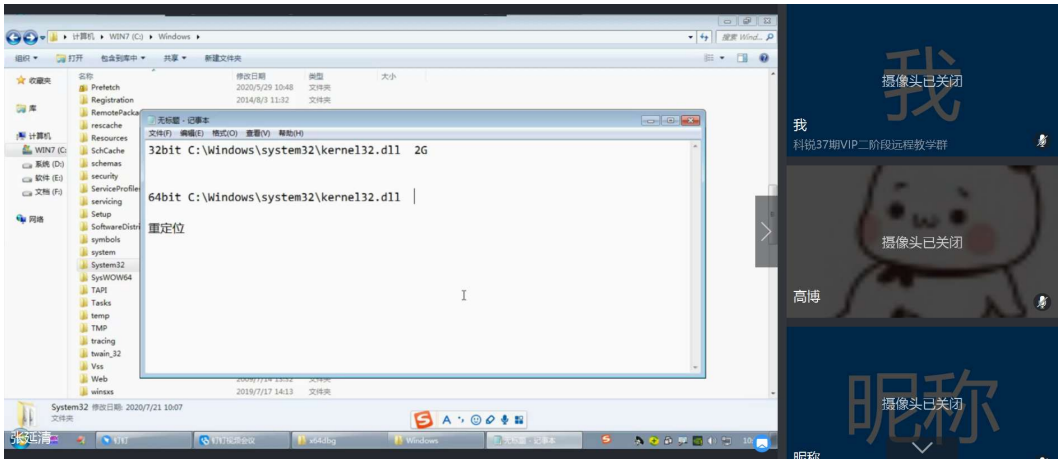
2020/07/21_Windows编程_第3课_DLL中做UI以及.ini文件的使用

笔记本: Windows编程
创建时间: 2020/7/21 星期二 10:16
作者: ileemi
标签: .ini文件, Config文件的使用, 切换模块状态, 在DLL中使用MFC

作业讲解



.def -- EXPORTS

MFC 做库（库里含有框架）

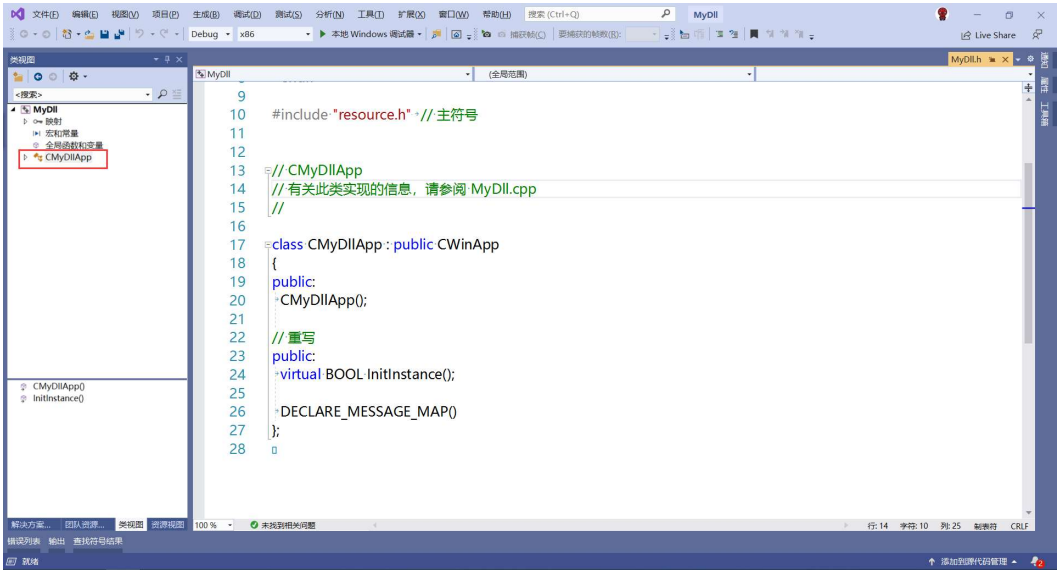
新建 **MFC 动态链接库**

DLL类型：DLL中使用MFC的代码

- 使用共享 MFC DLL 的常规DLL -- MFC中的库以动态方式链接进来（程序运行的时候会加载MFC的DLL）
- 静态链接到MFC的常规DLL -- MFC中的库以静态方式链接进来
- MFC扩展DLL -- 将MFC的类导出来使用

使用共享 MFC DLL 的常规DLL

项目初始化成功后，通过观察类试图可以发现项目含有一个默认的类 "...App"，这个类是用来做初始化的。使用这个程序生成的dll，要求使用的机器要有 MFC 的 dll（没有的话程序无法运行），



DLL中做UI

CImage -- 支持的图片格式较多

动态库中添加对话框并使用

动态库中添加对话框

1. 创建一个普通的动态链接库

2. 实现一个导出函数（弹出对话框）
3. 添加对话框等资源
4. 添加一个.def文件（添加导出函数），这里将 **ShowDlg** 函数进行导出

代码示例：

```
#include "pch.h"
#include "resource.h"

HINSTANCE g_hInstance;
BOOL APIENTRY DllMain( HMODULE hModule,
    DWORD ul_reason_for_call,
    LPVOID lpReserved
)
{
    // 获取对话框句柄，通过模块句柄才能够找到资源
    g_hInstance = (HINSTANCE)hModule;

    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
                break;
    }
    return TRUE;
}

// 对话框的回调函数
INT_PTR CALLBACK DialogProc(HWND hwndDlg,
    UINT uMsg,
    WPARAM wParam,
    LPARAM lParam)
{
    if (uMsg == WM_CLOSE)
    {
        EndDialog(hwndDlg, 0);
    }
    return FALSE;
}

// 导出函数，弹出对话框资源
void ShowDlg()
{
    /*
    老本版使用DialogBox
```

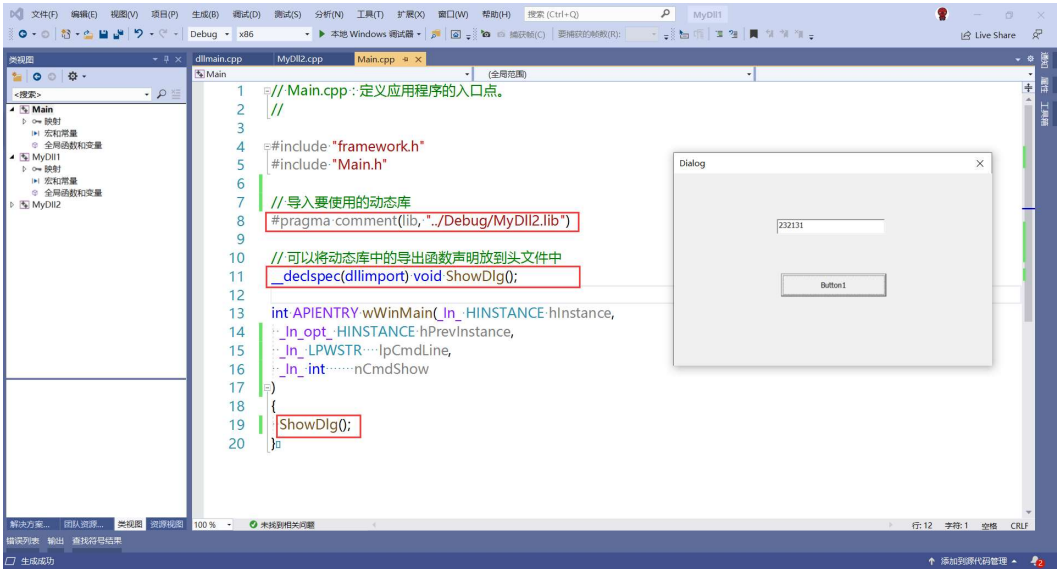
```
    新版本可以使用 DialogBoxParam
    */
    DialogBoxParam(g_hInstance,
        MAKEINTRESOURCE(IDD_DIALOG1),
        NULL,
        DialogProc,
        NULL);
}
```

使用普通的动态库

1. 新建一个 Windows 桌面程序
2. 静态或者动态使用动态库（这里选则静态使用）
3. 将要使用的导出函数使用关键字声明为导出（**__declspec(dllimport)**）。
4. 在程序的 wWinMain 中直接知用动态库中的导出函数
5. 运行程序即可

代码示例：

```
#include "framework.h"
#include "Main.h"

// 导入要使用的动态库
#pragma comment(lib, "../Debug/MyDll1.lib")

// 可以将动态库中的导出函数声明放到头文件中
__declspec(dllimport) void ShowDlg();

int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
    _In_opt_ HINSTANCE hPrevInstance,
    _In_ LPWSTR lpCmdLine,
    _In_ int nCmdShow
)
{
    ShowDlg();
}
```

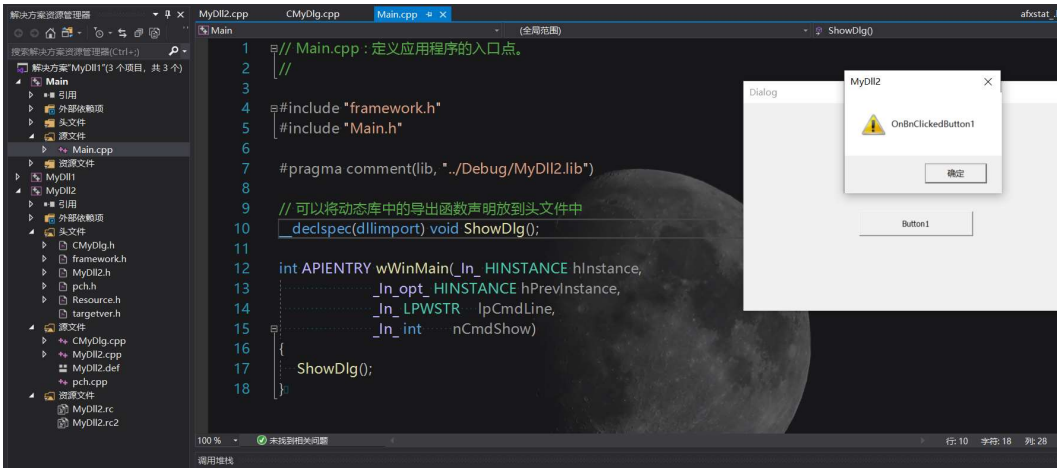


存在问题

无法响应对话框中的消息，动态库响应消息比较麻烦，需要使用SDK，可以使用MFC的框架进行消息的响应，可以新建一个MFC的动态库程序。

MFC动态库的创建

1. 新建一个MFC动态库 (使用共享 MFC DLL 的常规DLL)
2. 添加对话框，给对话框添加一个类（双击对话框）
3. 实现一个导出函数（弹出对话框），并在对应的.def文件中声明
4. 在对话框中添加控件并使用



模块状态

编写 **动态链接库** 时，项目属性 -- 高级 -- MFC的使用 -- 在 "**共享 DLL 中使用**" 的情况下需要切换 **模块状态**。

"**在静态库中使用 MFC**" 的情况下不需要切换模块状态：使用静态连接时，MFC库中的所有资源已经链接到我们的库中，HINSTANCE 不在需要获取。

使用 `AFX_MANAGE_STATE(AfxGetStaticModuleState());` 来切换模块状态，在DLL中使用MFC就需要注意 "切换模块状态"。

代码示例：

```
void ShowDlg()
{
    /*
    在共享DLL中使用MFC
    动态链接库的情况下需要切换模块状态
    模块状态
    */
    AfxGetStaticModuleState(); // 获取模块状态

    //切换模块状态
    AFX_MANAGE_STATE(AfxGetStaticModuleState());

    /*
    不切换模块状态，使用API获取HINSTANCE，
    对话框可以弹出，但是不能响应消息，这样写没有意义
    */
    //DialogBoxParam(AfxGetStaticModuleState()->m_hCurrentInstanceHandle,
    //MAKEINTRESOURCE(IDD_DIALOG1), NULL, NULL, NULL);

    // 不在需要使用API 来弹出对话框，直接创建对话框类对象
    // 将 dll 编译成静态链接库的时候，直接使用下面的两行代码即可
    CMyDlg dlg;
```

```
    dlg.DoModal();
}
```

```
279  AFX_MODULE_STATE* AFXAPI AfxGetAppModuleState();
280  #ifdef _AFXDLL
281  AFX_MODULE_STATE* AFXAPI AfxSetModuleState(AFX_MODULE_STATE* pNewState) throw();
282  #endif
283  AFX_MODULE_STATE* AFXAPI AfxGetModuleState();
284  BOOL AFXAPI AfxIsModuleDll();
285  BOOL AFXAPI AfxInitCurrentStateApp();
286  AFX_MODULE_STATE* AFXAPI AfxGetStaticModuleState();
287  HINSTANCE AFXAPI AfxGetInstanceHandleHelper();
288
```

AFX_MODULE_STATE *__stdcall AfxGetStaticModuleState()
联机搜索

```
198  // AFX_MODULE_STATE (global data for a module)
199  class AFX_MODULE_STATE : public CNoTrackObject
200  {
201  public:
202  #ifdef _AFXDLL
203      AFX_MODULE_STATE(BOOL bDLL, WNDPROC pfnAfxWndProc, DWORD dwVersion,
204          BOOL bSystem = FALSE);
205  #else
206      explicit AFX_MODULE_STATE(BOOL bDLL);
207  #endif
208      ~AFX_MODULE_STATE();
209
210      CWinApp* m_pCurrentWinApp;
211      HINSTANCE m_hCurrentInstanceHandle;
212      HINSTANCE m_hCurrentResourceHandle;
213      LPCTSTR m_lpszCurrentAppName;
214      BYTE m_bDLL;    // TRUE if module is a DLL, FALSE if it is an EXE
215      BYTE m_bSystem; // TRUE if module is a "system" module, FALSE if not
216      BYTE m_bReserved[2]; // padding
217
218      DWORD m_fRegisteredClasses; // flags for registered window classes
```

AFX_MANAGE_STATE

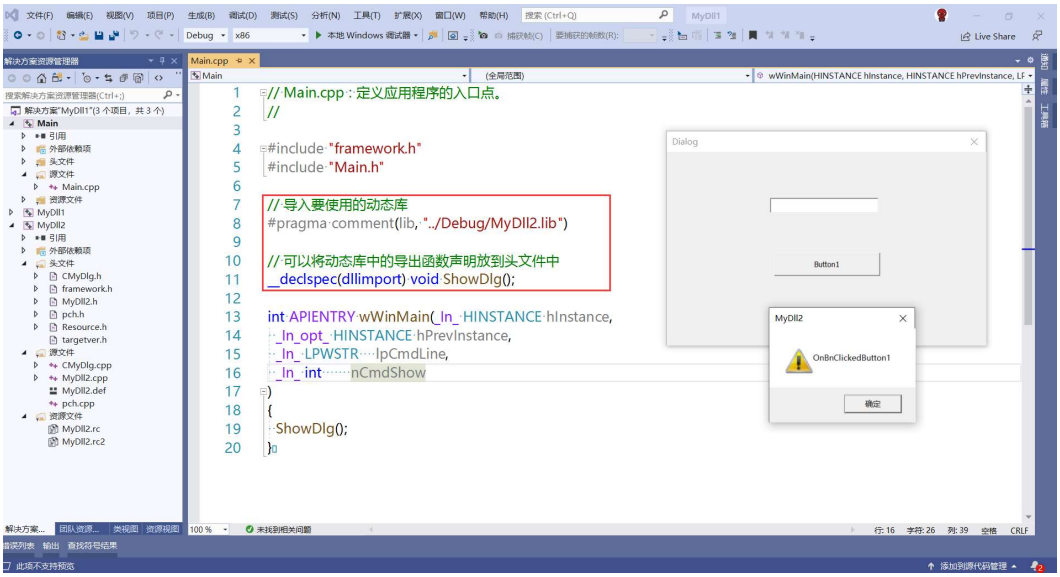
```
323  #define AFX_MANAGE_STATE_NO_INIT_MANAGED(p) AFX_MAINTAIN_STATE2 _ctlState(p);
324  #define AFX_MANAGE_STATE(p) _AfxInitManaged(); AFX_MANAGE_STATE_NO_INIT_MANAGED(p)
```

使用

1. 新建一个 Windows 桌面程序
2. 静态或者动态使用动态库（在共享 DLL 中使用 MFC）
3. 将要使用的导出函数使用关键字声明为导出（**__declspec(dllimport)**）。
4. 在程序的 wWinMain 中直接知用动态库中的导出函数
5. 运行程序即可
   代码示例：

```
#include "framework.h"
#include "Main.h"

// 导入要使用的动态库
#pragma comment(lib, "../Debug/MyDll1.lib")

// 可以将动态库中的导出函数声明放到头文件中
__declspec(dllimport) void ShowDlg();

int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
    _In_opt_ HINSTANCE hPrevInstance,
```

```
    _In_ LPWSTR lpCmdLine,
    _In_ int nCmdShow
)
{
    ShowDlg();
}
```



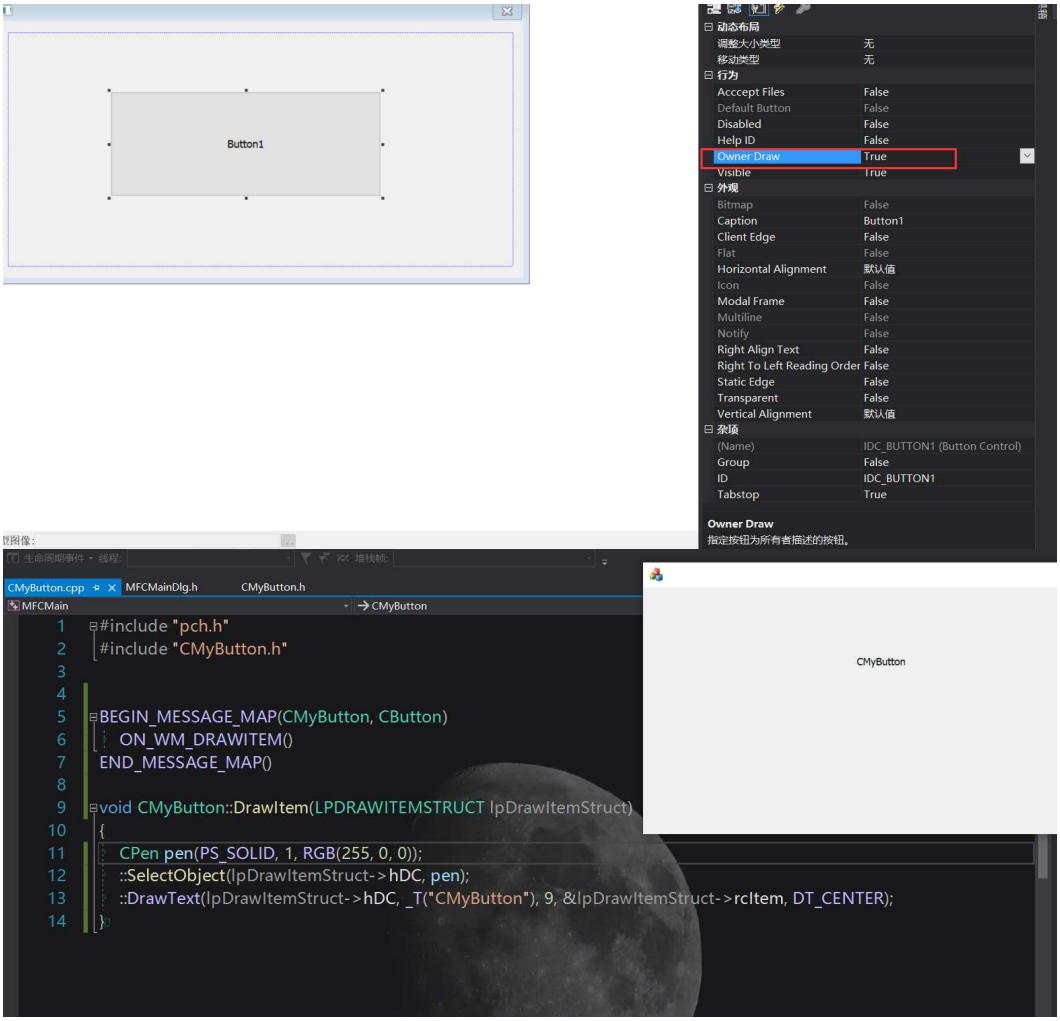
常规DLL

上面两种方式生成的DLL为常规DLL，编译的 Dll 所有程序都可以使用。还有一个中为 MFC DLL，生成的 DLL 只有 MFC 程序可以使用。

MFC DLL

新建MFC动态链接库，DLL类型为 "**MFC 扩展 DLL**"

按钮更改 Owner Draw 为 true 后，需要重写虚函数 **DrawItem**，在该函数内部可以修改按钮的相关样式。



DrawItem参数的定义：

DRAWITEMSTRUCT
说明：DRAWITEMSTRUCT 结构体为所有者窗口提供了确定如何绘制所有者绘制的控件或菜单项的信息。

```
/*
* DRAWITEMSTRUCT for ownerdraw
*/
typedef struct tagDRAWITEMSTRUCT {
    UINT        CtlType;
    UINT        CtlID;
    UINT        itemID;
    UINT        itemAction;
    UINT        itemState;
    HWND        hwndItem;
    HDC         hDC;
    RECT        rcItem;
    ULONG_PTR   itemData;
} DRAWITEMSTRUCT, NEAR *PDRAWITEMSTRUCT, FAR *LPDRAWITEMSTRUCT;
```

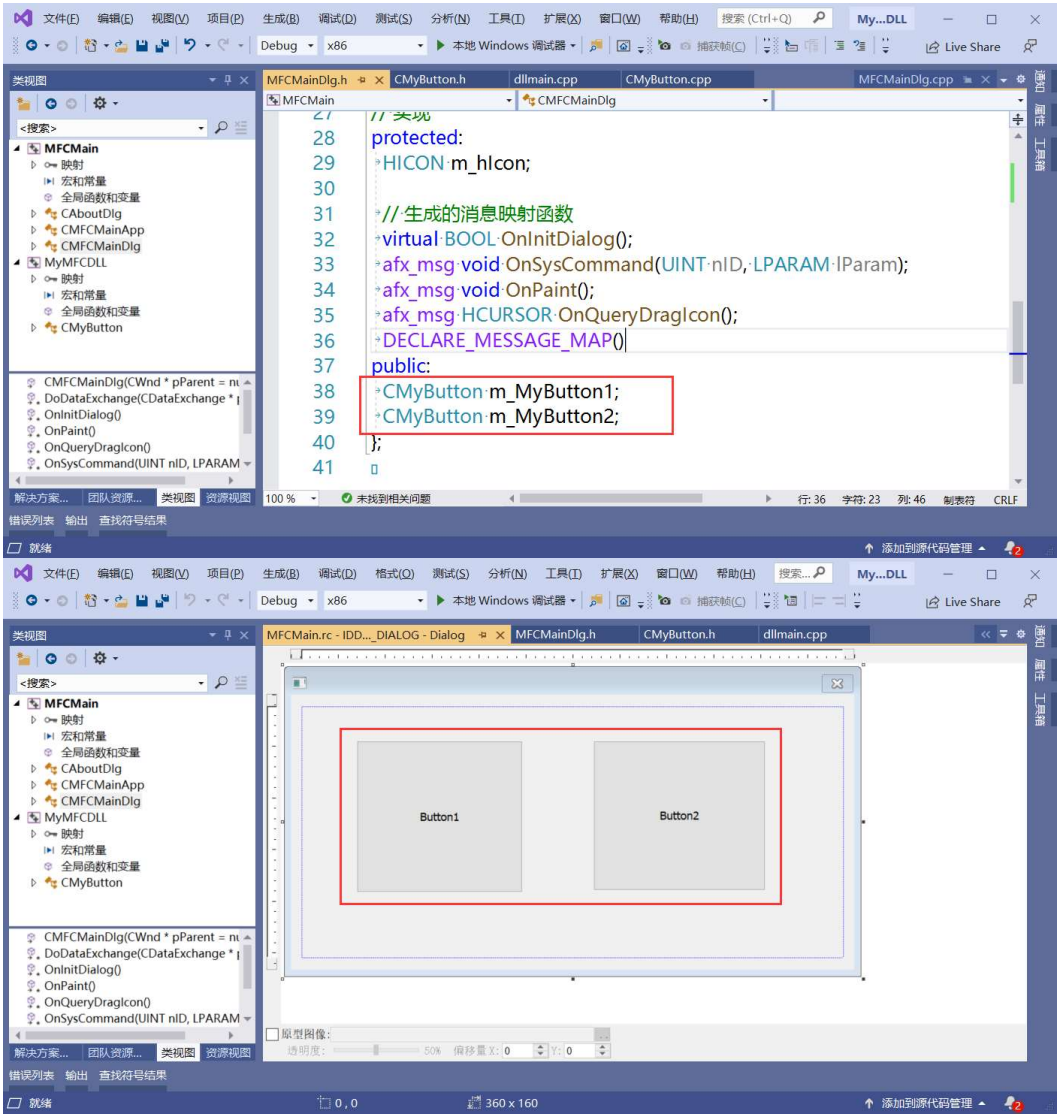
绘制3D按钮

API -- DrawItem

说明：调用此方法来绘制所有者绘制的CButton对象

步骤：

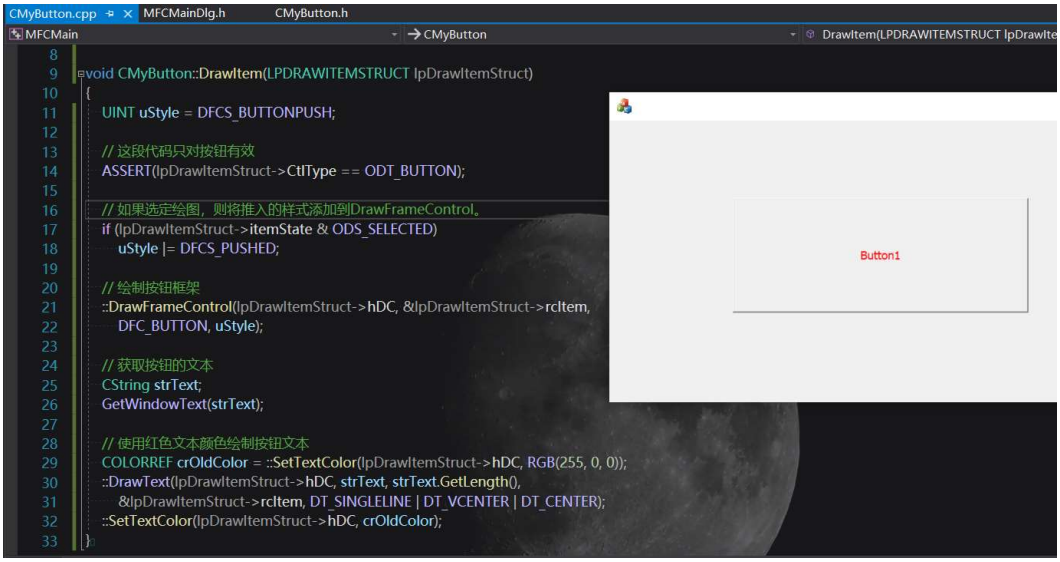
1. 在MFC程序中添加一个自定义的 MFC类 -- CMyButton 继承 CButton
2. 在 CMyButton 类中重写 DrawItem 函数
3. 在原来 MFC 类的对话框上添加按钮
4. 为添加的按钮添加成员，类型为 自定义的CMyButton
5. 在重写的 DrawItem 函数内进行代码编写



代码示例：

```
void CMyButton::DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct)
{
    UINT uStyle = DFCS_BUTTONPUSH;

    // 这段代码只对按钮有效
    ASSERT(lpDrawItemStruct->CtlType == ODT_BUTTON);

    // 如果选定绘图，则将推入的样式添加到DrawFrameControl。
    if (lpDrawItemStruct->itemState & ODS_SELECTED)
        uStyle |= DFCS_PUSHED;

    // 绘制按钮框架
    ::DrawFrameControl(lpDrawItemStruct->hDC,
        &lpDrawItemStruct->rcItem,
```

```
        DFC_BUTTON, uStyle);

    // 获取按钮的文本
    CString strText;
    GetWindowText(strText);

    // 使用红色文本颜色绘制按钮文本
    COLORREF crOldColor = ::SetTextColor(
        lpDrawItemStruct->hDC,
        RGB(255, 0, 0));
    ::DrawText(lpDrawItemStruct->hDC,
        strText,
        strText.GetLength(),
        &lpDrawItemStruct->rcItem,
        DT_SINGLELINE | DT_VCENTER | DT_CENTER);
    ::SetTextColor(lpDrawItemStruct->hDC, crOldColor);
}
```



绘制带图片的按钮

1. 在MFC程序中添加一个自定义的 MFC类 -- CMyButton 继承 CButton
2. 在 CMyButton 类中重写 DrawItem 函数
3. 在原来 MFC 类的对话框上添加按钮
4. 为添加的按钮添加成员，类型为 自定义的 CMyButton
5. 在重写的 DrawItem 函数内进行代码编写
6. 添加图片（.bmp）到资源中
7. 在 CMyButton 类的默认构造中加载位图资源
8. 在 DrawItem 中根据点击和不点击绑定位图即可

代码示例：

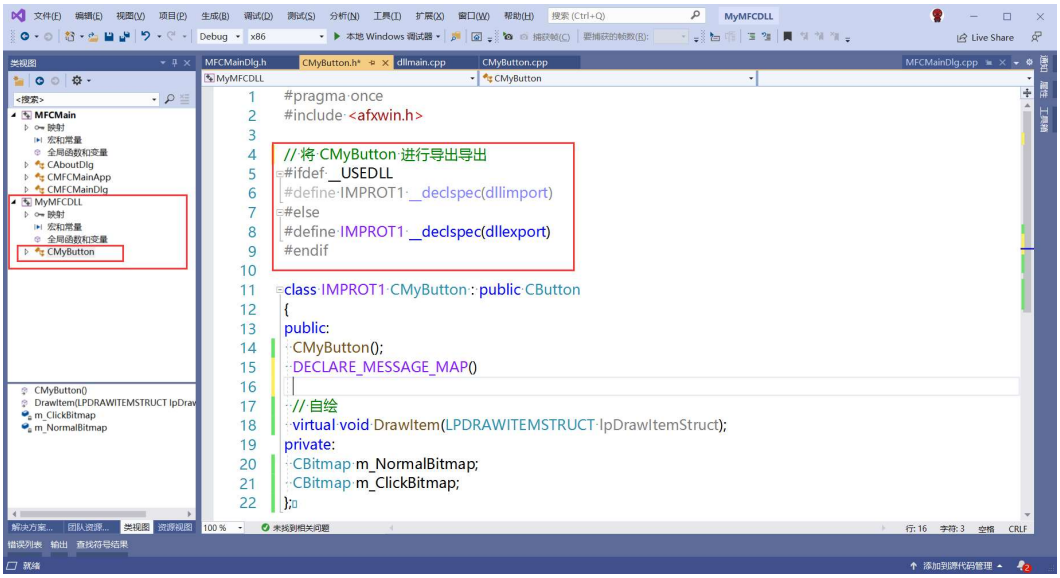
```
void CMyButton::DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct)
{
```

```
    UINT uStyle = DFCS_BUTTONPUSH;

    // 使用主程序的资源
    //AfxGetAppModuleState()->m_hCurrentInstanceHandle;
    // 使用当前程序的资源
    //AfxGetStaticModuleState()->m_hCurrentInstanceHandle;

    // 这段代码只对按钮有效
    ASSERT(lpDrawItemStruct->CtlType == ODT_BUTTON);

    // 绑定DC
    CDC dc;
    dc.Attach(lpDrawItemStruct->hDC);

    // 不能直接选中图片，就创建一个内存DC
    CDC memDC;
    // 创建一个兼容的DC
    memDC.CreateCompatibleDC(&dc);

    // 获取按钮的长和宽
    int nButtonWidth = lpDrawItemStruct->rcItem.right -
        lpDrawItemStruct->rcItem.left;
    int nButtonHeight = lpDrawItemStruct->rcItem.bottom -
        lpDrawItemStruct->rcItem.top;

    // 如果选定绘图，则将推入的样式添加到DrawFrameControl。
    if (lpDrawItemStruct->itemState & ODS_SELECTED)
    {
        // 根据点击按钮的情况绘制对应的图片
        memDC.SelectObject(m_ClickBitmap);
    }
    else
    {
        memDC.SelectObject(m_NormalBitmap);
    }
    // 拉伸拷贝位图
    dc.StretchBlt(0, 0,
        nButtonWidth,
        nButtonHeight,
        &memDC,
        0, 0,
        500, 500,
        SRCCOPY);
}
```

使用 MFC dll

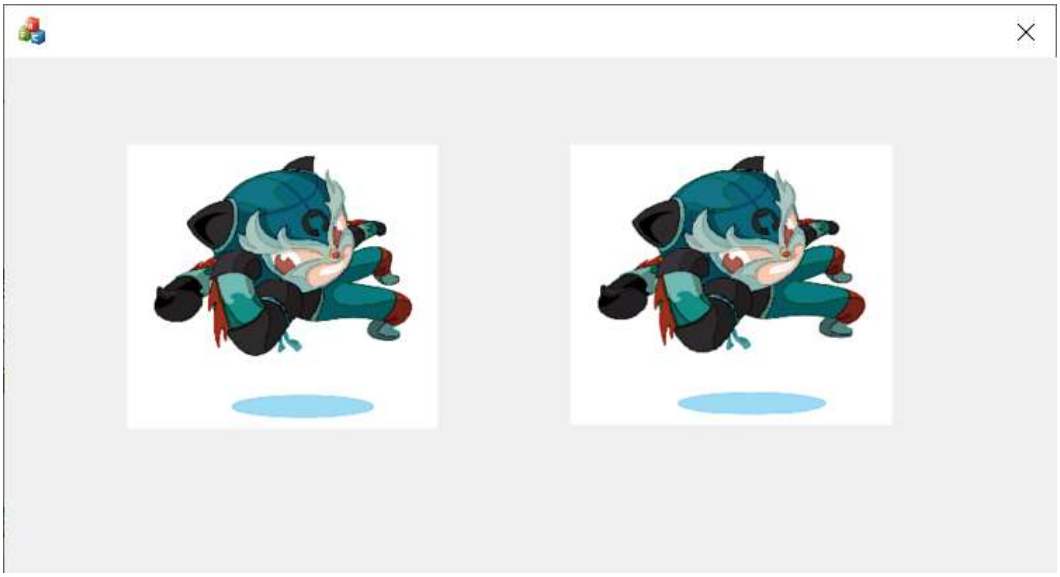
将上一步创建的 CMyButton 类进行封装（也就是将其添加到 "MFC 扩展 DLL" 的程序中），图标资源也要重新进行加载。

之后将该类进行导出，供 MFC 项目使用该 Dll 中的导出类。
代码如下：




```
void CMyButton::DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct)
{
    UINT uStyle = DFCS_BUTTONPUSH;

    // 使用主程序的资源
    //AfxGetAppModuleState()->m_hCurrentInstanceHandle;
    // 使用当前程序的资源
    //AfxGetStaticModuleState()->m_hCurrentInstanceHandle;

    // 这段代码只对按钮有效
    ASSERT(lpDrawItemStruct->CtlType == ODT_BUTTON);

    // 绑定DC
    CDC dc;
    dc.Attach(lpDrawItemStruct->hDC);

    // 不能直接选中图片，就创建一个内存DC
    CDC memDC;
    // 创建一个兼容的DC
    memDC.CreateCompatibleDC(&dc);

    // 获取按钮的长和宽
    int nButtonWidth = lpDrawItemStruct->rcItem.right -
        lpDrawItemStruct->rcItem.left;
    int nButtonHeight = lpDrawItemStruct->rcItem.bottom -
        lpDrawItemStruct->rcItem.top;

    // 如果选定绘图，则将推入的样式添加到DrawFrameControl。
```

```
    if (lpDrawItemStruct->itemState & ODS_SELECTED)
    {
        // 根据点击按钮的情况绘制对应的图片
        memDC.SelectObject(m_ClickBitmap);
    }
    else
    {
        memDC.SelectObject(m_NormalBitmap);
    }
    // 拉伸拷贝位图
    dc.StretchBlt(0, 0,
        nButtonWidth,
        nButtonHeight,
        &memDC,
        0, 0,
        500, 500,
        SRCCOPY);
}
```



注意：

- 在 dll 的编译前需要在 项目属性 -- 配置属性 -- 高级 -- MFC 的使用 改为 "**在共享 DLL 中使用 MFC**"，在使用该 dll 的MFC工程中也做同样的配置修改。

使用主程序的资源：
AfxGetAppModuleState()->m_hCurrentInstanceHandle;
使用当前程序的资源：
AfxGetStaticModuleState()->m_hCurrentInstanceHandle;

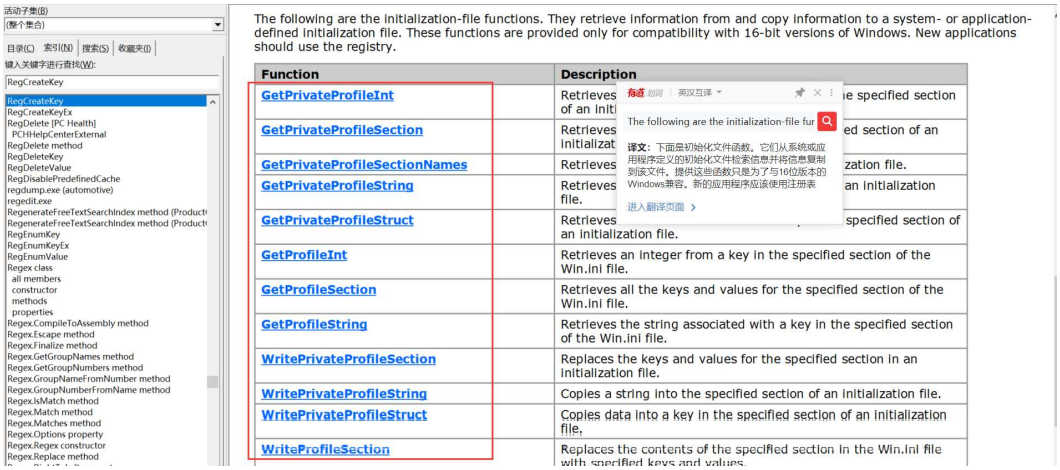
扩展 dll 的资源也是主程序的资源。

config文件的使用

.ini文件 -- API 注册表使用的API

相关API：

GetPrivateProfileInt



The following are the initialization-file functions. They retrieve information from and copy information to a system- or application-defined initialization file. These functions are provided only for compatibility with 16-bit versions of Windows. New applications should use the registry.

Function	Description
GetPrivateProfileInt	Retrieves
GetPrivateProfileSection	Retrieves
GetPrivateProfileSectionNames	Retrieves
GetPrivateProfileString	Retrieves
GetPrivateProfileStruct	Retrieves
GetProfileInt	Retrieves an integer from a key in the specified section of the Win.ini file.
GetProfileSection	Retrieves all the keys and values for the specified section of the Win.ini file.
GetProfileString	Retrieves the string associated with a key in the specified section of the Win.ini file.
WritePrivateProfileSection	Replaces the keys and values for the specified section in an initialization file.
WritePrivateProfileString	Copies a string into the specified section of an initialization file.
WritePrivateProfileStruct	Copies data into a key in the specified section of an initialization file.
WriteProfileSection	Replaces the contents of the specified section in the Win.ini file with specified keys and values.

配置文件的两种保存方式：

1. 程序的目录下
2. 系统目录

保存数据前创建节：

API -- **WritePrivateProfileSection**

节的概念：[Pen] [Brush] 分别代表一个节，放置同属性不同样式的值冲突。

[Pen] -- width

[Brush] -- width

将配置信息写入到配置文件中

代码示例：

```
void CConfigTestDlg::OnBnClickedSave()
{
    // 更新值到控件的变量中
    UpdateData(TRUE);

    // 保存画笔的设置

    // 创建节，没有创建会自动创建
    BOOL ret;
    ret = WritePrivateProfileSection(_T("Pen"), _T(""),
_T("config.ini"));
    ret = WritePrivateProfileSection(_T("Brush"), _T(""),
_T("config.ini"));

    // 将一个字符串复制到初始化文件的指定部分
     CString csPath = _T("D:\\CR37\\Works\\第二阶段\\Windows编程
\\Codes\\"
```

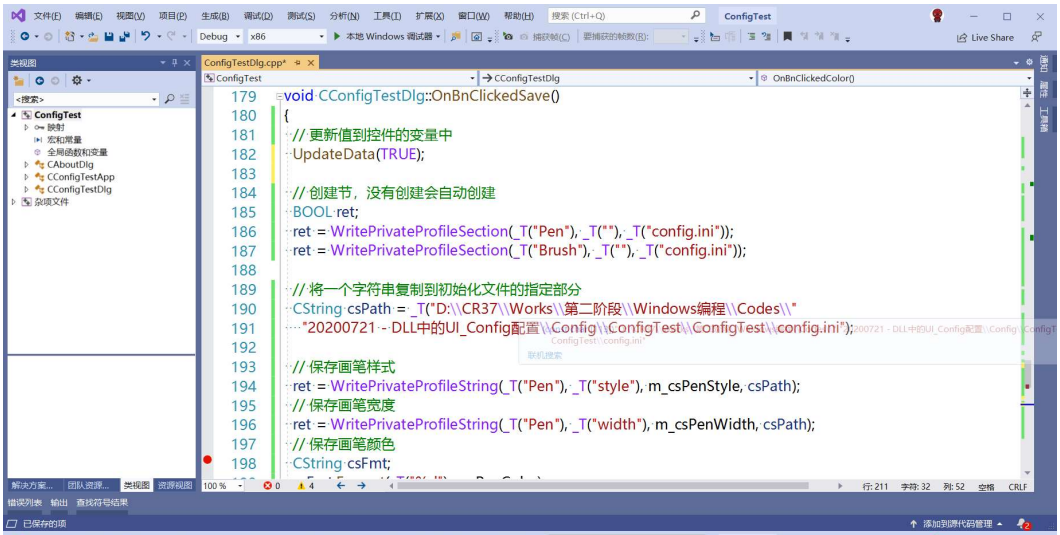
```
        "20200721 - DLL中的UI_Config配置"
        "\\Config\\ConfigTest\\ConfigTest\\config.ini");

    // 保存画笔样式
    ret = WritePrivateProfileString(_T("Pen"), _T("style"),
m_csPenStyle, csPath);
    // 保存画笔宽度
    ret = WritePrivateProfileString(_T("Pen"), _T("width"),
m_csPenWidth, csPath);
    // 保存画笔颜色
    CString csFmt;
    csFmt.Format(_T("%d"), m_PenColor);
    ret = WritePrivateProfileString(_T("Pen"), _T("color"), csFmt,
csPath);

    // 保存结构体 存储文件为十六进制
    //int nValue;
    //WritePrivateProfileStruct(_T("Pen"), _T("Test"), &nValue,
sizeof(nValue), csPath);
}
```

从配置文件中读取数据控件中

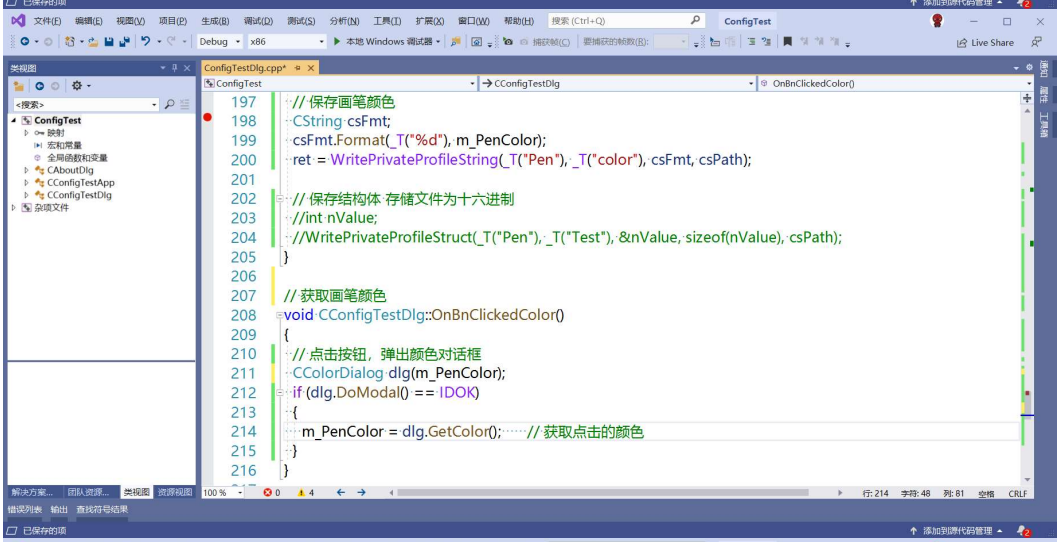
代码示例:

```
// 读取.ini文件
void CConfigTestDlg::OnBnClickedRead()
{
    // 获取配置文件的路径
    CString csPath = _T("D:\\CR37\\Works\\第二阶段\\Windows编程
\\Codes\\"
        "20200721 - DLL中的UI_Config配置"
        "\\Config\\ConfigTest\\ConfigTest\\config.ini");

    // 读取画笔样式
    GetPrivateProfileString(_T("pen"), _T("style"), _T("实心"),
        m_csPenStyle.GetBufferSetLength(255), 255, csPath);
    // 读取画笔宽度
    GetPrivateProfileString(_T("pen"), _T("width"), _T("1"),
        m_csPenWidth.GetBufferSetLength(255), 255, csPath);
    // 读取画笔颜色
    m_PenColor = GetPrivateProfileInt(_T("pen"), _T("color"), 0,
csPath);
```

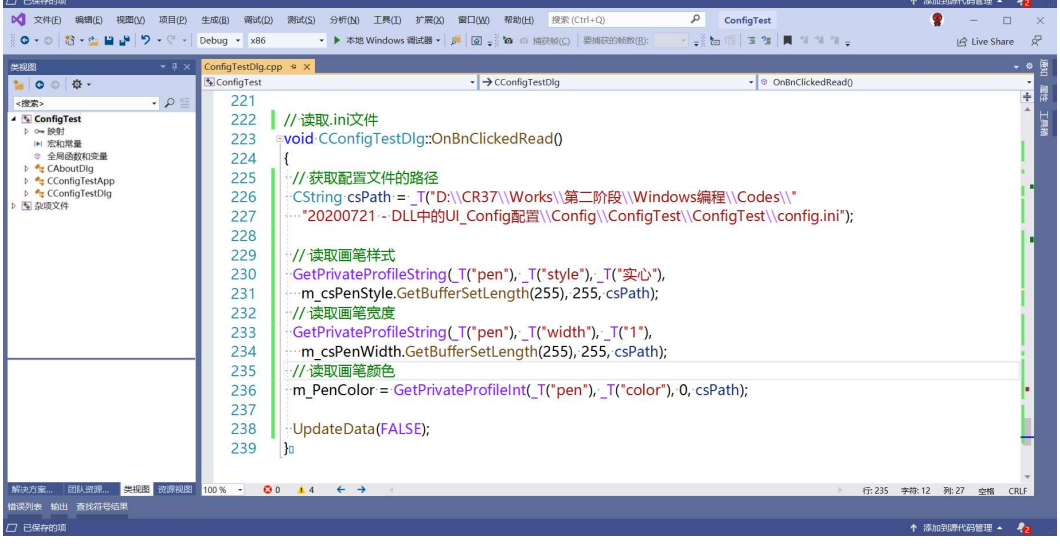
```
        UpdateData(FALSE);
    }
```



```
void CConfigTestDlg::OnBnClickedSave()
{
    // 更新值到控件的变量中
    UpdateData(TRUE);

    // 创建节，没有创建会自动创建
    BOOL ret;
    ret = WritePrivateProfileSection(_T("Pen"), _T(""), _T("config.ini"));
    ret = WritePrivateProfileSection(_T("Brush"), _T(""), _T("config.ini"));

    // 将一个字符串复制到初始化文件的指定部分
    CString csPath = _T("D:\\CR37\\Works\\第二阶段\\Windows编程\\Codes\\"
        "20200721 - DLL中的UI_Config配置\\Config\\ConfigTest\\ConfigTest\\config.ini");

    // 保存画笔样式
    ret = WritePrivateProfileString(_T("Pen"), _T("style"), m_csPenStyle, csPath);
    // 保存画笔宽度
    ret = WritePrivateProfileString(_T("Pen"), _T("width"), m_csPenWidth, csPath);
    // 保存画笔颜色
    CString csFmt;
```



```
    // 保存画笔颜色
    CString csFmt;
    csFmt.Format(_T("%d"), m_PenColor);
    ret = WritePrivateProfileString(_T("Pen"), _T("color"), csFmt, csPath);

    // 保存结构体 存储文件为十六进制
    //int nValue;
    //WritePrivateProfileStruct(_T("Pen"), _T("Test"), &nValue, sizeof(nValue), csPath);
}

// 获取画笔颜色
void CConfigTestDlg::OnBnClickedColor()
{
    // 点击按钮，弹出颜色对话框
    CColorDialog dlg(m_PenColor);
    if (dlg.DoModal() == IDOK)
    {
        m_PenColor = dlg.GetColor();    // 获取点击的颜色
    }
}
```



```
// 读取.ini文件
void CConfigTestDlg::OnBnClickedRead()
{
    // 获取配置文件的路径
    CString csPath = _T("D:\\CR37\\Works\\第二阶段\\Windows编程\\Codes\\"
        "20200721 - DLL中的UI_Config配置\\Config\\ConfigTest\\ConfigTest\\config.ini");

    // 读取画笔样式
    GetPrivateProfileString(_T("pen"), _T("style"), _T("实心"),
        m_csPenStyle.GetBufferSetLength(255), 255, csPath);
    // 读取画笔宽度
    GetPrivateProfileString(_T("pen"), _T("width"), _T("1"),
        m_csPenWidth.GetBufferSetLength(255), 255, csPath);
    // 读取画笔颜色
    m_PenColor = GetPrivateProfileInt(_T("pen"), _T("color"), 0, csPath);

    UpdateData(FALSE);
}
```

.ini 文件的缺点：

- 保存的类型支持太少（支持 字符串 整型 结构体）

.xml 文件优点：

- 支持复杂格式数据的存储
- 支持自定义类型