Win32 汇编学习 (9)：窗口控件

作者：Akkuman
原文链接：https://ld246.com/article/1518447052739
来源网站：链滴

这次我们将探讨控件，这些控件是我们程序主要的输入输出设备。

<!--more-->

理论：

WINDOWS 提供了几个预定义的窗口类以方便我们的使用。大多数时间内，我们把它们用在对话框，所以我们一般就它们叫做子窗口控件。子窗口控件会自己处理消息，并在自己状态发生改变时通知窗口。这样就大大地减轻了我们的编程工作，所以我们应尽可能地利用它们。本课中我们把这些控件在窗口中以简化程序，但是大多数时间内子窗口控件都是放在对话框中的。我们示例中演示的子窗口件包括：按钮、下拉菜单、检查框、单选按钮、编辑框等。使用子窗口控件时，先调用CreateWindow或 CreateWindowEx。在这里由于WINDOWS 已经注册了这些子控件，所以无须我们再注册。当然们不能改变它们的类名称。譬如：如果您想产生一个按钮，在调用上述两个函数时就必须指定类名为"utton"。其他必须指定的参数还有父窗口的句柄和将要产生的子控件的ID号。子控件的ID号是用来标子控件的，故也必须是唯一 的。**子控件产生后，当其状态改变时将会向父窗口发送消息。**一般我们应父窗口的WM_CREATE消息中产生子控件。**子控件向父窗口发送的消息是WM_COMMAND，并在递的参数wPara的低位中包括控件的ID号，消息号在wParam的高位，lParam中则包括了子控件的口的句柄。**各类控件有不同的消息代码集，详情请参见WIN32 API参考手册。**父窗口也可以通过调用数SendMessage向子控件发送消息，其中第一个参数是子控件的窗口句柄，第二个参数是要发送的息号，附加的参数可以在wParam和lParam中传递**，其实只要知道了某个窗口的句柄就可以用该函向其发送相关消息。所以产生了子窗口后必须处理WM_COMMAND消息以便可以接收到子控件的消。

例子：

我们将写一个窗口，在该窗口中有一个编辑框和一个按钮。当您按下按钮时 ，会弹出一个对话框其中示了您在编辑框中输入的内容。另外，该应用程序还有一个菜单，其中有四个菜单项：

1. Say Hello -- 把一个字符串输入编辑控件；
2. Clear Edit Box -- 清除编辑控件中的字符串；
3. Get Text -- 弹出对话框显示编辑控件中的字符串；
4. Exit -- 退出应用程序。

```
.386
.model flat,stdcall
option casemap:none
WinMain proto :DWORD,:DWORD,:DWORD,:DWORD

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib

.data
ClassName db "SimpleWinClass",0
AppName  db "Our First Window",0
MenuName db "FirstMenu",0
ButtonClassName db "button",0
ButtonText db "My First Button",0
EditClassName db "edit",0
```

```
TestString db "Wow! I'm in an edit box now",0

.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
hwndButton HWND ?
hwndEdit HWND ?
buffer db 512 dup(?)                    ; buffer to store the text retrieved from the edit box

.const
ButtonID equ 1                           ; The control ID of the button control
EditID equ 2                             ; The control ID of the edit control
IDM_HELLO equ 1
IDM_CLEAR equ 2
IDM_GETTEXT equ 3
IDM_EXIT equ 4

.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 hwnd: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,NULL
    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 CreateWindowEx,WS_EX_CLIENTEDGE,ADDR ClassName, \
                    ADDR AppName, WS_OVERLAPPEDWINDOW,\
                    CW_USEDEFAULT, CW_USEDEFAULT,\
                    300,200,NULL,NULL, hInst,NULL
    mov   hwnd,eax
    invoke ShowWindow, hwnd,SW_SHOWNORMAL
    invoke UpdateWindow, hwnd
    .WHILE TRUE
```

```
        invoke GetMessage, ADDR msg,NULL,0,0
        .BREAK .IF (!eax)
        invoke TranslateMessage, ADDR msg
        invoke DispatchMessage, ADDR msg
    .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_CREATE
        invoke CreateWindowEx,WS_EX_CLIENTEDGE, ADDR EditClassName,NULL,\
                    WS_CHILD or WS_VISIBLE or WS_BORDER or ES_LEFT or\
                    ES_AUTOHSCROLL,\
                    50,35,200,25,hWnd,8,hInstance,NULL
        mov  hwndEdit,eax
        invoke SetFocus, hwndEdit
        invoke CreateWindowEx,NULL, ADDR ButtonClassName,ADDR ButtonText,\
                    WS_CHILD or WS_VISIBLE or BS_DEFPUSHBUTTON,\
                    75,70,140,25,hWnd,ButtonID,hInstance,NULL
        mov  hwndButton,eax
    .ELSEIF uMsg==WM_COMMAND
        mov eax,wParam
        .IF lParam==0
            .IF ax==IDM_HELLO
                invoke SetWindowText,hwndEdit,ADDR TestString
            .ELSEIF ax==IDM_CLEAR
                invoke SetWindowText,hwndEdit,NULL
            .ELSEIF  ax==IDM_GETTEXT
                invoke GetWindowText,hwndEdit,ADDR buffer,512
                invoke MessageBox,NULL,ADDR buffer,ADDR AppName,MB_OK
            .ELSE
                invoke DestroyWindow,hWnd
            .ENDIF
        .ELSE
            .IF ax==ButtonID
                shr eax,16
                .IF ax==BN_CLICKED
                    invoke SendMessage,hWnd,WM_COMMAND,IDM_GETTEXT,0
                .ENDIF
            .ENDIF
        .ENDIF
    .ELSE
        invoke DefWindowProc,hWnd,uMsg,wParam,lParam
        ret
    .ENDIF
     xor    eax,eax
    ret
WndProc endp
end start
```

menu.rc

```
#define IDM_HELLO 1
#define IDM_CLEAR 2
#define IDM_GETTEXT 3
#define IDM_EXIT 4

SecondMenu MENU
{
POPUP "&PopUp"
   {
   MENUITEM "&Say Hello",IDM_HELLO
   MENUITEM "&Clear", IDM_CLEAR
   MENUITEM SEPARATOR
   MENUITEM "E&xit",IDM_EXIT
   }
MENUITEM "&Text",IDM_GETTEXT
}
```

分析：

我们现在开始分析，

```
   .ELSEIF uMsg==WM_CREATE
      invoke CreateWindowEx,WS_EX_CLIENTEDGE, \
                  ADDR EditClassName,NULL,\
                  WS_CHILD or WS_VISIBLE or WS_BORDER or ES_LEFT\
                  or ES_AUTOHSCROLL,\
                  50,35,200,25,hWnd,EditID,hInstance,NULL
      mov  hwndEdit,eax
      invoke SetFocus, hwndEdit
      invoke CreateWindowEx,NULL, ADDR ButtonClassName,\
                  ADDR ButtonText,\
                  WS_CHILD or WS_VISIBLE or BS_DEFPUSHBUTTON,\
                  75,70,140,25,hWnd,ButtonID,hInstance,NULL
      mov  hwndButton,eax
```

我们在WM_CREATE中产生子控件，其中在函数CreateWindowEx中给子控件窗口一个WS_EX_CLIENEDGE风格，它使得子控件窗口看上去边界下凹，具有立体感。每一个子控件的类名都是预定义的，如：按钮的预定义类名是"button"，编辑框是"edit"。接下来的参数是窗口风格，除了通常的窗口风外，每一个控件都有自己的扩展风格，譬如：按钮类的扩展风格前面加有BS_，编辑框类则是：ES_，IN32 API 参考中有所有的扩展风格的描述。注意：您在CreateWindowsEx函数中本来要传递菜单句的地方传入子窗口空间的ID号不会有什么副作用，因为子窗口控件本身不能有菜单。产生控件后，我保存它们的句柄，然后调用SetFocus把焦点设到编辑控件上以便用户立即可以输入。接下来的是如何理控件发送的通知消息WM_COMMAND：

```
   .ELSEIF uMsg==WM_COMMAND
      mov eax,wParam
      .IF lParam==0
```

我们以前讲过选择菜单项也会发送WM_COMMAND消息，那我们应如何区分呢？看了下表您就会一了然：

```
/---------+--------------------+--------------------+--------------------\
|         | Low word of wParam |High word of wParam |       lParam       |
+---------+--------------------+--------------------+--------------------+
```

```
| Menu   |      Menu ID      |         0         |         0          |
+--------+ ------------------+-------------------+--------------------+
| Control |     Control ID    |  Notification code | Child Window Handle |
\--------+-------------------+-------------------+--------------------/
```

其中我们可以看到不能用wParam来区分，因为菜单和控件的ID号可能相同，而且子窗口空间的消息也有可能为0。

```
.IF ax==IDM_HELLO
    invoke SetWindowText,hwndEdit,ADDR TestString
.ELSEIF ax==IDM_CLEAR
    invoke SetWindowText,hwndEdit,NULL
.ELSEIF  ax==IDM_GETTEXT
    invoke GetWindowText,hwndEdit,ADDR buffer,512
    invoke MessageBox,NULL,ADDR buffer,ADDR AppName,MB_OK
```

您可以调用SetWindowText函数把一字符串复制到编辑控件中去，为了清空，传入NULL值。SetWindowText是一个通用函数，即可以用它来设定一个窗口的标题，也可以用它来改变一个按钮上的文字如果是要得到按钮上的文字，则调用GetWindowText。

```
.IF ax==ButtonID
    shr eax,16
    .IF ax==BN_CLICKED
        invoke SendMessage,hWnd,WM_COMMAND,IDM_GETTEXT,0
    .ENDIF
.ENDIF
```

上面的片段是处理用户按钮事件的。他首先检查wParam的高字节看是否是按钮的ID 号，若是则检查字节看发送的消息号是否BN_CLICKED，该消息是在按钮按下时发送的，如果一切都对，则转入处理消息，我们可以从处理消息IDM_GETTEXT处复制全部的代码，但是更专业的办法是在发送一条IDM_ETTEXT消息让主窗口过程处理，这只要把传送的消息设置为WM_COMMAND，再把wParam的低节中设置为IDM_GETTEXT即可。这样一来您的代码就简洁了许多，所以尽可能利用该技巧。最后，然不是或有或无，必须在消息循环中调用函数TranslateMessage，因为您的应用程序需要在编辑框输入可读的文字。如果省略了该函数，就不能在编辑框中输入任何东西。