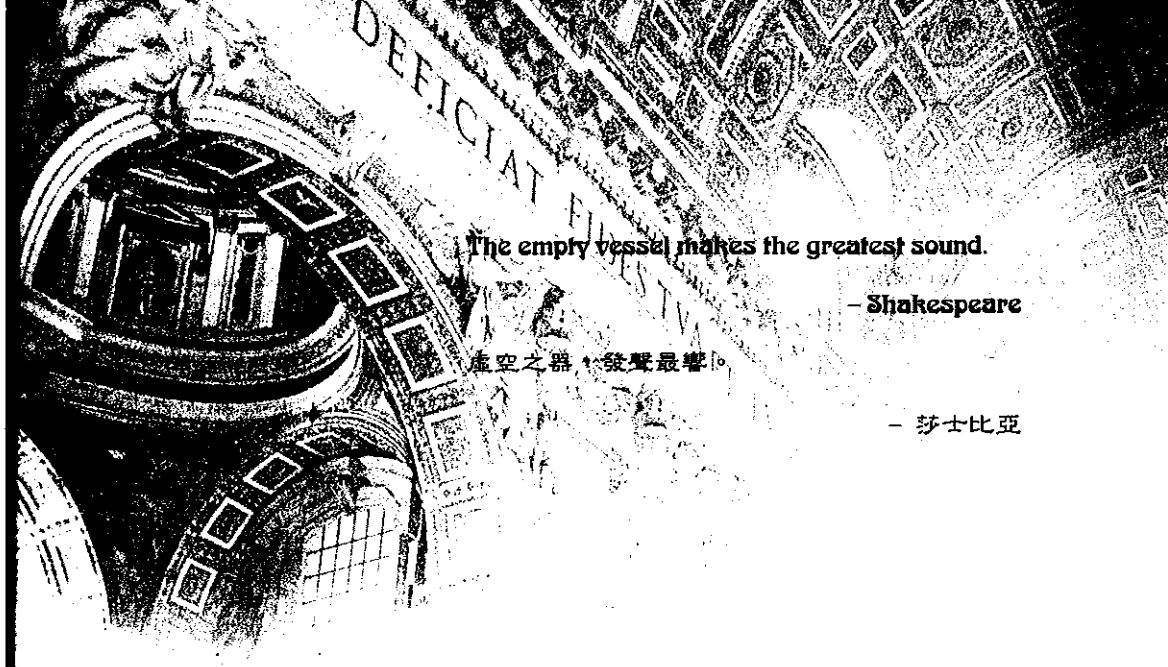


The empty vessel makes the greatest sound.

– Shakespeare

虛空之器，發聲最響。

– 莎士比亞

5

視窗的訊息處理

本章導讀

第 1 章裡，我們提到過視窗程式是以訊息為運作的基礎。這一章裡，將更進一步說明訊息的種類，以及該如何建立視窗的訊息回應機制。

各節標題

5-1 視窗訊息的傳遞與處理

1-1-1 節介紹了視窗程式的執行是以訊息回應為基礎，在談如何回應訊息前，首先要讓各位知道，視窗作業系統下，電腦系統的周邊裝置將不斷發出訊息，就連您輕輕地移動一下滑鼠，系統也會收到滑鼠移動的訊息。

當然並不是所有的訊息都必須回應，比如：您很無聊地在作業系統的桌面上隨意移動滑鼠，這種訊息系統就沒有必要理會。在這樣多而繁雜的訊息中，如何讓應用程式知道究竟哪些訊息需要回應？以及該如何回應呢？這運用訊息映射表建立訊息與回應函數的連結。

訊息映射表的宣告

利用 MFC 撰寫視窗程式時，應用程式回應訊息的機制是靠建立**訊息映射表**完成。首先，在負責回應訊息的類別裡，加入訊息映射表的宣告。

```
DECLARE_MESSAGE_MAP()  //宣告訊息映射表
```

然後，在該類別外，運用以下語法建立訊息映射表，宣告欲處理的訊息回應項目。

```
BEGIN_MESSAGE_MAP (類別名稱 , 基礎類別名稱)
   訊息回應項目
   ...
END_MESSAGE_MAP ()
```

其中，訊息回應項目的定義，由於牽涉到訊息的種類與回應函數的定義方式，詳細說明於後。

視窗訊息的種類與訊息回應項目的定義

對於**訊息回應項目**的定義，依據訊息種類的不同可分為以下二種：

一、標準系統訊息

標準系統訊息是由作業系統產生的訊息，比如：滑鼠左鍵被按下或移動滑鼠...等。而接收並處理這類標準系統訊息的類別，必須衍生自 CWnd 類別。

回應這類訊息之訊息項目名稱的命名規則為 ON_WM_XXX，如：滑鼠移動產生訊息的訊息回應項目名稱為『ON_WM_MOUSEMOVE』。以下敘述將於訊息映射表定義回應滑鼠移動訊息之訊息回應項目。

```
BEGIN_MESSAGE_MAP (..., ...)
   ON_WM_MOUSEMOVE()
   ...
END_MESSAGE_MAP ()
```

對於回應標準系統訊息的函數名稱，MFC 均已定義，如：回應滑鼠移動訊息的函數名稱為『OnMouseMove』，以下為於程式內宣告該函數的語法：

```
afx_msg void 類別名稱::OnMouseMove(UINT nFlags, CPoint point)
{                                   //如果宣告在類別內，不須宣告類別名稱
   ...      //回應敘述
}
```

其中 nFlags 與 point 為傳入回應函數的參數，詳細說明請參考 5-3 節。

二、命令訊息

命令訊息大致是由使用者建立的功能表、工具列、控制項...等視窗元件，被選取時產生。回應這類訊息的類別必須衍生自 CCmdTarget 類別，以下為這類訊息回應項目的定義語法：

```
BEGIN_MESSAGE_MAP (…, …)
   ON_COMMAND(訊息代號 , 回應函數)
   …
END_MESSAGE_MAP ()
```

與標準系統訊息不同的是，回應命令訊息的訊息回應項目，必須定義欲回應之命令訊息的代號與回應的函數名稱。其中，訊息代號為建立資源物件時所設定的**識別子**，比如：4-2 節 MyFrame 範例內，指定給 File 功能表 Exit 選項的 ID_FILE_EXIT 識別子。

當宣告訊息處理函數時，必須在該函數前加上 afx_msg，以下為宣告訊息處理函數的語法。

```
afx_msg  void  回應函數名稱(    )
```

因此，宣告 OnExit()函數處理 ID_EXIT1 訊息的訊息回應項目的語法如下：

```
BEGIN_MESSAGE_MAP (…, …)
   ON_COMMAND(ID_EXIT1 , OnExit)
   …
END_MESSAGE_MAP ()
```

函數的宣告方式如下：

```
afx_msg  void  OnExit( )  { … }
```

對於一些大部份視窗程式常使用的命令，MFC 預先定義了**標準命令識別子**（Standard Command IDs），並完成對應的回應函數，例如：關閉視窗、關閉應用程式...等。因此，當使用 MFC 已定義的命令訊息，可以不必定義回應函數，也不必宣告回應的函數。以下將以建立 MyFrame 類別的訊息映射表為例，說明如何建立訊息映射表。

```
class MyFrame : public CFrameWnd
{
   …
   DECLARE_MESSAGE_MAP() //宣告 MyFrame 類別擁有一個訊息映射表
```

```
);
BEGIN_MESSAGE_MAP(MyFrame, CFrameWnd) //宣告訊息映射表回應的訊息
    ON_COMMAND(ID_Exit1, OnExit) //與處理函數
    ON_WM_LBUTTONDOWN()
    ON_WM_MOUSEMOVE()
    ON_WM_LBUTTONUP()
END_MESSAGE_MAP()
```

命令訊息

標準系統訊息

以上宣告裡，定義 MyFrame 類別將接收並處理四種訊息。一是點選功能表中，ID 為 ID_EXIT1 的功能表項目，並以 OnExit()函數處理該訊息。另外三個則是滑鼠發出的標準系統訊息，而處理標準系統訊息的函數之名稱已由 MFC 預設，因此並不需要再標明（有關處理滑鼠訊息的說明請參考 5-4 節的說明）。

5-2 Message 程式範例

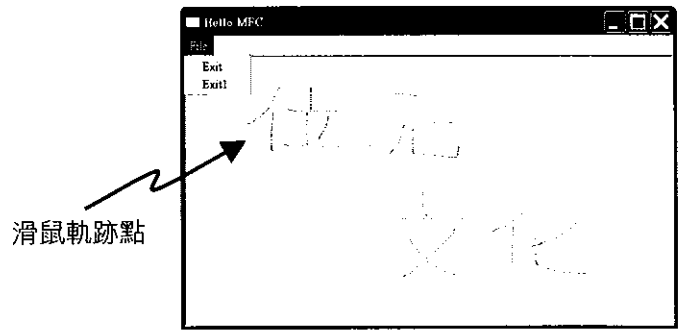
上一個小節大致講解了訊息的種類與建立方式，而本節的程式範例將承襲上一個範例，增加回應訊息的部份。此範例裡，將回應兩種訊息，一是由滑鼠傳出的**標準系統訊息**，另一是由功能表傳出的**命令訊息**。本範例回應滑鼠訊息時，將在視窗中繪出滑鼠移動的軌跡。命令訊息的部份，則示範如何關閉視窗。本程式範例的重點如下：

1.訊息映射表與回應函數的建立（詳細說明於 5-3 節）

2.利用滑鼠繪圖（詳細說明於 5-4 節）

3.訊息方塊的使用與視窗的破壞（詳細說明於 5-5 節）

以下是 Message 程式範例的執行結果。



Message 程式範例 - 訊息映射表的建立

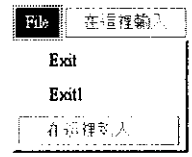
檔案位置：Message\Message.cpp

```
1   /*
2   範例檔名：Message.cpp
3   程式開發：郭尚君
4   */
5   #include <afxwin.h>
6   #include "Message.h" //載入資源檔所使用之標頭檔
7
8   class MyFrame : public CFrameWnd
9   {
10  private:
11     CMenu *FMenu;
12  public:
13     MyFrame()  //建構子
14     {
15        Create(NULL,"Hello MFC");
16        FMenu = new CMenu;
17        FMenu->LoadMenu(IDR_MENU1);
18        SetMenu(FMenu);
19     }
20
21     ~MyFrame(){ delete FMenu;} //解構子
22
23     afx_msg void OnExit()  //ID_EXIT1 的回應函數
24     {
25        MessageBox("Exit1");
26        DestroyWindow(); //破壞視窗
27     }
```

```
28
29      afx_msg void OnLButtonDown(UINT nFlags, CPoint point)
30      { SetCapture(); }    //當滑鼠左鍵按下後的回應函數，取得滑鼠訊息接收權
31
32      afx_msg void OnMouseMove(UINT nFlags, CPoint point)
33      {            //當滑鼠移動時的回應函數
34         if (this == GetCapture())     //判斷滑鼠游標是否在視窗之上
35         {
36            CClientDC aDC(this);  //建立一個畫布
37            aDC.SetPixel(point, RGB(255, 0 ,0));
              //利用 SetPixel 在畫布上點出紅點
38         }
39      }
40
41      afx_msg void OnLButtonUp(UINT nFlags, CPoint point)
42      { ReleaseCapture();} //當滑鼠左鍵放開後的回應函數，釋放滑鼠訊息接收權
43
44      DECLARE_MESSAGE_MAP()  //宣告訊息映射表
45   };
46
47   BEGIN_MESSAGE_MAP(MyFrame, CFrameWnd)
48   //建立 MyFrame 類別的訊息映射表
49      ON_COMMAND(ID_Exit1, OnExit)     //回應 ID_EXIT1 訊息
50      ON_WM_LBUTTONDOWN() //回應滑鼠左鍵被按下
51      ON_WM_MOUSEMOVE()   //回應滑鼠游標移動
52      ON_WM_LBUTTONUP()   //回應滑鼠左鍵被放開
53   END_MESSAGE_MAP()
54
55   class MyApp : public CWinApp  //應用程式類別
56   {
57   public:
58      BOOL InitInstance() //程式進入點 <-- 程式進入點
59      {
60         CFrameWnd *Frame = new MyFrame;
61         m_pMainWnd = Frame;
62         Frame->ShowWindow(SW_SHOW);  //顯示視窗
63
64         return true;
65      }
66   } a_app; //宣告應用程式物件
```

✤ 使用資源

一、功能表資源：以下是程式範例所使用的功能表資源。

識別子	IDR_MENU1
功能表	
用途說明	關閉視窗功能表

二、選項代號：

選項	ID
Exit	ID_APP_EXIT
Exit1	IDM_Exit1

5-3 訊息映射表與回應函數的建立

訊息映射表的建立

對於滑鼠訊息與視窗關閉訊息的回應，將由 MyFrame 類別處理。因此，在 MyFrame 類別的定義裡，將宣告該類別擁有訊息映射表，並在類別定義外，建立 MyFrame 類別所接受訊息的映射項目。

```
'摘自 Message\Message.cpp 檔
8       class MyFrame : public CFrameWnd
9       {
…           …
44          DECLARE_MESSAGE_MAP()    //宣告訊息映射表
45      };
```

```
46
47      BEGIN_MESSAGE_MAP(MyFrame, CFrameWnd)
48                              //建立 MyFrame 類別的訊息映射表
49          ON_COMMAND(ID_Exit1, OnExit) //回應 ID_EXIT1 訊息
50          ON_WM_LBUTTONDOWN() //回應滑鼠左鍵被按下
51          ON_WM_MOUSEMOVE()    //回應滑鼠游標移動
52          ON_WM_LBUTTONUP()    //回應滑鼠左鍵被放開
53      END_MESSAGE_MAP()
```

訊息的回應函數

以下敘述為程式範例的回應函數。

```
'摘自 Message\Message.cpp 檔
8       class MyFrame : public CFrameWnd
9       {
…           …
23          afx_msg void OnExit() //ID_EXIT 的回應函數
…           { … }
29          afx_msg void OnLButtonDown(UINT nFlags, CPoint point)
30          { SetCapture(); }
            //當滑鼠左鍵按下後的回應函數，取得滑鼠訊息接收權
31
32          afx_msg void OnMouseMove(UINT nFlags, CPoint point)
…           { … }
40
41              afx_msg void OnLButtonUp(UINT nFlags, CPoint point)
42              { ReleaseCapture(); }
                //當滑鼠左鍵放開後的回應函數，釋放滑鼠訊息接收權
43
44              DECLARE_MESSAGE_MAP() //宣告訊息映射表
45      };
```

滑鼠訊息傳入的參數

產生滑鼠訊息時，將一併把滑鼠產生此訊息的幾個特殊按鍵狀態利用 nFlags 傳入回應函數，而滑鼠游標的座標，則經由 point 參數傳入。下表是 nFlags 參數傳入旗標的意義。

旗標	說明
MK_CONTROL	按住鍵盤的 Ctrl 鍵
MK_MBUTTON	滑鼠的中間按鍵被按下
MK_RBUTTON	滑鼠的右邊按鍵被按下
MK_SHIFT	按住鍵盤的 ⇧Shift 鍵

預設的命令訊息

您有沒發現一件奇怪的事，在訊息映射表中，並沒有看到回應 File 功能表 Exit 選項的項目。但點選該選項時，卻可關閉視窗。這是因為 MFC 已經為 ID_APP_EXIT 預設了回應函數，而 ID_APP_EXIT 就是 5-1 節提及 MFC 預設的**標準命令識別子**（Standard Command IDs）。

5-4 利用滑鼠繪圖

如何繪圖以及裝置內文的建立

如何在視窗的工作區繪圖呢？第一步，必須為這個視窗的工作區建立**畫布**物件，或稱**裝置內文**物件（Device Context，說明於 12-1-2 節）。然後，再呼叫畫布物件提供的繪圖函數，即可在畫布上繪圖。

Message 程式範例裡，將利用 CDC::SetPixel()函數在工具區繪出滑鼠的軌跡。對於繪圖的基本觀念，將於第 12 章詳細說明。

```
CClientDC aDC(this); //建立一個畫布物件
aDC.SetPixel(point, RGB(255, 0 ,0));//利用 SetPixel 在畫布上點出紅點
```

回應滑鼠訊息並繪出滑鼠軌跡點

前面曾說到，Message 程式範例對於滑鼠訊息的回應，是讓滑鼠在視窗的工作區中留下移動軌跡。整個回應過程如下：

一、按下滑鼠左鍵

回應滑鼠左鍵被按下的 OnLButtonDown()函數，將呼叫 SetCapture()函數，設定目前執行的視窗應用程式，擁有接收滑鼠所發出訊息的接收權，即使滑鼠的位置不在該視窗之上。若未呼叫 SetCapture()函數，則只有當滑鼠位於視窗之上時，視窗才會收到滑鼠訊息。

```
'摘自 Message\Message.cpp 檔
29      afx_msg void OnLButtonDown(UINT nFlags, CPoint point)
30      { SetCapture(); }    //當滑鼠左鍵按下後的回應函數，取得滑鼠訊息接收權
```

二、移動滑鼠

移動滑鼠時，作業系統將不斷傳出滑鼠的移動訊息，OnMouseMove()函數也將不斷被呼叫。該函數將先利用 GetCapture()函數檢查滑鼠游標是否在這個應用程式的視窗上（第 34 行），若是，則宣告一個畫布（第 36 行），並將隨著滑鼠移動訊息傳入的座標點(point)，利用 CDC::SetPixel()函數在畫布上將該點繪成紅色（RGB(255, 0 ,0)）。

```
'摘自 Message\Message.cpp 檔
32      afx_msg void OnMouseMove(UINT nFlags, CPoint point)
33      {           //當滑鼠移動時的回應函數
34          if (this == GetCapture())     //判斷滑鼠游標是否在視窗之上
35          {
36              CClientDC aDC(this);  //建立一個畫布
37              aDC.SetPixel(point, RGB(255, 0 ,0));
                //將畫布上的點設爲紅色
38          }
39      }
```

三、放開滑鼠左鍵

當放開滑鼠左鍵時，則將釋放視窗程式接收滑鼠訊息的權力，並交還給作業系統。

```
'摘自 Message\Message.cpp 檔
41        afx_msg void OnLButtonUp(UINT nFlags, CPoint point)
42        { ReleaseCapture(); }
          //當滑鼠左鍵放開後的回應函數，釋放滑鼠訊息接收權
```

由以上說明將可以瞭解到滑鼠訊息會一直不斷傳給視窗，而滑鼠訊息的回應函數，也將不斷被呼叫。

有關滑鼠訊息的相關函數

CWnd * CWnd::SetCapture( )

□ **函數說明**

設定目前執行中的視窗程式取得滑鼠訊息接收權。若成功則傳回之前取得滑鼠訊息接收權的視窗物件指標。在此次設定滑鼠訊息接收權之前，若未設定取得滑鼠訊息接收權的視窗物件，則傳回 NULL 值。

CWnd * CWnd::GetCapture()

□ **函數說明**

傳回目前取得滑鼠訊息接收權之視窗的指標。若目前執行中的視窗未取得滑鼠訊息接收權則傳回 NULL。

BOOL ReleaseCapture()

□ **函數說明**

此函數將釋放目前視窗所取得的滑鼠訊息接收權，若成功則傳回非零值。

將滑鼠軌跡繪製於畫布上的函數說明

```
COLORREF CDC::SetPixel( int x, int y, COLORREF crColor )
COLORREF CDC::SetPixel( POINT point, COLORREF crColor )
```

□ **函數說明**

在畫布的(x,y)座標上畫出一個顏色為 crColor 的點。若繪製成功則傳回繪於畫布上的顏色值（資料型態為 COLORREF）。若失敗，則傳回-1。

◆ **參數說明**

✓ int x, int y, POINT point
代表欲繪製點的(x,y)座標，或以 POINT 資料型態傳入欲繪製的點座標。

✓ COLORREF crColor
設定繪製顏色。

5-5 訊息方塊的使用與視窗的破壞

訊息方塊的使用

當按下視窗框架中 File 功能表的 Exit 選項時，將執行視窗的關閉。在視窗關閉前，將蹦現一個訊息方塊，告訴您點選 Exit1 選項。

以下程式碼為回應 Exit 選項被點選動作的 OnExit()函數。

```
'摘自 Message\Message.cpp 檔
23      afx_msg void OnExit()  //ID_EXIT1 的回應函數
24      {
25          MessageBox("Exit1");
26          DestroyWindow(); //破壞視窗
27      }
```

訊息方塊的產生很簡單，只要利用 CWnd::MessageBox()函數即可，以下將介紹該函數。

```
int CWnd::MessageBox( LPCTSTR lpszText,
                      LPCTSTR lpszCaption = NULL, UINT nType = MB_OK )
```

□ 函數說明

建立訊息方塊，若沒有足夠的記憶體空間，則傳回 0。若成功建立，則傳回使用者在該訊息方塊中按下按鈕所傳出的訊息。所有的傳回值列於下表。

傳回的常數	說明
0	無法建立訊息方塊
IDABORT	使用者按下了訊息方塊中的放棄（Abort）按鈕。
IDCANCEL	使用者按下了訊息方塊中的取消（CANCEL）按鈕。
IDIGNORE	使用者按下了訊息方塊中的忽略（IGNORE）按鈕。
IDNO	使用者按下了訊息方塊中的否（NO）按鈕。
IDOK	使用者按下了訊息方塊中的確定（OK）按鈕。
IDRETRY	使用者按下了訊息方塊中的重試（RETRY）按鈕。
IDYES	使用者按下了訊息方塊中的是（YES）按鈕。

◆ 參數說明

✓ LPCTSTR lpszText
欲顯示在訊息方塊中的文字。

✓ LPCTSTR lpszCaption = NULL
欲顯示在訊息方塊標題列中的文字。

✓ UINT nType = MB_OK

設定訊息方塊的樣式。可由下表中的四類常數組成，運用兩個以上常數時（必須為不同類的旗標），可使用『|』運算子連接各旗標。

旗　標	說　明
訊息方塊的形式	
MB_ABORTRETRYIGNORE	訊息方塊包含放棄（Abort）、重試（Retry）、忽略（Ignore）三種按鈕。
MB_OK	訊息方塊僅包含確定（OK）按鈕。
MB_OKCANCEL	訊息方塊包含確定（OK）與取消（Cnacel）兩個按鈕。
MB_RETRYCANCEL	訊息方塊包含重試（Retry）與取消（Cnacel）兩個按鈕。
MB_YESNO	訊息方塊包含是（Yes）與否（No）兩個按鈕。
MB_YESNOCANCEL	訊息方塊包含是（Yes）、否（No）與取消（Cnacel）三個按鈕。
訊息方塊的執行模式（所謂的執行模式您可參考 21-1-1 節的說明）	
MB_APPLMODAL	當訊息方塊產生時，將會中斷產生此訊息方塊之視窗的執行，直到關閉訊息方塊為止。當訊息方塊均未被設定為 MB_SYSTEMMODAL 與 MB_TASKMODAL 時，MB_APPLMODAL 將為訊息方塊的預設值。
MB_SYSTEMMODAL	當此訊息方塊產生時，所有執行中的應用程式都將會被暫停，直到使用者回應此訊息方塊為止。
MB_TASKMODAL	與 MB_APPLMODAL 類似，但是此旗標並不是提供給 MFC 的類別所使用。而是提供給無法取得視窗標頭的程式所使用。
顯示於訊息方塊中的圖示	
MB_ICONEXCLAMATION	在訊息方塊顯示的文字旁顯示 ⚠ 圖示。
MB_ICONINFORMATION	在訊息方塊顯示的文字旁顯示 ℹ 圖示。
MB_ICONQUESTION	在訊息方塊顯示的文字旁顯示 ❓ 圖示。
MB_ICONSTOP	在訊息方塊顯示的文字旁顯示 ⛔ 圖示。
訊息方塊中預設的按鈕	
MB_DEFBUTTON1	訊息方塊中的第一個按鈕被設定為預設按下的按鈕。如果 MB_DEFBUTTON2 或 MB_DEFBUTTON3 未被設定則此為預設值。
MB_DEFBUTTON2	訊息方塊中的第二個按鈕被設定為預設按下的按鈕。
MB_DEFBUTTON3	訊息方塊中的第三個按鈕被設定為預設按下的按鈕。

破壞視窗

按下訊息方塊的 確定 按鈕後，程式將繼續執行 MyFrame::OnExit() 函數中，破壞視窗的 CWnd::DestoryWindow()函數（第 26 行），結束視窗程式的執行。

```
'摘自 Message\Message.cpp 檔
23      afx_msg void OnExit()  //ID_EXIT1 的回應函數
24      {
25          MessageBox("Exit1");
26          DestroyWindow(); //破壞視窗
27      }
```

virtual BOOL CWnd::DestroyWindow( )

□ **函數說明**

此函數將消滅視窗物件。如果成功則傳回非零值，失敗則傳回 0。