Win32 汇编学习 (8)：菜单

作者：Akkuman
原文链接：https://ld246.com/article/1518442762756
来源网站：链滴

这次我们将在我们的应用程序中加入一个菜单。

<!--more-->

理论：

菜单可以说是WINDOWS最重要的元素之一。有了它，用户可以方便地选择操作命令.用户只要细读下所有的菜单项就可以明了应用程序所提供的大概功能,而且可以立即操作,无须去阅读手册了.正因为单给了用户一种方便的方式,所以您在应用程序中加入菜单时就要遵守一般的标准.譬如:一般头两个菜项是"File"和"Edit",最后是"Help",您可以在这中间插入您要定义的菜单项.如果所运行的菜单命令会弹一个对话框,那么就要在该菜单项后加入省略符(...).菜单是一种资源,除菜单外还有其它像对话框,字符串图标,位图资源等.在链接时链接程序将把资源加入到可执行程序中去,最后我们的执行程序中就既包括器指令又包括了资源. 您可以在任何文本编辑器中编写脚本文件,在文件中您可以指定资源呈现出来的观和其它的一些属性.当然更直观的方法是用资源编辑器,通常资源编辑器都打包在编译环境中,像Visual C++带了资源编辑器. 我们可以按以下方式来定义一个菜单资源：

```
MyMenu  MENU
{
  [menu list here]
}
```

这和C语言中的结构体的定义非常相似。 MyMenu类似于被定义的变量，而MENU则类似于关键字。然您可以用另外一种办法，那就是用BEGIN和END来代替花括号，这和PASCAL语言中的风格相同。

在菜单项的列表中是一大串的MENUITEM和POPUP语句。 MENUITEM定义了一个菜单项，当选择后会激活对话框。它的语法如下：

```
MENUITEM "&text", ID [,options]
```

它由关键字MENUITEM开头，紧跟在MENUITEM后的是指菜单项的名称字符串，**符号“&“后的第个字符将会带下划线**，它也是该菜单项的快捷键。ID的作用当该菜单被选中时，WINDOWS的消息处过程用来区分菜单项用的。毫无疑问，ID号必须唯一。

options有以下可供选择的属性：

- GRAYED 代表该菜单项处于非激活状态，即当其被选中时不会产生 WM_COMMAND 消息。该菜以灰色显示。
- INACTIVE 代表该菜单项处于非激活状态，即当其被选中时不会产生 WM_COMMAND 消息。该单以正常颜色显示。
- MENUBREAK 该菜单项和随后的菜单项会显示在新列中。
- HELP 该菜单项和随后的菜单项右对齐。

POPUP的语法如下：

```
POPUP "&text" [,options]
{
  [menu list]
}
```

POPUP定义了一个菜单项当该菜单项被选中时又会弹出一个子菜单。

另外有一种特别类型的MENUITEM语句 MENUITEM SEPARATOR，它表示在菜单项位置画一条分线。定义完菜单后，您就可以在程序中使用脚本中定义的菜单资源了。您可以在程序的两个地方（或

做用两种方式）使用它们：

● 在 WNDCLASSEX结构体的成员 lpszMenuName中。譬如，您有一个菜单“FirstMenu“，您以按如下方法把它联系到您的窗口：

```
.DATA
MenuName  db "FirstMenu",0
.....
.....
.CODE
.....
mov   wc.lpszMenuName, OFFSET MenuName
.....
```

● 在 CreateWindowEx函数中指明菜单的句柄：

```
.DATA
MenuName  db "FirstMenu",0
hMenu HMENU ?
.....
.....
.CODE
.....
invoke LoadMenu, hInst, OFFSET MenuName
mov   hMenu, eax
invoke CreateWindowEx,NULL,OFFSET ClsName,\
           OFFSET Caption, WS_OVERLAPPEDWINDOW,\
           CW_USEDEFAULT,CW_USEDEFAULT,\
           CW_USEDEFAULT,CW_USEDEFAULT,\
           NULL,\
          hMenu,\
           hInst,\
           NULL\
.....
```

您也许会问，这两着之间有什么不同呢？**当您用第一种方法时，由于是在窗口类中指定，故所有由窗口类派生的窗口都将有相同的菜单。如果您想要从相同的类中派生的窗口有不同的菜单那就要使用二中方法，该方法中通过函数 CreateWindowEx指定的菜单会“覆盖” WNDCLASSEX结构体中指的菜单。**接下来我们看看当用户选择了一个菜单项时它是如何通知WINDOWS 窗口过程的：当用选择了一个菜单项时，WINDOWS窗口过程会接收到一个 WM_COMMAND消息，**传进来的参数 wPram的低字节包含了菜单项的 ID号。**好了，上面就是关于菜单项的一切，下面我们就来实践。

例子：

第一个例子显示了指定一个菜单项的第一种方法：

```
.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 "SimpleWinClass",0
AppName db "Our Sixth Window",0
MenuName db "FirstMenu",0
Test_string db "你选择了测试菜单项",0
Hello_string db "你好",0
Goodbye_string db "再见",0

.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?

.const
IDM_TEST equ 1
IDM_HELLO equ 2
IDM_GOODBYE equ 3
IDM_TEXT 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_WINDOW+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,NULL,addr ClassName,addr AppName,WS_OVERLAPPEDWINDOW
CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,NULL,NULL,hInst,NUL
```

```
    mov hwnd,eax
    invoke ShowWindow,hwnd,SW_SHOWNORMAL
    invoke UpdateWindow,hwnd
    .while TRUE
        invoke GetMessage,addr msg,NULL,0,0
        .break .if (!eax)
        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_COMMAND
        mov eax,wParam
        .if ax==IDM_TEST
            invoke MessageBox,NULL,addr Test_string,offset AppName,MB_OK
        .elseif ax==IDM_HELLO
            invoke MessageBox,NULL,addr Hello_string,offset AppName,MB_OK
        .elseif ax==IDM_GOODBYE
            invoke MessageBox,NULL,addr Goodbye_string,offset AppName,MB_OK
        .else
            invoke DestroyWindow,hWnd
        .endif
    .else
        invoke DefWindowProc,hWnd,uMsg,wParam,lParam
        ret
    .endif
    xor eax,eax
    ret

WndProc endp
end start
```

Menu.rc

```
#define IDM_TEST 1
#define IDM_HELLO 2
#define IDM_GOODBYE 3
#define IDM_EXIT 4

FirstMenu MENU
{
 POPUP "&PopUp"
       {
        MENUITEM "&Say Hello",IDM_HELLO
        MENUITEM "Say &GoodBye", IDM_GOODBYE
        MENUITEM SEPARATOR
        MENUITEM "E&xit",IDM_EXIT
```

```
    }
 MENUITEM "&Test", IDM_TEST
}
```

分析:

我们先来分析资源文件:

```
#define IDM_TEST 1                /* equal to IDM_TEST equ 1*/
#define IDM_HELLO 2
#define IDM_GOODBYE 3
#define IDM_EXIT 4
```

上面的几行定义了菜单项的ID号。只要注意菜单项ID号必须唯一外，您可以给ID号任何值。

```
FirstMenu MENU
```

用关键字MENU定义菜单。

```
 POPUP "&PopUp"
    {
     MENUITEM "&Say Hello",IDM_HELLO
     MENUITEM "Say &GoodBye", IDM_GOODBYE
     MENUITEM SEPARATOR
     MENUITEM "E&xit",IDM_EXIT
    }
```

定义一个有四个菜单项的子菜单，其中第三个菜单项是一个分隔线。

```
 MENUITEM "&Test", IDM_TEST
```

定义主菜单中的一项。下面我们来看看源代码。

```
MenuName db "FirstMenu",0
Test_string db "你选择了测试菜单项",0
Hello_string db "你好",0
Goodbye_string db "再见",0
```

MenuName是资源文件中指定的菜单的名字。因为您可以在脚本文件中定义任意多个菜单，所以**在用前必须指定您要使用那一个**，接下来的行是在选中菜单项时显示在相关对话框中的字符串。

```
IDM_TEST equ 1
IDM_HELLO equ 2
IDM_GOODBYE equ 3
IDM_EXIT equ 4
```

定义用在WINDOWS窗口过程中的菜单项ID号。这些值必须和脚本文件中的相同。

```
   .ELSEIF uMsg==WM_COMMAND
      mov eax,wParam
      .IF ax==IDM_TEST
         invoke MessageBox,NULL,ADDR Test_string,OFFSET AppName,MB_OK
      .ELSEIF ax==IDM_HELLO
         invoke MessageBox, NULL,ADDR Hello_string, OFFSET AppName,MB_OK
      .ELSEIF ax==IDM_GOODBYE
         invoke MessageBox,NULL,ADDR Goodbye_string, OFFSET AppName, MB_OK
```

```
    .ELSE
        invoke DestroyWindow,hWnd
    .ENDIF
```

在本窗口过程中我们处理WM_COMMAND消息。当用户选择了一个菜单项时，该菜单项的ID放入参数Param中被同时送到WINDOWS的窗口过程，我们把它保存到eax寄存器中以便和预定义的菜单项ID较用。前三种情况下，当我们选中Test、Say Hello、Say GoodBye菜单项时，会弹出一个对话框其显示一个相关的字符串，选择Exit菜单项时，我们就调用函数DestroyWindow，其中的参数是我们窗的句柄，这样就销毁了窗口。就像您所看到的，通过在一个窗口类中指定菜单名的方法来给一个应用序生成一个菜单是简单而直观的。除此方法外您还可以用另一种方法，其中资源文件是一样的，源文中也只有少数的改动，这些改动如下：

```
.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
hMenu HMENU ?                   ; handle of our menu
```

定义了一个变量来保存我们的菜单的句柄，然后：

```
    invoke LoadMenu, hInst, OFFSET MenuName
    mov    hMenu,eax
    INVOKE CreateWindowEx,NULL,ADDR ClassName,ADDR AppName,\
      WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,\
      CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,NULL,hMenu,\
      hInst,NULL
```

调用LoadMenu函数，该函数需要实例句柄和菜单名的字符串，调用的结果返回指向菜单的句柄，然传给函数CreateWindowEx刚返回的菜单句柄就可以了。