深入了解 MFC 中的文挡/视结构

李泽宇　金 刚　熊联欢　姜 军
（华中理工大学图象识别与人工智能研究所）

Visual C++ 5.0 以其功能强大、用户界面友好而倍受程序员们的青睐。但是，在当前的 Microsoft 基本类库 4.2 版本中，大约有将近 200 个类，数千个函数，加之 Microsoft 公司隐藏了一些技术细节，使得人们深入学习 MFC 变得十分困难。

MFC 的 AppWizard 可以生成三种类型的应用程序：基于对话框的应用、单文档应用（SDI）和多文档应用（MDI）。前两者的结构较简单，本文不再赘叙。笔者拟从 MFC 中的文档/视结构入手，分析一些函数的流程，并解决编制 MDI 应用程序过程中的一些常见问题。

（一）、了解文档/视结构

MFC 应用程序模型历经多年以有了相当大的发展。有一个时期，它只是个使用应用程序对象和主窗口对象的简单模型。在这个模型中，应用程序的数据作为成员变量保持在框架窗口类中，在框架窗口的客户区中，该数据被提交显示器。随着 MFC2。0 的问世，一种应用程序结构的新方式----MFC 文档/视结构出现了。在这种结构中，CFrameWnd 繁重的任务被委派给几个不同类，实现了数据存储和显示的分离。一般情况下，采用文档/视结构的应用程序至少应由以下对象组成：

。应用程序是一个 CwinApp 派生对象，它充当全部应用程序的容器。应用程序沿消息映射网络分配消息给它的所有子程序。

。框架窗口是一 CfrmeWnd 派生对象。

。文档是一个 CDocument 派生对象，它存储应用程序的数据，并把这些信息提供给应用程序的其余部分。

。视窗是 Cview 派生对象，它与其父框架窗口用户区对齐。视窗接受用户对应用程序的输入并显示相关联的文档数据。

通常，应用程序数据存在于简单模型中的框架窗口中。在文档/视方式中，该数据移入称为 document 的独立数据对象。当然，文档不一定是文字，文档是可以表现应用程序使用的数据集的抽象术语。而用户输入处理及图形输出功能从框架窗口转向视图。单独的视窗完全遮蔽框架窗口的客户区，这意味着即使程序员直接绘画至框架窗口的客户区，视图仍遮蔽绘画，在屏幕上不出现任何信息。所以输出必须通过视图。框架窗口仅仅是个视图容器。

CDocument 类对文档的建立及归档提供支持并提供应用程序用于控制其数据的接口。MDI 应用程序可以处理多个类型的文档，每个类型的文档拥有一个相关联的文档模板对象。文档对象驻留在场景后面，提供由视图对象显示的信息。文档至少有一个相关联的视图。视图只能与一个文档相关联。

在文档/视方式中，对象的建立是由文档模板来管理的，它是 CDocTemplate 派生对象，建立并维护框架窗口，文档及视。

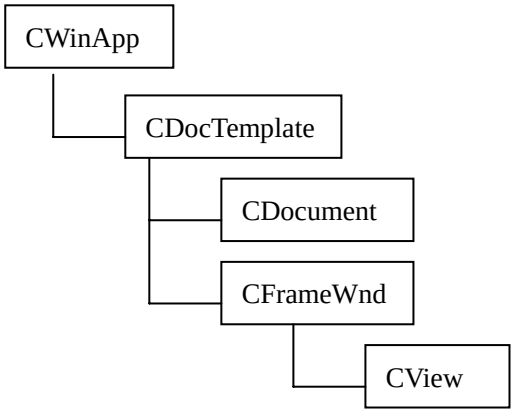
MFC 调用命令处理程序以响应发生在应用程序中的事件。命令发送的优先级是：

活动的视图->框架窗口->文档->应用程序->默认窗口过程(DefWindowsProc)

总之，在文档/视方式中，文档和视是分离的，即：文档用于保存数据，而视是用来显示这些数据。文档模板维护它们之间的关西。这种文档/视结构在开发大型软件项目时特别有用。

（二）、了解与文档/视结构有关的各种类之间的关系。

在文档/视应用程序中，CWinApp 对象拥有并控制文档模板，后者产生文档、框架窗口及视窗。这种相互关系如图（1）所示：

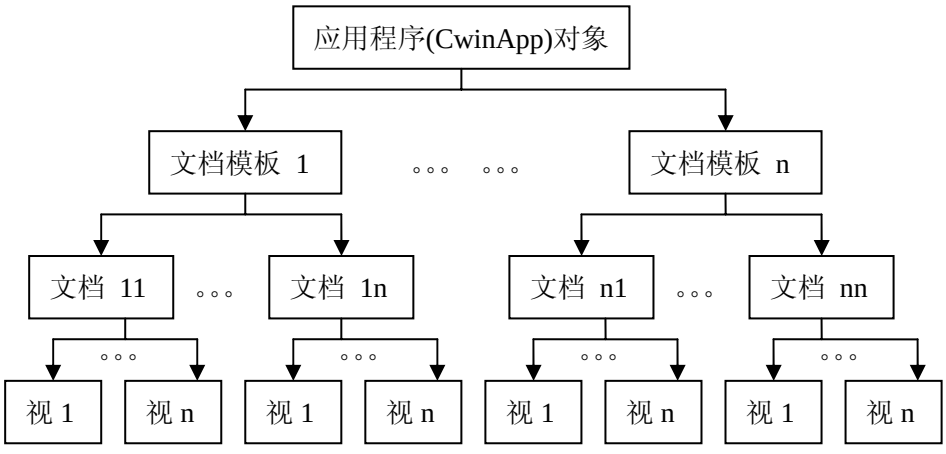


图（1）、应用程序、文档模板、文档、框架窗口及视窗对象间的关系

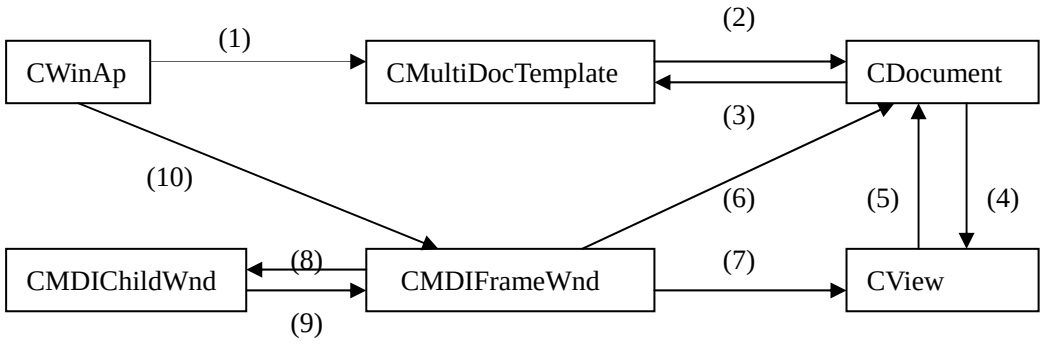
从用户的角度来看，“视”实际上是一个普通的窗口。象其他基于 Widnows 应用的窗口一样，人们可以改变它的尺寸，对它进行移动，也可以随时关闭它。若从程序员的角度来看，视实际上是一个从 MFC 类库中的 Cview 类所派生出的类的对象。文档对象是用来保存数据的，而视对象是用来显示数据的，并且允许对数据进行编辑。SDI 或 MDI 的文档类是由 Cdocument 类派生出来的，它可以有一个或多个视类，而这些视类最终都是由 Cview 类派生出来的。视对象只有一个与之相联系的文档对象，它所包含的 CView::GetDocument 函数允许应用在视中得到与之相联系的文档，据此，应用程序可以对文档类成员函数及公共数据成员进行访问。如果视对象接受到了一条消息，表示用户在编辑控制中输入了新的数据，此时，视就必须通知文档对象对其内部数据进行相应的更新。

如果文档数据发生了变化，则所有的视都必须被通知到，以便它们能够对所显示的数据进行相应的更新。Cdocument::UpdateAllViews 函数即可完成此功能。当该函数被调用时，派生视类的 CView::OnUpdate 函数被触发。通常 OnUpdate 函数要对文档进行访问，读取文档数据，然后再对视的数据成员或控制进行更新，以便反映出文档的变化。另外，还可以利用 OnUpdate 函数使视的部分客户区无效，以便触发 Cview::OnDraw 函数，利用文档数据来重新对窗口进行绘制。

在 MDI 应用程序中，可以处理多个文档类型，即多个文档模板，每个模板又可以有多个文档，每个文档又可以多视显示。为管理方便，上一级往往保留了下一级的指针列表。如图（2）所示：



图（2）、MDI 中的指针列表



图（3）。存取关系（箭头表示获得关系）

解释如下：

（1）、每个应用程序类(CwinApp 的派生类)都保留并维护了一份所有文档模板的指针列表，这是一个链表结构。应用程序为所要支持的每个文档类型动态分配一个 CMultiDocTemplate 对象，

```
CmultiDocTemplate(UINT nIDResource,
                  CruntimeClass * pDocClass,
                  CruntimeClass * pFrameClass,
                  CruntimeClass * pViewClass );
```

并在应用程序类的 CWinApp::InitInstance 成员函数中将每个 CMultiDocTemplate 对象传递给 CWinApp::AddDocTemplate。 该函数将一个文档模板加入到应用程序可用文档模板的列表中。函数原形为：

```
void AddDocTemplate(CdocTemplate * pTemplate);
```

应用程序可以用 CWinApp::GetFirstDocTemplatePostion 获得应用程序注册的第一个文档模板的位置，利用该值来调用 CWinApp::GetNextDocTemplate 函数，获得第一个 CDocTemplate 对象指针。函数原形如下：

```
POSITION GetFirstDocTemplate( ) const;
CDocTemplate *GetNextDocTemplate( POSITION & pos ) const;
```

第二个函数返回由 pos 标识的文档模板。POSITION 是 MFC 定义的一个用于迭代或对象指针检索的值。通过这两个函数，应用程序可以遍历整个文档模板列表。如果被检索的文档模板是模板列表中的最后一个，则 pos 参数被置为 NULL。

（2）、一个文档模板可以有多个文档，每个文档模板都保留并维护了一个所有对应文档的指针列表。应用程序可以用 CDocTemplate::GetFirstDocPosition 函数获得与文档模板相关的文档集合中第一个文档的位置，并用 POSITION 值作为 CDocTemplate::GetNextDoc 的参数来重复遍历与模板相关的文档列表。函数原形为：

```
viaual   POSITION  GetFirstDocPosition( ) const = 0;
visual   Cdocument *GetNextDoc(POSITION & rPos) const = 0;
```

如果列表为空，则 rPos 被置为 NULL.

（3）、在文档中可以调用 CDocument::GetDocTemplate 获得指向该文档模板的指针。函数原形如下：

```
CDocTemplate * GetDocTemplate ( ) const;
```

如果该文档不属于文档模板管理，则返回值为 NULL。

（4）、一个文档可以有多个视。每一个文档都保留并维护一个所有相关视的列表。CDocument::AddView 将一个视连接到文档上，将该视加入到文档相联系的视的列表中，并将视的文档指针指向该文档。当有 File/New、File/Open、Windows/New 或 Window/Split 的命令而将一个新创建的视的对象连接到文档上时， MFC 会自动调用该函数，框架通过文档/视的结构将文档和视联系起来。当然，程序员也可以根据自己的需要调用该函数。

Virtual POSITION GetFirstViewPosition( ) const;

Virtual CViw * GetNextView(POSITION &rPosition) cosnt;

应用程序可以调用 CDocument::GetFirstViewPosition 返回与调用文档相联系的视的列表中的第一个视的位置，并调用 CDocument::GetNextView 返回指定位置的视，并将 rPositon 的值置为列表中下一个视的 POSITION 值。如果找到的视为列表中的最后一个视，则将 rPosition 置为 NULL.

当在文档上新增一个视或删除一个视时，MFC 会调用 OnChangeViewList 函数。如果被删除的视是该文档的最后一个视，则删除该文档。

（5）、一个视只能有一个文档。在视中，调用 CView::GetDocument 可以获得一个指向视的文档的指针。函数原形如下：

CDocument *GetDocument ( ) const;

如果该视不与任何文档相，则返回 NULL.

（6）、MDI 框架窗口通过调用 CFrameWnd::GetActiveDocument 可以获得与当前活动的视相连的 CDocument 指针。函数原形如下：

virtual CDocument * GetActiveDocument( );

（7）、通过调用 CFrameWnd::GetActiveView 可以获得指向与 CFrameWnd 框架窗口连接的活动视的指针，如果是被 CMDIFrameWnd 框架窗口调用，则返回 NULL。MDI 框架窗口可以首先调用 MDIGetActive 找到活动的 MDI 子窗口，然后找到该子窗口的活动视。函数原形如下：

virtual Cdocument * GetActiveDocument( );

（8）、MDI 框架窗口通过调用 CFrameWnd::GetActiveFrame, 可以获得一个指向 MDI 框架窗口的活动多文档界面子窗口的指针。

（9）、CMDIChildWnd 调用 GetMDIFrame 获得 MDI 框架窗口(CMDIFrameWnd)。

（10）、CWinApp 调用 AfxGetMainWnd 得到指向应用程序的活动主窗口的指针。

下面一段代码，就是利用 CDocTemplate、CDocument 和 CView 之间的存取关系，遍历整个文档模板、文档以及视。

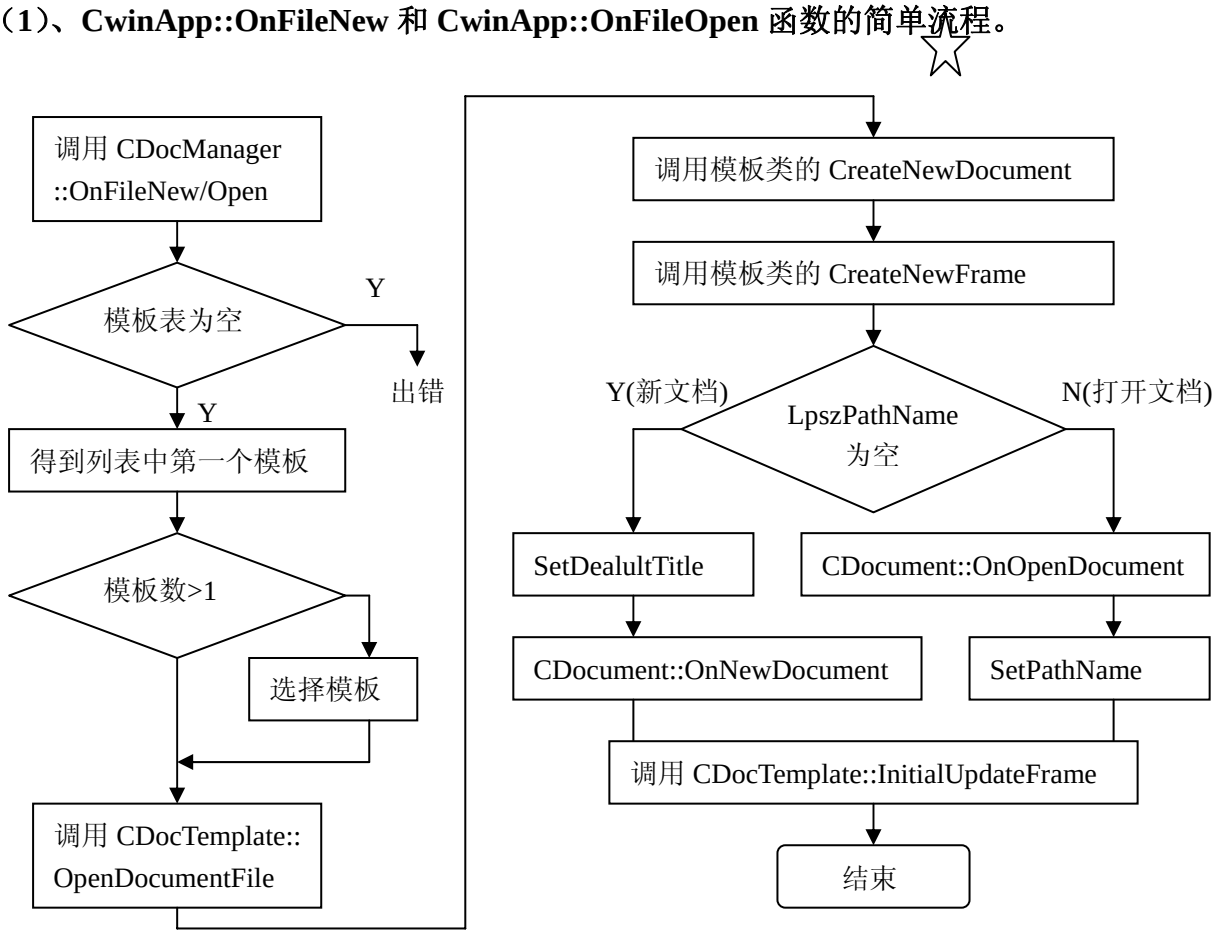
```
CMyApp * pMyApp = (CMyApp *)AfxGetApp();
POSITION    p = pMyApp->GetFirstDocTemplatePosition();
while(p!= NULL) {
    CDocTemplate * pDocTemplate = pMyApp->GetNextDocTemplate(p);
    POSITION p1 = pDocTemplate->GetFirstDocPosition();
    while(p1 != NULL) {
        CDocument * pDocument = pDocTemplate->GetNextDoc(p1);
        POSITION p2 = pDocument->GetFirstViewPosition();
        while(p2 != NULL) {
            CView * pView = pDocument->GetNextView(p2);
        }
    }
}
```

（图 4）、遍历整个文档模板、文档和视

在应用程序的任何地方，程序员都可以调用 AfxGetApp( )获得应用程序的对象指针。由于本文着重介绍文档/视的关系，至于框架窗口之间的关系没能列全，读者可以查相应的文档。

（三）、了解 CwinApp::OnFileNew、CwinApp::OnFileOpen 和 Window/New 的程序流程。

（1）、CwinApp::OnFileNew 和 CwinApp::OnFileOpen 函数的简单流程。



（图 5）、File New/Open 流程图

在 CWinApp::OnFile/new 或 CwinApp::OnFileOpen 函数中，核心操作是 CDocTemplate::OpenDocument 函数。其函数原型为：

virtual CDocument* CDocTemplate::OpenDocumentFile(LPCTSTR lpszPathName, BOOL bMakeVisible = TRUE) = 0;

图（4）中星号标注之后即是该函数的流程，简要介绍如下：

（1）、CDocTemplate::CreateNewDocument 函数创建一个新文档，其类型与文档模板相关，并通过函数 CDocTemplate::AddDocument 加入该文档模板的文档指针列表中。此时，文档类的构造函数被执行，程序可以在此进行文档的初始化。

（2）、函数 CDocTemplate::CreateNewFrame 调用 MDI 子窗口类(CMDIChildWnd)的构造函数,生成 MDI 子窗口对象。接着调用 CMDIChildWnd::PreCreateWindow。然后，生成一个 CCreateContext 对象,（CcreateContext 是 MFC 框架所使用的一种结构，它将构成文档和视的组件联系起来。后文将详细介绍之。）并将该对象值传给 CMDIChildWnd::OnCreateClient 函数。MFC 调用此函数，用 CCreateContext 对象提供的信息创建一个或多个 CView 对象。此时，各视的构造函数被依次调用。

（3）、接着，判断 lpszPathName 是否为空。分为两种情况：

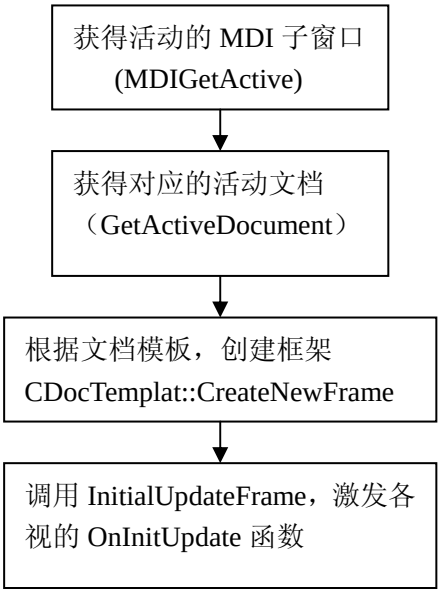
(a)、若为空，则表明要创建一个新文档：调用 SetDefaultTitle 函数装载文档的缺省标题，并显示在文档的标题栏中；然后执行 CDocument::OnNewDocument。该函数调用 DeleteContents 以保证文档为空，然后置新文档为清洁。可以重载该函数。

(b)、否则，表明要打开一个已存在的文档：调用 CDocument::OnOpenDocument 打开指定的文件；执行 DeleteContext，保证文档为空；调用 CObject::Serialize 读入该文件的内容。（程序员可在此进行文件的读入操作。当然，也可以在 CDocument::OnOpenDocument 中读入文件）。然后置文档为清洁；最后，调用 CDocTemplate::SetPathName，并把文件名加入到最近文件列表中。

（4）、调用 CDocTemplate::InitialUpdateFrame 函数，使框架窗口中的各个视收到 OnInitialUpdate 调用。框架窗口的主视（子窗 ID 等于 AFX_IDW_PANE_FIRST 的视）被激活。程序员可以在此对视对象进行初始化。

（2）、Window/New 命令的程序流程

当主框架窗口上有子窗口时，选择 Window/New 命令可以生成该活动子窗口的影象。它们有相同的文档模板、相同的文档。其流程如下：



执行 Window/New 的过程与 File/New 的过程差不多。所不同的是，File/New 须要创建一个新文档，而 Window/New 则是获得已存在的 MDI 子窗口的文档。因此以前存在的视和 New 以后生成的视均为该文档的视，都是该文档的内容的显示。当调用 CDocument::UpdateAllViews 函数时，它们(视)的 OnUpdate 函数都将被激活。此时，在该文档的视指针列表中，将有多于一个的视（具体数目视 Window/New 执行的次数而定）。读者可以利用（图 3）中的代码跟踪程序结果。

（四）、几种情况的讨论

上面，笔者就 MFC 中文档/视的关系进行了分析，下面，笔者将结合具体情况进行讨论：

（1）、如何根据自己的要求来选择文档模板，及相应的视和文档。

在通常的 MDI 应用程序中，只有一个文档模板，程序员只能打开一种类型的文档。因此，程序员只要调用 File/New 或者 File/Open 创建或者打开文档即可，至于文档、视和框架窗口之间的关系，由文

档模板在幕后控制，不须要对文档模板进行操作。但是，如果应用程序需要处理多种类型的文档，并且何时打开何种文档均需程序员手工控制，此时，程序员必须对文档模板进行编程。

例如，笔者需要处理 AVI 和 BMP 两种文件类型。AVI 和 BMP 的数据存放格式不同，不能用同一的数据结构来描述，因此，把它们的数据都存入一个文档是不合适的。同时，由于 AVI 是图象序列，BMP 仅是一幅图象，它们的显示是肯定不一样的，即它门的视不同。基于此，笔者决定分别建立两套文档模板，两套框架窗口，两套文档和两套视，分别用于 AVI 和 BMP 的数据存放和显示。程序可以根据用户选择的文件名来分别处理 AVI 和 BMP。具体步骤如下：

（Step 1）、在应用程序类(CWinApp)的派生类中增加文档模板成员变量，以便对文档模板进行操作。

```
class C3dlcsApp : public CWinApp
{           。。。  。。。
    public:
        CMultiDocTemplate * m_pAVIDocTemplate;
        CMultiDocTemplate * m_pBMPDocTemplate;
}
```

（Step 2）、在主框架中增加菜单响应：

```
void CMainFrame::OnFileOpen()  {
    CFileDialog my(true);
    if(my.DoModal()==IDOK)  {
        CString FileName = my.GetPathName();
        CString FileExt = my.GetFileExt();

        if((FileExt == "AVI") || (FileExt == "avi")) {
            CMyApp * pMyApp = (CMyApp *)AfxGetApp();
            CMultiDocTemplate*pAVIDocTemplate=pMyApp->m_pAVIDocTemplate;
            pAVIDocTemplate->OpenDocumentFile(FileName);
        }
        else if((FileExt == "BMP") || (FileExt == "bmp")) {
            CMyApp * p3dlcsApp = (CMyApp *)AfxGetApp();
            CMultiDocTemplate* pDATDocTemplate=pMyApp->m_pBMPDocTemplate;
            pDATDocTemplate->OpenDocumentFile(FileName);
        }
        else {
            AfxMessageBox("Yor select a file not supported!");
            return;
        }
    }
}
```

笔者把用户输入文件名的后缀作为分支条件，如果是 AVI 文件，则先获得关于 AVI 文件的文档模板，然后调用 CDocTemplate::OpenUpdateFrame (lpszFileName)函数打开此文档。正如前面所分析，此函数将依次生成新文档，新框架，在 CMDIChildWnd::OnCreateClient 中创建视，最后向框架中所有的视发送初始化消息，使其显示在屏幕上。如果是 BMP 文件，操作类似。

当然，程序员也可以在程序的任何位置实现此操作：通过全局函数 AfxGetApp 获得应用程序对象指针，从而获得相应的文档模板指针。

由于由 AppWizard 生成的应用程序会缺省调用 CWinApp::OnFileNew，所以当程序开始执行时，

会在主框架上显示一个新的空窗口。如果想去掉这个空窗口，只须重载 CWinApp::OnFileNew 函数，不许要任何代码，即可。

（2）、切分窗口与文档/视结构

一个文档可以有多个视，切分窗口即是表示多视的一种方法。切分窗口是通过类 CSplitterWnd 来表示的，对 Window 来说，CSplitterWnd 对象是一个真正的窗口，它完全占据了框架窗口的客户区域，而视窗口则占据了切分窗口的窗片区域。切分窗口并不参与命令传递机制，（窗片中）活动的视窗从逻辑上来看直接被连到了它的框架窗口中。

切分窗口可以分为动态和静态两种。前者较简单，本文仅讨论后者。创建切分窗口的步骤如下：

（Step 1）、在自己的框架窗口中声明成员变量，用以对切分窗口进行操作。

```
class CMyFrame : public CMDIChildWnd
{      。。。  。。。
    CSplitterWnd m_Splitter;
    CSplitterWnd m_Splitter2;
}
```

（Step 2）、重载 CMDIChildWnd::OnCreateClient 函数，创建切分窗口。

```
BOOL CMyFrame::OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext* pContext)
{
    BOOL btn = m_Splitter.CreateStatic(this,1,2);
    btn |= m_Splitter.CreateView(0,0, RUNTIME_CLASS(CAVIDispView),
                                 CSize(100,100), pContext);
    m_Splitter2.CreateStatic(&m_Splitter,
                             2, 1,
                             WS_CHILD | WS_VISIBLE | WS_BORDER,
                             m_Splitter.IdFromRowCol(0, 1));
    btn |= m_Splitter2.CreateView(0, 0, RUNTIME_CLASS(CBMPView),
                                  CSize(100,100), pContext);
    btn |= m_Splitter2.CreateView(1, 0, RUNTIME_CLASS(CAVIView),
                                  CSize(100,100), pContext);
    return btn;
    //return CMDIChildWnd::OnCreateClient(lpcs, pContext);
}
```

CFrameWnd::OnCreateClient 函数原形为：

```
virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs, CcreateContext * pContext);
```

缺省的 CMDIChildWnd::OnCreateClient 函数根据 pContext 参数提供的信息，调用 CFrameWnd::CreateView 函数创建一个视。可以重载该函数，加载 CCreateContext 对象中传递的值，或改变框架窗口主客户区中控制的创建方式。在上面的程序中，笔者 创建了 3 个切分窗口。比如打开了一个名为“a.avi”的文档，此时该文档将有 3 个视，一个框架窗口。如果执行了 Window/New 操作，则此时有一个文档，6 个视和 2 个框架窗口。若该文档调用 CDocument::UpdateAllViews 函数，则这 6 个视的 CView::OnUpdate 函数都会被激发。

(3)、关于 CCreateContext 的讨论。

CCreateContext 是 MFC 框架所使用的一种结构，它将构成文档/视的组件联系起来。这个结构包括指向文档的指针，框架窗口，视以及文档模板，它还包含一个指向 CRuntimeClass 的指针，以指明所创建的视的类型。其数据成员如下：

m_pNewViewClass：指向创建上下文的视的 CRuntimeClass 的指针。

m_pCurrentDoc：指向文档对象的指针，以和新视联系起来。

m_pNewDocTemplate：指向与框架窗口的创建相联系文档模板的指针。

m_pLastView：指向已存在的视，它是新产生的视的模型。

m_pCurrentFrame：指向已存在的框架窗口，它是新产生的框架窗口的模型。

程序员可以通过改变 CCreateContext 对象的值，来创建更加灵活的视。由于过程较复杂，笔者不再赘许叙，读者可参阅相关的 Visual C++ Help 文档。

（五）、结束语

Visual C++5.0 的文档/视结构代表了一种新的程序设计方式，其核心是文档与视的分离，即数据存放与显示（操作）的分离。在 MFC 类库中，各个对象之间的关系很复杂，但，只要深入了解后，会发现它们之间是相互联系的，可以相互存取的。如果大家想设计出灵活、健壮的应用程序，就必须深入了解 MFC。跟踪原代码就是一个较好的方法。文档/视的关系的确非常复杂，如果能知道每个函数是在哪调用的，执行了何种操作，就能游人刃有余，写出优美的应用程序。