链滴

Win32 汇编学习 (10)：对话框 (1)

作者：Akkuman
原文链接：https://ld246.com/article/1518513308665
来源网站：链滴

现在我们开始学习一些有关GUI编程的有趣的部分：以对话框为主要界面的应用程序。

<!--more-->

理论：

如果您仔细关注过前一个程序就会发现：您无法按TAB键从一个子窗口控件跳到另一个子窗口控件，想转移的话只有 用鼠标一下一下地去点击。对用户来说这是不友好的。另一件事是如果您象前一课中样把主窗口的背景色从白色改成 灰色，为了子窗口控件无缝地作相应地改变，您必须仔细分类所有子口。 造成上述诸多不便的原因是子窗口控件本来是为对话框而设计的，像子窗口控件的背景色是灰色，而对话框的背景色也是 灰色的，这样它们本来就相互协调了，而无须程序员加入其他的处理。 在们深入讨论对话框前我们必须知道何谓对话框。

何谓对话框

一个对话框其实是有很多的子窗口控件的一个窗口，WINDOWS在对话框 内部有一个管理程序，由来处理象按下TAB键则输入焦点从一个子窗口空间条到另一个子窗口控件、按下ENTER键等于在当 前有输入焦点的子窗口控件上点击了鼠标 等等这些杂事，这样程序员就可以集中精力于他们的逻辑事务。对话框主要用 作输入输出接口，人们无须知道它们内部的工作原理，而只要知道如何和他们进行交就可以了。这也是面向对象设计中的 所谓信息隐藏。只要这个黑盒子中的实现足够完美，我们就可以心地使用，当然我们必须强调的是“黑盒子”必须完美。 WIN32 API 内部 的实现即是一个“黑盒子。 现在让我们回到正题来，对话框的设计是为了减少程序员的工作量的，一般您如果在窗口中 自己一个子窗口控件您就必须自己处理其中的按键逻辑和细分类它的窗口过程。如果您把它放到对话框中则这些杂事 对话框会自己处理，您只要知道如何获得用户输入的数据和如何把数据放入到子窗口控件去就可以了。 在程序中对话框和菜单一样被定义成一种资源，您可以在脚本文件中写一个对话框模板其中包含该对话框和子窗口的特性， 然后用资源编辑器编辑。需要注意的是所有的资源必须放在同一脚本文件中。 虽然可以用文本编辑器去编辑脚本文件，但是象要调整子窗口控件位置时要涉及到一些标值时最好还是用一些可视化的编 辑器，这样方便多了。一般在编译器的开发包中都会带资源编辑器您可以用它们来产生一个模板然后增删一些子窗口控件。

对话框的分类

有两种主要的对话框：模式对话框和无模式对话框。无模式对话框允许您把输入焦点切换到（同一个用程序的）另一个窗口，而该对话框无须关闭 。比如WORD 中的FIND对话框。模式对话框又有两类应用程序模式对话框和系统对话框。应用程序对话框不允许您在本 应用程序中切换输入焦点，但是可切换到其它的应用程序中去，而系统对话框则必须您对该对话框做出响应否则不能切换到 任何的应用序中去。要创建一个无模式对话框调用API函数CreateDialogParam，而创建一个模式对话框则调用AI函数DialogBoxParam。 其中**应用程序模式对话框和系统模式对话框之间的差别是style参数不同**，想创建一个系统模式对话框该参数必须“or”上 DS_SYSMODAL标志位。在对话框中若要和子窗口件通讯则调用函数SendDlgItemMesage。该函数的语法如下：

```
SendDlgItemMessage proto hwndDlg:DWORD,\
                         idControl:DWORD,\
                         uMsg:DWORD,\
                         wParam:DWORD,\
                         lParam:DWORD
```

该API函数对于用在向子窗口控件发送消息方面是非常有用的。譬如:如果您想得到edit控件中的字符可以这么做:

```
call SendDlgItemMessage, hDlg, ID_EDITBOX, WM_GETTEXT, 256, ADDR text_buffer
```

具体要发送那些消息应当查询有关的WIN32 API 参考手册。 WINDOWS 还 提供几个快速存取控件据的函数。譬如：GetDlgItemText、CheckDlgButton等。这样一来，您就可以不用去查询每个消息的Param和lParam参数获得相关信息了。您应尽可能地使用这些API 函数，这样使得您的代码将来比较易维护。**对话框的管理函数会把一些消息发送给一个特定的回调函数：对话框过程处理函数**，该函数格式为：

```
DlgProc  proto hDlg:DWORD ,\
               iMsg:DWORD ,\
               wParam:DWORD ,\
               lParam:DWORD
```

该函数的格式非常类似于窗口的过程函数，除了返回值是TURE和FALSE，而不是HRESULT，**存在于WNDOWS内部的对话框管理器才是对话框真正的窗口过程函数。它会把某些消息传递给我们的窗口过函数。**所以当我们的窗口过程函数处理这些消息时就返回TRUE，否则就在eax中返回FALSE。**这也意着我们的窗口过程函数在接受到自己不处理的消息时并不会调用DefWindowProc函数，因为它本身是一个真正的窗口过程函数。**对于对话框有两种用法：一种是把它作为一个主窗口来用，一种是把它为一种输入输出设备使用。这次我们将示范第一种用法。“把对话框用作主窗口”有两种意思：

1. 您可以调用 RegisterClassEx函数把对话框模板注册为一个窗口类。这样该对话框的行为就类似一个普通的窗口了：它通过在注册窗口时指定的窗口过程来处理所有的消息，通过这种方法来使用对框的好处是您不需要显示地创建子窗口控件，WINDOWS本身会帮您创建好，另外还会帮您处理所有按键逻辑，另外您还可以指定您窗口类结构中的光标和图标；

2. 您的应用程序创建没有父窗口的对话框窗口，这种方法中， **没有必要需要一段处理消息循环的代，因为所有的消息被直接送到对话框过程处理函数**，这样您也可以不要注册一个窗口类。我们将先使第一种方法然后使用第二种方法。

例子：

```
;-------------------------------------------------------------------------------

;dialog.asm

;-------------------------------------------------------------------------------

.386
.model flat,stdcall
option casemap:none

WinMain proto :DWORD,:DWORD,:DWORD,:DWORD

include windows.inc
include user32.inc
include kernel32.inc
includelib user32.lib
includelib kernel32.lib

.data
ClassName db "DLGCLASS",0
MenuName db "MyMenu",0
DlgName db "MyDialog",0
AppName db "Our First Dialog Box",0
TestString db "Wow! I'm in an edit box now",0
```

```
.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
buffer db 512 dup(?)

.const
IDC_EDIT    equ 3000
IDC_BUTTON  equ 3001
IDC_EXIT    equ 3002
IDM_GETTEXT equ 32000
IDM_CLEAR   equ 32001
IDM_EXIT    equ 32002

.code
start:
invoke GetModuleHandle,NULL
mov hInstance,eax
invoke GetCommandLine
mov CommandLine,eax
invoke WinMain,hInstance,NULL,CommandLine,SW_SHOWDEFAULT
invoke ExitProcess,eax

WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD

    LOCAL wc:WNDCLASSEX
    LOCAL msg:MSG
    LOCAL hDlg:HWND
    mov wc.cbSize,SIZEOF WNDCLASSEX
    mov wc.style,CS_HREDRAW or CS_VREDRAW
    mov wc.lpfnWndProc,offset WndProc
    mov wc.cbClsExtra,NULL
    mov wc.cbWndExtra,DLGWINDOWEXTRA
    push hInst
    pop wc.hInstance
    mov wc.hbrBackground,COLOR_BTNFACE+1
    mov wc.lpszMenuName,offset MenuName
    mov wc.lpszClassName,offset ClassName
    invoke LoadIcon,NULL,IDI_APPLICATION
    mov wc.hIcon,eax
    mov wc.hIconSm,eax
    invoke LoadCursor,NULL,IDC_ARROW
    mov wc.hCursor,eax
    invoke RegisterClassEx,addr wc
    invoke CreateDialogParam,hInstance,addr DlgName,NULL,NULL,NULL
    mov hDlg,eax
    invoke ShowWindow,hDlg,SW_SHOWNORMAL
    invoke UpdateWindow,hDlg
    invoke GetDlgItem,hDlg,IDC_EDIT
    invoke SetFocus,eax
    .while TRUE
        invoke GetMessage,addr msg,NULL,0,0
        .break .if (!eax)
        invoke IsDialogMessage,hDlg,addr msg
        .if eax==FALSE
```

```
            invoke TranslateMessage,addr msg
            invoke DispatchMessage,addr msg
        .endif
    .endw
    mov eax,msg.wParam
    ret

WinMain endp

WndProc proc hWnd:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM

    .if uMsg==WM_DESTROY
        invoke PostQuitMessage,NULL
    .elseif uMsg==WM_COMMAND
        mov eax,wParam
        .if lParam==0
            .if ax==IDM_GETTEXT
                invoke GetDlgItemText,hWnd,IDC_EDIT,addr buffer,512
                invoke MessageBox,NULL,addr buffer,addr AppName,MB_OK
            .elseif ax==IDM_CLEAR
                invoke SetDlgItemText,hWnd,IDC_EDIT,NULL
            .else
                invoke DestroyWindow,hWnd
            .endif
        .else
            mov edx,wParam
            shr edx,16
            .if dx==BN_CLICKED
                .if ax==IDC_BUTTON
                    invoke SetDlgItemText,hWnd,IDC_EDIT,addr TestString
                .elseif ax==IDC_EXIT
                    invoke SendMessage,hWnd,WM_COMMAND,IDM_EXIT,0
                .endif
            .endif
        .endif
    .else
        invoke DefWindowProc,hWnd,uMsg,wParam,lParam
        ret
    .endif
    xor eax,eax
    ret

WndProc endp
end start

;-----------------------------------------------------------------------------

;Dialog.rc

;-----------------------------------------------------------------------------

#include "c:\masm32\include\RESOURCE.h"

#define IDC_EDIT    3000
```

```
#define IDC_BUTTON  3001
#define IDC_EXIT    3002

#define IDM_GETTEXT 32000
#define IDM_CLEAR   32001
#define IDM_EXIT    32003

MyDialog DIALOG 10,10,205,60
STYLE 0x0004 | DS_CENTER | WS_CAPTION | WS_MINIMIZEBOX | WS_SYSMENU | WS_VISIBLE |
WS_OVERLAPPED | DS_MODALFRAME | DS_3DLOOK
CAPTION "Our First Dialog Box"
CLASS "DLGCLASS"
{
   EDITTEXT       IDC_EDIT,   15,17,111,13, ES_AUTOHSCROLL | ES_LEFT
   DEFPUSHBUTTON   "Say Hello", IDC_BUTTON,    141,10,52,13
   PUSHBUTTON      "E&xit", IDC_EXIT,  141,26,52,13, WS_GROUP
}

MyMenu  MENU
{
   POPUP "Test Controls"
   {
      MENUITEM "Get Text", IDM_GETTEXT
      MENUITEM "Clear Text", IDM_CLEAR
      MENUITEM "", , 0x0800 /*MFT_SEPARATOR*/
      MENUITEM "E&xit", IDM_EXIT
   }
}
```

分析：

我们先来分析第一个例子：

该例显示了如何把一个对话框模板注册成一个窗口类，然后创建一个由该窗口类派生的窗口。由于您有必要自己去创建子窗口控件，所以就简化了许多的工作。

我们先来分析对话框模板。

```
MyDialog DIALOG 10, 10, 205, 60
```

先是对话框的名字，然后是关键字DAILOG。接下来的四个数字中，前两个是对话框的坐标，后两个对话框的宽和高（注意：它们的单位是对话框的单位，而不一定是像素点）。

```
STYLE 0x0004 | DS_CENTER | WS_CAPTION | WS_MINIMIZEBOX | WS_SYSMENU | WS_VISIBLE |
WS_OVERLAPPED | DS_MODALFRAME | DS_3DLOOK
```

上面定义了对话框的风格。

```
CAPTION "Our First Dialog Box"
```

这是显示在对话框标题条上的标题。

```
CLASS "DLGCLASS"
```

这一行非常关键。正是有了关键字CLASS，我们才可以用它来声明把一个对话框当成一个窗口来用。

在关键字后面的是“窗口类”的名称。

```
{
   EDITTEXT        IDC_EDIT,   15,17,111,13, ES_AUTOHSCROLL | ES_LEFT
   DEFPUSHBUTTON   "Say Hello", IDC_BUTTON,    141,10,52,13
   PUSHBUTTON      "E&xit", IDC_EXIT,  141,26,52,13
}
```

上面的一块定义了对话框中的子窗口控件，它们是声明在{}之间的。

```
control-type  "text"   ,controlID, x, y, width, height [,styles]
```

控件的类型是资源编辑器定义好了的常数，您可以查找有关的手册。

现在我们来看看汇编源代码。先看这部分：

```
mov   wc.cbWndExtra,DLGWINDOWEXTRA
mov   wc.lpszClassName,OFFSET ClassName
```

通常cbWndExtra被设成NULL，但我们想把一个对话框模板注册成一个窗口类，我们必须把该成员的设成DLGWINDOWEXTRA。**注意类的名称必须和模板中跟在CLASS关键字后面的名称一样。**余下的员变量和声明一般的窗口类相同。填写好窗口类结构变量后调用函数RegisterClassEx进行注册。看去这一切和注册一个普通的窗口类是一样的。

```
invoke CreateDialogParam,hInstance,ADDR DlgName,NULL,NULL,NULL
```

注册完毕后，我们就创建该对话框。在这个例子中，我们调用函数CreateDialogParam产生一个无模对话框。这个函数共有5个参数，其中前两个参数是必须的：实例句柄和指向对话框模板名称的指针注意第二个参数是指向模板名称而不是类名称的指针。这时，WINDOWS将产生对话框和子控件窗口。**时您的应用程序将接收到由WINDOWS传送的第一个消息WM_CREATE。**

```
invoke GetDlgItem,hDlg,IDC_EDIT
invoke SetFocus,eax
```

在对话框产生后，我们把输入输出焦点设到编辑控件上。**如果在WM_CREATE消息处理段中假如设焦点的代码，GetDlgItem函数就会失败，因为此时空间窗口还未产生，为了在对话框和所有的子窗控件都产生后调用该函数我们把它安排到了函数UpdatWindow后，**GetDlgItem函数返回该控件的口句柄。

```
      invoke IsDialogMessage, hDlg, ADDR msg
       .IF eax ==FALSE
          invoke TranslateMessage, ADDR msg
          invoke DispatchMessage, ADDR msg
       .ENDIF
```

现在程序进入消息循环，在我们翻译和派发消息前，该函数使得对话框内置的对话框管理程序来处理关的键盘跳转逻辑。如果该函数返回TRUE，则表示消息是传给对话框的已经由该函数处理了。注意上篇文章中不同，当我们想得到控件的文本信息时调用GetDlgItemText函数而不是GetWindowText数，前者接受的参数是一个控件的ID 号，而不是窗口的句柄，这使得在对话框中调用该函数更方便。

好我们现在使用第二种方法把一个对话框当成一个主窗口来使用。在接下来的例子中，我们将产生一应用程序的模式对话框，您将会发现其中根本没有消息循环或窗口处理过程，因为它们根本没有必要！

```
;-------------------------------------------------------------------------
```

```
;dialog.asm (part 2)

;-------------------------------------------------------------------------

.386
.model flat,stdcall
option casemap:none

DlgProc proto :DWORD,:DWORD,:DWORD,:DWORD

include windows.inc
include user32.inc
include kernel32.inc
includelib user32.lib
includelib kernel32.lib

.data
DlgName db "MyDialog",0
AppName db "Our Second Dialog Box",0
TestString db "Wow! I'm in an edit box now",0

.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
buffer db 512 dup(?)

.const
IDC_EDIT    equ 3000
IDC_BUTTON  equ 3001
IDC_EXIT    equ 3002
IDM_GETTEXT equ 32000
IDM_CLEAR   equ 32001
IDM_EXIT    equ 32002

.code
start:
invoke GetModuleHandle,NULL
mov hInstance,eax
invoke DialogBoxParam,hInstance,addr DlgName,NULL, addr DlgProc,NULL
invoke ExitProcess,eax

DlgProc proc hWnd:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM

    .if uMsg==WM_INITDIALOG
        invoke GetDlgItem,hWnd,IDC_EDIT
        invoke SetFocus,eax
    .elseif uMsg==WM_CLOSE
        invoke SendMessage,hWnd,WM_COMMAND,IDM_EXIT,0
    .elseif uMsg==WM_COMMAND
        mov eax,wParam
        .if lParam==0
            .if ax==IDM_GETTEXT
                invoke GetDlgItemText,hWnd,IDC_EDIT,addr buffer,512
```

```
            invoke MessageBox,NULL,addr buffer,addr AppName,MB_OK
        .elseif ax==IDM_CLEAR
            invoke SetDlgItemText,hWnd,IDC_EDIT,NULL
        .elseif ax==IDM_EXIT
            invoke EndDialog,hWnd,NULL
        .endif
    .else
        mov edx,wParam
        shr edx,16
        .if dx==BN_CLICKED
            .if ax==IDC_BUTTON
                invoke SetDlgItemText,hWnd,IDC_EDIT,addr TestString
            .elseif ax==IDC_EXIT
                invoke SendMessage,hWnd,WM_COMMAND,IDM_EXIT,0
            .endif
        .endif
    .endif
.else
    mov eax,FALSE
    ret
.endif
mov eax,TRUE
ret

DlgProc endp
end start

;-----------------------------------------------------------------------------

;dialog.rc (part 2)

;-----------------------------------------------------------------------------

#include "c:\masm32\include\RESOURCE.h"

#define IDC_EDIT    3000
#define IDC_BUTTON  3001
#define IDC_EXIT    3002

#define IDM_GETTEXT 32000
#define IDM_CLEAR   32001
#define IDM_EXIT    32003

MyDialog DIALOG 10,10,205,60
STYLE 0x0004 | DS_CENTER | WS_CAPTION | WS_MINIMIZEBOX | WS_SYSMENU | WS_VISIBLE |
WS_OVERLAPPED | DS_MODALFRAME | DS_3DLOOK
CAPTION "Our Second Dialog Box"
MENU IDR_MENU1
{
    EDITTEXT        IDC_EDIT,    15,17,111,13, ES_AUTOHSCROLL | ES_LEFT
    DEFPUSHBUTTON   "Say Hello", IDC_BUTTON,     141,10,52,13
    PUSHBUTTON      "E&xit", IDC_EXIT,  141,26,52,13, WS_GROUP
}
```

```
IDR_MENU1 MENU
{
   POPUP "Test Controls"
   {
      MENUITEM "Get Text", IDM_GETTEXT
      MENUITEM "Clear Text", IDM_CLEAR
      MENUITEM "", , 0x0800 /*MFT_SEPARATOR*/
      MENUITEM "E&xit", IDM_EXIT
   }
}
```

分析如下：

```
   DlgProc proto :DWORD,:DWORD,:DWORD,:DWORD
```

我们已经定义了DlgProc函数的原型，所以可以用操作符ADDR来获得它的地址（ADDR可以在运行动态地获得标识符的有效地址）：

```
   invoke DialogBoxParam, hInstance, ADDR DlgName,NULL, addr DlgProc, NULL
```

上面的几行调用了函数DialogBoxParam，该函数有五个参数，分别是：实例句柄、对话框模板的名、父窗口的句柄、对话框过程函数的地址、和对话框相关的数据。该函数产生一个模式对话框。如果显式地关闭该函数不会返回。

```
   .IF uMsg==WM_INITDIALOG
      invoke GetDlgItem, hWnd,IDC_EDIT
      invoke SetFocus,eax
   .ELSEIF uMsg==WM_CLOSE
      invoke SendMessage,hWnd,WM_COMMAND,IDM_EXIT,0
```

除了不处理WM_CREATE消息外对话框的窗口处理过程函数和一般的窗口处理过程相似。该过程函数收到的第一个消息是WM_INITDIALOG。通常把初始化的代码放到此处。注意如果您处理该消息必须在ax中返回TRUE。内置的对话框管理函数不会把WM_DESTROY 消息发送到对话框的消息处理函数，以如果我们想在对话框关闭时进行处理，就把它放到WM_CLOSE消息的处理中。在我们的例子中我发送消息WM_COMMAND，并在参数wParam中放置IDM_EXIT，这和处理WM_CLOSE 消息效果样，在处理IDM_EXIT 中我们调用EndDialog函数。如果我们想要销毁一个对话框，必须调用EndDialg函数，该函数并不会立即销毁一个窗口，而是设置一个标志位，然后对话框管理器会处理接下去的毁对话框动作。好，现在我们来看看资源文件，其中最显著的变化是**在指定菜单时我们不是用字符串定该菜单的名称而是用了一个常量 IDR_MENU1。在调用DialogBoxParam产生的对话框中挂接一菜单必须这么做**，注意在该对话框模板中，在该标识符前必须加MENU关键字，这两个例子中的显著同是后者没有图标，这可以在处理WM_INITDIALOG中发送消息WM_SETICON消息，然后在该消息理代码中作适当的处理即可。