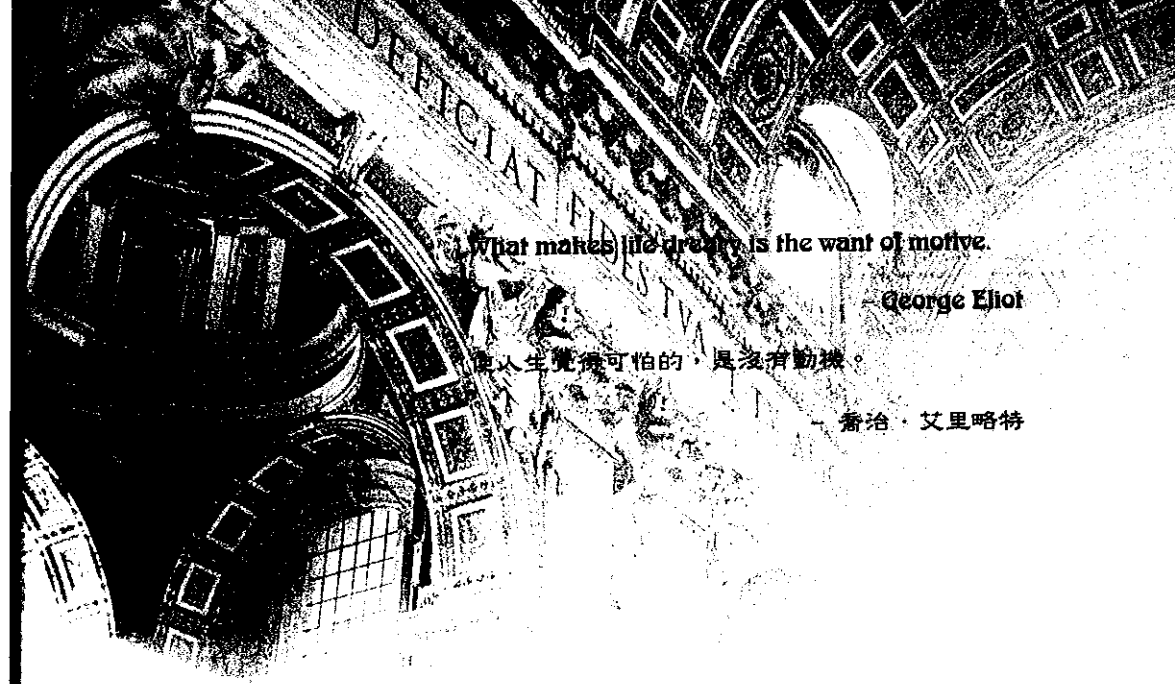


What makes life dreary is the want of motive.

– George Eliot

使人生覺得可怕的，是沒有動機。

– 喬治．艾里略特

# 6

# 視窗應用程式架構

## 本章導讀

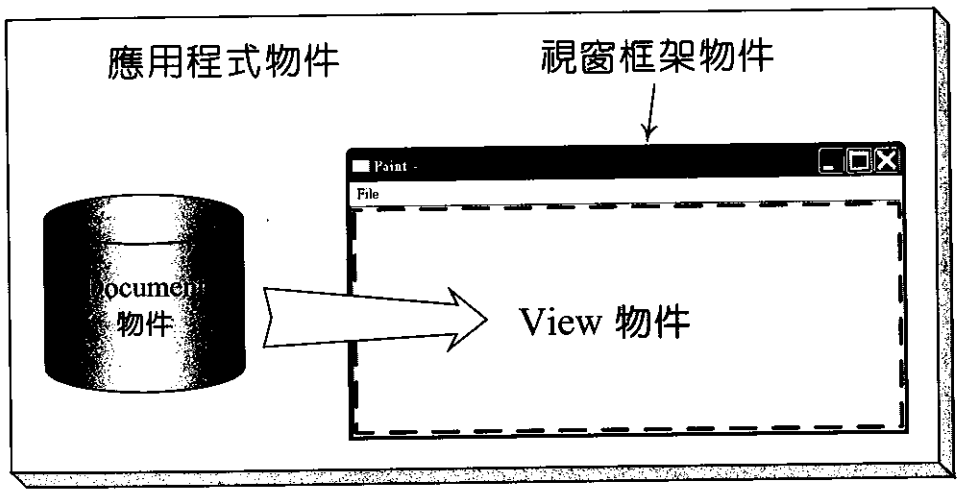
前面章節所建立的程式範例，都不是完整的應用程式。一個完整的應用程式架構應該具備的類別，除了前面幾章提到的應用程式類別（App 類別）、視窗框架類別（Frame 類別）外，還要有資料處理的文件類別（Document 類別），以及處理資料顯示的瀏覽類別（View 類別）。這個完整的視窗應用程式架構，稱之為 Doc/View 架構（文件/瀏覽架構）。此架構將視窗應用程式必須處理的工作，區分為**介面建立**、**資料儲存**與**資料顯示**，並分別交給不同類別處理。以各類別分工合作的方式，完成複雜視窗應用程式的建立。這一章將先以 Doc_View 程式範例，讓您瞭解如何建立 Doc/View 視窗應用程式架構，然後再透過 repaint 程式範例，告訴您如何利用 Doc/View 架構處理資料儲存與顯示的工作。

## 各節標題

# 6-1 什麼是 Doc/View 架構

## 完整的應用程式基本架構

相較於執行於命令提示字元模式的程式，視窗程式除了處理資料外，還要讓資料能顯示在視窗介面中。但在視窗介面下，資料的顯示並不是簡單的 cin 或 cout 就能夠解決。因此，必須建立 Doc/View（文件/瀏覽）架構，以便管理視窗介面下資料的儲存與顯示。分工上，Document 物件用於管理視窗程式的**資料儲存**，View 物件則負責將 Document 儲存的資料正確地**顯示**在視窗中。所以，一個完整的視窗應用程式，必須具備 App 類別（應用程式類別）、Frame 類別（視窗框架類別）、Document 類別（文件類別）、View 類別（瀏覽類別）。各類別間的關係如下圖。




從上圖可以看出 App 物件代表應用程式物件，建立 App 物件時，將建立 Frame 物件以產生視窗介面。Document 物件則用於運作、儲存資料，View 物件則用於將 Document 物件中的資料顯示於視窗框架的客戶區。

## 文件樣版類別

從前面的說明可瞭解到一個視窗應用程式應具備的基本物件，對於整個**資料**與**視窗介面**的運作、顯示，Frame 物件、Document 物件、View 物件都是運作核心。從資料的角度來看，Document 物件與 View 物件其實是一體的，只是運用 Doc/View 架構將資料的處理與顯示工作分開。

從視窗介面的角度來看，Frame 物件與 View 物件，也是一體的，只是將顯示資料的部份與視窗介面元件分開。從以上說明可以看出，Frame 物件、Document 物件、View 物件其實是三位一體的。撰寫程式時，建立這三者的關係，將透過 MFC 所提供的**文件樣版類別**完成。

文件樣版類別共有兩種，一是**單文件**樣版類別（6-2-3 節），另一是**多文件**樣版類別（17-4 節），兩者的差別在於視窗可以同時開啟的檔案個數。單文件樣版類別建立出來的應用程式一次只能開啟**一個**檔案，如 Windows 提供的 Notepad 就是這類應用程式。反之，多文件樣版類別建立出來的應用程式，則一次可以開啟**數個**檔案，像 Word 即為多文件應用程式。

# 6-2 以 Document/View 為架構的視窗應用程式

## 6-2-1 Doc_View 程式範例

這一節的 Doc_View 程式範例，將示範如何建立以 Document/View 為架構的視窗程式，讓各位清楚地看出 Document/View 視窗應用程式的基本架構。介紹重點如下：

1. Document/View 視窗應用程式的架構與建立步驟（6-2-2 節）
2. 文件樣版的應用（6-2-3 節）
3. CView 類別與 CDocument 類別的應用（6-2-4 節）

Doc_View 程式範例的程式碼如下：

### Doc_View 程式範例 - Document/View 的建立

檔案位置：Doc_View\Doc_View.cpp

```
1    /*
2    範例檔名：Doc_View.cpp
3    程式開發：郭尚君
4    */
5    #include <afxwin.h>
6    #include "Doc_View.h"
7
8    class MyDocument : public CDocument
9    {
10      DECLARE_DYNCREATE(MyDocument) //宣告 run-time 類別
11   };
12
13   IMPLEMENT_DYNCREATE(MyDocument, CDocument)
14   //宣告 MyDocument 爲 run-time 類別
```

```
15
16    class MyView : public CView
17    {
18    public:
19       void OnDraw(CDC * aDC) //必須覆寫的虛擬函數
20       { }
21
22       DECLARE_DYNCREATE(MyView) //宣告 run-time 類別
23    };
24
25    IMPLEMENT_DYNCREATE(MyView, CView)
26          //宣告 MyView 爲 run-time 類別
27
28    class MyFrame : public CFrameWnd
29    {
30       DECLARE_DYNCREATE(MyFrame) //宣告 run-time 類別
31    };
32
33    IMPLEMENT_DYNCREATE(MyFrame, CFrameWnd)
34    //宣告 MyFrame 爲 run-time 類別
35
36    class MyApp : public CWinApp
37    {
38    public:
39       BOOL InitInstance() <-- 程式進入點
40       {
41          CDocument *doc; //宣告指向文件的指標
42          CSingleDocTemplate* DocTemplate;
          //宣告指向單文件樣版物件的指標
43          DocTemplate=new CSingleDocTemplate( //建立具有單文件樣版物件
44             IDR_MENU1, //用於單文件框架之資源的識別子
45             RUNTIME_CLASS(MyDocument), //單文件視窗的 Document
46             RUNTIME_CLASS(MyFrame), //單文件視窗的視窗框架
47             RUNTIME_CLASS(MyView)); //單文件視窗的 View
48
49          AddDocTemplate(DocTemplate); //將單文件樣版物件設定給 MyApp
50          doc = DocTemplate->CreateNewDocument(); //建立新的文件
51
52          m_pMainWnd = DocTemplate->CreateNewFrame( doc, NULL );
53          //建立一個視窗框架
```

```
54          DocTemplate->InitialUpdateFrame(
                                        (CFrameWnd*)m_pMainWnd, doc );
55          //起始化視窗框架物件，並連結 View 物件
56          m_pMainWnd->ShowWindow ( SW_SHOW); //顯示視窗
57
58          return true;
59       }
60    } a_app; //建立應用程式物件
```

## ✤ 使用資源

一、功能表資源：以下是程式範例中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 |

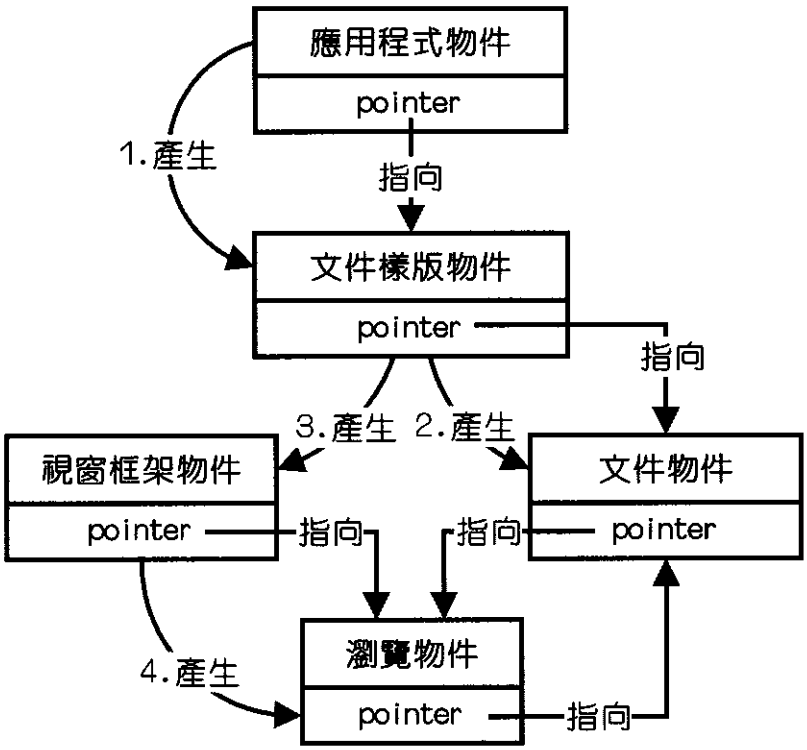
三、字串資源：

| 識別子 | 字串內容 |
|---|---|
| IDR_MENU1 | Paint\npaint\npaint\nPaint File (*.pnt)\n.pnt\n |

## 6-2-2 Doc/View 的架構與建立步驟

### SDI 視窗程式的架構

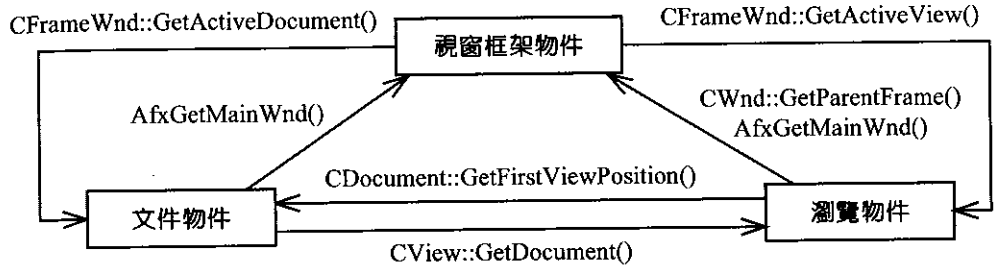
整個 Doc_View 程式範例中，各種物件的關係如下圖。

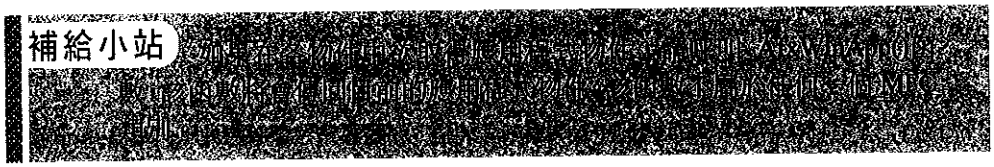



從上圖可以看出應用程式建立文件樣版物件後，將由該物件分別產生文件物件與視窗框架物件，最後再由視窗框架物件產生瀏覽物件。詳細的建立過程說明與程式碼內容，請參考以下各節的說明。

## 如何取得程式中其他物件的指標

在程式中，常常需要取得程式中其他物件的指標，以便操作該物件的成員函數。下圖說明了單文件視窗程式中，應用程式內各物件如何利用其成員函數，取得其他物件的指標。




### Document/View 的建立步驟

建立一個以 Document/View 為架構的應用程式，其步驟如下：

STEP 1、建立一個文件樣版類別的物件（第 43 行到第 47 行），並利用文件樣版物件，將已經建立好的視窗資源、Document 類別、View 類別與 Frame 類別結合在一起（本程式範例使用的是**單文件樣版類別** -CSignleDocTemplate 類別），並將文件樣版物件設定給應用程式物件。（第 49 行）

STEP 2、利用這個文件樣版物件，產生 Document 物件。（第 50 行）

STEP 3、建立視窗框架（第 52 行）

STEP 4、起始化視窗框架物件，並連結起始 View 物件（第 54 行）。最後顯示視窗框架。（第 56 行）

以下為 Doc_View 程式範例執行這整個過程的 MyApp::InitInstance() 函數。

```
'摘自 Doc_View\Doc_View.cpp 檔
39      BOOL InitInstance()  <-- 程式進入點
40      {
41          CDocument *doc;  //宣告指向文件的指標
42          CSingleDocTemplate* DocTemplate;
            //宣告指向單文件樣版物件的指標
43          DocTemplate=new CSingleDocTemplate(//建立具有單文件樣版物件
44              IDR_MENU1, //用於單文件框架之資源的識別子
45              RUNTIME_CLASS(MyDocument), //單文件視窗的 Document
46              RUNTIME_CLASS(MyFrame), //單文件視窗的視窗框架
47              RUNTIME_CLASS(MyView)); //單文件視窗的 View
48
49          AddDocTemplate(DocTemplate); //將單文件樣版物件設定給 MyApp
50          doc = DocTemplate->CreateNewDocument(); //建立新的文件
51
52          m_pMainWnd = DocTemplate->CreateNewFrame( doc, NULL );
53          //建立一個視窗框架
54          DocTemplate->InitialUpdateFrame(
                              (CFrameWnd*)m_pMainWnd, doc );
55          //起始化視窗框架
56          m_pMainWnd->ShowWindow(SW_SHOW); //顯示視窗
57
58          return true;
59      }
```

## 與建立單文件視窗應用程式的相關函數說明

```
CSingleDocTemplate::CSingleDocTemplate(
    UINT nIDResource, CRuntimeClass* pDocClass,
    CRuntimeClass* pFrameClass, CRuntimeClass* pViewClass )
```

### □ 函數說明

單文件樣版的建構子，更進一步說明請參考 6-2-3 節。

### ◆ 參數說明

- ✓ UINT nIDResource
  用於建立單文件視窗應用程式的資源。
- ✓ CRuntimeClass* pDocClass
  用於建立單文件視窗應用程式的 Document 物件的指標(此物件為 Run-Time 類別）。
- ✓ CRuntimeClass* pFrameClass
  用於建立單文件視窗應用程式的視窗框架物件的指標（此物件為 Run-Time 類別）。
- ✓ CRuntimeClass* pViewClass
  用於建立單文件視窗應用程式的 View 物件的指標（此物件為 Run-Time 類別）。

```
void CWinApp::AddDocTemplate( CDocTemplate* pTemplate)
```

### □ 函數說明

將完成建立的文件樣版物件設定給應用程式物件。

### ◆ 參數說明

- ✓ CDocTemplate* pTemplate
  欲設定給應用程式物件的文件樣版物件。

```
virtual CDocument* CDocTemplate::CreateNewDocument( )
```

### □ 函數說明

建立單文件視窗應用程式的 Document 物件。如果成功則傳回新建立的 Document 物件指標，若發生錯誤則傳回 NULL 值。

```
virtual CFrameWnd* CDocTemplate::CreateNewFrame(
                                CDocument* pDoc, CFrameWnd* pOther )
```

☐ 函數說明

建立單文件視窗應用程式的視窗框架物件。如果成功則傳回新建立的視窗框架物件指標，若發生錯誤則傳回 NULL 值。

◆ 參數說明

✓ CDocument* pDoc
建立視窗框架物件所連結 Document 物件的指標，若此視窗框架物件未使用則設定為 NULL。

✓ CFrameWnd* pOther
產生新視窗框架物件的視窗框架物件，若未設定則為 NULL。

```
virtual void CDocTemplate::InitialUpdateFrame (
                CFrameWnd* pFrame, CDocument* pDoc,
                BOOL bMakeVisible = TRUE )
```

☐ 函數說明

當利用 CDocTemplate::CreateNewFrame()完成單文件應用程式所使用視窗框架物件的建立，接著，將呼叫此函數起始化該視窗框架物件。若該視窗框架物件尚未連結有效的 View 物件，則執行此函數時，將完成一個有效的 View 物件連結。

◆ 參數說明

✓ CFrameWnd* pFrame
需要起始化視窗框架物件的指標。

✓ CDocument* pDoc
欲起始化視窗框架物件所連結的 Document 物件，若未連結則傳入 NULL。

✓ BOOL bMakeVisible = TRUE
欲起始化的視窗框架物件是否被設定為顯示。

## 6-2-3 單文件樣版類別的應用

6-1 節清楚地指出，文件樣版類別是用於管理與連結視窗框架類別、View 類別、Document 類別。這一節將說明如何使用**單文件樣版類別**，至於多文件樣版類別的使用，將說明於 17-4 節。

### CSingleDocTemplate 的使用

當建立單文件樣版物件時，需將建立這個文件樣版的相關資源與類別，一併傳入單文件樣版類別的建構子裡，以便將它們連結在一起。

```
'摘自 Doc_View\Doc_View.cpp 檔
43        DocTemplate=new CSingleDocTemplate(//建立具有單文件樣版物件
44            IDR_MENU1, //用於建立單文件視窗框架之資源的識別子
45            RUNTIME_CLASS(MyDocument), //單文件視窗的 Document
46            RUNTIME_CLASS(MyFrame), //單文件視窗的視窗框架
47            RUNTIME_CLASS(MyView)); //單文件視窗的 View
```

傳入單文件樣版類別建構子的第一個參數是建立單文件應用程式所使用資源的識別子。當建立功能表、工具列...等資源時，必須將這些資源的**識別子**定義成同一個。而建立文件樣版類別時，將該**識別子**傳入已載入資源檔中，用於建立該視窗程式的資源。這些資源裡，比較需要說明的是**字串資源**，請參考以下介紹。

### Doc/View 架構使用之字串資源

Doc_View 程式範例將程式所需要使用的字串資源定義如下（字串表的定義方法請參考 7-6 節的說明）：

| ID | Value | Caption |
|---|---|---|
| IDR_MENU1 | 101 | Paint\npaint\npaint\nPaint File (*.pnt)\n.pnt\n |

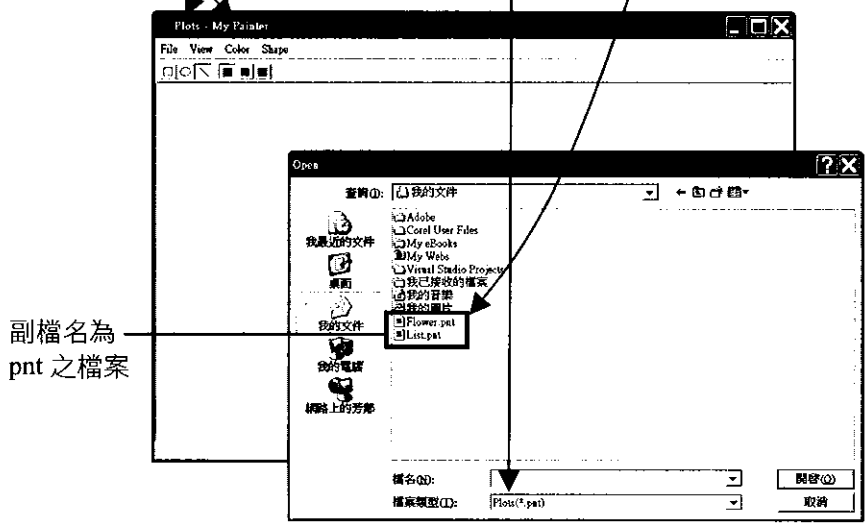
其中 IDR_MENU1 字串資源的 Caption 屬性，被『\n』分隔成 5 個字串，實際上應該是 7 個，在本程式範例中，僅用到前面 5 個，這 7 個子字串的意義說明如下表。

| 第 N 個子字串 | 字串名稱 | 說明 |
|---|---|---|
| 0 | 視窗標題字串 | 顯示在視窗標題區的字串 |
| 1 | 預設檔案名稱 | 開啟新檔時，預設使用的檔案名稱。 |
| 2 | 檔案類型名稱 | 若您的程式可開啟多種檔案時，需指定此字串，用於開啟檔案前，顯示於開啟對話盒內使用。 |
| 3 | 副檔名說明 | 副檔案的說明，將出現在開啟檔案對話盒中的『檔案類型』欄。 |
| 4 | 副檔名 | 可開啟檔案的副檔名，符合此副檔名者，才能出現於檔案對話盒內。 |
| 5 | 檔案類型代號 | 此代號用於作業系統的登錄資料庫中，登錄程式檔案類型之用。當您登錄程式使用的檔案後，系統即可利用此代號識別開啟檔案所使用的程式。若未註冊程式使用的檔案類型，這對於程式以滑鼠拖曳方式開啟檔案的功能將受到影響。 |
| 6 | 檔案類型名稱 | 儲存在檔案登錄資料庫中的檔案類型名稱。 |

註：對於第 5、6 個子字串的說明，請參考 17-4 節的說明。

由於 Doc_View 程式範例的目的僅在說明 Doc/View 應用程式的基本架構，所以，字串資源內的子字串許多並未使用。下圖將借用 14-2 節的 painter4 程式範例，說明程式中運用各個子字串的運用位置。

```
My Painter\nPlots\nPlots\nPlots(*.pnt)\n.pnt\n
```




## 宣告 Run-Time 類別

或許您會好奇，為何傳入 CSignleDocTemplate 的類別，在類別名稱前都要加上 RUNITME_CLASS 呢？這是一個有點深度的問題。簡單來說，是為了讓 CSignleDocument 類別在無法得知使用者自訂之視窗框架類別、View 類別、Document 類別的情況下，等到**執行時才決定**運用哪幾個類別建立應用程式。

其實這個動作是可以很輕易地用 C++的 template 觀念解決，只是早期 MFC 並不支援 template 觀念，所以，用了 Run-Time 類別這個旁門左道的方法解決這個問題。欲將某類別宣告為 Run-Time 類別時，必須在類別的定義敘述中，利用以下語法宣告。

```
DECLARE_DYNAMIC( 類別名稱 )
```

然後在該類別的定義敘述外，再利用如下語法進行宣告。

```
IMPLEMENT_DYNAMIC( 類別名稱, 衍生類別名稱 )
```

Doc_View 程式範例因為使用 CSignleDocTemplate 的關係，視窗框架類別、View 類別、Document 類別必須被宣告為 Run-Time 類別。

```
'摘自 Doc_View\Doc_View.cpp 檔
8     class MyDocument : public CDocument
9     {
10       DECLARE_DYNCREATE(MyDocument) //宣告 run-time 類別
11    };
12
13    IMPLEMENT_DYNCREATE(MyDocument, CDocument)
14          //宣告 MyDocument 爲 run-time 類別
15
16    class MyView : public CView
17    {
18    public:
19       void OnDraw(CDC * aDC) //必須覆寫的虛擬函數
20       { }
21
22       DECLARE_DYNCREATE(MyView) //宣告 run-time 類別
23    };
24
25    IMPLEMENT_DYNCREATE(MyView, CView)
26    //宣告 MyView 爲 run-time 類別
27
28    class MyFrame : public CFrameWnd
29    {
30       DECLARE_DYNCREATE(MyFrame) //宣告 run-time 類別
31    };
32
33    IMPLEMENT_DYNCREATE(MyFrame, CFrameWnd)
```

## 6-2-4 CView 類別與 CDocument 類別的使用

### CView 類別的使用

View 類別用於將 Document 類別內的資料顯示在視窗中，而 View 類別的建立，必須繼承於 MFC 的 CView 類別（或者是 CView 類別的衍生類別）。Doc_View 程式範例裡，自訂的 MyView 類別繼承於 CView 類別。必須注意的一件事為繼承 CView 類別時，必須覆寫 OnDraw()虛擬函數，此函數在重繪視窗以及列印時，將會被呼叫。關於此函數的說明請參考 6-3-4 節。

### CDocument 類別的使用

Document 類別用於儲存視窗程式內的資料。而 Document 類別的建立必須以繼承方式，利用 MFC 提供的 CDocument 類別，或其衍生類別。Doc_View 程式範例自訂的 MyDocument 類別即繼承於 CDocument 類別。

# 6-3 Doc/View 架構的應用

## 6-3-1 視窗的重繪

不曉得您有沒發現上一章的 Message 程式範例，有一個很大的缺點。當重繪視窗時，原本存在於視窗的滑鼠軌跡點將會消失（重繪視窗發生於視窗大小改變，或移開重疊於該視窗上之視窗時）。

也就是說，視窗無法在重繪時，將滑鼠軌跡**重新**畫到視窗上。這是為什麼呢？這是因為將滑鼠軌跡畫到畫布上時，並沒有紀錄，這使得視窗重繪時，視窗缺乏重現原有視窗中滑鼠軌跡的資料。

前面的 Doc_View 程式範例，清楚地介紹了 Document/View 架構的建立，這個小節裡，將把 Message 程式範例的架構改為 Document/View 架構，建立一個可以將滑鼠軌跡重繪到視窗上的程式範例，以進一步說明 Document/View 架構的實際應用。

## 6-3-2 repaint 程式範例

這一節裡，將利用這個範例，講解以下的重點：

1. 用 Document/View 機制儲存與顯示資料（第 6-3-3 節）
2. 視窗的重繪（第 6-3-4 節）

以下為 repaint 程式範例的內容。

### repaint 程式範例 - 視窗重繪

檔案位置：repaint\repaint.cpp

```
1    /*
2    範例檔名：repaint.cpp
3    程式開發：郭尚君
4    */
5    #include <afxwin.h>
6    #include "repaint.h"
7    #include <afxtempl.h> //定義樣版類別的標頭檔
8
9    class MyDocument : public CDocument
10   {
11   public:
```

```
12      CArray<CPoint, CPoint &> pArray; //容納滑鼠軌跡點的 Array 容器
13
14      void AddPoint(CPoint p) //將軌跡點加到容器內
15      { pArray.Add(p); }
16
17      CPoint GetPoint(int i) //將軌跡點從容器中取出
18      { return pArray[i]; }
19
20      int GetSize()
21      { return pArray.GetSize(); } //取得容器的大小
22
23      DECLARE_DYNCREATE(MyDocument) //宣告爲 run-time 類別
24      DECLARE_MESSAGE_MAP() //宣告訊息映射表
25  };
26
27  IMPLEMENT_DYNCREATE(MyDocument, CDocument)
28  //建立 run-time 類別
29  BEGIN_MESSAGE_MAP(MyDocument, CDocument)
30  END_MESSAGE_MAP()
31  //建立訊息映射表
32
33  class MyView : public CView
34  {
35  public:
36      void OnDraw(CDC * aDC) //覆寫 OnDraw()虛擬函數
37      {
38          MyDocument *doc = (MyDocument *)GetDocument();
39          //取得目前 Document 物件的指標
40          int num = doc->GetSize();//取得目前儲存的軌跡點點數
41          int i;
42
43          for(i = 0; i < num; ++i)
            //將 Document 中儲存的軌跡點重繪到視窗上
44          {
45              CPoint point = doc->GetPoint(i);
                //取出 doc 物件中儲存的第 i 個滑鼠軌跡點
46              aDC->SetPixel(point, RGB(255, 0 ,0));
                //將滑鼠軌跡點會於畫布上
47          }
48      }
49
```

```
50      afx_msg void OnLButtonDown(UINT, CPoint point)
51      { SetCapture(); } //取得滑鼠訊息的接收權
52
53      afx_msg void OnMouseMove(UINT, CPoint point)
54      {
55         if (this == GetCapture())
56         {
57            CClientDC aDC(this); //建立畫布
58            aDC.SetPixel(point, RGB(255, 0 ,0)); //將點畫在畫布上
59            MyDocument *doc = (MyDocument *)GetDocument();
60            //取得目前 Document 物件的指標
61            doc->AddPoint(point); //將軌跡點加入 Document 物件中
62         }
63      }
64
65      afx_msg void OnLButtonUp(UINT, CPoint point)
66      { ReleaseCapture(); } //釋放滑鼠訊息的接收權
67
68      DECLARE_DYNCREATE(MyView) //宣告爲 run-time 類別
69      DECLARE_MESSAGE_MAP() //宣告訊息映射表
70   };
71
72   IMPLEMENT_DYNCREATE(MyView, CView)
73      //建立 run-time 類別
74   BEGIN_MESSAGE_MAP(MyView, CView)
75      ON_WM_LBUTTONDOWN()
76      ON_WM_MOUSEMOVE()
77      ON_WM_LBUTTONUP()
78   END_MESSAGE_MAP()
79   //建立訊息映射表
80
81   class MyFrame : public CFrameWnd
82   {
83   public:
84      DECLARE_DYNCREATE(MyFrame) //宣告爲 run-time 類別
85      DECLARE_MESSAGE_MAP() //宣告訊息映射表
86   };
87
88   IMPLEMENT_DYNCREATE(MyFrame, CFrameWnd)
89   //建立 run-time 類別
90   BEGIN_MESSAGE_MAP(MyFrame, CFrameWnd)
```

```
91    END_MESSAGE_MAP()
92    //建立訊息映射表
93
94    class MyApp : public CWinApp
95    {
96    public:
97       BOOL InitInstance()  <-- 程式進入點
98       {
99          CDocument *doc;
100         CSingleDocTemplate* DocTemplate;
101         DocTemplate = new CSingleDocTemplate( //單文件樣版類別
102            IDR_MENU1,
103            RUNTIME_CLASS(MyDocument),
104            RUNTIME_CLASS(MyFrame),
105            RUNTIME_CLASS(MyView));
106         AddDocTemplate(DocTemplate); //將文件樣版物件加入應用程式
107         doc = DocTemplate->CreateNewDocument(); //建立新文件
108
109         m_pMainWnd = DocTemplate->CreateNewFrame( doc, NULL );
110         //建立新的視窗框架
111         DocTemplate->InitialUpdateFrame (
                                 (CFrameWnd*)m_pMainWnd, doc );
112         //起始化 View 物件
113         m_pMainWnd->ShowWindow ( SW_SHOW ); //顯示視窗
114
115         return true;
116      }
117 } a_app; //建立應用程式物件
```

# 6-3-3 Doc/View 的運作機制

## Document 的資料儲存

前面提過，Document/View 架構裡，Document 物件負責儲存資料，View 物件負責顯示 Document 物件的資料。repaint 程式範例裡，Document 物件將利用 MFC 所提供的容器類別 – CArray 類別，將滑鼠於視窗中移動時，傳入的軌跡點儲存起來。以下為以 CArray 類別宣告容器物件的語法：

**CArray<儲存的資料型態，讀取儲存資料的回傳值> 物件名稱**

第一個參數用於指定容器所儲存資料的型態，第二個參數則是指定讀取容器內元素的回傳值，通常第二個參數是第一個參數的參考型態。repaint 程式範例裡，Document 物件用於儲存軌跡點的 CArray 容器物件（pArray），將被宣告為 Document 物件的屬性（第 12 行）。

```
'摘自 repaint\repaint.cpp 檔
12        CArray<CPoint, CPoint &> pArray;//容納滑鼠軌跡點的 Array 容器
```

操作 Dcoument 物件內 CArray 容器（pArray 屬性）的操作介面為 AddPoint()、GetPoint()、GetSize()這三個成員函數，各函數的規格說明如下表：

| 函數原型 | 說明 | 傳入參數 | 回傳值 |
|---|---|---|---|
| void AddPoint(cpoint p) | 將軌跡點加入容器 | 欲加入容器的軌跡點 | 無 |
| CPoint GetPoint (int i) | 從容器中取得軌跡點 | 欲取得軌跡點的索引值（從 0 開始） | 傳回容器內第 i 個元素 |
| int GetSize() | 取得軌跡點的個數 | 無 | 傳回容器內儲存軌跡點的個數（元素個數） |

這個三個函數均運用到 CArray 類別提供的成員函數，說明如下表：

| 成員函數 | 使用函數 | 傳入參數 | 回傳值 |
|---|---|---|---|
| CArray::Add | AddPoint | 欲加入容器的軌跡點 | 無 |
| CArray::operator[] | GetPoint | 欲取得軌跡點的索引值（從 0 開始） | 傳回容器內第 i 個元素 |
| CArray::GetSize | GetSize | 無 | 傳回容器內儲存軌跡點的個數（元素個數） |

此三函數的程式碼內容如下：

```
'摘自 repaint\repaint.cpp 檔
14        void AddPoint(CPoint p) //將軌跡點加到容器內
15        { pArray.Add(p); }
16
17        CPoint GetPoint(int i) //將軌跡點從容器中取出
18        { return pArray[i]; }
19
20        int GetSize()
21        { return pArray.GetSize(); } //取得容器的大小
```

## View 儲存資料的機制

前面說明了 Document 如何儲存資料，但是並沒有交代究竟這些資料是如何進入到 Document 類別的儲存機制裡。在 5-2 節的 Message 程式範例裡，我們為 MyView 類別建立了訊息映射表，並以 OnLButtonDown()、OnMouseMove()、OnLButtonUp()這三個函數，處理滑鼠在視窗中移動所產生的事件，將滑鼠軌跡繪製在視窗的畫布上。

而儲存滑鼠軌跡點的機制，同樣可以利用這個滑鼠事件的回應機制完成。因此，滑鼠座標的儲存動作，便加入了 OnMouseMove()函數。以下為該函數的程式碼。

```
'摘自 repaint\repaint.cpp 檔
53        afx_msg void OnMouseMove(UINT, CPoint point)
54        {
```

```
55          if (this == GetCapture())
56          {
57              CClientDC aDC(this);  //建立畫布
58              aDC.SetPixel(point, RGB(255, 0 ,0));     //將點畫在畫布上
59              MyDocument *doc = (MyDocument *)GetDocument();
60                  //取得目前 Document 物件的指標
61              doc->AddPoint(point); //將軌跡點加入 Document 物件中
62          }
63      }
```

OnMouseMove()函數所執行的動作，只比 Message 程式範例增加兩行，一是取得目前程式所使用 Document 物件的指標（第 59 行）。另一個是呼叫 Document 物件的 AddPoint()成員函數，將軌跡點儲存到 Document 中（第 61 行）。

第 59 行取得 Document 物件指標的方式，將利用 CView 類別的 GetDocument()成員函數。此成員函數將傳回目前視窗程式使用 Document 物件之指標。由於回傳值的型態為 CDocument*，故還需要將其強制轉型為 MyDocument *型態，否則無法呼叫 MyDocument::AddPoint()函數。

**CDocument* CView::GetDocument( ) const**

□ **函數說明**

取得與目前使用中的 View 物件有關連的 Document 物件。如果 View 物件尚未與 Document 物件連結則傳回 NULL，有則傳回該 Document 物件的指標。

## 6-3-4 視窗的重繪

6-2-4 節曾提到當要建立一個 View 類別時，必須覆寫 OnDraw()虛擬函數，這個虛擬函數將在視窗重繪與列印時被呼叫。所以，當視窗程式欲具備重繪功能時，覆寫 CView::OnDraw()虛擬函數是相當重要的工作。以下為 MyView::OnDraw()成員函數的內容：

```
'摘自 repaint\repaint.cpp 檔
36      void OnDraw(CDC * aDC) //覆寫 OnDraw()虛擬函數
37      {
38         MyDocument *doc = (MyDocument *)GetDocument();
39               //取得目前 Document 物件的指標
40         int num = doc->GetSize();//取得目前儲存的軌跡點點數
41         int i;
42
43         for(i = 0; i < num; ++i)
                              //將 Document 中儲存的軌跡點重繪到視窗上
44         {
45            CPoint point = doc->GetPoint(i);
46            aDC->SetPixel(point, RGB(255, 0 ,0));
47         }
48      }
```

整個 OnDraw()成員函數所肩負的任務為重繪視窗時，顯示目前 Document 物件內的儲存資料，步驟如下：

第一步、 必須取得目前視窗程式所使用的 Document 物件（第 38 行）。

第二步、取得 Document 物件中儲存的軌跡點個數（第 40 行）。

第三步、利用 for 迴圈取得 Document 物件中儲存的滑鼠軌跡點(第 45 行），然後在繪製於 View 的畫布上（第 46 行）。