VS2013 MFC 基于对话框编程系列（19 个程序）

VS2013/MFC 基于对话框编程 :[1]创建 MFC 工程

更新：2014-08-07 14:18

VS2013 作为最新版的 **Visual studio**，界面和功能上相比之前的版本有了较大的改善和提高，**MFC** 作为集成 **API** 的简单版更适合编程开发，许多应用软件都是基于对话框的，这里就介绍一下如何在 **VS2013** 中创建一个基于对话框的 **MFC** 项目。

工具/原料

- **Visual Studio 2013**

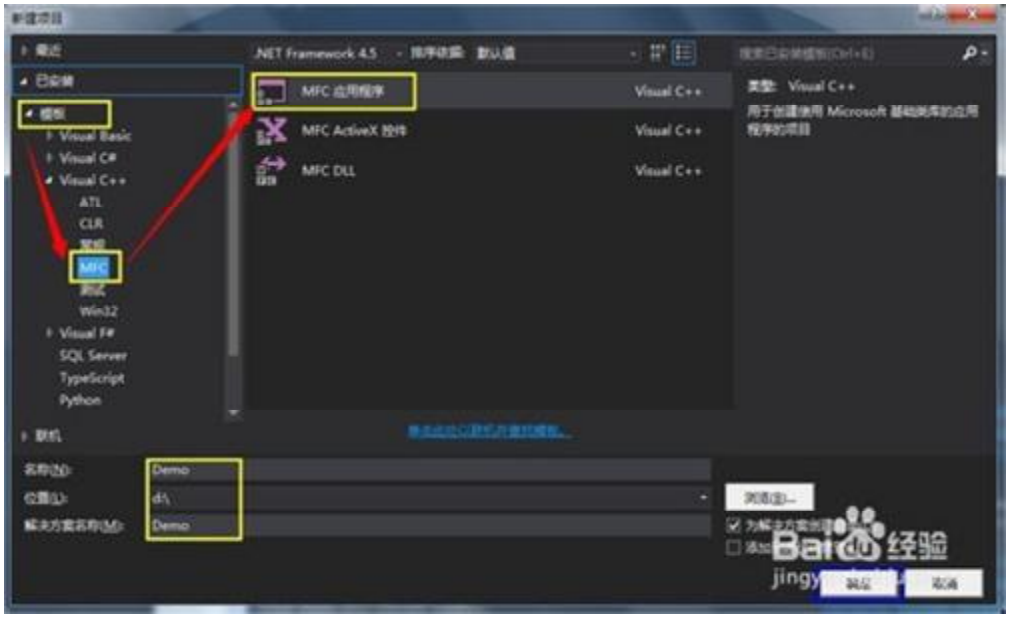
方法/步骤

1. 1

双击打开 **VS2013**，进入起始页，在左侧开始处选择“新建项目”；或者选择菜单栏中“文件”，依次选择“新建”、“项目”。可以看到起始页还包括许多介绍性文章的链接，包括 **VS2013** 新增功能的介绍等。

2. 2

选择新建项目后，在新建项目对话框中选择 模块-》**Visual C++-》MFC-》MFC** 应用程序，并确定好存放路径和项目名称，点击“确定”。



3. 3

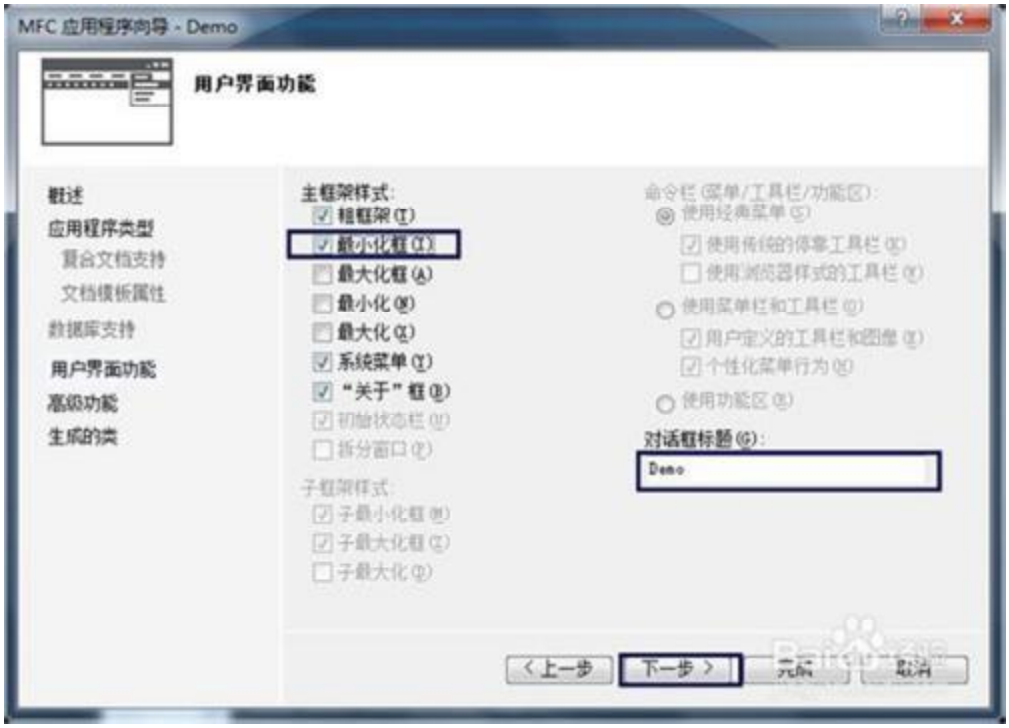
进入应用程序向导，一开始会给出默认的项目配置，点击“下一步”即可。

4. 4

选择“基于对话框”，**MFC** 可以选择在静态库中使用，或者在共享 **DLL** 中使用。一般选择共享使用就行，静态库中使用会把所有用到的 **dll** 集成到 **exe** 文件中，最终生成的文件一般可以直接使用，但占用更大空间。

5. 5

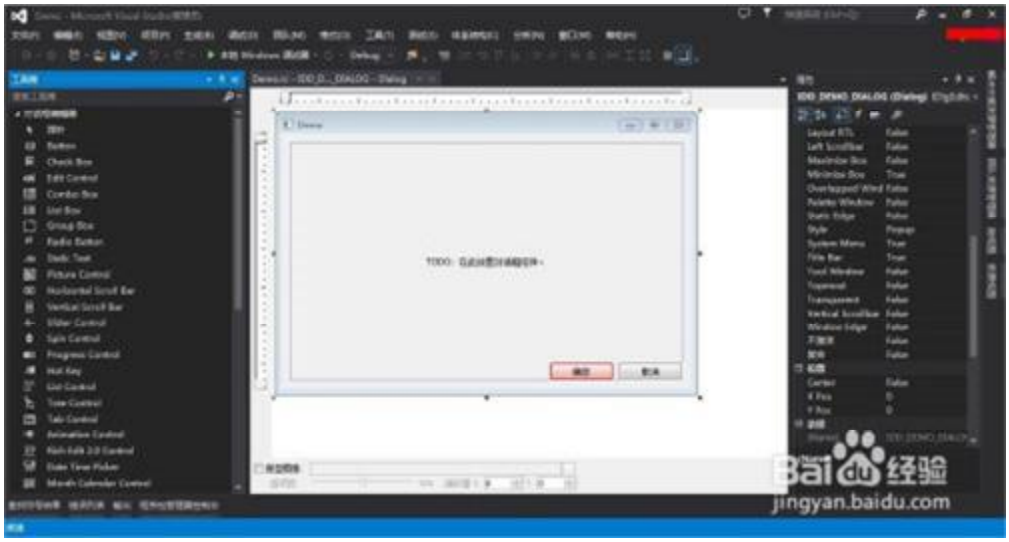
选择主框架样式，可以自由选择是否添加最小化框、最大化框。如果觉得没必要“关于”对话框也可以去掉，对话框标题一般不需要更改。



6. 6

高级功能一般默认即可，但如果用不上“**ActiveX** 控件”可以去掉勾选；如果需要涉及网络编程就把“**windows** 套接字”选上。不过没选上也不要紧，在程序中可以自己添加部分代码导入套接字。

最后是自动生成的两个类的头文件和源文件名称，可以修改基类，但一般不用改，默认完成就行。至此，一个基于对话框的 **MFC** 项目就创建好了。



END

注意事项

- 合理的向导设置可以减少后续编程的麻烦

VS2013/MFC 基于对话框编程：[2]项目整体结构

当我们创建好一个 **MFC** 项目以后，为了更快速的编程，快速找到需要修改的部分，这就得熟悉 **VS** 开发环境的各个模块的作用，各个模块之间的关系。

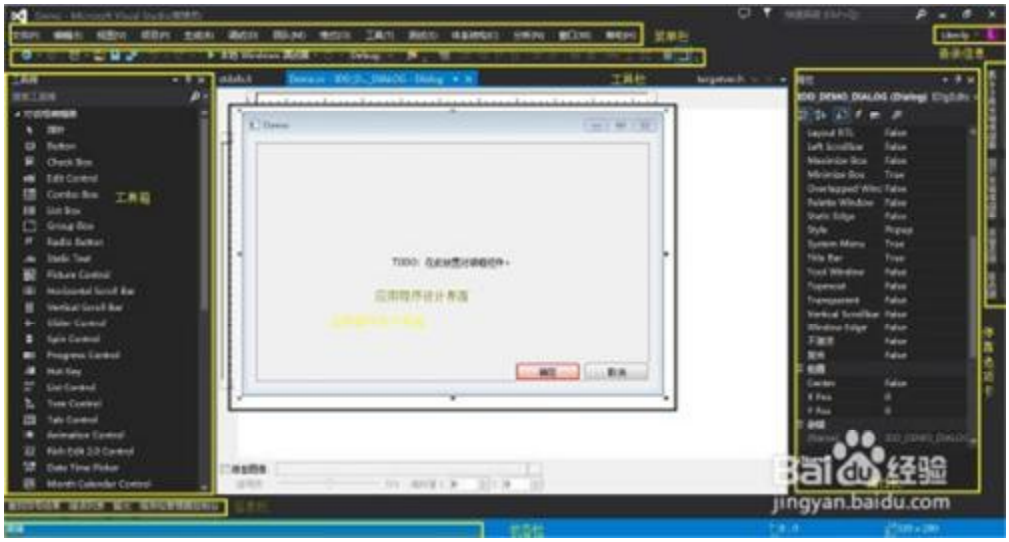
工具/原料

- **Visual Studio 2013**

项目整体结构

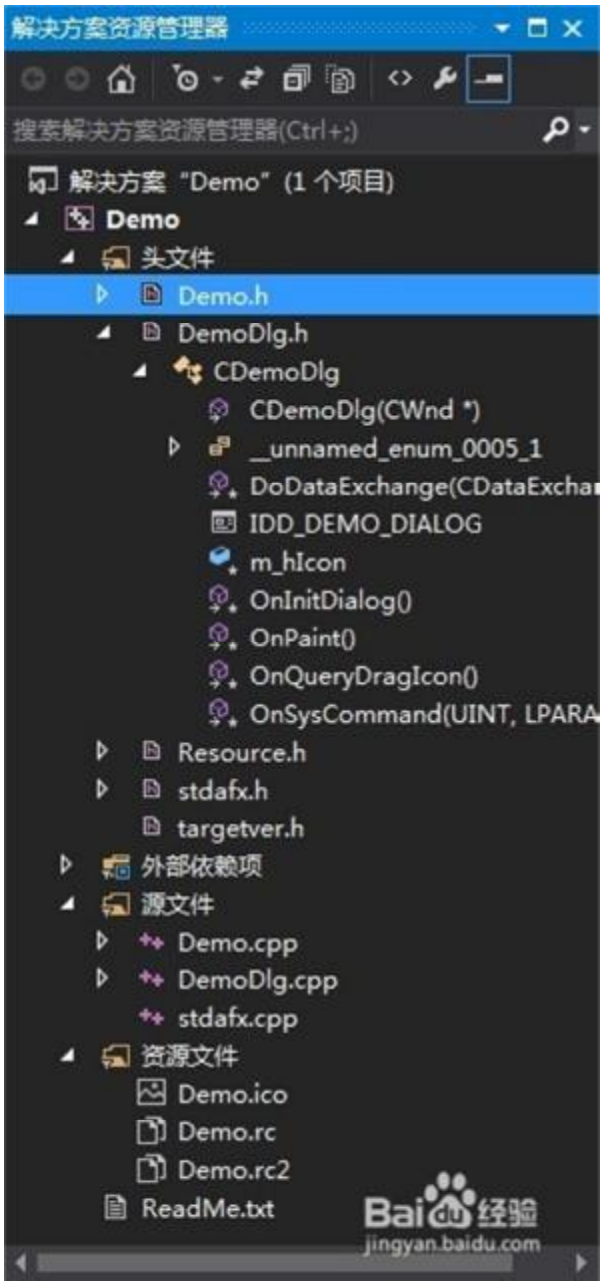
1. 1

创建好一个 **MFC** 工程，观察 **VS** 项目界面的布局，上面是菜单栏和工具栏，在设计界面的左侧可以调出工具箱；右侧是属性栏、类视图、资源视图和资源管理器等；底部是一些显示错误信息、符号查询结果、项目生成信息的栏目；最下方是状态栏，显示当前开发状态。



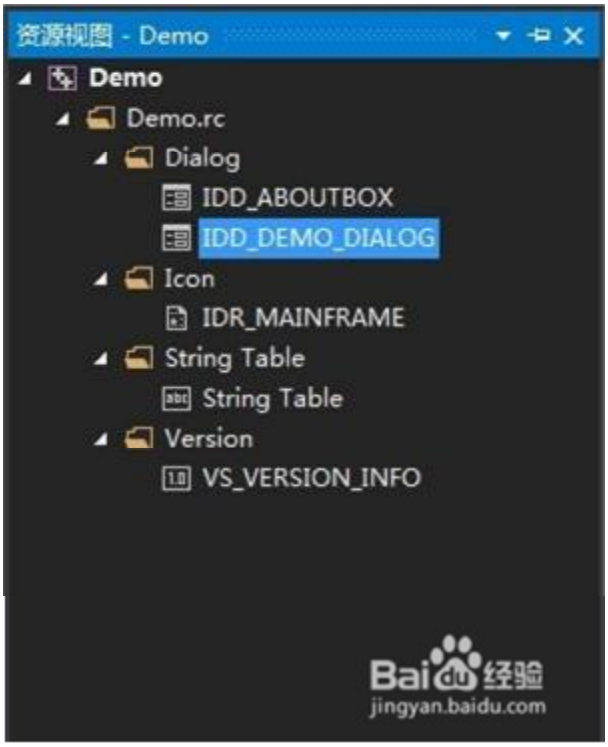
2. 2

打开解决方案资源管理器，可以看到所有的头文件、源文件以及资源文件都归类排序摆放，便于快速寻找需要打开的内容。点开头文件前面的三角号可以查看类的成员变量和函数。



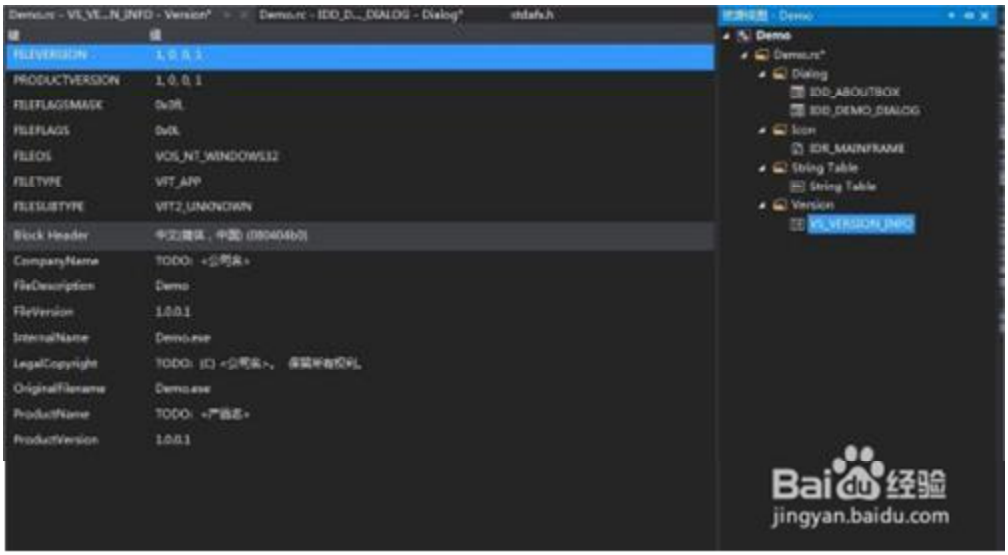
3. 3

看看资源视图，资源视图包含了应用程序用到的所有资源，包括对话框资源、应用图标、菜单、字符串、软件信息等等。



4. 4

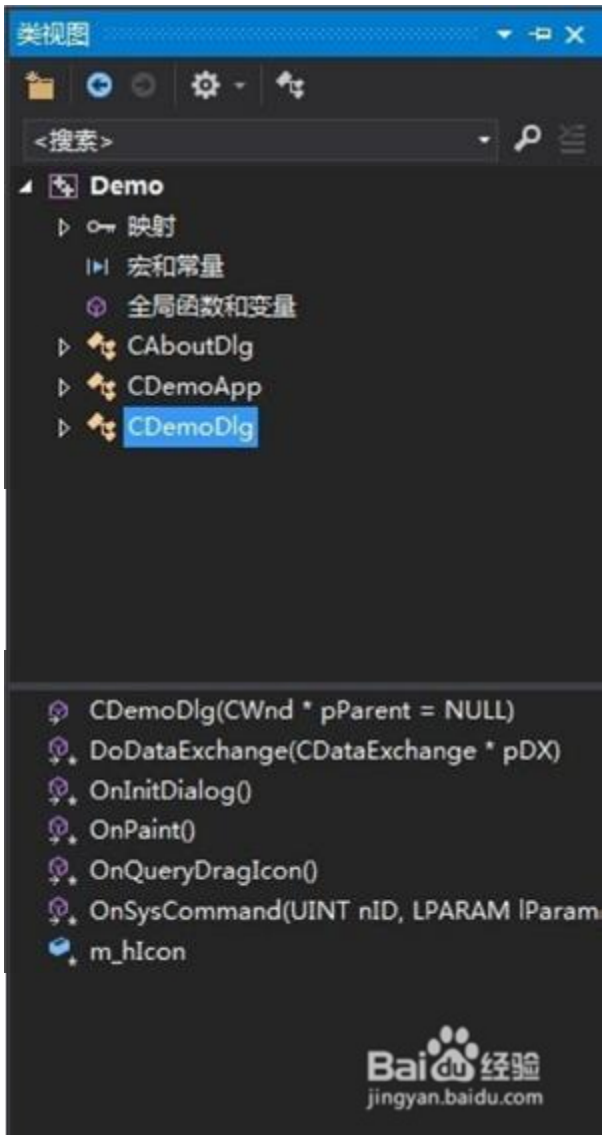
打开版本信息，可以看到自己编写的应用程序的相关信息，比如版本号、公司名称等，自己觉得需要修改的可以依情况修改。



5. 5

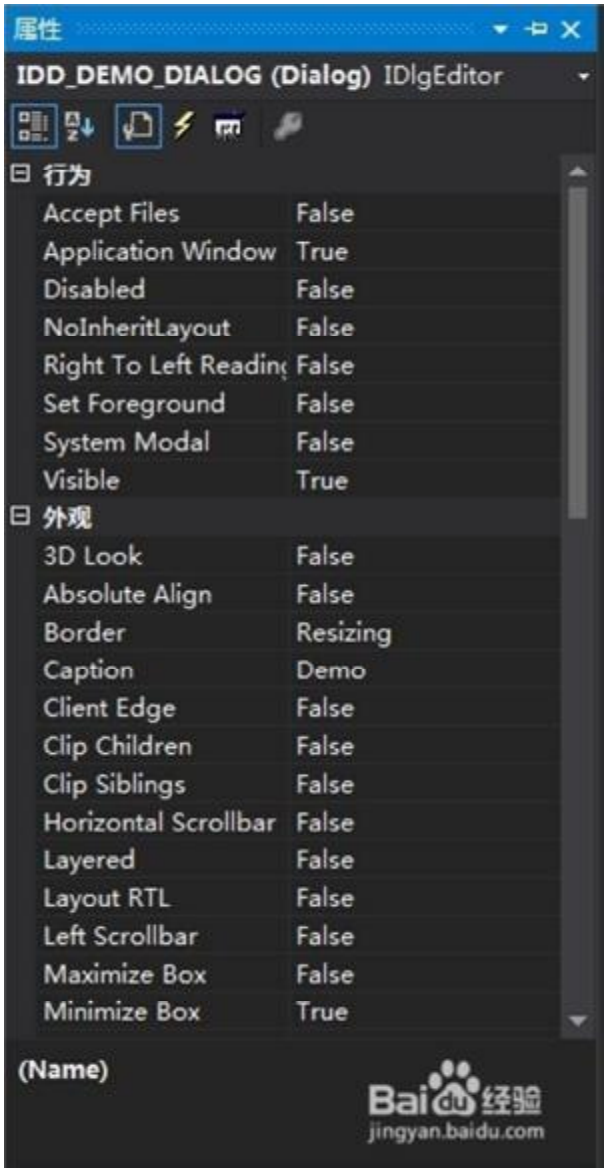
再看类视图，类视图包含了应用程序用到的所有类，包括自动生成的类以及开发者自己创建的类。点击类名，在下方会显示该类包含的所有成员变量和函数，不同类型的函数或者变量都有不同的符号标记。

通过类视图可以快速访问类的头文件和源文件，双击类名打开头文件，点击下方的函数可直接跳转到对应的函数定义处。



6. 6

属性页用来修改和查看对话框中控件（包括对话框本身）的各项属性，每类控件的属性项都不一样，需要根据情况修改，当然所有的属性都可以以代码形式实现，但能直接修改最好，方便快捷。



END

项目文件

1. 找到项目的生成文件，如果没有生成解决方案，应用文件是不包含 **Debug** 文件夹的；其中的**.sln** 文件就是项目启动文件，双击这个文件就会自动通过 **VS2013** 打开。

2. 整个项目文件系统中会有两个 **debug** 文件夹，项目的直接目录下的 **debug** 文件夹包含了生成的 **exe** 文件，其中的**.ilk** 文件用于编译连接。

3. 项目名称下的文件夹里包含了应用程序用到的资源，头文件，源文件等。

4. 项目名称下的文件夹包含一个 **res** 文件夹，是用来存放应用程序图标的，可以自己替换从而更改应用图标，不过名称和格式要一致，否则无法识别。

注意事项

- **VS2013** 开发环境使编程更加方便快捷 ,合理切换各个模块调用需要的内容可以提高编程效率。

VS2013/MFC 基于对话框编程：[3]程序执行流程

创建好一个 **MFC** 项目以后，如果是第一次创建，或者虽然创建了很多项目，却并没有认真分析自动生成的程序代码，那么就很有必要知道项目的程序框架如何，项目是从哪开始执行，到哪终止程序，如何响应消息，如何刷新界面等等问题。

工具/原料

- **visual studio 2013**

MFC 执行流程

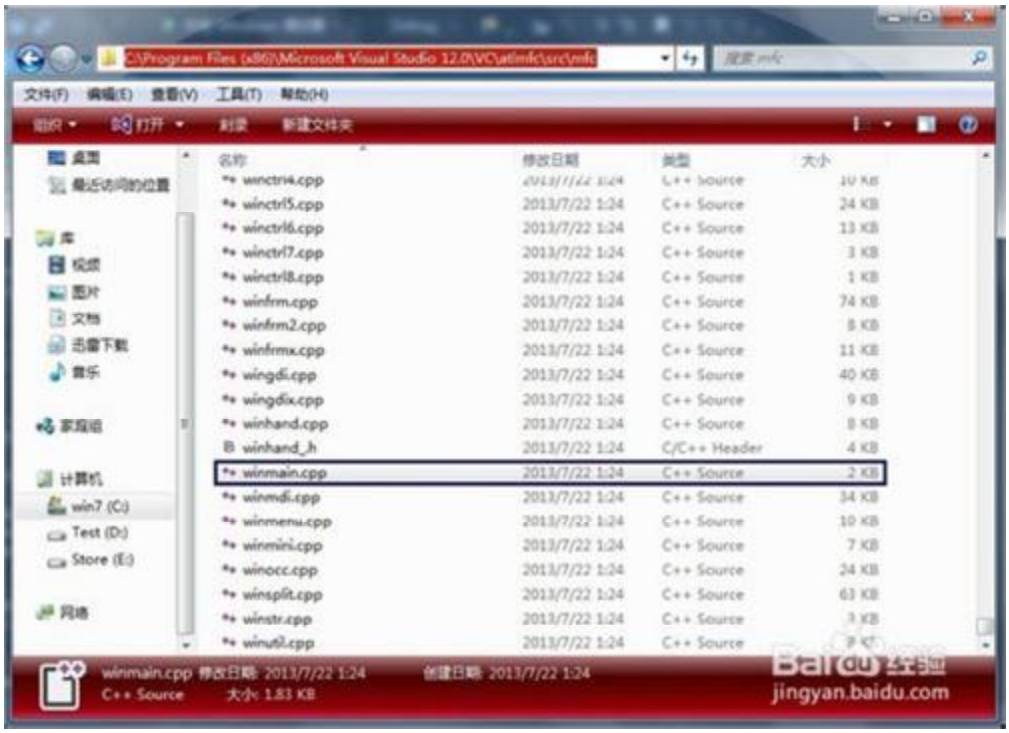
1. 每创建一个项目，一般会包含 **3** 个类，“关于”对话框类、主对话框类以及用于初始化项目的 **App** 类，假如项目名称为 **Demo**，那么这三个类分别为 **CAboutDlg**、**CDemoDlg**、**CDemoApp**。

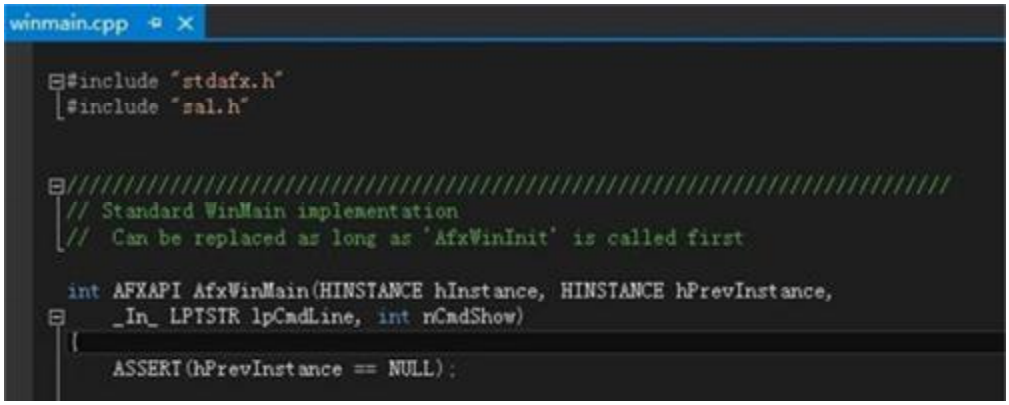
2. 项目生成后，都会生成一个属于 **CDemoApp** 类的 **theApp** 对象，对本应用程序实例化，这个在 **CDemoApp.cpp** 文件中定义，创建时调用构造函数 **CDemoApp::CDemoApp()**；这就是程序创建的第一步。

```
CDemoApp::CDemoApp()
{
    // 支持重新启动管理器
    m_dwRestartManagerSupportFlags = AFX_RESTART_MANAGER_SUPPORT_RESTART;

    // TODO:  在此处添加构造代码，
    // 将所有重要的初始化放置在 InitInstance 中
}


// 唯一的一个 CDemoApp 对象

CDemoApp theApp;
```

3. 接下来程序会调用 **winmain** 函数，这个在项目文件中找不到，但可以在 **VS2013** 的安装路径下找到，其函数声明为：

   int AFXAPI AfxWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,_In_ LPTSTR lpCmdLine, int nCmdShow)；





4. 调用完 **winmain** 后，就通过 **InitInstance()**函数初始化窗口，包括注册、创建和显示对话框，**InitInstance** 函数是 **CDemoApp** 类中除构造函数以外唯一的成员函数。

   一般不需要修改这部分程序，但有些时候可以添加部分代码，比如需要创建多个对话框时，刚启动时弹出的对话框（比如用于登录）不是主对话框，就可以在这个函数里创建主对话框之前调用登录对话框，这样就可以设置启动时的默认对话框了。

```
// CDemoApp 初始化

BOOL CDemoApp::InitInstance()
{
    // ...
    INITCOMMONCONTROLSEX InitCtrls;
    InitCtrls.dwSize = sizeof(InitCtrls);
    // ...
    InitCtrls.dwICC = ICC_WIN95_CLASSES;
    InitCommonControlsEx(&InitCtrls);

    CWinApp::InitInstance();


    AfxEnableControlContainer();
```



```
    CDemoDlg dlg;
    m_pMainWnd = &dlg;
    INT_PTR nResponse = dlg.DoModal();
```

5. 应用程序创建完了，程序开始运行了，于是进入消息循环，**windows** 程序的事件都是消息驱动的，每产生一个消息就触发一个响应事件，消息和事件通过消息映射**DECLARE_MESSAGE_MAP()**联系在一起。

   默认包含三个消息：

   ON_WM_SYSCOMMAND() **//**响应控制指令

   ON_WM_PAINT() **//**响应绘图消息，用于刷新窗口

   ON_WM_QUERYDRAGICON()//当用户拖动最小化窗口时取得光标

```
BEGIN_MESSAGE_MAP(CDemoDlg, CDialogEx)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_BUTTON1, &CDemoDlg::OnBnClickedButton1)
END_MESSAGE_MAP()
```

6. 当用户关闭应用程序时，会发送一个 **WM_CLOSE** 消息，程序响应后结束程序，如何在点击关闭时需要弹出其他对话框（比如用于提示保存），可以通过类向导添加 **WM_CLOSE** 消息处理函数，变添加相关处理程序，比如：

   void CDemoDlg::OnClose()

   {

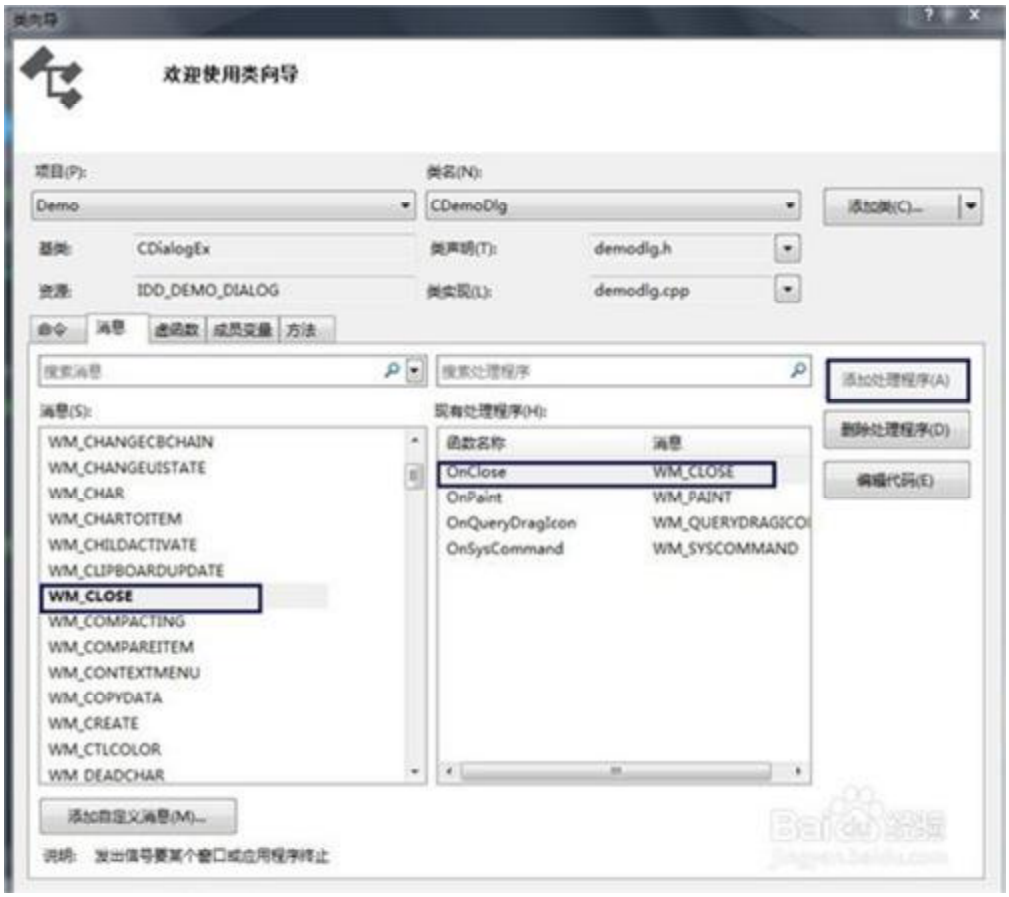
   if (MessageBox(_T("确定退出吗"), _T("提示"), MB_YESNO|MB_ICONWARNING)

   == IDNO)

return;

CDialogEx::OnClose();

}



7. 这样，一个应用程序通过定义，初始化，由 **winmain** 开始，注册、创建、显示窗口，消息响应，程序终止 完成了他的运行周期。

END

注意事项

- 消息响应在程序运行过程时间最长，用来处理所有消息对应的事件
- 本系列经验往后均已 **Demo** 项目例子进行讲解，以便更加连贯的学习。

VS2013/MFC 基于对话框编程：[4]对话框类

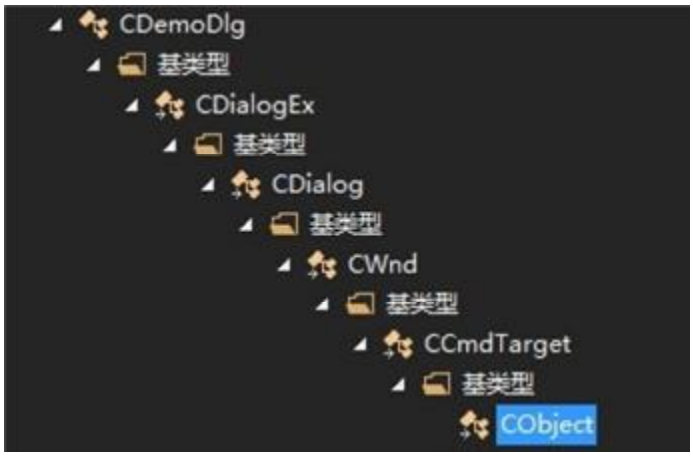
对于基于对话框类的 **MFC** 项目，自然得清楚对话框类的结构，包含的成员函数、虚函数等。项目创建完以后，至少得清楚自动生成的函数是用来干什么的，如何添加合适的代码在什么函数的什么位置，以及如何重写虚函数或者创建消息映射等，这些就是本经验的目的所在。

工具/原料

- **Visual studio 2013**

方法/步骤

1. 首先看看对话框类的继承关系，新建的项目类派生于 **CDialogEx** 类，**CDialogEx** 在 **CDialog** 类的基础上进行了扩展，而 **CDialog** 派生于窗口类 **CWnd**，说明对话框也属于一种窗口。这样对于对话框类的继承关系就有了一定了解。



2. 打开项目的头文件，最上面的**#pragma once** 表示后面的头文件只编译一次；默认生成的函数有：

CDemoDlg(CWnd* pParent = NULL); // 标准构造函数

virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV 支持

// 生成的消息映射函数

virtual BOOL OnInitDialog();

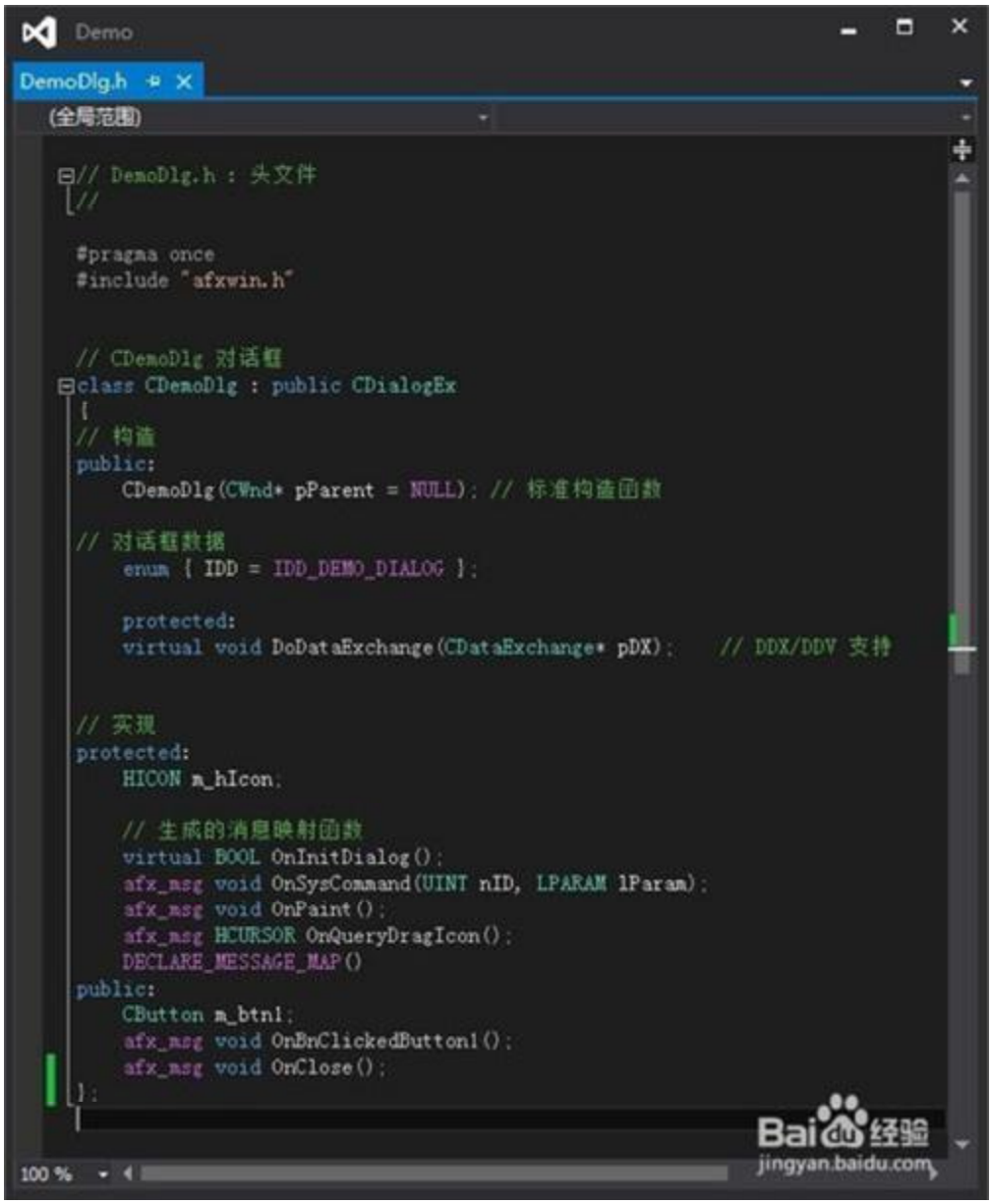
afx_msg void OnSysCommand(UINT nID, LPARAM lParam);

afx_msg void OnPaint();

afx_msg HCURSOR OnQueryDragIcon();

DECLARE_MESSAGE_MAP() **//** 消息映射的声明

三个消息在前一经验已经介绍过 ,构造函数自然是创建对话框时自动调用 ,**DoDataExchange** 函数用于存放各类控件的变量信息，**OnInitDialog** 用来初始化对话框。



3. 打开源文件可以看到各个函数的定义，“关于”对话框类的成员函数和消息映射也在这里，不多述。先讲述构造函数 **CDemoDlg()**，通过类向导生成的自定义变量或者控件的关联变量都会在构造函数中初始化一个值，不过开发者也可以直接在头文件定义变量，并手动在构造函数中添加初始化语句。

   所以说，构造函数是个赋初值的好地方。

```
CDemoDlg::CDemoDlg(CWnd* pParent /*=NULL*/)
    : CDialogEx(CDemoDlg::IDD, pParent)
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}
```

4. 再来看 **DoDataExchange** 函数，在对话框中添加新的控件并通过类向导定义了相关变量后，都会在这个用于数据交换的函数中说明，添加变量的最大最小值也会在这里体现。

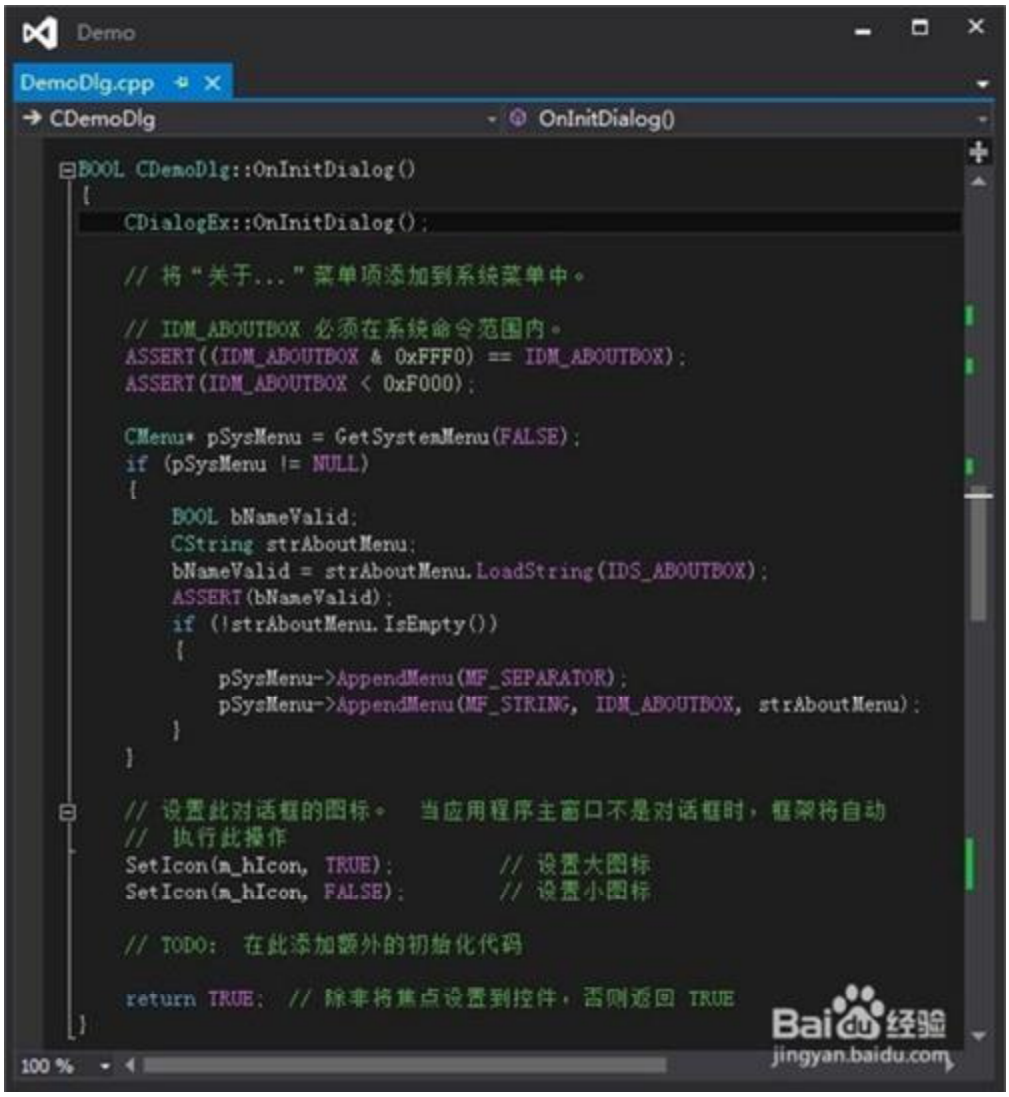
```
void CDemoDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialogEx::DoDataExchange(pDX);
    DDX_Control(pDX, IDC_BUTTON1, m_btn1);
}
```

5. 下面是消息映射，定义了所有消息的来源和处理函数，比如本例中添加的按钮按下消息，就是对于消息映射中的 **ON_BN_CLICKED(IDC_BUTTON1, &CDemoDlg::OnBnClickedButton1)**

   对于自定义消息或者某些无法通过类向导完成的消息，可以手动在这里添加映射关系，并在别处添加对应的处理函数。

```
BEGIN_MESSAGE_MAP(CDemoDlg, CDialogEx)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_BUTTON1, &CDemoDlg::OnBnClickedButton1)
    ON_WM_CLOSE()
END_MESSAGE_MAP()
```

6. 初始化函数 **OnInitDialog**，默认用来设置图标和菜单，很多时候有些操作需要在启动对话框前就做好，比如说某些控件的初始状态（按钮是否可视、是否可操作），这些初始化的设置都可以在 **OnInitDialog** 函数中添加 ,最好在提示语“**// TODO:** 在此添加额外的初始化代码”的下面添加。



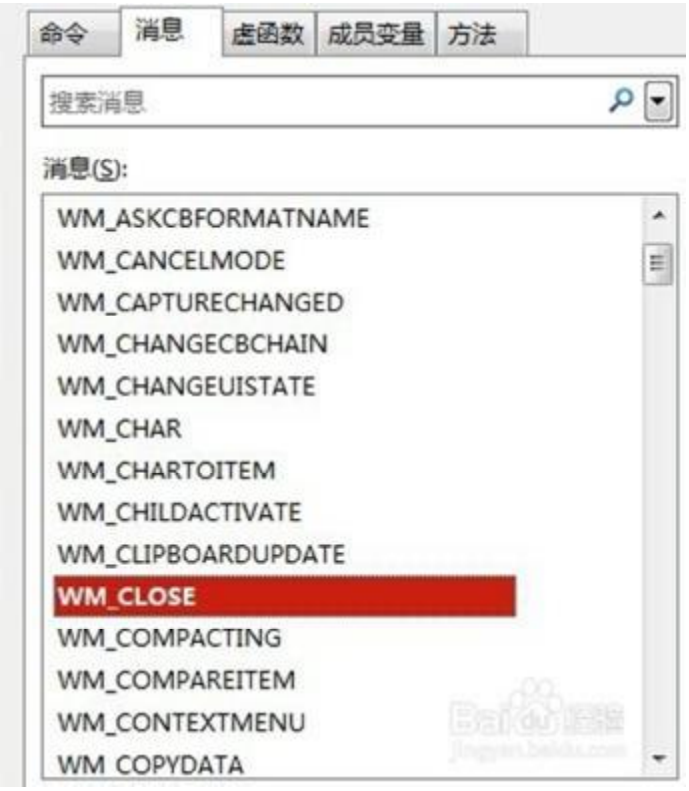
7. 系统指令响应函数 **OnSysCommand**，默认处理窗口最小化和最大化指令等，并会根据是否选中“关于”决定是否弹出“关于”对话框。这个函数一般不需要修改。

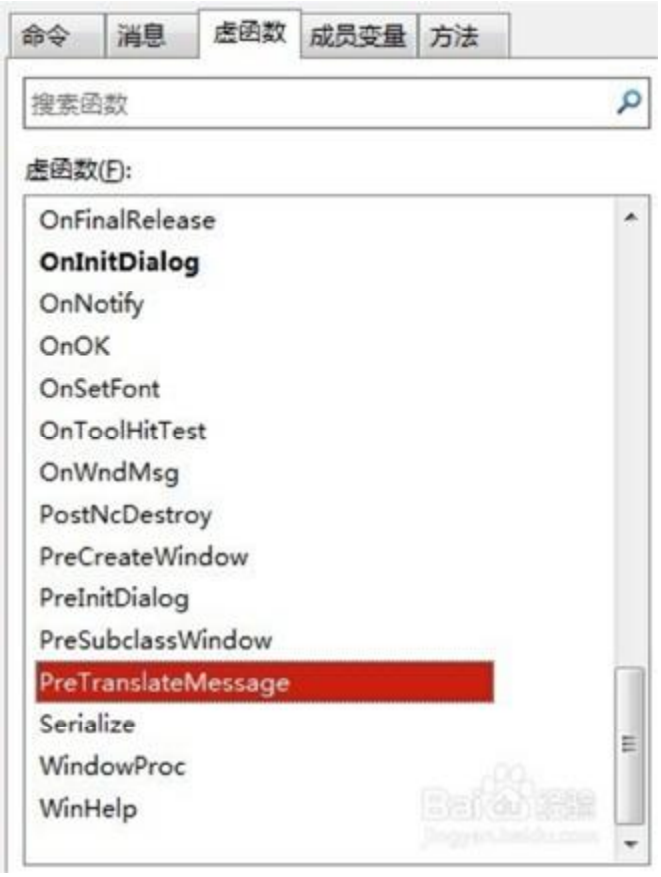
```
void CDemoDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialogEx::OnSysCommand(nID, lParam);
    }
}
```

8. 绘图消息响应函数 **OnPaint** 用于绘制窗口和图标 ,其中 **CRect** 是个存储窗口大小的结构体。**OnQueryDragIcon** 函数用于返回光标，这不多讲。

```
void CDemoDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // 用于绘制的设备上下文

        SendMessage(WM_ICONERASEBKGND, reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);

        // 使图标在工作区矩形中居中
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // 绘制图标
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialogEx::OnPaint();
    }
}
```


9. 默认生成的函数就这么多，其他常用的消息有定时器消息 **WM_TIMER**、窗口关闭消息 **WM_CLOSE**、应用程序结束消息 **WM_DESTROY**、按钮按下消息 **WM_KEYDOWN** 等。而常用的虚函数一般有 **PreTranslateMessage**、**PreCreateWindow** 等，这些以后用到再具体说明。





END

注意事项

- 类向导在菜单栏 “项目” **-**》“类向导”中可以找到，或者先后按下 **ALT+P+Z** 也可以调出，记得按键按完一个松一个。
- 在构造函数中赋初值，在初始化函数中初始化控件，这样更有条理性，大家可以试试。

VS2013/MFC 基于对话框编程：[5]按钮的使用

按钮（**Button**）可以说是对话框中最常用的控件之一，也是人机交互中必不可少的控件之一。许多事件都是通过按钮按下来触发的，在这我将介绍按钮的各种属性以及常用用法，希望对初学者有所帮助。

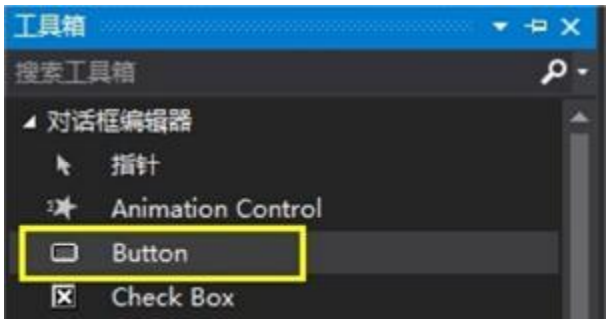
工具/原料

- **Visual Studio 2013**

添加按钮

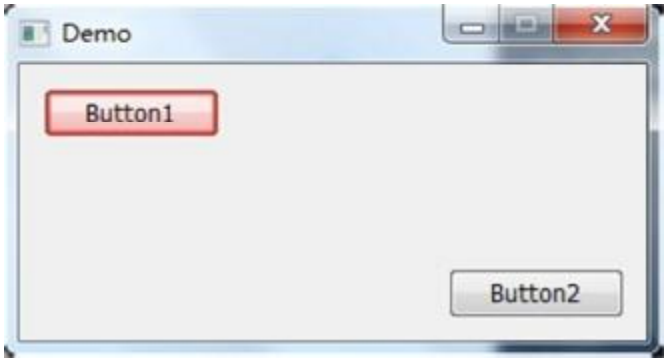
1. 1

创建好项目，在工具箱中找到对话框，按钮名称是 **Button**。



2. 2

把对话框原有的按钮和静态文本删除，从工具箱中添加两个按钮到对话框中。



3. 3

选中任意一个按钮，在属性页中可以查看按钮的所有属性。

一般需要修改的属性只有两个，“**Caption**”项和“**ID**”项，前者表示按钮文本，后者是按钮 **ID**，就像是人的身份证号一样是唯一的。

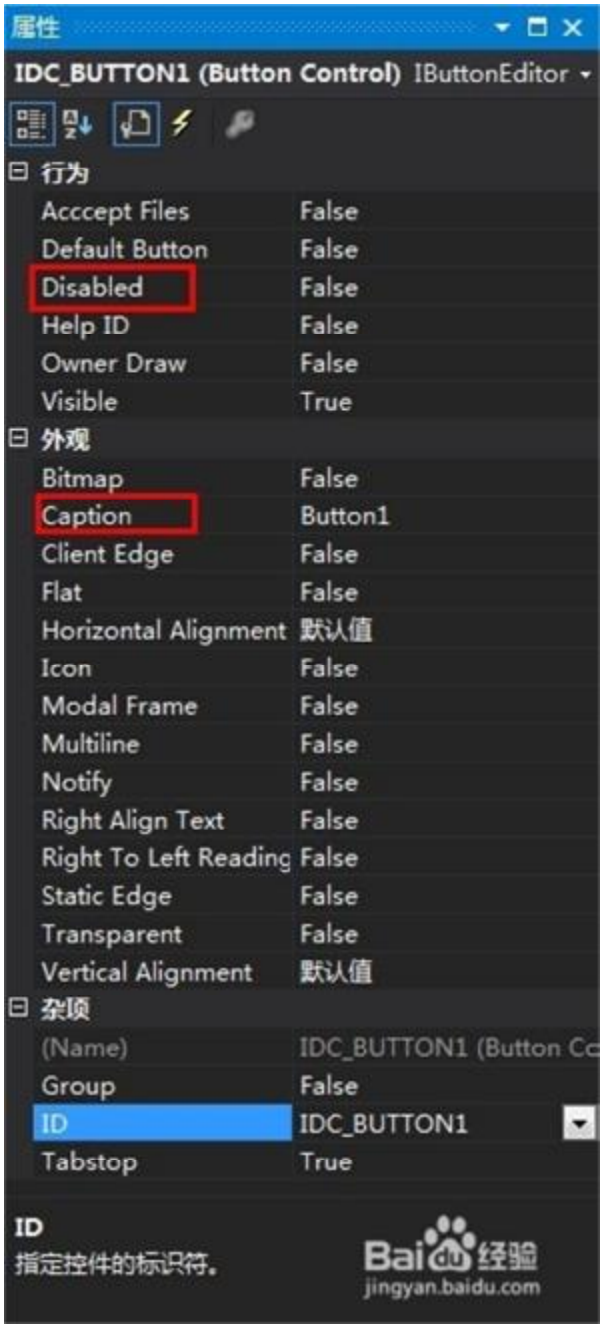
其余常用属性：

1、**Disabled**：使能，为真**(true)**表示按钮可以按下，为假**(false)**表示按钮为灰，不能操作

2、**Visible**：可见，为真表示按钮可见，为假表示按钮不可见

3、**Multiline**：多行，为真表示按钮文本可以多行显示，为假表示按钮文本不可换行显示

说明：对于只有真假两种选项的属性，可以双击属性名称进行切换**"true"**或**"false"**。选中属性名称，在属性页最下方都会有属性说明。

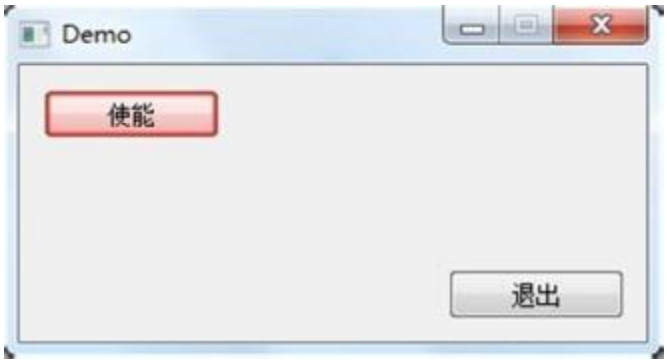


4. 4

修改两个按钮的 **Caption** 和 **ID**：

1、button1 的 **caption** 改为“使能”，**ID** 改为“**IDC_Enable**”；

2、button2 的 **caption** 改为“退出”，**ID** 改为“**IDC_Exit**”；



END

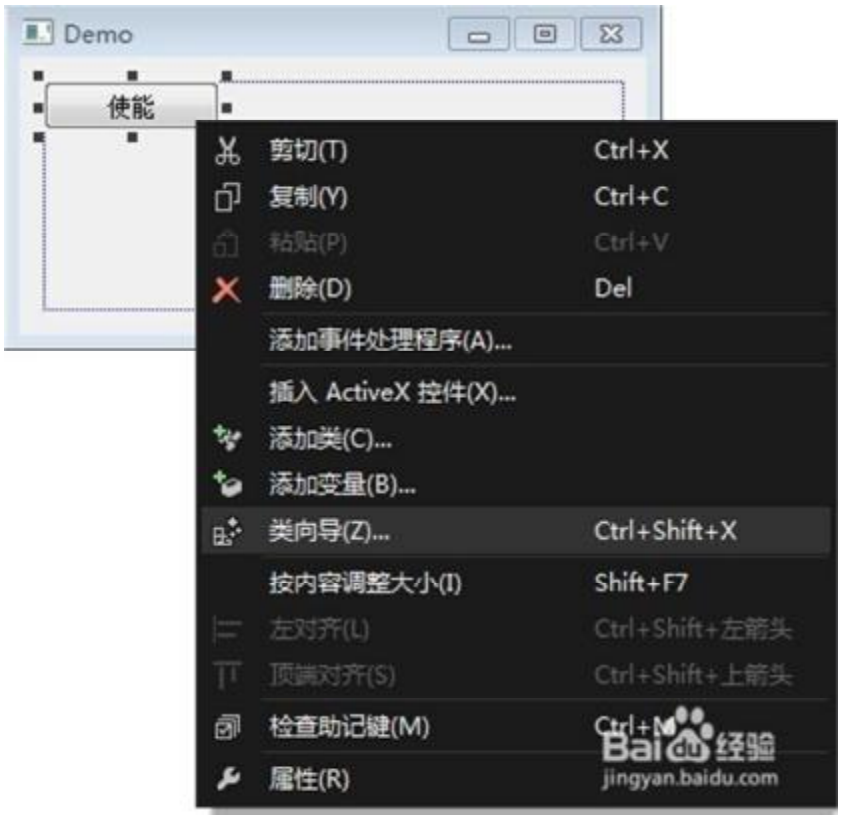
添加按钮事件处理函数

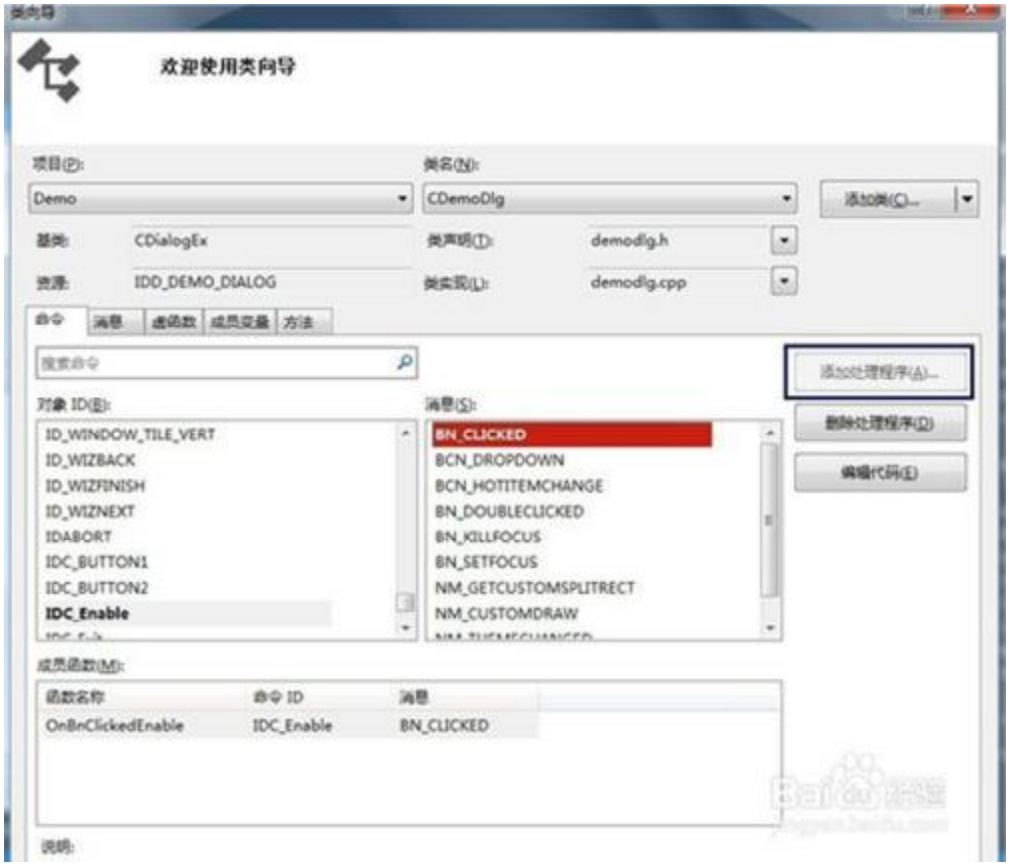
1. 1

方法一：双击按钮自动添加处理函数，自动以 **OnBnClicked** 开头，**ID** 结尾命名，这种方法最简单，但无法修改函数名称，只能生成默认的按钮按下消息的函数。

2. 2

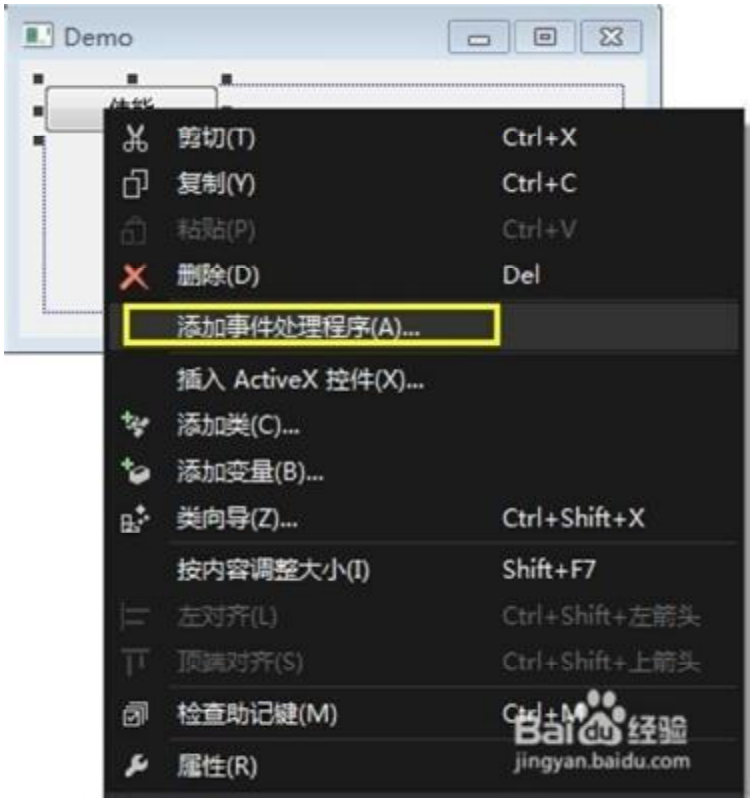
方法二：类向导，对按钮右键选择“类向导”，在消息栏中选择默认的“**BN_CLICKED**”，然后点击“添加处理程序”。

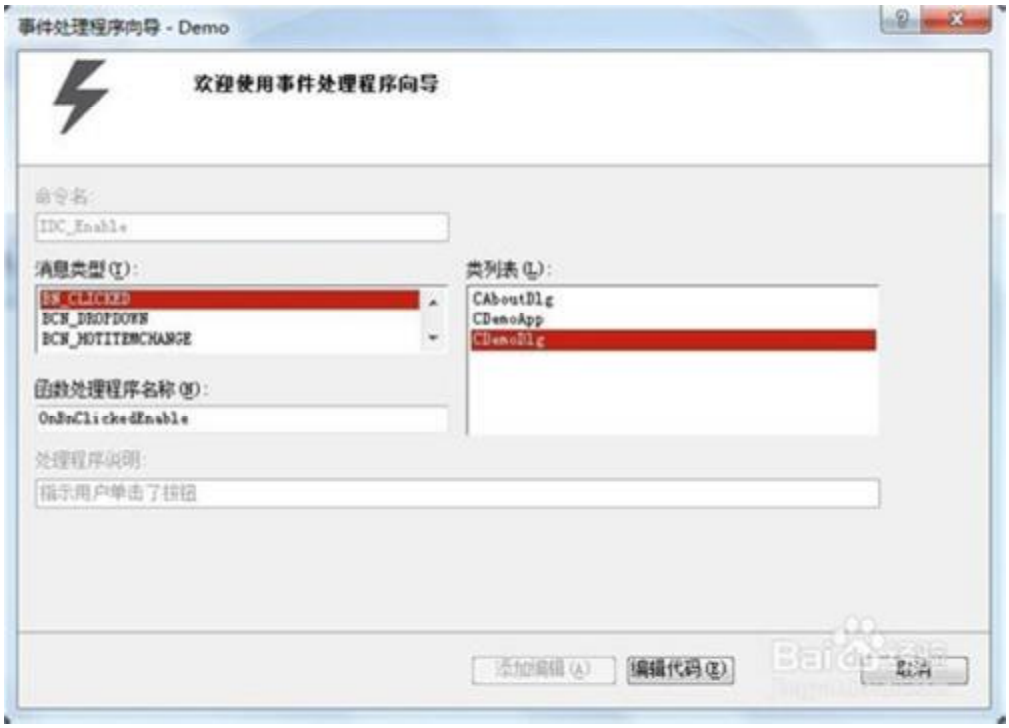




3. 3

方法三：对按钮右键选择“添加事件处理程序”，然后可以修改函数名称，点击“编辑程序”自动跳转到代码编辑界面。

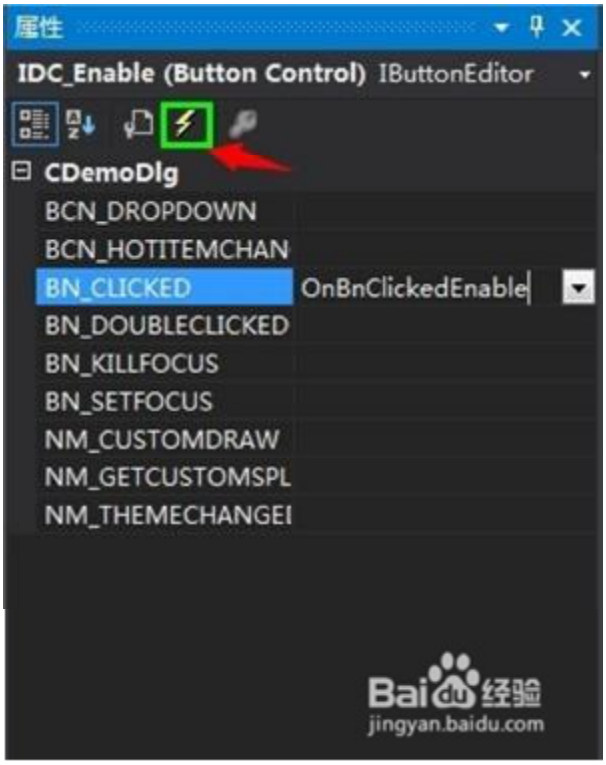




4. 4

方法四：选中按钮，在属性页中有个闪电符号，它表示控件事件，点击它，可以看到许多消息，在“**BN_CLICKED**”消息栏中添加函数，可以修改函数名，修改完按回车就能生成处理函数。

基本上所有的控件事件都可以通过这四种方法生成。



END

实现按钮功能

1. 添加按钮的控件型变量，添加方法与添加事件的方法二、三类似，以方法三为例，对“退出”按钮添加 **CButton** 型的控件变量 ,这里命名为 **m_Exit**。同理添加“使能”按钮变量 **m_Enable**；

   说明：

   1、变量名一般以**"m_"**开头，后面部分为按钮功能的说明，这样方便记忆和识别；

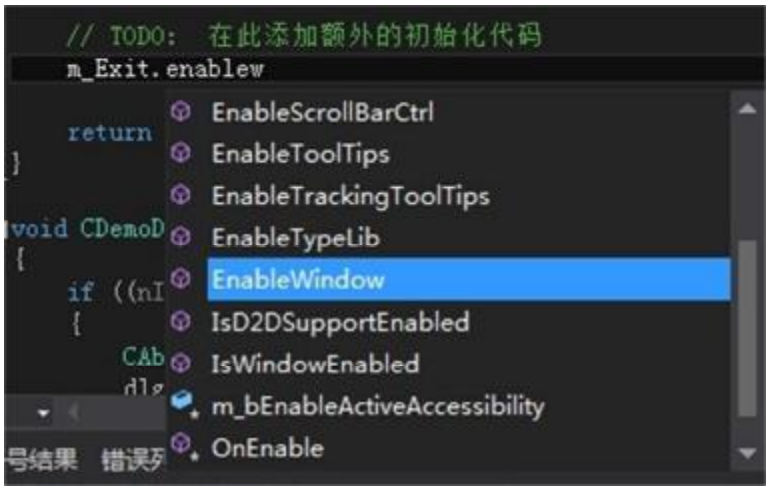
   2、按钮只有控件型的变量；

   3、如果有必要的话可以添加一定的注释。

2. 首先修改对话框初始化函数 **OnInitDialog**，添加一条语句：

   m_Exit.EnableWindow(false);//使“退出”按钮不可用

   说明：**Visual studio** 编程过程中只要输入几个字母就自动提示相关的函数或者变量，可以快速选取想要的函数和变量，有时候不确定函数或者变量名称，可以把记得的部分输入然后在提示中找到所需的函数或者变量。



3. 对两个按钮都添加按钮按下事件的处理函数，“使能”按钮用来使能退出按钮是否可用，退出按钮用于退出程序。

   void CDemoDlg::OnBnClickedEnable()

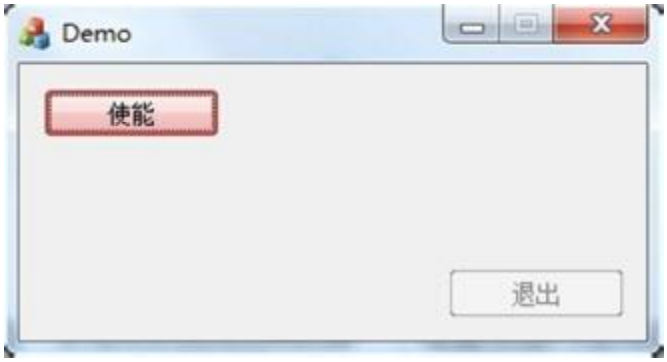
   {

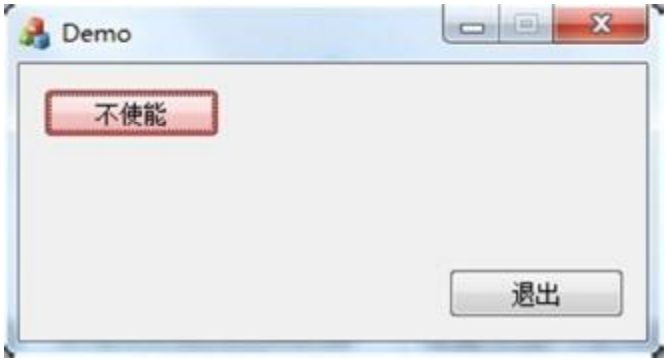
```
// TODO:  在此添加控件通知处理程序代码
CString str;
m_Enable.GetWindowTextW(str);
if (str == "使能")
{
m_Exit.EnableWindow(true);
m_Enable.SetWindowTextW(_T("不使能"));
}
else
{
m_Exit.EnableWindow(false);                    //不使能退出按钮
m_Enable.SetWindowTextW(_T("使能"));     //修改按钮文本
}
}
void CDemoDlg::OnBnClickedExit()
{
// TODO:  在此添加控件通知处理程序代码
SendMessage(WM_CLOSE,0,0);
}
```

```
void CDemoDlg::OnBnClickedEnable()
{
    // TODO:  在此添加控件通知处理程序代码
    CString str;
    m_Enable.GetWindowTextW(str);
    if (str == "使能")
    {
        m_Exit.EnableWindow(true);
        m_Enable.SetWindowTextW(_T("不使能"));
    }
    else
    {
        m_Exit.EnableWindow(false);
        m_Enable.SetWindowTextW(_T("使能"));
    }
}

void CDemoDlg::OnBnClickedExit()
{
    // TODO:  在此添加控件通知处理程序代码
    SendMessage(WM_CLOSE,0,0);
}
```


4. 测试程序，点击“本地 **Windows** 调试器”，或者点击 **F5** 生成应用程序，然后测试按下使能，退出按钮就可以操作，点击退出按钮退出程序。

   这样一个简单的按钮实例就完成了，希望对 **MFC** 的初学者有所帮助。





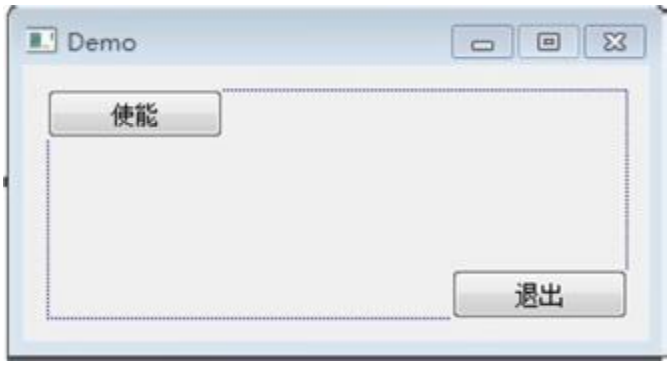
END

注意事项

- 合理对变量和函数进行命名，养成好的习惯对编程大有裨益。

VS2013/MFC 基于对话框编程：[6]手动删除控件

对于软件开发，很多时候需要修改软件界面，也许原本添加的控件不再需要了，需要彻底删除。如果只是简单的将控件从设计界面 **delete** 的话，是无法编译通过的，特别是控件对应着变量以及函数的情况下。这里我将以按钮控件为例介绍手动彻底删除控件的方法。

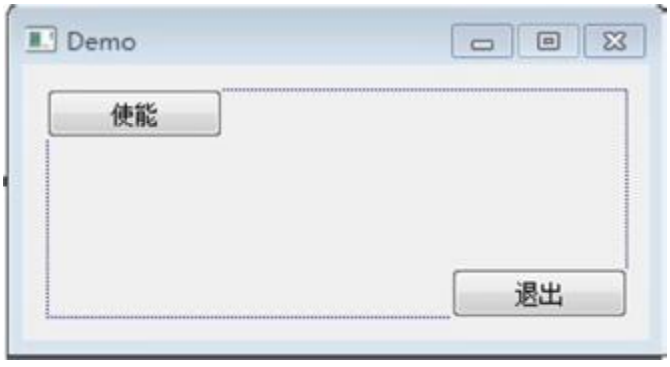


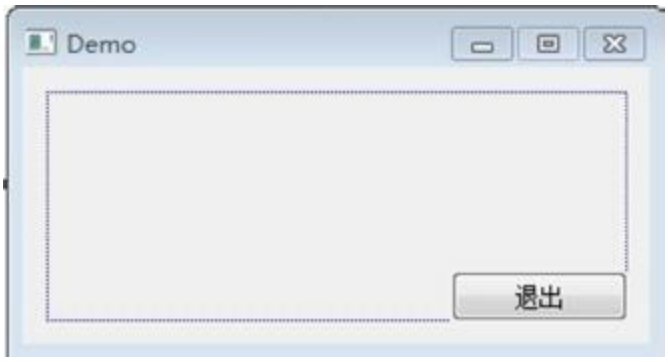
工具/原料

- **Visua Studio 2013**

方法一：手动删除

1. 首先确定按钮有哪些相关的变量和函数：以对话框中原有的“使能”按钮为例，对应着一个控件变量和一个按钮按下事件处理函数。然后选中按钮按 **delete** 删除对话框中的按钮。

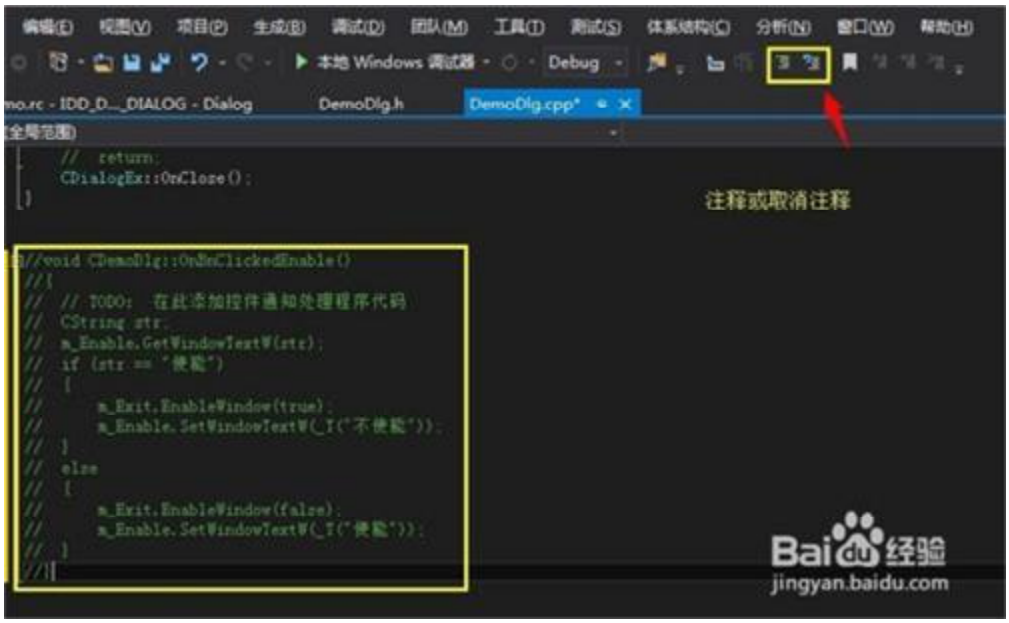




2. 打开对话框的头文件，找到原来通过类向导生成的变量定义和函数定义，将其注释掉或者直接删除。

   说明：类向导生成的函数和变量定义都在头文件的最后一个“**public**”栏中。

3. 打开对话框的源文件，找到按钮的处理函数定义，将其注释或者删除。



4. 接下来找到消息映射，可以看到按钮按下消息的那条语句底部有波浪线标志，将改语句删除。

   这样函数部分就删除完了。

```
BEGIN_MESSAGE_MAP(CDemoDlg, CDialogEx)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_WM_CLOSE()
    ON_BN_CLICKED(IDC_Enable, &CDemoDlg::OnBnClickedEnable)
    ON_BN_CLICKED(IDC_Exit, &CDemoDlg::OnBnClickedExit)
END_MESSAGE_MAP()
```

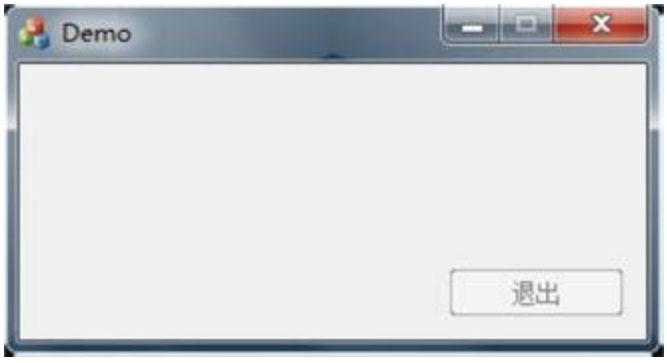
5. 5

最后删除变量的数据交换语句，找到 **DoDataExchange** 函数，默认就在消息映射的上面。将底部有波浪标志的语句删除。

```
void CDemoDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialogEx::DoDataExchange(pDX);
    DDX_Control(pDX, IDC_Exit, m_Exit);
    DDX_Control(pDX, IDC_Enable, m_Enable);
}
```

6. 6

所有步骤都完成后重新编译生成应用程序，可以正常编译通过。



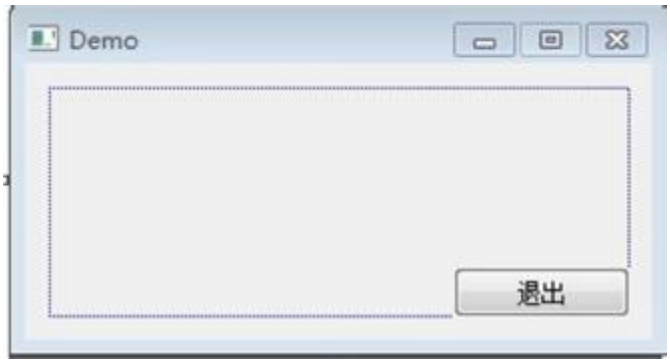
END

注意事项

- 下篇经验将讲述通过类向导删除控件

VS2013/MFC 基于对话框编程：[7]向导删除控件

手动删除控件的方法比较麻烦一点，需要清楚的知道变量定义的位置以及调用的位置，还有函数声明的位置，消息映射中的相关内容，以及函数调用的位置。所以为了减少劳动量可以通过类向导删除控件的变量和函数。

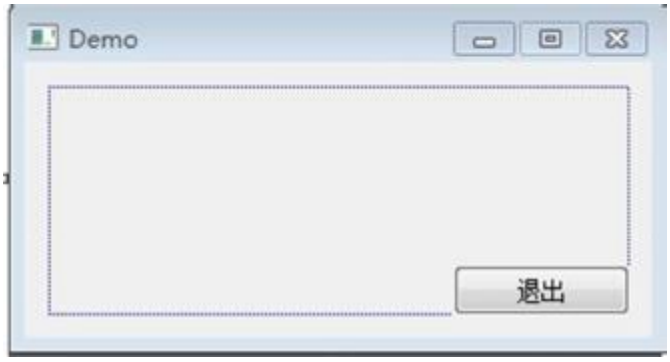


工具/原料

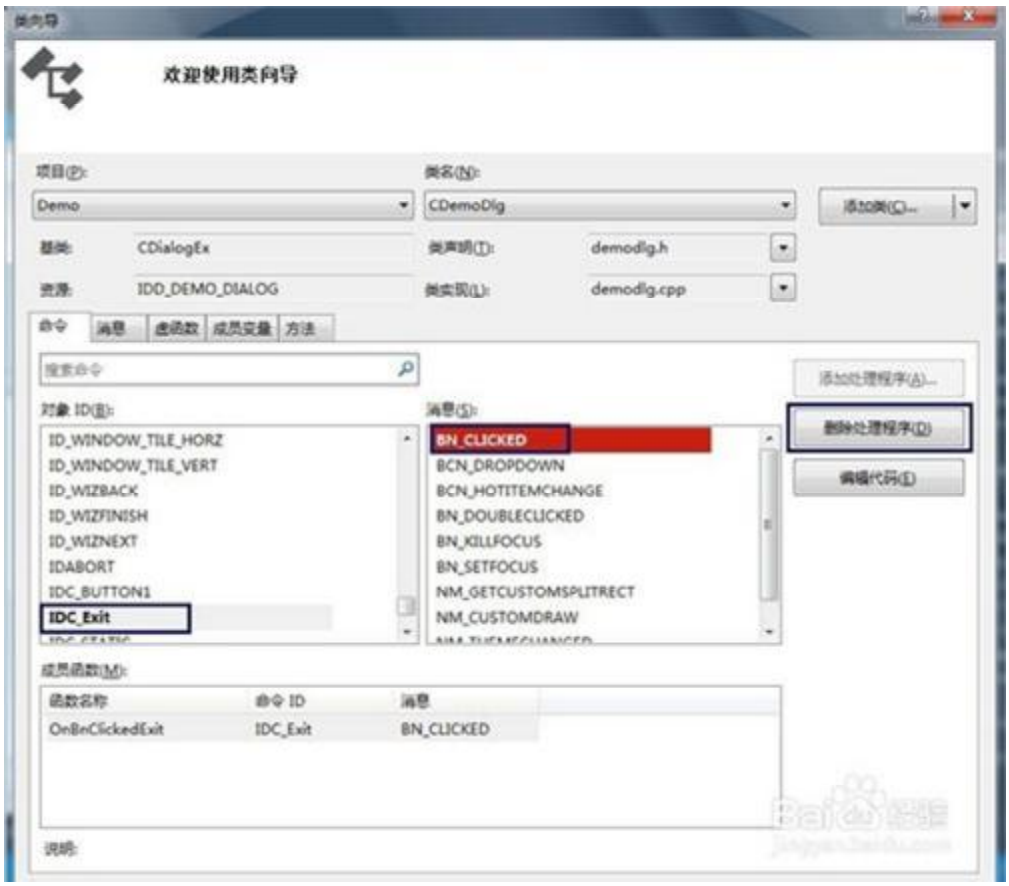
- Visual Studio 2013

方法/步骤

1. 确定需要删除的控件，以及控件对应的变量和函数。对话框仅存的一个退出按钮关联着一个控件变量和按钮按下事件处理函数。



2. 对按钮右键“类向导”，或者直接从菜单“项目”中找到类向导，并找到按钮对应的处理函数，选中事件函数后，点击“删除处理程序”。



3．转到成员变量选项卡，选中需要删除的控件变量，点击“删除变量”。

4．类向导的删除工作完成后，可以查看程序，你会发现变量的定义、函数的声明和定义、数据交换函数的相关语句、消息映射中的相关信息都被注释掉了。

```
//  afx_msg void OnBnClickedExit();
//  CButton m_Exit;
```

```
void CDemoDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialogEx::DoDataExchange(pDX);
    //  DDX_Control(pDX, IDC_Exit, m_Exit);
}

BEGIN_MESSAGE_MAP(CDemoDlg, CDialogEx)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_WM_CLOSE()
//  ON_BN_CLICKED(IDC_Exit, &CDemoDlg::OnBnClickedExit)
END_MESSAGE_MAP()
```

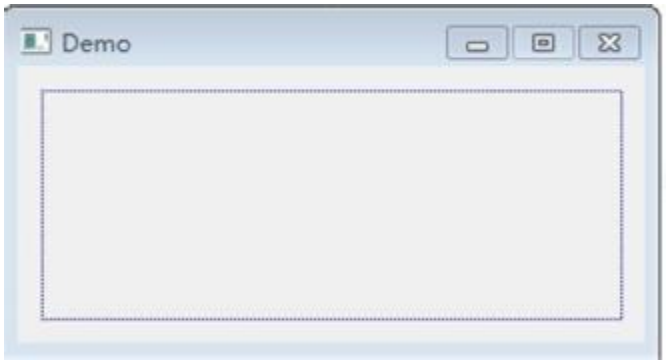
```
//void CDemoDlg::OnBnClickedExit()
//{
//  // TODO:  在此添加控件通知处理程序代码
//  SendMessage(WM_CLOSE,0,0);
//}
```

5. 通过以上步骤，只是将类向导生成的定义部分注释掉了，但难保开发者在别的函数中调用了该控件的变量或者函数。

   如果开发者清楚知道调用位置，可以直接找到后进行删除和修改，如果不知道也没关系，直接调试程序，编译过程中让软件自动发现错误。

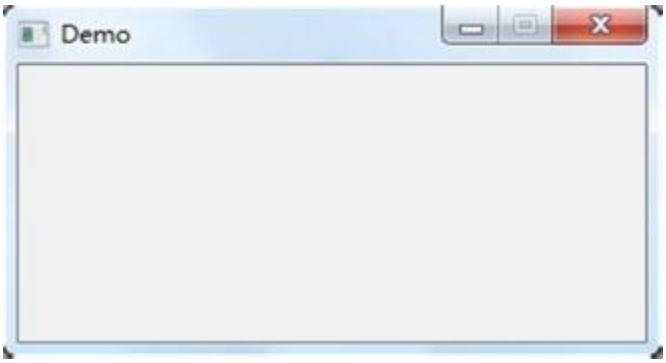
6. 编译出错后选择“否”，在错误列表中双击第一个错误选项，因为往往后面的错误都是由前面错误引起的先不用管。双击后会自动跳转到错误语句处，将相关语句删除或者修改就可以了。修改完再编译继续查错，直到没错为止。

7. 最后别忘了在对话框中将按钮 **delete**，再生成应用程序。



VS2013/MFC 基于对话框编程：[8]固定对话框

默认生成的对话框是可以通过鼠标拖动边缘从而改变大小的，然而很多时候我们并不需要对话框的大小被改变，这就需要修改相关属性来达到这个目的。

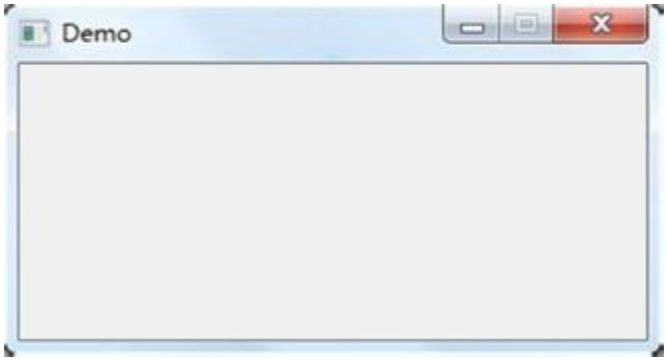


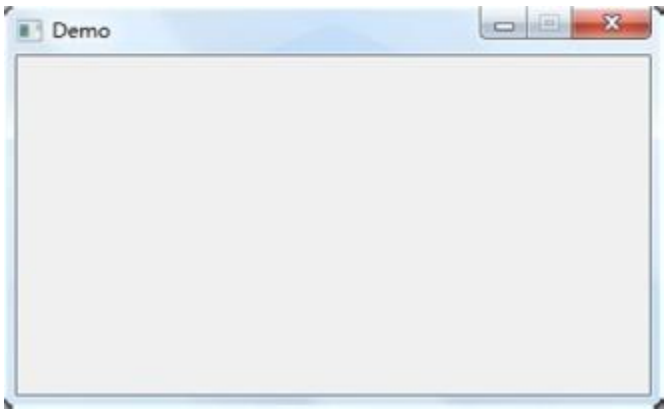
工具/原料

- Visual Studio 2013

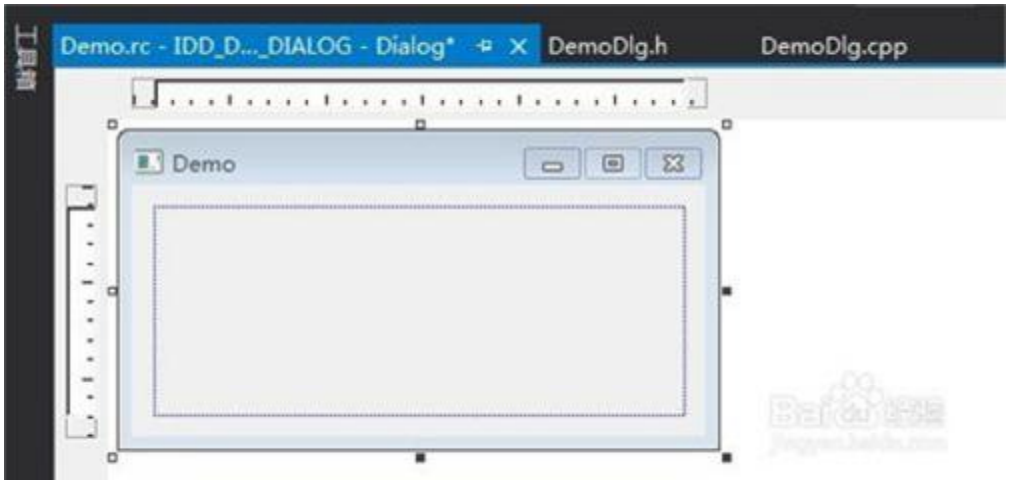
方法/步骤

1. 对于已经生成好的项目，可以通过鼠标拖动进行放大缩小。

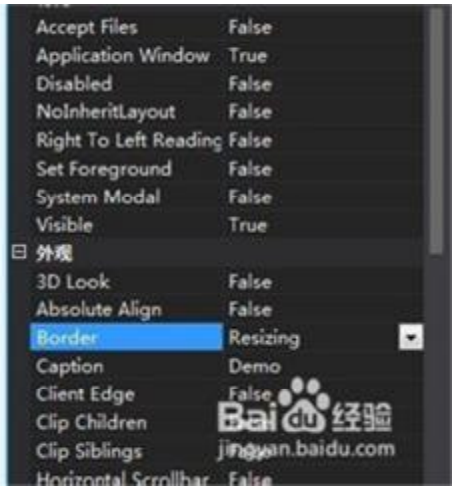




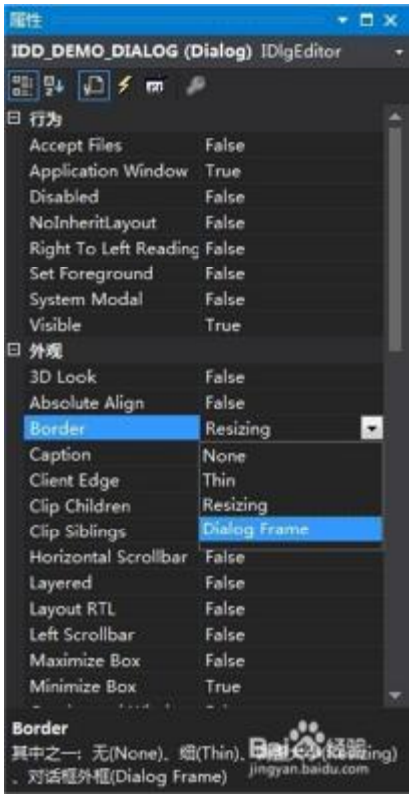
2. 打开对话框设计界面，选中对话框，并打开属性页。



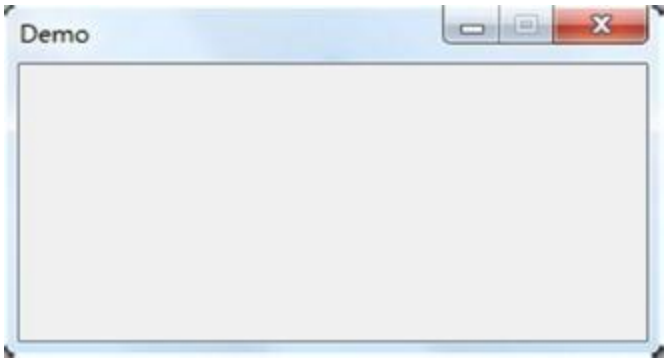
3. 在属性页的外观选项中找到“Boder”项，可以看到默认的属性值是“Resizing”，表示对话框大小可以被改变。



4. 将“Boder”属性修改为最后一项“Dailog Frame”，这样就把对话框的框架固定了。

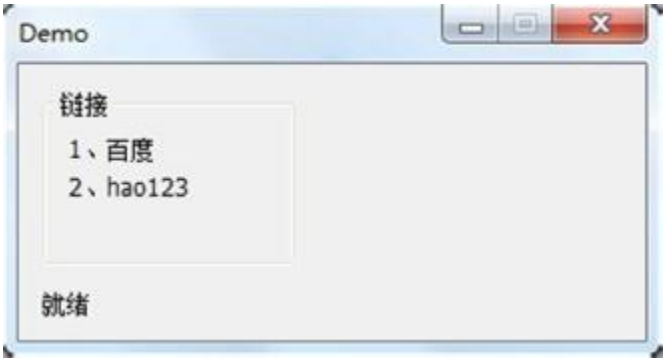


5. 重新测试对话框，看是否可以修改大小，可以发现，鼠标放在对话框边缘时无法缩放对话框。



VS2013/MFC 基于对话框编程：[9]文本超链接

静态文本（**static text**）作为对话框的常用控件之一，一般情况下起着指示说明的作用，让用户明白对话框中的相关信息和功能，这种情况一般不关联点击事件。但有时候需要通过文本来打开某个超链接，比如说某个网址，这时候就需要添加相关函数进行实现。

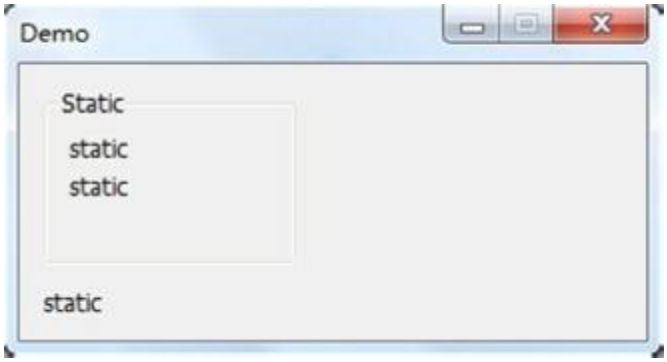


工具/原料

- **Visual Studio 2013**

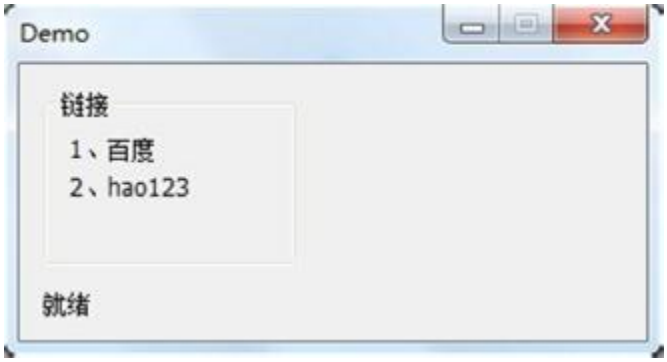
方法/步骤

1. 打开创建好的 **Demo** 项目，在对话框中添加一个 **group box**，**3** 个 **static text**。



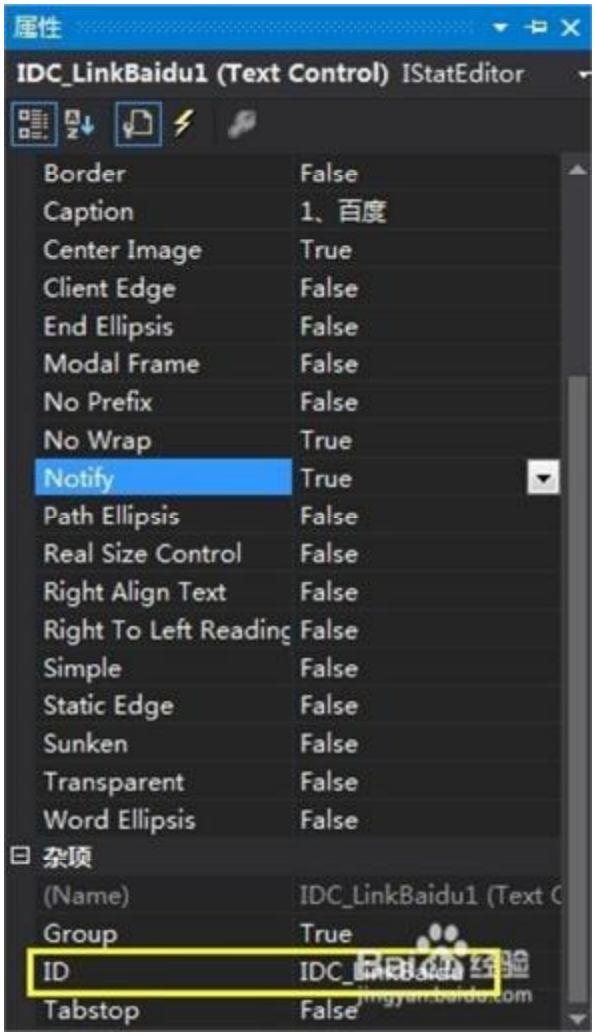
2. 修改所有控件的 **Caption** 属性，分别改为“链接”、“百度”、“**hao123**”以及“就绪”。

   最后一个文本相当于状态栏，用于指示当前状态。



3. 修改“百度”文本的 **ID** 为 **IDC_LinkBaidu**；“**hao123**”文本的 **ID** 为 **IDC_LinkHao123**；状态指示文本 **ID** 为 **IDC_Toast**。

   此外，需要修改的最重要的一条属性是 **Notify**，需要点击后打开超链接的文本属性中，修改 **Notify** 为 **true**，否则无法响应鼠标点击事件。



4. 给状态栏文本添加 **CStatic** 类型的变量 **m_toast**。

5. 分别双击“百度”文本和“**hao123**”文本，自动生成鼠标点击事件的处理函数。

```
void CDemoDlg::OnStnClickedLinkbaidu()
{
    // TODO:  在此添加控件通知处理程序代码
}

void CDemoDlg::OnStnClickedLinkhao123()
{
    // TODO:  在此添加控件通知处理程序代码
}
```

6. 修改两个函数，通过 **ShellExecute** 函数打开超链接，通过 **SetWindowTextW** 函数修改状态栏信息。

   ShellExecute(0, NULL, _T("这里填写网址**"), NULL, NULL, SW_NORMAL);**

   GetDlgItem(IDC_LinkBaidu)->SetWindowTextW(_T("你好百度！**"));**

   m_toast.SetWindowTextW(_T("已打开百度网页！**"));**

```
void CDemoDlg::OnStnClickedLinkbaidu()
{
    // TODO:  在此添加控件通知处理程序代码
    ShellExecute(0, NULL, _T("http://www.baidu.com"), NULL, NULL, SW_NORMAL);
    GetDlgItem(IDC_LinkBaidu)->SetWindowTextW(_T("你好百度！"));
    m_toast.SetWindowTextW(_T("已打开百度网页！"));
}


void CDemoDlg::OnStnClickedLinkhao123()
{
    // TODO:  在此添加控件通知处理程序代码
    ShellExecute(0, NULL, _T("http://www.hao123.com"), NULL, NULL, SW_NORMAL);
    GetDlgItem(IDC_LinkHao123)->SetWindowTextW(_T("你好123！"));
    m_toast.SetWindowTextW(_T("已打开hao123网页！"));
}
```


7. 生成应用程序并测试功能，分别点击两个文本，可以分别打开两个网页，同时在状态栏中更新提示。

END

注意事项

- 别忘了修改静态文本的 **Notify** 属性

VS2013/MFC 基于对话框编程：[10]处理多个事件

编写一个应用软件很多时候需要用到多个相同的控件，比如按钮，如果许多按钮的功能大体相似，比如计算器中的加、减、乘、除或者数字 **0-9**，它们的按下事件是类似的。

如果逐个生成按钮按下事件处理函数，就显得冗余和麻烦，我们可以通过手动添加消息映射实现一个函数响应多个控件消息。

工具/原料

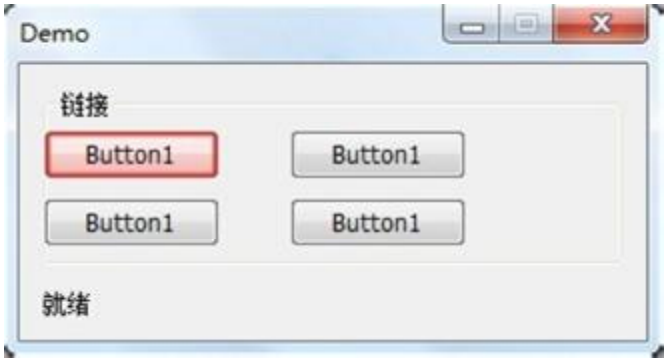
- **Visual Studio 2013**

方法/步骤

1. 1

在对话框中添加 **4** 个按钮和 **1** 个静态文本，注意按钮添加过程中不要添加别的控件，保证 **4** 个按钮的 **ID** 编号连续。

功能：不同按钮按下时修改静态文本，显示不同的信息。



2. 2

1、 修改按钮属性，特别是第一个和最后一个添加的按钮，最好修改其 **ID** 名称便于识别，这里默认为 **IDC_BUTTON1~IDC_BUTTON4.**

2、 修改静态文本 **ID** 为 **IDC_Toast**，添加变量 **CStatic m_toast;**

3. 3

在项目头文件中添加函数（函数名称可以修改）：

afx_msg void OnBnClickedXXX(UINT nID);

参数 **nID**：触发消息时对应的控件 **ID**。

```
public:
    afx_msg void OnClose();
    afx_msg void OnBnClickedXXX(UINT nID);
    CStatic m_toast;
```

4. 4

在源文件的消息映射中添加：

ON_CONTROL_RANGE(BN_CLICKED, IDC_BUTTON1, IDC_BUTTON4, OnBnClickedXXX)

参数 **1**：消息类型；

参数 **2**：第一个控件 **ID**；

参数 **3**：最后一个控件 **ID**；

参数 **4**：响应函数

```
BEGIN_MESSAGE_MAP(CDemoDlg, CDialogEx)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_WM_CLOSE()
    ON_CONTROL_RANGE(BN_CLICKED, IDC_BUTTON1, IDC_BUTTON4, OnBnClickedXXX)
END_MESSAGE_MAP()
```

5. 5

在源文件中实现函数 **OnBnClickedXXX(UINT nID)**，可以通过 **nID** 减去第一个按钮 **ID** 确定当前按下按钮属于第几个按钮。接下来可以通过两种方法实现文本信息的修改：

方法一：通过 **switch** 语句判断按钮编号，执行不同的程序。

方法二：定义 **CString** 变量，通过 **Format** 定义字符串格式，将按钮信息包含在字符串中，最后通过 **m_toast** 更新界面。

```
void CDemoDlg::OnBnClickedXXX(UINT nID)
{
    int ID = nID - IDC_BUTTON1;
    CString str;
    //str.Format(_T("%s%i%s"), _T("按钮"), ID + 1, _T("按下！"));
    //m_toast.SetWindowTextW(str);
    switch (ID)
    {
    case 0:m_toast.SetWindowTextW(_T("按钮1按下！")); break;
    case 1:m_toast.SetWindowTextW(_T("按钮2按下！")); break;
    case 2:m_toast.SetWindowTextW(_T("按钮3按下！")); break;
    case 3:m_toast.SetWindowTextW(_T("按钮4按下！")); break;
    }
}
```

测试程序功能，启动调试，分别按下四个按钮，查看静态文本信息的更新。怎么样，通过一个函数处理多个事件是不是可以节省不少代码，而且更加简洁明了。

注意事项

- 注意按顺序添加相同控件，否则控件编号不连续进而导致调用出错。

VS2013/MFC 基于对话框编程：[11]编辑框

编辑框（**Edit Control**）作为对话框中常用的控件之一，常用来输入文本或者显示文本，比如用户名和密码的输入，当前数据的显示等等都少不了编辑框。其实 **win7** 自带的记事本就是一个编辑框，这里我将介绍编辑框的一些常用用法。

工具/原料

- **Visual Studio 2013**

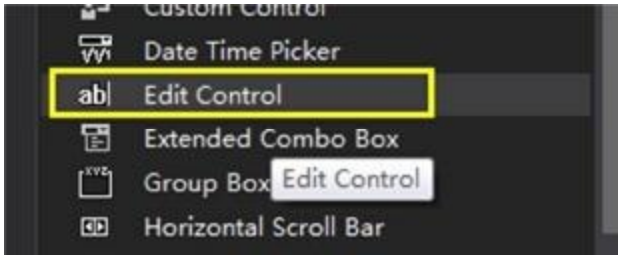
方法/步骤

1. 1

从工具箱中找到编辑框（**Edit Control**），拖动到对话框中，按图所示放置多个编辑框和一个按钮，用于说明编辑框的不同作用。

应用程序的功能：起初计算按钮不可用，需要输入正确密码，按回车确认，状态框提示密码正确与否，密码正确后启用按钮；

输入两个加数，点击按钮计算两数之和并在编辑框中显示结果，同时在记录框中显示计算记录。



2. 2

首先选中任意一个编辑框，看看编辑框都有哪些属性，其中常用到属性有：

Multiline：多行，表示内容可以多行显示，一般记录性的编辑框需要多行；

Password：密码，表示文本以密码形式呈现，一般用来输出密码；

Read only：只读，表示文本内容只能读不能写；

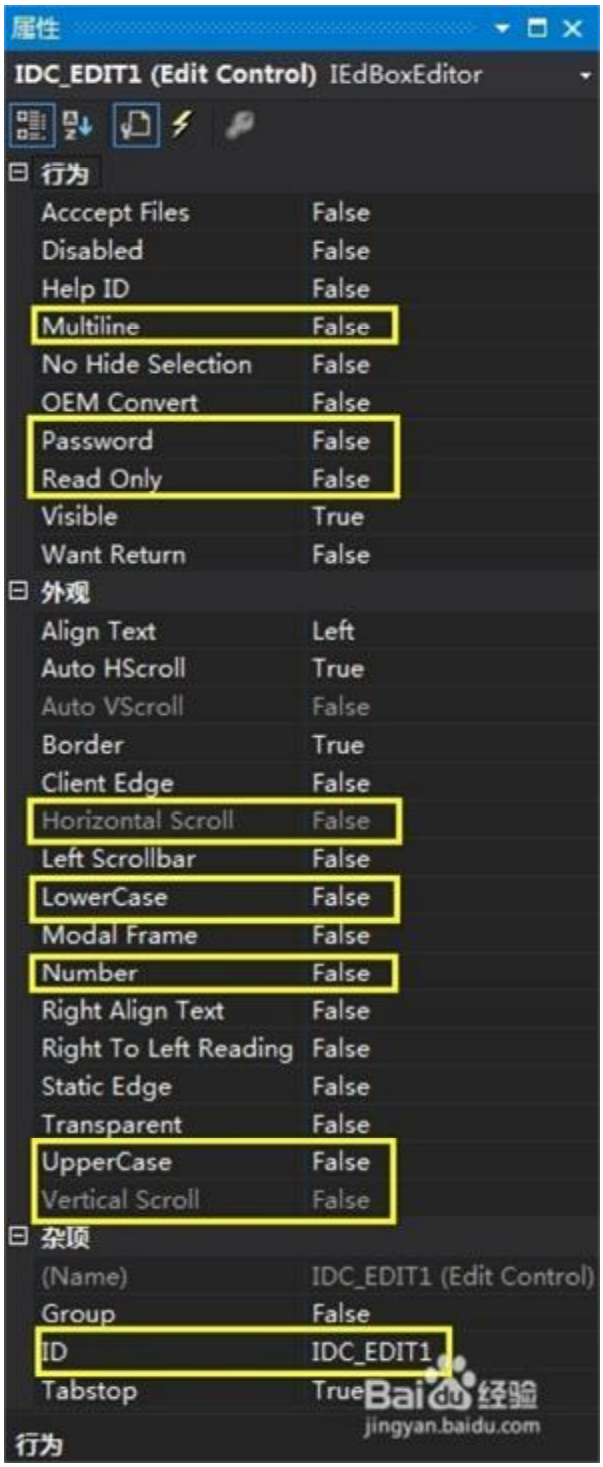
Number：数字，表示只能输入数字；

Lowercase：小写，所有字母全部自动转换为小写显示；

Uppercase：大写，所有字母全部自动转换为大写显示；

Horizontal scroll：水平滚动条，需要先选中 **Multiline** 属性；

Vertical scroll：垂直滚动条，需要先选中 **Multiline** 属性；



3. 3

根据不同编辑框需要实现的功能不一样，分别修改编辑框的各个属性：

1、两个加值编辑框修改 **Number** 一个为真、一个为假；

2、密码输入框修改 **Password** 为真；

3、数据之和显示框修改 **Read only** 为真；

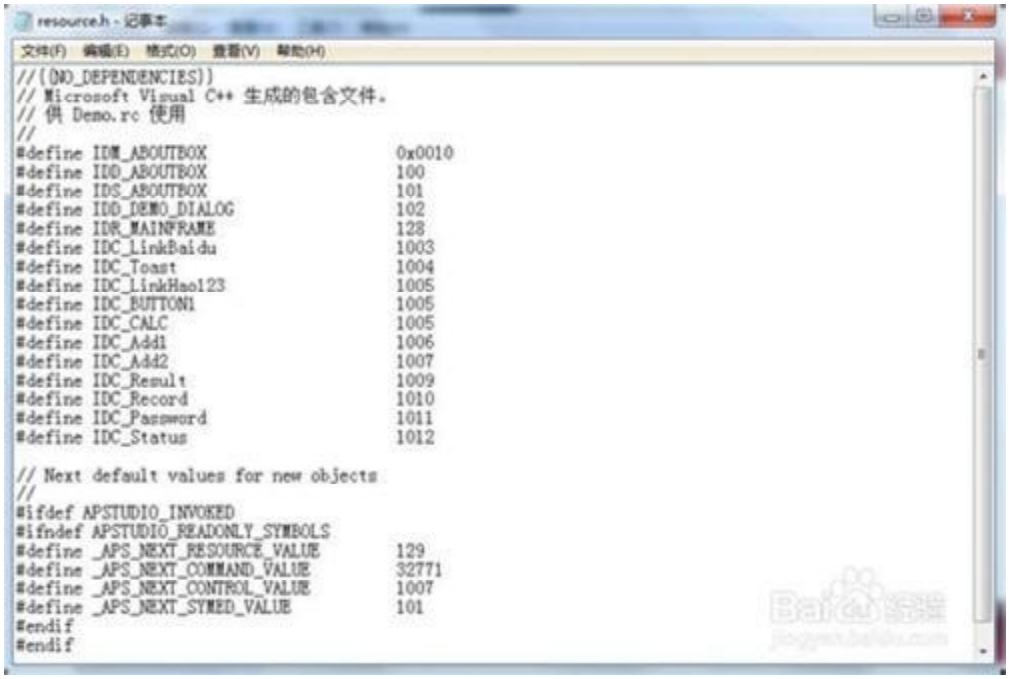
4、状态提示框修改 **Read only** 为真，**Uppercase** 为真；

5、记录编辑框修改 **Read only** 为真，**Multiline** 为真，**Vertical scroll** 为真；

4. 4

修改各个控件的 **ID** 分别为 **IDC_Add1**、**IDC_Add2**、**IDC_Result**、**IDC_Record**、**IDC_Password**、**IDC_Status**，按钮 **ID** 为 **IDC_CALC**；从 **resource** 文件中可以查看控件 **ID** 的定义。

双击按钮生成按钮按下事件处理函数。



5. 5

通过类向导为各个控件添加变量:

1、给两个加数以及加数之和添加 **double** 型的变量 **m_add1,m_add2,m_result;**

2、给记录框添加控件类型的 **CEdit m_record**；

3、给密码输入框添加字符串类型的变量 **CString m_passWord**；

4、状态框不添加变量，以便讲述如何通过 **ID** 直接访问控件。

6. 6

修改初始化函数 **OnInitDialog**，添加语句：

m_record.SetWindowTextW(_T("请输入密码！\n"));

GetDlgItem(IDC_CALC)->EnableWindow(false);

这样就设置了提示语句，并将计算按钮设置为不可用，**GetDlgItem** 函数通过 **ID** 获取控件，

这样就不必添加控件变量。

```
    // TODO:  在此添加额外的初始化代码
    m_record.SetWindowTextW(_T("请输入密码！\n"));
    GetDlgItem(IDC_CALC)->EnableWindow(false);
    return TRUE;  // 除非将焦点设置到控件，否则返回 TRUE
}
```

7. 7

类向导，添加虚函数 **PreTranslateMessage**。

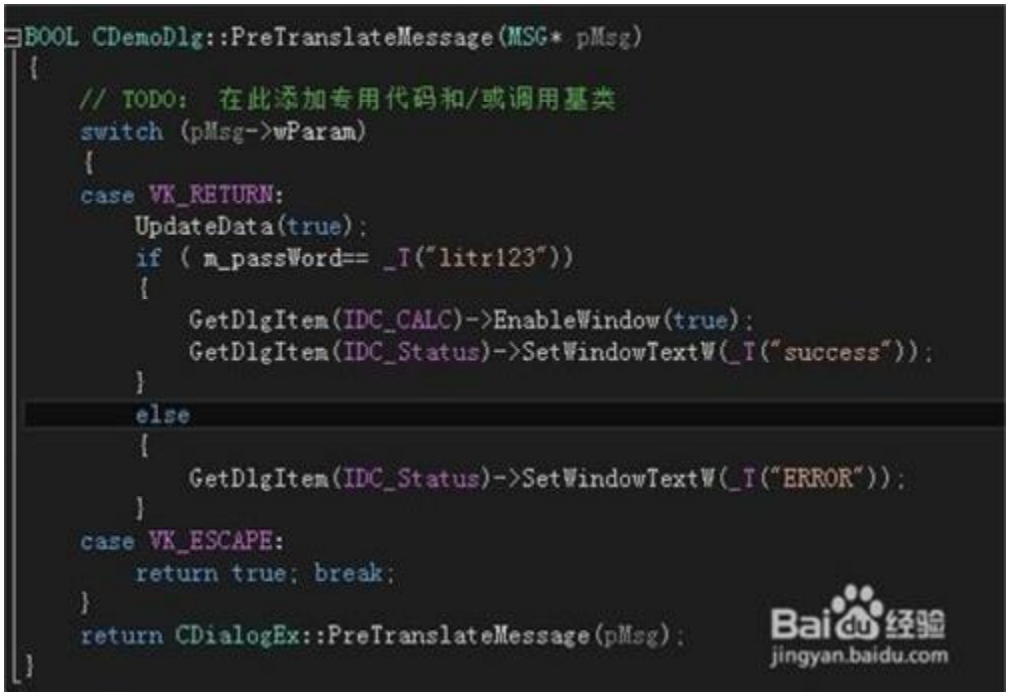
8. 8

修改虚函数：

```
BOOL CDemoDlg::PreTranslateMessage(MSG* pMsg)

{

// TODO:  在此添加专用代码和/或调用基类

switch (pMsg->wParam)

{

case VK_RETURN:

UpdateData(true);

if ( m_passWord== _T("litr123"))

{

GetDlgItem(IDC_CALC)->EnableWindow(true);

GetDlgItem(IDC_Status)->SetWindowTextW(_T("success"));
```

```
}
else
{
GetDlgItem(IDC_Status)->SetWindowTextW(_T("ERROR"));
}
case VK_ESCAPE:
return true; break;
}
return CDialogEx::PreTranslateMessage(pMsg);
}
```

函数功能：一来可以防止按下回车或者 **ESC** 按键时退出程序，二来可以判断密码是否正确并更新状态内容。



9. 接下来修改按钮按下事件处理函数，实现两数相加并更新记录列表。

```
void CDemoDlg::OnBnClickedCalc()
{
```

```
// TODO:  在此添加控件通知处理程序代码
UpdateData(true);            // 获取数据
m_result = m_add1 + m_add2;
UpdateData(false);           // 更新数据
CString str;
str.Format(_T("%g %s %g %s %g"), m_add1, _T("+"), m_add2, _T("="),
m_result);                   // 数据显示格式
str += _T("\r\n");           // 回车换行
int lastLine = m_record.LineIndex(m_record.GetLineCount() - 1);
m_record.SetSel(lastLine + 1, lastLine + 2, 0);
m_record.ReplaceSel(str);  // 在最后一行添加新的内容
}
```

10. 测试程序运行结果，看看编辑框不同属性会呈现什么不同的效果。

当记录框不够用时会自动添加滚动条。

注意事项

- 注意获取数据和更新数据时需要采用 **UpdateData** 函数
- 滚动条的加入需要配合 **Multiline** 属性

VS2013/MFC 基于对话框编程：[12]单选按钮

单选按钮（**Radio Button**）常用于多选一的情况，比如试卷的选择题，抽样调查的选项等等，但在实际中一般用来选择不同的设置选项以达到不同的目的，本经验通过简单的例子讲述如何使用单选按钮。

工具/原料

- **Visual Studio 2013**

方法/步骤

1. 1

新建项目，打开对话框设计界面，在对话框中添加若干控件：

3 个编辑框，**1** 个按钮，两组单选组合，若干个静态文本。

功能：两个编辑框选择加减乘除运算，并将结果送入第三个编辑框显示，计算按钮通过第二组单选按钮组选择启用或者禁用。

2. 2

修改属性，包括 **ID** 和 **Caption**，以及编辑框的只读属性等。

按钮 **ID** 为 **IDC_CALC**，编辑框 **ID** 为 **IDC_Add1**、**IDC_Add2**、**IDC_Result**；

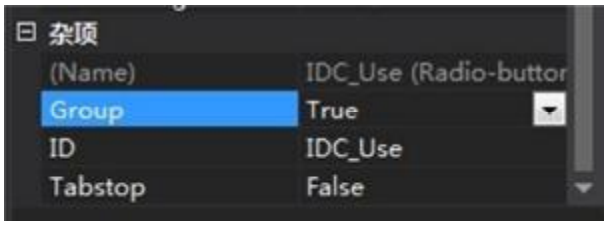
算法组的单选按钮 **ID** 为 **IDC_Plus**、**IDC_Minus**、**IDC_Multi**、**IDC_Div**；

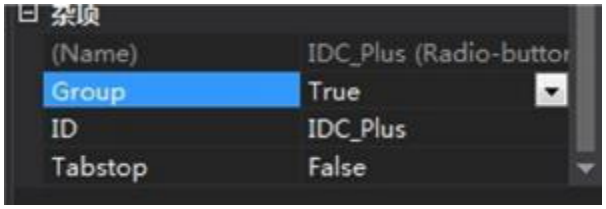
启用组的单选按钮 **ID** 为 **IDC_Use**、**IDC_NUse**。

两编辑框之间的静态文本 **ID** 为 **IDC_Algorithm**，表示当前选用的算法。

3. 3

最重要的一步：修改单选按钮的 **Group** 属性，本例程共有两组单选按钮，需要设定两个组，每一组的第一个单选按钮的 **Group** 属性需要为真。





4. 4

添加变量和函数：三个编辑框分别添加 **Double** 型的变量 **m_add1**、**m_add2**、**m_result**。

第一组单选按钮第一个按钮添加 **int m_plus** 第二组单选按钮第一个按钮添加 **BOOL m_use**；

IDC_Algorithm 静态文本对应变量 **CStatic m_algroithm**；

双击计算按钮添加按钮按下事件处理函数。

修改初始化函数 **OnInitDialog** 添加：

GetDlgItem(IDC_CALC)->EnableWindow(false);

5. 5

分别双击“启用”和“禁用”自动生成函数，添加代码启用或禁用“计算”按钮，这样做就不需要定义 **m_use** 变量了，但是如果别的地方需要判断到底有没有启用，就得通过变量判断了。

```
void CDemoDlg::OnBnClickedUse()
{
    // TODO:  在此添加控件通知处理程序代码
    GetDlgItem(IDC_CALC)->EnableWindow(true);
}


void CDemoDlg::OnBnClickedNuse()
{
    // TODO:  在此添加控件通知处理程序代码
    GetDlgItem(IDC_CALC)->EnableWindow(false);
}
```

6. 6

通过单选选择“加减乘除”算法，这里采用单个函数处理多个事件的方式编写程序，在头文件中声明：

afx_msg void OnBnClickedAlgor(UINT nID);

源文件中添加消息映射：

ON_CONTROL_RANGE(BN_CLICKED, IDC_Plus, IDC_Div, OnBnClickedAlgor)

并实现 **OnBnClickedAlgor** 函数。

这一步可以参考本系列经验第 **10** 篇“处理多个事件”。

当然也可以逐个双击单选按钮生成 **4** 个函数分别处理，不过那样的话就显得很麻烦。

```
BEGIN_MESSAGE_MAP(CDemoDlg, CDialogEx)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_WM_CLOSE()
    ON_BN_CLICKED(IDC_CALC, &CDemoDlg::OnBnClickedCalc)
    ON_BN_CLICKED(IDC_Use, &CDemoDlg::OnBnClickedUse)
    ON_BN_CLICKED(IDC_NUse, &CDemoDlg::OnBnClickedNuse)
    ON_CONTROL_RANGE(BN_CLICKED, IDC_Plus, IDC_Div, OnBnClickedAlgor)
END_MESSAGE_MAP()
```

```
void CDemoDlg::OnBnClickedAlgor(UINT nID)
{
    CString str;
    switch (nID - IDC_Plus)
    {
    case 0:str = _T("+"); break;
    case 1:str = _T("-"); break;
    case 2:str = _T("×"); break;
    case 3:str = _T("÷"); break;
    }
    m_algorithm.SetWindowTextW(str);
}
```

7. 7

修改计算函数，通过变量 **m_plus** 判断运算符，实现两数的加减乘除运算。

```
void CDemoDlg::OnBnClickedCalc()
{
    // TODO:  在此添加控件通知处理程序代码
    UpdateData(true);
    switch (m_plus)
    {
    case 0: m_result = m_add1 + m_add2; break;
    case 1: m_result = m_add1 - m_add2; break;
    case 2: m_result = m_add1 * m_add2; break;
    case 3: m_result = m_add1 / m_add2; break;
    }
    UpdateData(false);
}
```

8. 8

测试程序功能，启动调试，默认情况计算按钮不可用，选择启用后方可使用，然后选择运算符，点击计算得到结果，随时点击禁用都会使计算按钮不可用。

END

注意事项

- 单选按钮最重要的属性就是 **Group**，需要在同一组的按钮必须 **ID** 连续，这就要求添加同组按钮时连续。

VS2013/MFC 基于对话框编程：[13]复选框

复选框（Check Box）作为对话框中的常用控件之一，通常用来使能某种功能或选项，一个复选框选中与否表示某个功能的启用与否。本经验通过简单例程说明组合框的使用。

工具/原料

- Visual Studio 2013

方法/步骤

1. 新建项目，在对话框中添加三个复选框和四个编辑框，再加上一个按钮。

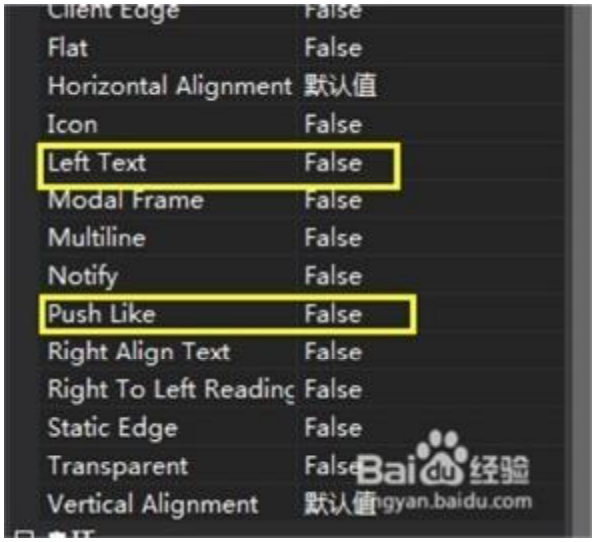
   程序功能：计算披萨选中项的总价。

2. 复选框的属性：有两项属性可以依情况选择。

   1、Left txet：将选框和文本的左右位置互换；

   2、Push like：将复选框的样式改为“按下”、“弹起”模式。

   本例使用默认的模式即可。



3. 修改 ID，三个复选框 ID 分别为 IDC_Pisa1、IDC_Pisa2、IDC_Pisa3；

   四个编辑框 ID 分别为 IDC_Num1、IDC_Num2、IDC_Num3、IDC_Result；

   按钮 ID 为 IDC_CALC。

```
#define IDC_Pisa1                       1016
#define IDC_Pisa2                       1017
#define IDC_Pisa3                       1018
#define IDC_Num1                        1019
#define IDC_Num2                        1020
#define IDC_Num3                        1021
```

4. 添加变量：

   三个编辑框的变量为 UINT 类型的 m_num1、m_num2、m_num3；

   三个复选框的变量为 CButton 类型的 m_pisa1、m_pisa2、m_pisa3；

5. 双击按钮生成按钮按下事件处理函数，并添加相关代码。

复选框常用函数

GetCheck()：获取复选框状态，判断是否被选中；

SetCheck()：设置复选框是否选中，1 表示选中，0 表示不选中。

```
void CDemoDlg::OnBnClickedCalc()
{
    // TODO:  在此添加控件通知处理程序代码
    UpdateData(true);
    m_result = 0;
    if (m_pisa1.GetCheck())
        m_result += m_num1 * 250;
    if (m_pisa2.GetCheck())
        m_result += m_num2 * 275;
    if (m_pisa3.GetCheck())
        m_result += m_num3 * 350;
    UpdateData(false);
}
```

6. 程序测试，启动调试，勾选好披萨类型以及数量，点击按钮进行计算。

VS2013/MFC 基于对话框编程：[14]定时器消息

定时器消息可以说是 **windows** 所有消息中最常用的消息，许多事件需要通过定时触发，比如最简单的秒表，还有工程软件中的定时采样等等都少不了定时器。本经验通过简单例子说明定时器的开启、响应和终止。

工具/原料

- **Visual Studio 2013**

方法/步骤

1. 新建 **Demo** 项目，打开对话框，添加两个按钮和一个静态文本，修改相应的 **Caption**。

   程序功能：按下计时按钮开始计时，将时间显示在静态文本中，复位按钮可以对文本复位全零并关闭定时器。

2. 修改 **ID**：计时按钮 **ID** 为 **IDC_Timer**，复位按钮 **ID** 为 **IDC_Reset**。

   静态文本 **ID** 为 **IDC_TimeDis**，并添加变量 **CStatic m_time**。

3. 双击计时按钮生成按钮按下事件处理函数，编辑函数，通过 **SetTimer** 函数启动定时器；

   参数 **1**：**nIDEvent**，定时器 **ID**；

   参数 **2**：**nElapse**，定时器定时时间，单位为毫秒；

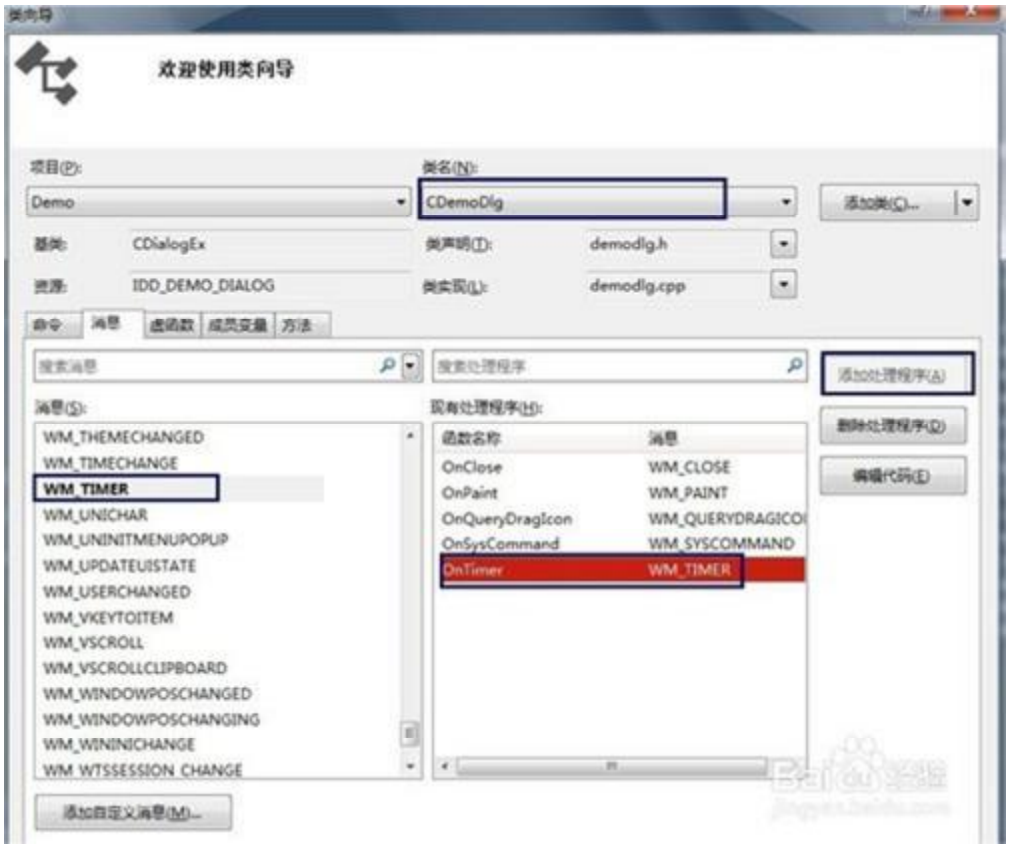
   参数 **3**：回调函数，设为 **NULL** 即可。

```
void CDemoDlg::OnBnClickedTimer()
{
    // TODO:  在此添加控件通知处理程序代码
    SetTimer(1, 10, NULL);
}
```

4. 双击复位按钮生成函数，编辑函数，通过 **KillTimer** 销毁定时器，只需输入一个参数（定时器 **ID**），同时将文本复位为“**00:00:00**”。

```
void CDemoDlg::OnBnClickedReset()
{
    // TODO:  在此添加控件通知处理程序代码
    KillTimer(1);
    m_time.SetWindowTextW(_T("00: 00: 00"));
}
```

5. 类向导，添加 **WM_TIMER** 消息处理函数。



6. 编辑定时器消息响应函数 **OnTimer**，判断定时器的 **ID**，如果只有一个定时器可以不用判断，但许多应用需要多个定时器，这时就适合 **switch** 语句进行判断。

   这里三个变量 **mm,ss,mss** 设为静态变量，否则每次进入都会重新赋值，不过如果将变量在头文件中定义成成员变量就不需要这样了。

```
void CDemoDlg::OnTimer(UINT_PTR nIDEvent)
{
    // TODO:  在此添加消息处理程序代码和/或调用默认值
    static UINT mm = 0;
    static UINT ss = 0;
    static UINT mss = 0;
    CString str;
    switch (nIDEvent)
    {
    case 1:
        mss++;
        if (mss == 100){ mss = 0; ss++; }
        if (ss == 60){ ss = 0; mm++; }
        str.Format(_T("%02i: %02i: %02i"), mm, ss, mss);
        m_time.SetWindowTextW(str);
        break;
    }
    CDialogEx::OnTimer(nIDEvent);
}
```


7. 本程序实现了定时器的启动、响应和终止，但是复位后重新计时时，那三个变量还是在原有基础上计数，相当于是暂停后重新开始。

如果需要重新开始计数，只要定义一个成员变量作为定时器销毁的标志，然后在定时器消息中判断标志位后对变量进行清零即可 ，清零后立马清除标志位。

END

注意事项

- 注意定时器的 **ID**，根据 **ID** 响应不同的功能。

VS2013/MFC 基于对话框编程：[15]自定义消息

有些时候光靠 **windows** 原有的消息是不够的，需要自定义消息来满足特定的功能，比如在与外部设备通讯时，如果接收到数据，就需要进行存储，但并没有直接的消息可以使用，需要自己定义。

当然自定义消息不局限于此，很多消息都可以通过自定义实现，本经验以简单例子进行说明自定义消息的创建和调用。

工具/原料

- **Visual Studio 2013**

方法/步骤

1. 1

打开 **Demo** 项目，在对话框中添加一个按钮，文本设置为“点击”。

程序功能：点击按钮，点击次数超过 **5** 次就会触发一个自定义消息，在自定义消息中弹出一个提示消息。

2. 2

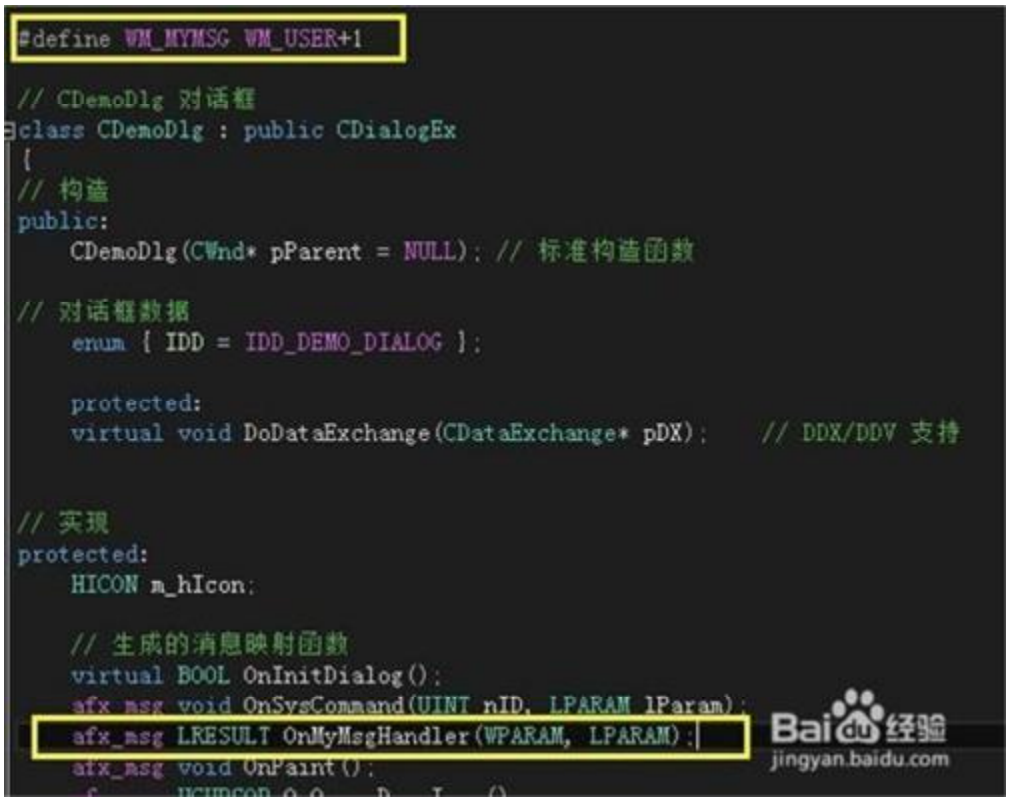
在头文件中添加：

#define WM_MYMSG WM_USER+1

WM_USER 以下的消息都是系统消息，所以自定义消息时，**ID** 要比 **WM_USE** 大；

添加函数声明（用于响应自定义消息）：

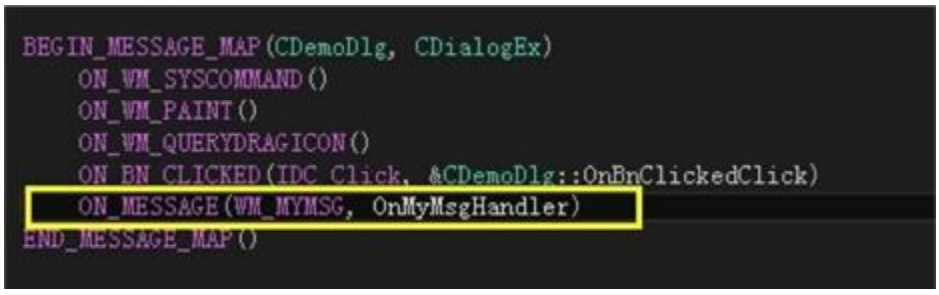
afx_msg LRESULT OnMyMsgHandler(WPARAM, LPARAM);



3. 3

在源文件的消息映射中添加：

ON_MESSAGE(WM_MYMSG,OnMyMsgHandler)



4. 4

添加一个宏定义：

#define IDC_New 2000　　// 数值不要与其他控件的 ID 重合

编写 **OnMyMsgHandler** 函数，提示时间到了，这个完全是测试，不代表具体意义。

```
LRESULT CDemoDlg::OnMyMsgHandler(WPARAM, LPARAM)
{
    //CButton btn;
    AfxMessageBox(_T("你的时间到了！"));
    return 0;
}
```

5. 5

修改按钮 **ID** 为 **IDC_Click**，双击生成按钮按下事件处理函数，编辑函数代码，

通过 **SendMessage** 函数触发消息。

```
void CDemoDlg::OnBnClickedClick()
{
    // TODO:  在此添加控件通知处理程序代码
    static int num = 0;
    CString str;
    str.Format(_T("%s%i%s"), _T("点击"), num+1, _T("次"));
    GetDlgItem(IDC_Click)->SetWindowTextW(str);
    num++;
    if (num == 5)
    {
        ::SendMessage(::AfxGetMainWnd()->m_hWnd, WM_MYMSG, 0, 0);
        num = 0;
    }
}
```

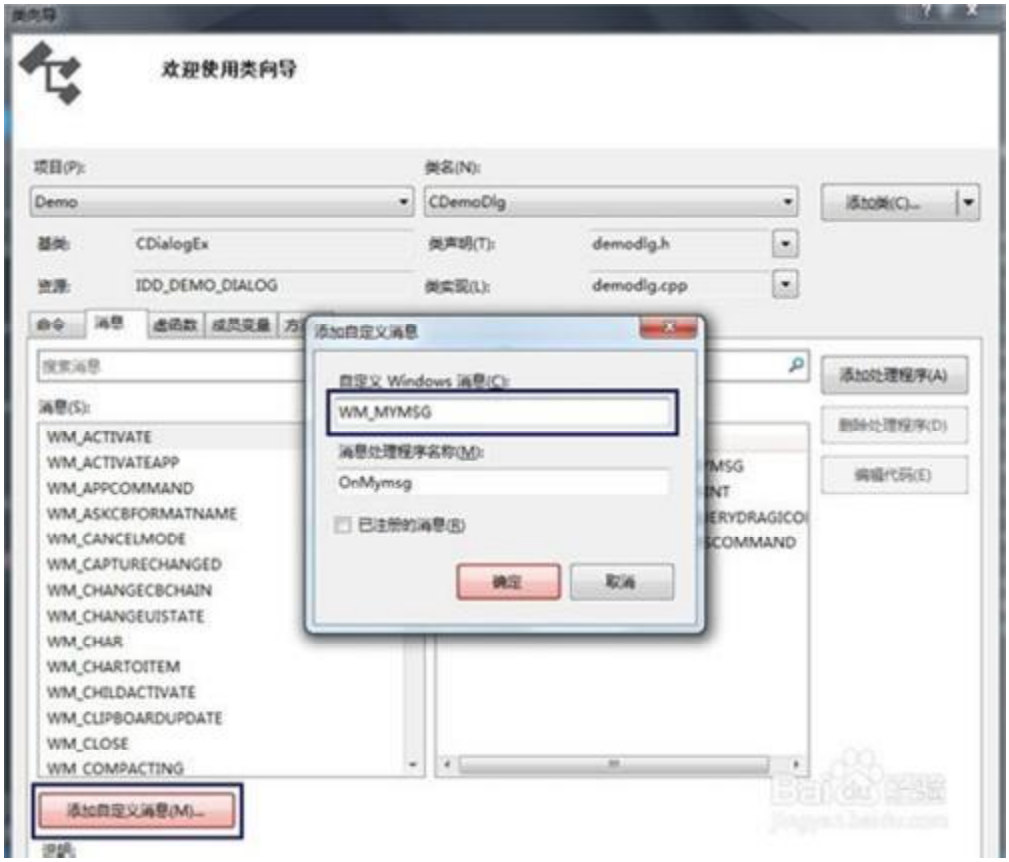
6. 6

测试程序，启动调试，每点击一次，按钮文本就更新一次，第五次弹出提示消息。

这个例程简单向大家介绍了如何在 **MFC** 中添加自定义消息并调用响应函数。

7. 7

此外，**VS2013** 提供了创建自定义消息的捷径，在类向导中的消息栏下面有“添加自定义消息按钮”，点击后填写自定义消息，类向导自动生成响应函数。但是消息的触发还是得自己添加的。



END

注意事项

- 注意自己定义的消息 **ID** 要大于 **WM_USER**

VS2013/MFC 基于对话框编程：[16]开机自启动

在我们设计软件过程中，有可能需要软件自启动以方便用户使用，那么如何编写程序实现这个功能呢，让我们一起来看看吧。

工具/原料

- **Visual Studio 2013**

方法/步骤

1. 1

打开一个已经创建好的项目，我以上篇经验中写到的项目为基础，在对话框上添加一个复选框。

功能：选中复选框则软件会自启动，否则取消自启动。

2. 2

修改复选框 **ID** 为 **IDC_AutoStart**，添加变量 **CButton m_autoStart;**

双击复选框自动生成函数 **OnBnClickedAutostart**。

3. 3

编辑函数 **OnBnClickedAutostart** 以实现自启动或取消自启动；

这个功能需要修改注册表：

1、定义注册表项 **HKEY heky;**

2、通过 **RegOpenKeyEx** 函数打开电脑的启动项；

3、通过 **GetModuleFileName** 函数获取本软件的全路径；

4、通过 **RegSetValueEx** 函数添加注册表项；

取消自启动：

1、通过 **RegOpenKeyEx** 函数打开电脑的启动项；

2、通过 **RegDeleteValue** 函数删除注册表项；

4. 4

程序代码：

void CDemoDlg::OnBnClickedAutostart()

{

// TODO: 在此添加控件通知处理程序代码

HKEY hKey;

CString strRegPath = _T("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run");

//找到系统的启动项

if (m_autoStart.GetCheck())

{

if (RegOpenKeyEx(HKEY_CURRENT_USER, strRegPath, 0, KEY_ALL_ACCESS, &hKey) == ERROR_SUCCESS) //打开启动项

{

TCHAR szModule[MAX_PATH];

GetModuleFileName(NULL, szModule, MAX_PATH);//得到本程序自身的全路径

```
RegSetValueEx(hKey, _T("Demo"), 0, REG_SZ, (LPBYTE)szModule, (lstrlen(szModule) +
1)*sizeof(TCHAR));
//添加一个子 Key,并设置值，"Demo"是应用程序名字（不加后缀.exe）
RegCloseKey(hKey); //关闭注册表
}
else
{
AfxMessageBox(_T("系统参数错误,不能随系统启动"));
}
}
else
{
if (RegOpenKeyEx(HKEY_CURRENT_USER, strRegPath, 0, KEY_ALL_ACCESS, &hKey)
== ERROR_SUCCESS)
{
RegDeleteValue(hKey, _T("Demo"));
RegCloseKey(hKey);
}
}
}
```

测试程序，启动调试。

5. 5

勾选“自启动”复选框，此时如果电脑带有 **360** 安全卫士的话，会弹框提示有用户在修改启动项。这里只要选择信任就行，同时说明了自启动功能已经成功加入到程序中，以后需要删除的时候只需去掉复选框选中状态即可。

怎么样，是不是立马感觉自启动也不是什么了不起的功能。

6. **win+R** 运行 **regedit** 打开注册表，查看注册表：

HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

可以看到新添加的注册表项项

END

注意事项

- 注意修改注册表时，应用程序的名称不需要加 **exe** 的后缀

VS2013/MFC 基于对话框编程：[17]组合框

组合框是对话框中的常用控件之一，可以说是列表框和编辑框的组合体，既可以选择已有的内容，也可以输入新的内容。本经验通过简单例子进行说明组合框的使用方法。

工具/原料

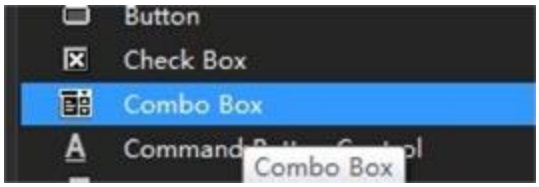
- **Visual Studio 2013**

方法/步骤

1. 1

打开 **Demo** 项目，在对话框中添加 **3** 个组合框，**1** 个编辑框，**1** 个静态文本以及 **1** 个按钮。

程序功能：通过第二个组合框选择加减乘除，第 **1** 个组合框选择数据，第 **3** 个组合框既可以选择数据，也可以输入数据；点击计算按钮计算结果并送入编辑框显示。



2. 2

打开组合框的属性页，可以在 **data** 属性中直接添加数据，用分号隔开，再把 **sort** 改为 **false**，否则数据的就会按它的排序方式进行排序，而不一定是数据输入的顺序了。最后说是 **Type** 属性，这个用于修改组合框的款式：

Simple：一般不用；

Drop Down：默认，可以输入数据，也可以选择数据；

Drop List：只能选择数据，无法输入数据；

第 **1** 个组合框选择 **Drop Down**，**data** 改为：**10;20;30;40;50**

第 **2** 个组合框选择 **Drop List**，**data** 添加为**+;-;*;/**

第 **3** 个组合框选择 **Drop Down**，**data** 暂时不填；

所有组合框的 **sort** 属性均为 **false**。

3. 3

修改 **ID**：

组合框 **1**：**IDC_Num1**；组合框 **2**：**IDC_Algor**；组合框 **3**：**IDC_Num2**；

编辑框：**IDC_Result**；按钮：**IDC_CALC**；

```
#define IDC_CALC                        1000
#define IDC_Num1                        1003
#define IDC_Algor                       1004
#define IDC_Num2                        1005
#define IDC_Result                      1006
```

4. 4

添加变量：

组合框 **1**：**CComboBox m_num1;**

组合框 **2**：**CComboBox m_algor;**

组合框 **3**：**CString m_num2;CComboBox m_num2Control;**

编辑框：**double m_result;**

添加函数：

按钮按下事件处理函数：**OnBnClickedCalc()**

修改初始化函数 **OnInitDialog()**：

通过组合框的成员函数 **AddString** 给第 **3** 个组合框添加项 **1**、**2**、**3**、**4**、**5**；

m_num2Control.AddString(_T("1"));

m_num2Control.AddString(_T("2"));

m_num2Control.AddString(_T("3"));

m_num2Control.AddString(_T("4"));

m_num2Control.AddString(_T("5"));

将三个组合框的默认选项设为第一项：

m_num2Control.SetCurSel(0);

m_num1.SetCurSel(0);

m_algor.SetCurSel(0);

```
    // TODO:  在此添加额外的初始化代码
    m_num2Control.AddString(_T("1"));
    m_num2Control.AddString(_T("2"));
    m_num2Control.AddString(_T("3"));
    m_num2Control.AddString(_T("4"));
    m_num2Control.AddString(_T("5"));
    m_num2Control.SetCurSel(0);
    m_num1.SetCurSel(0);
    m_algor.SetCurSel(0);
    return TRUE;  // 除非将焦点设置到控件，否则返回 TRUE
}
```

6. 6

编写按钮按下事件处理函数，先获取数据，然后根据选择的算法计算结果，最后进行更新。

UpdateData(true);

m_result = 0;

double num1 = (m_num1.GetCurSel() + 1) * 10;

double num2 = atof(str2char(m_num2));

//str2char 是自己写的函数，下一步有解释

switch (m_algor.GetCurSel())

{

case 0:m_result = num1 + num2; break;

case 1:m_result = num1 - num2; break;

case 2:m_result = num1 * num2; break;

case 3:m_result = num1 / num2; break;

}

UpdateData(false);

```
void CDemoDlg::OnBnClickedCalc()
{
    // TODO:  在此添加控件通知处理程序代码
    UpdateData(true);
    m_result = 0;
    double num1 = (m_num1.GetCurSel() + 1) * 10;
    double num2 = atof(str2char(m_num2));
    switch (m_algor.GetCurSel())
    {
    case 0:m_result = num1 + num2; break;
    case 1:m_result = num1 - num2; break;
    case 2:m_result = num1 * num2; break;
    case 3:m_result = num1 / num2; break;
    }
    UpdateData(false);
}
```


7. 7

由于在 **unicode** 编码中，无法直接从 **CString** 转换到 **const char***，隐藏 **atof** 函数无法直接把字符串转换为浮点型数据，需要经过中间转换步骤将字符串类型转为 **const char***类型。这就是 **str2char** 需要完成的任务：

```
char* CDemoDlg::str2char(CString str)
{
char *ptr;
#ifdef _UNICODE
LONG len;
len = WideCharToMultiByte(CP_ACP, 0, str, -1, NULL, 0, NULL, NULL);
ptr = new char[len + 1];
memset(ptr, 0, len + 1);
WideCharToMultiByte(CP_ACP, 0, str, -1, ptr, len + 1, NULL, NULL);
#else
ptr = new char[str.GetAllocLength() + 1];
sprintf(ptr, _T("%s"), str);
#endif
```

return ptr;

}

这个函数大家可以保存下来，相信肯定会有用到的时候。

```
char* CDemoDlg::str2char(CString str)
{
    char *ptr;
#ifdef _UNICODE
    LONG len;
    len = WideCharToMultiByte(CP_ACP, 0, str, -1, NULL, 0, NULL, NULL);
    ptr = new char[len + 1];
    memset(ptr, 0, len + 1);
    WideCharToMultiByte(CP_ACP, 0, str, -1, ptr, len + 1, NULL, NULL);
#else
    ptr = new char[str.GetAllocLength() + 1];
    sprintf(ptr, _T("%s"), str);
#endif
    return ptr;
}
```

8. 8

测试程序功能，启动调试，选择算法，选择数据或者输入数据，点击计算得出结果。

END

注意事项

- **CString** 在 **unicode** 编码类型下无法直接转换为 **const char***类型，可以通过自己编写的 **str2char** 函数实现。
- 如果定义了组合框的 **int** 型数据，它不代表具体的内容，只代表当前选取的内容的序号，序号从 **0** 开始逐一增加。

VS2013/MFC 基于对话框编程：[18]多线程

很多时候单个线程无法满足处理速度，当事情较多并且要求高速运行时就会出现卡顿状态，所以很多时候多线程是极为必要的，这里通过简单例子说明多线程的创建和终止。

工具/原料

- **Visual Studio 2013**

方法/步骤

1. 打开 **Demo** 项目，在对话框中添加一个按钮，两个静态文本和两个编辑框。

   程序功能：主线程用于启动或者停止子线程；子线程 **1** 每隔 **100** 毫秒输出显示一个 **0-1000** 的随机数，子线程 **2** 每隔 **1** 秒显示一次当前的系统时间。

2. 修改 **ID**，按钮 **ID** 改为 **IDC_StartThread**；两个编辑框的 **ID** 分别为 **IDC_Random**、**IDC_SysTime**。

   双击按钮自动生成事件处理函数 **OnBnClickedStartthread().**

```
#define IDC_StartThread                 1000
#define IDC_SysTime                     1001
#define IDC_Random                      1002
```

3. 在头文件中添加两个线程函数的声明，注意函数必须是静态（**static**）的，形参一般为 **void *param**。然后再定义一个标志 **flag**，用来表示子线程的关闭和启动。初始值可以在构造函数中设定为 **false**。

```
    static  UINT Thread1(void *param);
    static  UINT Thread2(void *param);
    bool flag;
```

4. 实现子线程 **1**，注意子线程中无法直接访问对话框的界面，需要通过重新定义指针变量与对话框和控件进行关联，**flag** 为真时执行线程，**flag** 为假时终止线程。

UINT CDemoDlg::Thread1(void *param)

{

CDemoDlg *dlg = (CDemoDlg*)param;

CEdit *randBox = (CEdit*)dlg->GetDlgItem(IDC_Random);

CString str;

while (dlg->flag)

{

Sleep(100);

str.Format(_T("%i"), rand()%1000);

randBox->SetWindowTextW(str);

}

return 0;

}

```
UINT CDemoDlg::Thread1(void *param)
{
    CDemoDlg *dlg = (CDemoDlg*)param;
    CEdit *randBox = (CEdit*)dlg->GetDlgItem(IDC_Random);
    CString str;
    while (dlg->flag)
    {
        Sleep(100);
        str.Format(_T("%i"), rand()%1000);
        randBox->SetWindowTextW(str);
    }
    return 0;
}
```

5. 实现子线程 **2**，与线程 **1** 类似，获取系统时间可以用 **CTime** 类型的 **GetCurrentTime** 函数

UINT CDemoDlg::Thread2(void *param)

{

```
CDemoDlg *dlg = (CDemoDlg*)param;
CEdit *SysTimeBox = (CEdit*)dlg->GetDlgItem(IDC_SysTime);
CString str;
while (dlg->flag)
{
Sleep(1000);
CTime time = CTime::GetCurrentTime();
str = time.Format(_T("%Y-%m-%d %H:%M:%S %A"));
SysTimeBox->SetWindowTextW(str);
}
return 0;
}
```

```
UINT CDemoDlg::Thread2(void *param)
{
    CDemoDlg *dlg = (CDemoDlg*)param;
    CEdit *SysTimeBox = (CEdit*)dlg->GetDlgItem(IDC_SysTime);
    CString str;
    while (dlg->flag)
    {
        Sleep(1000);
        CTime time = CTime::GetCurrentTime();
        str = time.Format(_T("%Y-%m-%d %H:%M:%S %A"));
        SysTimeBox->SetWindowTextW(str);
    }
    return 0;
}
```

6. 编程按钮按下函数，用来启动和关闭线程，采用 **AfxBeginThread** 函数

```
void CDemoDlg::OnBnClickedStartthread()
{
    // TODO:  在此添加控件通知处理程序代码
    if (flag)
    {
        flag = false;
        GetDlgItem(IDC_StartThread)->SetWindowTextW(_T("打开子线程"));
    }
    else
    {
        AfxBeginThread(Thread1, this, THREAD_PRIORITY_IDLE);
        AfxBeginThread(Thread2, this);

        flag = true;
        GetDlgItem(IDC_StartThread)->SetWindowTextW(_T("关闭子线程"));
    }
}
```


7. 测试程序，启动调试，点击打开线程查看执行效果。

END

注意事项

- 多个线程最后不要在一个函数中启动，否则可能会出现某个线程不执行的情况

VS2013/MFC 基于对话框编程：[19]保存 txt 文件

软件开发过程中，很多时候需要保存数据，而 **txt** 就是最常用也是最简单的保存方式，这里我将介绍如何在 **MFC** 中以 **txt** 格式保存数据。

工具/原料

- **Visual Studio 2013**

方法/步骤

1. 设计对话框界面，一个用于保存数据的按钮，三个单选按钮用于选择数据类型，两个编辑框分别用来输入数据个数和保存的文件名。

2. 修改控件的 **ID**：

   按钮：**IDC_Save**；

   单选按钮：**IDC_Rand**；**IDC_Fibonacci**；**IDC_Factorial**；

   编辑框：**IDC_DataNum**；**IDC_txtName**；

```
#define IDC_Save                        1000
#define IDC_Rand                        1008
#define IDC_Fibonacci                   1009
#define IDC_Factorial                   1010
#define IDC_DataNum                     1011
#define IDC_txtName                     1012
```

3. 修改属性，添加变量：

   第一个单选按钮的 **Group** 为 **true**，添加变量 **int DataType;**

   输入数据个数的编辑框的 **Number** 属性为 **true**，添加变量 **UINT m_DataNum**；

   输入文件名的编辑框：添加变量 **CString m_txtName**；

4. 修改构造函数中变量的初始值，将数据个数默认值改为 **100**，将文件名修改默认值为 **"Data.txt".**

```
CDemoDlg::CDemoDlg(CWnd* pParent /*=NULL*/)
    : CDialogEx(CDemoDlg::IDD, pParent)
    , m_DataType(0)
    , m_DataNum(100)
    , m_txtName(_T("Data.txt"))
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}
```

5. 编写保存函数，首先获取界面数据，然后判断数据类型，计算出对应的数据，通过 **CStdioFile** 类的对象将数据写入 **txt** 文件中，**Open** 函数可以指定保存路径，默认保存在工程文件的 **Demo\Demo**文件夹下。

1、随机数

```
for (int i = 0; i < m_DataNum; i++)
Data[i] = rand()%10000;
```

2、斐波那契数列

```
Data[0] = 1; Data[1] = 1;
for (int i = 2; i < m_DataNum; i++)
Data[i] = Data[i - 1] + Data[i - 2];
```

3、阶乘

```
for (int i = 0; i < m_DataNum; i++)
{
Data[i] = 1;
for (int j = 1; j <= i+1; j++)
Data[i] *= j;
}
```

4、文件保存

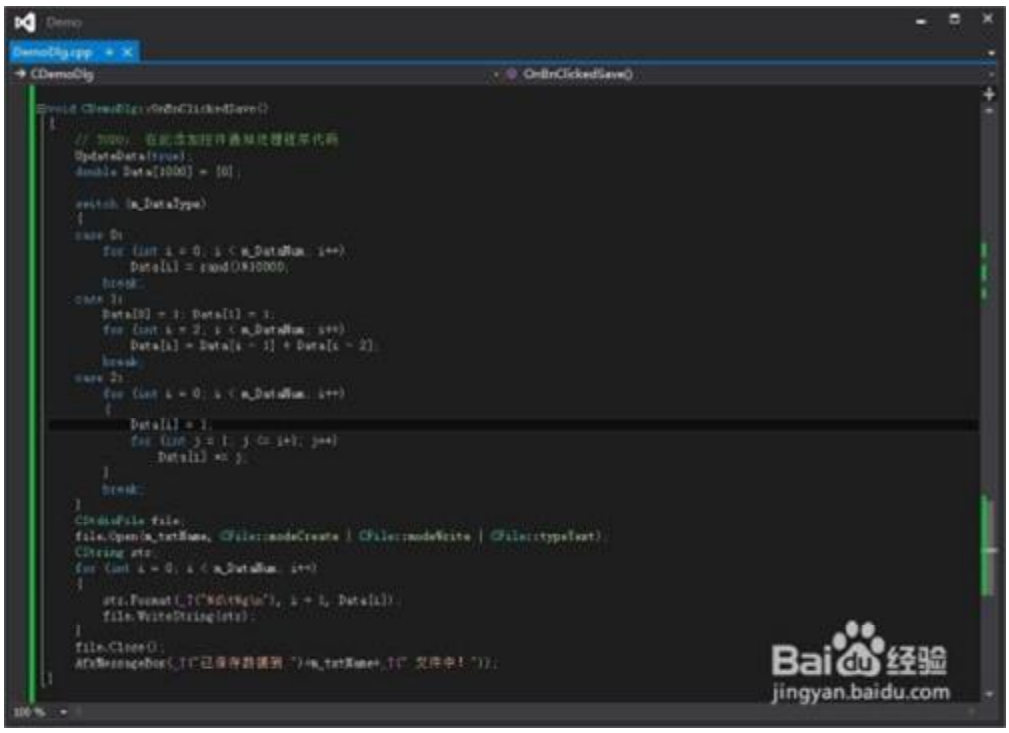
```
CStdioFile file;
file.Open(m_txtName, CFile::modeCreate | CFile::modeWrite | CFile::typeText);
CString str;
for (int i = 0; i < m_DataNum; i++)
{
str.Format(_T("%d\t%g\n"), i + 1, Data[i]);
```

file.WriteString(str);

}

file.Close();

AfxMessageBox(_T("已保存数据到 ")+m_txtName+_T(" 文件中！"));



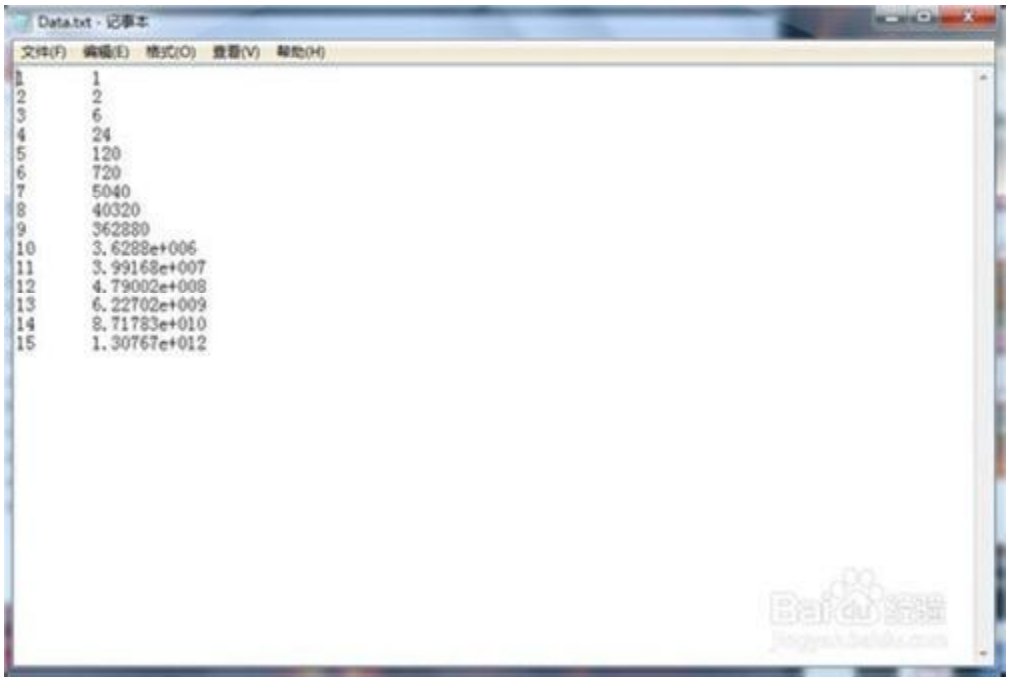
6. 测试程序，启动调试，选择数据类型，设置数据个数和文件名称。

1、随机数测试，设置 **100** 个数

每次保存完都有提示。

7. **2**、**fibonacci** 数列测试，设置 **20** 个数

8. **3**、阶乘测试，设置 **15** 个数，阶乘增长速度很快，所以没必要保存太多。



注意事项

- 保存数据后别忘了用文件类的 **close** 函数关闭文件。