



MFC之二

周艳

西南交通大学电气工程学院

西南交通大学

单文档/视图应用

利用向导生成一个MFC单文档应用程序MFCSDI

MFC 应用程序向导 - MFCSDI



应用程序类型

概述

应用程序类型

复合文档支持

文档模板属性

数据库支持

应用程序类型:

☒ 单个文档 (S)

☐ 多个文档 (M)

☐ 选项卡式文档 (B)

☐ 基于对话框 (D)

项目类型:

☒ MFC 标准 (A)

☐ Windows 资源管理器 (X)

☐ Visual Studio (O)

☐ Office (F)

生成的类(G):

CMFCSDIView
CMFCSDIApp
CMFCSDIDoc
CMainFrame

类名(L):

CMFCSDIView

.h 文件(E):

MFCSDIView.h

基类(A):

CView

.cpp 文件(P):

MFCSDIView.cpp

一、SDI应用程序的类结构

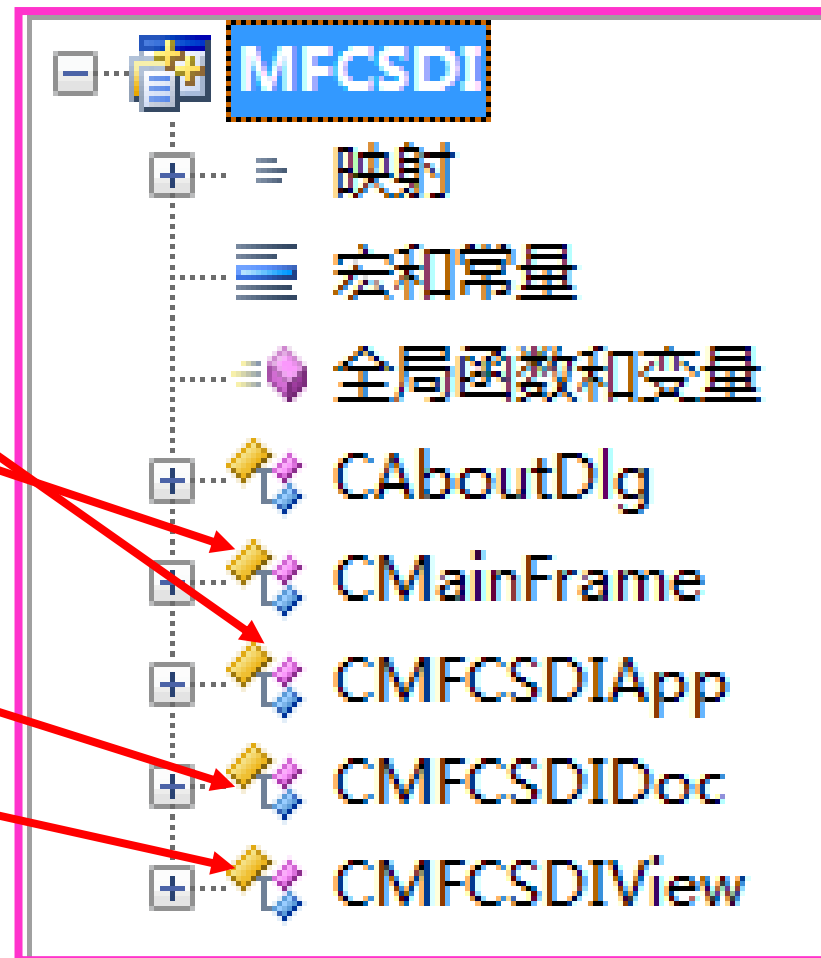
□ 向导AppWizard生成的单文档应用程序一般包括四个基本类：

■ 应用程序类

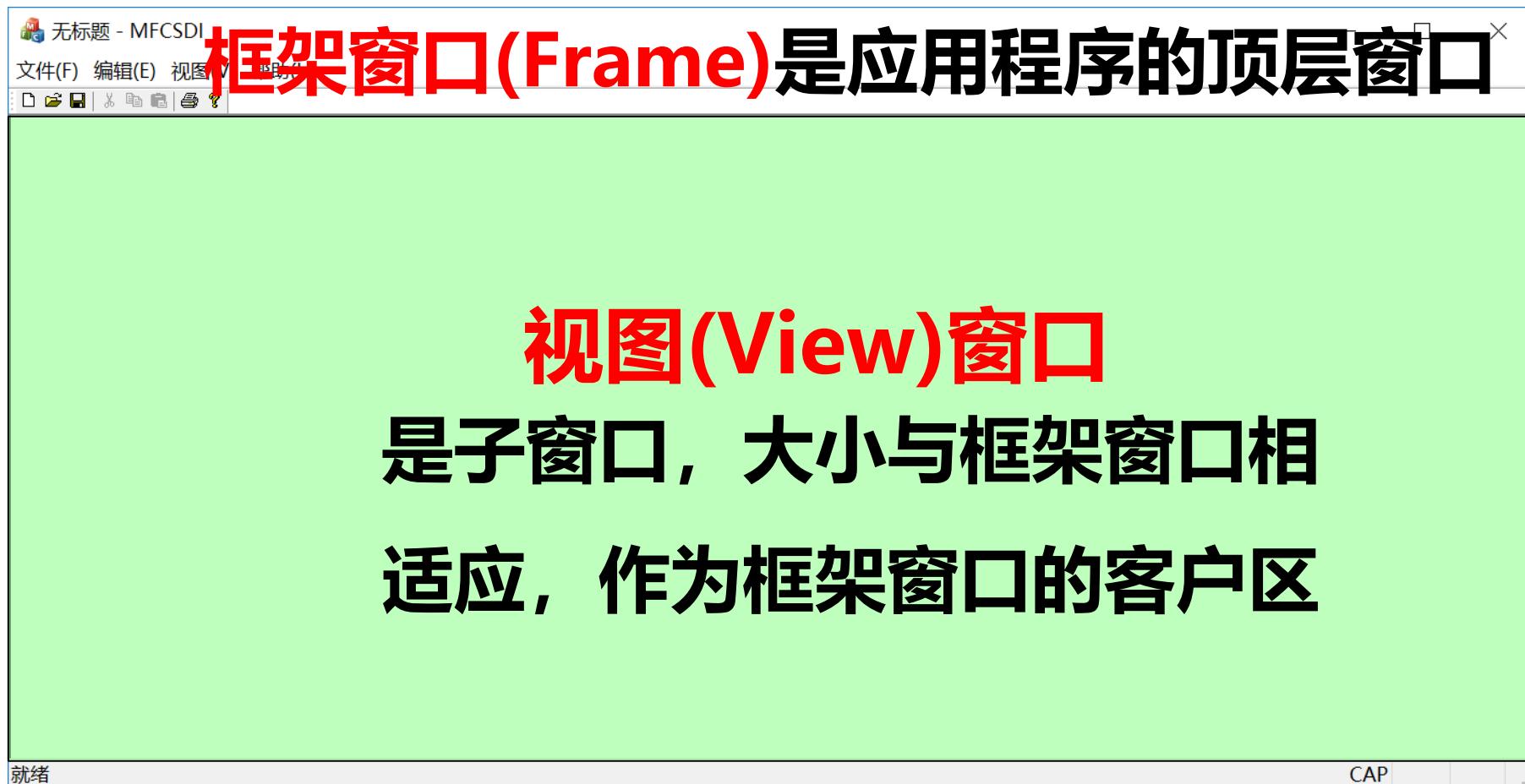
■ 主框架窗口类

■ 文档类

■ 视图类



MFC向导生成的SDI程序界面



MFC中的窗口实际是Frame和View共同作用的结果

(1)主框架窗口类

- 主框架窗口类名为 CMainFrame，它由 CWnd的一个子类CFrameWnd派生而来。

```
class CMainFrame : public CFrameWnd
```

- CMainFrame类管理主框架窗口，拥有如菜单、工具栏和状态栏等控件，同时扮演转发菜单和工具栏消息的角色。

(2)文档类

- 保存应用程序的数据，提供磁盘文件操作

(3)视图类

- 管理视图窗口，负责接受用户数据的输入和数据的输出显示。

以上三部分如何关联？

(4)文档模板(Document Template)

- 文档模板CDocTemplate定义了文档、视图和框架窗口这3个类的关系。
- 一般在应用程序的初始化函数InitInstance()中创建文档模板。

BOOL CMFCSDIApp::InitInstance()

{ // 注册应用程序的文档模板。文档模板

// 将用作文档、框架窗口和视图之间的连接

CSingleDocTemplate* pDocTemplate; // 单文档模板类对象

pDocTemplate = new CSingleDocTemplate(

IDR_MAINFRAME, // 字符串、菜单、光标资源

RUNTIME_CLASS(CMFCSDIDoc), // 文档

RUNTIME_CLASS(CMainFrame), // 主框架窗口

RUNTIME_CLASS(CMFCSDIView)); // 视图

AddDocTemplate(pDocTemplate);

.....}

MFC自动生成

二、 文档视图结构概述

- **文档**：管理和维护数据。File(Open)/File(New)时会打开一份文档。 一个文档可以同时拥有多个视图。
- **视图**：显示和编辑数据。是一个窗口(可视化的矩形区域)。视图必须依附于一个框架(SDI中是MainFrame) 一个视图只能拥有一个文档
- 一个**视图**总是与一个**文档**对象相关联。打开一个文档时，应用程序就会创建一个与之相关联的视图。对数据的编辑需要依靠鼠标与键盘操作，这些信息由**视图类接收**后进行处理或通知文档类。

文档和视图类常用的成员函数

1. CView视图类的成员函数GetDocument()

一个视图对象只有一个与之相关联的文档对象。视图对象通过调用成员函数GetDocument()得到与之相关联的文档对象的指针，利用该指针就可以访问文档类及其派生类的公有数据成员和成员函数。

2. CDocument类的成员函数UpdateAllViews()

当一个文档数据通过某个视图被修改后，与它关联的每个视图都必须反映出这种修改。因此，视图在需要时必须进行重绘，即**当文档数据发生改变时，必须通知所有相关联的视图对象，以便更新所显示的数据。更新与该文档有关的所有视图的方法是调用成员函数CDocument::UpdateAllViews()。**

3. CView视图类的成员函数OnUpdate()

应用程序调用UpdateAllViews()时，实际是调用所有相关视图的OnUpdate()，以更新相关视图。需要时，可以直接在视图派生类的成员函数中调用该函数刷新当前视图。

刷新视图时默认的函数调用过程是：

Cdocument::UpdateAllViews()→CView::OnUpdate()
→CWnd::Invalidate()→OnPaint()→OnDraw()

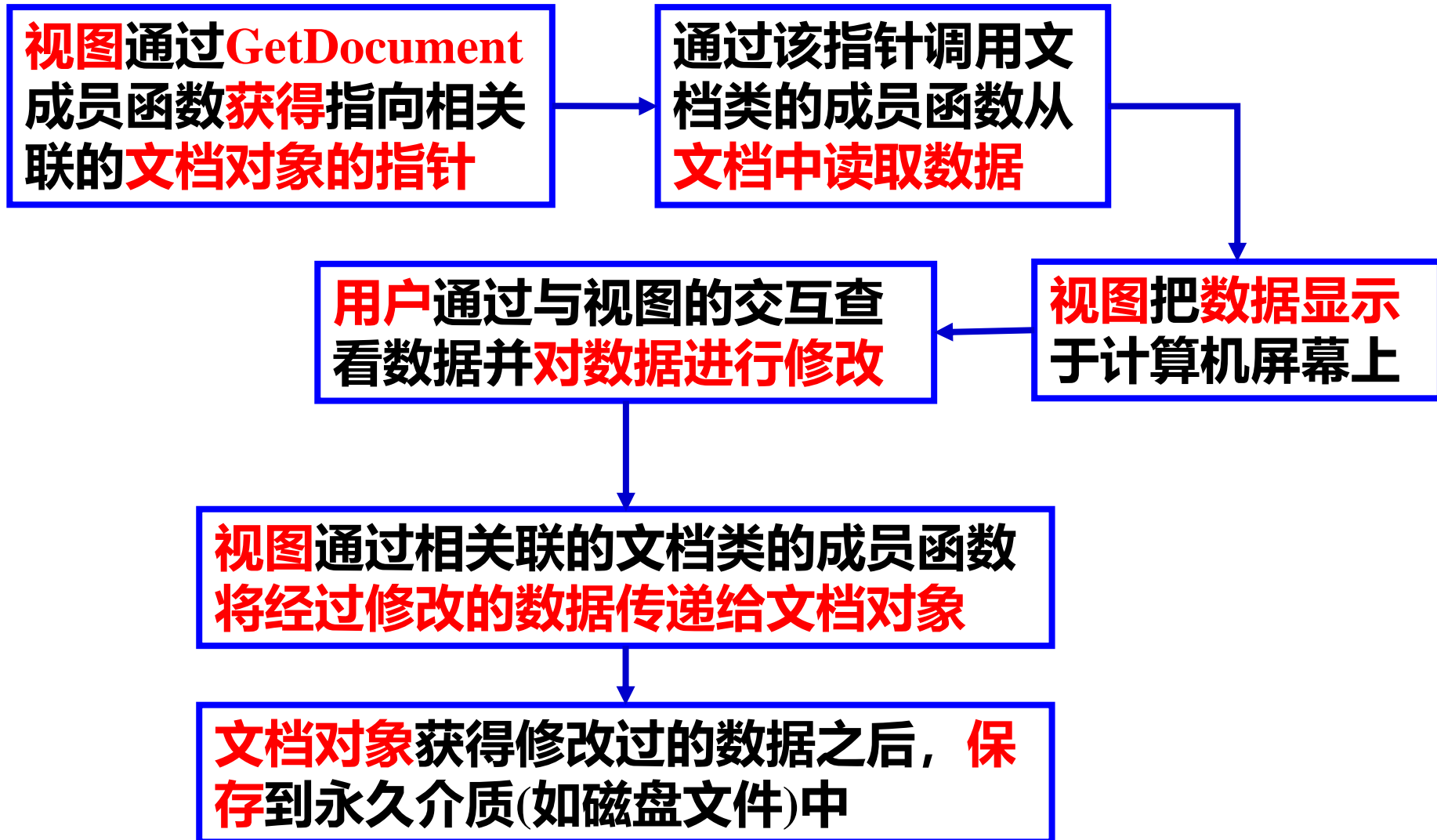
4、Invalidate、OnPaint、OnDraw

- **Invalidate**使原来的客户区无效，需要重绘。如一个被其它窗口遮住的窗口变成了前台窗口，那原来被遮住的部分就是无效的，需要重绘。这时Windows会发送**WM_PAINT**消息。
- **OnPaint**响应**WM_PAINT**消息，负责重绘窗口。
- **OnPaint**函数中调用了**OnDraw**函数，**实际的重绘工作由OnDraw来完成。**
- **对客户区进行绘图的所有代码都写在OnDraw中**

5、文档类的成员函数OnNewDocument()

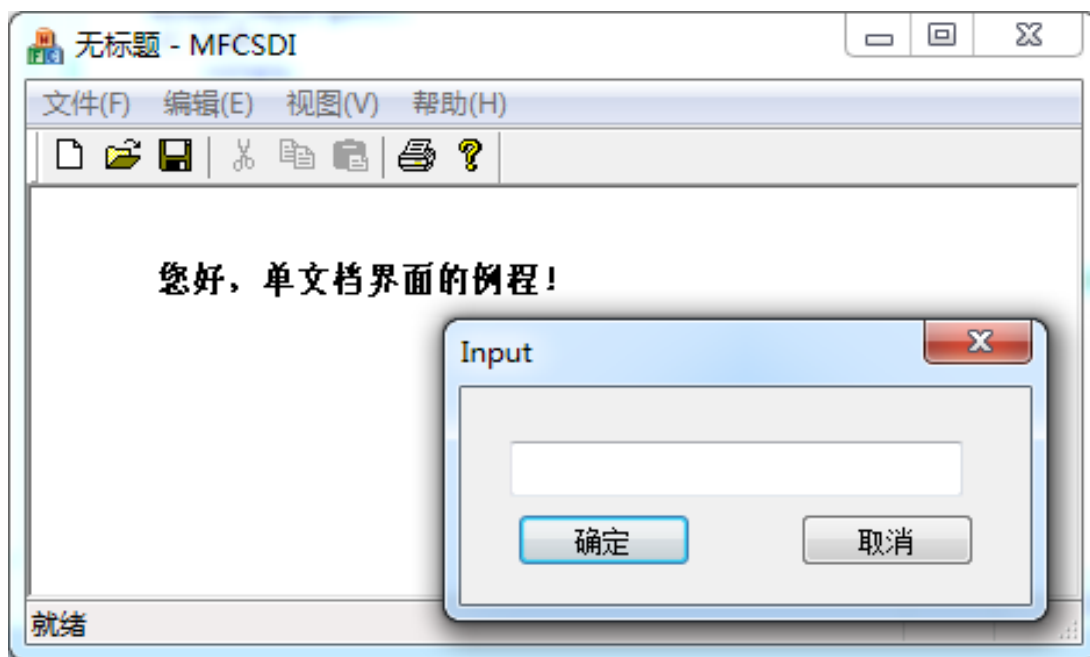
一般的，类的数据成员的初始化都是在构造函数中完成的，在构造函数调用结束时对象才真正存在。但对于文档来说却不同，文档类的数据成员初始化工作是在OnNewDocument()成员函数中完成的。

文档/视图结构的工作机制

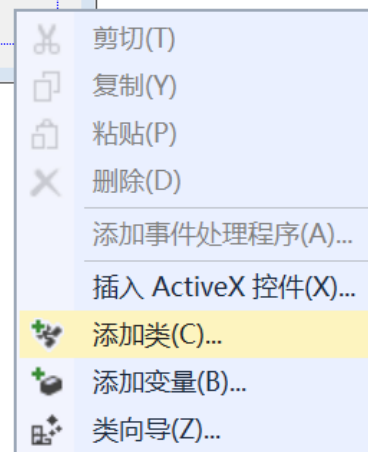
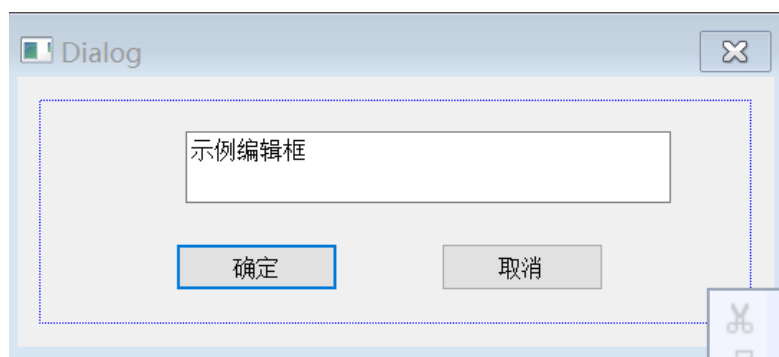
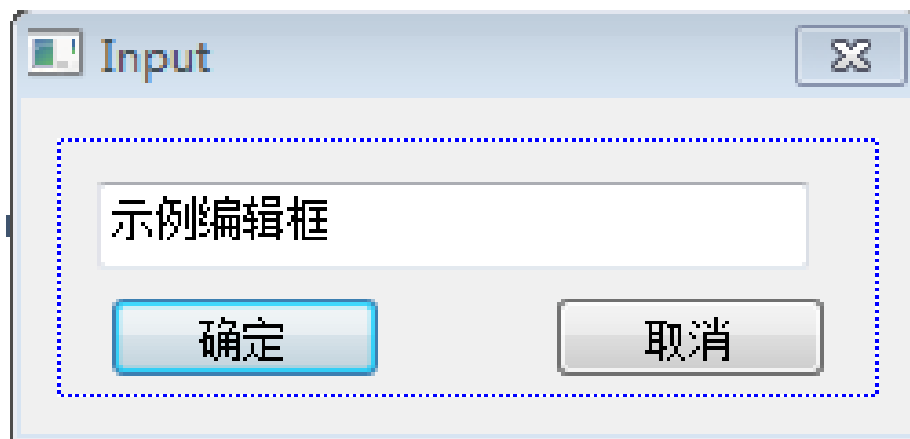


SDI编程实例

【例】在主窗口显示文本“您好, 单文档界面的例程!”。单击“改变显示文本”菜单项, 可弹出一个对话框, 通过此对话框可改变主窗口中的显示文本内容。



(1) 添加对话框资源



生成对话框类CInputDialog



欢迎使用 MFC 添加类向导

名称

文档模板属性

类名(L):

CInputDialog



基类(B):

CDialogEx



对话框 ID(D):

IDD_INPUTDLG



.h 文件(I):

InputDialog.h



.cpp 文件(P):

InputDialog.cpp

☐ Active Accessibility(Y)

DHTML 资源 ID(S):

IDR_HTML_INPUTDLG

.HTM 文件(M):

InputDialog.htm

自动化:

☒ 无(N)☐ 自动化(A)☐ 可按类型 ID 创建(E)

类型 ID(T):

MFCSDI.InputDlg

☐ 生成 DocTemplate 资源(G)

< 上一步

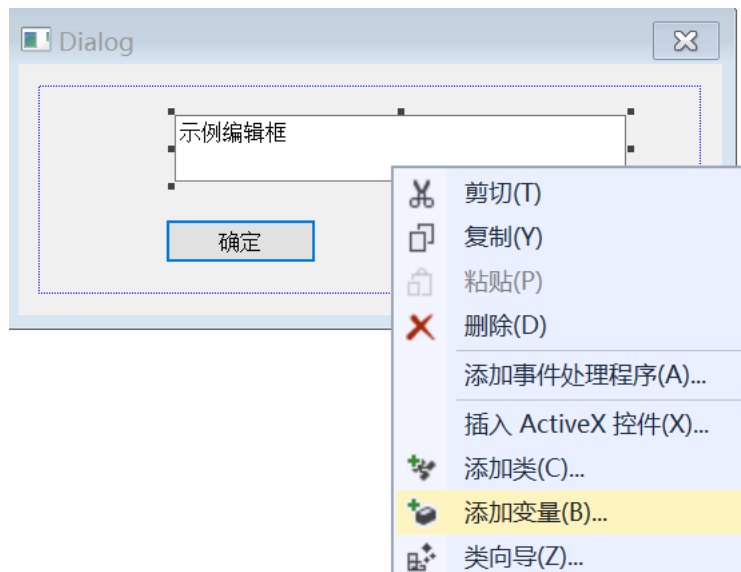
下一步 >

完成

取消

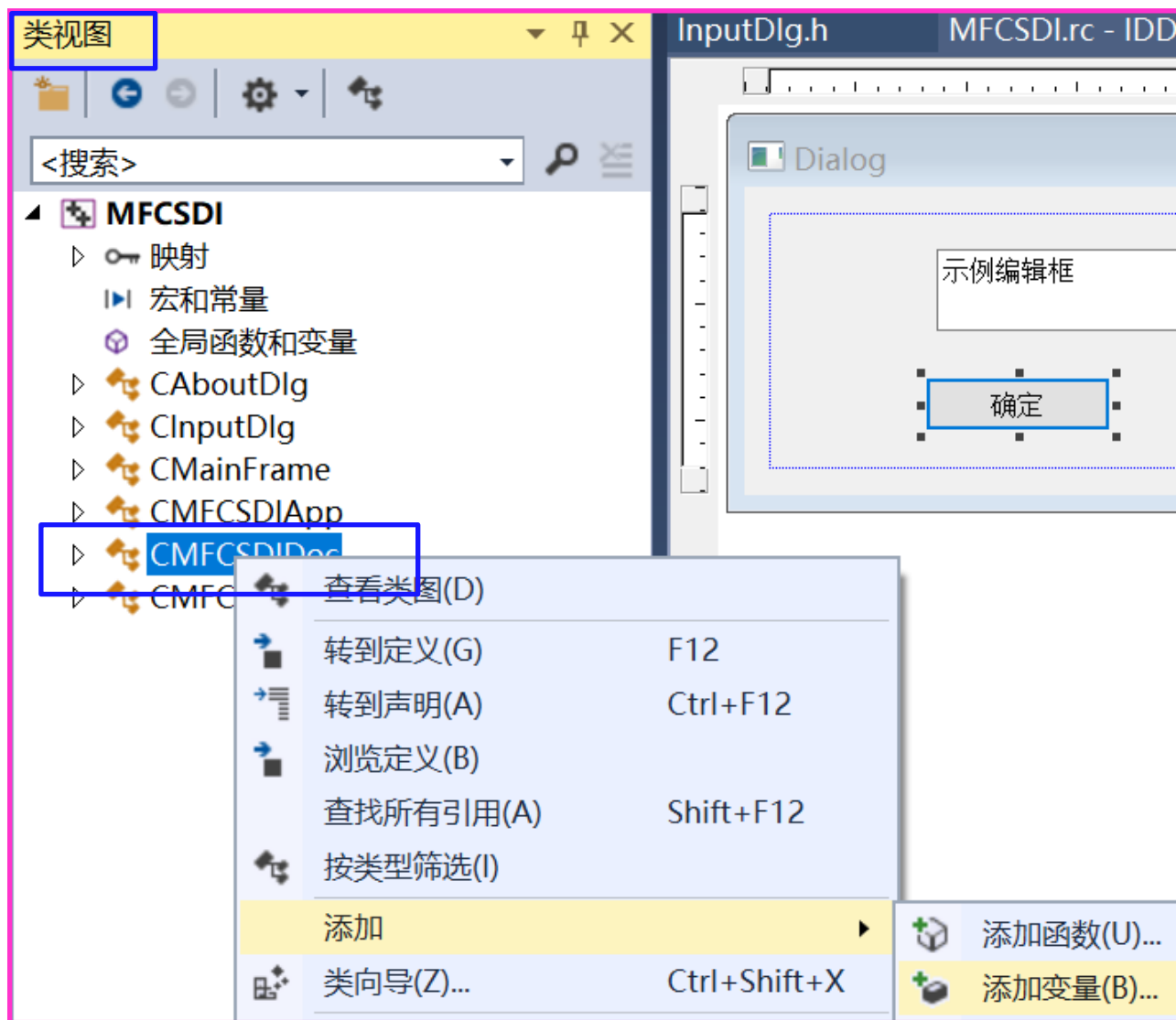
(2)为对话框的编辑框添加成员变量

添加CString类型的Value型成员变量m_input，存放要修改的字符串。



(3)文档变量初始化

为文档类添
加CString成
员变量m_str



欢迎使用添加成员变量向导



访问(A):

public

变量类型(V):

CString

变量名(N):

m_str

☐ 控件变量(O)

控件 ID(I):

控件类型(Y):

最小值(U):

.h 文件(F):

类别(T):

Control

最大字符数(X):

最大值(E):

.cpp 文件(P):

注释(不需要 // 表示法)(<U>M</U>):

完成

取消

系统会在CMFCSDIDoc.h 中自动加入

```
public:  
    CString m_str;
```

在CMFCSDIDoc.cpp的OnNewDocument()手动加入

```
BOOL CMFCSDIDoc::OnNewDocument()
```

```
{
```

```
.....
```

```
m_str="您好，单文档例程!";//初始化成员变量
```

```
}
```

(4) 视图的输出

读取文档数据到字符串str中，调用TextOut显示到视图

```
void CMFCSDIView::OnDraw(CDC* pDC)
```

绘图设备基类

```
{ CMFCSDIDoc* pDoc = GetDocument();//获取文档类指针
```

```
.....
```

```
CString str=pDoc->m_str; // 从文档中读取数据
```

```
pDC->TextOut(200, 200, str);
```

```
}
```

□ TextOut (*int x,int y,CString& str*)

- 参数*x*和*y*为文本显示在窗口用户区的水平位置和垂直位置，以像素为单位；
- *str*为要显示的字符串，是*CString*类的对象

(5) 添加菜单项

添加菜单项“改变显示文本” (ID_CHANGETEXT)

The screenshot shows the Visual Studio IDE with the Resource View on the left and the Menu Editor on the right. The Resource View shows the project structure for 'MFCSDI', including folders for Accelerator, Dialog, Icon, and Menu. The Menu Editor shows the menu structure for 'MFCSDI.rc - IDR...AINFRAME - Menu*'. The menu items are listed in a table:

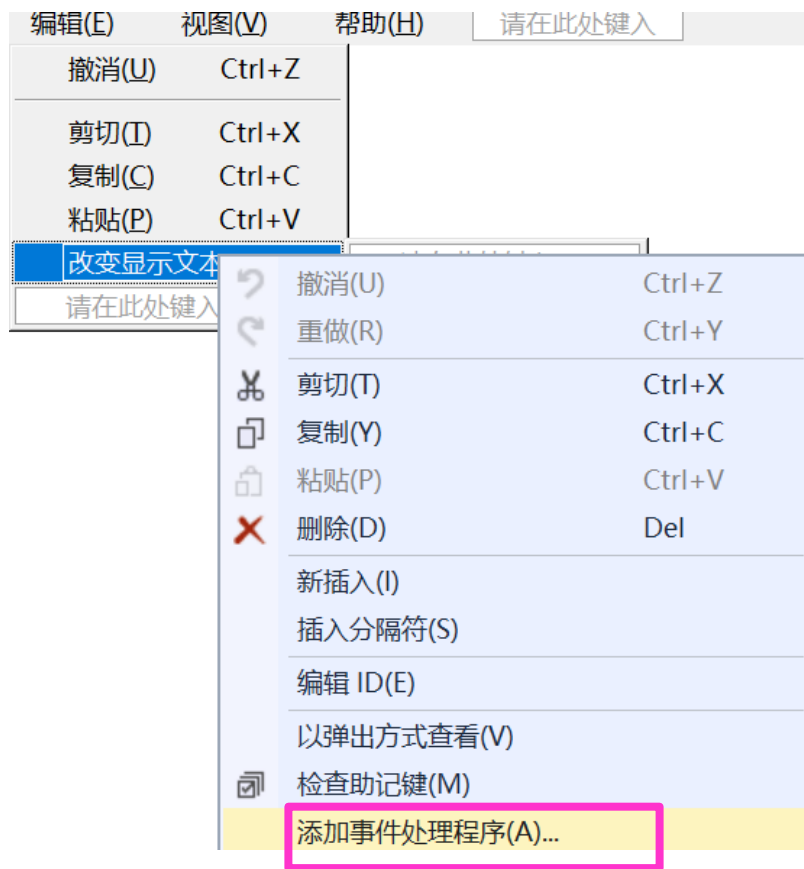
文件(E)	编辑(E)	视图(V)
	撤消(U)	Ctrl+Z
	剪切(I)	Ctrl+X
	复制(C)	Ctrl+C
	粘贴(P)	Ctrl+V
	改变显示文本	
	请在此处键入	

The '改变显示文本' menu item is highlighted with a pink box. Below the menu editor, the 'Right Order' dialog box is open, showing the properties for the selected menu item. The 'Caption' is '改变显示文本', and the 'ID' is 'ID_CHANGETEXT'. The 'ID' field is highlighted with a pink box.

修改菜单项属性ID

(6) 改变显示文本

单击此菜单项弹出对话框，通过对话框输入的内容改变视图的显示文本。右键菜单项“添加事件处理程序”。



(6) 改变显示文本

事件处理程序向导 - MFCSDI2

欢迎使用事件处理程序向导

命令名:
ID_CHANGETEXT

消息类型(Y):
COMMAND
UPDATE_COMMAND_UI

函数处理程序名称(N):
OnChangertext

类列表(L):
CAboutDlg
CInputDialog
CMFCSDI2App
CMFCSDI2Doc
CMFCSDI2View
CMainFrame

注意选择基类为
CMFCSDIDoc

需要通过文档类的成员
变量获取用户输入文本

菜单处理函数名

添加编辑(A) 编辑代码(E) 取消

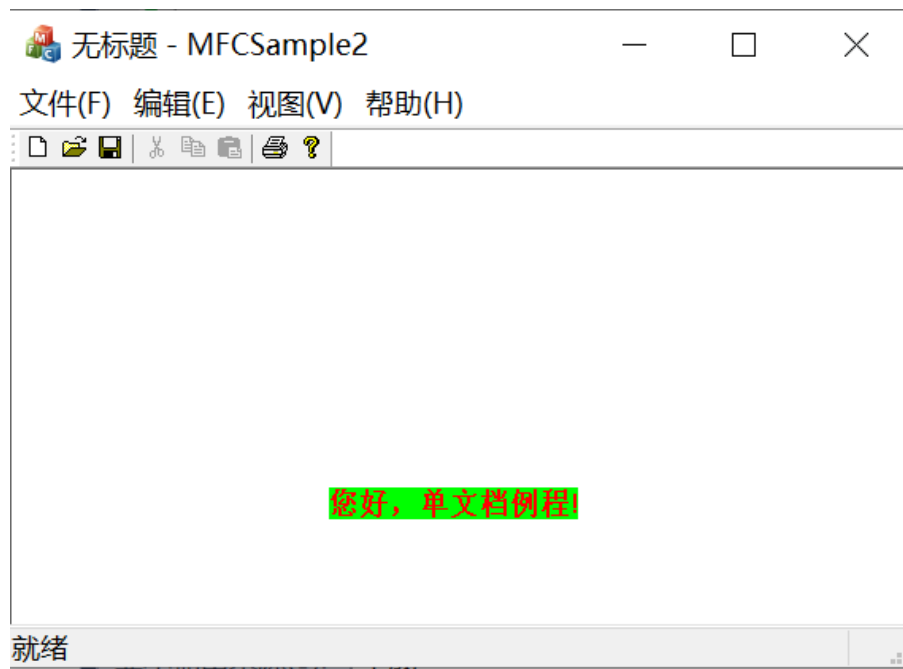
```
void CMFCSDIDoc::OnChangeText()
{   CInputDialog inputDlg; //创建CInputDialog类的对象inputDlg
    if(inputDlg.DoModal()==IDOK) //显示对话框
    {
        m_str=inputDlg.m_input; //获取输入的字符串
        UpdateAllViews(NULL); //文档被修改后调用此
        函数，把文档被修改的信息通知给每个视图。
    }
}
```

为在CMFCSDIDoc.cpp类中定义对话框类CInputDialog对象，需在MFCSDIDoc.cpp文件中加入：

```
#include "InputDialog.h" //加入头文件
```

- ❑ 还可以在视图中设置输出文本的颜色、背景等信息。如在CMFCSDIView::OnDraw中加入：

```
pDC->SetTextColor(RGB(255,0,0));  
pDC->SetBkColor(RGB(0,255,0));
```



Windows消息机制

1、组成：消息名称(UINT)+2参数(WPARAM,LPARAM)。

用户输入或窗口状态改变时系统都会发送消息到某个窗口

- **如执行菜单命令会发送WM_COMMAND消息，
WPARAM的高字 (HIWORD(wParam))是命令的ID号
，对菜单来讲就是菜单ID。用户可以定义自己的消息
名称，也可利用自定义消息来发送通知和传送数据**

Windows消息机制

2、谁将收到消息：一个消息必须由一个窗口接收。在窗口的过程(WNDPROC)中可以对消息进行分析和处理。

■ 例如对菜单选择进行处理可以定义对WM_COMMAND进行处理的代码，如果希望在窗口中进行图形输出就必须对WM_PAINT进行处理

3、未处理的消息到哪里去了：Windows为窗口编写了默认的窗口过程，这个窗口过程将负责处理那些你不处理的消息。

4、窗口句柄：系统通过窗口句柄唯一标识一个窗口，发送一个消息时必须指定一个窗口句柄表明该消息由哪个窗口接收。而每个窗口都会有自己的窗口过程，所以用户的输入就会被正确的处理。

- 系统维护一个或多个消息队列，所有产生的消息都会被放入或插入队列中。系统会在队列中取出每一条消息，根据消息的接收句柄而将该消息发送给拥有该窗口的程序的消息循环
- 每一个运行的程序都有自己的消息循环，在循环中得到属于自己的消息并根据接收窗口的句柄调用相应的窗口过程。

消息处理机制

□ Windows通过消息名访问消息，但不同类型的消息由应用程序的不同部分进行处理。

MFC中的消息分为：

- 窗口消息
- 控件通知消息
- 定时消息
- 命令消息

一、窗口消息 (Windows message)

- 通常指以WM开头的消息，但WM_COMMAND除外。
- 键盘消息和鼠标消息都属于Windows消息，由窗口和视图进行处理。
- Windows消息通常带有若干个参数传递给消息处理函数。

1、鼠标消息及其处理

如对话框示例1添加的鼠标消息处理

- 消息映射机制：将消息与处理函数相联系，当系统产生一条消息时，它能找到处理该消息的函数

MFC的三个常见消息映射宏

消息映射宏	功 能
DECLARE_MESSAGE_MAP	在头文件声明源文件中所含有的消息映射
BEGIN_MESSAGE_MAP	标记源文件消息映射的开始
END_MESSAGE_MAP	标记源文件消息映射的结束

2、键盘消息及其处理

为View类添加按键消息处理函数

```
void CMFCSDIView::OnKeyDown(UINT nChar, UINT  
nRepCnt, UINT nFlags)  
{  CClientDC dc(this);  
    if (nChar == VK_CAPITAL)  
        dc.TextOut(100, 50, _T("Cap Lock键按下!"));  
    if (nChar == VK_CONTROL)  
        dc.TextOut(100, 100, _T("CTRL键按下!"));  
    if (nChar == 13)  
        dc.TextOut(100, 150, _T("ENTER键被按下!"));  
}
```

CClientDC是CDC的衍生类，产生对应于Windows客户区的对象

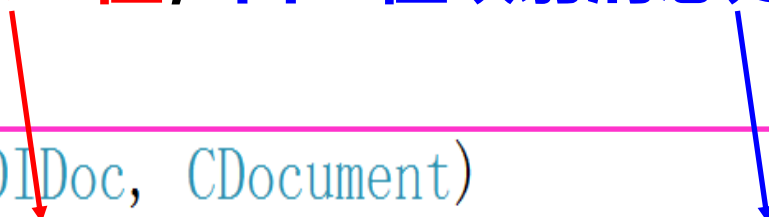
二、控件通知消息

- 控件状态改变时，控件向其父窗口发送的消息。如按钮的单击。
- MFC对控件通知消息的传递方式与其他以WM开头的Windows消息一样。
- 对于Windows消息和控件通知消息，MFC将消息传递给相应的窗口处理。

三、命令消息

- 命令消息主要包括由用户交互对象（**菜单**、工具栏按钮、快捷键等）发送的WM_COMMAND通知消息
- WM_COMMAND消息的**消息映射宏**OnCommand()。
所有命令消息都包含有一个相同类型的参数，即该命令消息需要操作的**资源ID值**，由**ID值映射消息处理函数**

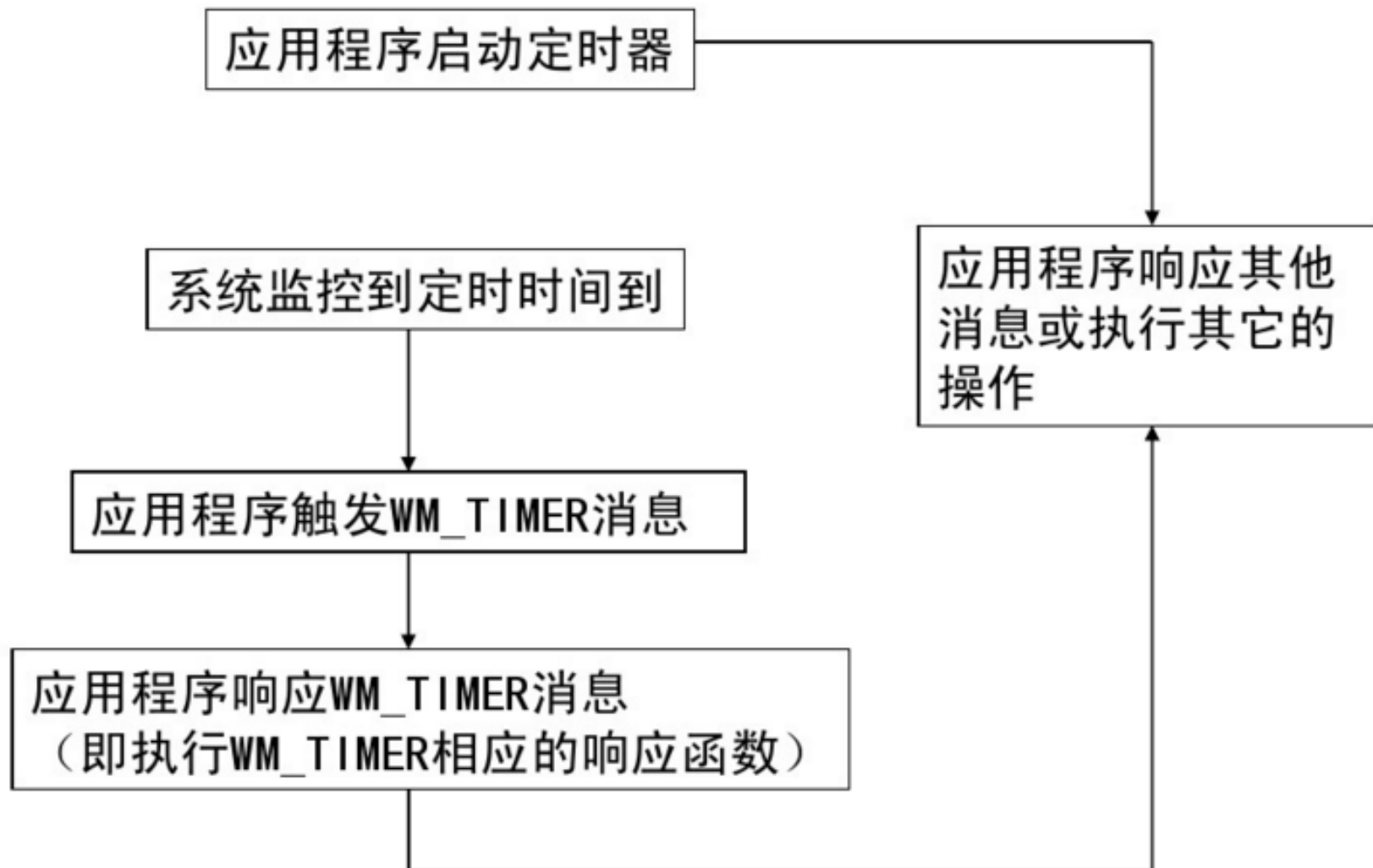
```
BEGIN_MESSAGE_MAP(CMFCSDIDoc, CDocument)
    ON_COMMAND(ID_EDIT_CHANGETEXT, &CMFCSDIDoc::OnChangertext)
END_MESSAGE_MAP()
```



四、定时消息

- 当需要应用程序每隔指定的时间间隔执行某一特定操作时，就需要使用定时消息**WM_TIMER**。
- 进行定时操作时，用户需调用**SetTimer**函数创建定时器，并设置定时器的事件标志**nIDEvent**及时间间隔**nElapse**，然后编写消息**WM_TIMER**的消息处理函数**OnTimer()**，实现定时操作。

□ WM_TIMER方式定时的工作原理



□ MFC中和定时器相关的有三个函数:

1.设置定时器(定义一个定时器的属性):

`SetTimer(UINT nIDEvent, UINT nElapse, void (CALLBACK EXPORT* lpfnTimer)(HWND,UINT,UINT,DWORD));`

- `nIDEvent`: 定时器ID, 确定是哪个定时器发送的消息。
- `nElapse`: 以毫秒为单位的定时时间间隔。
- `lpfnTimer`: 指向定时事件到达时调用的函数指针, 若为 `NULL`, 则调用 `OnTimer()`。

SetTimer(1,200,0): 设置并启动时间间隔为200ms的定时器，消息响应函数为OnTimer。

SetTimer(2,1000,Proc): 设置并启动时间间隔为1s的定时器，该定时器的响应函数为Proc。

2.定时器响应(响应系统定义WM_TIMER消息):

OnTimer(UINT nIDEvent);

3.释放定时器:

KillTimer(int nIDEvent);

定时器例子

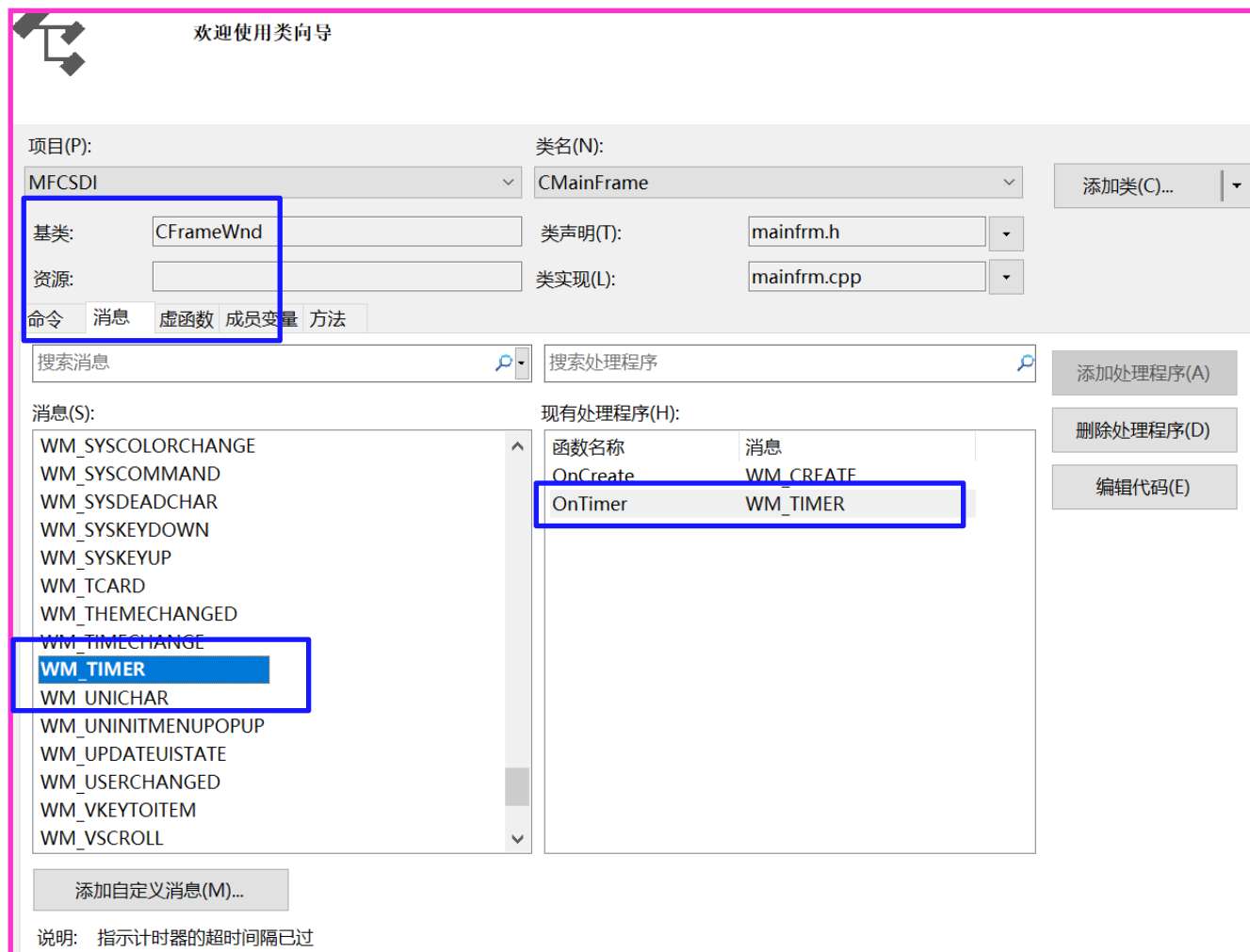
1、在resource.h中定义两个定时器的ID

```
#define TIMER1 1  
#define TIMER2 2
```

2、在CMainFrame的OnCreate函数中定义两个定时器的属性

```
SetTimer(TIMER1,4000,0);  
SetTimer(TIMER2,7000,0);
```

3、CMainFrame类消息中找到WM_TIMER, 添加响应函数OnTimer()



```
void CMainFrame::OnTimer(UINT_PTR nIDEvent)
{
    switch(nIDEvent)
    {
        case TIMER1:
            AfxMessageBox(_T("定时器1!"));break;
        case TIMER2:
            AfxMessageBox(_T("定时器2!"));break;
        default:break;
    }
}
```

nIDEvent表示定时器的序号，不同的定时器通过nIDEvent来标识区别

4、在CMainFrame的析构函数中添加释放定时器函数。

```
CMainFrame::~CMainFrame()
{
    KillTimer(TIMER1);
    KillTimer(TIMER2);
}
```

