ID  
   MB \_ 消息框  
   SND\_ 声音  
   WM\_ 窗口消息 WM\_PAINT  
   WS\_ 窗口样式
3. Windows的数据类型\_变量：  
   a.基本数据类型：（typedef叙述或者#define定义的某个基本类型）  
   **Windows Data Types**在MSDN LB  
   ms-help://MS.MSDNQTR.v90.chs/winprog/winprog/windows\_data\_types.htm  
     
   b.结构体数据类型：（typedef定义的struct）  
   MSG 消息结构  
   WNDCLASS 窗口类别结构   
   PAINTSTRUCT 绘图结构   
   RECT 矩形结构  
     
   比如：MSG的原型：  
   typedef struct {   
    HWND hwnd;  
    UINT message;  
    WPARAM wParam;  
    LPARAM lParam;  
    DWORD time;  
    POINT pt;   
   } MSG, \*PMSG;
4. 句柄\_变量：（句柄也是一种数据类型，其实是一个指针）  
   HINSTANCE 执行实体（程序自身）句柄  
   HWND窗口句柄  
   HDC 设备内容句柄  
   HICON图标句柄  
   HCURSOR鼠标光标句柄  
   HBRUSH画刷句柄
5. 匈牙利表示法  
   c char或WCHAR或TCHAR by BYTE （无正负号字符） n short i int x, y int分别用作x坐标和y坐标 cx, cy int分别用作x长度和y长度；C代表「计数器」 b或f BOOL (int)；f代表「旗标」 w WORD （无正负号短整数） l LONG （长整数） dw DWORD （无正负号长整数） fn function（函数） s string（字符串） sz 以字节值0结尾的字符串 h 句柄 p 指标
6. 待建立  
   1、捕获左键，弹出对话框  
   case WM\_LBUTTONDOWN:  
   MessageBox(NULL, TEXT("WM\_LBUTTONDOWN"), TEXT("HelloMsg"), 0);  
   return 0;
7. 待理解概念  
   1、静态成员变量和函数  
    类的静态成员变量只有一份，被所有同类对象共享，static int nTotalArea;  
    成员函数，static void PrintTotal ();不具体作用在某个对象上  
    两者的使用都必须带上类名， 类名::静态成员变量/函数  
   2、作用域限制符 ::  
    如果局部变量和全局变量重名，可以使用域限制符来使用全局变量  
   3、虚拟析构函数  
   4、内联  
   5、重复包涵头文件以及解决办法  
   6、typedef是撒意思： typedef CControlUI\* (CALLBACK\* FINDCONTROLPROC)(CControlUI\*, LPVOID);  
   7、重载运算符
8. 常用的函数  
   ①string转换为int  
   std::string str1 = "45";  
   int myint1 = std::stoi(str1);  
   ②int转换为string  
   #include <string>  
   std::string s = std::to\_string(42);   
   不包涵sting  
   CString \_temp = "100";  
   int \_int;  
   \_int = atoi(\_temp);
9. 解析：  
     
   1、文件：Test.cpp  
   主函数\_tWinMain()，①初始化CPaintManagerUI的参数，②new一个BaseDialog  
   ③调用了BaseDialog的UIBase::Create()、CWindowWnd::CenterWindow()、CWindowWnd::showWindow()、④最后调用静态成员函数CPaintManagerUI::MessageLoop()进入消息循环  
     
   2、UIlib文件  
   h文件包涵了所有的其他头文件，cpp文件定义了dll的入口   
     
   3、BaseDialog.cpp  
   此文件包涵UIlib.h，并使用DuiLib的命名空间，是CWindowWnd和INotifyUI的子类，并声明一个CPaintManagerUI类作为自己的成员变量，  
   函数调用了，BaseDialog::OnCreate()，  
   CDialogBuilder builder; //实例化后  
   CControlUI\* pRoot = builder.Create(\_T("main\_dlg.xml"), (UINT)0, NULL, &m\_pm);  
     
   4、文件UIDlgBuilder  
     
   5、文件UIManager.cpp  
   定义了通知类INotifyUI  
   定义了UI管理类 CPaintManagerUI  
   加载图片位置的函数：CPaintManagerUI::SetResourceZip(\_T("skin.zip"));   
     
   6、文件UIBase.cpp   
   定义了窗口类CWindowWnd  
     
   消息循环函数 ：\_\_WndProc ； 在程序1066行

窗口注册：

在UIBase.cpp的1027行，bool CWindowWnd::RegisterWindowClass()

消息处理程序设定为： wc.lpfnWndProc = CWindowWnd::\_\_WndProc; 1066

RegisterWindowClass这个函数在 Create中被调用，并调用m\_hWnd = ::CreateWindowEx(dwExStyle, GetWindowClassName(), pstrName, dwStyle, x, y, cx, cy, hwndParent, hMenu, CPaintManagerUI::GetInstance(), this);

BaseDialog.cpp中

CDialogBuilder builder;

CControlUI\* pRoot = builder.Create(\_T("main\_dlg.xml"), (UINT)0, NULL, &m\_pm);

解析xml文件

UIDlgBuilder.cpp

319行，解析到哪只类型，生成那种对象

UICombo.cpp

读取配置文件，生成

UIDlgBuilder.cpp中

CButtonUI类在 UICommonControls.cpp

控制消息的函数在这里：

BaseDialog.cpp，这里给出了3个按钮的消息相应，触发

void BaseDialog::Notify(TNotifyUI& msg)

{

if ( msg.sType == \_T("click"))

{

if( msg.pSender == static\_cast<CButtonUI\*>(m\_pm.FindControl(\_T("minbtn"))) )

SendMessage(WM\_SYSCOMMAND, SC\_MINIMIZE, 0);

if( msg.pSender == static\_cast<CButtonUI\*>(m\_pm.FindControl(\_T("closebtn"))) )

PostQuitMessage(0);

if( msg.pSender == static\_cast<CButtonUI\*>(m\_pm.FindControl(\_T("maxbtn"))) )

::IsZoomed(\*this) ? SendMessage(WM\_SYSCOMMAND, SC\_RESTORE, 0) : SendMessage(WM\_SYSCOMMAND, SC\_MAXIMIZE, 0);

}

}

在字符串前加一个L作用:    
  如 L"我的字符串" 表示将ANSI字符串转换成unicode的字符串，就是每个字符占用两个字节。    
  strlen("asd") = 3;     
  strlen(L"asd") = 6;

[**（转）读取注册表判断主机安装了哪些浏览器**](https://www.cnblogs.com/xiaoCCBlog/articles/9355119.html)

因为有些软件 不是安装在 HKEY\_lOACAL\_MACHINES下面，而是安在： HKEY\_UERS\S-1-5-21-\*-500\Software\Microsoft\Windows\CurrentVersion\Uninstall\下面

普通字符显示乱码，通过CSting转宽字符  
//⑴char 赋给CString，否则中文会是乱码  
char \* abc1 = "abcdf";  
CString abc = abc1;  
MessageBox(NULL, (LPCTSTR)abc, \_T("Message2"), 0);  
// w\_chart  
wchar\_t \* bcd = L"abcdef";  
MessageBox(NULL, (LPCTSTR)bcd, \_T("Message3"), 0);

**一些 duilib的学习网站：**

**2013 duilib入门简明教程**

<https://www.cnblogs.com/Alberl/p/3341956.html>

# [duilib中各控件响应的消息类型](https://www.cnblogs.com/MuyouSome/p/3424137.html)

|  |  |  |
| --- | --- | --- |
| **消息** | **说明** | **Sender** |
| click | 鼠标点击 | CButtonUI |
| dropdown | 下拉显示 | CComboUI |
| headerclick | 点击列标题 | CListHeaderItemUI |
| itemactivate |  | CListElementUI、CListContainerElementUI |
| itemclick | 单击选项 | CListLabelElementUI、CListContainerElementUI |
| itemselect | 选择选项 | CComboUI、CListUI |
| killfocus | 失去焦点 | CControlUI |
| link |  | CTextUI、CListTextElementUI |
| menu |  | CButtonUI、CControlUI |
| return | 回车 | CEditWnd、CRichEditUI |
| scroll | 滚动 | CScrollBarUI |
| selectchanged | 变更选项 | COptionUI |
| setfocus | 获得焦点 | CControlUI |
| showactivex |  | CActiveXUI |
| textchanged | 文本被改变 | CEditWnd |
| tabselect | 标签页被选中 | CTabLayoutUI |
| timer |  | CControlUI |
| valuechanged | 值发生变化 | CSliderUI |
| windowinit | 窗体初始化 |  |

# [Duilib中各个类的简单介绍](https://www.cnblogs.com/depend-wind/articles/9008272.html)

DirectUI意为直接在父窗口上绘图(Paint on parent dc directly)。即子窗口不以窗口句柄的形式创建(windowless)，只是逻辑上的窗口，绘制在父窗口之上。微软的“DirectUI”技术广泛的应用于Windows XP、Vista、Windows 7，如浏览器左侧的TaskPanel，控制面板导航界面，Media Player播放器，即时通讯工具MSN Messager等。

　　DirectUI好处在于可以很方便的构建高效、绚丽的、非常易于扩展的界面。国外如微软，国内如腾讯、百度等公司的客户端产品多采用这种方式来组织界面，从而很好的将界面和逻辑分离，同时易于实现各种超炫的界面效果如换色、换肤、透明等。

DirectUI可以理解为一个轻量级的WPF，可以让C++做出C#般绚丽的界面。

　　目前国内做DirectUI皮肤库的公司主要有两家：第一家是UIPower（即Skin++的公司），网址[www.uipower.com](http://www.uipower.com/)，其DirectUI产品网站[www.directui.com](http://www.directui.com/)，做的很专业；另一家是UIEASY，网址[www.uieasy.com](http://www.uieasy.com/)，一个轻量级皮肤库DSkinLite。但是，网上关于DirectUI的技术文章却很少；早在 6/26/2005，一个国外的程序员就已经写了一个开源的DirectUI例子，应该算是国内所有directui界面库的起源了，网址<http://www.viksoe.dk/code/windowless1.htm>，探讨一种实现的可能性和思路,和实际应用虽有一定距离,不过却是一个很好的学习资料。再后来，一个国内的程序员“当个傻瓜蛋”，基于此代码，进行重构和修正Bug，并放到了google上开源，网址<http://code.google.com/p/duilib/>，下载地址svn checkout <http://duilib.googlecode.com/svn/trunk/> duilib-read-only。

本人通过学习这个duilib的源代码，学到了很多知识、思路和经验，分享给看到此文的朋友，共同探讨DirectUI技术，并欢迎大家加入到开发开源DirectUI皮肤库的行列当中。分析难免有误，见谅！

一、**核心类**

　　1. CWindowWnd，窗口对象管理父类，主要作用：

　　　　1) 创建窗口。

　　　　2) 窗口消息过程处理。

　　　　3) 提供窗口子类化与超类化接口。

　　 2. CDialogBuilder，控件布局类，主要作用：

　　　　1) 读取XML脚本，分析脚本，构建控件树。

　　　　2) 创建控件对象。

　　3. CPaintManagerUI，窗口消息及图形绘制管理器类，与窗口绑定，主要作用：

　　　　1) 绘制控件。

　　　　2) 消息管理。

　　　　3) 事件通知。

　　4. INotifyUI，事件通知抽象类，主要作用：

　　　　1) 重载Notify虚函数，处理事件通知。

二、**控件类**

　　　1. CControlUI，控件管理父类，主要作用：

　　　　1) 控件的通用基类，提供控件通用属性管理。

　　　2. CLabelUI，静态标签类，父类CControlUI。

　　　3. CButtonUI，按钮类，父类CLabelUI。

　　　4. COptionUI，选择按钮类，父类CButtonUI。

　　　5. CTextUI，静态文本类，父类CLabelUI。

　　　6. CProgressUI，进度条类，父类CLabelUI。

　　　7. CSliderUI，父类CProgressUI。

　　　8. CEditUI，编辑框类，父类CLabelUI。

　　　9. CListUI，列表框类，父类CVerticalLayoutUI、IListUI。

　　　　1) CListHeaderUI，父类CHorizontalLayoutUI。

　　　　2) CListHeaderItemUI，列表头类，父类CControlUI。

　　　　3) CListTextElementUI，类表文本类，父类CListLabelElementUI。

　　　　4) CListLabelElementUI，父类CListElementUI。

　　　10. CComboUI，组合框类，父类CContainerUI、IListOwnerUI。

　　　11. CActiveXUI，ActiveX控件类，父类CControlUI、 IMessageFilterUI。

　　　12. CContainerUI，容器类，父类CControlUI、IContainerUI。

　　　13. CTabLayoutUI，选项页布局类，父类CContainerUI。

　　　14. CTileLayoutUI，父类CContainerUI。

　　　15. CDialogLayoutUI，对话框布局类，父类CContainerUI。、

　　　16. CVerticalLayoutUI，垂直布局类，父类CContainerUI。

　　　17. CHorizontalLayoutUI，水平布局类，父类CContainerUI。

　　　18. CListExpandElementUI，父类CListTextElementUI。

　　　19. CListContainerElementUI，父类CContainerUI、IListItemUI。

三、**辅助类**

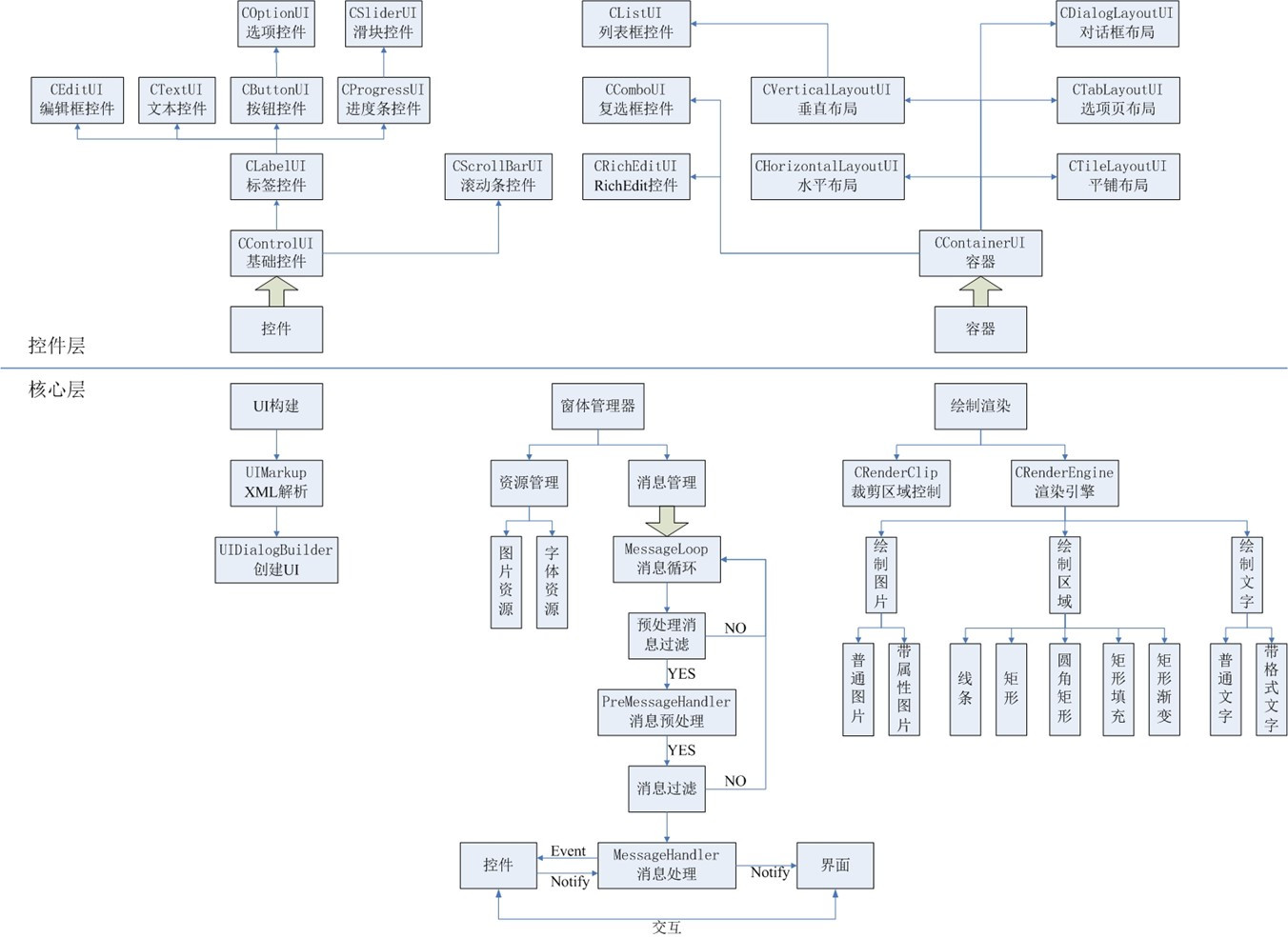
　　　1. CStdPtrArray，指针数组。

　　　2. CStdValArray，数据数组。

　　　3. CStdString，字符串数组。

　　　4. CStdStringPtrMap，字符串指针映射数组。

 DuiLib类图：



怎么在mfc中加duilib：把dulib的头文件和库包含一下，解决一些冲突就可以了。窗口可以是mfc的(绘制用duilib，mfc原有逻辑保持不变)，也可以使用duilib的。 直接自定义个控件，重写DoPaint。

@Combo 添加子项，组合框，代码添加 子项代码

CListLabelElementUI\* elem = new CListLabelElementUI();

elem->SetText(\_T("市本级"));

pCombo->Add(elem);

CListLabelElementUI\* elem1 = new CListLabelElementUI();

elem1->SetText(\_T("天心区"));

pCombo->Add(elem1);

代码添加了2条列表项，也可以做成函数；

@宽字符，Uniode

 Windows底层处理字符串是以Unicode形式处理的。Unicode 中\_T( ) 包住的字符串就成宽字节了， char b[8] 是ANSI的。

@文件写入方法

在ANSI C中，对文件的操作分为两种方式，即流式文件操作和I/O文件操作。

MFC：

CFile file;

CString ctx;

ctx.Format(\_T("TCE\_V11"));

file.Open(\_T("./test.txt"),CFile::modeCreate|CFile::modeWrite);

file.SeekToBegin();

file.Write(ctx,ctx.GetLength()\*2);

file.Flush();

file.Close();

API：

char str[256]; //也可以写char str[256]={"中文字符串\n"};那么就可以省略后面strcpy一句了

FILE \*fp;

if (fp = fopen("data.txt", "w+"))

{

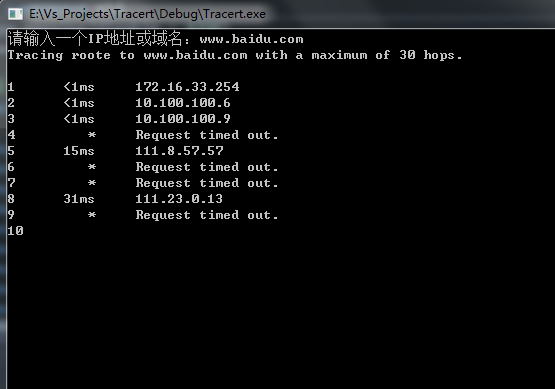
strcpy(str, "中文字符串\n");

fputs(str, fp);

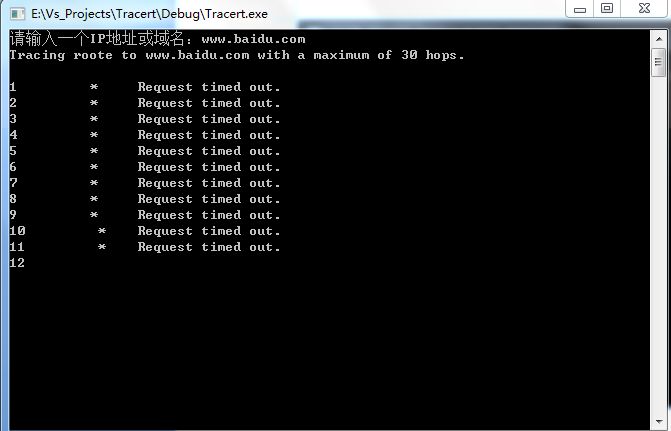
fclose(fp);

}





开启防火墙：



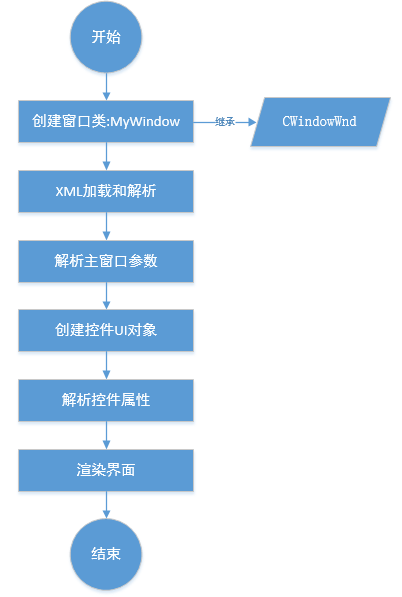
risen[ˈrɪzn] v.上升; rise的过去分词

reason [ˈriːzn] n.原因;

## Duilib源码浅析<一>之UI渲染流程

<https://my.oschina.net/u/3443876/blog/1825586>

# 1. 流程简图



# 2. 自己的窗口类

以360浏览器demo为例, 这个窗口类是C360SafeFrameWnd，它继承了两个类，一个是窗口类CWindowWnd，另一个是消息类INotifyUI。在UI这一块我们主要关注CWindowWnd类。CWindowWnd类：这个类可以叫它窗口类，它封装了窗口的基本属性和接口，便于理解，可以把它类比为MFC中的CWnd类。

# 3. 创建窗口

WindMain中的创建流程如下：

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE /\*hPrevInstance\*/, LPSTR /\*lpCmdLine\*/, int nCmdShow)

{

CPaintManagerUI::SetInstance(hInstance);

CPaintManagerUI::SetResourcePath(CPaintManagerUI::GetInstancePath() + \_T("skin"));

CPaintManagerUI::SetResourceZip(\_T("360SafeRes.zip"));

HRESULT Hr = ::CoInitialize(NULL);

if( FAILED(Hr) ) return 0;

C360SafeFrameWnd\* pFrame = new C360SafeFrameWnd();

if( pFrame == NULL ) return 0;

pFrame->Create(NULL, \_T("360安全卫士"), UI\_WNDSTYLE\_FRAME, 0L, 0, 0, 800, 572);

pFrame->CenterWindow();

::ShowWindow(\*pFrame, SW\_SHOW);

CPaintManagerUI::MessageLoop();

::CoUninitialize();

return 0;

}

这里主要在做三件事：

(1) 设置资源

SetResourcePath函数设置资源路径，并将这个值保存在成员变量"m\_pStrResourcePath"中，SetResourceZip函数是设置资源包，这里是压缩成zip格式，文件名保存在"m\_pStrResourceZip"中。这个zip压缩文件中包含了程序用到的资源，例如图片和xml文件，将360SafeRes.zip解压可以看到所有的资源文件，其中"skin.xml"是主窗口UI配置文件。

(2) 创建窗口

这一步做了很多事情，包括xml文件加载、解析，UI对象创建渲染，窗口显示等。

(3) 消息循环

消息相关的有机会再后续章节再介绍。

# 4. XML解析流程

上一步中说道了创建窗口，我们知道WM\_CREATE是常规意义上窗口收到的第一个消息，在windows程序开发中我们一般在这条消息中做一些初始化的工作。为了处理消息，需要实现虚函数HandleMessage:

LRESULT HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam)

{

LRESULT lRes = 0;

BOOL bHandled = TRUE;

switch( uMsg ) {

case WM\_CREATE: lRes = OnCreate(uMsg, wParam, lParam, bHandled); break;

case WM\_CLOSE: lRes = OnClose(uMsg, wParam, lParam, bHandled); break;

case WM\_DESTROY: lRes = OnDestroy(uMsg, wParam, lParam, bHandled); break;

case WM\_NCACTIVATE: lRes = OnNcActivate(uMsg, wParam, lParam, bHandled); break;

case WM\_NCCALCSIZE: lRes = OnNcCalcSize(uMsg, wParam, lParam, bHandled); break;

case WM\_NCPAINT: lRes = OnNcPaint(uMsg, wParam, lParam, bHandled); break;

case WM\_NCHITTEST: lRes = OnNcHitTest(uMsg, wParam, lParam, bHandled); break;

case WM\_SIZE: lRes = OnSize(uMsg, wParam, lParam, bHandled); break;

case WM\_GETMINMAXINFO: lRes = OnGetMinMaxInfo(uMsg, wParam, lParam, bHandled); break;

case WM\_SYSCOMMAND: lRes = OnSysCommand(uMsg, wParam, lParam, bHandled); break;

default:

bHandled = FALSE;

}

if( bHandled ) return lRes;

if( m\_pm.MessageHandler(uMsg, wParam, lParam, lRes) ) return lRes;

return CWindowWnd::HandleMessage(uMsg, wParam, lParam);

}

这样就可以接收我们想要处理的消息了。再看OnCreate()函数，这里会进行一系列初始化工作：

LRESULT OnCreate(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled)

{

LONG styleValue = ::GetWindowLong(\*this, GWL\_STYLE);

styleValue &= ~WS\_CAPTION;

::SetWindowLong(\*this, GWL\_STYLE, styleValue | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN);

m\_pm.Init(m\_hWnd);

CDialogBuilder builder;

CDialogBuilderCallbackEx cb;

CControlUI\* pRoot = builder.Create(\_T("skin.xml"), (UINT)0, &cb, &m\_pm);

ASSERT(pRoot && "Failed to parse XML");

m\_pm.AttachDialog(pRoot);

m\_pm.AddNotifier(this);

Init();

return 0;

}

这里又出现了一个新的类"CDialogBuilder"，主要功能是什么呢？看它的名字大概就是对话框创建类的意思。首先通过GetWindowLong和SetWindowLong两个函数改变了窗口风格，去掉了"WS\_CAPTION"风格，即去掉了标题栏和边框，然后加上了风格"WS\_CLIPSIBLINGS"和"WS\_CLIPCHILDREN"，简单来说作用是减少绘制。

接下来进入到CDialogBuilder::Create()函数，这里面会加载xml文件，这里又引出一个新类"CMarkup"，这个类完成xml的加载与解析等工作，具体怎么解析的我也就大概看了下，不好发表看法，以后有机会补上。回到CDialogBuilder::Create()，最开始解析出所有的顶层节点属性，包括"Font"、"Default"、"Window"等，这里主要是看"Window"，即主窗口。

(1) 解析主窗口属性

pstrClass = root.GetName();

if( \_tcsicmp(pstrClass, \_T("Window")) == 0 ) {

if( pManager->GetPaintWindow() ) {

int nAttributes = root.GetAttributeCount();

for( int i = 0; i < nAttributes; i++ ) {

pstrName = root.GetAttributeName(i);

pstrValue = root.GetAttributeValue(i);

pManager->SetWindowAttribute(pstrName, pstrValue);

}

}

}

进入函数CPaintManagerUI::SetWindowAttribute()，这里会设置主窗口的所有属性，像size(大小)，sizebox(边框)，caption(标题栏)等。

(2) 创建UI控件对象

解析完主窗口属性后，就开始解析所有的控件，进入函数CDialogBuilder::\_Parse():

SIZE\_T cchLen = \_tcslen(pstrClass);

switch( cchLen ) {

case 4:

if( \_tcsicmp(pstrClass, DUI\_CTR\_EDIT) == 0 ) pControl = new CEditUI;

else if( \_tcsicmp(pstrClass, DUI\_CTR\_LIST) == 0 ) pControl = new CListUI;

else if( \_tcsicmp(pstrClass, DUI\_CTR\_TEXT) == 0 ) pControl = new CTextUI;

else if( \_tcsicmp(pstrClass, DUI\_CTR\_TREE) == 0 ) pControl = new CTreeViewUI;

else if( \_tcsicmp(pstrClass, DUI\_CTR\_HBOX) == 0 ) pControl = new CHorizontalLayoutUI;

else if( \_tcsicmp(pstrClass, DUI\_CTR\_VBOX) == 0 ) pControl = new CVerticalLayoutUI;

break;

case 5:

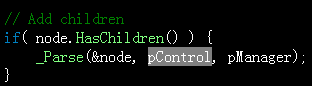
感兴趣的可以自己看下源码，这里只截取片段。功能就是根据节点名创建对应的对象，例如：

https://static.oschina.net/uploads/space/2018/0606/225609_zdDA_3443876.png

xml中配置的节点名是"Text"，c++中对应宏定义为:

https://static.oschina.net/uploads/space/2018/0606/225626_i5Du_3443876.png

字符创长度为4，进入分支"case 4"，然后根据节点名创建对应的对象"CTextUI"。如果有子节点，那么嵌套调用函数\_Parse()自己：



(3) 解析控件属性

创建了UI对象后，接着就是设置UI属性：

每个控件都有自己的属性，且每个控件对属性可能都有自己独特的理解，所以每个控件都重写了虚函数SetAttribute()，例如按钮的SetAttribute()如下：

// Init default attributes

if( pManager ) {

pControl->SetManager(pManager, NULL, false);

LPCTSTR pDefaultAttributes = pManager->GetDefaultAttributeList(pstrClass);

if( pDefaultAttributes ) {

pControl->SetAttributeList(pDefaultAttributes);

}

}

// Process attributes

if( node.HasAttributes() ) {

// Set ordinary attributes

int nAttributes = node.GetAttributeCount();

for( int i = 0; i < nAttributes; i++ ) {

pControl->SetAttribute(node.GetAttributeName(i), node.GetAttributeValue(i));

}

}

到这里就已经完成了xml的解析和UI对象的创建等一系列功能了。

void CButtonUI::SetAttribute(LPCTSTR pstrName, LPCTSTR pstrValue)

{

if( \_tcscmp(pstrName, \_T("normalimage")) == 0 ) SetNormalImage(pstrValue);

else if( \_tcscmp(pstrName, \_T("hotimage")) == 0 ) SetHotImage(pstrValue);

else if( \_tcscmp(pstrName, \_T("pushedimage")) == 0 ) SetPushedImage(pstrValue);

else if( \_tcscmp(pstrName, \_T("focusedimage")) == 0 ) SetFocusedImage(pstrValue);

else if( \_tcscmp(pstrName, \_T("disabledimage")) == 0 ) SetDisabledImage(pstrValue);

else if( \_tcscmp(pstrName, \_T("foreimage")) == 0 ) SetForeImage(pstrValue);

else if( \_tcscmp(pstrName, \_T("hotforeimage")) == 0 ) SetHotForeImage(pstrValue);

else if( \_tcscmp(pstrName, \_T("fivestatusimage")) == 0 ) SetFiveStatusImage(pstrValue);

else if( \_tcscmp(pstrName, \_T("fadedelta")) == 0 ) SetFadeAlphaDelta((BYTE)\_ttoi(pstrValue));

else if( \_tcscmp(pstrName, \_T("hotbkcolor")) == 0 )

{

if( \*pstrValue == \_T('#')) pstrValue = ::CharNext(pstrValue);

LPTSTR pstr = NULL;

DWORD clrColor = \_tcstoul(pstrValue, &pstr, 16);

SetHotBkColor(clrColor);

}

else if( \_tcscmp(pstrName, \_T("hottextcolor")) == 0 )

{

if( \*pstrValue == \_T('#')) pstrValue = ::CharNext(pstrValue);

LPTSTR pstr = NULL;

DWORD clrColor = \_tcstoul(pstrValue, &pstr, 16);

SetHotTextColor(clrColor);

}

else if( \_tcscmp(pstrName, \_T("pushedtextcolor")) == 0 )

{

if( \*pstrValue == \_T('#')) pstrValue = ::CharNext(pstrValue);

LPTSTR pstr = NULL;

DWORD clrColor = \_tcstoul(pstrValue, &pstr, 16);

SetPushedTextColor(clrColor);

}

else if( \_tcscmp(pstrName, \_T("focusedtextcolor")) == 0 )

{

if( \*pstrValue == \_T('#')) pstrValue = ::CharNext(pstrValue);

LPTSTR pstr = NULL;

DWORD clrColor = \_tcstoul(pstrValue, &pstr, 16);

SetFocusedTextColor(clrColor);

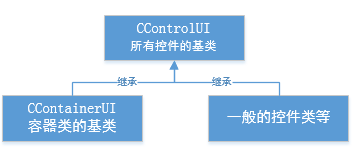
}

else CLabelUI::SetAttribute(pstrName, pstrValue);

}

# 5. 渲染

经过上面几个步骤，完成了xml文件的解析和UI对象的创建，接下来要做的就是把控件绘制到窗口上。Duilib中的控件分两种，一是容器，二是普通控件，且都是从类"CControlUI"继承而来，下面配一张简图说明：



什么是容器呢？从百度百科把定义拷贝过来：容器用来包装或装载物品的贮存器。Duilib里的容器也有类似的功能，就是可以包含其他的控件或容器。最常用的有两个类，"CVerticalLayoutUI"和"CHorizontalLayoutUI"，即纵向布局和横向布局。其实我觉得Duilib里的这种布局的概念还是有点儿蹩脚的，它的好处是简化了xml的编写，只有在布局中才能包含子控件，而一般的控件的属性全部是以元素的形式编写在同一行，这样做的话xml编写会变得简单，弊端就是表现形式不够灵活，例如不能在Button中包含一个子Button或Text等(也有可能是我不会用)，这样的话表达能力就很弱了。如果控件能随便包含子控件，想想是不是很灵活，像按钮上放置图片和文字，这是最常见的功能了吧。当然如果这样做的话，xml解析显然不会像现在这么简单。另外这个布局也显得很不灵活，不能随心所欲，例如我想让一个子控件相对于父控件(或兄弟控件)的九个顶点(left, right, top, bottom, center, topleft, topright, bottomleft, bottomright)中的任意一个布局，抱歉Duilib做不到。当然Duilib的定位是轻量级，在有限的代码内做力所能及的事，我们就不要要求那么多了，每个项目都有自己的优缺点，我们取其所长就好。

回过头来说绘制吧。windows的绘制一般都放在WM\_PAINT消息中进行，Duilib也一样，进入CPaintManagerUI类的消息处理函数MessageHandler，然后看WM\_PAINT分支，在这里会绘制整个窗口的内容。找到下面这一行代码:

https://static.oschina.net/uploads/space/2018/0606/225910_dniJ_3443876.png

m\_pRoot要么是一个"CVerticalLayoutUI"对象，要么就是一个"CHorizontalLayoutUI"对象，为什么呢，因为顶层肯定是一个容器，不然怎么包含子控件。跟进到Paint函数看一下：

bool CControlUI::Paint(HDC hDC, const RECT& rcPaint, CControlUI\* pStopControl)

{

if (pStopControl == this) return false;

if( !::IntersectRect(&m\_rcPaint, &rcPaint, &m\_rcItem) ) return true;

if( OnPaint ) {

if( !OnPaint(this) ) return true;

}

if (!DoPaint(hDC, rcPaint, pStopControl))

return false;

if( m\_pCover != NULL ) return m\_pCover->Paint(hDC, rcPaint);

return true;

}

看到一个函数DoPaint()，有前面知道，此时的this就是上面的两种布局对象中的一个，所以这个DoPaint会调到CContainerUI的DoPaint，截取部分代码：

bool CContainerUI::DoPaint(HDC hDC, const RECT& rcPaint, CControlUI\* pStopControl)

{

RECT rcTemp = { 0 };

if( !::IntersectRect(&rcTemp, &rcPaint, &m\_rcItem) ) return true;

CRenderClip clip;

CRenderClip::GenerateClip(hDC, rcTemp, clip);

CControlUI::DoPaint(hDC, rcPaint, pStopControl);

if( m\_items.GetSize() > 0 ) {

RECT rc = m\_rcItem;

rc.left += m\_rcInset.left;

rc.top += m\_rcInset.top;

rc.right -= m\_rcInset.right;

rc.bottom -= m\_rcInset.bottom;

if( m\_pVerticalScrollBar && m\_pVerticalScrollBar->IsVisible() ) rc.right -= m\_pVerticalScrollBar->GetFixedWidth();

if( m\_pHorizontalScrollBar && m\_pHorizontalScrollBar->IsVisible() ) rc.bottom -= m\_pHorizontalScrollBar->GetFixedHeight();

if( !::IntersectRect(&rcTemp, &rcPaint, &rc) ) {

for( int it = 0; it < m\_items.GetSize(); it++ ) {

CControlUI\* pControl = static\_cast<CControlUI\*>(m\_items[it]);

if( pControl == pStopControl ) return false;

if( !pControl->IsVisible() ) continue;

if( !::IntersectRect(&rcTemp, &rcPaint, &pControl->GetPos()) ) continue;

if( pControl->IsFloat() ) {

if( !::IntersectRect(&rcTemp, &m\_rcItem, &pControl->GetPos()) ) continue;

if( !pControl->Paint(hDC, rcPaint, pStopControl) ) return false;

}

}

}

主要是做两件事。一是绘制自己，二是绘制子控件，如果子控件也是布局，那么重复此过程，这样就可以对所用控件进行绘制。

控件的具体绘制动作在CControlUI::DoPaint()中：

bool CControlUI::DoPaint(HDC hDC, const RECT& rcPaint, CControlUI\* pStopControl)

{

// 绘制循序：背景颜色->背景图->状态图->文本->边框

if( m\_cxyBorderRound.cx > 0 || m\_cxyBorderRound.cy > 0 ) {

CRenderClip roundClip;

CRenderClip::GenerateRoundClip(hDC, m\_rcPaint, m\_rcItem, m\_cxyBorderRound.cx, m\_cxyBorderRound.cy, roundClip);

PaintBkColor(hDC);

PaintBkImage(hDC);

PaintStatusImage(hDC);

PaintText(hDC);

PaintBorder(hDC);

}

else {

PaintBkColor(hDC);

PaintBkImage(hDC);

PaintStatusImage(hDC);

PaintText(hDC);

PaintBorder(hDC);

}

return true;

}

这里是具体的绘制操作，再跟进到PaintBkColor，PaintBkImage这些函数，又会引出一个新类"CRenderEngine"，顾名思义就是渲染引擎。这个类封装了所有的绘图功能，像绘制矩形，绘制椭圆等。到这里整个渲染过程就已经完成了，写的比较简洁，对照着代码看相信还是很容易理清楚的。

最后还有一个小细节双缓冲绘图，在windows编程中经常会提到这个概念。双缓冲是用来解决闪烁问题的一种方式，当然仅仅只是众多方式中的一种，因为屏幕闪烁跟很多因素相关，双缓冲解决的是由屏幕刷新频率造成的闪烁，即绘制动作不是在同一个刷新周期内完成，给人的感觉有可能就是"闪烁"。双缓冲简单来说就是先将屏幕内容绘制到内存DC中，也就是先在内存中把要绘制的内容准备好，然后再一次全部贴到屏幕上，而不是每一个控件各自直接在屏幕上绘制。

多年前做过一个项目，客户端用的是Duilib，后来很长时间都没用过，直到最近才重新看了看代码，顺手就做下记录，以后有机会再把消息路由这一块也整理下，希望对新手有所帮助。这里并没有教你如何使用Duilib，因为相关教程网上有很多，也很详细，对照着做几天就会用的很熟练；如果你想看Duilib源码，那么这儿或许能帮你简单的理理思路。

## Dulib源码浅析<二>之消息路由

<https://my.oschina.net/u/3443876/blog/2459177>

## 1. 消息入口

窗口过程是在注册窗口类中绑定的，我们找到注册窗口类的代码，如下：

bool CWindowWnd::RegisterWindowClass()

{

WNDCLASS wc = { 0 };

wc.style = GetClassStyle();

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

wc.hIcon = NULL;

wc.lpfnWndProc = CWindowWnd::\_\_WndProc;

wc.hInstance = CPaintManagerUI::GetInstance();

wc.hCursor = ::LoadCursor(NULL, IDC\_ARROW);

wc.hbrBackground = NULL;

wc.lpszMenuName = NULL;

wc.lpszClassName = GetWindowClassName();

ATOM ret = ::RegisterClass(&wc);

ASSERT(ret!=NULL || ::GetLastError()==ERROR\\_CLASS\\_ALREADY\_EXISTS);

return ret != NULL || ::GetLastError() == ERROR\\_CLASS\\_ALREADY\_EXISTS;

}

即绑定了窗口过程"CWindowWnd::\_\_WndProc"，跟进这个函数看看，这里截取关键代码：

return pThis->HandleMessage(uMsg, wParam, lParam);

这里的pThis是CWindowWnd的对象，起始指向的是子类"C360SafeFrameWnd"，即我们自己定义的主窗口类，是在函数"CWindowWnd::Create"中在调用"CreateWindowEx"通过最后一个参数传进来的。那么通过这一步，就把消息通过函数"HandleMessage”传进了UI层，这是个虚函数，最终进入到C360SafeFrameWnd::HandleMessage()函数。好了，到这里就已经将windows窗口消息和我们自己的窗口类联系起来了，消息已经进入了我们的窗口类了，现在你可以在函数"HandleMessage"中处理消息响应。

整理下上面的流程：

WinMain()-->

pFrame->Create()-->

CWindowWnd::Create()-->

CWindowWnd::RegisterWindowClass()-->

CWindowWnd::\_\_WndProc()-->

C360SafeFrameWnd::HandleMessage()-->

## 2.控件消息

消息如何进入控件的呢？在xml中配置了一个Buttom，点击这个按钮，按钮是如何捕捉到消息的呢？下面接着分析。

### (1) UI管理器CPaintManagerUI

在主窗口"C360SafeFrameWnd"中定义了一个成员变量"CPaintManagerUI m\_pm"，顾名思义就叫它UI管理器，管理UI消息。

在**消息入口**这一节中，我们看到消息路由到了函数"HandleMessage"，接着看这个函数，可以看到有下面的调用：

\*if( m\_pm.MessageHandler(uMsg, wParam, lParam, lRes) ) return lRes;\*

即将消息传入进了UI管理器"CPaintManagerUI::MessageHandler"，这里管理所有的UI消息，然后将相应的消息传递给对应的控件，这里就解释了开头的疑问，消息是如何传递到UI控件的。

### (2) 消息通知监听

在OnCreate函数中添加事件监听m\_pm.AddNotifier(this);

bool CPaintManagerUI::AddNotifier(INotifyUI\* pNotifier)

{

if (pNotifier == NULL) return false;

ASSERT(m\_aNotifiers.Find(pNotifier)<0);

return m\_aNotifiers.Add(pNotifier);

}

分发消息通知\*CPaintManagerUI::SendNotify\*，向添加了监听的窗口发送消息通知。

### (3) 通知处理

重写虚函数\*Notify\*，在这里接收消息事件，处理自己的消息事件。

以单击按钮为例，看看是如何流程：

CPaintManagerUI::MessageHandler-->

pClick->Event--> 分发事件

CControlUI::Event-->

CButtonUI::DoEvent--> 响应事件

CButtonUI::Activate-->

CPaintManagerUI::SendNotify-->

CPaintManagerUI::SendNotify--> 分发消息通知

C360SafeFrameWnd::Notify--> 响应消息通知

## 基础教程推荐

最后推荐个系列基础入门教程，以前看过，新手再适合不过了，[<http://www.cnblogs.com/Alberl/>]