

网络程序设计

简单的window程序设计

3.1 Visual C++ 2019概述

3.1.1 Visual Studio与Visual C++

- Visual studio是微软公司提供的专门用于开发Windows应用程序的集成开发环境，最新版本为 Microsoft Visual studio 2019。
- 从Visual Studio 2002开始， Visual studio就引入了一种新的计算平台——Microsoft.NET。
- Visual studio.NET是一种支持多种语言的应用程序开发平台，在Visual studio.NET中可以使用VB.NET、C#.NET、VC++.NET等语言开发程序，甚至你可以用其中一种语言开发应用程序的一部分，用另外的语言开发程序的其他部分。

- Visual C++从发布起到现在已经有10多个大版本了。
- 微软1998年8月推出的Visual C++ 6.0堪称是历史上最经典的VC版本，大量的教材都是基于这个版本来写的，甚至到现在仍然还有一些企业在使用这一版本。
- 随着新版Windows系统的不断推出，VC6.0已越来越难以在新版Windows系统上适用，甚至出现了兼容性问题；并且VC6.0与新版VC的差别也越来越大，熟悉VC6.0对熟悉新版VC的帮助也越来越少，因此，VC6走向其终结点是必然的。
- Visual studio 2019中的Visual C++通常被称为Visual C++ 2019，其版本号是16.0，因此有时也被简称VC16。尽管该版本不是最新版本，但与最新版本相差不大。

3.1.2 使用Visual C++开发控制台应用程序

文件->新建->项目



配置新项目

控制台应用

C++

Windows

控制台


项目名称(N)

HelloWorld

位置(L)

D:\C

...

解决方案名称(M) 

HelloWorld



将解决方案和项目放在同一目录中(D)

上一步(B)

创建(C)

文件(F) 编辑(E) 视图(V) 项目(P) 生成(B) 调试(D) 测试(S) 分析(N) 工具(T) 扩展(X) 窗口(W) 帮助(H)

搜索 Visual Studio (Ctrl+Q)

Hell...World

1

Debug x86 本地 Windows 调试器

Live Share

HellowWorld.cpp

HellowWorld (全局范围)

```
1 // HellowWorld.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
2 //
3
4 #include <iostream>
5
6 int main()
7 {
8     std::cout << "Hello World!\n";
9 }
10
11 // 运行程序: Ctrl + F5 或调试 > "开始执行(不调试)" 菜单
12 // 调试程序: F5 或调试 > "开始调试" 菜单
13
14 // 入门使用技巧:
15 // 1. 使用解决方案资源管理器窗口添加/管理文件
16 // 2. 使用团队资源管理器窗口连接到源代码管理
17 // 3. 使用输出窗口查看生成输出和其他消息
18 // 4. 使用错误列表窗口查看错误
19 // 5. 转到“项目”>“添加新项”以创建新的代码文件，或转到“项目”>“添加现有项”以将现有代码文件添加到项目
20 // 6. 将来，若要再次打开此项目，请转到“文件”>“打开”>“项目”并选择 .sln 文件
21
```

解决方案资源管理器

搜索解决方案资源管理器(Ctrl+;)

解决方案“HellowWorld”(1 个项目/共 1 个)

HellowWorld

引用

外部依赖项

头文件

源文件

HellowWorld.cpp

资源文件

解决方案资源管理器 团队资源管理器

属性

HellowWorld.cpp 文件属性

杂项

(名称)	HellowWorld.cpp
包括在项目中	True
内容	False
完整路径	D:\C\HellowWorld\HellowWorld
文件类型	C/C++ 代码
相对路径	HellowWorld.cpp

(名称)

命名文件对象。

输出

显示输出来源(S):

错误列表 输出

就绪

行 1 列 1 字符 1

Ins

6

添加到源代码管理

将代码编辑器中的代码改写为如下代码，然后点击菜单“调试”→“开始执行（不调试）”：

// HelloWorld.cpp：定义控制台应用程序的入口点。

```
#include "iostream"
int main()
{
    using namespace std;
    cout << "Hellow World!" << endl;
    return 0;
}
```

3.1.3 使用VC++开发图形界面应用程序的方法

- 使用Microsoft Visual studio 2019中的任何一种语言都既可以开发字符界面的控制台应用程序，也可以开发具有图形界面的Windows应用程序。
- 在使用Visual C++开发具有图形用户界面的Windows应用程序可以采用以下三种方法之一。
 - **(1) 使用Windows API函数**
 - **(2) 使用MFC**
 - **(3) 使用Visual C++.NET**
- **我们主要学习使用Visual C++ 2019创建基于MFC的对话框应用程序相关的基本概念和基本方法。**

3.2 Visual C++的数据类型

- 在使用Visual C++编写Windows 程序时，除了可以直接使用标准C++中的数据类型外，为了提高程序的可读性，Visual C++还定义了一些自己特有的数据类型，这些数据类型大都是标准C++中的数据类型的重新定义。

数据类型	对应的基本数据类型	说明
BOOL	bool	布尔值
BSTR	unsigned short*	16位字符指针
BYTE	unsigned char	8位无符号整数
DWORD	unsigned long	32位无符号整数，段地址和相关的偏移地址
LONG	long	32位带符号整数
LPARAM	long	作为参数传递给窗口过程或回调函数的32位值
LPCSTR	const char*	指向字符串常量的32位指针
LPSTR	char*	指向字符串的32位指针
LPVOID	void*	指向未定义的地类型的32位指针
UNIT	unsigned int	32位无符号整数
WORD	unsigned short	16位无符号整数
WPARAM	unsigned int	当作参数传递给窗口过程或回调函数的32位值

- VC++中的数据类型都是以大写字符出现的，这主要是为了与标准C++的基本数据类型相区别。
- VC++中的数据类型的命名也是有规律的，从数据类型的名字基本可以看出数据类型的意义。
 - 指针类型的命名方式一般是在其指向的数据类型前加“LP”或“P”，比如指向DWORD的指针类型为“LPDWORD”和“PDWORD”；
 - 无符号类型一般是以“U”开头，比如“INT”是符号类型，“UINT”是无符号类型。

- VC++还提供一些宏来处理基本数据类型：
 - LOBYTE和HIBYTE这两个宏分别用来获取16位数值中的低位和高位字节；
 - LOWORD和HIWORD分别用来获取32位数值中的低16位和高16位字；
 - MAKEWORD则是将两个16位无符号数合成一个32位无符号数，等等。

字符串类型

- 字符串类型CString本质上是VC++提供的一个字符串类。它提供了很多非常有用的操作和成员函数，使用CString可以方便地对字符串进行处理。
- 例如，你可以利用如下方法连接字符串：
 CString gray("Gray");
 CString cat("Cat");
 CString graycat = gray + cat;
 //执行后graycat的值为"GrayCat"

- 利用格式化可以把其它不是CString类型的数据转化为CString类型
- 例：把一个整数转化成CString类型
 CString s;
 s.Format("%d", total);

- 将char* 转化为 CString则直接可采用如下方法：

```
char * p = "This is a test";
```

```
CString s = p;
```

```
CString s(p);
```

```
或 CString s = "This is a test";
```

- 获取存储字符串的存储区的指针（char* 类型），调用GetBuffer()则可，调用getlength()则可获得字符串长度。
- 常用到的成员函数还包括：
 - BOOL IsEmpty(); //测试CString类对象包含的字符串是否为空。
 - void Empty(); //使CString类对象包含的字符串为空字符串。
 - TCHAR GetAt(int nIndex) ;//获取字符串中指定位置（由nIndex指定）的字符。
 - void SetAt(int nIndex,TCHAR ch);//将字符串指定的位置处的字符设定为ch。
 - CString Left(int nCount); //截取字符串左边nCount长度的字符串。
 - CString Right(int nCount); //截取字符串右边nCount长度的字符串。

- CString用于存储字符串的存储区是动态分配的，即它支持动态内存分配，因此完全不用担心CString的大小。另外，如果项目选择使用UNICODE字符集，则CString对象将采用UNICODE字符，否则则使用ANSI字符。
- CString的声明位于头文件"atlstr.h"中，在程序中如果要使用CString，通常需要添加：`#include <atlstr.h>`。在MFC程序中，由于MFC的头文件中已包含了该文件，因此一般不再需要在包含此文件。

句柄类型

- 句柄类型是Windows程序设计中的一种特殊数据类型，是Windows用来唯一标识应用程序所建立或使用的对象的一个32位的无符号整数值。
- Windows程序中需要使用句柄标识的对象包括应用程序实例、窗口、位图、内存块，文件、任务等。
- 各种句柄类型的命名方式一般都是在对象名前加“H”，比如位图（BITMAP）对应的句柄类型为“HBITMAP”。

- 句柄本质上是一种指向指针的指针。
- 一个应用程序启动后，组成该程序的各对象是驻留在内存的。理论上只要获知对象所在内存的首地址就可以随时访问该对象。但为了优化内存，Windows内存管理器会经常移动内存中的对象，对象移动就意味着其首地址的变化。
- 为了让应用程序能够找到首地址经常变化的对象，Windows系统为各应用程序预留了一些内存空间，专门用于登记各对象在内存中的地址变化，这些预留空间的地址是不变的，句柄就是指这些预留空间的地址。
- Windows内存管理器在移动了对象后，会把对象新的地址保存于句柄对应的存储单元，这样只需记住句柄地址就可以间接地知道对象在内存中的位置了。

- 句柄是在对象被装载(Load)到内存中时由系统分配的，当系统卸载(Unload)该对象时，句柄将被释放给系统。
- Windows应用程序几乎总是通过调用一个WINDOWS API函数来获得一个句柄，当我们调用WINDOWS API函数获取某个对象时，API函数通常会给该对象分配一个确定的句柄，并将该句柄返回给应用程序，然后应用程序可通过句柄来对该对象进行操作。

- 在WINDOWS编程中会用到大量的句柄，比如：HINSTANCE（实例句柄），HBITMAP（位图句柄），HDC（设备描述表句柄），HICON（图标句柄）等。教材中的表2.3列出了一些常见的句柄类型。
- 除此之外，还有一个经常用到的通用的句柄，就是HANDLE。

3.3 UNICODE字符集

- 计算机最早使用的字符集是标准ASCII码字符，但标准ASCII码字符不能满足中国、日本等亚洲国家的需求，于是各国便针对本国的语言文字制定了相应的字符编码标准，如GB2312、BIG5、JIS等。这些字符编码标准统称为ANSI编码标准。
- 通常，ANSI编码标准都与ASCII码兼容，并且字符数目要比ASCII字符多很多，因此必需使用多个字节来表示一个字符。
- ANSI字符集只规定了本国或地区所使用的语言的“字符”，并未考虑其他国家或地区的ANSI码，导致各种ANSI编码空间重叠，使得同一个二进制编码在使用不同的ANSI字符集的系统中被解释成不同的符号，这会导致使用不同ANSI编码标准的系统交换信息时出现乱码。

- Unicode组织联合国际标准化组织，制定了Unicode字符集，其目的是将世界上绝大多数国家的文字符号都编入其中，并为每种语言的每个字符设定唯一的一个二进制编码，以满足跨语言、跨平台进行文本传输和处理的要求。
- Unicode是Universal Multiple-Octet Coded Character Set的缩写，中文翻译为“通用多八位编码字符集”。从1991年10月的Unicode1.0到2014年6月的Unicode7.0，Unicode总共已发布了20个版本。这20个版本从2.0开始，都保是向后兼容的，也就是说，新版本只是增加字符，对原有字符不作任何改动。

Unicode编码与UTF

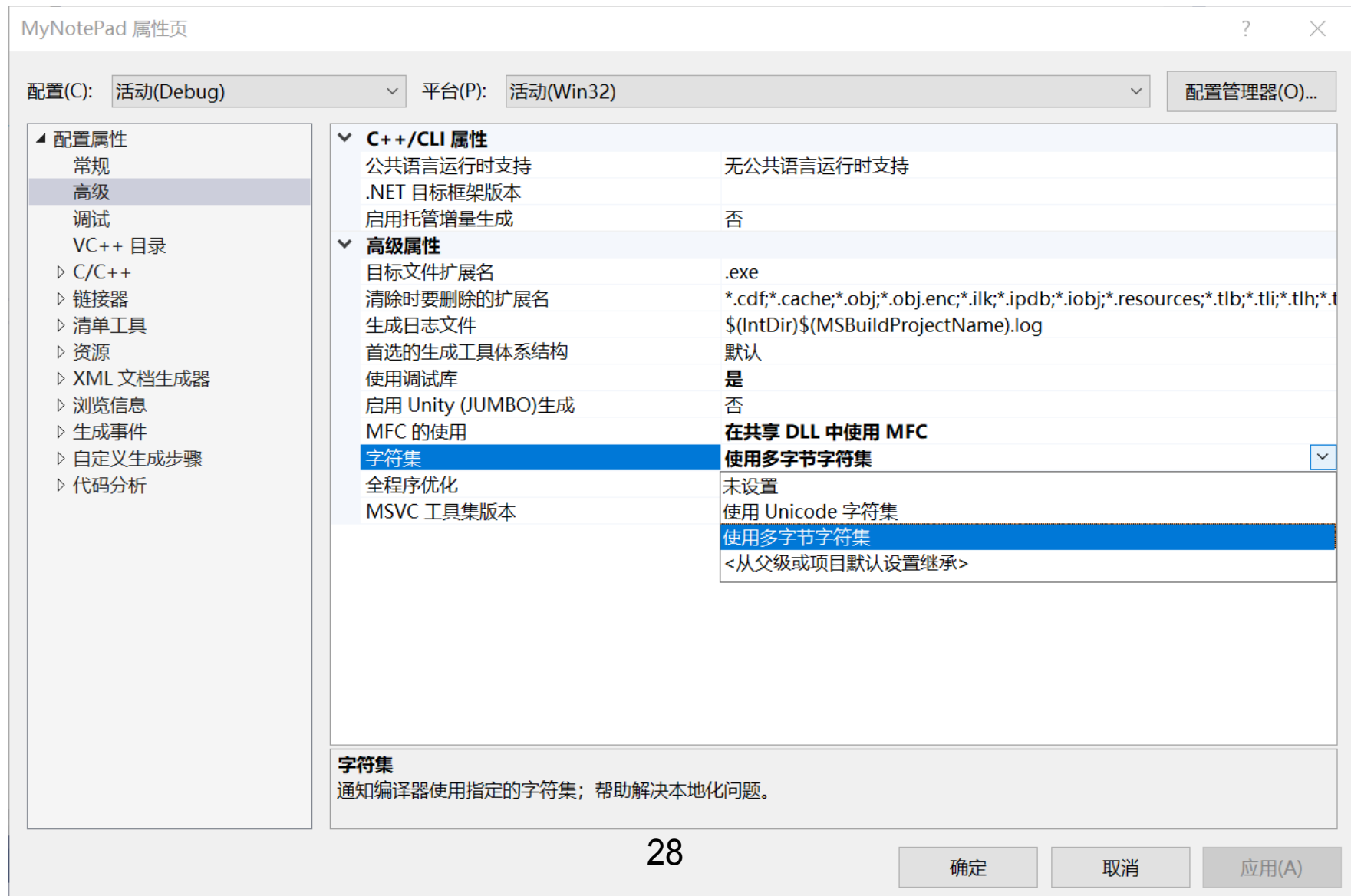
- Unicode是一种多字节编码方案，最初的版本采用两个字节来表示一个字符，即UCS-2 (Universal Character Set coded in 2 octets)，其取值范围为 U+0000 ~ U+FFFF。
- 在这个范围中每个数字都称为一个码位 (code point)，每个码位对应一个字符，由此不难算出，UCS-2最多只能表示65536个字符。
- 为了能表示更多的文字，后来的Unicode版本采用四个字节来表示代码点，即UCS-4，即它的范围为 U+00000000 ~ U+7FFFFFFF，其中 U+00000000 ~ U+0000FFFF同UCS-2完全一样，书写时通常也不写前面全0的两个字节，因此，UCS-4可以看成是在UCS-2基础上又增加了一些4字节表示的字符。

- UCS-2和UCS-4只规定了代码点和字符的对应关系，但并没有规定代码点在计算机中如何存储。
- 比如，“计算机”三个汉字的Unicode代码分别是U+8BA1、U+7B97和U+673A，那么在计算机中存储这三个汉字时是存为“8B A1 7B 97 67 3A”还是“A1 8B 97 7B 3A 67”？
- 回答是都有可能，因为人们在发明计算机之初，对双字节数据的存储就存在有两种方案，大端顺序（Big Endian）和小端顺序（Little Endian），前者，即字节存储顺序跟书写顺序一致的是大端顺序，后者称为小端顺序。对Unicode编码的字符，在具体现实时到底是采用大端顺序还是小端顺序，必须要给出一个规定。

- 规定代码点在计算机中存储方式的是Unicode转换格式（ Unicode Transformation Format , UTF ）编码。
- 事实上，提出UTF的原因并不只是“大端顺序”和“小端顺序”，还有一个原因是为了节省存储空间。
- 例如，英文字母及英文标点符号所对应的Unicode编码均是2个字节，而且编码第一字节的8位均为0，第二字节则是相应得ASCII编码，如果要使用原始Unicode编码存储一篇只包含纯英文字符的文章，显然就会造成比较大的浪费，因为一半的存储空间存储的都是字节0。
- 目前常用的UTF编码包括UTF-8、UTF-16、UTF-16BE（ Big Endian ）、UTF-16LE（ Little Endian ）。

- Visual C++ 2019对ANSI和Unicode两种字符集都支持，UTF-16是Windows默认的Unicode编码方式，默认采用小端顺序存储。

- 在一个已打开的VC++项目中，可以修改它所使用的字符集。



- 当在程序中使用ANSI字符集时，使用单字节字符（char类型）数组存储字符或字符串。如果在程序中使用Unicode字符集，则使用双字节（16位）字符类型，也称为宽字符类型，其类型符为wchar_t。
- 无论项目使用的是ANSI字符集还是Unicode字符集，程序中用双引号引起来的字符串常量都是单字节字符串，如果需要以双字节字符存储，则需要在字符串前加上L。

- 例：定义并初始化一个宽字符数组

```
wchar_t Str[] = L"Hello World!";
```

- 如果将L丢掉，写成 `wchar_t Str[] = "Hello World!"`; 程序编译就不会通过，会出现这样的错误提示：

error C2440 : “初始化” : 无法从 “const char [13]” 转换为 “wchar_t []” 。

- 在控制台应用中，双字节字符串与单字节字符串所使用的输入输出函数是不同的，输入、输出双字节字符串数据的函数分别是wscanf()和wprintf()，这两个函数使用方法与scanf和printf类似，请看下面的例子：

```
wchar_t a[20];  
wscanf(L"%s",a);  
wprintf(L"%s\n",a);
```

- 不仅仅是输入输出函数，其它的一些字符串处理函数，对宽体字符与单字节字符也是不同的。
- 所有的Unicode字符串函数均以wcs开头，wcs是宽字符串的英文缩写。若要调用Unicode函数只需用前缀wcs来取代ANSI字符串函数的前缀str即可。

- 不难看出，使用不同字符集时程序代码的差别是比较大的。为了实现同样的代码既适用于宽体字符也适用于窄体字符，微软将这两套字符集的操作进行了统一，对两套字符集所使用的不同的关键字和库函数名，都定义了一致的宏来替代，并使用条件编译来控制。
- 例如，无论使用那种字符集，字符类型的定义符可使用TCHAR，因为如果要使用窄体字符，则不定义_UNICODE宏，此时TCHAR就是char；如果使用宽体字符，则必须定义_UNICODE宏，这时TCHAR就是wchar_t。

- 双引号引起的字符串常量可写为 `_T("Hello World!")`，当定义了 `_UNICODE` 宏，就相当于 `L"Hello World!"`，没定义则相当于 `"Hello World!"`。
- 相关的字符串操作函数也都有对应的替换函数，比如，`tcslen()` 函数，在定义了 `_UNICODE` 宏时它就相当于 `wcslen()`，没定义时则相当于 `strlen()`。
- 这些宏的定义都在头文件 `tchar.h` 中，因此，在程序前面需要有编译预处理命令 `#include <tchar.h>`。但在 Visual C++ 2019 中，通常是不需要手工添加的，因为在头文件 `stdafx.h` 中已包含了这一命令。`stdafx.h` 是 Visual C++ 2019 创建的标准工程时自动创建的一个头文件，主要作用是包含系统所需的一些头文件。

- 由于大家大都只熟悉在ANSI字符集下的编程，因此本课程中的所有例题，均使用ANSI字符集，如果没有特别说明，相关代码都是ANSI字符集下的写法。

3.4 MFC对话框应用程序

- 对话框（Dialog Box）是一种特殊的窗口，主要功能是显示输出信息或者接收用户输入。
- 对话框应用程序是指主窗口样式为“对话框”的应用程序。
- 对话框是一种简单的框架窗口，只有标题栏和边框，没有菜单条、工具条和状态条等，通常包含若干控件(Control)，通过这些控件，它可以接收用户的输入和选择、向用户显示信息或者响应用户各种操作。

- 控件是指窗口上的编辑框（ Edit Box ）、命令按钮（ Command Button ）、下拉列表框（ List Box ）等具有一定输入输出功能的小部件。
- 控件本质上也是一种简单窗口，只不过它必须存在于一个称为其父窗口的窗口内，在对话框中，对话框窗口就是其上各种控件的父窗口。
- 使用Visual Stdio 2010的应用程序向导，可以创建一个简单的对话框应用程序框架。

3.4.1 对话框应用程序的创建



配置新项目

MFC 应用

C++

Windows

桌面


项目名称(N)

lx

位置(L)

D:\C



解决方案名称(M) 

lx

☐ 将解决方案和项目放在同一目录中(D)

上一步(B)

创建(C)

MFC 应用程序

应用程序类型选项

应用程序类型

文档模板属性

用户界面功能

高级功能

生成的类

应用程序类型(T)

基于对话框

应用程序类型选项:

☐ 选项卡式文档(B)

☐ 文档/视图结构支持(V)

基于对话框的选项(I)

<无>

复合文档支持

<无>

文档支持选项:

☐ 活动文档服务器(A)

☐ 活动文档容器(D)

☐ 支持复合文件(U)

项目样式

MFC standard

视觉样式和颜色(Y)

Windows Native/Default

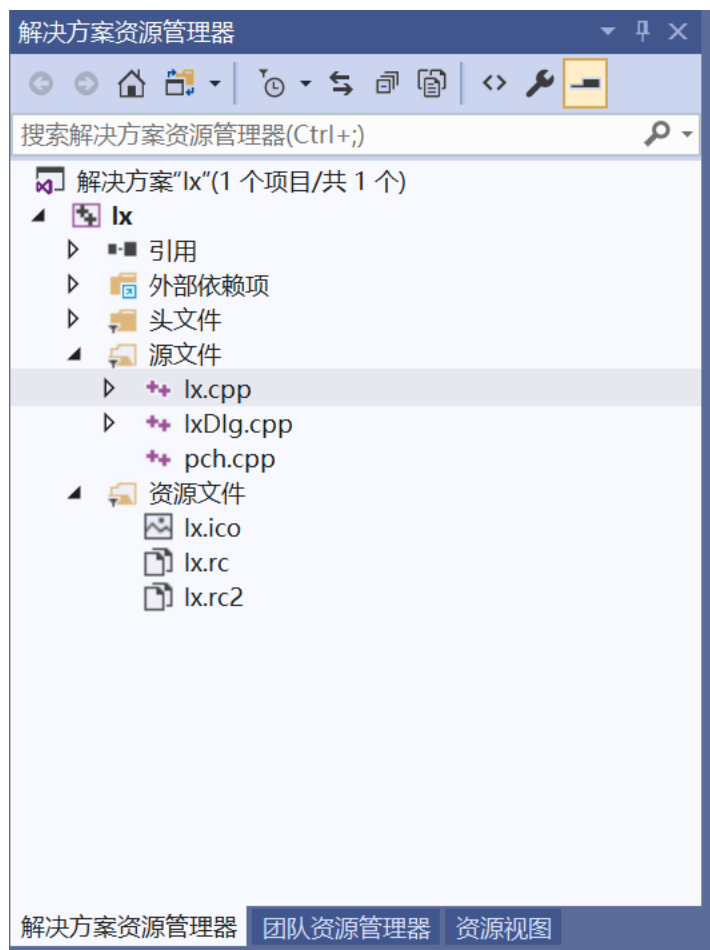
☐ 启用视觉样式切换(C)

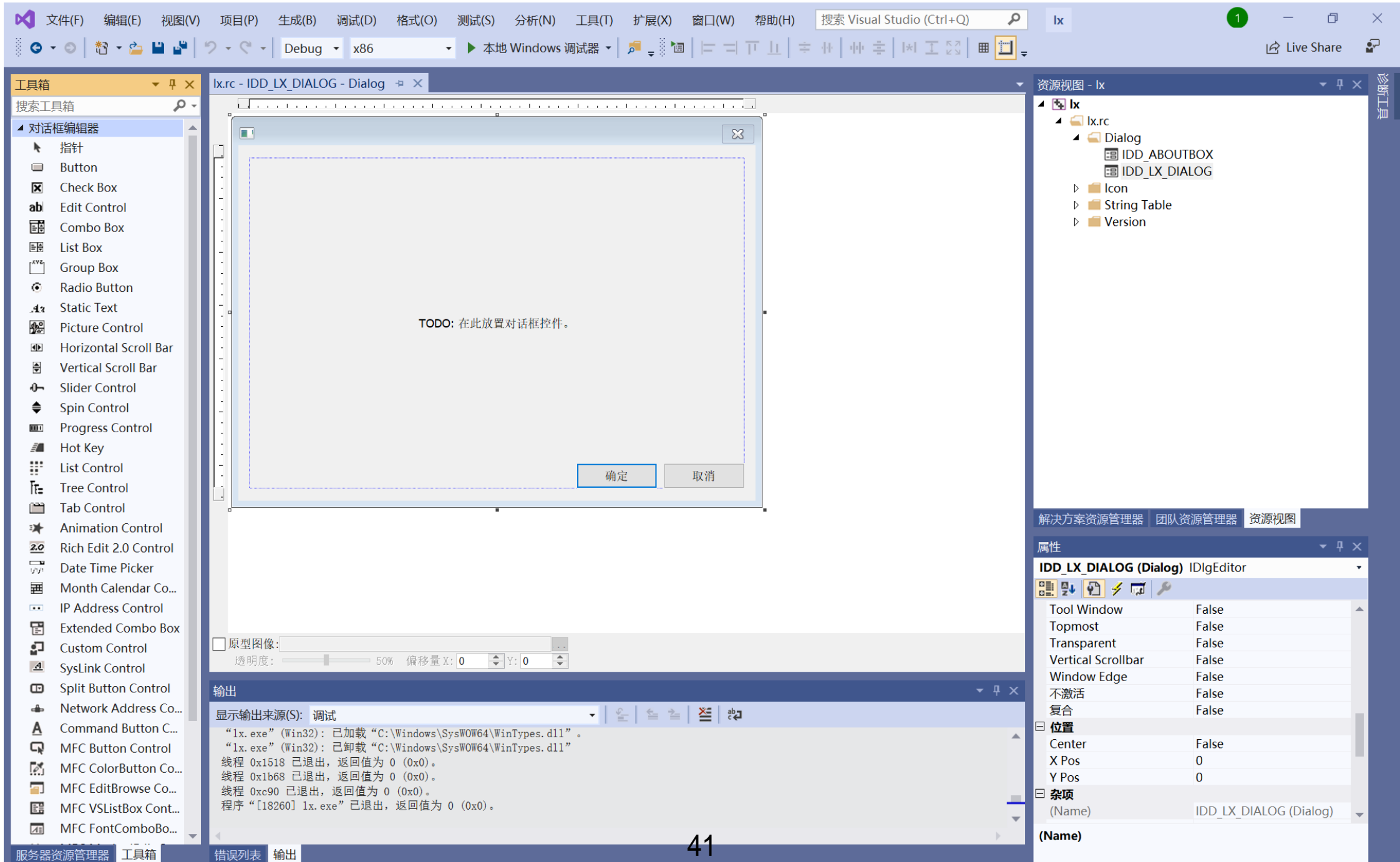
资源语言(L)

English (United States)

使用 MFC

在共享 DLL 中使用 MFC





3.4.2 VC程序结构

- 头文件(.h)和cpp文件
- 类声明和类成员声明在头文件中
- 类实现在cpp文件中

3.4.3 MFC对话框应用程序结构

- MFC应用程序向导会自动生成三个类：CaboutDlg，ClxApp，ClxDlg
- ClxApp是最重要的一个类，该类派生于CWinApp类。
 - CWinApp是MFC的主应用程序类，用于控制应用程序的初始化、运行和终止。
 - MFC应用程序向导生成的应用程序必须有且仅有一个从 CWinApp 派生的类的对象，该对象在创建窗口之前构造。

lx.h lx.rc - IDD_LX_DIALOG - Dialog

lx (全局范围)

```
1
2 // lx.h: PROJECT_NAME 应用程序的主头文件
3 //
4
5 #pragma once
6
7 #ifndef __AFXWIN_H__
8     #error "在包含此文件之前包含 'pch.h' 以生成 PCH"
9 #endif
10
11 #include "resource.h" // 主符号
12
13
14 // ClxApp:
15 // 有关此类的实现，请参阅 lx.cpp
16 //
17
18 class ClxApp : public CWinApp
19 {
20 public:
21     ClxApp();
22
23     // 重写
24 public:
25     virtual BOOL InitInstance();
26
27     // 实现
28
29     DECLARE_MESSAGE_MAP()
30 };
31
32 extern ClxApp theApp;
33
```

100 % 未找到相关问题

```
lx.cpp  lx.h  lx.rc - IDD_LX_DIALOG - Dialog
lx (全局范围)
1
2 // lx.cpp: 定义应用程序的类行为。
3 //
4
5 #include "pch.h"
6 #include "framework.h"
7 #include "lx.h"
8 #include "lxDlg.h"
9
10 #ifdef _DEBUG
11 #define new DEBUG_NEW
12 #endif
13
14 // C1xApp
15
16
17 BEGIN_MESSAGE_MAP(C1xApp, CWinApp)
18 | ON_COMMAND(ID_HELP, &CWinApp::OnHelp)
19 END_MESSAGE_MAP()
20
21
22 // C1xApp 构造
23
24 C1xApp::C1xApp() { ... }
25
26
27 // 唯一的 C1xApp 对象
28
29 C1xApp theApp;
30
31
32 // C1xApp 初始化
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
```

- ClxDlg类的声明如下：
class ClxDlg : public CDialogEx
 - CDialogEx是CDialog类的派生类，而CDialog又是CWnd的派生类，因此在此在CLXDlg类的实现中，可以使用CDialog类以及CWnd类的所有方法。

```
lxDlg.h  X
lx (全局范围)
1
2 // lxDlg.h: 头文件
3 //
4
5 #pragma once
6
7
8 // ClxDlg 对话框
9 class ClxDlg : public CDialogEx
10 {
11 // 构造
12 public:
13     ClxDlg(CWnd* pParent = nullptr);    // 标准构造函数
14
15 // 对话框数据
16 #ifdef AFX_DESIGN_TIME
17     enum { IDD = IDD_LX_DIALOG };
18 #endif
19
20 protected:
21     virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV 支持
22
23
24 // 实现
25 protected:
26     HICON m_hIcon;
27
28     // 生成的消息映射函数
29     virtual BOOL OnInitDialog();
30     afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
31     afx_msg void OnPaint();
32     afx_msg HCURSOR OnQueryDragIcon();
33     DECLARE_MESSAGE_MAP()
34 };
35
```

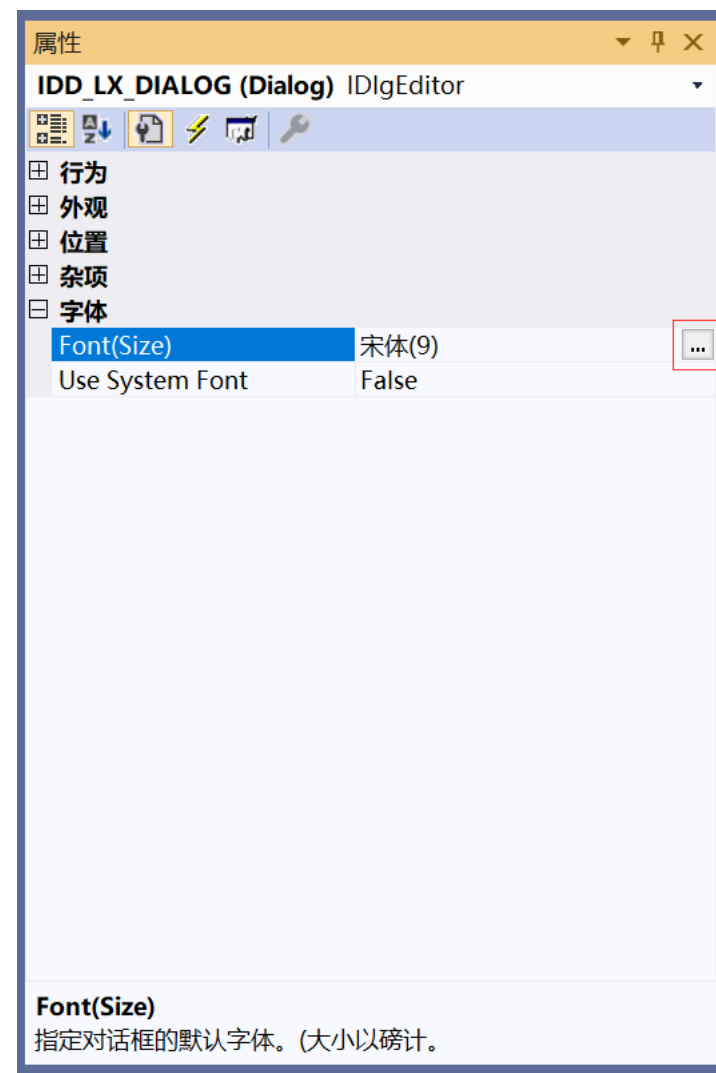
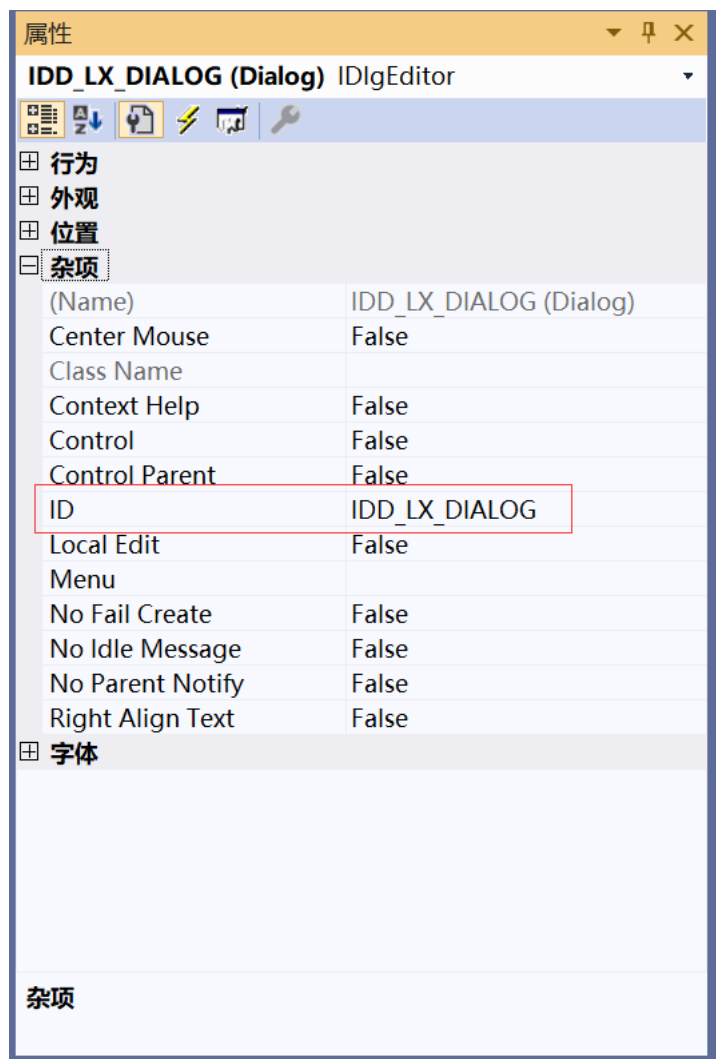
3.4.4 设置对话框的属性

- 对话框窗口的外观和行为特性可通过修改它的属性值进行改变。对话框窗口的属性可以通过“属性窗口”设置。
- 要设置对话框属性，首先打开“对话框资源编辑器”和“属性窗口”，在新建项目中，这两个窗口默认是被打开的。

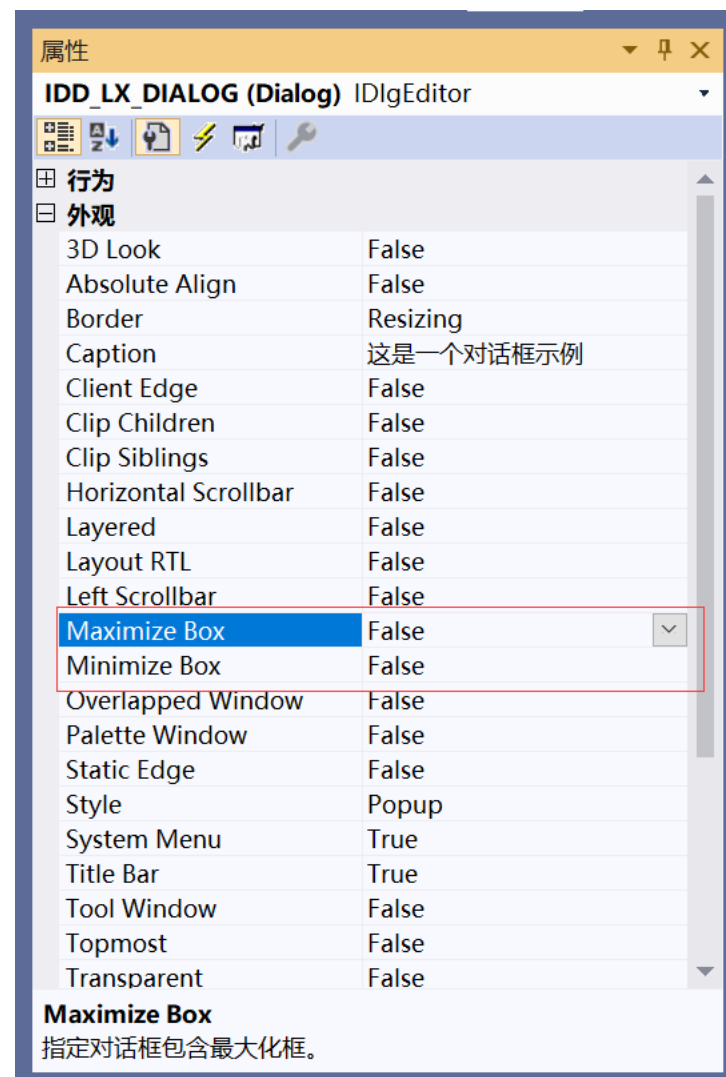
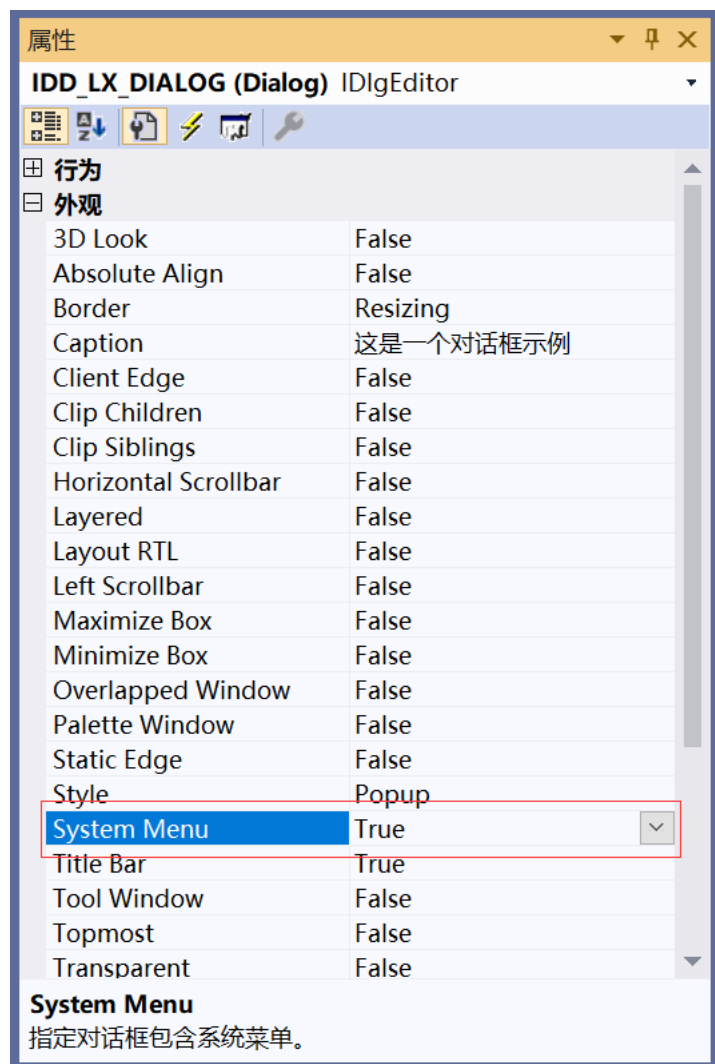
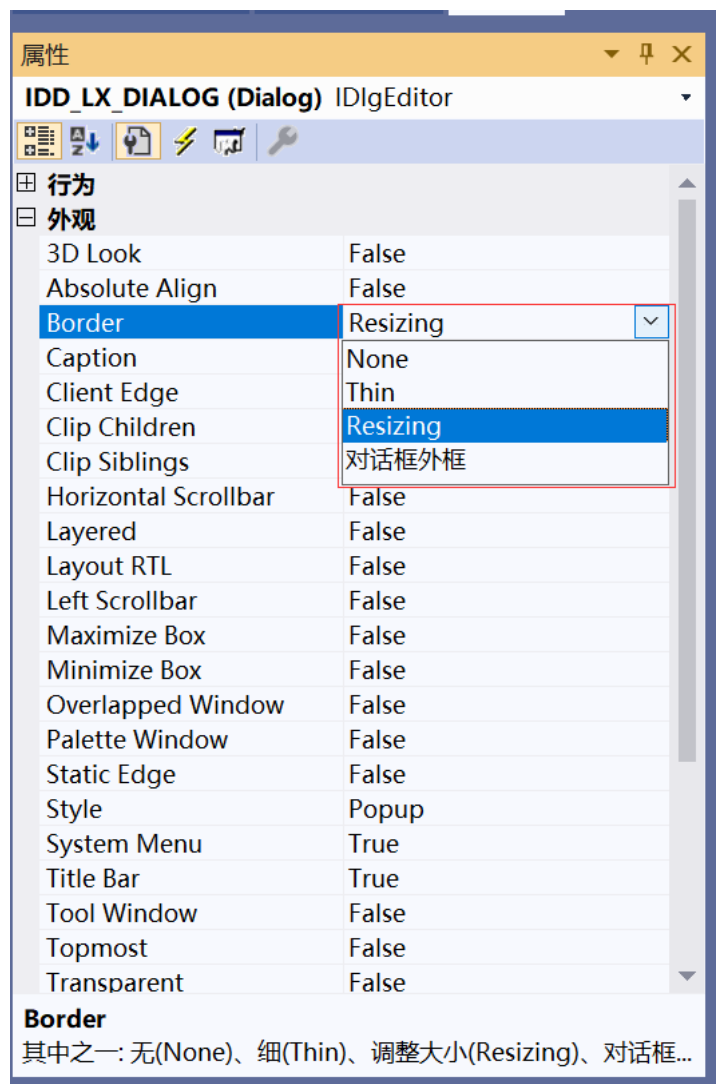


常用的对话框属性

- ID：ID是应用程序用来唯一标识该对话框资源的标识符，通常使用默认值，也可以对它进行修改。
- Caption：对话框的标题，即对话框窗口标题栏显示的文字。默认值为项目名称，通常需要改为编程者认为合适的文字。
- Font：在“属性窗口”中该属性的属性值编辑框后有一按钮，单击此按钮可弹出字体对话框，通过该对话框可选择对话框窗口中控件上显示的文字的字体、字形和字号。



- Border：用于确定窗口的边框类型，有四个值可选，None无边框，不显示标题栏；Thin细边框，也不显示标题栏；Dialog Fram，对话框边框，程序运行时窗口大小不可改变；Resizing，可调整尺寸，程序运行时对话框窗口的大小可用鼠标拖曳调整。
- System menu：该属性用于指定是否为对话框创建系统菜单。该属性值为 Bool类型，缺省值为True。
- Minimize box /Maximize box：指定对话框是否有最小化/最大化按钮，这两个属性值均为 Bool类型，缺省值为False。当对话框无标题栏时，即Border属性值为None或Thin时，这两个属性无效。



3.5 Windows 控件

- 一个完整的控件包括两部分，一是控件资源，即在其父窗口上绘出的控件的图形，另一部分则是封装有控件属性和方法的控件类。
- MFC提供了众多的控件类，每一个类都封装一种控件。在对话框中创建控件后，需要为控件定义一个对应控件类的变量（对象）对它进行控制。

3.5.1 创建控件

- 控件的创建有静态创建和动态创建两种。
- 静态创建是指使用对话框模板创建控件，并通过属性窗口设置控件的属性，当打开对话框时，系统将自动按预设的属性创建控件。
- 动态创建则是指在程序的运行中根据需要，通过预先定义的控件类对象调用CreateWindow（）函数创建控件。这里我们只简单介绍控件的静态创建方法。

- 第一步是打开“对话框资源编辑器”和“工具箱”。
- 第二步，在“工具箱”中单击选中要创建的控件，然后将鼠标移到“对话框资源编辑器”中的对话框上，按下鼠标左键进行拖曳，在相应位置画出控件。
- 第三步，选中画好的控件，通过鼠标拖曳调整控件的大小和位置。
- 第四步，使用“属性窗口”设置控件的属性。

文件(F) 编辑(E) 视图(V) 项目(P) 生成(B) 调试(D) 格式(O) 测试(S) 分析(N) 工具(T) 扩展(X) 窗口(W) 帮助(H)

搜索 Visual Studio (Ctrl+Q)

1

Live Share

Debug x86 本地 Windows 调试器

工具箱

搜索工具箱

对话框编辑器

指针

Button

Check Box

ab Edit Control

Combo Box

List Box

Group Box

Radio Button

Static Text

Picture Control

Horizontal Scroll Bar

Vertical Scroll Bar

Slider Control

Spin Control

Progress Control

Hot Key

List Control

Tree Control

Tab Control

Animation Control

Rich Edit 2.0 Control

Date Time Picker

Month Calendar Co...

IP Address Control

Extended Combo Box

Custom Control

SysLink Control

Split Button Control

Network Address Co...

Command Button C...

MFC Button Control

MFC ColorButton Co...

MFC EditBrowse Co...

MFC VListBox Cont...

MFC FontComboBo...

lx.rc - String Table*

lx.rc - IDD_LX_DIALOG - Dialog*

lxDlg.cpp

lxDlg.h

这是一个对话框示例

你好, 世界

确定 取消

原型图像: 透明度: 50% 偏移量 X: 0 Y: 0

输出

显示输出来源(S):

资源视图 - lx

lx

lx.rc*

Dialog

IDD_ABOUTBOX

IDD_LX_DIALOG

Icon

String Table

String Table

Version

属性

IDC_STATIC2 (Text Control) IStatEditor

动态布局

行为

外观

Align Text Left

Border False

Caption 你好, 世界

Center Image False

Client Edge False

End Ellipsis False

Modal Frame False

No Prefix False

No Wrap False

Notify False

Path Ellipsis False

Real Size Control False

Right Align Text False

Right To Left Reading Orc False

Simple False

Static Edge False

Sunken False

Transparent False

Word Ellipsis False

杂项

Caption

指定由控件显示的文本。

服务器资源管理器 工具箱

错误列表 输出

57

27, 22 122 x 22

添加到源代码管理

设置控件属性

- 每个控件都有一个属性集，通过设置控件的属性可以控制控件的外观和行为等。对于静态创建的控件，可以在对话框模板中打开控件的属性对话框直接设置控件的初始属性值，也可以通过程序代码动态设置控件属性值。
- 控件的属性往往有很多，但编程时并不是每个属性的值都需要设置，多数情况下使用默认值就可以了。不同类型的控件有不同的属性，但是也有一些属性是共有的。

控件3个常用的共有属性

属性	功能
ID	程序通过控件 ID 来访问一个控件。控件创建后系统会自动生成一个ID，可修改。除 静态控件 外，同一程序中的控件的 ID 必须互不相同。
Visible	该属性决定程序运行时控件是否可见。属性值是布尔型，默认为 TRUE 。
Disabled	设置当对话框在打开时该控件是否不可用，也是布尔类型，默认为 FALSE 。

- 静态控件是指一组控件类型，包括静态文本（ Static Text ）控件、图片控件（ Picture Control ）和组框（ Group Box ），这些控件主要用来显示文本或图形信息。
- 同一应用程序中的多个静态控件可以使用相同的ID，通常是它们的默认ID——IDC_STATIC。如果需要在程序中区分和操纵各个不同的静态控件，必须重新为它们分别指定一个唯一的ID。

3.5.2 常用控件

- 静态文本（ Static Text ）控件：要用来显示文字，要显示的文本串为该控件的属性Caption的属性值。该属性值可以静态修改，也可以在程序中用赋值语句动态修改。
- 编辑控件（ Edit Control ）：又称编辑框，通常与静态文本控件一起使用，用于数据的输入和输出。通过设置不同的属性值，编辑控件可以表现为多行编辑框、密码编辑框、只读编辑框等多种不同类型的样式。

- 命令按钮（ Button ）：命令按钮在被按下时会立即执行某个命令。该命令所执行的操作，是由命令按钮的消息处理函数规定的。命令按钮上显示的文字是由Caption属性指定的，通过修改Caption属性的值可改变命令按钮上显示的文字。
- 列表框（ List Box ）：常用来显示类型相同的一系列清单，如文件、字体和用户等，适用于从若干数据项中进行选择的场合。列表框是一个矩形窗口，包含若干列表项（每项为一个字符串）。

控件名称	MFC类	功能描述
静态控件	CStatic	用来显示一些几乎固定不变的文字或图形
按钮	CButton	用来产生某些命令或改变某些选项，包括单选按钮、复选框和组框
编辑框	CEdit	用于完成文本和数字的输入和编辑
列表框	CListBox	显示一个列表，让用户从中选取一个或多个项
组合框	CComboBox	是一个列表框和编辑框组合的控件
滚动条	CScrollBar	通过滚动块在滚动条上的移动和滚动按钮来改变某些量
进展条	CProgressCtrl	用来表示一个操作的进度
滑动条	CSliderCtrl	通过滑动块的移动来改变某些量，并带有刻度指示
日期时间控件	CDateTimeCtrl	用于选择指定的日期和时间
图像列表	CImageList	一个具有相同大小的图标或位图的集合
标签控件	CTabCtrl	类似于一个文件柜上的标签，使用它可以将一个窗口或对话框的相同区域定义为多个页面

3.6 Windows的消息驱动机制与消息映射

- Windows程序是通过操作系统发送的消息来处理用户输入的。所有Windows应用程序都是消息驱动的，消息处理是所有Windows应用程序的核心部分。
- 无论是系统产生的动作或是运行应用程序产生的动作，都称为事件(Events)产生的消息(Message)。
- 在应用程序中，通过接收消息、分发消息、处理消息来和用户进行交互。
- 系统中许多消息都经过了严格的定义，并且适用于所有的应用程序。

3.6.1 Windows的消息驱动机制

- 当用户单击鼠标、敲击键盘或调整窗口大小时，都将向对应的窗口发送消息。每个消息都对应于某个特定的事件，比如单击鼠标事件、双击鼠标事件、鼠标移动事件等。
- Windows系统将应用程序的输入事件转换为消息，并将消息发送给应用程序的窗口，这些窗口通过调用消息处理函数来接收和处理这些消息。
- 消息处理函数通常是窗口类的成员函数，编写消息处理函数是编写MFC应用程序的主要任务。

- Windows消息大致可分为标准Windows消息、控件通知和命令消息等类型。
- 标准Windows消息通常由窗口类处理，也就是说，这类消息的处理函数通常是MFC窗口类的成员函数。
 - 除WM_COMMAND消息外，所有以“WM_”为前缀的消息都是标准Windows消息。
 - 标准Windows消息都有默认的处理函数，这些函数是CWnd类的成员函数，函数的名称都是“On”开头，后跟去掉“WM_”前缀的消息名称。

- 命令消息包括来自菜单选项、工具栏按钮、快捷键等用户接口界面的WM_COMMAND通知消息，属于用户应用程序自己定义的消息。
 - 通过消息映射机制，MFC框架把命令消息按一定的路径分发给多种类型的对象来处理，这些对象包括文档、文档模板、应用程序对象、以及窗口和视图等。
 - 能处理消息映射的类必定是从MFC的CCmdTarget类派生的。

- 控件通知是由控件传给父窗口的消息，但也有一个例外，当用户单击“命令按钮”按钮控件时，发出的BN_CLICKED消息将作为命令消息来处理。
 - 控件消息通常也是由MFC窗口类处理的。

3.6.2 消息映射

- 应用程序运行之后，当控件的状态发生改变时，控件就会向其父窗口发送消息，这个消息称为“控件通知消息”。
- Windows是通过调用相应消息处理函数来处理每一条消息的，编程时，如何让一条消息与其消息处理函数建立关联关系呢？答案是通过消息映射。

实现消息映射

- MFC是通过“类向导”帮助实现消息映射的
 - 类向导会在类的定义（头文件）里增加消息处理函数声明，并添加一行声明消息映射的宏
 `DECLARE_MESSAGE_MAP。`
 - 还会自动在类的实现文件（CPP文件）中添加消息映射的内容，并添加消息处理函数的实现框架（具体功能代码需要由编程者自己编写）。
 - 一般情况下，消息处理函数的声明和实现框架均是由MFC的类向导自动添加和维护的。

文件(F) 编辑(E) 视图(V) 项目(P) 生成(B) 调试(D) 格式(O) 测试(S) 分析(N) 工具(T) 扩展(X) 窗口(W) 帮助(H)

搜索 Visual Studio (Ctrl+Q)

1

Live Share

工具箱

搜索工具箱

对话框编辑器

指针

Button

Check Box

abEdit Control

Combo Box

List Box

Group Box

Radio Button

Static Text

Picture Control

Horizontal Scroll Bar

Vertical Scroll Bar

Slider Control

Spin Control

Progress Control

Hot Key

List Control

Tree Control

Tab Control

Animation Control

Rich Edit 2.0 Control

Date Time Picker

Month Calendar Co...

IP Address Control

Extended Combo Box

Custom Control

SysLink Control

Split Button Control

Network Address Co...

Command Button C...

MFC Button Control

MFC ColorButton Co...

MFC EditBrowse Co...

MFC VSListBox Cont...

MFC FontComboBo...

lx.rc - VS_VERSION_INFO - Version

lx.rc - String Table

lx.rc - IDD_LX_DIALOG - Dialog

lxDlg.cpp

lxDlg.h

资源视图 - lx

lx

lx.rc

Dialog

IDD_ABOUTBOX

IDD_LX_DIALOG

Icon

String Table

String Table

Version

VS_VERSION_INFO

解决方案资源管理器 团队资源管理器 资源视图

属性

IDC_MOD_LABEL (Button Control) IButtonEditor

动态布局

行为

外观

杂项

输出

显示输出来源(S): 生成

1>lx.vcxproj -> D:\C\lx\Debug\lx.exe

===== 生成: 成功 1 个, 失败 0 个, 最新 0 个, 跳过 0 个 =====

错误列表 输出

这是一个对话框示例

你好, 世界

示例编辑框

示例编辑框

更改

确定

取消

双击

服务器资源管理器 工具箱

就緒

219, 117

50 x 14

添加到源代码管理

lx.rc - VS_VERSION_INFO - Versionlx.rc - String Tablelx.rc - IDD_LX_DIALOG - DialoglxDlg.cpplxDlg.h

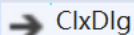
lxClxDlg

```
4
5  #pragma once
6
7
8  // ClxDlg 对话框
9  class ClxDlg : public CDialogEx
10 {
11 // 构造
12 public:
13     ClxDlg(CWnd* pParent = nullptr);    // 标准构造函数
14
15 // 对话框数据
16 #ifdef AFX_DESIGN_TIME
17     enum { IDD = IDD_LX_DIALOG };
18 #endif
19
20 protected:
21     virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV 支持
22
23
24 // 实现
25 protected:
26     HICON m_hIcon;
27
28     // 生成的消息映射函数
29     virtual BOOL OnInitDialog();
30     afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
31     afx_msg void OnPaint();
32     afx_msg HCURSOR OnQueryDragIcon();
33     DECLARE_MESSAGE_MAP()
34 public:
35     afx_msg void OnBnClickedModLabel();
36 };
37
```

100 % 未找到相关问题



lx

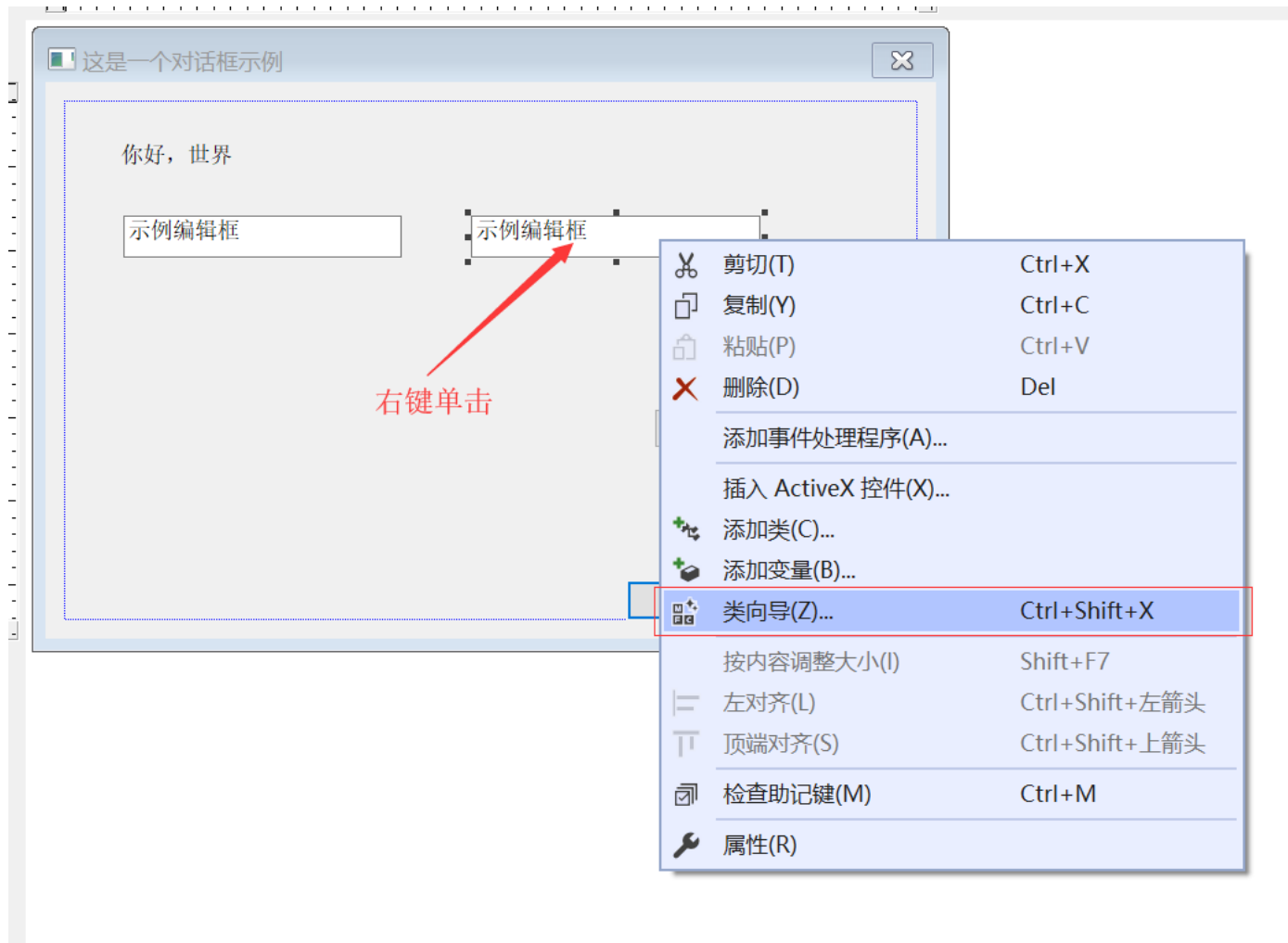


ClxDlg



OnBnClickedModLabel()

```
60 {
61     CDialogEx::DoDataExchange(pDX);
62 }
63
64 BEGIN_MESSAGE_MAP(ClxDlg, CDialogEx)
65     ON_WM_SYSCOMMAND()
66     ON_WM_PAINT()
67     ON_WM_QUERYDRAGICON()
68     ON_BN_CLICKED(IDC_MOD_LABEL, &ClxDlg::OnBnClickedModLabel)
69 END_MESSAGE_MAP()
70
71 // ClxDlg 消息处理程序
72
73
74 BOOL ClxDlg::OnInitDialog() { ... }
75
76 void ClxDlg::OnSysCommand(UINT nID, LPARAM lParam) { ... }
77
78 // ...
79
80 void ClxDlg::OnPaint() { ... }
81
82 //当用户拖动最小化窗口时系统调用此函数取得光标
83 //显示。
84 HCURSOR ClxDlg::OnQueryDragIcon() { ... }
85
86
87 void ClxDlg::OnBnClickedModLabel()
88 {
89     // TODO: 在此添加控件通知处理程序代码
90 }
```





欢迎使用类向导

项目(P): lx 类名(N): ClxDlg 添加类(C)...

基类: CDialogEx 类声明(I): lxDlg.h

资源: IDD_LX_DIALOG 类实现(L): lxDlg.cpp

命令 消息 虚函数 成员变量 方法

搜索命令

对象 ID(B):

- ID_WIZBACK
- ID_WIZFINISH
- ID_WIZNEXT
- IDABORT
- IDC_EDIT1
- IDC_EDIT2
- IDC_MOD_LABEL**
- IDC_STATIC
- IDCANCEL

消息(S):

- EN_CHANGE**
- EN_ERRSPACE
- EN_HSCROLL
- EN_KILLFOCUS
- EN_MAXTEXT
- EN_SETFOCUS
- EN_UPDATE
- EN_VSCROLL
- EN_ALIGN_LTR_EC

成员函数(M):

函数名	命令 ID	消息
OnBnClickedModLabel	IDC_MOD_LABEL	BN_CLICKED

说明:

添加成员函数

成员函数名称(E):

OnChangeEdit2

消息: EN_CHANGE

对象 ID: IDC_EDIT2

确定

取消

添加处理程序(A)...

删除处理程序(D)

编辑代码(E)

已映射的消息

确定

75

取消

应用

```
lx.rc - VS_VERSION_INFO - Version | lx.rc - IDD_LX_DIALOG - Dialog | lxDlg.cpp | lxDlg.h | lx.rc - String Table
lx
4
5     #pragma once
6
7
8     // ClxDlg 对话框
9     class ClxDlg : public CDialogEx
10    {
11    // 构造
12    public:
13        ClxDlg(CWnd* pParent = nullptr);    // 标准构造函数
14
15    // 对话框数据
16    #ifdef AFX_DESIGN_TIME
17        enum { IDD = IDD_LX_DIALOG };
18    #endif
19
20    protected:
21        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV 支持
22
23
24    // 实现
25    protected:
26        HICON m_hIcon;
27
28        // 生成的消息映射函数
29        virtual BOOL OnInitDialog();
30        afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
31        afx_msg void OnPaint();
32        afx_msg HCURSOR OnQueryDragIcon();
33        DECLARE_MESSAGE_MAP()
34    public:
35        afx_msg void OnBnClickedModLabel();
36        afx_msg void OnChangeEdit2();
37    };
38
```

lx.rc - VS_VERSION_INFO - Versionlx.rc - IDD_LX_DIALOG - DialoglxDlg.cpplxDlg.hlx.rc - String Table

lxClxDlgOnChangeEdit2()

```
63
64 BEGIN_MESSAGE_MAP(ClxDlg, CDialogEx)
65     ON_WM_SYSCOMMAND()
66     ON_WM_PAINT()
67     ON_WM_QUERYDRAGICON()
68     ON_BN_CLICKED(IDC_MOD_LABEL, &ClxDlg::OnBnClickedModLabel)
69     ON_EN_CHANGE(IDC_EDIT2, &ClxDlg::OnChangeEdit2)
70 END_MESSAGE_MAP()
71
72
73 // ClxDlg 消息处理程序
74
75 BOOL ClxDlg::OnInitDialog() { ... }
108
109 void ClxDlg::OnSysCommand(UINT nID, LPARAM lParam) { ... }
121
122 // ...
125
126 void ClxDlg::OnPaint() { ... }
150
151 //当用户拖动最小化窗口时系统调用此函数取得光标
152 //显示。
153 HCURSOR ClxDlg::OnQueryDragIcon() { ... }
157
158
159
160 void ClxDlg::OnBnClickedModLabel() { ... }
164
165
166 void ClxDlg::OnChangeEdit2()
167 {
168     // TODO: 如果该控件是 RICHEDIT 控件，它将不
169     // 发送此通知，除非重写 CDialogEx::OnInitDialog()
170     // 函数并调用 CRichEditCtrl().SetEventMask(),
171     // 同时将 ENM_CHANGE 标志“或”运算到掩码中。
172 }
```

100 % 未找到相关问题

CDialog类对消息映射宏的处理

```
BEGIN_MESSAGE_MAP(CDialog, CWnd)
#ifdef _AFX_NO_GRAYDLG_SUPPORT
    ON_WM_CTLCOLOR()
#endif
//{{AFX_MSG_MAP(CDialog)
    ON_COMMAND(IDOK, OnOK)
    ON_COMMAND(IDCANCEL, OnCancel)
    ON_MESSAGE(WM_COMMANDHELP, OnCommandHelp)
    ON_MESSAGE(WM_HELPHITTEST, OnHelpHitTest)
    ON_MESSAGE(WM_INITDIALOG, HandleInitDialog)
    ON_MESSAGE(WM_SETFONT, HandleSetFont)
//}}AFX_MSG_MAP
#ifdef _AFX_NO_CTL3D_SUPPORT
    ON_MESSAGE(WM_QUERY3DCONTROLS, OnQuery3dControls)
#endif
END_MESSAGE_MAP()
```

由于Cdialog已经预处理一些ID的命令消息的映射，
对这些ID如果没有额外要处理的内容，可以不用再映射

退出对话框

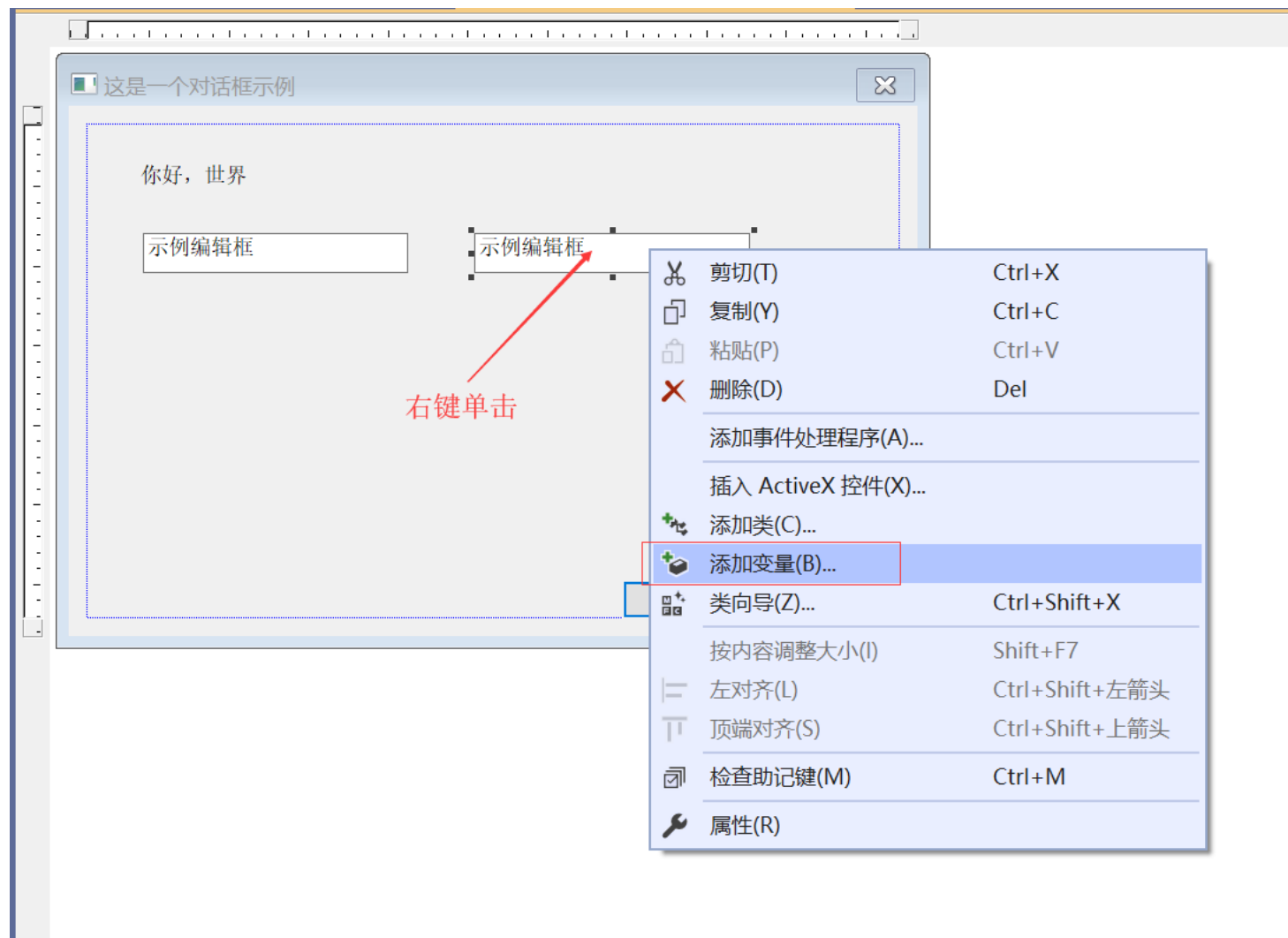
- **给对话框加一个关闭按钮，但是不用IDOK和IDCANCEL，怎么做？**
- 在消息映射函数中调用CDialog::OnOK();

3.7 使用控件变量访问控制控件

- 控件的主要功能就是完成输入输出，也就是使用户可以与程序交流信息；
- 对话框中某个控件上输入的数据可以保存到程序中的某个变量中，程序中某个变量的值也可以在对话框中的某个控件上输出显示，就是所谓**控件与变量的数据交换**。
- 对话框内的控件使用基于ID的变量映射来实现数据交换。

- 通过类向导，我们可以将一个变量和一个控件进行关联（映射）来实现数据交换，并且可设置其数据范围。
- 与控件关联的变量通常都被称为“控件变量”。
- 变量的类别对命令按钮来说只有“Control”一种，但对于文本框则会有“Control”和“Value”两种。
- “Control”类别的控件变量在程序中代表控件本身，通过该类变量可直接引用该控件类的各种方法，并可以对控件的各种属性直接操作。
- 只有那些用于输入数据和输出数据的控件才能关联“Value”类别的控件变量，该类控件变量只用于存放控件输出或输入的数据。

- 对 “Control” 类别的控件变量，其变量类型只有一种，即该控件对应的控件类。
- 而 “Value” 类别的变量类型则可以是任何该控件所能输入或输出的数据类型，对文本框而言，数值类型可以是CString、int、UINT、long、DWORD、float、double、BYTE、short、BOOL等。



×

添加控件变量

常规设置

控件

其他

控件 ID(I)

IDC_EDIT2

控件类型(Y)

EDIT

类别(I)

值

名称(N)

strEdit2

访问(A)

public

变量类型(V)

CString

注释(M)

选择控件或值

变量命名要容易理解

上一步

下一步

完成

取消



lx



ClxDlg

```
4
5     #pragma once
6
7
8     // ClxDlg 对话框
9     class ClxDlg : public CDialogEx
10    {
11    // 构造
12    public:
13        ClxDlg(CWnd* pParent = nullptr);    // 标准构造函数
14
15    // 对话框数据
16    #ifdef AFX_DESIGN_TIME
17        enum { IDD = IDD_LX_DIALOG };
18    #endif
19
20    protected:
21        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV 支持
22
23
24    // 实现
25    protected:
26        HICON m_hIcon;
27
28        // 生成的消息映射函数
29        virtual BOOL OnInitDialog();
30        afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
31        afx_msg void OnPaint();
32        afx_msg HCURSOR OnQueryDragIcon();
33        DECLARE_MESSAGE_MAP()
34    public:
35        afx_msg void OnBnClickedModLabel1();
36        afx_msg void OnChangeEdit2();
37        CString strEdit2;
38    };
39
```

lx.rc - VS_VERSION_INFO - Versionlx.rc - IDD_LX_DIALOG - DialoglxDlg.cpplxDlg.hlx.rc - String Table

lxClxDlgOnChangeEdit2()

```
44
45     BEGIN_MESSAGE_MAP(CAboutDlg, CDialogEx)
46     END_MESSAGE_MAP()
47
48     // ClxDlg 对话框
49
50
51
52
53     ClxDlg::ClxDlg(CWnd* pParent /*=nullptr*/)
54     : CDialogEx(IDD_LX_DIALOG, pParent)
55     , strEdit2(_T(""))
56     {
57         m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
58     }
59
60     void ClxDlg::DoDataExchange(CDataExchange* pDX)
61     {
62         CDialogEx::DoDataExchange(pDX);
63         DDX_Text(pDX, IDC_EDIT2, strEdit2);
64     }
65
66     BEGIN_MESSAGE_MAP(ClxDlg, CDialogEx)
67     ON_WM_SYSCOMMAND()
68     ON_WM_PAINT()
69     ON_WM_QUERYDRAGICON()
70     ON_BN_CLICKED(IDC_MOD_LABEL, &ClxDlg::OnBnClickedModLabel)
71     ON_EN_CHANGE(IDC_EDIT2, &ClxDlg::OnChangeEdit2)
72     END_MESSAGE_MAP()
73
74     // ClxDlg 消息处理程序
75
76
77     BOOL ClxDlg::OnInitDialog() { ... }
110
111     void ClxDlg::OnSysCommand(UINT nID, LPARAM lParam) { ... }
```

初始化

变量与控件绑定

100 % 未找到相关问题

×

添加控件变量

常规设置

控件

其他

控件 ID(I)

IDC_EDIT2

类别(I)

控件

访问(A)

public

控件类型(Y)

EDIT

名称(N)

edit2

变量类型(V)

CEdit

注释(M)

上一步

下一步

完成

取消

```
lxDlg.h  lx.cpp  lx.rc - IDD_LX_DIALOG - Dialog  framework.h  lxDlg.c
lx  ClxDlg

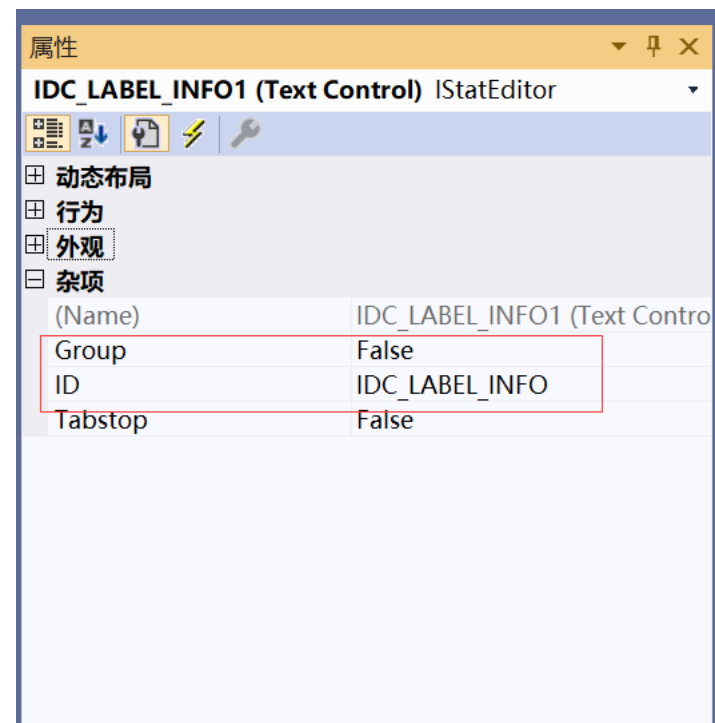
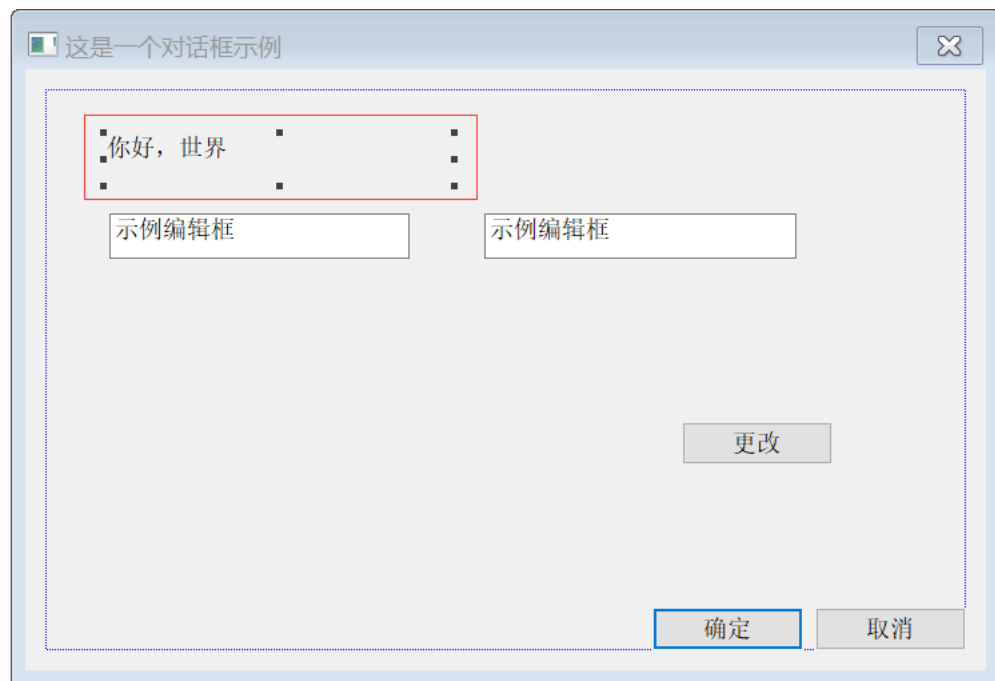
25  protected:
26      HICON m_hIcon;
27
28      // 生成的消息映射函数
29      virtual BOOL OnInitDialog();
30      afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
31      afx_msg void OnPaint();
32      afx_msg HCURSOR OnQueryDragIcon();
33      DECLARE_MESSAGE_MAP()
34  public:
35      afx_msg void OnBnClickedModLabel1();
36      afx_msg void OnChangeEdit2();
37      CString strEdit2;
38      CString strLabelInfo;
39      CEdit edit2;
40  };
41
```



```
lxDlg.h  lx.cpp  lx.rc - IDD_LX_DIALOG - Dialog  framework.h  lxDlg.cpp  DoDataExchange

lx  ClxDlg  DoDataExchange

50
51
52
53  CClxDlg::CClxDlg(CWnd* pParent /*=nullptr*/)
54      : CDialogEx(IDD_LX_DIALOG, pParent)
55      , strEdit2(_T(""))
56      , strLabelInfo(_T(""))
57      {
58      m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
59      }
60
61  void CClxDlg::DoDataExchange(CDataExchange* pDX)
62      {
63      CDialogEx::DoDataExchange(pDX);
64      DDX_Text(pDX, IDC_EDIT2, strEdit2);
65      DDX_Text(pDX, IDC_LABEL_INFO, strLabelInfo);
66      DDX_Control(pDX, IDC_EDIT2, edit2);
67      }
68
69  BEGIN_MESSAGE_MAP(CClxDlg, CDialogEx)
70      ON_WM_SYSCOMMAND()
71      ON_WM_PAINT()
72      ON_WM_QUERYDRAGICON()
73      ON_BN_CLICKED(IDC_MOD_LABEL, &CClxDlg::OnBnClickedModLabel)
74      ON_EN_CHANGE(IDC_EDIT2, &CClxDlg::OnChangeEdit2)
75  END_MESSAGE_MAP()
76
```



要为静态控件映射变量，需要把Group设置为false并设置ID

×

添加控件变量

常规设置

控件

其他

控件 ID(I)

IDC_LABEL_INFO

控件类型(Y)

LTEXT

类别(I)

值

名称(N)

strLabelInfo

访问(A)

public

变量类型(V)

CString

注释(M)

上一步

下一步

完成

取消

- 当为一个控件添加了一个Value类别的控件变量后，可以使用CWnd::UpdateData函数实现该控件变量与控件件的数据交换。该函数是CWnd类的成员函数，其原型如下：

`BOOL UpdateData(BOOL bSaveAndValidate = TRUE);`

- 该函数只有一个BOOL类型的参数，调用UpdateData(FALSE)时，数据由控件变量向控件传输，当调用UpdateData(TRUE)或不带参数的UpdateData时，数据从控件向相关联的成员变量复制。
- 函数执行成功则返回TRUE，否则返回FALSE。

- 如果为一个控件添加了一个“Control”类别的控件变量，也可以使用该“Control”类型的变量调用控件类从CWnd类继承来的成员函数SetWindowText和GetWindowText来改变或获取控件显示的文字。

```
void C1xDlg::OnChangeEdit2()
{
    // TODO: 如果该控件是 RICHEDIT 控件，它将不
    // 发送此通知，除非重写 CDialogEx::OnInitDialog()
    // 函数并调用 CRichEditCtrl().SetEventMask()，
    // 同时将 ENM_CHANGE 标志“或”运算到掩码中。

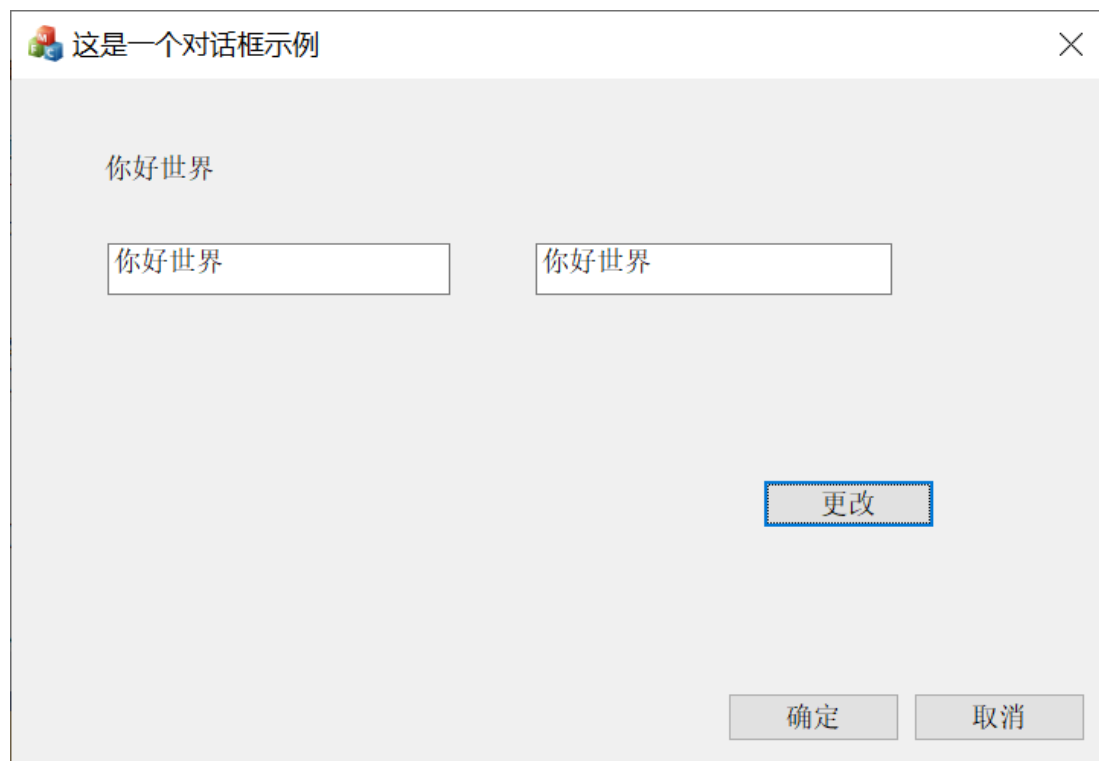
    // TODO: 在此添加控件通知处理程序代码
    UpdateData(true); // 控件更新数据到变量
    strLabelInfo = strEdit2;
    UpdateData(false); // 变量更新数据到控件
}
```

```
void C1xDlg::OnBnClickedModLabel()
{
    // TODO: 在此添加控件通知处理程序代码
    LPCTSTR str1 = _T("你好世界");
    edit2.SetWindowTextW(str1);
}
```

通过控件变量来对控件进行设置

```
void C1xDlg::OnBnClickedModLabel1()
{
    // TODO: 在此添加控件通知处理程序代码
    LPCTSTR str1 = _T("你好世界");
    edit2.SetWindowTextW(str1);
    //根据ID获取控件
    CEdit* pEdit1;
    pEdit1 = (CEdit*)GetDlgItem(IDC_EDIT1);
    pEdit1->SetWindowTextW(str1);
}
```

也可以用CWnd::GetDlgItem方法来由ID获取控件进行操作



执行效果

3.8 添加用户自定义消息

- Visual C++ 允许用户自己定义并发送消息，对自定义的消息用户也必须为其添加消息处理函数。
- 第一步，在 resource.h 或 stdafx.h(现改名为pch.h) 文件深红添加如下代码定义一个自己的消息：

```
#define WM_MY_MESSAGE WM_USER + 100
```

- 其中WM_USER是为了防止用户定义的消息ID与系统的消息ID冲突 Visual C++ 提供的一个宏，小于WM_USER的ID被系统使用，大于WM_USER的ID才可以被用户使用。因此自定义消息一般是WM_USER+XXX的形式，XXX表示任意一个正整数。

- 然后，打开“MFC类向导”，选择要处理消息的对话框类，选中“消息”选项卡。单击选项卡下部的“添加自定义消息”按钮，弹出“添加自定义消息”对话框。
- 分别填入自定义消息和消息处理函数的名称，自定义消息一定是第一步中定义的消息，输入自定义消息后，向导会自动生成一个缺省的消息处理函数名，一般直接采用这个缺省的消息处理函数名则可，当然也可以重新命名该函数。
- 单击确定返回“MFC类向导”，这时在现有处理程序一列发现刚添加的自定义消息及消息处理函数。

- 除了函数实现代码外，类向导还在消息处理函数所在的类的头文件中添加了如下的消息处理函数的声明代码：

```
afx_msg LRESULT OnMyMessage(WPARAM wParam, LPARAM lParam);
```

- 在消息处理函数所在的类的cpp文件中的消息映射添加类似下面的消息映射代码：

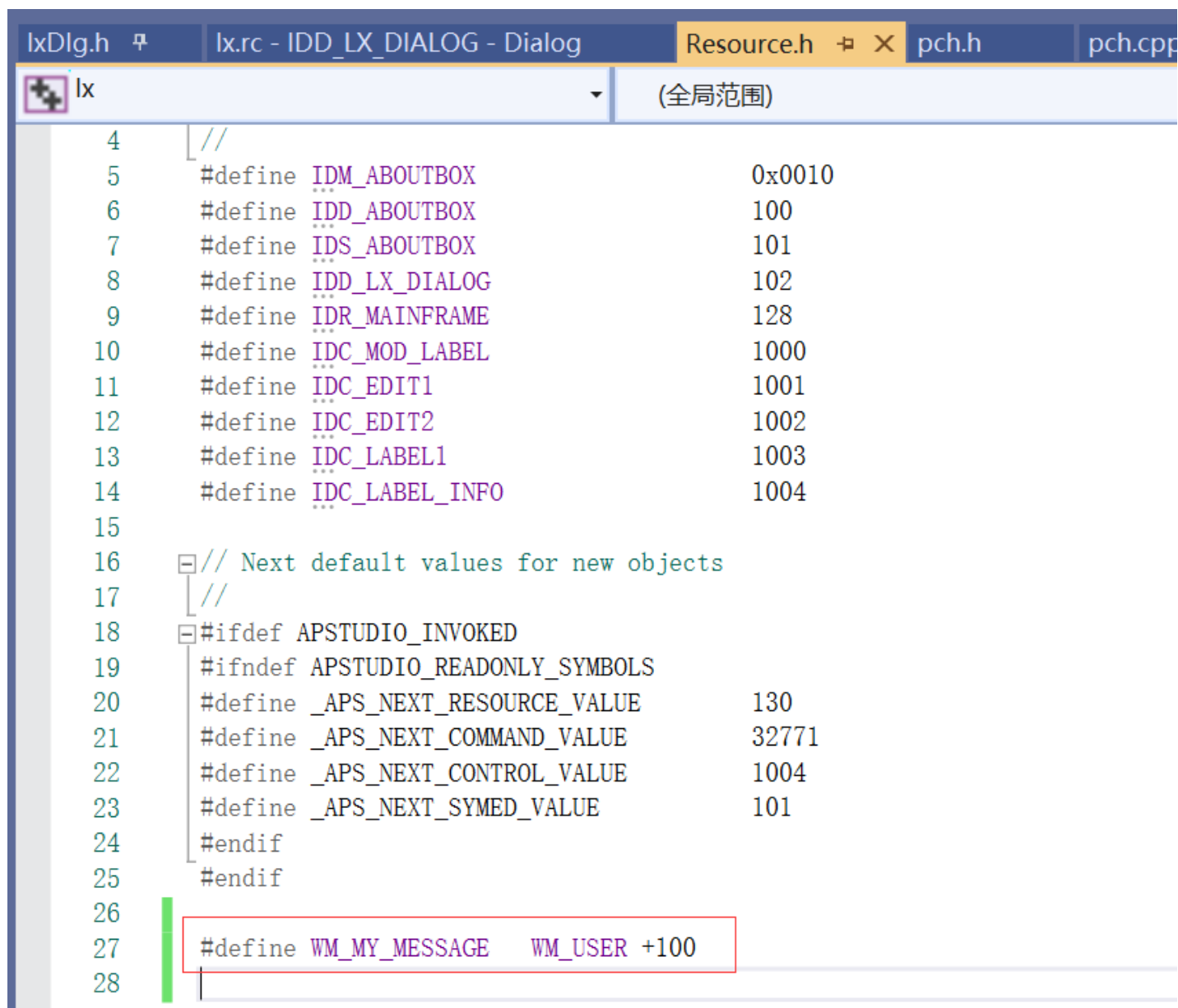
```
BEGIN_MESSAGE_MAP(CDlg, CDialogEx)
```

```
...
```

```
...
```

```
ON_MESSAGE(WM_MY_MESSAGE, &CDlg::OnMyMessage)
```

```
END_MESSAGE_MAP()
```



```
4 //
5 #define IDM_ABOUTBOX 0x0010
6 #define IDD_ABOUTBOX 100
7 #define IDS_ABOUTBOX 101
8 #define IDD_LX_DIALOG 102
9 #define IDR_MAINFRAME 128
10 #define IDC_MOD_LABEL 1000
11 #define IDC_EDIT1 1001
12 #define IDC_EDIT2 1002
13 #define IDC_LABEL1 1003
14 #define IDC_LABEL_INFO 1004
15
16 // Next default values for new objects
17 //
18 #ifdef APSTUDIO_INVOKED
19 #ifndef APSTUDIO_READONLY_SYMBOLS
20 #define _APS_NEXT_RESOURCE_VALUE 130
21 #define _APS_NEXT_COMMAND_VALUE 32771
22 #define _APS_NEXT_CONTROL_VALUE 1004
23 #define _APS_NEXT_SYMED_VALUE 101
24 #endif
25 #endif
26
27 #define WM_MY_MESSAGE WM_USER +100
28
```



欢迎使用类向导

项目(P):

lx

类名(N):

ClxDlg

添加类(C)...

▼

基类:

CDialogEx

类声明(I):

lxDlg.h

▼

资源:

IDD_LX_DIALOG

类实现(L):

lxDlg.cpp

▼

命令

消息

虚函数

成员变量

方法

搜索消息

搜索处理程序

消息(S):

WM_NCMBUTTONDOWN
WM_NCMBUTTONUP
WM_NCMOUSEHOVER
WM_NCMOUSELEAVE
WM_NCMOUSEMOVE
WM_NCPAINT
WM_NCRBUTTONDBLCLK
WM_NCRBUTTONDOWN
WM_NCRBUTTONUP
WM_NCXBUTTONDBLCLK
WM_NCXBUTTONDOWN
WM_NCXBUTTONUP
WM_NEXTMENU
WM_NOTIFYFORMAT
WM_PAINT

现有处理程序(H):

函数名	消息
OnPaint	WM_PAINT
OnQueryDragIcon	WM_QUERYDR...
OnSysCommand	WM_SYSCOMM...

添加处理程序(A)

删除处理程序(D)

编辑代码(E)

添加自定义消息(M)...

说明: 指示窗口框架需要绘制(不用于视图)

添加自定义消息

自定义 Windows 消息(C):

WM_MY_MESSAGE

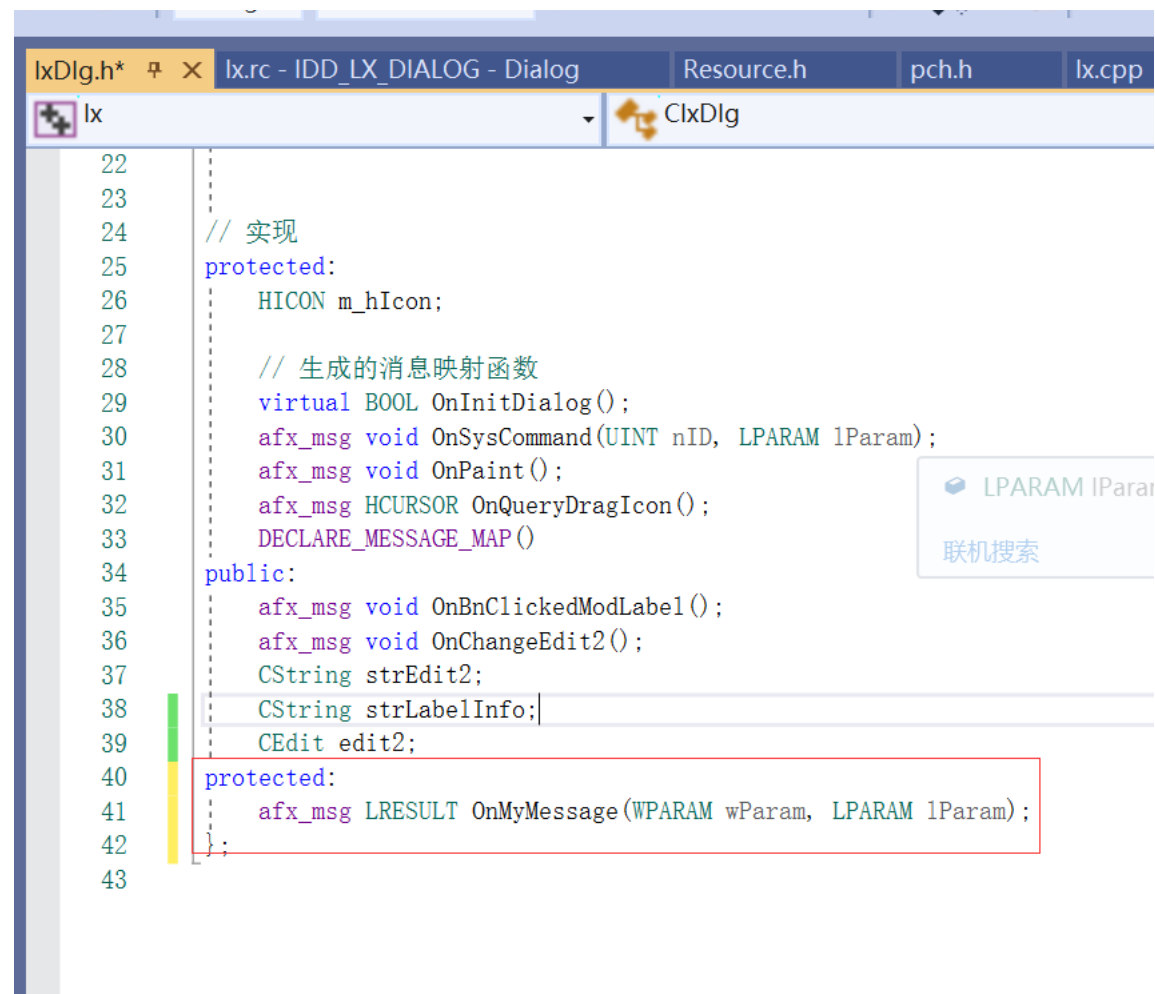
消息处理程序名称(M):

OnMyMessage

☐ 已注册的消息(R)

确定

取消



```
22
23
24 // 实现
25 protected:
26     HICON m_hIcon;
27
28     // 生成的消息映射函数
29     virtual BOOL OnInitDialog();
30     afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
31     afx_msg void OnPaint();
32     afx_msg HCURSOR OnQueryDragIcon();
33     DECLARE_MESSAGE_MAP()
34 public:
35     afx_msg void OnBnClickedModLabel1();
36     afx_msg void OnChangeEdit2();
37     CString strEdit2;
38     CString strLabelInfo;
39     CEdit edit2;
40 protected:
41     afx_msg LRESULT OnMyMessage(WPARAM wParam, LPARAM lParam);
42 };
43
```

LPARAM lParam

联机搜索

```
lxDlg.h  lx.rc - IDD_LX_DIALOG - Dialog  Resource.h  pch.h  lx.cpp  framework.h  lxDlg.cpp*  X
lx  ClxDlg  OnBnClickedModLabel()
62  {
63      CDialogEx::DoDataExchange(pDX);
64      DDX_Text(pDX, IDC_EDIT2, strEdit2);
65      DDX_Text(pDX, IDC_LABEL_INFO, strLabelInfo);
66      DDX_Control(pDX, IDC_EDIT2, edit2);
67  }
68
69  BEGIN_MESSAGE_MAP(ClxDlg, CDialogEx)
70      ON_WM_SYSCOMMAND()
71      ON_WM_PAINT()
72      ON_WM_QUERYDRAGICON()
73      ON_BN_CLICKED(IDC_MOD_LABEL, &ClxDlg::OnBnClickedModLabel)
74      ON_EN_CHANGE(IDC_EDIT2, &ClxDlg::OnChangeEdit2)
75      ON_MESSAGE(WM_MY_MESSAGE, &ClxDlg::OnMyMessage)
76  END_MESSAGE_MAP()
77
```

```
Dlg.h  lx.rc - IDD_LX_DIALOG - Dialog  Resource.h  pch.h  lx.cpp  framework.h  lxDlg.cpp*  X
lx  ClxDlg  OnBnClickedModLabel()
157  //当用户拖动最小化窗口时系统调用此函数取得光标
158  //显示。
159  HCURSOR ClxDlg::OnQueryDragIcon() { ... }
163
164
165
166  void ClxDlg::OnBnClickedModLabel() { ... }
176
177
178  void ClxDlg::OnChangeEdit2() { ... }
190
191
192  afx_msg LRESULT ClxDlg::OnMyMessage(WPARAM wParam, LPARAM lParam)
193  {
194      return 0;
195  }
196
```


发送自定义消息

- 用户可以在需要的地方通过PostMessage或SendMessage函数来发送自定义消息，例如：
 PostMessage(WM_MY_MESSAGE);
- PostMessage只把消息放入消息队列，不管消息处理程序是否处理该函数都返回，然后继续执行，也就是说该函数是一个异步消息发送函数；
- SendMessage函数则必须等待消息处理函数处理完了消息之后才返回调用该函数处继续执行，这是个同步消息发送函数；
- PostMessage的返回值表示PostMessage函数是否正确执行，而SendMessage的返回值表示消息处理函数的返回值。
- 注意，在没用多线程情况下不要用SendMessage,以免死锁。

```

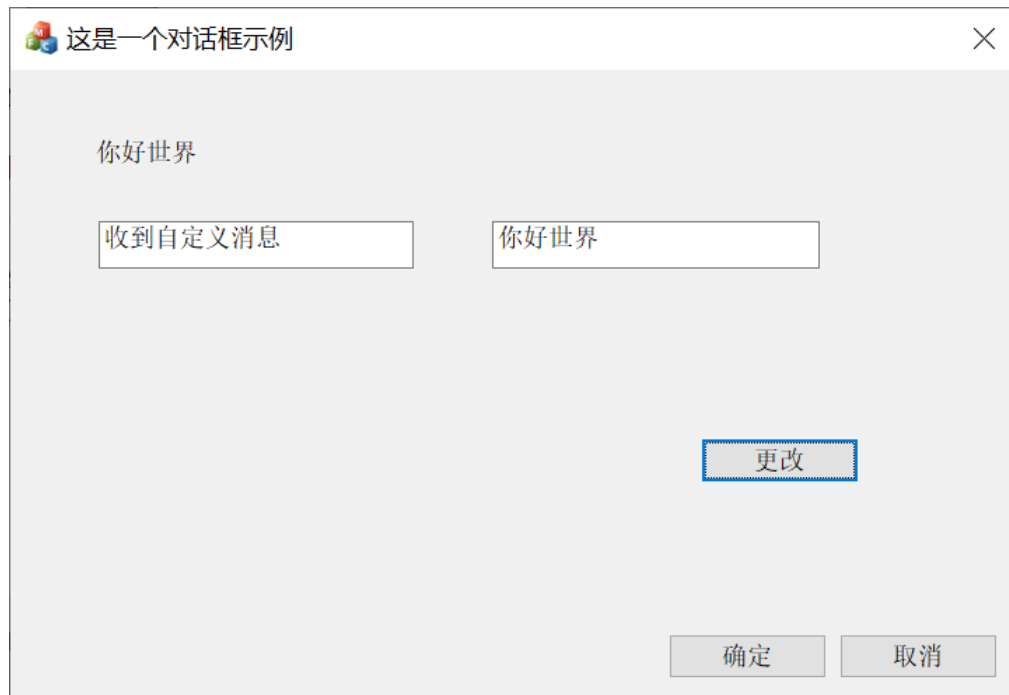
void C1xDlg::OnBnClickedModLabel()
{
    // TODO: 在此添加控件通知处理程序代码
    LPCTSTR str1 = _T("你好世界");
    edit2.SetWindowTextW(str1);
    //根据ID获取控件
    CEdit* pEdit1;
    pEdit1 = (CEdit*)GetDlgItem(IDC_EDIT1);
    pEdit1->SetWindowTextW(str1);
    UpdateData(true);
    if (strEdit2.Compare(str1) == 0)
        PostMessage(WM_MY_MESSAGE);
}

```

```

afx_msg LRESULT C1xDlg::OnMyMessage(WPARAM wParam, LPARAM lParam)
{
    LPCTSTR str1 = _T("收到自定义消息");
    //根据ID获取控件
    CEdit* pEdit1;
    pEdit1 = (CEdit*)GetDlgItem(IDC_EDIT1);
    pEdit1->SetWindowTextW(str1);
    return 0;
}

```



执行效果

3.9 MFC的文件操作

- 在MFC中，与文件处理关系较为密切的类有两个，一个是CFile类，另一个是CFileDialog 类。
- CFile类是MFC 文件类的基类，它封装了几乎所有的用于文件访问的Win32 API，提供基本的文件输入、输出操作。
- CFileDialog 类并不参与文件的处理，它只是封装了应用程序中经常会使用的Windows的文件对话框。
- 文件对话框提供了一种简单的与Windows 标准相一致的文件打开和文件存盘的对话框功能，它只提供“选择（或输入）要打开或保存的文件路径”的功能。

3.9.1 CFile类

- 要使用CFile类操作文件，首先要构造CFile对象。
- 使用缺省构造函数构造 CFile类对象 :缺省构造函数仅仅构造一个没有关联文件的空CFile对象，必须使用Open成员函数来打开文件。

```
BOOL CFile::Open( LPCTSTR lpszFileName, UINT  
    nOpenFlags, CFileException* pError = NULL );
```

- lpszFileName：带路径的文件名，指定要打开的文件。
- nOpenFlags：指定的文件打开方式，nOpenFlags的取值可以是多个标志的组合，
例如：CFile::modeCreate | CFile::modeWrite。

- 常用的文件打开方式标志

CFile::modeCreate	调用构造函数构造一个新文件，如果文件已存在，则长度变成0。
CFile::modeNoTruncate	此值与modeCreate组合使用。如果所创建的文件已存在则其长度不变为0；若不存在则由modeCreate标志创建一个新文件。
CFile::modeRead	以只读方式打开文件。
CFile::modeReadWrite	以读、写方式打开文件。
CFile::modeWrite	以只写方式打开文件。
CFile::modeNoInherit	阻止文件被子进程继承。

- pError：该参数为一个指向CFileException对象的指针，指向的CFileException对象用于保存打开文件出错时抛出的异常，该参数长缺省。

例：

```
CFile file; //创建对象
```

```
file.Open( "d:\\a.txt" ,CFile::modeRead );  
//以读写方式打开文件
```

- 使用指定文件名和打开类型的构造函数

`CFile(LPCTSTR lpszFileName, UINT nOpenFlags)`

- `lpszFileName`指定的文件名
- `nOpenFlags`指定的文件打开方式

例：

```
CFile file("d:\\a.txt",CFile::modeRead );
```

//以只读方式打开D:盘根目录上的文件并构造CFile对象file。

文件读取

- CFile类中读取文件的函数是Read()，该函数的格式如下：

virtual UINT Read (void* lpBuf, UINT nCount);

- lpBuf：指向用户提供的缓冲区以接收从文件中读取的数据。
- nCount：允许从文件中读出的字节数的最大值，对文本模式的文件，回车换行作为一个字符。
- 该函数从与CFile 对象相关联的文件中读取数据到lpBuf指定的缓冲区。函数的返回值是实际读取到缓冲区的字节数，如果读到文件尾，则返回值可能比参数nCount指定的值小。如果函数出错，则会抛出CFileException异常。

- 例：

```
CFile fileOpen;  
try  
{  
    fileOpen.Open("d:\\a.txt",CFile::modeRead );  
    int i=fileOpen.GetLength();  
    fileOpen.Read(s,i);  
    fileOpen.Close();  
}  
catch(CFileException *e)  
{  
    CString str;  
    str.Format("读取数据失败的原因是:%d",e->m_cause);  
    MessageBox(str);  
    fileOpen.Abort();  
    e->Delete();  
}
```

文件写入

- 写文件的成员函数为Write()，该函数的格式如下：
virtual void Write(const void* lpBuf,UINT nCount);
 - lpBuf：指向用户提供的缓冲区，包含将写入文件中的数据。
 - nCount：要写入的字节数。对文本模式的文件，回车换行作为一个字符。
 - 该函数将数据从缓冲区写入与CFile 对象相关联的文件。如果函数出错，则会抛出CFileException异常。Write 在几种情况下均产生异常，包括磁盘满的情况。

- 例 :


```

CFile fileSave;
CString
m_Edit1( "aaaaaaaaabbbbbbbbbbccccccccccdddddddddd\neeeeee
ee" );
try
{
    fileSave.Open
("d:\\a.txt",CFile::modeCreate|CFile::modeWrite);
    fileSave.Write(m_Edit1,m_Edit1.GetLength());
    fileSave.Close();
}
catch(CFileException *e)
{
    CString str;
    str.Format("保存数据失败的原因是:%d",e->m_cause);
    MessageBox(str);
    fileSave.Abort();
    e->Delete();
}

```

文件的读写定位

virtual LONG Seek(LONG IOff,UINT nFrom);

- IOff：指定针移动的字节数。
- nFrom：指针移动的模式，可为以下值之一：
- CFile::begin 从文件开始，把指针向后移动IOff 字节。
- CFile::current 从当前位置开始，把指针向后移动IOff 字节。
- CFile::end 从文件尾开始，把指针向前移动IOff 字节。注意，此时IOff 应为负。如果为正值，则超出文件尾。
- 该函数在一个已打开的文件中重新定位文件位置指针。如果要求移动道德的位置合法，则返回位置指针从文件开始起的新的偏移量。否则，位置指针将不移动并抛出CFileException 异常。

- 用于定位文件读写位置的成员函数还有：

void SeekToBegin();

- 将文件指针指向文件开始处。

DWORD SeekToEnd(); //返回文件长度（字节数）

- 将文件指针指向文件逻辑尾部

文件关闭

- CFile类用于关闭文件的成员函数有两个，一个是Close()，一个是Abort()。

void Close();

void Abort();

- Abort()与Close()区别在于Abort()忽略失败，不会因失败而抛出异常而Close()遇到错误则会抛出CFileException异常；

3.9.2 CFileDialog类

- 要使用CFileDialog提供的文件对话框，首先要定义一个CFileDialog类的对象。

CFileDialog类的构造函数

```
CFileDialog(  
    BOOL bOpenFileDialog,  
    LPCTSTR lpszDefExt = NULL,  
    LPCTSTR lpszFileName = NULL,  
    DWORD dwFlags = OFN_HIDEREADONLY |  
        OFN_OVERWRITEPROMPT,  
    LPCTSTR lpszFilter = NULL,  
    CWnd* pParentWnd = NULL  
);
```

- 参数说明：
 - bOpenFileDialog：如果为TRUE，则创建一个文件打开对话框；如果为FALSE，则构造一个File Save As(另存为)对话框。
 - lpszDefExt：缺省文件扩展名，如果用户在文件名编辑框中不包含扩展名，则lpszDefExt定义的扩展名自动加到文件名后。如果为NULL，则不添加扩展名。
 - lpszFileName：初始显示于文件名编辑框中的文件名，如果为NULL，则不显示初始文件名。
 - dwFlags：一个或多个标志的组合，使你可定制对话框。相关标志的使用请读者自己查阅相关资料。
 - lpszFilter：一系列字符串对，指定可以应用到文件的过滤器。如果指定过滤器，仅被过滤器允许的文件显示于文件列表框中。
 - pParentWnd：指向文件对话框对象的父窗口或拥有者窗口。

对LpszFilter参数的进一步说明：

- 该参数用于指定哪些类型的文件可在文件列表框中显示。文件的类型决定于扩展名。
- 该参数为一字符串，可同时指定多个要显示的文件类型，每个类型由“文件类型描述”和“文件扩展名”两部分组成，“文件类型描述”和“扩展名”间用“|”分隔，同种类型文件的多个扩展名间可以用“;”分割，每种文件类型间也是用“|”分隔，字符串的末尾用“||”指明。
- 例如，Microsoft Excel 允许用户用.XLC 扩展名（表）或.XLS（工作表）打开文件，Excel 过滤器应如下：

```
char szFilter[] = "Chart Files (*.xlc)|*.xlc|Worksheet  
Files (*.xls)|*.xls| Data Files (*.xlc;*.xls)|*.xls;*.xls| All  
Files (*.*)|*.* ||";
```

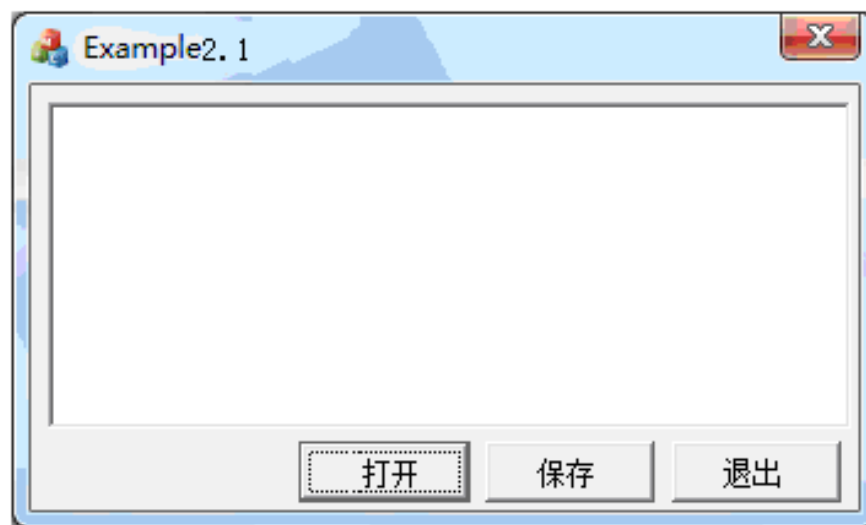
- 创建CFileDialog对象后通过调用DoModal() 成员函数来显示对话框，用户输入路径和文件名。
- DoModal()函数格式如下：
- int DoModal();
- 返回值为IDOK 或IDCANCEL，如果返回IDCANCEL，可调用CommDlgExtendedError 函数来判断是否是发生错误。IDOK或IDCANCEL 是表明用户选择了OK 还是Cancel 按钮的常数。

- 用CFileDialog打开的文件，可以使用成员函数 GetFileName()， GetFileTitle()， GetFilePath()， GetFileExt()来取得相关信息。

```
char szFilter[] = "数据文件 (*.TXT;*.DAT)| All  
Files (*.*)|*.*||";  
CFileDialog a(true, ".xls", 0, 0, szFilter);  
int x=a.DoModal();  
if(x==IDOK)  
    MessageBox(a.GetPathName());  
else  
    MessageBox( "没有输入文件名" );
```

例题1

- 编写一个简单的文本文件编辑器，其界面如图所示。



- 完成这个程序需要经过如下步骤：
 - (1) 利用应用程序向导，创建一个对话框应用程序框架。注意，本例使用的是ANSI字符集，而不是Unicode字符集。
 - (2) 编辑主对话框资源（界面设计）。
 - (3) 使用类向导为文本编辑框添加控件变量，变量名为m_Edit1，类别为“Value”，变量类型为CString。

(4) 双击 “打开” 按钮为其添加消息处理函数，并在函数中添加代码如下：

```
char s[10000];
char szFilter[] = "文本文件(*.txt)|*.txt| All Files(*.*)|*.*||";
CFileDialog OpenDlg(true, ".txt", 0, 0, szFilter);
int x=OpenDlg.DoModal();  int i=0;
if(x==IDOK){
CFile fileOpen;
try{
        fileOpen.Open(OpenDlg.GetPathName(),CFile::modeRead );
        i=fileOpen.GetLength();
        fileOpen.Read(s,i);
        fileOpen.Close();
}
catch(CFileException *e){
        CString str;
        str.Format("读取数据失败的原因是:%d",e->m_cause);
        MessageBox(str);
        fileOpen.Abort();
        e->Delete();
}
}
CString str(s,i);
m_Edit1=str;
UpdateData(false);
```


- 为“保存”按钮添加如下的处理函数：

```
void CExample11Dlg::OnBnClickedSave(){
    UpdateData();
    char szFilter[] = "文本文件(*.txt)|*.txt| All Files(*.*)|*.*||";
    CFileDialog SaveDlg(false, ".txt", 0, 0, szFilter);
    int x=SaveDlg.DoModal();
    if(x==IDOK){
        CFile fileSave;
        try{
            fileSave.Open
(SaveDlg.GetPathName() ,CFile::modeCreate|CFile::modeWrite);
            fileSave.Write(m_Edit1,m_Edit1.GetLength());
            fileSave.Close();
        }
        catch(CFileException *e){
            CString str;
            str.Format("保存数据失败的原因是:%d",e->m_cause);
            MessageBox(str);
            fileSave.Abort();
            e->Delete();
        }
    }
}
```

```

void CMyNotePadDlg::OnBnClickedOpen()
{
    // TODO: 在此添加控件通知处理程序代码
    TCHAR s[1000];
    LPCTSTR szFilter = _T("文本文件(*.txt)|*.txt|All File(*.*)|*.*|");
    CFileDialog OpenDlg(true, _T(".txt"), 0, 0, szFilter, NULL);
    int x = OpenDlg.DoModal();
    int i = 0;
    if (x == IDOK) {
        CFile fileOpen;
        try {
            fileOpen.Open(OpenDlg.GetPathName(), CFile::modeRead);
            i = fileOpen.GetLength();
            fileOpen.Read(s, i);
        }
        catch(CFileException *e) {
            CString str;
            str.Format(_T("读取数据失败的原因是:%d"), e->m_cause);
            MessageBox(str);
            fileOpen.Abort();
            e->Delete();
        }
    }
    CString str(s, i);
    m_Edit1 = str;
    UpdateData(false);
}

```

```

void CMyNotePadDlg::OnBnClickedSave()
{
    // TODO: 在此添加控件通知处理程序代码
    UpdateData();
    TCHAR szFilter[] = _T("文本文件(*.txt)|*.txt| All Files(*.*)|*.*|");
    CFileDialog SaveDlg(false, _T(".txt"), 0, 0, szFilter);
    int x = SaveDlg.DoModal();
    if (x == IDOK) {
        CFile fileSave;
        try {
            fileSave.Open(SaveDlg.GetPathName(), CFile::modeCreate | CFile::modeWrite);
            fileSave.Write(m_Edit1, m_Edit1.GetLength());
            fileSave.Close();
        }
        catch (CFileException* e) {
            CString str;
            str.Format(_T("保存数据失败的原因是:%d"), e->m_cause);
            MessageBox(str);
            fileSave.Abort();
            e->Delete();
        }
    }
}

```

MSDN联机帮助文档

- <https://docs.microsoft.com/zh-cn/cpp/mfc/mfc-desktop-applications?view=vs-2019>