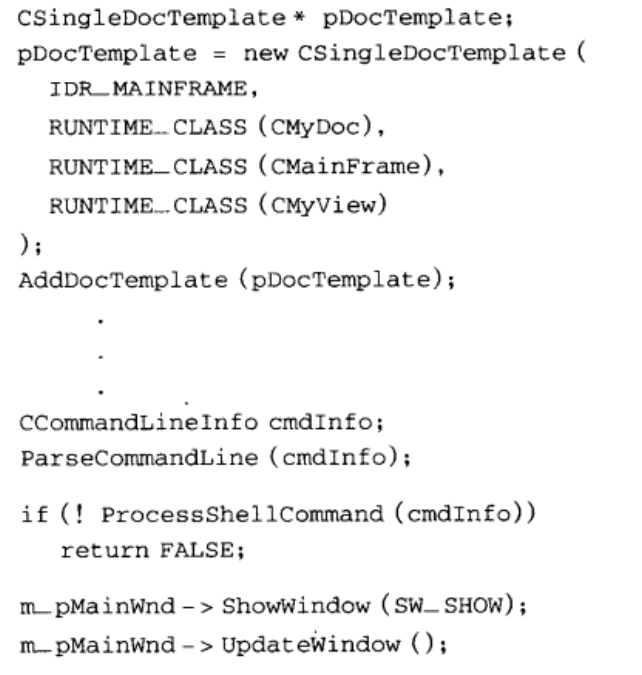
文档视图体系结构

# 文档、视图和单文档界面

## 文档视图基础知识

### InitInstance函数

SDI应用程序的InitInstance函数，含有以下内容：



首先，从MFC的CSingleDocTemplate类创建了一个SDI文档模板。该模板是SDI文档/视图应用程序的最主要的部分。它表示了用来管理应用程序数据的文档类、包含数据视图的框架窗口类、用来绘制可视数据表示的视图类。文档模板还保存了资源ID，主结构用它来加载菜单、加速键以及其他形成应用程序用户界面的资源。

其次，使用AddDocTemplate将它添加到由应用程序对象保存的文档模板列表中。用此方法注册的每个模板都定义了一个应用程序支持的文档类型。SDI应用程序只注册一个文档类型，MDI应用程序可以注册多个类型。

然后，通过CWinApp::ParseCommandLine并用反映命令行输入参数的值来初始化CCommandLineInfo对象，其中通常包含文档文件名。

然后，使用ProcessShellCommand处理命令行参数。它首先调用CWinApp::OnFileNew来启动应用程序，如果文件名没有在命令行上输入就使用空文档，或者在指定了文件名的情况下使用CWinApp::OpenDocumentFile来加载一个文档。正是在程序执行的这个阶段，主结构用保存在文档模板中的信息来创建文档、框架窗口和视图对象。初始化成功返回TRUE，否则返回FALSE。

最后在屏幕上显示应用程序的框架窗口。

在应用程序被启动，文档、框架窗口和视图被创建之后，消息循环就开始工作了。

### 文档对象

文档是程序数据的抽象表示，在数据的保存和给用户提供数据之间划分了清晰的界限。通常，文档对象为其他对象，特别是视图，提供了公用成员函数，使用它可以访问文档的数据。

CDocument的主要操作包括：

|  |  |
| --- | --- |
| 函数 | 说明 |
| GetFirstViewPosition | 返回POSITION，用来传递给GetNextView，以开始列举与文档关联的视图 |
| GetNextView | 返回CView指针，指向与文档关联的视图列表中的下一个视图 |
| GetPathName | 检索文档的文件名称和路径，例如C:\Documents\MyFile.doc；如果文档还未命名则返回空字符串 |
| GetTitle | 检索文档的标题，例如MyFile；如果文档没命名则返回空字符串 |
| IsModified | 如果文档包含未保存的数据就返回非零值，否则为零 |
| SetModifiedFlag | 设置或清除文档中已修改的标志，该标志说明文档是否包含没有保存的数据 |
| UpdateAllViews | 通过调用每个视图的OnUpdate函数来更新与文档关联的所有视图 |

每次修改了文档数据之后都要调用SetModifiedFlag。

OnUpdate是CView的保护型成员函数，所以在AfxWin.h中将CDocument声明为CView的友元。

CDocument主要的可覆盖函数：

|  |  |
| --- | --- |
| 函数 | 说明 |
| OnNewDocument | 在新文档被创建时由主结构调用。覆盖它是为了每次创建新文档时都对文档对象应用专门的初始化 |
| OnOpenDocument | 在从磁盘上装载文档时由主结构调用。覆盖它是为了每次装载新文档时都对文档对象进行专门的初始化 |
| DeleteContents | 主结构调用它来删除文档的内容。覆盖它是为了在文档关闭之前释放分配给文档的内存和其他资源 |
| Serialize | 主结构调用它在文档和磁盘之间串行化输出和输入。覆盖它是为了提供针对文档的串行化程序以便文档可以被装载和保存 |
| OnCloseDocument | 在文档关闭时调用 |
| OnSaveDocument | 在文档保存时调用 |
| SaveModified | 在包含未保存数据的文档被关闭之前调用，用来询问用户是否保存已做的改动 |
| ReportSaveLoadException | 串行化过程中有错误发生时调用 |

在SDI应用程序中，MFC只在应用程序启动时实例化文档对象一次，因此文档类的构造函数也只执行一次初始化。如果想随时在文档被创建或打开时进行某种初始化工作，就必须覆盖OnNewDocument或OnOpenDocument。

MFC提供了OnNewDocument和OnOpenDocument的默认实现，所以在覆盖这个两个函数时，应该调用基类中相同的函数。

一般来说，MFC应用程序更经常覆盖OnNewDocument而不是OnOpenDocument。因为OnOpenDocument间接调用了Serialize函数，它使用从文档文件中检索到的值初始化文档的永久数据成员。只有非永久数据成员才需要在OnOpenDocument中初始化。

SDI程序在新文档被创建或打开之前，主结构调用文档对象的虚拟DeleteContents函数来删除文档中已存在的数据。因此可覆盖DeleteContents并利用它释放分配给文档的任意资源，还可以执行其他必要的清理工作为重新使用文档对象做准备。MDI程序通常也是这样，不同之处在于，它们是在用户打开和关闭文档时各自创建和销毁的。

### 视图对象

视图类的GetDocument函数返回m\_pDocument，m\_pDocument数据成员中保存了指向视图相关文档对象的指针。

CView的可覆盖函数：

|  |  |
| --- | --- |
| 函数 | 说明 |
| OnDraw | 被调用来绘制输出文档的数据。覆盖是为了绘制文档视图 |
| OnInitialUpdate | 视图第一次附加到文档时被调用。覆盖是为了每次在文档被创建或加载时都初始化视图对象 |
| OnUpdate | 在文档的数据已经修改或视图需要更新时调用。覆盖是为了实现有效更新功能，只重画视图中需要重画的部分而不重画整个视图 |

文档试图应用程序中，WM\_PAINT消息由OnPaint函数处理，它使用CPaintDC对象来完成绘制工作，并用指向CPaintDC对象的指针来调用视图的OnDraw函数。OnDraw函数使用的是主结构提供的设备描述表指针而不是初始化自己的设备描述表，这样可以使用相同的程序实现向窗口输出、打印和打印预览。

SDI应用程序中，视图和文档一样，只构造一次然后重复使用。每当文档被打开或新建时都要调用视图的OnInitialUpdate函数。OnInitialUpdate的默认实现要调用OnUpdate，而OnUpdate的默认实现将使视图客户区无效并执行重绘。在覆盖后的OnInitialUpdate中应该调用基类的版本，否则新文档被打开或创建时视图不会被更新。使用OnInitialUpdate来初始化视图类的数据成员，并在单文档基础上执行其他与视图相关的初始化。

MDI应用程序中，覆盖OnActiveView视图可确定何时激活何时无效。如果视图被激活，OnActiveView的第一个参数是非零值，如果将要无效则为零。第2个和第3个参数分别标识即将被激活或无效的视图。

### 框架窗口对象

SDI应用程序只有一个框架窗口CFrameWnd，它被用作应用程序的顶层窗口并用来包含视图；MDI应用程序使用CMDIFrameWnd用作顶层窗口，而CMDIChildWnd窗口在顶层窗口中浮动用来包含应用程序文档的视图。

CFrameWnd类提供了OnClose和OnQueryEndSession处理程序，确保用户有机会在应用程序被关闭或Windows关闭之前保存未保存的数据。

### 动态对象创建

MFC的DECLARE\_DYNCREATE和IMPLEMENT\_DYNCREATE宏提供了创建动态创建的类的功能。需要做的工作：

* 从CObject派生对象；
* 在类声明中调用DECLARE\_DYNCREATE。DECLARE\_DYNCREATE只接收一个参数，即可动态创建类的名字；
* 在类声明的外部调用IMPLEMENT\_DYNCREATE。IMPLEMENT\_DYNCREATE接收两个参数：可动态创建类的名字和其基类的名字。

用类似下列语句使用这些宏来创建类实例：

RUNTIME\_CLASS(CMyClass)->CreateObject();

DECLARE\_DYNCREATE宏在类声明中添加了3个成员：一个静态CRuntimeClass数据成员、一个名为GetRuntimeClass的虚拟函数以及一个静态函数CreateObject。

DECLARE\_DYNCREATE(CMyClass)

等价于

Public:

static const AFX\_DATA CRuntimeClass classCMyClass;

virtual CRuntimeClass\* GetRuntimeClass() const;

static CObject\* PASCAL CreateObject();

IMPLEMENT\_DYNCREATE通过类名和类实例大小等这样的信息来初始化CRuntimeClass结构，它还提供了GetRuntimeClass和CreateObject函数。CreateObject的实现方式：

CObject\* PASCAL CMyClass::CreateObject()

{

return new CMyClass;

}

### SDI文档模板的其他内容

SDI应用程序创建文档模板对象时第一个参数是一个等于IDR\_MAINFRAME的整数值。它用来表示下面四个资源：

* 应用程序图标；
* 应用程序的菜单；
* 伴随菜单的加速键列表；
* 文档字符串，首先指定应用程序为文档生成的默认文件扩展名以及未命名文档的默认名字。

SDI程序中，主结构创建顶层窗口的过程为：先用保存在文档模板中的运行时创建类的信息来创建一个框架窗口对象，然后再调用该对象的LoadFrame函数。LoadFrame创建一个框架窗口并一次装载相关的菜单、加速键和图标。因此应给所有的资源分配相同的ID，并把ID传给LoadFrame。

文档字符串是一个字符串资源，由7个用”\n”字符分割的字符串组成。对SDI应用程序，子字符串从左到右依次具有如下含义：

1. 出现在框架窗口标题栏的标题。通常是应用程序的名称。
2. 分配给新文档的标题。如果被忽略，默认值是”Untitled”。
3. 一个描述文档类型的名字，在注册了两个以上文档类型的MDI应用程序中，当用户选中New命令后，它将和其他文档类型一起出现在对话框中。SDI应用程序中不使用此字符串。
4. 一个描述文档类型的名字，带有包含文件扩展名的通配符文件说明。例如“Drawing Files(\*.drw)”在Open和Save As对话框中使用次字符串。
5. 对于某类型文档的默认文件扩展名，例如”.drw”。
6. 不带空格的名字用来标识注册时的文档类型，例如“Draw.Document”。如果应用程序调用CWinApp::RegisterShellFileTypes来注册此文档类型，此字符串就成了以文档的文件扩展名命名的HKEY\_CLASSES\_ROOT子键的默认值。
7. 一个描述文档类型的名字。例如“Microsoft Draw Document”。与文档字符串中的前一个字符串不同，此字符串中可以包含空格。如果应用程序使用CWinApp::RegisterShellFileTypes来注册此文档类型，此字符串就是命令解释器在属性页中显示的便于人们阅读的名字。

如果需要忽略个别字符串，在”\n”后紧跟一个”\n”即可。

可以使用CDocTemplate::GetDocString函数从文档字符串中检索各个子字符串，如：

CString strDefExt;

pDocTemplate->GetDocString(strDefExt, CDocTemplate::filterExt);

将文档的默认文件扩展名复制到strDefExt中。

### 用操作系统命令解释器注册文档类型

要实现通过应用程序打开被双击的文档等类似操作，应用程序必须用操作系统命令解释器来注册其文档类型，这就需要在注册表的HKEY\_CLASSES\_ROOT部分添加一系列输入项，用来标识每个文档的文件扩展名以及用来打开和打印该类文件的命令。

可以使用两种方法进行这种注册：一种是提供REG文件，用户可以把它合并到注册表中；另一种是使用::RegCreateKey、::RegSetValue和其他win32注册函数通过编程将必要的输入项添加到注册表中。

MFC应用程序简化了这种处理，调用CWinApp::RegisterShellTypes并在调用AddDocTemplate之后传递一个TRUE参数可以在应用程序、其创建的文档及操作系统命令解释器之间铸就关键的链接。

CWinApp函数EnableShellOpen给MDI应用程序添加了漂亮的特性。如果MDI应用程序使用RegisterShellOpen和EnableShellOpen注册了其文档类型，在应用程序运行时如果用户双击了文档图标，命令解释器并不会自动启动第二个应用程序实例；它会使用动态数据交换DDE给已存在的实例发送“打开”命令及文档的文件名称。这样，文档就出现在顶层MDI框架内的新窗口中。

### 命令传送

SDI应用程序中“命令消息”的传送过程为（只有命令消息和用户界面更新才遵循传送机制）：



习惯上，New、Open和Exit命令由应用程序对象处理，CWinApp提供了OnFileNew、OnFileOpen、OnAppExit命令处理程序。Save、SaveAs命令通常由文档对象处理，它提供了Document::OnFileSave和Document::OnFileSaveAs的默认处理程序。显示和隐藏工具栏和状态栏的命令由框架窗口使用CFrameWnd成员函数来处理。其他大多数命令由文档或视图处理。

标准Windows消息如WM\_CHAR、WM\_LBUTTONDOWN、WM\_CREATE以及WM\_SIZE必须在接收这些消息的窗口中进行处理。鼠标和键盘消息通常传给视图，大多数其他消息则传给框架窗口。文档对象和应用程序对象从不接收非命令消息。

## MFC提供的视图类

常用的视图类：

|  |  |
| --- | --- |
| 类名 | 说明 |
| CView | 所有视图类的基类 |
| CCtrlView | CEditView、CRichEditView、CListView和CTreeView的基类，用来创建基于其他控件类型的视图类 |
| CEditView | 封装了多行编辑控件，并添加了打印、查找及查找替换功能 |
| CRichEditView | 封装了多功能编辑控件 |
| CListView | 封装了列表视图控件 |
| CTreeView | 封装了树视图控件 |
| CHtmlView | 从HTML文件和其他Microsoft Internet Explorer WebBrowser控件支持的媒体创建视图 |
| CScrollView | 给CView添加了滚动功能，是CFormView的基类 |
| CFormView | 实现可滚动的从对话框模板创建的“窗体”视图 |
| CRecordView | CFormView的派生类，用来显示从ODBC数据库获得的记录 |
| CDaoRecordView | CRecordView的DAO版本 |
| COleDBRecordView | CRecordView的OLE DB版本 |

### 滚动视图CScrollView

CScrollView靠自己处理滚动任务，除了实现OnDraw以外可以不对它做任何工作。实现OnDraw的方法与在CView中一样。除了希望有效使用它优化滚动操作。

要使用CScrollView，除指定CScrollView为基类外，还需覆盖视图类的OnInitialUpdate，并调用SetScrollSizes来指定视图的逻辑尺寸。通过此方式告诉MFC可滚动视图占据的区域大小。

CScrollView::SetScrollSizes接受4个参数，其中2个为可选参数，参数按顺序为：

* 指定映射模式的整数(必要，可以指定除MM\_ISOTROPIC和MM\_ANISOTROPIC以外的任何模式)
* 指定视图逻辑尺寸的SIZE结构或CSize对象(必要)
* SIZE结构或CSize对象，用来指定“页尺寸”，即单击滚动轴时MFC对视图的滚动量(可选)
* SiZE结构或CSize对象，用来指定行尺寸，即单击滚动条箭头时MFC对视图的滚动量(可选)

使用CScrollView时应该记住两个原则：

* 如果在视图中的OnDraw函数之外绘制输出，就要调用CScrollView::OnPrepareDC让MFC在输出中考虑映射模式和滚动位置的影响。
* 如果响应鼠标消息时执行命中测试，就要使用CDC::DPtoLP将单击处的坐标从设备坐标转换为逻辑坐标，从而在命中测试中考虑到映射模式和滚动位置的影响。

CSCrollView的工作方式为：当滚动事件发生时，CScrollView就用OnVScroll和OnHScroll消息处理程序捕获随后的消息，并调用::ScrollWindow来水平或垂直滚动视图。稍后，视图的OnPaint函数将被调用，以绘制由::ScrollWindow造成的失效窗口的一部分。CScrollView的OnPaint程序：

CPaintDC dc(this);

OnPrepareDC(&dc);

OnDraw(&dc);

CScrollView覆盖OnPrepareDC并在其中调用CDC::SetMapMode来设置映射模式，调用CDC::SetViewportOrg将视口原点转换为等于水平和垂直滚动位置的量。因此，在OnDraw重绘视图时滚动位置就被自动考虑进去了。

#### CScrollView的常用操作函数：

|  |  |
| --- | --- |
| 函数 | 说明 |
| GetScrollPosition | 检索当前水平或垂直滚动位置，返回CPoint |
| ScrollToPosition | 滚动到给定位置 |
| GetTotalSize | 测量视图的逻辑宽度和高度，返回CSize |
| SetScaleToFitSize | 将整个逻辑视图缩放到物理视图中。要恢复视图的默认滚动方式，再次调用SetScrollSizes即可。 |

#### 优化滚动操作

优化OnDraw的关键是CDC函数GetClipBox。它在传递给OnDraw的设备描述表对象中被调用，GetClipBox用无效矩形的逻辑坐标下的尺寸和位置来初始化RECT结构或CRect对象，如下所示：

CRect rect;

pDC->GetClipBox(&rect);

初始化的CRect可以区分需要重绘的视图部分。如何使用此信息完全取决于应用程序。可以将返回的坐标转换为行列号等等。

如果OnDraw视图绘制整个视图，GDI就会通过剪切无效矩形外的像素点来消除不必要的输出。但是剪切需要时间，结果是根据OnDraw视图绘制无效矩形外的内容而浪费的CPU周期的不同，滚动操作的效果可能很好，也可能非常糟糕。