第3章 MFC应用程序框架

CHAPTER

如果把设计 Windows 应用程序框架所需要的 API 函数和数据封装成类，便可以利用类的继承性实现代码的重用，并在派生过程中对它进行必要的改造，从而快速地获得所需要的类，提高应用程序框架的开发效率。MFC 正是满足上述要求的一个类库，它有一组专门的类，可以快速创建应用程序的框架。

本章主要内容：

- MFC 的基本应用程序框架类。
- Windows 应用程序的文档/视图结构。
- 文档/视图结构的应用程序框架类。
- 对象的动态创建。

3.1 早期的应用程序框架及其 MFC 类

早期的 MFC，就像第 2 章所介绍的那样，在应用程序类中嵌入一个窗口类对象就构成了程序的框架。尽管这种应用程序框架比较简单，但是它体现了 MFC 程序的主体结构，并且它的一些类仍然是当前复杂应用程序框架的基本类，因此，从学习的角度来说，了解早期的 MFC 应用程序框架仍然是必要的。

3.1.1 早期的应用程序框架

早期 MFC 应用程序框架结构与第 2 章例 2-3 的程序结构基本相同。应用程序框架由两个对象组成：应用程序类 CWinApp 的派生类对象和窗口类 CFrameWnd 的派生类对象，后者作为一个成员对象嵌在前者之中，如图 3-1 所示。图中，CMyApp 是应用程序类 CWinApp 的派生类；而 CMyWnd 是窗口类 CFrameWnd 的派生类。

在应用程序主函数 WinMain( )中，CWinApp 派生类的对象 theApp 通过调用自己的各个成员函数来完成程序的初始化及消息循环等一系列

工作。在 CWinApp 成员函数 InitInstance( )中形成应用程序的主窗口对象 pMainWnd（类 CMyWnd 的对象），在完成窗口的创建和显示后，主窗口对象 pMainWnd 将被赋给 CWinApp 的成员 m_pMainWnd。

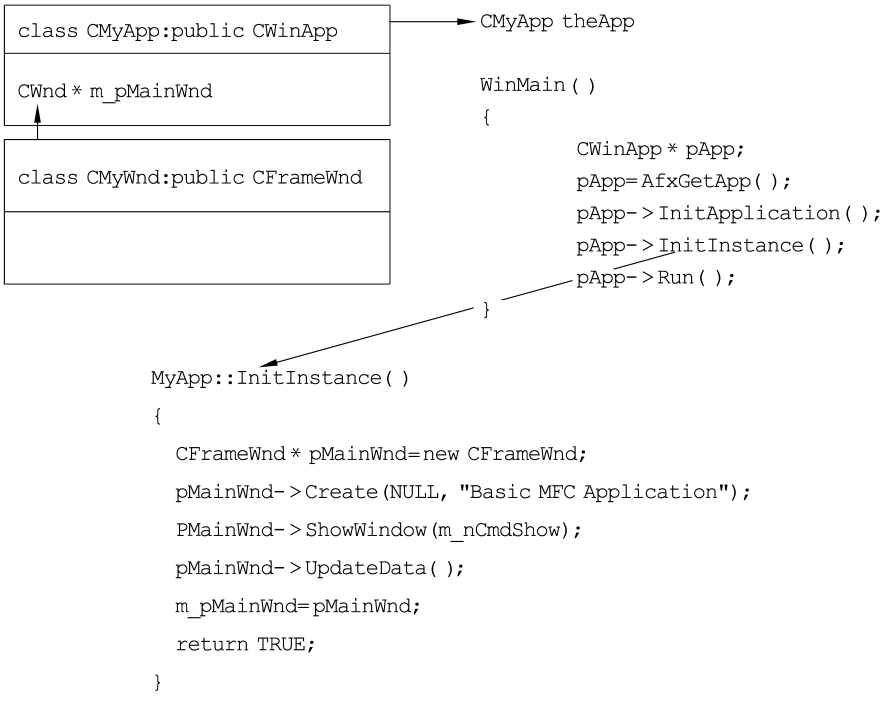

图 3-1　MFC 程序的基本结构

3.1.2　MFC 的窗口类

窗口类 CFrameWnd 是一个重要的类，它的对象通常就是应用程序的主窗口。因此，作为程序设计人员，必须对它和它的基类有一个比较清楚的了解。

在 MFC 类库中的类大多都有一个较长的族系，窗口类 CFrameWnd 也一样，它由基类 CObject 经 CCmdTarget、CWnd 派生而来，它与它的基类之间的继承关系见图 3-2。

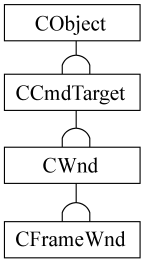

图 3-2　窗口类在类族中的位置

1. CObject 类

仔细观察 MFC 类的层次结构可以发现，在类族中有相当一部分类的共同基类是 CObject 类。

CObject 类为其派生类不仅提供了程序调试诊断信息输出通用功能，并且还对运行期对象类型识别（RTTI）、对象的动态创建、对象的序列化提供了相应的支持。因此，凡是需要具有上述功能的类，必须以 CObject 或其派生类为基类来派生（CObject 类的详细信息请参见附录 E）。

2. CCmdTarget 类

为了支持消息处理，MFC 以 CObject 类为基类派生了 CCmdTarget 类，并在这个类中封装了窗口函数，因此凡是希望具有处理 Windows 消息的能力的类都必须以 CCmdTarget 类或其派生类为基类来派生。

3. CWnd 类

Windows 把应用程序窗口界面上的许多图形元素，例如，控制栏、对话框、视图、属性页和控件等，都看作是子窗口。为了对这些窗口类提供应有的通用属性和方法。MFC 以 CCmdTarget 类为基类派生了 CWnd 类。所以，凡是以窗口形式（方形）为外观并且可以响应消息的类（例如，按钮类 CButton、滚动条类 CScrollBar 等），它们的基类都是 CWnd 类。CWnd 类的部分成员函数见表 3-1。

表 3-1　CWnd 类的部分成员函数

函　　数	说　　明
Create( )	创建一个子窗口
EnableWindow( )	使窗口的鼠标和键盘输入有效
ModifyStyle( )	改变窗口的样式
MoveWindow( )	改变窗口的位置和大小
PreCreateWindow( )	在程序显示窗口之前改变窗口的样式
SetWindowText( )	设置窗口标题的文本
ShowWindow( )	显示或隐藏窗口

4. CFrameWnd 类

应用程序窗口类 CFrameWnd 是一个特殊的 CWnd 类，它或它的派生类对象要承担应用程序主窗口的任务，所以它除了需要 CWnd 类的一些通用功能之外，还需要一些特殊功能，因此它由 CWnd 类派生。由于按钮、滚动条、菜单条这些子窗口都放置在主窗口之上，所以它也是其他子窗口对象的容器。

3.1.3　CWinApp 类

MFC 希望把程序的主函数的函数体部分也作为一个对象来处理，为此提供了应用程序类 CWinApp，它在 MFC 类族中的位置见图 3-3。由图中可见，CWinApp 类具有 CObject 类和 CCmdTarget 类的全部特性。为了支持 Windows 多线程工作方式，MFC 在 CCmdTarget 和 CWinApp 类之间构建了一个线程类 CWinThread。这个线程类 CWinThread 中封装了一些用于线程管理的功能函数。

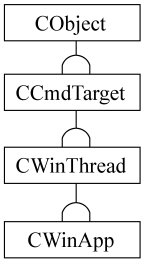

图 3-3　应用程序类在类族中的位置

值得程序设计人员注意的是，MFC 把原来在 CWinApp 类中定义的 CWnd * 类型的数据成员

m_pMainWnd(程序的主窗口对象)放在 CWinThread 类中来定义了，所以在 CWinApp 类声明中看不到这个对象。

除此之外，CWinApp 类中还有三个可以重写的虚成员函数 InitApplication( )、InitInstance( )和 Run( )。其中，成员函数 InitInstance( )是为程序创建窗口和显示窗口所设置的。因此在设计程序时，必须在 CWinApp 类的基础上派生出自己的应用程序类，并对函数 InitInstance 进行重写，以实现对窗口的不同的要求。例如：

```
class MyApp:public CWinApp            //由 CWinApp 派生自己的应用程序类
{
public:
    BOOL InitInstance( );
};
MyApp theApp;
BOOL MyApp::InitInstance( )           //重写 InitInstance
{
    由程序员编写的窗口创建代码;
}
```

3.2 最简单的 MFC 程序实例

为了更清楚地了解 MFC 的应用程序框架类及开发这种应用程序的步骤，下面给出了一个实例。

3.2.1 程序的编写

例 3-1 使用早期 MFC 应用程序框架类设计的一个最简单的 Windows 应用程序，它只创建了一个窗口。

具体设计步骤如下：

(1) 选择 VC ++ 的菜单 File|New 选项，打开 New 对话框。选择 Projects 选项卡。在左下小窗口中选择 Win32 平台，在 Project name(工程名)文本框中添入工程名称，例如 MyApp，然后在 Location 文本框中选择工作目录，再选择工程类型为 Win32 Application (见图 3-4)，最后单击 OK 按钮。这样就在编程环境中建立了一个空的工作空间。

(2) 第二次选择 VC ++ 的菜单 File|New 选项，使用 Files 选项卡，在 File 文本框中添入文件名称(MyApp)，选择 Add to project 复选项。然后在右窗口中选择文件类型 C ++ Source File，最后单击 OK 按钮，这样 VC ++ 就提供了一个空白的 C ++ 源文件，参见图 3-5。

(3) 在文件中写入如下代码。

```
#include <afxwin.h>
//由 CWinApp 派生的应用程序类声明
class MyApp : public CWinApp
```

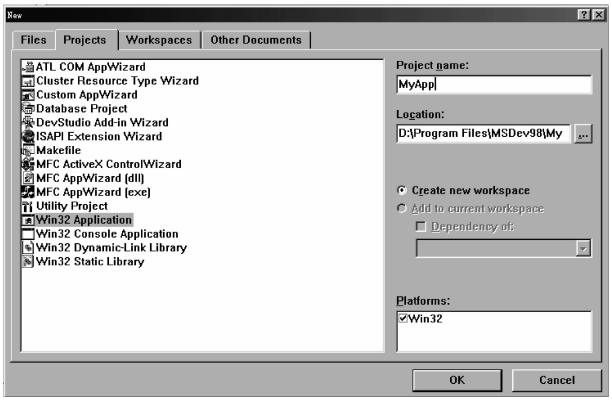

图 3-4　创建工作空间

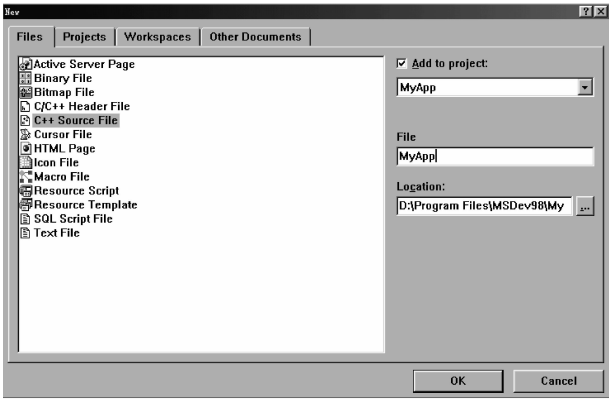

图 3-5　向工程加入源文件

```
{
public:
    BOOL InitInstance( );                    //声明 InitInstance( )函数
};
//定义应用程序类的全局对象
MyApp theApp;
//InitInstance 函数的实现
MyApp::InitInstance( )
{
    CFrameWnd * pMainWnd=new CFrameWnd;     //创建窗口框架类的对象
    pMainWnd->Create(NULL, "Basic MFC Application");
    pMainWnd->ShowWindow(m_nCmdShow);       //显示窗口
    pMainWnd->UpdateData( );
    m_pMainWnd=pMainWnd;
    return TRUE;
}
```

（4）选择菜单 Project | Settings 选项，在出现的 Project Settings 对话框中，打开 Setting For 下拉列表，在表中选择 All Configurations。在 General 选项卡的 Microsoft Foundation Classes 下拉列表中选择 Use MFC in a Static Library 来指定工程使用静态 MFC 类库，如图 3-6 所示。

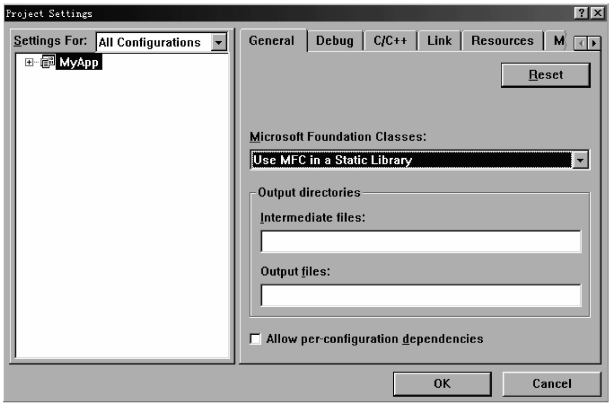

图 3-6　设置工程选项

（5）按下 Ctrl＋F5 键，系统将会编译、链接并运行该程序。这个程序运行后将会出现一个窗口。

3.2.2　程序主函数的代码

在前面的源文件中，没有看到主函数，那么主函数哪里去了呢？主函数是自动生成的。其代码为：

```
AFXAPI AfxWinMain( HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPTSTR lpCmdLine,
                   int nCmdShow)
{
        ASSERT(hPrevInstance==NULL);
        int nReturnCode=-1;
        CWinThread * pThread=AfxGetThread( );
        CWinApp * pApp=AfxGetApp( );          //获得程序的指针
        if (!AfxWinInit(hInstance, hPrevInstance, lpCmdLine, nCmdShow))
             goto InitFailure;
        if (pApp!=NULL && !pApp->InitApplication( ))
             goto InitFailure;

        if (!pThread->InitInstance( ))     //初始化
        {
             if (pThread->m_pMainWnd!=NULL)
             {
```

```
                TRACE0("Warning: Destroying non-NULL m_pMainWnd\n");
                pThread->m_pMainWnd->DestroyWindow( );
            }
            nReturnCode=pThread->ExitInstance( );
            goto InitFailure;
        }
        nReturnCode=pThread->Run( );        //消息循环
    InitFailure:
    #ifdef _DEBUG
    //Check for missing AfxLockTempMap calls
        if (AfxGetModuleThreadState( )->m_nTempMapLock != 0)
        {
            TRACE1("Warning: Temp map lock count non-zero (%ld).\n",
            AfxGetModuleThreadState( )->m_nTempMapLock);
        }
        AfxLockTempMaps( );
        AfxUnlockTempMaps(-1);
    #endif
        AfxWinTerm( );
        return nReturnCode;
    }
```

其中，重要的语句为 CWinApp * pApp＝AfxGetApp( )，它的作用是获得由用户定义的应用程序类 MyApp 的对象 theApp；然后使用该对象由用户重写的初始化函数 InitInstance( )，并在这个函数中创建和显示程序的窗口；最后，使用 MyApp 对象的成员函数 Run( )进入消息循环。再就是要注意到主函数的名称变为 AfxWinMain。

3.3　应用程序的文档/视图结构

目前，用 MFC 设计的 Windows 应用程序几乎都采用文档/视图结构。这种新程序框架与原先简单程序框架相比，其最重要的区别是原来的应用程序主窗口对象被拆分成窗口框架类 CFrameWnd 对象、视图类 CView 对象和文档类 CDocument 对象三个对象。

由于文档/视图结构的应用程序比较复杂，因此很少有人自己去编写程序框架的代码，而是利用 Visual C ++ 环境提供的应用程序设计向导 MFC AppWizard 自动生成，程序员的工作就是在这个框架的基础上根据需要来添加自己的代码。

3.3.1　文档/视图结构的基本概念

从简单应用程序框架的介绍中可以知道，窗口对象的任务极其繁重：它既要管理窗口本身的一些事务(例如，窗口的最大化、最小化、关闭、响应主菜单命令等)，又要管理应用程序的数据，同时还要负责数据的显示和接受用户区的消息及处理等任务。随着应用程序规模的扩大，这个矛盾就更为突出。所以，为了减轻窗口对象的负担，分工更为明确，

每个对象的任务更为专业，MFC 把早期窗口类的功能分解成 3 个部分：数据存储、管理部分，数据显示与用户交互部分，管理窗口框的大小、标题、菜单条、状态条的窗框部分。进而形成了 3 个类，即文档类 CDocument、视图类 CView 和窗口框架类 CFrameWnd。

现在，窗口框架类 CFrameWnd 只承担应用程序窗口边框那部分任务，而把程序窗口的用户区那部分功能单独分割出来构建了一个新的类 CView——视图类，由它的对象来完成数据的显示、用户区消息的响应和处理等工作；至于程序数据的存储、运算和管理等工作则交给了文档类 CDocument 对象。

在应用程序中，上面所说的 3 个对象由一个叫做文档模板的对象来统一创建和管理，使它们能够形成一个相互配合、相互协调的实体。这样，它们的分工合作就形成了一个 CDocument 对象在后台，作为应用程序的数据库；CFrameWnd 对象和 CView 对象在前台，作为应用程序界面的“后库前店”结构。

这种把数据的存储与数据的显示分开的结构，带来的最大好处是能够使一份文档（数据）可以由多个视图从不同的角度显示，从而能形成“一库多店”的结构，使得应用程序结构更加轻灵，界面更为人性化。

上面 3 个类对象之间的关系类似于房屋的窗户，窗口框架类 CFrameWnd 相当于窗框，视图类 CView 相当于窗框上的玻璃，而文档类 CDocument 就相当于室内的物品。在房屋的外面，透过玻璃可以窥见室内的部分物品。作为房屋，它可以有多个窗户从不同的角度来查看房屋中的同一物品。应用程序对象、文档模板对象、窗口框架对象、视图对象和文档对象之间的关系见图 3-7。

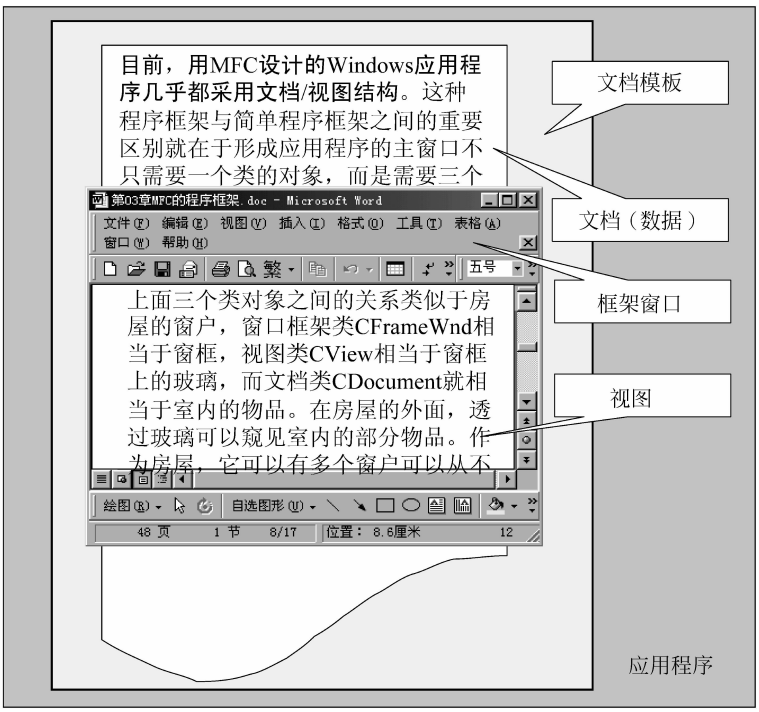

图 3-7　框架窗口类、视图类、文档类、文档模板、应用程序类之间的关系

应用程序类对象则是上述所有对象的容器和消息传递中心。

3.3.2　单文档界面和多文档界面结构

有两种类型的文档/视图结构程序：单文档界面(SDI)应用程序和多文档界面(MDI)应用程序。

在单文档界面程序中，用户在一个时刻只能操作一个文档，Windows 下的 NotePad 记事本程序(如图 3-8 所示)就是一个单文档界面程序的例子。在这种应用程序中，打开文档时会自动关闭当前文档，若文档修改后尚未保存，会提示是否保存所做的修改。这种程序相对比较简单，常见的应用程序有终端仿真程序和诸如杀毒软件等一些工具程序。

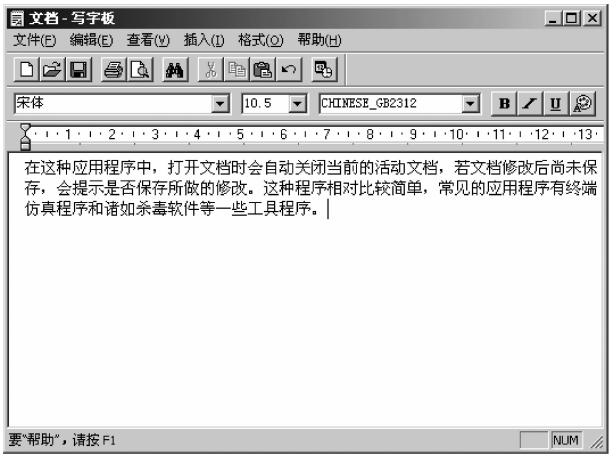

图 3-8　Windows 的 NotePad 记事本程序界面

多文档界面应用程序允许同时打开和操作多个文档，而且可以是不同类型的文档，如图 3-9 所示。多文档应用程序提供一个 File 菜单，用于新建、打开、保存文档。与单文档应用程序不同的是，它往往还提供一个 Close(关闭)菜单项，用于关闭当前打开的文档。

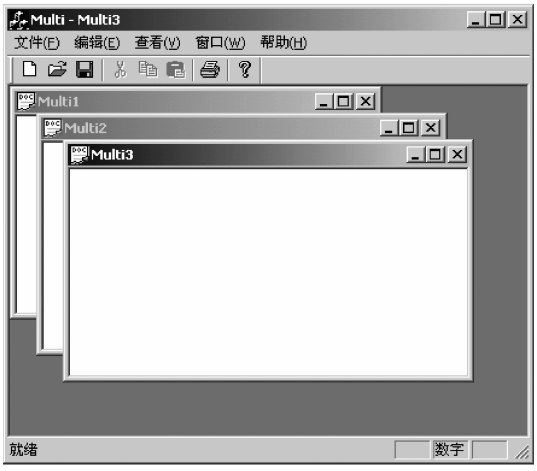

图 3-9　多文档应用程序界面

多文档应用程序还提供一个 Windows 菜单，用以对所有打开的子窗口进行管理，这个菜单包括了对子窗口的新建、关闭、层叠、平铺等操作选项。关闭一个窗口时，窗口内的文档也被自动关闭。

从图 3-9 中可以看到，多文档程序除了需要一个应用程序框架窗口外，每个文档还需要一个文档框架窗口（子窗口）。

3.4 文档类 CDocument 的派生类

程序员在用 MFC AppWizard 生成应用程序框架时，MFC AppWizard 会自动以文档类 CDocument 为基类，为应用程序派生一个类名称中含有工程名的文档类，这个派生文档类的主要代码如下（例如，工程名为 My）：

```
class CMyDoc : public CDocument
{
protected:
    CMyDoc( );
    DECLARE_DYNCREATE(CMyDoc)
public:
    virtual BOOL OnNewDocument( );
    virtual void Serialize(CArchive& ar);
    virtual ~ CMyDoc( );
    DECLARE_MESSAGE_MAP( )
};
```

可以看到，这个类基本就是一个框架，但它是应用程序的数据库，是程序员定义程序数据和对这些数据进行操作的成员函数的地方。

文档类的派生类中还准备了两个用户可以重写的虚函数，其中比较重要的是 Serialize( )函数。当用户用菜单对文件进行新建、打开、保存等操作时，应用程序会自动调用这个函数，它既负责由文件（存储在永久存储介质中的数据）读取数据，也负责向文件存储数据（参见第 9 章）。所以这个函数也是程序设计人员编码较多的地方。

例 3-2 在文档类中定义数据成员及其成员函数的实例。

```
class CMyDoc : public CDocument
{
private:
    int Array[3];                              //定义一个整型数组
protected:
    CMyDoc( );
    DECLARE_DYNCREATE(CMyDoc)
public:
    void SetMem(int i,int x);                  //给数组元素赋值的成员函数
    int GetMem(int i);                         //获取数组元素值的成员函数
public:
```

```
    virtual BOOL OnNewDocument( );
    virtual void Serialize(CArchive& ar);
    virtual~CMyDoc( );
    DECLARE_MESSAGE_MAP( )
};
CMyDoc::CMyDoc( )
{
     for(int i=0;i<3;i++) Array[i]=0;          //数组元素赋初值
}
void CMyDoc::SetMem(int i,int x)               //给数组元素赋值的成员函数
{
     Array[i]=x;
}
int CMyDoc::GetMem(int i)                      //获取数组元素值的成员函数
{
     return Array[i];
}
```

由于文档类派生自 CCmdTarget 类，故它可以接收来自菜单或工具条发来的命令消息（WM_COMMAND 消息）。

3.5　视图类 CView 的派生类

视图类 CView 对象没有自己的边框，它的作用是为框架窗口提供用户区。听起来比较抽象，实际上就是把原来窗口框架类承担数据显示和接受用户对用户区操作（消息映射）的代码单独分出来，形成了一个单独的类。它的对象是应用程序与用户进行交互的界面，也是程序员编写代码最多的地方。

如果使用 MFC AppWizard 来创建应用程序，向导会为程序员自动生成一个含有工程名的 CView 类的派生类。例如，工程名为 My，则向导派生的视图类名称就叫做 CMyView，这个类主要代码为：

```
class CMyView : public CView
{
protected:
     CMyView( );
     DECLARE_DYNCREATE(CMyView)
public:
     CMyDoc * GetDocument( );
     virtual void OnDraw(CDC * pDC);
     virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
     protected:
     ⋮
public:
```

```
    virtual~CMyView( );
protected:
    DECLARE_MESSAGE_MAP( )
};
```

视图类有几个重要的成员函数 GetDocument( )、OnDraw( )和 PreCreateWindow( )，其中最重要的是前两个函数 GetDocument( )和 OnDraw( )。

1. GetDocument( )函数

GetDocument( )函数用于获得文档类对象的指针，因此它是视图类对象与文档类对象进行联系的通道。这个函数是视图类对象获取文档数据的重要手段，在程序设计时，视图类对象必须通过它来访问文档类对象中的数据。所以也有人把这个函数叫做“店到库的后门”。

2. OnDraw( )函数

这是一个消息处理函数，它的作用是用来更新视图的显示。系统向这个函数传递了一个指向 CDC 类对象的指针。如果把窗口用户区看成是一张画布，把 OnDraw( )函数看作是程序用来做画的画室，那么 CDC 类对象就是作画所需要使用的工具箱，它提供了画笔、画刷、调色板等绘图工具，程序员可以使用这些工具把来自文档的数据显示到窗口的画布（窗口用户区）上。

这个函数也叫做重画函数，因为当应用程序窗口出现及其大小发生变化时，系统会自动调用 OnDraw( )函数，对窗口进行重画。除此之外，其他对象也可通过发出更新视图命令的方法来产生重画消息以调用 OnDraw( )函数。

例 3-3 在 CMyView 类中的函数 OnDraw( )中，把例 3-2 在文档类中定义数组元素 Array[1]赋值为 100，并利用窗口显示时会调用函数 OnDraw( )的特点，在用户区以 Array[1]的值为边长画一个正方形。

```
//CMyView::OnDraw( )的代码为
CMyView::OnDraw(CDC * pDC)
{
    CMyDoc * pDoc=GetDocument( );                    //获得文档对象
    pDoc->SetMem(1,100);                             //调用文档对象的成员函数进行赋值
    CRect rt(40,40,                                  //定义一个矩形类对象
            40+pDoc->GetMem(1),40+pDoc->GetMem(1));
    pDC->Rectangle(&rt);                             //调用 CDC 类成员函数画矩形
}
```

3.6 窗口框架类 CFrameWnd 的派生类

如果使用 MFC AppWizard 来创建应用程序，向导会为程序员自动从 CFrameWnd 类派生一个叫做 CMainFrame（程序主窗口框架）的派生类。派生类 CMainFrame 的主要代码如下：

```
class CMainFrame : public CFrameWnd
{
protected:
    CMainFrame( );
    DECLARE_DYNCREATE(CMainFrame)
public:
    virtual ~ CMainFrame( );
protected:
    DECLARE_MESSAGE_MAP( )
};
```

从上面的代码中看不出什么东西来，之所以会这样，是因为程序的数据部分已交由文档类对象负责，与用户交互的消息处理和显示已交由视图类对象负责，那么它的事情当然就不多了。所以对于一般用户来说，MFC AppWizard 自动生成的这个派生类已经由其基类继承了相当完善的功能，足够一般应用程序使用，也就没有什么工作需要用户再做的了。

当然，它还为高级程序员提供了一些可以使用的方法。为了使读者有一个概念，把 CFrameWnd 的部分代码列举如下：

```
class CFrameWnd : public CWnd
{
    DECLARE_DYNCREATE(CFrameWnd)
public:
    ⋮
    BOOL Create(…);
    CWnd * CreateView(…);                          //定义视图对象
    virtual CDocument * GetActiveDocument( );      //取得活动文档对象指针
    CView * GetActiveView( ) const;                //取得活动视图对象指针
    void SetActiveView(…);                         //设置活动视图对象
    virtual CFrameWnd * GetActiveFrame( );         //取得活动框架对象指针
    void SetTitle(LPCTSTR lpszTitle);              //设置标题
    CString GetTitle( ) const;                     //获取标题
    ⋮
    CControlBar * GetControlBar(UINT nID);         //获取控制条
    ⋮
//命令处理函数
public:
    afx_msg void OnContextHelp( );
    afx_msg void OnUpdateControlBarMenu(CCmdUI * pCmdUI);
    afx_msg BOOL OnBarCheck(UINT nID);
    //Windows 消息处理函数
    afx_msg void OnDestroy( );
    afx_msg void OnClose( );
```

```
    afx_msg void OnInitMenu(CMenu * );
    afx_msg void OnInitMenuPopup(CMenu * , UINT, BOOL);
    ⋮
    afx_msg void OnHScroll(…);
    afx_msg void OnVScroll(…);
    afx_msg void OnSize(…);
    ⋮
    DECLARE_MESSAGE_MAP( )
    friend class CWinApp;
};
```

3.7 文档模板类 CDocTemplate

为了把视图对象、框架窗口对象和文档对象组装在一起并统一管理，MFC 使用了一个叫做文档模板的抽象类 CDocTemplate。CDocTemplate 的两个派生类：单文档模板类 CSingleDocTemplate 和多文档模板类CMultiDocTemplate，一个用于单文档界面程序，一个用于多文档界面程序。

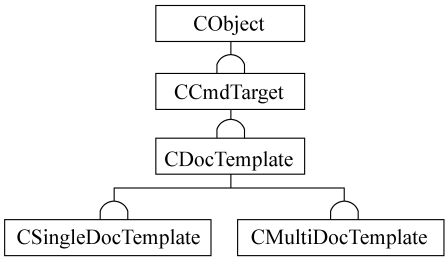

图 3-10 文档模板的类继承关系

文档模板的类继承关系见图 3-10。

单文档模板类的部分代码为：

```
class CSingleDocTemplate :
public CDocTemplate
{
    DECLARE_DYNAMIC(CSingleDocTemplate)

//构造函数
public:
    CSingleDocTemplate(UINT nIDResource,                    //程序资源
                       标识
                         CRuntimeClass * pDocClass,         //文档
                          CRuntimeClass * pFrameClass,      //窗口框架
                         CRuntimeClass * pViewClass         //视图
                         );
⋮
protected:
    CDocument * m_pOnlyDoc;                                 //文档指针
};
```

多文档模板类的部分代码为：

```
class CMultiDocTemplate : public CDocTemplate
```

```
{
    DECLARE_DYNAMIC(CMultiDocTemplate)
//构造函数
public:
    CMultiDocTemplate(UINT nIDResource,
                      CRuntimeClass * pDocClass,
                      CRuntimeClass * pFrameClass,
                      CRuntimeClass * pViewClass
                      );
⋮
protected:    //standard implementation
   CPtrList m_docList;        //文档链表
};
```

这两个派生类之间的重要区别就在于 CSingleDocTemplate 中只有一个文档指针，而在 CMultiDocTemplate 中是一个文档链表，也就是说，单文档界面应用程序每次只能打开一个文档，而多文档界面应用程序则可以同时打开多个文档。

如果是使用 MFC AppWizard 创建应用程序，向导会根据程序员的要求自动生成一个合适的文档模板对象，程序员一般不需要对它进行修改。

3.8　应用程序类的派生类

3.8.1　应用程序类派生类的代码

作为应用程序，还需要一个应用程序类对象作为上述各类对象的容器，并实现应用程序的初始化和消息循环。当使用 MFC AppWizard 生成程序时，向导会为程序员自动生成一个 CWinApp 的派生类。例如，一工程名为 My 的应用程序类的代码如下所示。

```
class CMyApp : public CWinApp
{
public:
    CMyApp( );
public:
    virtual BOOL InitInstance( );
    afx_msg void OnAppAbout( );
    DECLARE_MESSAGE_MAP( )
};
```

应用程序对象在程序运行之前由系统创建，其中的 OnAppAbout( )是在应用程序用户单击菜单【帮助】|【关于】选项时的消息处理函数。InitInstance( )是进行程序初始化工作的虚函数，程序设计人员可以通过改写进行自己的初始化工作。

由程序设计向导生成的成员函数 CMyApp::InitInstance( )的部分代码如下：

```
BOOL CMyApp::InitInstance( )
```

```
{
    ⋮
    CSingleDocTemplate * pDocTemplate;                  //声明文档模板指针
    pDocTemplate=new CSingleDocTemplate(                //创建文档模板对象
        IDR_MAINFRAME,                                  //文档模板使用的资源 ID
        RUNTIME_CLASS(CMyDoc),                          //创建文档对象
        RUNTIME_CLASS(CMainFrame),                      //创建 SDI 框架窗口对象
        RUNTIME_CLASS(CMyView));                        //创建视图对象
    AddDocTemplate(pDocTemplate);                       //将文档模板加入模板链表
    ⋮
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow( );
    return TRUE;
}
```

当程序启动之后，应用程序对象首先创建文档模板，并且在文档模板的构造函数中传递了四个参数：第一个参数是文档模板使用的资源 ID，剩下的其余 3 个参数都是由宏 RUNTIME_CLASS( )来创建的对象。它们分别是文档对象、框架窗口对象和视图对象，是由文档模板统一管理的 3 个内嵌对象。

MDI 应用程序可以有多个文档模板，它们被连接成一个链表。所以当创建了一个文档模板对象之后，还需要调用函数 AddDocTemplate( )将该模板加入模板链表。请读者自行阅读 MDI 应用程序中函数 InitInstance( )的代码。

如果把上面的代码与早期的 MFC 程序的代码比较一下，就会看到，函数中的粗体字代码相当于第 2 章例 2-3 应用程序窗口定义的代码：

```
MyWnd * pMainWnd=new MyWnd;
pMainWnd->CreateWin( );
```

即在文档/视图框架程序中，文档模板对象就相当于早期 MFC 程序中的窗口对象，只不过文档模板作为管理者把现在的文档对象、视图对象和窗口框架对象装到了一起。

3.8.2 程序员的主要工作

如果使用 MFC AppWizard 创建程序框架，向导会自动提供程序所应有的派生类，并同时定义应用程序类的全局对象和生成主函数。因此程序员的工作主要是如下几个方面：

（1）重写 CWinApp 派生类的虚函数 InitInstance( )。在这个函数中，按自己的需要创建和显示窗口。

（2）在 CDocument 的派生类中，声明程序所需的数据和对这些数据进行必要操作的接口函数。

（3）在 CView 类的派生类中编写处理消息的代码。如果在消息处理中需要文档中的数据，则应该调用该类的成员函数 GetDocument( )来获取文档对象，然后通过文档对

象的接口函数对文档中的数据进行操作。

(4) 在 CView 类的派生类的 OnDraw( )函数中编写窗口重绘时的代码。

(5) 用宏实现类的消息映射表。

3.9　MFC 文档/视图应用程序框架中各个对象的关系

从前面的叙述中可以知道,文档/视图应用程序框架的构成是比较复杂的,因此,正确理解程序各个部分之间的关系和沟通方法是设计程序的关键。

3.9.1　应用程序各对象创建的顺序

应用程序对象是全局对象,它在程序启动之前由系统创建。应用程序启动之后,程序的主函数首先调用应用程序对象的初始化函数 InitInstance( ),并在该函数中创建文档模板对象。在函数 InitInstance( )中创建单文档模板对象的代码如下:

```
CSingleDocTemplate * pDocTemplate;              //声明文档模板指针
pDocTemplate=new CSingleDocTemplate(            //创建文档模板对象
        IDR_MAINFRAME,                          //文档模板使用的资源 ID
        RUNTIME_CLASS(CMyDoc),                  //创建文档对象
        RUNTIME_CLASS(CMainFrame),              //创建主 SDI 框架窗口对象
        RUNTIME_CLASS(CMyView)                  //创建视图对象
        );
AddDocTemplate(pDocTemplate);                   //将文档模板加入链表
```

在函数 InitInstance( )中创建多文档模板对象的代码如下:

```
CMultiDocTemplate * pDocTemplate;               //声明文档模板指针
pDocTemplate=new CMultiDocTemplate(             //创建文档模板
        IDR_MYTYPE,                             //加载文档资源
        RUNTIME_CLASS(CMyDoc),                  //创建文档对象
        RUNTIME_CLASS(CChildFrame),             //创建子窗口对象
        RUNTIME_CLASS(CMyView)                  //创建视图对象
        );
AddDocTemplate(pDocTemplate);                   //将文档模板加入链表
//创建多文档主窗口
CMainFrame * pMainFrame=new CMainFrame;         //创建应用程序主窗口
if (!pMainFrame->LoadFrame(IDR_MAINFRAME))      //加载资源
        return FALSE;
m_pMainWnd=pMainFrame;                          //主窗口对象赋予指针 m_pMainWnd
```

在用文档模板构造函数创建文档模板对象的时候,在文档模板构造函数的参数列表中除了传递所需要的资源 ID 之外,还用 MFC 的宏 RUNTIME_CLASS( )传递了文档类、框架窗口类和视图类的类信息表,然后由模板类的构造函数根据资源和类信息表动态地创建文档、视图、窗口框架 3 个对象。其中,视图对象是由框架窗口对象创建并管理的。

应用程序各对象的创建顺序如图 3-11 所示。

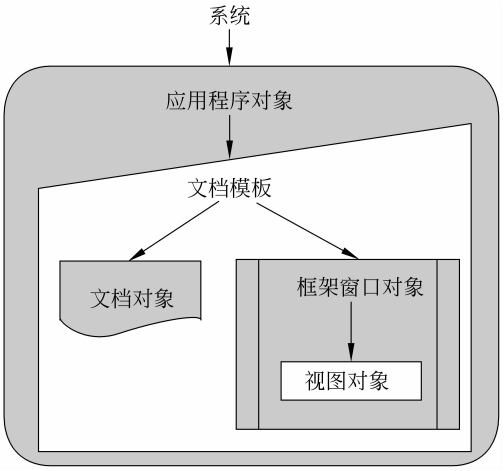

图 3-11　应用程序创建各对象的顺序

最后，应用程序创建文档模板对象并将其加入由应用程序对象维护的文档模板链表。

3.9.2　应用程序各对象之间的联系

1. 以文档为中心的结构

一个应用程序只有一个应用程序类 CWinApp 或其派生类的对象。如果是多文档结构的应用程序，则应用程序对象必须维护一个文档模板链表对多个文档模板进行管理，应用程序每新建或打开一个文档就会创建一个文档模板并将其加入文档模板链表。应用程序对象与文档模板链表之间的关系如图 3-12 所示。

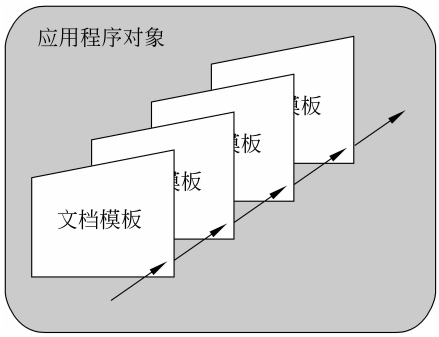

图 3-12　应用程序与文档模板对象链表

在文档模板中，文档模板委托一个 CDocManager 类对象来负责文档对象和窗口框架的管理工作。其中的每个文档对象都有一个指向其所属文档模板的指针，并且每个文档对象还要管理一个链表，这个链表的每一个节点都指向与该文档相关联的视图对象。

框架窗口对象作为视图对象的容器，也有一个由它管理的视图对象链表，并有一个指向当前活动视图对象的指针。

视图对象为了能和与之关联的文档对象沟通，每个视图对象都有一个指向与其关联的文档指针，视图对象可以通过调用自己的成员函数 GetDocument( )获取这个指针，并通过这个指针对文档对象中的数据和函数进行访问或调用。

文档模板、文档对象、框架窗口对象、视图对象之间关系的示意图如图 3-13 所示。

从图 3-13 中可以知道，这是一个以文档为中心的结构。

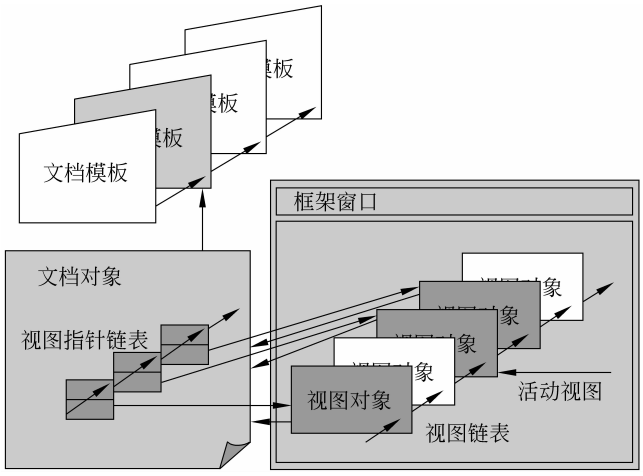

图 3-13　文档模板、文档对象、框架窗口对象、视图之间关系示意图

2. 应用程序框架对象之间的联系方法

MFC 应用程序框架的各个对象都从各自的基类继承了一些获得其他对象指针的方法，从而可以使各对象通过这些指针与其他对象的成员来互相联系。

例如，视图对象可以使用其基类 CView 的成员函数 GetDocTemplate( )获得文档模板对象；可以使用 GetDocument( )获得与其关联的文档对象；可以使用 GetParentFrame( )获得所属的框架窗口对象。

框架窗口对象可以使用继承来的 GetActiveView( )获得活动视图对象，使用 GetActiveDocument( )获得活动视图的文档。

文档对象可以使用继承来的 GetFirstViewPosition( )和 GetNextView( )获得视图链表中的视图对象。

再例如，视图对象可以通过调用 CDocument::UpdateAllViews( )向与这个视图关联的文档发送一个消息，使所有与这个文档相关联的视图对象进行显示更新。文档对象可以通过调用 CView::OnUpdate( )去更新一个视图对象的显示。框架窗口对象可以调用 CView::OnActivateView( )使一个视图对象为活动视图等。

SDI 应用程序框架对象之间的联系方法如图 3-14 所示，MDI 应用程序框架对象之间的联系方法如图 3-15 所示。

例 3-4　一个可以在视图对象中显示文档数据成员 m_Text 和文档标题的应用程序。代码如下：

```
class CMFCexp3_4Doc : public CDocument
{
⋮
public:
    char * m_Text;                                    //在文档的派生类中定义一个字符指针
};
CMFCexp3_4Doc::CMFCexp3_2Doc( )
```

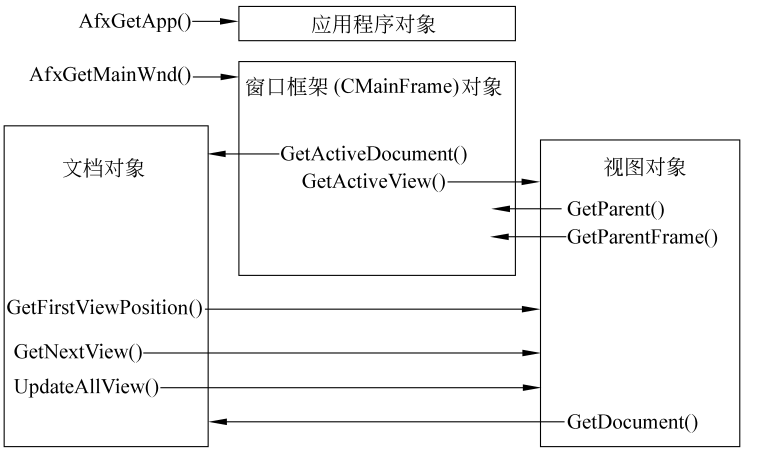

图 3-14　SDI 应用程序框架各对象之间的联系方法

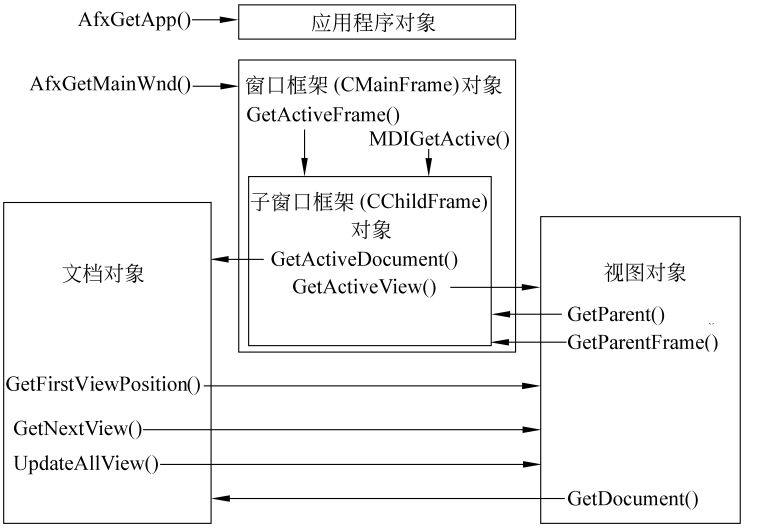

图 3-15　MDI 应用程序框架各对象之间的联系方法

```
{
    m_Text="Hello!";                              //在文档构造函数中初始化字符指针
}
//视图对象的 WM_PAINT 消息响应函数
void CMFCexp3_4View::OnDraw(CDC * pDC)
{
    CMFCexp3_4Doc * pDoc=GetDocument( );          //获取与视图关联的文档指针
    pDC->TextOut(50,50,pDoc->m_Text,6);           //显示字符指针的数据
    pDC->TextOut(190,50,pDoc->GetTitle( ));       //显示文档标题
}
```

在这个例子中用到了视图对象的方法 GetDocument( ) 和文档对象的方法 GetTitle( )，程序运行结果如图 3-16 所示。