# 从这一节开始连续7节都是学习进程间通信

# 学习大纲

|  |
| --- |
|  |

# API说明

## RegisterWindowMessage函数的语法

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 定义保证在整个系统中唯一的新窗口消息。 发送或发布消息时可以使用消息值。 语法 C++复制  UINT RegisterWindowMessageA(  [in] LPCSTR lpString  ); 参数 [in] lpString  类型： **LPCTSTR**  要注册的消息。 返回值 类型： **UINT**  如果成功注册消息，则返回值是0xC000到0xFFFF范围内的消息标识符。  如果函数失败，则返回值为零。 要获得更多的错误信息，请调用 GetLastError。 注解 **RegisterWindowMessage** 函数通常用于注册消息，以便在两个协作应用程序之间进行通信。  如果两个不同的应用程序注册相同的消息字符串，则应用程序将返回相同的消息值。 消息保持注册状态，直到会话结束。  仅当多个应用程序必须处理同一消息时，才使用 **RegisterWindowMessage** 。 若要在窗口类中发送私人消息，应用程序可以使用 [WM\_USER到0x7FFF](https://learn.microsoft.com/zh-cn/windows/desktop/winmsg/wm-user) 范围内的任意整数。 (此范围内的消息是窗口类的专用消息，而不是应用程序专用的。例如，预定义的控件类（如 **BUTTON**、 **EDIT**、 **LISTBOX** 和 **COMBOBOX** ）可能使用此范围中的值。) 示例 有关示例，请参阅 [查找文本](https://learn.microsoft.com/zh-cn/windows/desktop/dlgbox/using-common-dialog-boxes)。  **备注**  winuser.h 标头将 RegisterWindowMessage 定义为别名，该别名根据 UNICODE 预处理器常量的定义自动选择此函数的 ANSI 或 Unicode 版本。 将非特定编码别名与非非特定编码的代码混合使用可能会导致不匹配，从而导致编译或运行时错误。 有关详细信息，请参阅 [**函数原型的约定**](https://learn.microsoft.com/zh-cn/windows/win32/intl/conventions-for-function-prototypes)。 要求  |  |  | | --- | --- | | **标头** | winuser.h (包括 Windows.h) | | **Library** | User32.lib | | **DLL** | User32.dll | | **API 集** | 在 Windows 8) 中引入的 ext-ms-win-ntuser-message-l1-1-0 ( |  请参阅 **概念性**  [消息和消息队列](https://learn.microsoft.com/zh-cn/windows/desktop/winmsg/messages-and-message-queues)  [PostMessage](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-postmessagea)  **引用**  [SendMessage](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-sendmessage) |

## \_ttoi函数的语法

# 演练

## 我们这一节需要两个项目，一个负责发送消息，另外一个复制接收消息

## 1.新建一个文件夹取名Lesson66-processes-com，然后在里面新建一个MFC对话框应用程序，取名MsgSender

|  |
| --- |
|  |

## 2.在这个解决方案里面添加一个MFC对话框应用程序项目，取名：MsgReveiver

|  |
| --- |
|  |

## 3.把MsgSender项目的对话框里面的默认控件删除，然后我们给这个对话框类添加一个PreTranslateMessage函数，其实是重写，用来屏蔽回车键和esc键

|  |
| --- |
|  |

## 4.把MsgSender设置位启动项目，然后编译运行，按下回车键，发现没有作用，说明回车键成功屏蔽。

|  |
| --- |
|  |

## 5.我们也把MsgReceiver项目的默认控件删除，然后重新PreTranslateMessage函数

|  |
| --- |
|  |
|  |

## 6.把MsgReceiver项目设置为启动项目，然后测试按下回车，也没有反应，说明设置成功

|  |
| --- |
|  |

## 7.然后我们把MsgSender项目设置为启动项目

|  |
| --- |
|  |

## 8.给发送对话框添加一个静态文本，内容为：消息，然后添加一个编辑框，id为：IDC\_EDIT\_MSG,然后添加一个按钮，文本为发送，id为：IDC\_BTN\_SEND

|  |
| --- |
|  |
|  |

## 9.给编辑框绑定一个CEdit变量，方便编程

|  |
| --- |
|  |

## 10然后可以双击按钮进入点击事件处理函数，先处理用户没有输入，直接点击按钮的情况，代码如下：

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |
|  |

### 说明代码是正确的。

## 11.由于发送消息有2种方法，我们需要修改一下发送对话框的界面，把上面的静态文本内容改为WM\_USER消息，然后把这三个控件赋值一份,修改如下

|  |
| --- |
|  |
|  |

## 12.然后给第二个编辑框绑定一个成员变量

|  |
| --- |
|  |
|  |

## 13.然后给第二个按钮的点击事件也做编辑框判空处理

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |
|  |

## 14.我们先来做用户信息，先定义一个用户消息

|  |
| --- |
|  |

### 注意，我们这个程序比较简单，是用来发送整数的。

## 15.我们先完成第一个按钮的事件事件处理函数

|  |
| --- |
| void CMsgSenderDlg::OnBnClickedBtnSend()  {  // TODO: 在此添加控件通知处理程序代码  CString str;  GetDlgItemText(IDC\_EDIT\_MSG,str);  if(0==str.GetLength())  {  MessageBox(\_T("请输入消息！！"));  m\_edit\_msg.SetFocus();  return;//用户没有输入，程序就不要往下执行  }  //MessageBox(str);  //需要使用FindWindow函数找到需要接收消息的窗口的指针  CWnd\* pwnd = CWnd::FindWindow(NULL,\_T("MsgReceiver"));  if(pwnd == NULL)  {  MessageBox(\_T("查找窗口失败"));  return;  }  UINT msg;  msg = \_ttoi(str);  //给目标窗口发送消息  ::SendMessage(pwnd->m\_hWnd,WM\_COMM,NULL,(LPARAM)msg);  } |

### 测试一下，此时会找不到窗口，因为目标程序没有运行

|  |
| --- |
|  |

## 16.我们来做接收端，给接收对话框添加两个静态文本和两个编辑框控件

|  |
| --- |
|  |
|  |

## 17.给这两个编辑框控件绑定两个CString类型的成员变量

|  |
| --- |
|  |
|  |
|  |

## 19.先别着急写代码，我们编译两个项目，然后把发送程序和接收程序都启动起来，然后用发送查询网接收程序发送一条消息，看看还有没有报找不到窗口的错误，经过测试发现没有这个错误了

|  |
| --- |
|  |

## 20.我们来给接收程序添加接收并且处理信息的代码，首先，需要在接收程序里面定义一个和发送程序一样的消息

|  |
| --- |
|  |

## 21.然后需要给这个消息做消息映射

|  |
| --- |
|  |

## 22.然后在接收对话框的头文件里面声明这个信息响应函数

|  |
| --- |
|  |

## 23.然后添加这个消息处理函数的实现代码

|  |
| --- |
|  |

## 24.把这两个项目重新生成一下，然后把他们都启动，在发送程序的第一个消息框里面输入9999，然后点击发送按钮，效果如下

|  |
| --- |
|  |

## 上面这种方法只能发送整数，我们来看看注册信息的方式

## 25.回到发送对话框，我们来实现第二个按钮的点击事件处理代码，我们来用注册消息的办法来发送消息，先在cpp文件的开头调用注册信息API注册消息一个常量

|  |
| --- |
|  |

## 26.然后我们来完成按钮点击代码，其实和上面的几乎一模一样，整数改了消息

|  |
| --- |
| void CMsgSenderDlg::OnBnClickedBtnRegSend()  {  // TODO: 在此添加控件通知处理程序代码  CString str;  GetDlgItemText(IDC\_EDIT\_MSG\_REG,str);  if(0==str.GetLength())  {  MessageBox(\_T("请输入消息！！"));  m\_edit\_reg\_msg.SetFocus();  return;//用户没有输入，程序就不要往下执行  }  //MessageBox(str);  //需要使用FindWindow函数找到需要接收消息的窗口的指针  CWnd\* pwnd = CWnd::FindWindow(NULL,\_T("MsgReceiver"));  if(pwnd == NULL)  {  MessageBox(\_T("查找窗口失败"));  return;  }  UINT msg;  msg = \_ttoi(str);  //给目标窗口发送消息  ::SendMessage(pwnd->m\_hWnd,wm\_RegMsg,NULL,(LPARAM)msg);//使用我们注册的消息  } |

## 27.然后在接收对话框里面也响应一样的定义常量代码

|  |
| --- |
|  |

## 28.注意:用RegisterWindowMessage方法生成的消息，在接受方的消息映射里面就不能使用ON\_MESSAGE宏来处理，它需要使用ON\_REGISTERED\_MESSAGE宏来处理

|  |
| --- |
|  |

## 29.下面的步骤都是一样的

|  |
| --- |
|  |
|  |

### 效果

|  |
| --- |
|  |

# 这一节的学习到此为止，完整代码如下主要是对话框类的代码，因为应用程序类不需要操作

# 发送程序

## MsgSenderDlg.h

|  |
| --- |
| // MsgSenderDlg.h : 头文件  //  #pragma once  #include "afxwin.h"  // CMsgSenderDlg 对话框  class CMsgSenderDlg : public CDialogEx  {  // 构造  public:  CMsgSenderDlg(CWnd\* pParent = NULL); // 标准构造函数  // 对话框数据  enum { IDD = IDD\_MSGSENDER\_DIALOG };  protected:  virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持  // 实现  protected:  HICON m\_hIcon;  // 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()  public:  virtual BOOL PreTranslateMessage(MSG\* pMsg);  afx\_msg void OnBnClickedBtnSend();  // 绑定编辑框控件的成员变量  CEdit m\_edit\_msg;  // 给第二个编辑框控件绑定成员变量  CEdit m\_edit\_reg\_msg;  afx\_msg void OnBnClickedBtnRegSend();  }; |

## MsgSenderDlg.cpp

|  |
| --- |
| // MsgSenderDlg.cpp : 实现文件  //  #include "stdafx.h"  #include "MsgSender.h"  #include "MsgSenderDlg.h"  #include "afxdialogex.h"  #ifdef \_DEBUG  #define new DEBUG\_NEW  #endif  //定义消息常量  #define WM\_COMM WM\_USER+100  //方式2,用window API生成一个消息常量  const UINT wm\_RegMsg = RegisterWindowMessage(\_T("reg\_data"));  // 用于应用程序“关于”菜单项的 CAboutDlg 对话框  class CAboutDlg : public CDialogEx  {  public:  CAboutDlg();  // 对话框数据  enum { IDD = IDD\_ABOUTBOX };  protected:  virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持  // 实现  protected:  DECLARE\_MESSAGE\_MAP()  };  CAboutDlg::CAboutDlg() : CDialogEx(CAboutDlg::IDD)  {  }  void CAboutDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  }  BEGIN\_MESSAGE\_MAP(CAboutDlg, CDialogEx)  END\_MESSAGE\_MAP()  // CMsgSenderDlg 对话框  CMsgSenderDlg::CMsgSenderDlg(CWnd\* pParent /\*=NULL\*/)  : CDialogEx(CMsgSenderDlg::IDD, pParent)  {  m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);  }  void CMsgSenderDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  DDX\_Control(pDX, IDC\_EDIT\_MSG, m\_edit\_msg);  DDX\_Control(pDX, IDC\_EDIT\_MSG\_REG, m\_edit\_reg\_msg);  }  BEGIN\_MESSAGE\_MAP(CMsgSenderDlg, CDialogEx)  ON\_WM\_SYSCOMMAND()  ON\_WM\_PAINT()  ON\_WM\_QUERYDRAGICON()  ON\_BN\_CLICKED(IDC\_BTN\_SEND, &CMsgSenderDlg::OnBnClickedBtnSend)  ON\_BN\_CLICKED(IDC\_BTN\_REG\_SEND, &CMsgSenderDlg::OnBnClickedBtnRegSend)  END\_MESSAGE\_MAP()  // CMsgSenderDlg 消息处理程序  BOOL CMsgSenderDlg::OnInitDialog()  {  CDialogEx::OnInitDialog();  // 将“关于...”菜单项添加到系统菜单中。  // IDM\_ABOUTBOX 必须在系统命令范围内。  ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);  ASSERT(IDM\_ABOUTBOX < 0xF000);  CMenu\* pSysMenu = GetSystemMenu(FALSE);  if (pSysMenu != NULL)  {  BOOL bNameValid;  CString strAboutMenu;  bNameValid = strAboutMenu.LoadString(IDS\_ABOUTBOX);  ASSERT(bNameValid);  if (!strAboutMenu.IsEmpty())  {  pSysMenu->AppendMenu(MF\_SEPARATOR);  pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);  }  }  // 设置此对话框的图标。当应用程序主窗口不是对话框时，框架将自动  // 执行此操作  SetIcon(m\_hIcon, TRUE); // 设置大图标  SetIcon(m\_hIcon, FALSE); // 设置小图标  // TODO: 在此添加额外的初始化代码  return TRUE; // 除非将焦点设置到控件，否则返回 TRUE  }  void CMsgSenderDlg::OnSysCommand(UINT nID, LPARAM lParam)  {  if ((nID & 0xFFF0) == IDM\_ABOUTBOX)  {  CAboutDlg dlgAbout;  dlgAbout.DoModal();  }  else  {  CDialogEx::OnSysCommand(nID, lParam);  }  }  // 如果向对话框添加最小化按钮，则需要下面的代码  // 来绘制该图标。对于使用文档/视图模型的 MFC 应用程序，  // 这将由框架自动完成。  void CMsgSenderDlg::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();  }  }  //当用户拖动最小化窗口时系统调用此函数取得光标  //显示。  HCURSOR CMsgSenderDlg::OnQueryDragIcon()  {  return static\_cast<HCURSOR>(m\_hIcon);  }  BOOL CMsgSenderDlg::PreTranslateMessage(MSG\* pMsg)  {  // TODO: 在此添加专用代码和/或调用基类  if (pMsg->message == WM\_KEYDOWN)  {  switch (pMsg->wParam)  {  case VK\_RETURN: //屏蔽回车键  return TRUE;  case VK\_ESCAPE: //屏蔽ESC键  return TRUE;    default:  break;  }  }  return CDialogEx::PreTranslateMessage(pMsg);  }  void CMsgSenderDlg::OnBnClickedBtnSend()  {  // TODO: 在此添加控件通知处理程序代码  CString str;  GetDlgItemText(IDC\_EDIT\_MSG,str);  if(0==str.GetLength())  {  MessageBox(\_T("请输入消息！！"));  m\_edit\_msg.SetFocus();  return;//用户没有输入，程序就不要往下执行  }  //MessageBox(str);  //需要使用FindWindow函数找到需要接收消息的窗口的指针  CWnd\* pwnd = CWnd::FindWindow(NULL,\_T("MsgReceiver"));  if(pwnd == NULL)  {  MessageBox(\_T("查找窗口失败"));  return;  }  UINT msg;  msg = \_ttoi(str);  //给目标窗口发送消息  ::SendMessage(pwnd->m\_hWnd,WM\_COMM,NULL,(LPARAM)msg);  }  void CMsgSenderDlg::OnBnClickedBtnRegSend()  {  // TODO: 在此添加控件通知处理程序代码  CString str;  GetDlgItemText(IDC\_EDIT\_MSG\_REG,str);  if(0==str.GetLength())  {  MessageBox(\_T("请输入消息！！"));  m\_edit\_reg\_msg.SetFocus();  return;//用户没有输入，程序就不要往下执行  }  //MessageBox(str);  //需要使用FindWindow函数找到需要接收消息的窗口的指针  CWnd\* pwnd = CWnd::FindWindow(NULL,\_T("MsgReceiver"));  if(pwnd == NULL)  {  MessageBox(\_T("查找窗口失败"));  return;  }  UINT msg;  msg = \_ttoi(str);  //给目标窗口发送消息  ::SendMessage(pwnd->m\_hWnd,wm\_RegMsg,NULL,(LPARAM)msg);//使用我们注册的消息  } |

## 接收程序

## MsgReceiverDlg.h

|  |
| --- |
| // MsgReceiverDlg.h : 头文件  //  #pragma once  // CMsgReceiverDlg 对话框  class CMsgReceiverDlg : public CDialogEx  {  // 构造  public:  CMsgReceiverDlg(CWnd\* pParent = NULL); // 标准构造函数  // 对话框数据  enum { IDD = IDD\_MSGRECEIVER\_DIALOG };  protected:  virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持  // 实现  protected:  HICON m\_hIcon;  // 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();  afx\_msg LRESULT OnUserMsg(WPARAM,LPARAM);//注意这个消息响应函数是要求返回LRESULT类型，不能是void  afx\_msg LRESULT OnRegMsg(WPARAM,LPARAM);//处理用户自己注册的消息的响应函数  DECLARE\_MESSAGE\_MAP()  public:  virtual BOOL PreTranslateMessage(MSG\* pMsg);  // 绑定第一个编辑框的成员变量  CString m\_msgUser;  // 绑定第二个编辑框控件的成员变量  CString m\_msgReg;    }; |

## MsgReceiverDlg.cpp

|  |
| --- |
| // MsgReceiverDlg.cpp : 实现文件  //  #include "stdafx.h"  #include "MsgReceiver.h"  #include "MsgReceiverDlg.h"  #include "afxdialogex.h"  #ifdef \_DEBUG  #define new DEBUG\_NEW  #endif  #define WM\_COMM WM\_USER+100 //定义和发送程序一样的消息,注意不要有分号  //方式2,用window API生成一个消息常量，需要分号  const UINT wm\_RegMsg = RegisterWindowMessage(\_T("reg\_data"));  // 用于应用程序“关于”菜单项的 CAboutDlg 对话框  class CAboutDlg : public CDialogEx  {  public:  CAboutDlg();  // 对话框数据  enum { IDD = IDD\_ABOUTBOX };  protected:  virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持  // 实现  protected:  DECLARE\_MESSAGE\_MAP()  };  CAboutDlg::CAboutDlg() : CDialogEx(CAboutDlg::IDD)  {  }  void CAboutDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  }  BEGIN\_MESSAGE\_MAP(CAboutDlg, CDialogEx)  END\_MESSAGE\_MAP()  // CMsgReceiverDlg 对话框  CMsgReceiverDlg::CMsgReceiverDlg(CWnd\* pParent /\*=NULL\*/)  : CDialogEx(CMsgReceiverDlg::IDD, pParent)  , m\_msgUser(\_T(""))  , m\_msgReg(\_T(""))  {  m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);  }  void CMsgReceiverDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  DDX\_Text(pDX, IDC\_EDIT\_USERMSG, m\_msgUser);  DDX\_Text(pDX, IDC\_EDIT\_REGMSG, m\_msgReg);  }  BEGIN\_MESSAGE\_MAP(CMsgReceiverDlg, CDialogEx)  ON\_WM\_SYSCOMMAND()  ON\_WM\_PAINT()  ON\_WM\_QUERYDRAGICON()  ON\_MESSAGE(WM\_COMM,OnUserMsg)//这里只是建立映射，响应在头文件声明这个函数在cpp文件实现它  ON\_REGISTERED\_MESSAGE(wm\_RegMsg,OnRegMsg)//用户自己注册的消息不能用ON\_MESSAGE宏，没有效果  END\_MESSAGE\_MAP()  // CMsgReceiverDlg 消息处理程序  BOOL CMsgReceiverDlg::OnInitDialog()  {  CDialogEx::OnInitDialog();  // 将“关于...”菜单项添加到系统菜单中。  // IDM\_ABOUTBOX 必须在系统命令范围内。  ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);  ASSERT(IDM\_ABOUTBOX < 0xF000);  CMenu\* pSysMenu = GetSystemMenu(FALSE);  if (pSysMenu != NULL)  {  BOOL bNameValid;  CString strAboutMenu;  bNameValid = strAboutMenu.LoadString(IDS\_ABOUTBOX);  ASSERT(bNameValid);  if (!strAboutMenu.IsEmpty())  {  pSysMenu->AppendMenu(MF\_SEPARATOR);  pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);  }  }  // 设置此对话框的图标。当应用程序主窗口不是对话框时，框架将自动  // 执行此操作  SetIcon(m\_hIcon, TRUE); // 设置大图标  SetIcon(m\_hIcon, FALSE); // 设置小图标  // TODO: 在此添加额外的初始化代码  return TRUE; // 除非将焦点设置到控件，否则返回 TRUE  }  void CMsgReceiverDlg::OnSysCommand(UINT nID, LPARAM lParam)  {  if ((nID & 0xFFF0) == IDM\_ABOUTBOX)  {  CAboutDlg dlgAbout;  dlgAbout.DoModal();  }  else  {  CDialogEx::OnSysCommand(nID, lParam);  }  }  // 如果向对话框添加最小化按钮，则需要下面的代码  // 来绘制该图标。对于使用文档/视图模型的 MFC 应用程序，  // 这将由框架自动完成。  void CMsgReceiverDlg::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();  }  }  //当用户拖动最小化窗口时系统调用此函数取得光标  //显示。  HCURSOR CMsgReceiverDlg::OnQueryDragIcon()  {  return static\_cast<HCURSOR>(m\_hIcon);  }  BOOL CMsgReceiverDlg::PreTranslateMessage(MSG\* pMsg)  {  // TODO: 在此添加专用代码和/或调用基类  if(pMsg->message == WM\_KEYDOWN)  {  switch(pMsg->wParam)  {  case VK\_RETURN:  return TRUE;  case VK\_ESCAPE:  return TRUE;  }  }  return CDialogEx::PreTranslateMessage(pMsg);  }  LRESULT CMsgReceiverDlg::OnUserMsg(WPARAM wParam,LPARAM lParam)  {  UINT msg = lParam;  m\_msgUser.Format(\_T("消息内容:%d"),msg);  UpdateData(FALSE);  return 0;  }  LRESULT CMsgReceiverDlg::OnRegMsg(WPARAM wParam,LPARAM lParam)  {  UINT msg = lParam;  m\_msgReg.Format(\_T("消息内容:%d"),msg);  UpdateData(FALSE);  return 0;  } |