2020/07/01_MFC_第3课_SDK子类化以及MFC子类化

笔记本: MFC
创建时间: 2020/7/1 星期三 15:34
作者: ileemi
标签: DDX, MFC_子类化

SDK 子类化

> 在 SDK 编程中，Windows 提供的控件会有不满足我们的需求的时候，例如，文本编辑框输入的数据我们只想用户输入数字和字母（不能输入特殊符号），该怎样操作?
> SDK中的Edit Control 默认提供一种只允许输入数字的情况，但是依然与我们的需求差点。

为此，需要给文本编辑框扩展新的功能，使其文本编辑框内仅支持大小写字母和数字。

使用 SetWindowLong 以及 GetWindowLong

文本编辑框功能的实现来自于**过程函数**，可以尝试修改过程函数，但是考虑到文本编辑框是Windows已经封装好的，考虑一些别的方法，但是总之还是需要修改过程函数。

为此，Windows 提供了两个API函数。

- **SetWindowLong** -- 修改窗口的一些风格（风格、过程函数、实例句柄、消息ID等），改变指定窗口的属性。
- **GetWindowLong** -- 获取窗口的一些风格（风格、过程函数、实例句柄、消息ID等），检索有关指定窗口的信息。

> **实现思路**：使用 Windows 提供的 SetWindowLong 替换原来 Windows 封装好的窗口过程函数，之后在自己的窗口过程函数内实现只输入数字和字母的这个功能。-- **这种做法就叫做子类化。**

GetDlgItem -- 获取指定对话框中控件的句柄
WM_KEYDOWN 的 **wParam** 参数 -- 获取键盘虚拟码
WM_CHAR -- 当一个 **WM_KEYDOWN** 消息被 **TranslateMessage** 函数翻译时，**WM_CHAR** 消息被发布到带有键盘焦点的窗口。**WM_CHAR** 消息包含按下的键的字符代码，支持组合按键（特殊符号一般都是由组合按键按出来的）。

实现过程：

- 在消息框的处理函数中，在对应的 WM_CHAR 内添加子类化按钮的消息判断，在其内部，调用我们自己的文本编辑框的过程函数。**在替换之前，首先需要获取原先文本编辑框的窗口过程函数。**
- 自定义文本编辑框的的窗口过程函数
- 在自定义文本编辑框的的窗口过程函数中添加自己想要的功能实现
- wParam 获取键盘的虚拟码，添加判断消息，限制文本编辑框输入的内容
- 符合判断条件的 使用 GetWindowLong 获取原来的窗口过程函数，将符合条件的文本显示到文本编辑框内，不符合的什么也不做，暂时不做处理。

> 实现原理：将原来的文本框的过程函数进行的劫持，将自己定义的文本框过程函数的方法 去替换掉劫持的原来的文本框的过程函数。

示例：

```
89   //· “关于”框的消息处理程序。
90   INT_PTR·CALLBACK·About(HWND·hDlg,·UINT·message,·WPARAM·wParam,·LPARAM·lParam)
91   {
92   ··UNREFERENCED_PARAMETER(lParam);
93   ··switch·(message)
94   ··{
95   ··case·WM_INITDIALOG:
96   ····return·(INT_PTR)TRUE;
97
98   ··case·WM_COMMAND:
99   ··{
100  ····if·(LOWORD(wParam)·==·IDOK·||·LOWORD(wParam)·==·IDCANCEL)
101  ····{
102  ····EndDialog(hDlg,·LOWORD(wParam));
103  ····return·(INT_PTR)TRUE;
104  ····}
105  ····else·if·(LOWORD(wParam)·==·BTN_SUBCLASS)
106  ····{
107  #if·0 非活动预处理器块
114  #endif·
115  ····/*
116  ····这样写当按下两次子类化的时候，会获取两次原来的文本编辑框的窗口过程函数
117  ····但是·最后一次获取的是自定义的文本编辑框的窗口过程函数
118  ····*/
119
120  ····//·获取文本框的句柄
121  ····HWND·hEdit·=·GetDlgItem(hDlg,·EDIT_SUBCLASS);
122  ····//·替换掉原来窗口的过程函数前需要检查一个·g_pfnOldEditProc·等不等于·NULL
123  ····if·(g_pfnOldEditProc·==·NULL)
124  ····{
125  ····//·保存原来的文本编辑框的窗口过程函数
126  ····g_pfnOldEditProc·=·(WNDPROC)GetWindowLong(hEdit,·GWL_WNDPROC);
127  ····//·替换掉原来窗口的过程函数
128  ····SetWindowLong(hEdit,·GWL_WNDPROC,·(LONG)MyEditProc);
129  ····}
```

```
/*
使用 GetWindowLong 和 SetWindowLong
*/

//存储原来的文本编辑框的过程函数
WNDPROC g_pfnOldEditProc = NULL;

//自定义文本编辑框的的窗口过程函数
LRESULT CALLBACK MyEditProc(HWND hEdit, UINT message, WPARAM wParam, LPARAM lParam)
{
    //这里添加自己想要的功能实现
    if (message == WM_CHAR)
    {
        // wParam 获取键盘的虚拟码，添加判断消息，限制文本编辑框输入的内容
        if ((('0' <= wParam) && (wParam <= '9')) ||
            (('a' <= wParam) && (wParam <= 'z')) ||
            (('A' <= wParam) && (wParam <= 'z')))
        {
            //调用原来的窗口过程函数，将符合条件的文本显示到文本编辑框内
            return g_pfnOldEditProc(hEdit, message, wParam, lParam);
        }
        else
        {
            //不显示输入的字符
            return 0;
        }
    }

    /*
    其它的消息（鼠标点击消息等）也要交给原来的文本编辑框的过程函数
    进行处理
    */
    return g_pfnOldEditProc(hEdit, message, wParam, lParam);
}
```

SetClassLong 也可以替换过后过程函数

> SetClassLong 和 SetWindowLong 的效果不一样，SetClassLong --> 将指定的32位(长)值替换到额外的类内存或指定窗口所属的类的 WNDCLASSEX 结构中，GetClassLong --> 获取原来窗口的过程函数

SetClassLong 可以通过 GetClassLong 获取到原来文本编辑框的过程函数，但是在文本框内输入的数据我们自定义的消息过程函数并不会捕捉到，检测不到输入的字符。

原因：控件的窗口类都是系统预定的窗口类，自己也可以使用 CreateWindow 创建出来。

动态创建文本框和创建窗口的操作基本一样：

- SetClassLong 和 GetClassLong 修改的是窗口类中的属性，当创建窗口示例以后，就和窗口类已经没有关系了，当再次创建窗口实例的时候，需要去窗口类中取各种属性，由于原先的窗口类中的属性被 SetClassLong 所修改，所以 SetClassLong 影响的只是以后创建的窗口实例，已经存在的窗口不受影响。
- SetWindowlong 修改的是对应窗口的属性，窗口的属性被修改，对应的窗口也就收到影响。

总结： SDK中子类化的思路就是将自己定义的控件过程函数 替换 成原来控件的过程函数，**在自己的窗口过程函数内对消息进行过滤**以满足自己的需求，处理自己需要的消息，丢掉不需要的消息。

示例：

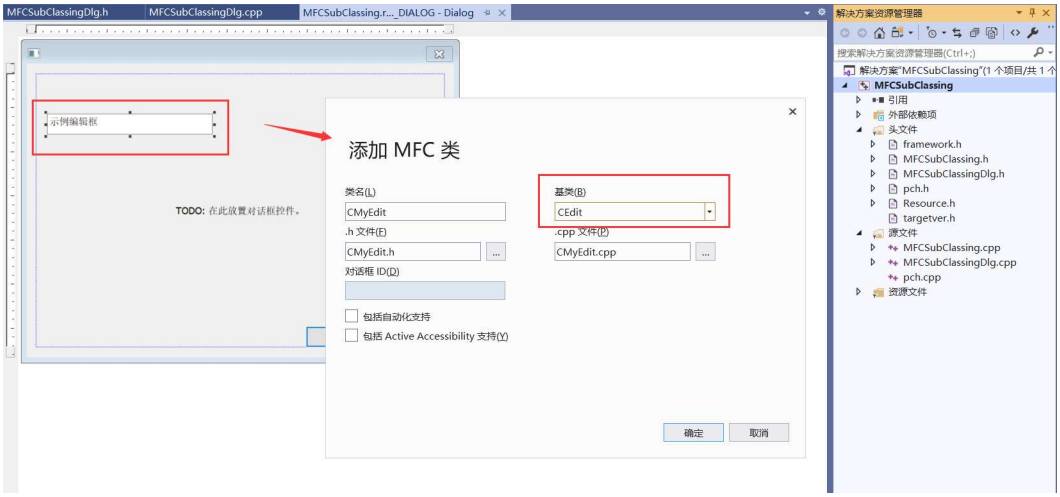
```
        //使用SetClassLong替换过程函数
        else if (LOWORD(wParam) == BTN_SUBCLASS2)
        {
            //获取文本编辑框的窗口句柄
            HWND hEdit = GetDlgItem(hDlg, EDIT_SUBCLASS2);
            if (g_pfnOldEditProc2 == NULL)
            {
                //保存原来的文本编辑框的窗口过程函数
                g_pfnOldEditProc2 = (WNDPROC)GetClassLong(hEdit, GCL_WNDPROC);
                //替换掉原来窗口的过程函数
                SetClassLong(hEdit, GCL_WNDPROC, (LONG)MyEditProc2);
            }
        }
        //动态创建文本框
        else if (LOWORD(wParam) == BTN_CREATEEDIT)
        {

            HWND hNewEditBox = CreateWindow(
                "EDIT",
                "Hello World",
                WS_CHILDWINDOW,
                30, 200, 250, 30,
                hDlg,
                NULL,
                g_hInstance,    //示例句柄
                NULL
            );
            ShowWindow(hNewEditBox, SW_SHOW);
        }
        break;
    }
//自定义文本编辑框的的窗口过程函数
LRESULT CALLBACK MyEditProc2(HWND hEdit, UINT message, WPARAM wParam, LPARAM lParam)
{
    //这里添加自己想要的功能实现
    if (message == WM_CHAR)
    {
        //wParam 获取键盘的虚拟码，添加判断消息，限制文本编辑框输入的内容
        if ((('0' <= wParam) && (wParam <= '9')) ||
            (('a' <= wParam) && (wParam <= 'z')) ||
            (('A' <= wParam) && (wParam <= 'z')))
        {
            //调用原来的窗口过程函数，将符合条件的文本显示到文本编辑框内
            return g_pfnOldEditProc2(hEdit, message, wParam, lParam);
        }
        else
        {
            //不满足条件的特殊字符不显示
            return 0;
        }
    }

    //其它的消息也要交给原来的文本编辑框的过程函数进行处理
    return g_pfnOldEditProc2(hEdit, message, wParam, lParam);
}
```

MFC 子类化

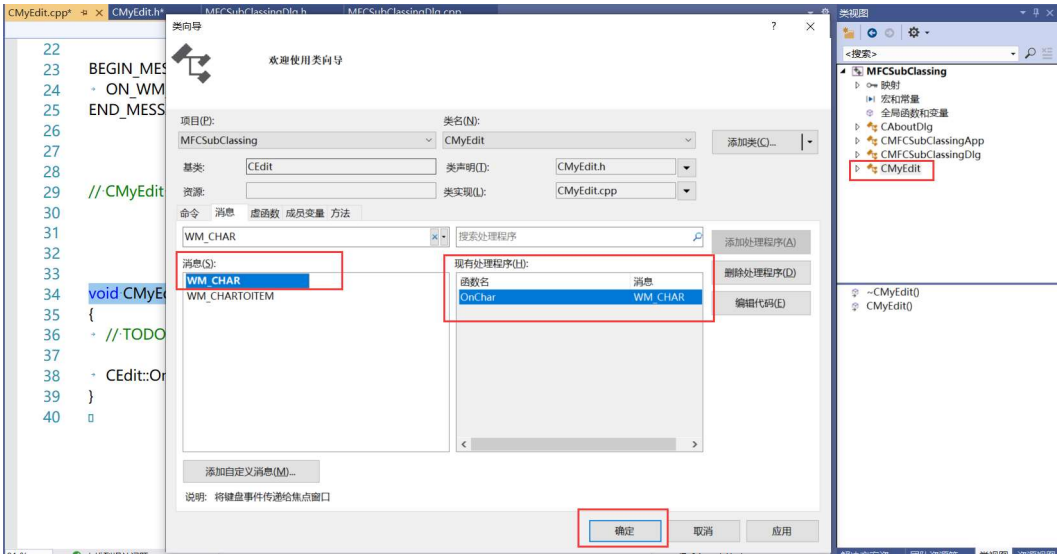
给一个文本编辑框扩展功能，直接继承，因为 MFC 以类的形式对 SDK 的 API 函数进行了封装。

MFC 对常用的控件都做了封装。

为控件添加对应的类，解决方案资源管理器 -- 添加 -- 新建项 -- MFC类 -- 填写对用的类名以及基类（CEdit）。



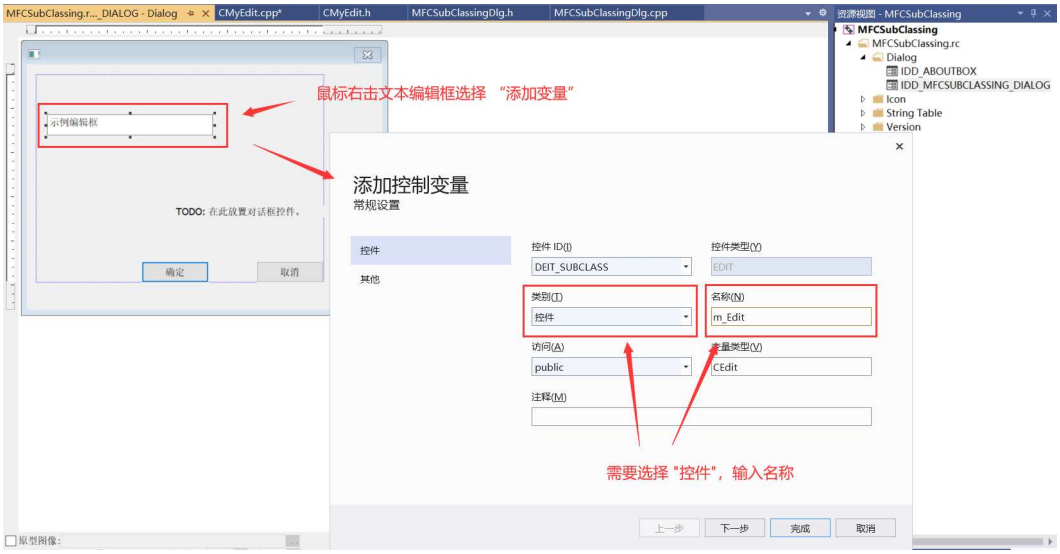
这里需要处理 **WM_CHAR** 消息，所以需要在刚才创建的类中通过类向导添加对应的消息处理。

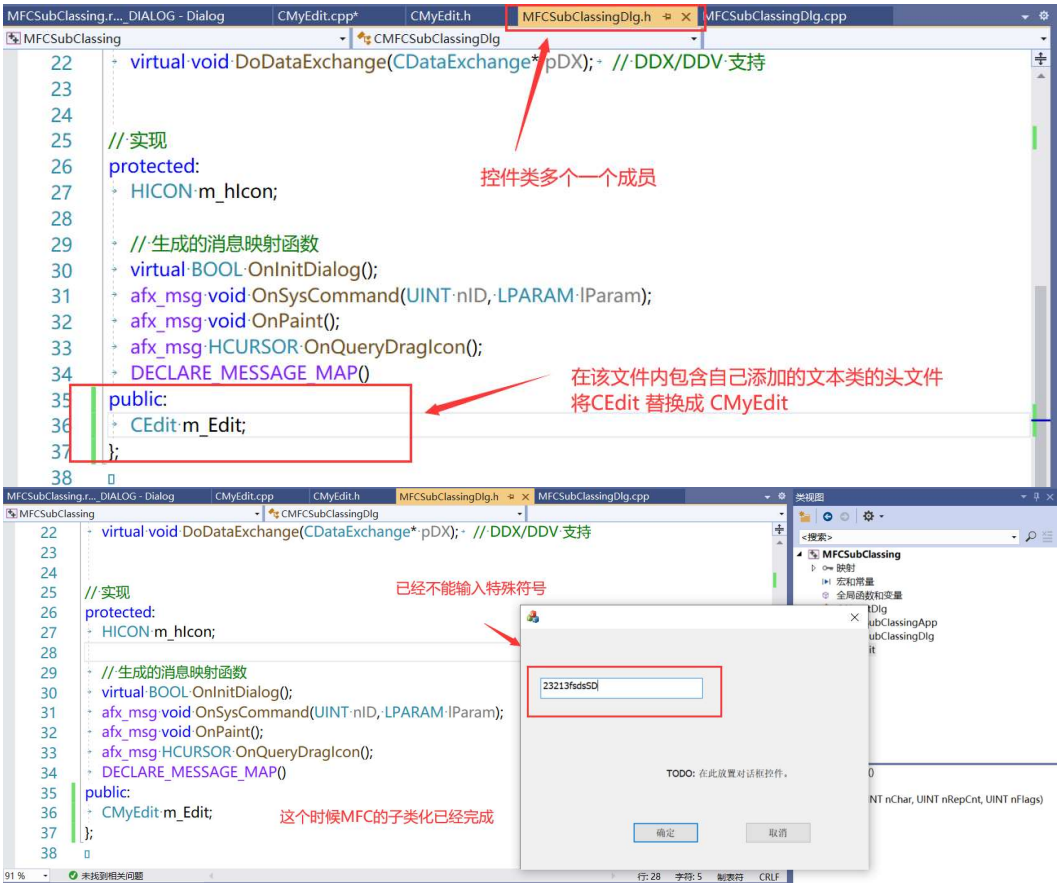


在对应的消息方法内写上自己对应的逻辑代码，示例：

```
void CMyEdit::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    //这里添加自己想要的功能实现
    if (('0' <= nChar) && (nChar <= '9')) ||
        (('a' <= nChar) && (nChar<= 'z')) ||
        (('A' <= nChar) && (nChar <= 'z')))
    {
        //调用原来的窗口过程函数，将符合条件的文本显示到文本编辑框内
        return CEdit::OnChar(nChar, nRepCnt, nFlags);
    }
}
```

创建完成后，需要将文本控件和创建好的类进行绑定，使其关联起来。操作如下图所示：





总结：

MFC 子类化步骤：

1. 新建一个控件类，并继承MFC对应的控件类
2. 在新建的控件类中添加消息处理
3. 为控件添加变量
4. 将添加的变量类型替换为自己创建的控件类的类型

在MFC中如何操控控件

MFC 是以 SDK 类的形式进行的封装，SDK 中控件的使用方法在 MFC 中都可以使用。

在MFC中使用Windows提供的API函数时，在函数名前添加一个 "::" 四饼符，表示使用的 API 函数为 SDK 中的 API 函数。不添加有可能出现同名的成员函数。

文本的获取

方法一：使用 SDK 的方法获取文本编辑框中的文本内容

```
164  void CMFCSubClassingDlg::OnBnClickedGettext()
165  {
166      //通过按钮获取文本编辑框中的文本
167      //使用SDK的API来获取文本编辑框内的字符串文本
168  #if 1
169      /*
170      这里使用GetDlgItemText不需要再过去对话框的句柄，
171      因为CMFCSubClassingDlg内部存了一个
172      */
173      char szBuff[MAXBYTE] = { 0 };
174
175      //m_hWnd --为窗口句柄，数据成员一般是不可以直接使用的
176      //Windows还提供了一个Get方法 --- GetSafeHwnd(),使用方法直接替换m_hWnd即可
177      ::GetDlgItemText(GetSafeHwnd(), EDIT_SUBCLASS2, szBuff, sizeof(szBuff));
178
179      AfxMessageBox(szBuff);  //将缓冲区中的文本数据弹出
180      /*
181      GetSafeHwnd();
182      _AFXWIN_INLINE HWND CWnd::GetSafeHwnd() const
183      { return this == NULL ? NULL : m_hWnd; }
184      */
185  #endif //0
```

方法二：使用 MFC 方法

GetDlgItemText 是对于对话框的封装，在MFC中 Windows 提供了同名函数。

可以查看 MSDN，CDialog 类中没有相关的 API 函数，在 CWnd 类中。

CWnd Class Members

Initialization	Dialog-Box Item Functions	Initialization Message Handlers
Window State Functions	Data-Binding Functions	System Message Handlers
Window Size and Position	Menu Functions	General Message Handlers
Window Access Functions	Tool Tip Functions	Control Message Handlers
Update/Painting Functions	Timer Functions	Input Message Handlers
Coordinate Mapping Functions	Alert Functions	Nonclient-Area Message Handlers
Window Text Functions	Window Message Functions	MDI Message Handlers
Scrolling Functions	Clipboard Functions	Clipboard Message Handlers
Drag-Drop Functions	ActiveX Controls	Menu Loop Notification
Caret Functions	Overridables	

CWnd::GetDlgItemText

int GetDlgItemText(int *nID*, LPTSTR *lpStr*, int *nMaxCount*) const;

int GetDlgItemText(int *nID*, CString& *rString*) const;

Return Value

Specifies the actual number of bytes copied to the buffer, not including the terminating null character. The value is 0 if no text is copied.

1. MFC 方法一：使用GetDlgItemText

```
186
187      //MFC方法
188      //方法一
189      CString strText;
190      GetDlgItemText(EDIT_SUBCLASS2, strText);
191      AfxMessageBox(strText);
192
```

2. MFC方法二：使用GetWindowText

```
// 方法二
CString  strText;
// 这里运行程序点击按钮的时候，会获取父窗口的标题
```

```
GetWindowText(strText);
AfxMessageBox(strText);
```

使用 GetWindowTest 需要使用 GetDlgItem 获取文本编辑框的指针，然后在使用文本编辑框的句柄获取对应文本编辑框内的文本数据。

```
CString  strText;
CWnd*  pEdit  =  GetDlgItem(EDIT_SUBCLASS2);
pEdit->GetWindowText(strText);
AfxMessageBox(strText);
```

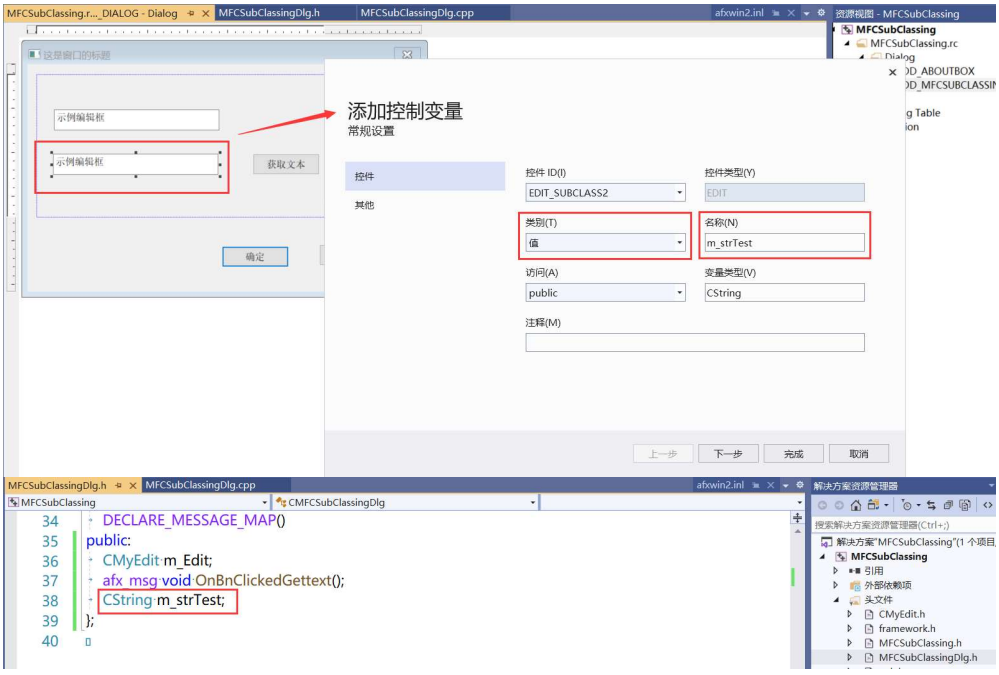
3. MFC方法三：使用WM_GETTEXT消息，输入的文本数据会保存到缓冲区中，通过 WM_GETTEXT 获取缓冲区的数据后将缓冲区中的数据弹出（同样需要通过GetDlgItem来获取文本编辑框的句柄）。

```
CWnd*  pEdit  =  GetDlgItem(EDIT_SUBCLASS2);
char  szBuff[MAXBYTE]  =  {  0  };
pEdit->SendMessage(WM_GETTEXT,  sizeof(szBuff),  (LPARAM)szBuff);
AfxMessageBox(szBuff);
```

4. MFC方法四：MFC DDX

对编辑框进行操作的时候，MFC提供的机制允许把编辑框这个窗口和一个CString类型绑定在一块，对CString 进行赋值，这个数值就会显示在编辑框内，从CString 中获取数据就是从编辑框内获取数据，其它控件也可以使用。

使用之前需要将文本编辑框进行添加 控制变量 操作，类别选择 "值"，并给个名称，点击完成即可。



使用示例：

使用时直接将输入的文本数据从CString中弹出就行，使用前还需要一个API函数 -- UpdateData

UpdateData -- 只有一个参数，参数有两个值可选，TRUE，FALSE

参数为TRUE，将文本编辑框内的文本数据存到 CString 对象内，参数为 FALSE 时，从 CString 对象中，获取文本数据到文本编辑框内。

```
void CMFCSubClassingDlg::OnBnClickedGettext()
{
    // MFC DDX
    // 对编辑框进行操作的时候，MFC提供的机制允许把编辑框这个窗口和一个CString类型绑定在一块
    // 对CString 进行赋值，这个数值就会显示在编辑框内
    // 从CString 中获取数据就是从编辑框内获取数据
    // 其它控件也可以进行类似的使用
    UpdateData(TRUE);
    AfxMessageBox(m_strTest);
}
void CMFCSubClassingDlg::OnBnClickedSettext()
{
    // TODO: 在此添加控件通知处理程序代码
    m_strTest = "Hello World";
    UpdateData(FALSE);
}
```

UpdateData 内部还是使用的 GetWindowText 函数。

```
204 void AFXAPI DDX_Text(CDataExchange* pDX, int nIDC, CString& value)
205 {
206     HWND hWndCtrl = pDX->PrepareEditCtrl(nIDC);
207     if (pDX->m_bSaveAndValidate)
208     {
209         int nLen = ::GetWindowTextLength(hWndCtrl);
210         ::GetWindowText(hWndCtrl, value.GetBufferSetLength(nLen), nLen+1);
211         value.ReleaseBuffer();
212     }
213     else
214     {
215         AfxSetWindowText(hWndCtrl, value);
```

运行效果：

上述就是MFC中对控件的操作， 对API进行了封装，传参的时候不需要传递窗口的句柄，同时还提供了一些重置API，方便使用CString进行相关的操作。

控件和变量的绑定可以看下图中的对应关系，或者类向导中的**成员变量**进行查

看：

```
60  void CMFCSubClassingDlg::DoDataExchange(CDataExchange* pDX)
61  {
62      CDialogEx::DoDataExchange(pDX);
63      DDX_Control(pDX, EDIT_SUBCLASS, m_Edit);
64      DDX_Text(pDX, EDIT_SUBCLASS2, m_strTest);
65  }
66
```

CString的用法

最常用的就是使用CString当作缓冲区来使用。**GetBufferSetLength** 和 **ReleaseBuffer** 的使用。
GetBufferSetLength -- 根据字符串的长度申请对应的缓冲区，获取到缓冲区的地址，往该缓冲区中填充字符串。
ReleaseBuffer -- 使用 ReleaseBuffer 结束使用由 GetBuffer 分配的缓冲区，使用 memcpy 时，在缓冲区的末尾添加 '\0'