MFC視窗程式基礎

MFC?

- MFC 幫助我們把這些浩繁的API，利用物件導向的原理，邏輯地組織起來，使它們具備抽象化、封裝化、繼承性、多型性、模組化的性質。
- C++ 是一個複雜的語言，AFX 小組（1989 年微軟公司所成立Application Framework 技術團隊）預期MFC 的使用者不可能人人皆爲C++ 專家，所以他們並沒有採用所有的C++ 高階性質（例如多重繼承）。許多「麻煩」但「幾乎一成不變」的Windows 程式動作都被隱藏在MFC 類別之中， 例如*WinMain* 、*RegisterClass*、Window Procedure 等等等。

MFC 程式運作過程

- 以傳統的C/SDK 撰寫Windows 程式，最大的好處是可以清楚看見整個程式的來龍去脈和訊息動向，然而這些重要的動線在MFC 應用程式中卻隱晦不明，因爲它們被Application Framework 包起來了。
- 了解釋MFC 應用程式的長像及從MFC 原始碼中檢驗出一個Windows 程式原本該有的程式進入點（*WinMain*）、視窗類別註冊（*RegisterClass*）、視窗產生（*CreateWindow*）、訊息迴路（Message Loop）、視窗函式（Window Procedure）等等動作，以了解一個MFC 程式的誕生與結束，以及生命過程。

Hello MFC

```
1.  //File name：HelloMFC.cpp
2.  //Function: My first window program
3.  #include <afxwin.h>                          //載入afxwin標頭檔

4.  class MyApp : public CWinApp                  //繼承CWinApp
5.  {
6.  public:
7.      BOOL InitInstance()                       //程式進入點
8.      {
9.          CFrameWnd *Frame = new CFrameWnd();   //建立CFrameWnd物件
10.         m_pMainWnd = Frame;                    //將m_pMainWnd設定爲Frame

11.         Frame->Create(NULL,"Hello MFC");       //建立視窗
12.         Frame->ShowWindow(SW_SHOW);

13.         return true;
14.     }
15. };

16. MyApp a_app;                                  //建立應用程式物件
```

- SDK 程式只要含入windows.h ，因所有API 的函式宣告、訊息定義、常數定義、巨集定義、都在windows.h 檔中。
- 利用MFC撰寫視窗應用程式時，需要聯結一個所謂的MFC 函式庫，或稱爲AFX 函式庫(它是MFC這個application framework 的本體)。可以靜態聯結之，也可以動態聯結之。故撰寫視窗時，必須載入afxwin.h，該標頭檔中定義了所有的MFC類別。

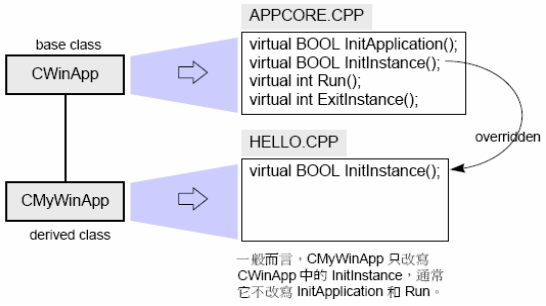
- SDK視窗 程式的寫法，其主體在於*WinMain* 和*WndProc*，而這兩個部份其實都有相當程度的不變性。MFC 就把有著相當固定行爲之*WinMain* 內部動作包裝在*CWinApp* 中，把有著相當固定行爲之*WndProc* 內部動作包裝在*CFrameWnd* 中。
- 也就是說：
  - *CWinApp* 代表程式本體
  - *CFrameWnd* 代表一個主框視窗（Frame Window）

- ***a_app*** 就是 Hello MFC 程式的 application object，每一個 MFC 應用程式都只能有一個。當你執行 Hello MFC，這個全域物件產生，於是建構式執行起來。由於我們並沒有定義 *MyApp* 建構式，於是呼叫父類別的建構式，所以繼承自*CWinApp* 之中的成員變數將因爲 ***a_app*** 這個全域物件的誕生而獲得配置與初值。
- *a_app* 配置完成後，*WinMain* 登場。我們並未撰寫 *WinMain* 程式碼，這是 MFC 早已準備好並由聯結器直接加到應用程式碼中的,並放在建構子中去執行.

```
int AFXAPI AfxWinMain (...)
{
        ....
        CWinApp* pApp = AfxGetApp();
        AfxWinInit(...);
        pApp->InitApplication();
        pApp->InitInstance();
        nReturnCode = pApp->Run();
          ....
}
```

```
class MyApp : public CWinApp      //繼承CWinApp
{
public:
        BOOL InitInstance() //程式進入點
        {
          CFrameWnd *Frame = new CFrameWnd();
            //建立CFrameWnd物件

          m_pMainWnd = Frame;
            //將m_pMainWnd設定爲Frame

          Frame->Create(NULL,"Hello MFC");
            //建立視窗

          Frame->ShowWindow(SW_SHOW);

          return true;
        }
};

MyApp a_app;
```

應用程式一定要改寫虛擬函式*InitInstance*，因爲它在*CWinApp* 中只是個空函式，沒有任何預設動作。



- *MyApp*::*InitInstance* 一開始 *new* 了一個 *CMyFrameWnd* 物件的空間給繼承至父類別CWinApp的成員指標變數 m_pMainWnd ，然後呼叫Create()來創造視窗。
- *Create* 是*CFrameWnd* 的成員函式，它將產生一個視窗。而且是由 *RegisterClass* 註冊的一份資料結構所定義的視窗類別。

```
BOOL Create( LPCTSTR lpszClassName,
             LPCTSTR lpszWindowName,
             DWORD dwStyle = WS_OVERLAPPEDWINDOW,
             const RECT& rect = rectDefault,
             CWnd* pParentWnd = NULL,
             LPCTSTR lpszMenuName = NULL,
             DWORD dwExStyle = 0,
             CCreateContext* pContext = NULL );
```

- 八個參數中的後六個參數都有預設值，只有前兩個參數必須指定。第一個參數*lpszClassName* 指定 *WNDCLASS* 視窗類別，若設爲 *NULL* 代表要以 MFC 內建的視窗類別產生一個標準的外框視窗。*Create* 函式在產生視窗之前會引發視窗類別的註冊動作
- 第二個參數 *lpszWindowName* 指定視窗標題，本例指定 “Hello MFC”。

真實中的WinMain()大概的樣子

```
int CALLBACK WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
LPSTR lpCmdLine, int nCmdShow)
{
    if (!hPrevInstance)
    if (!InitApplication(hInstance))
            return (FALSE);
    if (!InitInstance(hInstance, nCmdShow))
            return (FALSE);
    ...
}
//----------------------------------------------
BOOL InitApplication(HINSTANCE hInstance)
{
    WNDCLASS wc;
    ...
    return (RegisterClass(&wc));
}
//----------------------------------------------
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    hWnd = CreateWindow(...);
    ...
}
```

- 當視窗誕生出來後， 於是呼叫 *ShowWindow* 函式令視窗顯示出來，程式進行到這裡，視窗類別註冊好了，視窗誕生並顯示出來了。
- 接著呼叫*UpdateWindow* 函式令Hello 程式送出*WM_PAINT* 訊息。放入訊息佇列中，等待被處理。
- 接著，執行的腳步到達*pApp->Run(*相當於呼叫：**MyApp::Run();)** ，而 *MyApp* 繼承自 *CWinApp*，而 *Run* 又是 *CWinApp* 的一個虛擬函式。我們並沒有改寫它（大部份情況下不需改寫它），所以上述動作相當於呼叫：**CWinApp::Run();**
- 獲得的訊息如何交給適當的常式去處理呢？SDK 程式的作法是呼叫 *DispatchMessage*，把訊息丟給視窗函式；MFC 也是如此。但我們並未在Hello MFC程式中提供任何視窗函式，是的，視窗函式事實上由 MFC 提供。
- *WinMain* 已由MFC 提供，視窗類別已由MFC 註冊完成、連視窗函式也都由MFC提供。那麼我們（程式員）如何爲特定的訊息設計特定的處理常式？MFC 應用程式對訊息的辨識與判別是採用所謂的「Message Map 機制」