第三部分　内部处理实例

并非所有的应用程序都有交互界面，如读写文件和时间事件等大量的处理是在内部进行的。虽然MFC给人的印象是一个界面开发系统，但是 MFC库中的许多类也支持应用程序的非交互部分。

本部分中的例子涉及应用程序内部的各种处理。这些处理包括在应用程序内部和外部发送消息、读写磁盘上或内存中的文件等。还有一些例子涉及维护列表或数组中的数据、剪贴屏幕数据和基于时间的播放等处理。

消息

第12章的例子都是讲述在应用程序内部和外部发送数据和消息。这些例子包括子分类、超分类的生成以及创建自己的消息类型等等。

文件、串行化和数据库

所有相关的问题在第13章都已涉及，从平面文件到访问主要数据库供应商提供的数据库。当然也包括串行化数据，以便更好地组织和修改数据。

杂类

第14章是关于其他内部处理的内容，如剪切、粘贴、列表、数组、时间等等。

第12章　消　　息

若假定窗口应用程序自身由许多潜在自主的窗口组成，并给定 C++封装和消息处理要求的情况下，一个MFC应用程序变成一个复杂的问题。从技术上讲，在窗口应用程序中，消息只能从一个窗口发送到另外一个窗口。那么，怎样让这些消息进入到 C++类的内核且遵循MFC机制呢？要了解这方面的详细情况，请参阅第 3章。

例59　添加消息处理函数或重载MFC类　利用Class Wizard进入MFC消息处理系统，让一特定消息进入相应的类成员函数。 Class Wizard允许自动重载其所继承的MFC类中的任何虚函数。

例60　添加命令范围消息处理函数　手动添加一个消息处理函数到一个类中，它能够处理所有的命令消息，因此，我们不必利用 Class Wizard为每一个命令消息都添加一处理函数。

例61　重定向命令消息　讨论怎样定向一个命令消息到一个类中，而通常该消息不被传输到该类。

例62 创建自己的窗口消息　创建一个可以在应用程序中进行控制的消息。

12.1 例59 添加消息处理函数或重载MFC类

目标

在类中，添加一个消息处理函数或者重载一个 MFC成员函数。

策略

首先，用Class Wizard自动地添加一个消息处理函数或重载成员函数；然后，讨论当需要的处理函数或重载函数在Class Wizard指令系统的外面时，怎样手工完成该工作。

步骤

1. 利用Class Wizard添加消息处理函数或重载类

1) 单击Developer Studio的View/Class Wizard菜单命令，打开Class Wizard。确定想添加消息处理函数或重载函数的类，并从 Class name组合框中选取。在Object IDs列表框中，选取类的名称(其他项是菜单和工具栏命令 ID，这些在例13中已讨论)。此时，Messages列表框显示该类的大多数(若不是全部的话)可重载成员函数和窗口消息。类重载显示在列表的上部，以实际虚构成员函数的大小写字母来表示。其他为窗口消息，以大写字母出现，描述了实际窗口所能响应的消息 ID。为了添加所需的重载函数或处理函数，单击 Add Function按钮。Class Wizard将添加该函数到它维护的源代码的表中，并附简单注释 (见图12-1)

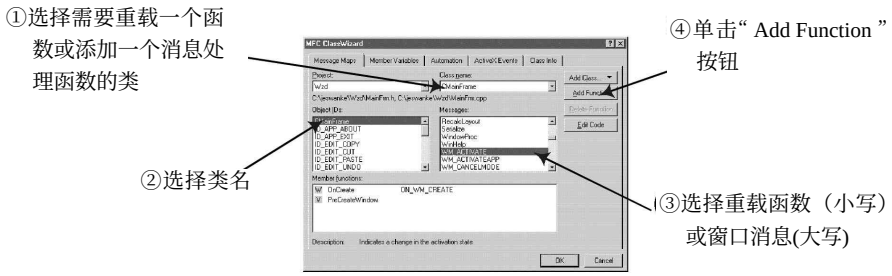

图12-1 使用ClassWizard添加一个消息处理函数

2) 若在Message列表中找不到所要的消息或重载函数，则可以选取 Class Wizard上Class Info标签以扩展消息列表。在该页中，找到 Message Filter组合框，可以改变首页中 Messages列表框中的选项。例如，选择 Not a Window，将从消息列表中去掉所有的窗口消息 (从技术上讲，只有窗口能接收消息，因此没必要添加消息处理函数)；选择Select Window会显示所有的窗口消息，如图12-2所示。

注意 若在Messages列表框中没有找到相应的处理函数或重载函数，那可能是该处理函数或重载函数对类是无效的。然而，有时ClassWizard会简单地忽略一些消息，因为它们根本就很少用。

3) 要删除一个处理函数或重载函数，从 Message列表中选中它并单击Delete Function按钮。实际上，所有ClassWizard能删除的都是在它所维护的源文件表中处于 {[ ]}括号之间的条目，

因此，有必要把该函数的其他代码删掉。即使临时地添加一个处理函数，并立即用ClassWizard删除它，也必须定位到该空函数，并手动地删除它。

如果一个重载函数或窗口消息没有列在“Message Maps”选项卡中，则尝试在该组合框中选中选择别的过滤器

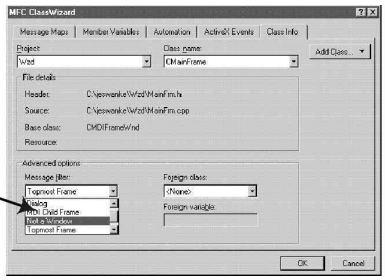

图12-2 使用新的消息过滤器以改变在消息列表框中的选择

2. 手动地添加消息处理函数

如果ClassWizard并没有列出想要处理的消息，该消息可能是被系统忽略掉或者是你自己创建的，这时，必须自己手工添加处理函数。一个消息处理函数需要对类作三处编辑：声明和实现处理函数、进入类的消息映像中的处理函数。

1) 在类的.h文件中添加处理函数的声明，紧接在 //}}AFX_MSG行之后加入声明，这是类的.h文件中的消息映像区，通过添加之后，看起来应该如下所示 (本例中消息处理函数称为“OnDestroy()”)：

```
// Generated message map functions
protected:
   //{{AFX_MSG(CWzdClass)
   //}}AFX_MSG
   afx_msg void OnDestroy(); <<<< add declarations after "}}"
   DECLARE_MESSAGE_MAP()
```

通常，添加处理函数声明的最好的地方是源代码中 ClassWizard维护的表下面，但是在它标记其领域的 {{}} 括弧外面。这些括弧中的任何东西都将会被 ClassWizard销毁。

2) 接着，在用户类的 .cpp文件中的消息映像区中，添加一个消息映像宏。定位 //}}AFX_MSG_MAP行，紧接在它之后加入宏。

```
BEGIN_MESSAGE_MAP(CWzdClass, CClass)
   //{{AFX_MSG_MAP(CWzdClass)
   //}}AFX_MSG_MAP
   ON_WM_DESTROY()  //<<<< add macros after "}}"
END_MESSAGE_MAP()
```

要想了解消息映像宏的全部内容，参见附录 B。

3) 然后，可以在该文件中添加消息处理函数的实体。

```
void CWzdClass::OnDestroy()
{

}
```

对于一特定的消息映像宏，参见附录 B中有关详细的调用语法。

3. 手动地添加一个类重载函数

1) 要想手工添加一个重载函数，只须对文件进行两处编辑。由于是从其基类直接获取控制，而基类可能自身正对消息进行处理，所以不必在消息映像中添加任何内容。首先，在 .h文

件中ClassWizard维护的表处添加重载函数的声明。(本例中，重载函数PreCreateWindow( )。)

```
// Overrides
   // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CWzdClass)
   public:
      :  :  :
   //}}AFX_VIRTUAL
   public:                    << make sure to match virtual
                              << function's protection
   virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
```

2) 然后，在相应的.cpp文件中加入重载函数实体，如例所示。当然，也可以有选择地调用基类的函数。

```
BOOL CWzdClass::PreCreateWindow(CREATESTRUCT& cs)
{
   return CClass::PreCreateWindow(cs);
}
```

说明

- 实际上，有许多种方法激发 ClassWizard。通常，用鼠标右击类的资源文件，将打开ClassWizard并显示该类。也可以通过单击 Tools/Customize菜单命令，打开环境的Customize对话框以创建一个工具栏按钮。在 Commands标签下，有一个Categories列表框，在列表框中选择 View。ClassWizard是此处所列的按钮之一，可以把它拖进任何一个已存在的工具栏，或者拖进一个空档处以创建一个新的工具栏。
- 当添加自己的消息处理函数，或者使用过滤器以展开可以添加的消息处理函数列表时，这个类可能没有权利接收该消息。这就是说，即使 ClassWizard允许添加某个消息处理函数，但是类可能没有资格接收该消息。详细情况参见第 3章和例61有关添加类到命令消息路径下的方法。
- 当手工添加一个重载函数到用户类时，其基类必须有相应的被重载的函数。好在如果不能重载，编译器会立即告诉你的。但是对消息处理函数就不一定如此，只有当运行测试时，你才可能发现他添加了一个消息处理函数到一个根本不接收该消息的类中。
- 如果没有被文档化， ClassWizard将不列出即使是可重载的函数。可以在 MFC提供的源代码中找到这些未文档化的重载函数。然而，既然它们没有被文档化，微软公司在以后的MFC发行版本中也不会支持这些重载函数。若必须要使用未文档化的重载函数，当移植到MFC的下一个版本时，就无法再访你所写的代码。
- 当使用ClassWizard添加一个消息处理函数到自己的类中时， ClassWizard一般会加入调用基类相应函数的代码，并且把用户消息处理函数接收到的参数传递给它。

```
void CAppView::OnLButtonDown(UINT nFlags, CPoint point)
{
   // TODO: Add your message handler code here and/or call default

   CView::OnLButtonDown(nFlags, point);
}
```

不要嫌烦改变这些参数以影响基类的行为。这些参数放在这儿只不过添加麻烦而已。当

基类获取这些参数时，它不会对此置之不理，相反，它将使用最开始传给你的消息。

在消息未被基类处理之前，若确实想修改这个消息或者修改缺省的窗口进程，则需要重新构造原来的消息，修改其中的参数 wParam和lParam，不是调用基类函数，而是直接调用DefWindowProc( )。要想知道如何构造关于 DefWindowProc( )函数的消息参数，参见 MFC的关于该消息的说明文档。在本例中，可以用：

```
DefWindowProc (WM_LBUTTONDOWN, nFlags, Point);
```

■ 为了获取基类当前正在处理的消息，用户可以使用如下语句：

```
MSG msg=CWnd::GetCurrentMessage( );
```

这里，MSG是如下形式的一个消息结构：

```
typedef struct tagMSG {
    HWND   hwnd;
    UINT   message;
    WPARAM wParam;
    LPARAM lParam;
    DWORD  time;
    POINT  pt;
} MSG;
```

■ 然后，可以用获取的值调用 DefWindowProc( )。

```
MSG msg=CWnd::GetCurrentMessage();
msg.wParam = ...changed...
DefWindowProc(msg.message,msg.wParam,msg.lParam);
```

CD说明

在CD上没有该例的相应工程。

12.2 例60 添加命令范围消息处理函数

目标

在类中添加处理某个范围的命令消息或者修改命令接口的处理函数。

策略

手工添加两个消息映像宏： ON_COMMAND_RANGE和ON_UPDATE_COMMAND_UI_RANGE到用户类的消息映像中。当前 ClassWizard并不能够处理其中任意一个宏。

步骤

1. 使用命令范围宏

ON_COMMAND_RANGE宏接收所有ID在某一范围之内的命令消息 (WM_COMMAND消息)。使用这个宏可以在一个函数中处理多条命令。

1) 在类的.h文件的{{ }}之后定义命令处理函数，以便不受 ClassWizard的干扰。

```
protected:
    //{{AFX_MSG(CWzdView)
```

```
    //}}AFX_MSG
    afx_msg void OnWzdCommandRange(UINT nID);<<<<<<
    DECLARE_MESSAGE_MAP()
};
```

2) 同样在{{ }}之后，加入ON_COMMAND_RANGE宏到类的消息映像。头两个参数定义了要处理的命令ID的范围。这些ID有序列大小，且最后一个ID比第一个大。最后一个参数是在第一步中定义的消息处理函数的名称。

```
BEGIN_MESSAGE_MAP(CWzdView, CView)
    //{{AFX_MSG_MAP(CWzdView)
    //}}AFX_MSG_MAP
    ON_COMMAND_RANGE(ID_TEST_WZD1,ID_TEST_WZD4,OnWzdCommandRange)
END_MESSAGE_MAP()
```

3) 用下面的语句添加消息处理函数。参数nID是要处理命令的ID。

```
void CWzdView::OnWzdCommandRange(UINT nID)
{
    switch (nID)
    {
    case ID_TEST_WZD1:
        break;
    case ID_TEST_WZD2:
        break;
    case ID_TEST_WZD3:
        break;
    case ID_TEST_WZD4:
        break;
    }
}
```

2. 使用用户界面命令范围宏

ON_UPDATE_COMMAND_UI_RANGE宏截取一定范围内的消息中更新用户界面的请求。使用这个宏，可以启用所有的菜单命令或者处理一组工具栏按钮。

1) 在类的.h文件映像的{{ }}之后定义用户界面命令处理函数，以便不受ClassWizard的干扰。

```
protected:
    //{{AFX_MSG(CWzdView)
    //}}AFX_MSG
    afx_msg void OnUpdateWzdCommandRange(CCmdUI* pCCmdUI);
    DECLARE_MESSAGE_MAP()
};
```

2) 同样，在{{ }}之后，加入ON_UPDATE_COMMAND_UI_RANGE宏到用户类的命令映像中。头两个参数定义了要处理的消息 ID的范围。这些ID有序列大小，最后一个ID值比第一个大。最后一个参数是第一步中定义的用户界面命令处理函数的名字。

```
BEGIN_MESSAGE_MAP(CWzdView, CView)
    //{{AFX_MSG_MAP(CWzdView)
    //}}AFX_MSG_MAP
    ON_UPDATE_COMMAND_UI_RANGE(ID_TEST_WZD1,ID_TEST_WZD4,
        OnUpdateWzdCommandRange)
END_MESSAGE_MAP()
```

3) 处理界面命令消息的语句如下所示。正如所看到的，用户可以根据 CCmdUI类的成员

变量m_nID的值决定更新哪个菜单项、工具栏按钮等。

```
void CWzdView::OnUpdateWzdCommandRange(CCmdUI *pCmdUI)
{
   switch (pCmdUI->m_nID)
   {
      case ID_TEST_WZD1:
         break;
      case ID_TEST_WZD2:
         pCmdUI->SetRadio();
         break;
      case ID_TEST_WZD3:
         break;
      case ID_TEST_WZD4:
         break;
   }
}
```

说明

- 命令范围宏一般用来处理一组命令消息，这些消息处理的差别只在于被调用的命令的不同。这些宏可以大大简化类，否则类中就需要很多调用相同函数的多个消息处理函数。
- 另外两个较少使用的消息宏是ON_CONTROL_RANGE和ON_NOTIFY_RANGE，它们用于较新或较旧类型的控制通知。详细内容参见第3章和附录B。
- 处理任意类型的某一范围内消息的另外一种方法是重载用户的 CWnd类的消息函数(WindowProc( )、OnCommand( )、OnNotify( )等)。关于这些消息函数的详细内容参见第3章。

CD说明

在CD运行该工程时，应注意到在同一个 OnWzdCommandRange( )函数中处理了4个菜单命令，该函数定义在WzdView.cpp中。

12.3 例61 重定向命令消息

目标

让正常情况下不接受命令消息的对话条、无模式对话框或者其他类接收主菜单命令。

策略

在CMainFrame类中，使用 ClassWizard重载 OnCmdMsg( )函数，这里我们将在OnCmdMsg( )中调用想使之接收命令消息的类的OnCmdMsg( )。

步骤

重定向命令消息

1) 用ClassWizard在CMainFrame中重载OnCmdMsg( )。

2) 使用如下语句来实现该重载函数，其中 m_WzdDialogBar是目标类的对象。

```
BOOL CMainFrame::OnCmdMsg(UINT nID, int nCode, void* pExtra,
   AFX_CMDHANDLERINFO* pHandlerInfo)
{
   if (m_WzdDialogBar.OnCmdMsg(nID,nCode,pExtra,pHandlerInfo))
      return TRUE;

   return CMDIFrameWnd::OnCmdMsg(nID, nCode, pExtra,
      pHandlerInfo);
}
```

说明

- 主菜单和工具栏命令被自动地定向到 Frame、Document、View和Application类中。必须自己使用这种方法重定向这些命令到其他类中。
- 只有从CCmdTarget派生的类才有资格接收命令消息。当要求处理一个命令消息时，该类进行实际的消息映像处理，用户在新目标类中调用的 OnCmdMsg( )函数实际上是CCmdTarget的一个成员函数。
- 关于命令消息的自动定向和其他消息机制的详细内容参见第 3章；利用ClassWizard增加一个重载函数的例子参见例59。

CD说明

在CD上执行该工程时，应注意到在 Test菜单中Wzd菜单命令是在CWzdDialogBar类中被OnWzdOptions( )函数处理。

12.4 例62 创建自己的窗口消息

目标

在应用程序中创建自己的窗口消息。

策略

定义一个新的窗口消息，实际上是一个数字 ID。然后，讨论在应用程序的类之间发送和接收该窗口消息。

步骤

1. 创建一个新的窗口消息

使用以下方式定义一个新的消息。若想定义多个消息，只要使用 WM_USER的更大增量即可。

```
# define WM_WZD_MESSAGE WM_USER+1
```

2. 发送一个新的窗口消息

利用SendMessage( )发送新消息：

```
AfxGetMainWnd( )-> SendMessage (WM_WZD_MESSAGE, wParam, lParam);
```

也可以自己定义存储在参数 wParam和lParam中的值，二者都是DWORD变量。有两种方法来接收这个新窗口消息：第一种方法是增加 ON_MESSAGE宏和消息处理函数本身到接收消息类的消息映像中；第二种方法是为此新消息创建自己的消息映像宏。第一种的好处是快而且简单，第二种方法的好处是在消息进行处理之前用户可以定制 wParam和lParam参数。如果想在整个应用程序中使用这个消息，这一点很重要。这两种方法都不能利用 ClassWizard所提供的便利，因为新消息并不在ClassWizard的列表中。

3. 使用ON_MESSAGE( )接收新窗口消息

1) 使用文本编辑器，在接收消息的类中加入消息处理函数的定义。利用如下语法，并保证是在ClassWizard的{{ }}框架之后，以防止ClassWizard删去这里所做的工作。

```
//{{AFX_MSG(CMainFrame)
//}}AFX_MSG
afx_msg LRESULT OnMyMessage(WPARAM wParam, LPARAM lParam);<<<<
DECLARE_MESSAGE_MAP()
```

2) 然后，在实现文件中，添加消息映像宏到消息映像中。

```
BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
   //{{AFX_MSG_MAP(CMainFrame)
   //}}AFX_MSG_MAP
   ON_MESSAGE(WM_WZD_MESSAGE,OnMyMessage)<<<<<
END_MESSAGE_MAP()
```

3) 最后，加入消息处理函数。

```
LRESULT CMainFrame::OnWzdMessage(WPARAM wParam, LPARAM lParam)
{
   //process
   return 0;
}
```

正如传送的wParam和lParam的值由你决定一样，该处理函数返回什么仍由你来决定。

4. 创建新的消息映像宏

1) 用如下方式为新消息定义一个新的消息映像宏，理想情况下，其定义在一个包含文件中，以便可被应用程序的所有文件共享。

```
#define ON_WZD_MESSAGE() \
{WM_WZD_MESSAGE, 0, 0, 0, AfxSig_vW, (AFX_PMSG)(AFX_PMSGW)(void \
(AFX_MSG_CALL CWnd::*)(CWnd *))&OnWzdMessage},
```

AfxSig_vW的值告诉MFC新消息带一个指向CWnd类的指针，且返回值是Void类型。然后，把正常情况下的 wParam和lParam参数转换为调用参数。 AfxSig_vW的其他可用值，请参阅MFC子目录下AFXMSG_.H文件。改变AfxSig_vW的值意味着改变这个宏定义的函数的调用参数。

2) 使用文本编辑器添加新消息处理函数的定义到接收消息类中。使用以下语法并保证是在ClassWizard的{{ }}框架之后，以防止ClassWizard删去所做的工作。

```
//{{AFX_MSG(CMainFrame)
//}}AFX_MSG
afx_msg void OnMyMessage(CWnd *pWnd);<<<<
DECLARE_MESSAGE_MAP()
```

3) 在实现文件中，加入新消息映像宏到消息映像中。

```
BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
```

```
    //{{AFX_MSG_MAP(CMainFrame)
    //}}AFX_MSG_MAP
    ON_WZD_MESSAGE(WM_WZD_MESSAGE,OnMyMessage)<<<<<
END_MESSAGE_MAP()
```

4) 最后，加入新消息处理函数。

```
void CMainFrame::OnWzdMessage(CWnd *pWnd)
{
//process
}
```

记住，这里传递的参数不是通常的wParam和lParam，而是更友好的CWnd指针。

说明

- 窗口消息的主要目的是窗口之间的内部消息处理，而不是处理诸如来自菜单的命令消息。添加一个新的命令消息，则要定义一个新的命令ID，而不是使用WM_COMMAND。
- 添加任何内容到映像中时，应避免把代码放在{{ }}框架之间。否则ClassWizard会把它删去。

```
//{{AFX_MSG_MAP(CMainFrame)
                                <---- NOT HERE!!!
//}}AFX_MSG_MAP
ON_WZD_MESSAGE()                <---- HERE!!!!
```

- 关于消息的详细内容参见第3章。例8有一个关于消息处理的实际例子。

CD说明

在CD上执行该工程时，单击 Options/Wzd菜单命令，视图类发送一个客户窗口消息(WM_WZD_MESSAGE)到主框架类。