CHAPTER 第5章 框架窗口、文档和视图

在文档应用程序框架中，框架窗口、文档和视图占有非常重要的地位。框架窗口用作文档和视图的容器。文档代表一个数据单元，用户可使用“文件”菜单的“打开”和“保存”命令进行文档数据操作。视图是框架窗口的子窗口，它与文档紧密相联，是用户与文档之间的交互接口。本章将详细讨论框架窗口、文档和视图的使用方法和技巧。

5.1 框架窗口

框架窗口可分为两类：一类是应用程序主框架窗口，另一类是文档窗口。

5.1.1 主框架窗口和文档窗口

主框架窗口是应用程序直接放置在桌面(DeskTop)上的那个窗口，每个应用程序只能有一个主框架窗口，主框架窗口的标题栏上往往显示应用程序的名称。主框架窗口负责管理各个用户交互对象(包括菜单、工具栏、状态栏以及加速键)并根据用户操作相应地创建或更新文档窗口及其视图。

文档窗口对于单文档应用程序来说，它和主框架窗口是一致的，即主框架窗口就是文档窗口；而对于多文档应用程序来说，文档窗口是主框架窗口的子窗口，如图 5.1 所示。

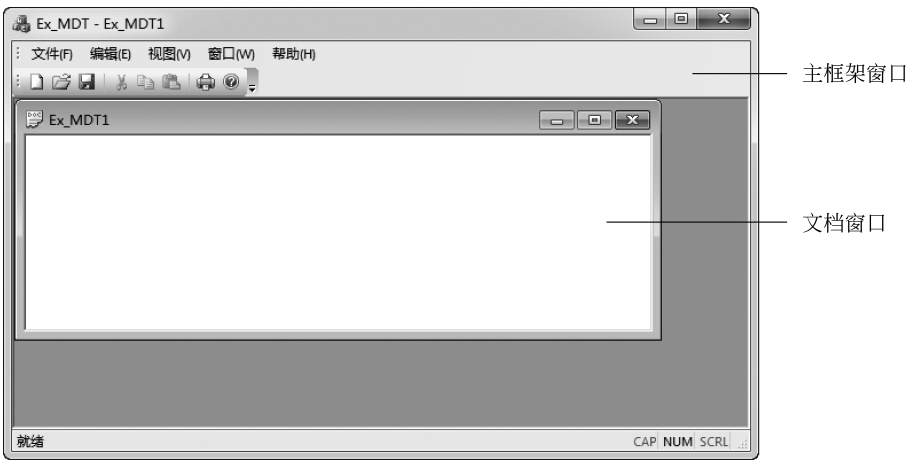

图 5.1 多文档应用程序的框架窗口

文档窗口一般都有相应的可见边框，它的客户区（除了窗口标题栏、边框外的区域）是由相应的视图来构成的，因此可以说视图是文档窗口内的子窗口。文档窗口时刻跟踪当前处于活动状态的视图的变化，并将用户或系统产生的消息传递给当前活动视图。

5.1.2　框架窗口初始状态的改变

“MFC 应用程序向导”为每一个文档应用程序的框架窗口设置了相应的大小和位置，但默认的窗口状态有时并不那么令人满意，这时就需要对窗口状态进行适当的改变。

当文档应用程序运行时，Windows 会自动调用应用程序框架内部的 WinMain()函数，并自动查找该应用程序类的全局变量 theApp，然后自动调用用户应用程序类的虚函数 InitInstance()，该函数会进一步调用相应的函数来完成主窗口的构造和显示工作，如下面的代码（以标准的视觉样式单文档应用程序 Ex_SDT 为例）。

```
BOOL CEx_SDTApp::InitInstance()
{...
    m_pMainWnd->ShowWindow(SW_SHOW);          //显示窗口
    m_pMainWnd->UpdateWindow();               //更新窗口
    return TRUE;
}
```

其中，m_pMainWnd 是主框架窗口指针变量，ShowWindow()是 CWnd 类的成员函数，用来按指定的参数显示窗口，该参数的值如表 5.1 所示。

表 5.1　ShowWindow ()函数的参数值

参　数　值	含　义
SW_HIDE	隐藏此窗口并将激活状态移交给其他窗口
SW_MINIMIZE	将窗口最小化并激活系统中的顶层窗口
SW_RESTORE	激活并显示窗口，且恢复到原来的大小和位置
SW_SHOW、SW_SHOWNORMAL	用当前的大小和位置激活并显示窗口
SW_SHOWMAXIMIZED、SW_MAXIMIZE	激活窗口并使之最大化
SW_SHOWMINIMIZED	激活窗口并使之最小化
SW_SHOWMINNOACTIVE	窗口显示成为一个图标并保留其激活状态
SW_SHOWNA	用当前状态显示窗口
SW_SHOWNOACTIVATE	用最近的大小和位置状态显示窗口并保留其激活状态

通过指定 ShowWindow()函数的参数值可以改变窗口显示状态。例如，下面的代码是将窗口的初始状态设置为“最大化”。

```
BOOL CEx_SDTApp::InitInstance()
{...
    m_pMainWnd->ShowWindow(SW_MAXIMIZE);      //最大化
    m_pMainWnd->UpdateWindow();               //更新窗口
```

```
    return TRUE;
}
```

5.1.3 窗口样式

在 Visual C++ 中,窗口样式决定了窗口的外观及功能,通过样式设置可以增加或减少窗口中所包含的功能,这些功能一般都是由系统内部定义的,不需要用户再去编程实现。

窗口样式通常有一般(以 WS_为前缀)和扩展(以 WS_EX_为前缀)两种形式。这两种形式的窗口样式可在函数 CWnd::Create()或 CWnd::CreateEx()参数中指定,其中,CWnd::CreateEx()函数可同时支持以上两种样式,而 CWnd::Create()只能指定窗口的一般样式。需要说明的是,对于控件和对话框这样的窗口来说,它们的窗口样式可直接通过其属性对话框来设置。常见的一般窗口样式如表 5.2 所示。

表 5.2 窗口的一般样式

样 式	含 义
WS_BORDER	窗口含有边框
WS_CAPTION	窗口含有标题栏(含边框),但它不能和 WS_DLGFRAME 组合
WS_CHILD	创建子窗口,它不能和 WS_POPUP 组合
WS_DISABLED	窗口最初时是禁用的
WS_DLGFRAME	窗口含有双边框,但没有标题
WS_GROUP	此样式被控件组中第一个控件窗口指定
WS_HSCROLL	窗口含有水平滚动条
WS_MAXIMIZE	窗口最初时处于最大化
WS_MAXIMIZEBOX	在窗口的标题栏上含有“最大化”按钮
WS_MINIMIZE	窗口最初时处于最小化,它只和 WS_OVERLAPPED 组合
WS_MINIMIZEBOX	在窗口的标题栏上含有“最小化”按钮
WS_OVERLAPPED	创建可覆盖窗口,一个覆盖窗口通常有一个标题和边框
WS_OVERLAPPEDWINDOW	创建覆盖窗口,包含 WS_OVERLAPPED、标题、系统菜单、可调粗框架、“最小化”按钮和“最大化”按钮
WS_POPUP	创建弹出子窗口,不能和 WS_CHILD 组合。仅用于 CreateEx()函数
WS_POPUPWINDOW	创建弹出窗口,包含 WS_POPUP、边框。当 WS_CAPTION 和 WS_POPUPWINDOW 样式组合时才能使系统菜单可见
WS_SYSMENU	窗口的标题栏上含有“系统菜单”,它仅用于含有标题栏的窗口
WS_TABSTOP	用户可以用 Tab 键选择控件组中的下一个控件
WS_THICKFRAME	窗口含有边框,并可调整窗口的大小(可调粗框架)
WS_VISIBLE	窗口最初是可见的
WS_VSCROLL	窗口含有垂直滚动条

需要说明的是,除了上述样式外,框架窗口还有以下 3 个自己的样式。它们都可以在

PreCreateWindow()重载函数中指定(后面有应用示例)。

(1) FWS_ADDTOTITLE：该样式指定一个文档名添加到框架窗口标题中，例如，在 Ex_MDT-Ex_MDT1 中，Ex_MDT1 是文档名。对于单文档应用程序来说，默认的文档名是“无标题”。

(2) FWS_PREFIXTITLE：该样式使得框架窗口标题中的文档名显示在应用程序名之前。例如，若未指定该样式时的窗口标题为 Ex_MDT-Ex_MDT1，当指定该样式后就变成了 Ex_MDT1-Ex_MDT。

(3) FWS_SNAPTOBARS：该样式用来调整窗口的大小，使它刚好包含框架窗口中的控制栏(如工具栏)。

5.1.4　窗口样式设置

窗口样式既可以通过“MFC 应用程序向导”在创建时设置，也可以在主框架窗口或文档窗口类的 PreCreateWindow()函数中修改 CREATESTRUCT 结构，或是可以调用 CWnd 类的成员函数 ModifyStyle()和 ModifyStyleEx()来更改。

1. 在“MFC 应用程序向导”创建时设置

在用“MFC 应用程序向导”创建单文档或多文档应用程序过程中，其“用户界面功能”页面有一个“主框架样式”栏，如图 5.2 所示。其中，“初始化状态栏”选项用来向应用程序添加状态栏，并对状态栏窗格数组进行初始化，状态栏上最右边的窗格分别显示出 Caps Lock、Num Lock 和 Scroll Lock 键的状态。“主框架样式”栏其他各选项的选中含义所对应的样式可从表 5.2 中找到，但在“用户界面功能”页面中，只能设定少数几个窗口样式。

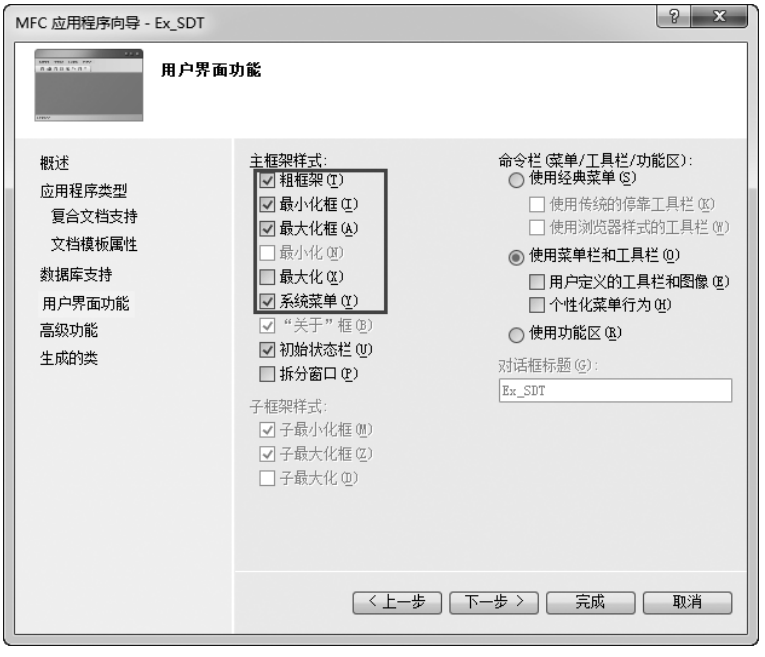

图 5.2　“用户界面功能”页面

2. 修改 CREATESTRUCT 结构

当创建窗口之前,将自动调用 PreCreateWindow()虚函数。在用“MFC 应用程序向导”创建文档应用程序框架时,MFC 已为主框架窗口或文档窗口类自动重载了该虚函数。可以在此函数中通过修改 CREATESTRUCT 结构来设置窗口的绝大多数样式。

例如,在单文档应用程序中,框架窗口默认的样式是 WS_OVERLAPPEDWINDOW 和 FWS_ADDTOTITLE 的组合,更改其样式可用下列的代码。

```
BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if(!CFrameWndEx::PreCreateWindow(cs))     return FALSE;
    //新窗口不带有"最大化"按钮,即禁用最大化
    cs.style &= ~WS_MAXIMIZEBOX;
    cs.style &= ~FWS_ADDTOTITLE;          //取消 FWS_ADDTOTITLE 样式
    //将窗口的大小设为 1/3 屏幕并居中
    cs.cy = ::GetSystemMetrics(SM_CYSCREEN) / 3;
    cs.cx = ::GetSystemMetrics(SM_CXSCREEN) / 3;
    cs.y = ((cs.cy * 3) -cs.cy) / 2;
    cs.x = ((cs.cx * 3) -cs.cx) / 2;
    return TRUE;
}
```

代码中,前面有“::”域作用符的函数是指全局函数,一般都是一些 API 函数。“cs.style &= ~WS_MAXIMIZEBOX;”中的“~”是按位取“反”运算符,它将 WS_MAXIMIZEBOX 的值按位取反后,再和 cs.style 值按位“与”,其结果是将 cs.style 值中的 WS_MAXIMIZEBOX 标志位清零。

再如,对于多文档应用程序,文档窗口的样式可用下列的代码更改。

```
BOOL CChildFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if(!CMDIChildWndEx::PreCreateWindow(cs))     return FALSE;
    cs.style &= ~WS_MAXIMIZEBOX;          //创建不含有"最大化"按钮的子窗口
    return TRUE;
}
```

注意:若上述代码运行的结果与预想的不一样,建议先运行 regedit 命令,进入注册表编辑区,找到 HKEY_CURRENT_USER\Software\应用程序向导生成的本地应用程序,删除里面的整个“xxx”项(xxx 是创建时指定的应用程序项目名称)。

3. 使用 ModifyStyle 和 ModifyStyleEx

CWnd 类中的成员函数 ModifyStyle()和 ModifyStyleEx()也可用来更改窗口的样式,其中,ModifyStyleEx()还可更改窗口的扩展样式。这两个函数具有相同的参数,其原型如下。

```
BOOL ModifyXXXX(DWORD dwRemove, DWORD dwAdd, UINT nFlags = 0);
```

其中，参数 dwRemove 用来指定需要删除的样式，dwAdd 用来指定需要增加的样式，nFlags 表示 SetWindowPos 的标志，0（默认值）表示更改样式的同时不调用 SetWindowPos() 函数。

由于框架窗口在用“MFC 应用程序向导”创建时不能直接设定其扩展样式，因此只能通过调用 ModifyStyle() 函数来进行。例如，将 CChildFrame 类的“属性窗口”切换到“重写”页面，找到并添加虚函数 OnCreateClient() 的重写（重载），然后在函数中添加下列代码。

```
BOOL CChildFrame::OnCreateClient(LPCREATESTRUCT lpcs,
                                 CCreateContext* pContext)
{
    ModifyStyle(0, WS_VSCROLL, 0);
    return CMDIChildWndEx::OnCreateClient(lpcs, pContext);
}
```

这样，当文档子窗口创建客户区时就会调用虚函数 OnCreateClient()。编译运行，结果如图 5.3 所示。

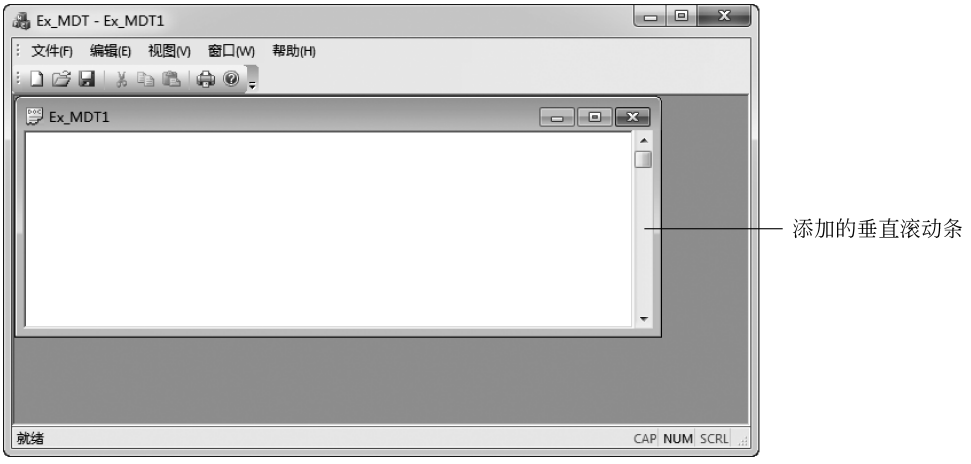

图 5.3　为文档子窗口添加垂直滚动条

5.1.5　改变窗口大小和位置

用 CWnd 类的成员函数 SetWindowPos() 或 MoveWindow() 可以改变窗口的大小和位置。

SetWindowPos() 是一个非常有用的函数，它不仅可以改变窗口的大小、位置，而且可以改变窗口在堆栈排列的次序（Z 次序），这个次序是根据它们在屏幕出现的先后来确定的。

```
BOOL SetWindowPos(const CWnd* pWndInsertAfter, int x, int y,
                  int cx, int cy, UINT nFlags);
```

其中，参数 pWndInsertAfter 用来指定窗口对象指针，它可以是下列预定义值。

```
wndBottom          //将窗口放置在 z 次序中的底层
wndTop             //将窗口放置在 z 次序中的顶层
wndTopMost         //设置最顶窗口
wndNoTopMost       //将窗口放置在所有最顶层的后面,若此窗口非最顶窗口,则无效
```

x 和 y 表示窗口新的左上角坐标,cx 和 cy 分别表示窗口新的宽度和高度,nFlags 表示窗口新的大小和位置方式,如表 5.3 所示。

表 5.3 常用 nFlags 值及其含义

nFlags 值	含 义
SWP_HIDEWINDOW	隐藏窗口
SWP_NOACTIVATE	不激活窗口。如该标志没有被指定,则依赖 pWndInsertAfter 参数
SWP_NOMOVE	不改变当前的窗口位置(忽略 x 和 y 参数)
SWP_NOOWNERZORDER	不改变父窗口的 Z 次序
SWP_NOREDRAW	不重新绘制窗口
SWP_NOSIZE	不改变当前的窗口大小(忽略 cx 和 cy 参数)
SWP_NOZORDER	不改变当前的窗口 Z 次序(忽略 pWndInsertAfter 参数)
SWP_SHOWWINDOW	显示窗口

函数 CWnd::MoveWindow()也可用来改变窗口的大小和位置,与 SetWindowPos()函数不同的是,使用 MoveWindow()函数还须指定窗口的大小。

```
void MoveWindow(int x, int y, int nWidth, int nHeight, BOOL bRepaint = TRUE);
void MoveWindow(LPCRECT lpRect, BOOL bRepaint = TRUE);
```

其中,参数 x 和 y 表示窗口新的左上角坐标,nWidth 和 nHeight 表示窗口新的宽度和高度,bRepaint 用于指定窗口是否重绘,lpRect 表示窗口新的大小和位置。

作为示例,这里将使用上述两个函数把主窗口移动到屏幕的(100,100)处(代码添加在 InitInstance()中 return TRUE 语句之前)。

```
//使用 SetWindowPos()函数的示例
m_pMainWnd->SetWindowPos(NULL,100,100,0,0,SWP_NOSIZE|SWP_NOZORDER);
//使用 MoveWindow()函数的示例
CRect rcWindow;
m_pMainWnd->GetWindowRect(rcWindow);
m_pMainWnd->MoveWindow(100,100,rcWindow.Width(),rcWindow.Height(),TRUE);
```

其中,CRect 是一个矩形类,GetWindowRect()是一个 CWnd 类的成员函数,用来获取窗口在屏幕的位置和大小。当然,改变窗口的大小和位置的 CWnd 成员函数还不止以上两个。例如,CenterWindow()函数是使窗口居于父窗口中央,就像下面的代码。

```
CenterWindow(CWnd::GetDesktopWindow());          //将窗口置于屏幕中央
AfxGetMainWnd()->CenterWindow();                 //将主框架窗口居中
```

5.2　文 档 模 板

用“MFC 应用程序向导”创建的单文档(SDI)或多文档(MDI)应用程序项目均包含应用程序类、文档类、视图类和框架窗口类,这些类通过文档模板来有机地联系在一起。

5.2.1　文档模板类

文档应用程序框架是在程序运行时就开始构造的,在一个单文档应用程序(设项目名为 Ex_SDT)的应用程序类 InitInstance()函数中,可以看到这样的代码。

```
BOOL CEx_SDTApp::InitInstance()
{
    ...
    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,                  //资源 ID
        RUNTIME_CLASS(CEx_SDTDoc),      //文档类
        RUNTIME_CLASS(CMainFrame),      //主框架窗口类
        RUNTIME_CLASS(CEx_SDTView));    //视图类
    if(!pDocTemplate)    return FALSE;
    AddDocTemplate(pDocTemplate);
    ...
    return TRUE;
}
```

代码中,pDocTemplate 是类 CSingleDocTemplate 的指针对象。CSingleDocTemplate 是一个单文档模板类,它的构造函数中有 4 个参数,分别表示菜单和加速键等的资源 ID 以及三个由宏 RUNTIME_CLASS 指定的运行时类对象。AddDocTemplate()是类 CWinApp 的一个成员函数,当调用了该函数后,就建立了应用程序类、文档类、视图类以及主框架类之间的相互联系。

类似地,多文档模板类 CMultiDocTemplate 的构造函数也有相同的定义,如下面的代码(设项目名为 Ex_MDT)。

```
BOOL CEx_MDTApp::InitInstance()
{
    ...
    CMultiDocTemplate* pDocTemplate;
    pDocTemplate = new CMultiDocTemplate(
        IDR_Ex_MDTTYPE,                 //资源 ID
        RUNTIME_CLASS(CEx_MDTDoc),      //文档类
        RUNTIME_CLASS(CChildFrame),     //子框架窗口类
        RUNTIME_CLASS(CEx_MDTView));    //视图类
```

```
    if(!pDocTemplate)
        return FALSE;
    AddDocTemplate(pDocTemplate);
    //创建主框架窗口
    CMainFrame* pMainFrame = new CMainFrame;
    if(!pMainFrame || !pMainFrame->LoadFrame(IDR_MAINFRAME))
    {
        delete pMainFrame;
        return FALSE;
    }
    m_pMainWnd = pMainFrame;
    …
    return TRUE;
}
```

由于多文档模板只是用来建立资源、文档类、视图类和子框架窗口(文档窗口)类之间的关联,因而对于多文档主框架窗口的创建需要额外的代码。上述代码中,LoadFrame()是CFrameWnd(CFrameWndEx)类成员函数,用来加载与主框架窗口相关的菜单、加速键、图标等资源。需要说明的是,多文档主框架窗口的创建应在多文档模板创建后进行,以便MFC 程序框架将多文档模板和多文档主框架窗口建立联系。

5.2.2 文档模板字符串资源

在“MFC 应用程序向导”创建的文档应用程序资源中,许多资源标识符都是 IDR_MAINFRAME,这就意味着这些具有同名标识的资源将被框架自动加载到应用程序中。其中,String Table(字符串)资源列表中也有一个 IDR_MAINFRAME 项,它是用来标识文档类型、标题等内容的,称为“文档模板字符串资源”。其内容如下(设创建的单文档应用程序为 Ex_SDT)。

```
Ex_SDT\n\nEx_SDT\n\n\nExSDT.Document\nEx_SDT.Document
```

可以看出,IDR_MAINFRAME 所标识的字符串被“\n”分成了 7 段子串,每段都有特定的用途,其含义如表 5.4 所示。实际上,文档模板字符串资源内容既可直接通过字符串资源编辑器进行修改,也可以在文档应用程序创建向导的“文档模板属性”页面来指定,如图 5.4 所示,为单文档应用程序 Ex_SDT 中的情况,图中的数字表示该项的含义与表 5.4 中对应串号的含义相同。

表 5.4 文档模板属性的含义

IDR_MAINFRAME 子串	串号	用 途
Ex_SDT\n	0	应用程序窗口标题
\n	1	文档根名。对 MDI 来说,若子窗口标题显示“Sheet1”,则其中的 Sheet 就是文档根名。若该子串为空,则文档名为默认的“无标题”

续表

IDR_MAINFRAME 子串	串号	用　　途
Ex_SDT\n	2	新建文档的类型名。若有多个文档类型，则这个名称将出现在“新建”对话框中
\n	3	通用对话框的文件过滤器正文
\n	4	通用对话框的文件扩展名
ExSDT.Document\n	5	在注册表中登记的文档类型标识
Ex_SDT.Document	6	在注册表中登记的文档类型名称(全称)

图 5.4　“文档模板属性”页面

但对于多文档应用程序(MDI)来说，上述字符串内容分别由 IDR_MAINFRAME 和 IDR_Ex_MDTTYPE(若项目名为 Ex_MDT)组成；其中，IDR_MAINFRAME 表示应用程序窗口标题，而 IDR_Ex_MDITYPE 表示后 6 项内容，它们的内容如下。

```
IDR_MAINFRAME: Ex_MDT
IDR_Ex_MDTTYPE: \nEx_MDT\nEx_MDT\n\n\nExMDT.Document\nEx_MDT.Document
```

5.3　文档序列化

用户处理的数据往往需要存盘作永久备份。将文档类中的数据成员变量的值保存在磁盘文件中，或者将存储的文档文件中的数据读取到相应的成员变量中。这个过程称为序列化(Serialize)。