第十九章　多重文件介面

多重文件介面（MDI）是 Microsoft Windows 文件处理应用程式的一种规范，该规范描述了视窗结构和允许使用者在单个应用程式中使用多个文件的使用者介面（如文书处理程式中的文字文件和试算表程式中的试算表）。简单地说，就像 Windows 在一个萤幕上维护多个应用程式视窗一样，MDI 应用程式在一个显示区域内维护多个文件视窗。Windows 中的第一个 MDI 应用程式是 Windows 下的 Microsoft Excel 的第一个版本。紧接著又出现了许多其他的应用程式。

MDI 概念

尽管 MDI 规范随著 Windows 2.0 的推出已经很普及，但在那时，MDI 应用程式写起来很困难，并且需要一些非常复杂的程式设计工作。从 Windows 3.0 起，其中许多工作就都由 Windows 为您做好了。Windows 95 中增强的支援也已经被添加进 Windows 98 和 Microsoft Windows NT 中。

MDI 的组成

MDI 程式的主应用程式视窗是很普通的：它有一个标题列、一个功能表、一个缩放边框、一个系统功能表图示和最大化/最小化/关闭按钮。显示区域经常被称为「工作空间」，它不直接用於显示程式输出。这个工作空间包括零个或多个子视窗，每个视窗都显示一个文件。

这些子视窗看起来与通常的应用程式视窗以及 MDI 程式的主视窗很相似。它们有一个标题列、一个缩放边框、一个系统功能表图示和最大化/最小化/关闭按钮，可能还包括卷动列。但是文件视窗没有功能表，主应用程式视窗上的功能表适用於文件视窗。

在任何时候都只能有一个文件视窗是活动的（加亮标题列来表示），它出现在其他所有文件视窗之前。所有文件视窗都由工作空间区域加以剪裁，而不会出现在应用程式视窗之外。

初看起来，对 Windows 程式写作者来说，MDI 似乎是相当简单。需要程式写作者做的工作好像就是为每个文件建立一个 WS_CHILD 视窗，并使程式的主应用程式视窗成为文件视窗的父视窗。但对现有的 MDI 应用程式稍加研究，就会发现一些导致程式写作困难的复杂问题。例如：

- MDI 文件视窗可以最小化。它的图示出现在工作空间的底部。一般来说，MDI 应用程式可以将不同的图示分别用於主应用程式视窗和每一类文件

应用。

- MDI 文件视窗可以最大化。在这种情况下，文件视窗的标题列（一般用来显示视窗中文件的档案名称）消失，档案名称出现在应用程式视窗标题列的应用程式名称之後，文件视窗的系统功能表图示成为应用程式视窗的顶层功能表中的第一项。关闭文件视窗按钮变成顶层功能表中的最後一项，且出现在最右边。
- 用以关闭文件视窗的系统键盘加速键与关闭主视窗的系统键盘加速键一样，只是 Ctrl 键代替了 Alt 键。这也就是说，Alt+F4 用於关闭应用程式视窗，而 Ctrl+F4 用於关闭文件视窗。此外，Ctrl+F6 可以在活动 MDI 应用程式的子文件视窗之间切换。与平时一样，Alt+空白键启动主视窗的系统功能表，Alt+-（减号）启动活动子文件视窗的系统功能表。
- 当使用游标键在功能表项间移动时，控制项权通常从系统功能表转到功能表列中的第一项。在 MDI 应用程式中，控制项权是从应用程式系统功能表转到活动文件系统功能表，然後再转到功能表列中的第一项。
- 如果应用程式能够支援若干种型态的子视窗（如 Microsoft Excel 中的工作表和图表文件），那么功能表应能反映出与这种型态的文件有关的操作。这就要求当不同的文字视窗变成活动视窗时，程式能更换功能表。此外，当没有文件视窗存在时，功能表应该被缩减到只剩下与打开新文件有关的操作。
- 顶层功能表上有一个叫做「视窗（Window）」的功能表项。按照习惯，这是顶层功能表上「Help」之前的那一项，即倒数第二项。「视窗」子功能表上通常包含在工作空间内安排文件视窗的选项。文件视窗可以从左上方开始「平铺」或「层叠」。在前一种方式下，可以完整地看到每一个文件视窗。这个子功能表同时也包含所有文件视窗的列表。从中选择一个文件视窗，就可以把此文件视窗移到前景。

Windows 98 支援 MDI 的所有这些方面。当然，需要您做一些工作（如下面的范例程式所示），但是，这远不是要您程式写作来直接支援所有这些功能。

MDI 支援

探讨 Windows 的 MDI 支援时需要发表一些新术语。主应用程式视窗称为「框架视窗」，就像传统的 Windows 程式一样，它是 WS_OVERLAPPEDWINDOW 样式的视窗。

MDI 应用程式还根据预先定义的视窗类别 MDICLIENT 建立「客户视窗」，这一客户视窗是用这种视窗类别和 WS_CHILD 样式呼叫 CreateWindow 来建立的。

这一呼叫的最後一个参数是指向一个 CLIENTCREATESTRUCT 型态的结构的指标。这个客户视窗覆盖框架视窗的显示区域，并提供许多 MDI 支援。此客户视窗的颜色是系统颜色 COLOR_APPWORKSPACE。

文件视窗被称为「子视窗」。通过初始化一个 MDICREATESTRUCT 型态的结构，以一个指向此结构的指标为参数将讯息 WM_MDICREATE 发送给客户视窗，就可以建立这些文件视窗。

文件视窗是客户视窗的子视窗，而客户视窗又是框架视窗的子视窗。父-子视窗分层结构如图 19-1 所示。

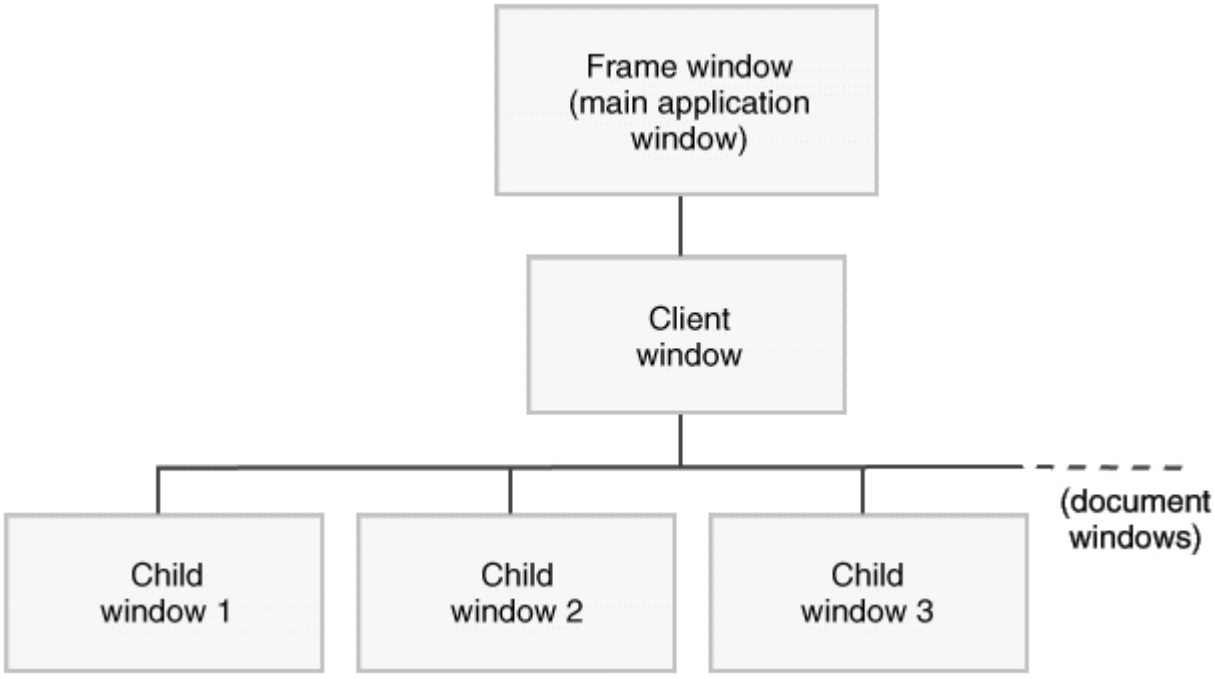

图 19-1 Windows MDI 应用程式的父-子层次图

您需要框架视窗的视窗类别（及视窗讯息处理程式）和一个由应用程式支援的每类子视窗的视窗类别（及视窗讯息处理程式）。由於已经预先注册了视窗类别，所以不需要客户视窗的视窗讯息处理程式。

Windows 98 的 MDI 支援包括一个视窗类别、五个函式、两个资料结构和 12 个讯息。前面已经提到了 MDI 视窗类别，即 MDICLIENT，以及资料结构 CLIENTCREATESTRUCT 和 MDICREATESTRUCT。在 MDI 应用程式中，这五个函式中的两个用於取代 DefWindowProc：不再将 DefWindowProc 呼叫用於所有未处理的讯息，而是由框架视窗程序呼叫 DefFrameProc，子视窗程序呼叫 DefMDIChildProc。另一个 MDI 特有的函式 TranslateMDISysAccel 与第十章中讨论的 TranslateAccelerator 的使用方式相同。MDI 支援也包括 ArrangeIconicWindows 函式，但有一条专用的 MDI 讯息使得此函式对 MDI 程式来说不再必要。

第五个 MDI 函式是 CreateMDIWindow，它使得子视窗可以在单独的执行绪中

被建立。这个函式不需要在单执行绪的程式中，我会展示这一点。

在下面的程式中，我将展示 12 条 MDI 讯息中的 9 条（其他三个讯息一般不用），这些讯息的字首是 WM_MDI。框架视窗向客户视窗发送其中某个讯息，以便在子视窗上完成一项操作或者取得关於子视窗的资讯（例如，框架视窗发送一个 WM_MDICREATE 讯息给客户视窗，以建立子视窗）。讯息 WM_MDIACTIVATE 讯息有点特别：框架视窗可以发送这个讯息给客户视窗来启动一个子视窗，而客户视窗也把这个讯息发送给将被启动或者失去活动的子视窗，以便通知它们这一变化。

MDI 的范例程式

程式 19-1 MDIDEMO 程式说明了编写 MDI 应用程式的基本方法。

程式 19-1 MDIDEMO

```
MDIDEMO.C
/*-----------------------------------------------------------------------------
-
      MDIDEMO.C -- Multiple-Document Interface Demonstration
                                             (c) Charles Petzold, 1998
-----------------------------------------------------------------------------*
/

#include <windows.h>
#include "resource.h"

#define INIT_MENU_POS                    0
#define HELLO_MENU_POS                2
#define RECT_MENU_POS                    1

#define IDM_FIRSTCHILD            50000

LRESULT     CALLBACK FrameWndProc              (HWND, UINT, WPARAM, LPARAM) ;
BOOL              CALLBACK CloseEnumProc       (HWND, LPARAM) ;
LRESULT     CALLBACK HelloWndProc              (HWND, UINT, WPARAM, LPARAM) ;
LRESULT     CALLBACK RectWndProc               (HWND, UINT, WPARAM, LPARAM) ;

      // structure for storing data unique to each Hello child window

typedef struct tagHELLODATA
{
      UINT                    iColor ;
      COLORREF          clrText ;
}
HELLODATA, * PHELLODATA ;
      // structure for storing data unique to each Rect child window
```

```
typedef struct tagRECTDATA
{
      short            cxClient ;
      short            cyClient ;
}
RECTDATA, * PRECTDATA ;
      // global variables
TCHAR                        szAppName[]            = TEXT ("MDIDemo") ;
TCHAR                        szFrameClass[]         = TEXT ("MdiFrame") ;
TCHAR                        szHelloClass[]         = TEXT ("MdiHelloChild") ;
TCHAR                        szRectClass[]          = TEXT ("MdiRectChild") ;
HINSTANCE        hInst ;
HMENU                        hMenuInit, hMenuHello, hMenuRect ;
HMENU                        hMenuInitWindow,               hMenuHelloWindow,
hMenuRectWindow ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                                    PSTR    szCmdLine,    int
iCmdShow)
{
      HACCEL                 hAccel ;
      HWND                         hwndFrame, hwndClient ;
      MSG                          msg ;
      WNDCLASS               wndclass ;

      hInst = hInstance ;
                        // Register the frame window class
      wndclass.style                     = CS_HREDRAW | CS_VREDRAW ;
      wndclass.lpfnWndProc               = FrameWndProc ;
      wndclass.cbClsExtra                = 0 ;
      wndclass.cbWndExtra                = 0 ;
      wndclass.hInstance                 = hInstance ;
      wndclass.hIcon                     = LoadIcon (NULL, IDI_APPLICATION) ;
      wndclass.hCursor                   = LoadCursor (NULL, IDC_ARROW) ;
      wndclass.hbrBackground       = (HBRUSH) (COLOR_APPWORKSPACE + 1) ;
      wndclass.lpszMenuName        = NULL ;
      wndclass.lpszClassName       = szFrameClass ;

      if (!RegisterClass (&wndclass))
      {
            MessageBox (     NULL, TEXT ("This program requires Windows NT!"),
                                                    szAppName, MB_ICONERROR) ;
                        return 0 ;
      }

                        // Register the Hello child window class
      wndclass.style                                 = CS_HREDRAW | CS_VREDRAW ;
      wndclass.lpfnWndProc               = HelloWndProc ;
```

```
     wndclass.cbClsExtra              = 0 ;
     wndclass.cbWndExtra              = sizeof (HANDLE) ;
     wndclass.hInstance               = hInstance ;
     wndclass.hIcon                   = LoadIcon (NULL, IDI_APPLICATION) ;
     wndclass.hCursor                 = LoadCursor (NULL, IDC_ARROW) ;
     wndclass.hbrBackground      = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
     wndclass.lpszMenuName       = NULL ;
     wndclass.lpszClassName      = szHelloClass ;

     RegisterClass (&wndclass) ;
                    // Register the Rect child window class
     wndclass.style                   = CS_HREDRAW | CS_VREDRAW ;
     wndclass.lpfnWndProc             = RectWndProc ;
     wndclass.cbClsExtra              = 0 ;
     wndclass.cbWndExtra              = sizeof (HANDLE) ;
     wndclass.hInstance               = hInstance ;
     wndclass.hIcon                   = LoadIcon (NULL, IDI_APPLICATION) ;
     wndclass.hCursor                 = LoadCursor (NULL, IDC_ARROW) ;
     wndclass.hbrBackground      = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
     wndclass.lpszMenuName       = NULL ;
     wndclass.lpszClassName      = szRectClass ;

     RegisterClass (&wndclass) ;
                    // Obtain handles to three possible menus & submenus
     hMenuInit        = LoadMenu  (hInstance, TEXT ("MdiMenuInit")) ;
     hMenuHello       = LoadMenu  (hInstance, TEXT ("MdiMenuHello")) ;
     hMenuRect        = LoadMenu  (hInstance, TEXT ("MdiMenuRect")) ;

     hMenuInitWindow        = GetSubMenu     (hMenuInit,   INIT_MENU_POS) ;
     hMenuHelloWindow       = GetSubMenu     (hMenuHello,    HELLO_MENU_POS)
     ;
     hMenuRectWindow        = GetSubMenu     (hMenuRect,   RECT_MENU_POS) ;

                    // Load accelerator table

     hAccel = LoadAccelerators (hInstance, szAppName) ;
                    // Create the frame window
     hwndFrame   = CreateWindow (szFrameClass, TEXT ("MDI Demonstration"),
                  WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
                  CW_USEDEFAULT, CW_USEDEFAULT,
                  CW_USEDEFAULT, CW_USEDEFAULT,
                  NULL, hMenuInit, hInstance, NULL) ;
     hwndClient = GetWindow (hwndFrame, GW_CHILD) ;
     ShowWindow (hwndFrame, iCmdShow) ;
     UpdateWindow (hwndFrame) ;

                    // Enter the modified message loop
     while (GetMessage (&msg, NULL, 0, 0))
```

```
     {
                if (  !TranslateMDISysAccel (hwndClient, &msg) &&
                           !TranslateAccelerator (hwndFrame, hAccel, &msg))
                     {
                           TranslateMessage (&msg) ;
                           DispatchMessage (&msg) ;
                     }
     }
                     // Clean up by deleting unattached menus
     DestroyMenu (hMenuHello) ;
     DestroyMenu (hMenuRect) ;

     return msg.wParam ;
     }

LRESULT CALLBACK FrameWndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
     static HWND                  hwndClient ;
     CLIENTCREATESTRUCT           clientcreate ;
     HWND                    hwndChild ;
     MDICREATESTRUCT              mdicreate ;

     switch (message)
     {
     case  WM_CREATE:                              // Create the client window

      clientcreate.hWindowMenu          = hMenuInitWindow ;
           clientcreate.idFirstChild          = IDM_FIRSTCHILD ;

      hwndClient = CreateWindow   (     TEXT ("MDICLIENT"), NULL,
                                   WS_CHILD | WS_CLIPCHILDREN | WS_VISIBLE,
                                   0, 0, 0, 0, hwnd, (HMENU) 1, hInst,
                                   (PSTR) &clientcreate) ;
                return 0 ;
     case  WM_COMMAND:
                     switch (LOWORD (wParam))
                     {
                     case  IDM_FILE_NEWHELLO:  // Create a Hello child window
                      mdicreate.szClass           = szHelloClass ;
                     mdicreate.szTitle           = TEXT ("Hello") ;
                     mdicreate.hOwner            = hInst ;
                     mdicreate.x                      = CW_USEDEFAULT ;
                     mdicreate.y                      = CW_USEDEFAULT ;
                     mdicreate.cx                = CW_USEDEFAULT ;
                     mdicreate.cy                = CW_USEDEFAULT ;
                     mdicreate.style             = 0 ;
                     mdicreate.lParam            = 0 ;
```

```
            hwndChild = (HWND) SendMessage (hwndClient,
        WM_MDICREATE, 0, (LPARAM) (LPMDICREATESTRUCT) &mdicreate) ;
            return 0 ;

            case  IDM_FILE_NEWRECT:      // Create a Rect child window

                  mdicreate.szClass              = szRectClass ;
                  mdicreate.szTitle              = TEXT ("Rectangles") ;
                  mdicreate.hOwner               = hInst ;
                  mdicreate.x                 = CW_USEDEFAULT ;
                  mdicreate.y                 = CW_USEDEFAULT ;
                  mdicreate.cx                = CW_USEDEFAULT ;
                  mdicreate.cy                = CW_USEDEFAULT ;
                  mdicreate.style                = 0 ;
                  mdicreate.lParam               = 0 ;

                  hwndChild = (HWND) SendMessage (hwndClient,
                        WM_MDICREATE, 0,
                      (LPARAM) (LPMDICREATESTRUCT) &mdicreate) ;
                                  return 0 ;

                  case  IDM_FILE_CLOSE:        // Close the active window

                  hwndChild = (HWND) SendMessage (hwndClient,
                      WM_MDIGETACTIVE, 0, 0) ;

                  if (SendMessage (hwndChild, WM_QUERYENDSESSION, 0, 0))
                        SendMessage (hwndClient, WM_MDIDESTROY,
               (WPARAM) hwndChild, 0) ;
                        return 0 ;

                  case  IDM_APP_EXIT:    // Exit the program

                  SendMessage (hwnd, WM_CLOSE, 0, 0) ;
                  return 0 ;

                  // messages for arranging windows

                  case  IDM_WINDOW_TILE:
                              SendMessage (hwndClient, WM_MDITILE, 0, 0) ;
                              return 0 ;

                  case  IDM_WINDOW_CASCADE:
                        SendMessage (hwndClient, WM_MDICASCADE, 0, 0) ;
                              return 0 ;

                  case  IDM_WINDOW_ARRANGE:
```

```
                              SendMessage (hwndClient, WM_MDIICONARRANGE, 0, 0) ;
                                    return 0 ;

                        case  IDM_WINDOW_CLOSEALL:   // Attempt to close all
children

                              EnumChildWindows (hwndClient, CloseEnumProc, 0) ;
                                    return 0 ;

                        default:               // Pass to active child...

                                    hwndChild = (HWND) SendMessage (hwndClient,
                                    WM_MDIGETACTIVE, 0, 0) ;
                                    if (IsWindow (hwndChild))
                        SendMessage (hwndChild, WM_COMMAND, wParam, lParam) ;

                              break ;    // ...and then to DefFrameProc
                        }
                        break ;

      case  WM_QUERYENDSESSION:
      case  WM_CLOSE:  // Attempt to close all children

                  SendMessage (hwnd, WM_COMMAND, IDM_WINDOW_CLOSEALL, 0) ;

                if (NULL != GetWindow (hwndClient, GW_CHILD))
                  return 0 ;

                        break ;     // i.e., call DefFrameProc
      case  WM_DESTROY:
                        PostQuitMessage (0) ;
                        return 0 ;
      }
       // Pass unprocessed messages to DefFrameProc (not DefWindowProc)

      return DefFrameProc (hwnd, hwndClient, message, wParam, lParam) ;
}

BOOL CALLBACK CloseEnumProc (HWND hwnd, LPARAM lParam)
{
      if (GetWindow (hwnd, GW_OWNER)) // Check for icon title
                        return TRUE ;

      SendMessage (GetParent (hwnd), WM_MDIRESTORE, (WPARAM) hwnd, 0) ;
      if (!SendMessage (hwnd, WM_QUERYENDSESSION, 0, 0))
       return TRUE ;
      SendMessage (GetParent (hwnd), WM_MDIDESTROY, (WPARAM) hwnd, 0) ;
      return TRUE ;
```

```
}

LRESULT CALLBACK HelloWndProc (HWND hwnd, UINT message,
WPARAM wParam, LPARAM lParam)
{
      static COLORREF clrTextArray[] = {      RGB (0,   0, 0), RGB (255, 0,   0),
       RGB (0, 255, 0), RGB (  0, 0, 255),
       RGB (255, 255, 255) } ;
      static HWND            hwndClient, hwndFrame ;
      HDC                    hdc ;
      HMENU                  hMenu ;
      PHELLODATA             pHelloData ;
      PAINTSTRUCT            ps ;
      RECT                   rect ;

      switch (message)
      {
      case  WM_CREATE:
             // Allocate memory for window private data

                        pHelloData = (PHELLODATA) HeapAlloc (GetProcessHeap (),
                               HEAP_ZERO_MEMORY, sizeof (HELLODATA)) ;
                        pHelloData->iColor  = IDM_COLOR_BLACK ;
                        pHelloData->clrText = RGB (0, 0, 0) ;
                        SetWindowLong (hwnd, 0, (long) pHelloData) ;

                                    // Save some window handles

                        hwndClient = GetParent (hwnd) ;
                        hwndFrame  = GetParent (hwndClient) ;
                        return 0 ;

      case  WM_COMMAND:
                        switch (LOWORD (wParam))
                        {
                              case IDM_COLOR_BLACK:
                              case IDM_COLOR_RED:
                              case IDM_COLOR_GREEN:
                              case IDM_COLOR_BLUE:
                              case IDM_COLOR_WHITE:
                    // Change the text color

             pHelloData = (PHELLODATA) GetWindowLong (hwnd, 0) ;

             hMenu = GetMenu (hwndFrame) ;

             CheckMenuItem (hMenu, pHelloData->iColor, MF_UNCHECKED) ;
             pHelloData->iColor = wParam ;
```

```
            CheckMenuItem (hMenu, pHelloData->iColor, MF_CHECKED) ;

            pHelloData->clrText = clrTextArray[wParam - IDM_COLOR_BLACK] ;

            InvalidateRect (hwnd, NULL, FALSE) ;
              }
              return 0 ;

     case  WM_PAINT:
                // Paint the window

                     hdc = BeginPaint (hwnd, &ps) ;

                     pHelloData = (PHELLODATA) GetWindowLong (hwnd, 0) ;
                     SetTextColor (hdc, pHelloData->clrText) ;

                     GetClientRect (hwnd, &rect) ;

                     DrawText (hdc, TEXT ("Hello, World!"), -1, &rect,
                DT_SINGLELINE | DT_CENTER | DT_VCENTER) ;
                     EndPaint (hwnd, &ps) ;
                     return 0 ;

     case  WM_MDIACTIVATE:
                // Set the Hello menu if gaining focus

                     if (lParam == (LPARAM) hwnd)
                SendMessage (hwndClient, WM_MDISETMENU,(WPARAM)
 hMenuHello, (LPARAM) hMenuHelloWindow) ;

                                     // Check or uncheck menu item

                     pHelloData = (PHELLODATA) GetWindowLong (hwnd, 0) ;
                     CheckMenuItem (hMenuHello, pHelloData->iColor,
                                (lParam == (LPARAM) hwnd) ? MF_CHECKED :
MF_UNCHECKED) ;

                                     // Set the Init menu if losing focus

                     if (lParam != (LPARAM) hwnd)
                                     SendMessage              (hwndClient,
WM_MDISETMENU, (WPARAM)
hMenuInit,(LPARAM) hMenuInitWindow) ;

                     DrawMenuBar (hwndFrame) ;
                     return 0 ;

     case  WM_QUERYENDSESSION:
```

```
      case  WM_CLOSE:
       if (IDOK != MessageBox (hwnd, TEXT ("OK to close window?"),
                               TEXT ("Hello"),
                  MB_ICONQUESTION | MB_OKCANCEL))
                        return 0 ;

                        break ;          // i.e., call DefMDIChildProc

      case  WM_DESTROY:
                        pHelloData = (PHELLODATA) GetWindowLong (hwnd, 0) ;
                        HeapFree (GetProcessHeap (), 0, pHelloData) ;
                        return 0 ;
      }
                              // Pass unprocessed message to DefMDIChildProc

      return DefMDIChildProc (hwnd, message, wParam, lParam) ;
}
LRESULT CALLBACK RectWndProc (     HWND hwnd, UINT message,WPARAM wParam, LPARAM
lParam)
{
      static HWND       hwndClient, hwndFrame ;
      HBRUSH      hBrush ;
      HDC         hdc ;
      PRECTDATA         pRectData ;
      PAINTSTRUCT       ps ;
      int                           xLeft, xRight, yTop, yBottom ;
      short                               nRed, nGreen, nBlue ;

      switch (message)
     {
      case  WM_CREATE:
                                    // Allocate memory for window private data

                        pRectData = (PRECTDATA) HeapAlloc (GetProcessHeap (),
                            HEAP_ZERO_MEMORY, sizeof (RECTDATA)) ;

                        SetWindowLong (hwnd, 0, (long) pRectData) ;

                                                  // Start the timer going

                        SetTimer (hwnd, 1, 250, NULL) ;

                                                        // Save some window
handles
                        hwndClient  = GetParent (hwnd) ;
                        hwndFrame   = GetParent (hwndClient) ;
                        return 0 ;
```

```
     case  WM_SIZE:    // If not minimized, save the window size

                       if (wParam != SIZE_MINIMIZED)
                       {
                             pRectData = (PRECTDATA) GetWindowLong (hwnd, 0) ;

                             pRectData->cxClient = LOWORD (lParam) ;
                             pRectData->cyClient = HIWORD (lParam) ;
                       }

                       break ;     // WM_SIZE must be processed by DefMDIChildProc

     case  WM_TIMER:                   // Display a random rectangle

                       pRectData  = (PRECTDATA) GetWindowLong (hwnd, 0) ;
                       xLeft                  = rand () % pRectData->cxClient ;
                       xRight           = rand () % pRectData->cxClient ;
                       yTop                   = rand () % pRectData->cyClient ;
                       yBottom          = rand () % pRectData->cyClient ;
                       nRed                   = rand () & 255 ;
                       nGreen           = rand () & 255 ;
                       nBlue                  = rand () & 255 ;

                       hdc = GetDC (hwnd) ;
                       hBrush = CreateSolidBrush (RGB (nRed, nGreen, nBlue)) ;
                       SelectObject (hdc, hBrush) ;

                       Rectangle (hdc, min (xLeft, xRight), min (yTop, yBottom),
                                  max (xLeft, xRight), max (yTop, yBottom)) ;

                       ReleaseDC (hwnd, hdc) ;
                       DeleteObject (hBrush) ;
                       return 0 ;

     case  WM_PAINT:                                   // Clear the window

                       InvalidateRect (hwnd, NULL, TRUE) ;
                       hdc = BeginPaint (hwnd, &ps) ;
                       EndPaint (hwnd, &ps) ;
                       return 0 ;

     case  WM_MDIACTIVATE:                   // Set the appropriate menu
                       if (lParam == (LPARAM) hwnd)
                  SendMessage (hwndClient, WM_MDISETMENU, (WPARAM) hMenuRect,
(LPARAM) hMenuRectWindow) ;
                       else
                  SendMessage (hwndClient, WM_MDISETMENU, (WPARAM) hMenuInit,
(LPARAM) hMenuInitWindow) ;
```

```
                              DrawMenuBar (hwndFrame) ;
                              return 0 ;

      case  WM_DESTROY:
                         pRectData = (PRECTDATA) GetWindowLong (hwnd, 0) ;
                         HeapFree (GetProcessHeap (), 0, pRectData) ;
                         KillTimer (hwnd, 1) ;
                         return 0 ;
    }
         // Pass unprocessed message to DefMDIChildProc
    return DefMDIChildProc (hwnd, message, wParam, lParam) ;
}
```

MDIDEMO.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////////////////////////////////////////////
/
// Menu
MDIMENUINIT MENU DISCARDABLE
BEGIN
      POPUP "&File"
      BEGIN
                         MENUITEM "New &Hello",
      IDM_FILE_NEWHELLO
                         MENUITEM "New &Rectangle",
      IDM_FILE_NEWRECT
                         MENUITEM SEPARATOR
                         MENUITEM "E&xit", IDM_APP_EXIT
      END
END
MDIMENUHELLO MENU DISCARDABLE
BEGIN
      POPUP "&File"
      BEGIN
MENUITEM "New &Hello",       IDM_FILE_NEWHELLO
MENUITEM "New &Rectangle",   IDM_FILE_NEWRECT
MENUITEM "&Close",           IDM_FILE_CLOSE
MENUITEM SEPARATOR
MENUITEM "E&xit",                 IDM_APP_EXIT
END
      POPUP "&Color"
      BEGIN
       MENUITEM "&Black",               IDM_COLOR_BLACK
       MENUITEM "&Red",          IDM_COLOR_RED
       MENUITEM "&Green",           IDM_COLOR_GREEN
```

```
        MENUITEM "B&lue",           IDM_COLOR_BLUE
        MENUITEM "&White",              IDM_COLOR_WHITE
      END
      POPUP "&Window"
      BEGIN
        MENUITEM "&Cascade\tShift+F5",          IDM_WINDOW_CASCADE
        MENUITEM "&Tile\tShift+F4",             IDM_WINDOW_TILE
        MENUITEM "Arrange &Icons",                  IDM_WINDOW_ARRANGE
        MENUITEM "Close &All",                      IDM_WINDOW_CLOSEALL
      END
END
MDIMENURECT MENU DISCARDABLE
BEGIN
      POPUP "&File"
      BEGIN
        MENUITEM "New &Hello",                      IDM_FILE_NEWHELLO
        MENUITEM "New &Rectangle",              IDM_FILE_NEWRECT
        MENUITEM "&Close",                  IDM_FILE_CLOSE
        MENUITEM SEPARATOR
        MENUITEM "E&xit",                           IDM_APP_EXIT
      END
      POPUP "&Window"
      BEGIN
        MENUITEM "&Cascade\tShift+F5",          IDM_WINDOW_CASCADE
        MENUITEM "&Tile\tShift+F4",             IDM_WINDOW_TILE
        MENUITEM "Arrange &Icons",              IDM_WINDOW_ARRANGE
        MENUITEM "Close &All",                IDM_WINDOW_CLOSEALL
      END
END

//////////////////////////////////////////////////////////////////////////////
/
// Accelerator
MDIDEMO ACCELERATORS DISCARDABLE
BEGIN
    VK_F4,    IDM_WINDOW_TILE,    VIRTKEY, SHIFT, NOINVERT
    VK_F5,    IDM_WINDOW_CASCADE,  VIRTKEY, SHIFT, NOINVERT
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by MDIDemo.rc

#define IDM_FILE_NEWHELLO                                       40001
#define IDM_FILE_NEWRECT                                        40002
#define IDM_APP_EXIT                                            40003
#define IDM_FILE_CLOSE                                          40004
#define IDM_COLOR_BLACK                                         40005
#define IDM_COLOR_RED                                           40006
```

```
#define IDM_COLOR_GREEN                                    40007
#define IDM_COLOR_BLUE                                     40008
#define IDM_COLOR_WHITE                                    40009
#define IDM_WINDOW_CASCADE                                40010
#define IDM_WINDOW_TILE                                    40011
#define IDM_WINDOW_ARRANGE                                40012
#define IDM_WINDOW_CLOSEALL                               40013
```

MDIDEMO 支援两种型态的非常简单的文件视窗：第一种视窗在它的显示区域中央显示"Hello, World!"，另一种视窗显示一系列随机矩形（在原始码列表和识别字名中，它们分别叫做「Hello」文件和「Rect」文件）。这两类文件视窗的功能表不同，显示"Hello, World!"的文件视窗有一个允许使用者修改文字颜色的功能表。

三个功能表

现在让我们先看看 MDIDEMO.RC 资源描述档，它定义了程式所使用的三个功能表模板。

当文件视窗不存在时，程式显示 MdiMenuInit 功能表，这个功能表只允许使用者建立新文件或退出程式。

MdiMenuHello 功能表与显示「Hello, World!」的文件视窗相关联。「File」子功能表允许使用者打开任何一类新文件、关闭活动文件或退出程式。「Color」子功能表允许使用者设定文字颜色。Window 子功能表包括以平铺或者重叠的方式安排文件视窗、安排文件图示或关闭所有视窗等选项，这个子功能表也列出了它们建立的所有文件视窗。

MdiMenuRect 功能表与随机矩形文件相关联。除了不包含「Color」子功能表外，它与 MdiMenuHello 功能表一样。

RESOURCE.H 表头档案定义所有的功能表识别字。另外，以下三个常数定义在 MDIDEMO.C 中：

```
#define INIT_MENU_POS              0
#define HELLO_MENU_POS             2
#define RECT_MENU_POS              1
```

这些识别字说明每个功能表模板中 Windows 子功能表的位置。程式需要这些资讯来通知客户视窗文件列表应出现在哪里。当然，MdiMenuInit 功能表没有 Windows 子功能表，所以如前所述，文件列表应附加在第一个子功能表中（位置 0）。不过，实际上永远不会在此看到文件列表（在後面讨论此程式时，您可以发现这样做的原因）。

定义在 MDIDEMO.C 中的 IDM_FIRSTCHILD 识别字不对应於功能表项，它与出现在 Windows 子功能表上的文件列表中的第一个文件视窗相关联。这个识别字

的值应当大於所有其他功能表 ID 的值。

程式初始化

在 MDIDEMO.C 中，WinMain 是从注册框架视窗和两个子视窗的视窗类别开始的。视窗讯息处理程式是 FrameWndProc、HelloWndProc 和 RectWndProc。一般来说，这些视窗类别应该与不同的图示相关联。为了简单起见，我们将标准 IDI_APPLICATION 图示用於框架视窗和子视窗。

注意，我们已经定义了框架视窗类别的 WNDCLASS 结构的 hbrBackground 栏位为 COLOR_APPWORKSPACE 系统颜色。由於框架视窗的显示区域被客户视窗所覆盖并且客户视窗具有这种颜色，所以上面的定义不是绝对必要的。但是，在最初显示框架视窗时，使用这种颜色似乎要好一些。

这三种视窗类别中的 lpszMenuName 栏位都设定为 NULL。对「Hello」和「Rect」子视窗类别来说，这是很自然的。对於框架视窗类别，我在建立框架视窗时在 CreateWindow 函式中给出功能表代号。

「Hello」和「Rect」子视窗的视窗类别将 WNDCLASS 结构中的 cbWndExtra 栏位设为非零值来为每个视窗配置额外空间，这个空间将用於储存指向一个记忆体块的指标(HELLODATA 和 RECTDATA 结构的大小定义在 MDIDEMO.C 的开始处)，这个记忆体块被用於储存每个文件视窗特有的资讯。

下一步，WinMain 用 LoadMenu 载入三个功能表，并把它们的代号储存到整体变数中。呼叫三次 GetSubMenu 函式可获得 Windows 子功能表（文件列表将加在它上面）的代号，同样也把它们储存到整体变数中。LoadAccelerators 函式载入加速键表。

在 WinMain 中呼叫 CreateWindow 建立框架视窗。在 FrameWndProc 中 WM_CREATE 讯息处理期间，框架视窗建立客户视窗。这项操作涉及到再一次呼叫函式 CreateWindow。视窗类别被设定为 MDICLIENT，它是预先注册的 MDI 显示区域视窗类别。在 Windows 中许多对 MDI 的支援被放入了 MDICLIENT 视窗类别中。显示区域视窗讯息处理程式作为框架视窗和不同文件视窗的中间层。当呼叫 CreateWindow 建立显示区域视窗时，最後一个参数必须被设定为指向 CLIENTCREATESTRUCT 型态结构的指标。这个结构有两个栏位：

- hWindowMenu 是要加入文件列表的子功能表的代号。在 MDIDEMO 中，它是 hMenuInitWindow，是在 WinMain 期间获得的。後面将看到如何修改此功能表。
- idFirstChild 是与文件列表中的第一个文件视窗相关联的功能表 ID。它就是 IDM_FIRSTCHILD。

再让我们回过头来看看 WinMain。MDIDEMO 显示新建立的框架视窗并进入讯息回圈。讯息回圈与正常的回圈稍有不同：在呼叫 GetMessage 从讯息伫列中获得讯息之後，MDI 程式把这个讯息传送给了 TranslateMDISysAccel（以及 TranslateAccelerator，如果像 MDIDEMO 程式一样，程式本身也有功能表加速键的话）。

TranslateMDISysAccel 函式把可能对应特定 MDI 加速键（例如 Ctrl-F6）的按键转换成 WM_SYSCOMMAND 讯息。如果 TranslateMDISysAccel 或 TranslateAccelerator 都传回 TRUE（表示某个讯息已被这些函式之一转换），就不能呼叫 TranslateMessage 和 DispatchMessage。

注意传递到 TranslateMDISysAccel 和 TranslateAccelerator 的两个视窗代号：hwndClient 和 hwndFrame。WinMain 函式通过用 GW_CHILD 参数呼叫 GetWindow 获得 hwndClient 视窗代号。

建立子视窗

FrameWndProc 的大部分工作是用於处理通知功能表选择的 WM_COMMAND 讯息。与平时一样，FrameWndProc 中 wParam 参数的低字组包含著功能表 ID。

在功能表 ID 的值为 IDM_FILE_NEWHELLO 和 IDM_FILE_NEWRECT 的情况下，FrameWndProc 必须建立一个新的文件视窗。这涉及到初始化 MDICREATESTRUCT 结构中的栏位（大多数栏位对应於 CreateWindow 的参数），并将讯息 WM_MDICREATE 发送给客户视窗，讯息的 lParam 参数设定为指向这个结构的指标。然後由客户视窗建立子文件视窗。（也可以使用 CreateMDIWindow 函式。）

MDICREATESTRUCT 结构中的 szTitle 栏位一般是对应於文件的档案名称。样式栏位设定为视窗样式 WS_HSCROLL、WS_VSCROLL 或这两者的组合，以便在文件视窗中包括卷动列。样式栏位也可以包括 WS_MINIMIZE 或 WS_MAXIMIZE，以便在最初时以最小化或最大化状态显示文件视窗。

MDICREATESTRUCT 结构的 lParam 栏位为框架视窗和子视窗共用某些变数提供了一种方法。这个栏位可以设定为含有一个结构的记忆体块的记忆体代号。在子文件视窗的 WM_CREATE 讯息处理期间，lParam 是一个指向 CREATESTRUCT 结构的指标，这个结构的 lpCreateParams 栏位是一个指向用於建立视窗的 MDICREATESTRUCT 结构的指标。

客户视窗一旦接收到 WM_MDICREATE 讯息就建立一个子文件视窗，并把视窗标题加到用於建立客户视窗的 MDICLIENTSTRUCT 结构中所指定的子功能表的底部。当 MDIDEMO 程式建立它的第一个文件视窗时，这个子功能表就是「MdiMenuInit」功能表中的「File」子功能表。後面将看到这个文件列表将如

何移到「MdiMenuHello」和「MdiMenuRect」功能表的「Windows」子功能表中。

功能表上可以列出 9 个文件，每个文件的前面是带有底线的数字 1 至 9。如果建立的文件视窗多於 9 个，则这个清单後跟有「More Windows」功能表项。该项启动带有清单方块的对话方块，清单方块列出了所有文件。这种文件列表的维护是 Windows MDI 支援的最好特性之一。

关於框架视窗的讯息处理

在把注意力转移到子文件视窗之前，我们先继续讨论 FrameWndProc 的讯息处理。

当从「File」功能表中选择「Close」时，MDIDEMO 关闭活动子视窗。它通过把 WM_MDIGETACTIVE 讯息发送给客户视窗，而获得活动子视窗的代号。如果子视窗以 WM_QUERYENDSESSION 讯息来回应，那么 MDIDEMO 将 WM_MDIDESTROY 讯息发送给客户视窗，从而关闭子视窗。

处理「File」功能表中的「Exit」选项只需要框架视窗讯息处理程式给自己发送一个 WM_CLOSE 讯息。

处理 Window 子功能表的「Tile」、「Cascade」和「Arrange」选项是极容易的，只需把讯息 WM_MDITILE、WM_MDICASCADE 和 WM_MDIICONARRANGE 发送给客户视窗。

处理「Close All」选项要稍微复杂一些。FrameWndProc 呼叫 EnumChildWindows，传送一个引用 CloseEnumProc 函式的指标。此函式把 WM_MDIRESTORE 讯息发送给每个子视窗，紧跟著发出 WM_QUERYENDSESSION 和 WM_MDIDESTROY。对图示平铺视窗来说并不就此结束，用 GW_OWNER 参数呼叫 GetWindow 时，传回的非 NULL 值可以显示出这一点。

FrameWndProc 没有处理任何由「Color」功能表中对颜色的选择所导致的 WM_COMMAND 讯息，这些讯息应该由文件视窗负责处理。因此，FrameWndProc 把所有未经处理的 WM_COMMAND 讯息发送到活动子视窗，以便子视窗可以处理那些与它们有关的讯息。

框架视窗讯息处理程式不予处理的所有讯息都要送到 DefFrameProc，它在框架视窗讯息处理程式中取代了 DefWindowProc。即使框架视窗讯息处理程式拦截了 WM_MENUCHAR、WM_SETFOCUS 或 WM_SIZE 讯息，这些讯息也要被送到 DefFrameProc 中。

所有未经处理的 WM_COMMAND 讯息也必须送给 DefFrameProc。具体地说，FrameWndProc 并不处理任何 WM_COMMAND 讯息，即使这些讯息是使用者在 Windows 子功能表的文件列表中选择文件时产生的（这些选项的 wParam 值是以

IDM_FIRSTCHILD 开始的）。这些讯息要传送到 DefFrameProc，并在那里进行处理。

注意框架视窗并不需要维护它所建立的所有文件视窗的视窗代号清单。如果需要这些视窗代号（如处理功能表上的「Close All」选项时），可以使用 EnumChildWindows 得到它们。

子文件视窗

现在看一下 HelloWndProc，它是用於显示「Hello, World!」的子文件视窗的视窗讯息处理程式。

与用於多个视窗的视窗类别一样，所有在视窗讯息处理程式（或从该视窗讯息处理程式中呼叫的任何函式）中定义的静态变数由依据该视窗类别建立的所有视窗共用。

只有对於每个唯一於视窗的资料才必须采用非静态变数的方法来储存。这样的技术要用到视窗属性。另一种方法（我使用的方法）是使用预留的记忆体空间；可以在注册视窗类别时将 WNDCLASS 结构的 cbWndExtra 栏位设定为非零值以便预留这部分记忆体空间。

MDIDEMO 程式使用这个记忆体空间来储存一个指标，这个指标指向一块与 HELLODATA 结构大小相同的记忆体块。在处理 WM_CREATE 讯息时，HelloWndProc 配置这块记忆体，初始化它的两个栏位（它们用於指定目前选中的功能表项和文字颜色），并用 SetWindowLong 将记忆体指标储存到预留的空间中。

当处理改变文字颜色的 WM_COMMAND 讯息（回忆一下，这些讯息来自框架视窗讯息处理程式）时，HelloWndProc 使用 GetWindowLong 获得包含 HELLODATA 结构的记忆体块的指标。利用这个结构，HelloWndProc 清除原来对功能表项的选择，设定所选功能表项为选中状态，并储存新的颜色。

当视窗变成活动视窗或不活动的时候，文件视窗讯息处理程式都会收到 WM_MDIACTIVATE 讯息（lParam 的值是否为这个视窗的代号表示了该视窗是活动的还是不活动的）。您也许还能记起 MDIDEMO 程式中有三个不同的功能表：当无文件时为 MdiMenuInit；当「Hello」文件视窗是活动视窗时为 MdiMenuHello；当「Rect」文件视窗为活动视窗时为 MdiMenuRect。

WM_MDIACTIVATE 讯息为文件视窗提供了一个修改功能表的机会。如果 lParam 中含有本视窗的代号（意味著本视窗将变成活动的），那么 HelloWndProc 就将功能表改为 MdiMenuHello。如果 lParam 中包含另一个视窗的代号，那么 HelloWndProc 将功能表改为 MdiMenuInit。

HelloWndProc 经由把 WM_MDISETMENU 讯息发送给客户视窗来修改功能表，

客户视窗透过从目前功能表上删除文件列表并把它添加到一个新的功能表上来处理这个讯息。这就是文件列表从 MdiMenuInit 功能表（它在建立第一个文件时有效）传送到 MdiMenuHello 功能表中的方法。在 MDI 应用程式中不要用 SetMenu 函式改变功能表。

另一项工作涉及到「Color」子功能表上的选中旗标。像这样的程式选项对每个文件来说都是不同的，例如，可以在一个视窗中设定黑色文字，在另一个视窗中设定红色文字。功能表选中旗标应能反映出活动视窗中选择的选项。由於这种原因，HelloWndProc 在视窗变成非活动视窗时清除选中功能表项的选中旗标，而当视窗变成活动视窗时设定适当功能表项的选中旗标。

WM_MDIACTIVATE 的 wParam 和 lParam 值分别是失去活动和被启动视窗的代号。视窗讯息处理程式得到的第一个 WM_MDIACTIVATE 讯息的 lParam 参数被设定为目前视窗的代号。而当视窗被消除时，视窗讯息处理程式得到的最後一个讯息的 lParam 参数被设定为另一个值。当使用者从一个文件切换到另一个文件时，前一个文件视窗收到一个 WM_MDIACTIVATE 讯息，其 lParam 参数为第一个视窗的代号（此时，视窗讯息处理程式将功能表设定为 MdiMenuInit）；後一个文件视窗收到一个 WM_MDIACTIVATE 讯息，其 lParam 参数是第二个视窗的代号（此时，视窗讯息处理程式将功能表设定为 MdiMenuHello 或 MdiMenuRect 中适当的那个）。如果所有的视窗都关闭了，剩下的功能表就是 MdiMenuInit。

当使用者从功能表中选择「Close」或「Close All」时，FrameWndProc 给子视窗发送一个 WM_QUERYENDSESSION 讯息。HelloWndProc 将显示一个讯息方块并询问使用者是否要关闭视窗，以此来处理 WM_QUERYENDSESSION 和 WM_CLOSE 讯息（在真实的应用程式中，讯息方块会询问是否需要储存档案）。如果使用者表示不能关闭视窗，那么视窗讯息处理程式传回 0。

在 WM_DESTROY 讯息处理期间，HelloWndProc 释放在 WM_CREATE 期间配置的记忆体块。

所有未经处理的讯息必须传送到用於内定处理的 DefMDIChildProc（不是 DefWindowProc）。不论子视窗讯息处理程式是否使用了这些讯息，有几个讯息必须被传送给 DefMDIChildProc。这些讯息是：WM_CHILDACTIVATE、WM_GETMINMAXINFO、WM_MENUCHAR、WM_MOVE、WM_SETFOCUS、WM_SIZE 和 WM_SYSCOMMAND。

RectWndProc 与 HelloWndProc 非常相似，但是它比 HelloWndProc 要简单一些（不含功能表选项并且无需使用者确认是否关闭视窗），所以这里不对它进行讨论了。但应该注意到，在处理 WM_SIZE 之後 RectWndProc 使用了「break」叙述，所以 WM_SIZE 讯息被传给 DefMDIChildProc。

结束处理

在 WinMain 中，MDIDEMO 使用 LoadMenu 载入资源描述档中定义的三个功能表。一般说来，当功能表所在的视窗被清除时，Windows 也要清除与之关联的功能表。对於 Init 功能表，应该清除那些没有联系到视窗的功能表。由於这个原因,MDIDEMO 在 WinMain 的末尾呼叫了两次 DestroyMenu 来清除「Hello」和「Rect」功能表。