MFC 对话框程序键盘消息响应与快捷键的实现

魏　欣　吴　涛　（装备指挥技术学院试验装备系 101416）
张银发　（装备指挥技术学院测控中心 101416）

摘要：在 VC 中使用 MFC 完成的对话框程序，在键盘消息响应和快捷键的实现上，没有提供直接的实现方式。通过分析 MFC 对对话框程序消息处理过程，使用重载虚函数的方法进行消息预处理，实现了对话框程序的键盘消息响应和快捷键功能。

关键词：MFC　对话框　快捷键

1　引言

在各种单文档/多文档程序中，许多菜单项，都有对应的快捷键。使用快捷键，可以省去操作者在各个菜单项中的烦琐搜寻工作。例如在各种编辑环境中使用 Ctrl + C、Ctrl + V 和 Ctrl + X 分别对应“编辑”菜单中的子菜单“复制”、“粘贴”和“剪切”，就极大地提高了用户的操作速度。

在一些工控机和商用计算机如自动取款机上，通常的操作通常需要使用面板上提供的按钮，来达到选择特定服务的目的，而在使用 Windows 操作系统的主机上，这些程序基本上都是以对话框的形式出现的。但遗憾的是，VC 中使用的 MFC 程序结构框架却没有直接在对话框程序中提供快捷键功能。

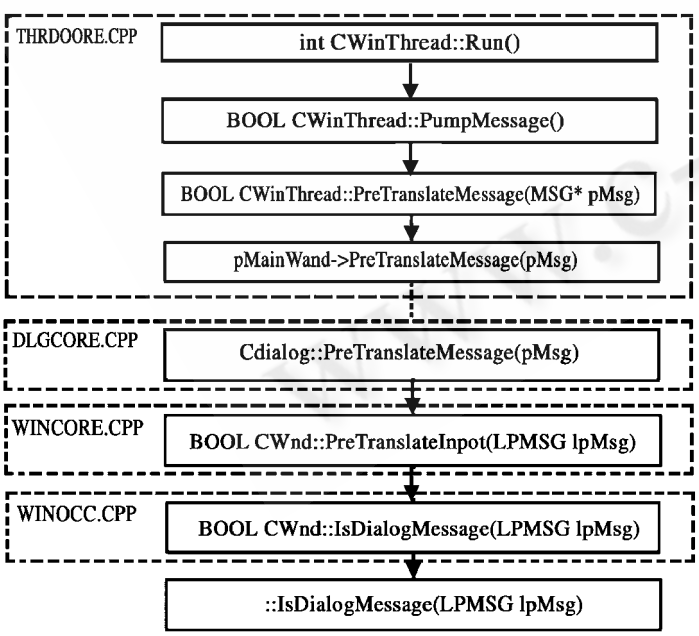

图 1　对话框程序的消息响应过程

通过对 MFC 对话框程序的消息的处理，程序员可以自己在对话框程序中实现快捷键功能。

2　对话框程序消息处理过程与键盘消息响应的实现

对于快捷键的实现，一种简单思路是在对话框程序中，添加 WM_KEYDOWN、WM_KEYUP 等消息的消息响应函数，并提取键值进行判断，然后依据键值来进行不同的程序设计，从而实现类似快捷键的消息响应模式。但实践表明，如果不做特殊处理，只是简单地添加消息响应函数，则这些消息响应函数根本得不到响应。分析 VC 提供的 MFC 源程序，就可以知道，这样的结果，与 MFC 对消息的处理过程有关。图 1 所示为 MFC 对对话框程序的消息基本处理过程。

首先，对话框程序在完成程序的初始化后，就在程序主线程中，调用 CWinThread::Run 函数。在该函数中，首先调用 API 函数 PeekMessage，PeekMessage 函数检查线程消息队列，如果消息存在，就将该消息放于指定的 MSG 结构中，以后的消息处理都将针对这一 MSG 结构对象。捕获消息后，该函数将捕获的消息进行预处理，然后再将该消息传递给相应的窗口处理函数。

键盘消息被拦截而得不到正常响应，其中的

关键就是 run 函数对消息的预处理。在 run 函数中,调用了函数 CWinThread::PumpMessage,就是利用这一函数,MFC 实现了对消息的分流,使得消息沿着 MFC 对各种消息规定的路线流动,直到被正确响应。

函数 PumpMessage 调用了函数 CwinThread::PreTranslateMessage 对消息进行处理,如果该函数不对消息进行处理,则调用 API 函数 TranslateMessage 函数将虚拟键消息转换为字符消息并调用 DispatchMessage 分发消息给窗口处理程序。在对话框程序中,程序使用 CwinThread::PreTranslateMessage 处理了键盘按下和弹起的消息,所以对话框程序将是否响应 WM_KEYDOWN、WM_KEYUP 等消息的处理函数将完全由 CwinThread::PreTranslateMessage 决定。

函数 CwinThread::PreTranslateMessage 对消息的处理,最终调用了函数 pMainWnd->PreTranslateMessage,其中 pMainWnd 等于 m_pMainWnd,是一个指向主窗口的指针,在对话框程序中,它指向的就是主对话框,所以实际调用是主窗口的消息预处理函数。

在 CWnd 及其派生类的成员函数 PreTranslateMessage 是一个虚函数,可以通过重载来改变其处理过程。在默认情况下,没有重载这一函数,所以 pMainWnd->PreTranslateMessage 直接调用了 CDialog::PreTranslateMessage,该函数调用 CWnd::PreTranslateMessage 处理 tooltip 消息,并处理了在编辑框中时 ESC 键的按下等消息。而我们需要关注的是该函数最后调用的 CWnd::PreTranslateInput。

函数 CWnd::PreTranslateInput(LPMSG lpMsg) 在做真正的消息处理之前,使用:

```
if ( (lpMsg->message < WM_KEYFIRST || lpMsg->message > WM_KEYLAST) &&
     (lpMsg->message < WM_MOUSEFIRST || lpMsg->message > WM_MOUSELAST))
    return FALSE;
```

使得该函数最后处理的消息都是键盘或鼠标消息。最后调用了函数 CWnd::IsDialogMessage。

在 CWnd::IsDialogMessage 中,只对消息做了一个简单的处理,就是直接调用 Windows API 函数 IsDialogMessage。下面是一些关于该 API 函数的说明。

API 函数 IsDialogMessage(LPMSG lpMsg) 的? 函数功能是判断一个消息是否属于给指定的对话框,如果是,则处理该消息。

函数原型:BOOL IsDialogMessage(HWND hDlg, LPMSG lpMsg);?

参数:hDlg:标识对话框。

lpMsg:指向一个含有将被检测的消息的 MSG 结构。

返回值:如果消息被处理,则返回值为非零值;如果消息没有被处理,则返回值为零。

当 IsDialogMessage 处理一个消息时,它检测键盘信息并把它们转变成对响应对话框的选择命令。例如当按下 tab 时选择下一个控件或控件组,当按下 down 时选择控件组的下一个控件。

因为 IsDialogMessage 函数要执行消息所有必要的转变和传送,即在该函数完成的同时,消息已经被处理,所以在 MFC 结构中,经 IsDialogMessage 函数处理的消息不会传送给 TranslateMessage 或 DispatchMessage 函数处理,也就是说,在函数执行完毕之后,消息的响应处理就完成了。

由以上说明可以看出,IsDialogMessage 使用默认的方式处理了对话框程序中所有的键盘消息,所以由程序员定义的键盘消息处理函数就根本没有机会得到响应。

要想达到预定的目的,就要对程序进行一些处理,使得键盘消息在函数 IsDialogMessage 被调用之前得到响应,这可以通过重载虚函数 PreTranslateMessage 来实现。以键盘按下的消息为例,首先建立一个名为 my 的对话框 MFC 工程,然后添加 WM_KEYDOWM 的消息响应函数 OnKeyDown,最后重载虚函数 PreTranslateMessage,其处理过程如下。

```
BOOL CMyDlg::PreTranslateMessage(MSG* pMsg)
{
  if(pMsg->message == WM_KEYDOWN)
  {
    OnKeyDown(pMsg->wParam, LOWORD(pMsg->lParam), HIWORD(pMsg->lParam));
    return TRUE;
  }
  return CDialog::PreTranslateMessage(pMsg);
}
```

然后在 OnKeyDown 通过不同键值进行不同的程

序设计,就可以实现类似快捷键的功能。

ID	Key	Type
IDC_TEST	VK_F2	VIRTKEY
ID_MY_F3	VK_F3	VIRTKEY

图 2　快捷键表

3　对话框程序中快捷键的实现

上面用键盘消息响应的方式,实现了对话框程序中类似快捷键的功能。其实程序员可以在对话框程序中实现真正的快捷键功能。并且使用上述方式时,需要手工处理比较多的信息,而且对 Ctrl + C 等组合键的响应,更是需要更为复杂的处理才能实现,所以实现真正的快捷键功能还是非常有必要的。下面是 CFrameWnd::PreTranslateMessage 中的一段源程序:

```
if (pMsg->message >= WM_KEYFIRST && pMsg->message <= WM_KEYLAST)
{
  HACCEL hAccel = GetDefaultAccelerator();
  return hAccel != NULL && ::TranslateAccelerator(m_hWnd, hAccel, pMsg);
}
```

正是这段源程序,实现了单文档程序的快捷键,在多文档程序中也有类似的代码。在这段源程序中,首先获取快捷键表的句柄,然后调用了 API 函数 TranslateAccelerator 来处理快捷键。该函数根据 pMsg 查询快捷键表,如果在给定的快捷键表中有消息对应键的入口,则将该消息翻译成一个对应 ID 的 WM_COMMAND 或其它消息,然后将这一消息直接送到相应的窗口处理过程。??? 函数? TranslateAccelerator 在窗口过程处理完消息后返回。

在对话框程序中,要实现快捷键,也需要作类似的处理,即获取快捷键表,并调用 TranslateAccelerator,然后,就可以做相应的处理了。下面以项目 my 为例,完成有对应控件的 F2 和没有对应控件的 F3 的快捷键,来说明快捷键的添加使用过程。

(1) 添加控件。首先,在对话框上添加 Edit 控件:ID 为 IDC_TEST_INFO,对应成员变量为 m_sTestInfo;添加按钮控件,ID 为 IDC_TEST,caption 为"测试",其响应函数为 OnTest。

(2) 添加快捷键表。项目中插入一个快捷键表,其 ID 为 IDR_MY_ACCEL,其内容如图 2 所示。其中,ID_MY_F3 为一个独立定义的 ID,没有对应的控件。

(3) 进行消息预处理。在类 CMyDlg 中重载虚函数 PreTranslateMessage,获取快捷键句柄,拦截键盘消息,查询快捷键表,完成消息转换。函数内容如下:

```
BOOL CMyDlg::PreTranslateMessage(MSG* pMsg)
{
  if((pMsg->message >= WM_KEYFIRST) && (pMsg->message <= WM_KEYLAST))
  {
    HACCEL hAccel = ::LoadAccelerators(AfxGetResourceHandle(),
                    MAKEINTRESOURCE(IDR_MY_ACCEL));
    if (hAccel && ::TranslateAccelerator(m_hWnd, hAccel, pMsg))
      return TRUE;
  }
  return CDialog::PreTranslateMessage(pMsg);
}
```

(4) 添加消息处理函数,F2 的消息响应函数就是 OnTest,F3 的消息响应函数则必须手工添加,添加过程为:

① 在 myDlg.h 中添加成员函数声明"afx_msg void OnMyF3();"于 DECLARE_MESSAGE_MAP() 之前;

② 在 myDlg.cpp 中添加 ON_COMMAND(ID_MY_F3, OnMyF3) 于 BEGIN_MESSAGE_MAP(CMyDlg, CDialog) 之后;

③ 在 myDlg.cpp 中添加 OnMyF3 函数的定义。

两个消息响应函数如下:

```
void CMyDlg::OnTest()
{
  m_sTestInfo = "单击测试按钮或按下键盘F2";
  UpdateData(FALSE);
}
void CMyDlg::OnMyF3()
{
```

(下转第 93 页)

(上接第 84 页)

```
    m_sTestInfo = "按下键盘 F3";
    UpdateData(FALSE);
}
```

这样两种快捷键的处理过程就完成了。编译执行,事实证明快捷键已经生效。当然,使用其他键或其他组合键替代 F2 或 F3 也会有相同的效果。

使用快捷键,还可以更加有效的处理对话框程序的 ESC 键空格键或 Enter 键按下时的对程序的默认处理,例如修改前面完成的快捷键表中某一项的键值为 VK_ESCAPE、VK_SPACE 或 VK_RETURN,则按下 ESC 键、空格键或 Enter 键将会有类似的反应。

需要注意的是,如果要使用 F1 作为快捷键,则需要去除在主线程中的默认消息处理过程,方法为将 my. cpp 中 ON_COMMAND(ID_HELP, CWinApp::OnHelp)这行代码注释掉就可以了。

4 结束语

本文从实际应用的角度出发,分析了 MFC 对话框程序的消息基本处理过程,通过重载虚函数 PreTranslateMessage 的方法,实现了 MFC 对话框程序的键盘消息响应和快捷键。

参考文献

1 侯俊杰, 深入浅出 MFC(第 2 版),武汉华中科技大学出版社,2001 -01。

2 朱友芹等,新编 Windows API 参考大全,电子工业出版社,2000 -03。