MFC 技术内幕

(美) Al Williams 著

龚 波 赵军锁 陈 胜 等译

机械工业出版社
China Machine Press

利用MFC开发Windows程序已成为编程的主流。本书引导读者逐渐深入了解MFC，并向读者展示如何避免走弯路，以及如何创建非同寻常的程序。本书还包括某些MFC的高级特性，如多线程、数据库、扩展DLL和自定义向导。

本书版权登记号：图字：01-99-0033

图书在版编目(CIP)数据

MFC技术内幕/(美)威廉姆斯(Williams, A.)著；龚波等译. -北京：机械工业出版社，1999.5
书名原文：MFC Black Book
ISBN 7-111-07210-3

Ⅰ.M… Ⅱ.①威… ②龚… Ⅲ.程序设计-参考资料 Ⅳ.TP311

中国版本图书馆CIP数据核字(1999)第10912号

出 版 人：马九荣（北京市百万庄大街22号 邮政编码100037）
责任编辑：陈剑瓯
北京市昌平环球印刷厂印刷·新华书店北京发行所发行
1999年5月第1版第1次印刷
787mm × 1092mm1/16 · 21.25印张
印 数：0001－ 6000册
定 价：49.00 元(附光盘)

译 者 序

MFC(Microsoft Foundation Class，微软基本类)系统为各种Windows实体提供了基本类，封装了大多数API函数、数据结构和消息结构。现在，利用MFC开发基于微机的Windows程序已成为编程的主流。你只须了解MFC系统各种基本类及其成员函数的使用方法，并借用Visual C++的集成开发环境，不必完全掌握数百个Windows API系统函数的使用方法和各种Windows系统消息的内部结构，就可以编写出出色的程序。

《MFC技术内幕》一书是由著名的语言大师Al Williams数年编程之心得凝聚而成的。作者高屋建瓴，以一种独到的角度从全局引导我们逐渐认识MFC。作者提供了详尽而准确的示范程序，帮助你理解MFC的原理。尤其可贵的是，作者把对MFC研究所得倾囊献出，剖析了MFC中的不足和缺陷，并提出了自己的解决方法。

我们相信，本书不仅对于MFC入门者有帮助，也可以给那些熟悉MFC的编程者带来“山穷水尽疑无路，柳暗花明又一村”的感觉，在许多方面可以给你提供许多独到的思维方式。

全书由前导工作室的龚波组织翻译，参加翻译工作的还有赵军锁、陈胜、李志、张巧莉、李林、田野、李小将、李琢颖、李琢琳、聂宛忻、相昌民、龚建、龚露、李士心、田丽韫、萧东等。全书由李志、赵军锁审校。前导工作室的全体同志为本书的翻译、录排、校对都付出了辛勤的劳动。

由于时间仓促，且译者经验和水平有限，译文难免有不正确之处，恳请读者批评指正。

1999年3月

前　言

你是一个MFC程序员吗？那太好了。MFC程序员可以分为两类。你是哪一类呢？第一类程序员规规矩矩地按照MFC希望的方式编写他们的程序。第二类是极端活跃的，他们热衷于按自己的方式行事。我是属于第二类的。如果你和我志同道合（或愿意如此），那么这本书便是为你写的。

本书不是教你如何使用MFC，不是传统意义上的那种。在你对基于MFC程序设计有了很好的了解并且期望能够以一种不同的方式做事的情况下，本书对你再适合不过了。本书并非一本初级教科书（虽然第1章对一些基本的东西进行了复习和回顾）。在本书中你将学会如何对MFC程序进行精简优化，并发现如何使用、否定以及放弃文档/视结构。如果你想定制档案，在本书中也可以找到相关的信息。

本书的目的

在Windows程序设计中，由于OLE很难，因此你会选择使用MFC。它较为容易一些。当然，你可以自由地实现自己的OLE（可能需要1人年的开发时间），但如果你想在3天之内完成它，最好坚持使用MFC。

通常，MFC很不错。但是如果你要用它，必须购买它提供的所有功能。你不能只使用OLE部分（或打印预览和切分窗口）。当你使用MFC时，就必须同意MFC的做事方式。

即使在MFC内部，你也会发现同样的现像。你知道你可以对哪些子窗口控制使用DDX（动态数据交换）吗？然而，Class Wizard 只能帮你在某些特定的窗口（如对话框）上使用DDX。因而在Class Wizard 帮助之外使用DDX的文档很少，这就等于限制了你使用DDX的范围。

过去的年代

在我用来进行正式程序设计的第一台计算机中，嵌入了我设计的微处理程序。那时，创新高于一切。我可以梦想一切，我能够按自己的想法编写程序（在4KB的ROM上），问题是你必须把梦做得很细，细到任何一个细节。在过去的年代，我花费了大量的时间仅仅是为了分开字节并将之在软件驱动的RS-232口上发送出去。我必须数中断个数来计算一天的时间。很显然，太多的创造并不是什么好事。

现在，你不用再处理这些烦人的细节了。如果你想向串口发送点什么，只须打开它并将字节倒出去就行了。还有，如果你想和鼠标、调制解调器或打印机打交道，则有更高级的访问方式，你甚至不必关心将它们连到计算机上的串行口。但是在生产效率提高的同时，你失去了创造性。虽然你曾经一度能够控制每一个细节，但现在你必须受到各种奇怪的限制，不管使用的是什么操作系统。

这并非什么坏事。谁愿意编写他们自己的读写磁盘的程序呢？该程序能适用于市场上所有的磁盘吗？你真愿意为每种打印机和显示卡编写代码吗？通常，答案是否定的。但是，这

些是低级的细节。对于高级的细节，用户界面风格又如何呢？Windows在这一级别上也提供了一定的工具支持。但问题是，工具的级别越高，对你的限制越大。

最后的例子是像Visual Basic这样的语言。VB在做某些特定的事情时很不错。如果你的程序所做的事情正是VB擅长的，你不用它反而显得不明智了。但是，如果你的程序在做VB不擅长的事情呢？那就困难了。你常常会强迫VB去做它没有想到的事情，但这会花费太多的精力。最常见的是，工作量大大增长。通常，你用更灵活的语言会更好些。因此，VB，像所有的工具一样，在一定程度上限制了你解决问题时所采用的方案。

MFC的限制

作为一个MFC程序员，你受到的限制是什么呢？太多了，虽然你现在可能还没有意识到。MFC对你的限制有以下几个方面：

- MFC有一个独特的结构，其许多部分都依赖于该结构。
- 当MFC提供一个组件或类时，通常使用它比开发新的（或改进老的版本）要容易得多。
- MFC的工具（App Wizard 和Class Wizard ）只适用于特定的程序；但在创建其他的程序时就困难了，因为它们无法帮你。

本书所讲的是为了打破这些限制。有时这意味着大量的工作。但如果你知道如何去做，有时也很容易。在任何情况下，你都可以用MFC来达到目的，你没有时间去等MFC的后续版本为你做这些。当然，将你的File菜单放在菜单栏中间违反了著名的SMP原则。这个软件工程定理指出：软件的工作方式应该对大多数用户的震动最小。

另一方面，在大多数情况下，你希望以一种不同的方式工作。你喜欢实现自己的创造性并开发出没有人见过的东西，对吗？这也是本书的目的：帮你实现自己的编程风格。

本书的使用

为了从本书获得最大的收益，你应该有一份Microsoft Visual C++ 5.0或更高版本的拷贝。你可以用其他版本（微软或其他厂商）的C++验证本书中的大多数技术，但程序清单中的代码或CD-ROM上的代码都是用Microsoft Visual C++ 5.0编写的。

每章针对一个特定的主题。每章的第一部分包含的是关于本章主题的细节。第二部分是实践的问题和对它们的菜谱风格的解决方案。这可能引用了MFC技术注释、Web页面或杂志上的文章。

每章都是独立的。如果你有一个特殊的问题，可以通过浏览实战指南部分（每章的最后）来找到类似的问题。当然，你应该仅把解决方案当作指南，毕竟，你想验证自己的创造性思维。

你可能有自己的软件版本。可能你的顾客、你的老板或你的竞争对手都在催促你以一种特定的方式生产软件。无论如何，挑选一章读一下，并学习如何以自己的方式来做事。

原书信息

原书书名：MFC Black Book
原书书号：ISBN 1-57610-185-1
原出版社站点地址：WWW.Coriolis.com

目　录

第1章 体系结构

在接触高级MFC程序设计技术之前，你要具有扎实的基本技能和知识。本章用于检查和复习你对MFC基本结构框架的理解，包括文档/视结构、消息映射、命令行分析、子类窗口和集合的使用。

古罗马人不擅长数学。当然，他们没有发现数字0的事实仅是一个方面。在古老的西方世界中，阿拉伯人是最伟大的数学家。他们从印度人那里学到了很多，而印度人在公元前200年便发明了数字0。另外，玛亚人也使用0，并且具备高度的数学文明。他们使用的日历甚至比我们今天使用的还要精确。

假设古代的人们使用二进制而不是十进制进行计数，那么我们便可以用我们的手指数到1023了。这并不像你想象的那么牵强。日耳曼人和塞尔特人基于12进行计数。这就是我们为何用“打”(一打为12个)进行计数以及钟表上被划分为12个小时的原因之一(不考虑1英尺等于12英寸了)；在英语中，11和12为eleven和twelve，而不是oneteen和twoteen也是由于这个原因。

以上事实说明了一点，你所用的工具决定了你能够解决的问题的范围。我们不妨看一下象形文字，如汉语和日语。这些语言不能很好地适应计算机。这是造成远东地区计算机技术迅速发展的主要障碍。例如，人们不得不根据字音在膝上机输入时使用Kanji字母表，并从膝上机显示的几个同音异义字中选取用户希望的那个字。这样做效率很低。另一方面，象形文字对于手写识别系统很便利。许多日文掌上型计算机允许你在屏幕上进行手写输入。只有很少的英文掌上型计算机能够做到这一点，但通常效果也不甚理想。英文字母不便于计算机进行区分。

你所用的开发环境的方方面面无不影响着你的程序。例如，如果你为Windows开发应用程序、在编程时就要遵守其特定的要求。MFC、C++以及你用来生成代码的工具也都会对程序产生影响。

尽管本书的目的之一是揭示如何用一种不同的方式来编程，你仍然需要理解有关工具(指MFC和Visual C++)的工作机制。这样做的理由有两个。首先，你需要知道MFC如何工作、这样你才能使之按你的意愿行事。其次，你通常并不十分清楚MFC工作的细节，因为它使用的工具(App Wizard和Class Wizard)为你做了许多事情。向导像VCR一样使用了VCR Plus的代码。只要你有这部分代码，就不会有问题。但是如果你想处理某些不具备VCR Plus代码的事物时，你就遇到麻烦了。你不但要编写VCR，而且可能还不知道如何去做。

本章的首要目的是讨论MFC的每一个重要的组成部分以及它们与整个结构的关系。其次，通过解剖一个App Wizard生成的程序，了解它的工作机制。

如果你曾写过大量的MFC代码，或许认为无需阅读这部分内容了。但这样做是否行，这要在你回答了下面的问题后才能知道。如你能够回答下列问题，便可以跳过这部分内容而不会有任何麻烦。也许，你仍想研究一下本章后面的实战指南部分。问题如下：

- MFC程序所必须具有的一个类是什么？
- 如何手工构造一个消息映射？

- 如何向一个文档附加多个视?
- 如何创建一个新的文档类型?
- 什么时候文档不处理I/O?
- 何时视不执行程序绘画?
- 你能在文档类中放入一个菜单处理程序吗?
- 为什么CRect不能由CObject导出?
- 文档模板必须位于应用程序对象之内吗?
- 为什么难以用模板创建一个二维CArray?

1.1 MFC的主要成员

MFC由许多类组成。但是只有少数几个核心的组件对你的程序结构框架有影响。其他的类不会直接关系到你的编程方式。

核心的类为CWinApp、CView、CDocument、CFrameWnd和CDocTemplate。尽管这五个类构成了大多数MFC程序的核心，但它们并非总是必要的。事实上，MFC程序必不可少的唯一一个类就是CWinApp 。但是，为了从MFC得到更多，你也会用到其他的类(或从它们导出的类)。

MFC的强大之处在于它使用了一种可以称之为“design by difference”的技术。该思想便是MFC提供构成Windows程序模型的类。然而，这个原始类型的程序所完成的功能单调乏味。作为MFC程序员，你的工作便是编写代码，使你的应用程序与众不同。MFC负责所有的缺省行为，而且允许你有选择地重载一些函数，使之符合你的意愿。

CWinApp代表你的应用程序在内存中的运行。用户从来不会看到与CWinApp直接相关的任何东西。这是获取与应用程序相关的数据的关键地方(例如，命令行、资源、实例句柄等等)。

表1-1列出的是CWinApp的主要成员。其中最重要的是两个可重载的函数InitInstance和ExitInstance。通常，App Wizard在InitInstance中放入代码进行主窗口的创建并执行其他的初始化任务。然而，你可以在这里完成所有的工作，然后返回FALSE并终止程序。如果你用App Wizard创建了一个基于对话框的应用程序，那么其行为便确是如此(见程序清单1-1)。此时，程序创建了一个对话框，但它还不能做你期望的任何事情(只要它不要求一个事件循环)。

表1-1 CWinApp的主要成员。

成 员	描 述
m_pszAppName	应用程序名称
m_lpCmdLine	命令行
m_hInstance	应用程序的句柄
m_hPrevInstance	已存在的实例句柄
m_pMainWnd	应用程序的主窗口
m_bHelpMode	在上下文帮助模式下为TRUE
m_pszExeName	EXE文件名
m_pszProfileName	INI文件名
m_pszRegistryKey	替代INI文件的注册键名称
LoadCursor	加载一个应用程序光标
LoadStandardCursor	加载系统光标(IDC_ *)

(续)

成 员	描 述
LoadOemCursor	加载OEM光标(OCR_ *)
LoadIcon	加载应用程序的图标
LoadStandardIcon	加载系统图标(IDI_ *)
LoadOemIcon	加载OEM图标(OIC_ *)
ParseCommandLine	初始化一个基本命令行的CCommandLineInfo对象
ProcessShellCommand	在CCommandLineInfo对象中处理外壳命令
GetProfileInt	从INI文件中取一个整数
WriteProfileInt	向INI文件中写一个整数
GetProfileString	从INI文件中获取一个字符串
WriteProfileString	向INI文件中写一个字符串
AddDocTemplate	向应用程序增加一个文档模板
GetFirstDocTemplatePosition	找到列表中第一个文档模板前面的POSITION
GetNextDocTemplate	从列表中获取下一个文档模板
OpenDocumentFile	用名字打开文档文件
AddToRecentFileList	向“最近打开的文件”菜单上加入一个名字
CreatePrinterDC	基于所选择的打印机获取一个打印机DC
GetPrinterDeviceDefaults	从WIN.INI或上一次打印设置获取打印机缺省值
InitInstance	每个实例初始化一次
Run	事件循环
OnIdle	空闲时间处理
ExitInstance	程序终止
PreTranslateMessage	消息筛选
SaveAllModified	如果需要的话，提示用户保存文档
DoMessageBox	重载改变AfxMessageBox()
ProcessMessageFilter	为筛选挂钩处理消息
ProcessWndProcException	异常的缺省处理程序
DoWaitCursor	打开或关闭沙漏标
WinHelp	打开帮助文件
LoadStdProfileSettings	加载标准INI文件设置和MRU文件列表
SetDialogBkColor	设置对话框背景色
SetRegistryKey	通知MFC使用注册表而不是INI文件进行永久存储
EnableShellOpen	允许拖放打开文档
RegisterShellFileTypes	注册文件类型
OnFileNew	缺省菜单处理程序
OnFileOpen	缺省菜单处理程序
OnFilePrintSetup	缺省菜单处理程序
OnContextHelp	缺省菜单处理程序
OnHelp	缺省菜单处理程序
OnHelpIndex	缺省菜单处理程序
OnHelpFinder	缺省菜单处理程序
OnHelpUsing	缺省菜单处理程序

重载InitInstance和ExitInstance是进行不同方式程序设计的重要手段。缺省的InitInstance例程其实什么也没有做。但可以肯定，你的程序需要创建一个主窗口并执行其他一些初始化任务。因此你几乎总是要对InitInstance进行重载。另一方面，当程序终止时你可能并不需要

做什么。如果是这样的话，就不用重载ExitInstance函数了。当然，如果你需要进行某些终止处理，就可以按自己的意愿对之重载。

你从MFC获取的绝大部分强大的功能，如打印预览、切分窗口等等、唯有在使用文档/视结构时才会出现，而文档/视结构是大部分MFC程序的主要结构特征。

程序清单1-1 App Wizard 对话框应用程序的摘要

```
BOOL CDlgdemoApp::InitInstance()
{
   AfxEnableControlContainer ();

   // Standard initialization
   // If you are not using these features and wish to reduce the size
   // of your final executable, you should remove from the following
   // the specific initialization routines you do not need.

#ifdef _AFXDLL
   Enable3dControls();       // Call this when using MFC in a shared DLL
#else
   Enable3dControlsStatic(); // Call this when linking to MFC statically
#endif

   CDlgdemoDlg dlg;
   m_pMainWnd = &dlg;
   int nResponse = dlg.DoModal();
   if (nResponse == IDOK)
   {
     // TODO: Place code here to handle when the dialog is
     // dismissed with OK
   }
   else if (nResponse == IDCANCEL)
   {
     // TODO: Place code here to handle when the dialog is
     // dismissed with Cancel
   }

   // Since the dialog has been closed, return FALSE so that we exit the
   // application rather than start the application's message pump.
   return FALSE;
}
```

为了更好地理解文档/视结构，不妨先考虑一个不用该结构的实例。假定我们编写了一个简单的电子表格程序，处理从气象传感器接收的数据。这并不十分新奇，只是一个普通的网格，可以处理数字和公式。因为这很简单，所以用不着麻烦文档/视结构。我们只是简单地画了一些网格，将之作为处理的一个组成部分。几个月后，你继承了我们的电子表格，而老板让你添加一个条形图显示。

现在的问题就不再那么简单了。你必须修改所有的绘画代码，以确定它是描绘图表还是描绘网格。如果你想两者兼顾，并且使用同样的数据，又会如何呢？如果你想以两种不同的方式显示两部分独立的数据，又会发生什么呢？你遇到了大麻烦。

但如果使用文档/视结构，事情就简单多了。首先定义一个文档对象(由CDocument导出)。该对象负责维护程序中的数据(在这里指数字、公式和传感器数据)。它应该能够加载和保存数据(可能是写入文件)，并且重新计算每个公式。但是，它并不负责在屏幕上显示任何东西。它也不负责用户界面。

绘制和处理用户界面是视类(由CView导出)的职责。每个视都指向一个文档。视的工作如下：

- 以某种方式显示相关文档中的数据。

• 接受用户对数据的操作(例如，鼠标点击)，并据此更新文档。

这个方案有许多优点。再来考虑一下电子表格的例子。这一次，假定使用的是文档/视结构，现在来增加条形图显示就变得容易多了。改变文档(如读取不同的传感器)也简单多了。

一个文档同时驱动多视也不复杂。在本章的实践部分你将看到该例子。用户从不直接查看或操作文档对象。用户看到的只是视。

1.1.1 我的程序与上述不同

通常，人们都认为他们的程序不适合文档/视结构，有时，他们是正确的，但在更多的情况下，仔细考虑一下，他们的程序便适合该模型了。为此，我要责备微软，因为它在名字的选择上做得太糟糕了。

问题出在名词“文档”上。通常，你认为文档是一个磁盘上的文件。其实，文档对象只是在大多数时候(并非总是)代表这样的文件。例如，如果你在写一个电子表格，那么文档可能便表示一个文件。但是如果你在写一个显示从一组仪器得来的实时大气数据的程序时，情况又如何呢？这时，文档就是从仪器读取的数据。如果你编写的是一个网络监视器程序，那么文档包含的便是网络的数据性能。

Smalltalk对其窗口化应用使用了相同的设计类型。他们称之为“Model View Controller”(模型、视、控制台)结构(MFC的视相当于Smalltalk的视和控制台的组合)。Smalltalk模型与MFC文档相同。我偏爱用“模型”这个词，因为这样就可以帮助人们不将其想当然地当作文件来对待。

提示 什么是文档?

记住：文档只是程序数据的抽象表示。至于数据从何而来就无关紧要了。

有时在决定文档(或者，也可以称之为文件)中存放什么的问题上需要思考一下。不久前，我编写过一个哑终端程序。这是一个文档/视方案看来不可行的典型例子，尤其是因为我使用CEditView作为视类。在这样的程序里你向文件中存放些什么呢？答案是配置信息。当你向该程序加载一个文件时，你感兴趣的并不是远程计算机的最近200行输出是什么。你希望的是程序记住你是在9600波特的COM2端口上，并且没有奇偶校验。

提示 保存文档数据

将数据保存到非永久的文档对象即可。你只须存储日后要重新恢复的数据。

1.1.2 军官能力测试

你做过军官能力测试吗？只要花费1分钟就够了。假定你是一个尉官，你手下有一个士官、两个士兵、两个9英寸长的竿子、一个11英寸长的竿子、三条6英寸长的绳子、一个挖洞器和一面美国国旗。在30秒或更短的时间之内，告诉我你如何制成一个14.5英寸高的旗杆。

正确的答案是：“士官，把旗杆做好立起来！”

这与MFC有很大的相似之处。你或许注意到了，在前面的部分中，我从未说过文档或视做些什么，而只是谈及了它们各自的任务。尽管MFC坚持每个对象负责一定的行为，但它不需要该对象去实际完成该任务。

在绝大部分情况下，所涉及的对象真的执行了这些任务。但是，也可能该对象将任务分派给了其他对象。例如，或许有些非MFC对象知道如何描绘你的数据。但这不是问题，你只须在视中使用这些对象即可。

另外的一个例子是CEditView (MFC提供的一个文本编辑视类)。该类知道如何读写ASCII码文本文件。最终仍然是文档负责读写文件。如果文档选择了找到该视类并将实际的操作交由它来处理，则MFC就不关心这些了。

CEditView是你为何想委托某项操作的绝佳例子。假定你正在编写一个处理GIF文件的类。你希望能与其他的各种工程程序员共享该类。当然，你可以编写CGIFView和CGIFDoc，但是这样可重用性不大。有些程序员或许想在其程序中放入GIF文件和其他东西。当然，你可以导出新类，但是，如果他们也想使用CPCXView 和CPCXDoc(来处理PCX文件)呢？因为MFC不支持多重继承，所以这便成了问题。

如果使用委托，那么答案至少有两种可能。第一，你可以创建一个CGIFDoc用于读写GIF文件。它也知道如何描绘GIF文件。视负责调用文档的绘画程序。如果程序被多个文档调用，你最好是创建一个超类用于管理各种所需文档。

另一种解决方案是创建一个无关类(可能是CGIF)，它具有文档和视可以调用以完成所需工作的函数。该类无需从MFC类导出(当然，你可以从CObject 导出该类，CObject 类在本章后面还会进一步详述)。

提示 MFC类的使用

你无需将全部代码都放入一个MFC类中。可以自由导出新类来表示你的程序。这样有助于重用现有的代码或使用MFC还不能很好地支持的特征(如多重继承)。

1.1.3 框架窗口

视并非简单地显现在用户屏幕的中央。用户期望视具有所有有用的装备：可调整大小的边框、标题栏、系统菜单，等等。虽然用户一般来说并不清楚，这些东西其实都属于框架窗口。用户通常认为一个窗口分为两部分：框架和视。框架有两种类型。SDI(Single Document Interface，单文档界面)仅有一个窗口。一个SDI程序的常见例子便是标准的NotePad 编辑器(见图1-1)。它只有一个窗口，并且每次只能打开一个文件。如果NotePad是一个MFC程序(它不是，我知道的情况是这样)，那么其内部应该是一个视，而边框、菜单栏、系统菜单则属于从CFrameWnd 导出的类。

MDI(Multiple Document Interface,多文档界面)程序的主框架窗口使用的是CMDIFrameWnd。这些程序(如Microsoft Word)将每个打开的文档放在其子窗口中(见图1-2)。这些子窗口包括一个CView 导出的类和CMDIChildWnd类。

视和框架都是由一般窗口类(CWnd)导出的。你对窗口的一切操作都适用于视或框架。SDI程序有两个不同的窗口(框架和视)。MDI程序包含一个框架窗口，对每个打开的文档都有一个框架和一个视。MDI程序还具有MFC忽略的一个客户窗口(见图1-2)。这是位于主框架窗口内的窗口。虽然你不能真正地直接看到它，但使用侦察程序可以看到，它确实是一个窗口。它是文档框架的父窗口。没有它，文档框架可能会覆盖工具栏和状态栏。在这两种情形下，主框架窗口都有自己的菜单和工具栏、状态栏以及其他类似的东西。

这就引发了一个有趣的问题：全部的菜单处理代码都属于框架吗？答案是NO。要找出究竟是为什么，你须知道MFC程序是如何处理消息的。

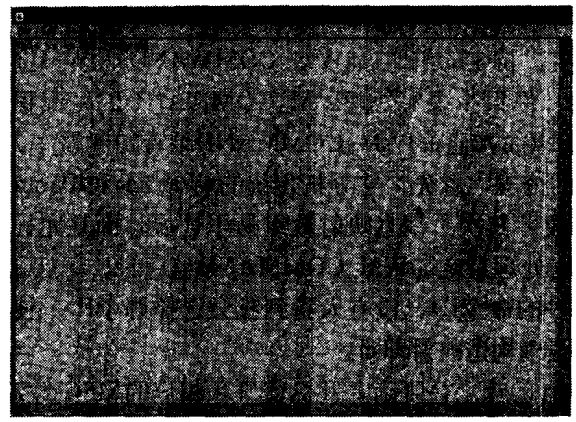

图1-1 NotePad，一个典型的SDI程序

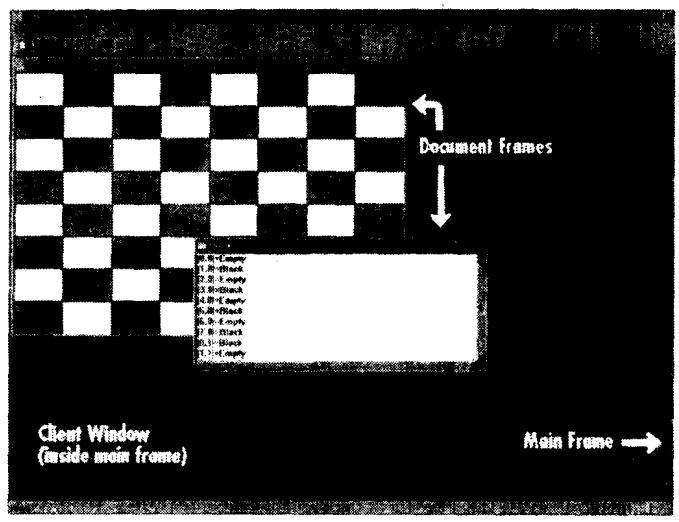

图1-2 一个MDI程序(已被加了注解)

1.1.4 消息映射

如果你只使用Class Wizard 处理消息，或许还不知道它是如何操作的。但通过某种方法它使得Windows消息的到来导致了类中特定的函数被调用。

如果你想设计一个类库，或许会用虚函数处理消息。例如，CWnd 就有一个叫做OnSize的虚函数。所以你的标准事件循环中可以安排响应WM_SIZE消息的OnSize的调用。

在大多数情况下这都是可行的。但是，这样做有两个重要的不足：

- 大多数窗口只处理少量的消息，但是每个窗口却需要一个庞大的虚函数表和每个消息的入口。
- 虚函数处理用户定义的消息、注册消息和其他自定义情况不是很容易的。

为避免这些问题，Class Wizard 用消息映射模拟虚函数。消息映射就是一个入口数组，当MFC确定如何传递消息时要用到该表。数组中存储了几段重要的关键信息：

- 所处理的消息。

- 消息应用于的控件ID，有多少算多少(下面有具体解释)。
- 消息所传递的参数。
- 消息所期望的返回值类型。

第二项内容尤其重要。特定的消息(如WM_COMMAND 和WM_NOTIFY)被进一步划分，应用于特定的ID。因此，你很少编写处理WM_COMMAND的处理程序，而是为带有一个参数ID_MENU_FILE_OPEN(或其他)的WM_COMMAND编写处理程序。

该方案有几个重要的分支。首先，它分析传统的wParam 和lParam参数，使你的函数接收到正确类型的未包装参数。其次，消息映射数组易于扩展，而且允许放入你想要的任何消息的处理程序。当消息映射处理特定的WM_COMMAND消息时，这就显得特别重要。在一个程序中我可以有成百个唯一的命令ID。没有人猜到我会选择哪个ID。使用消息映射，ID的选择不再重要，因为我只要正确构造映射即可。

当MFC收到大部分消息后，它便确定目标窗口和相应的MFC类实例。然后它便搜索窗口的消息映射寻求匹配。如果窗口没有该消息的处理程序，MFC便进一步搜索窗口的基类。如果再没有基类了并且MFC没有发现处理程序，MFC便将消息传递给该窗口的原窗口过程(换句话说，非MFC消息处理代码)。

当然，手工构造消息映射数据是一项繁重且易出错的工作。在更高级别上，MFC提供了Class Wizard来做这项工作。但即便是在较低层次上，有些Class Wizard可以使用的宏也能够简化该过程。

有几个宏Class Wizard可以使用，有些你能够使用而Class Wizard却不能使用。向现有的消息映射添加自己的宏应该不会有什么问题，但要确保你的添加放在Class Wizard的特殊注释之外。考虑如下这个由App Wizard产生的程序的消息映射:

```
BEGIN_MESSAGE_MAP(CLinkckView, CFormView)
        //{{AFX_MSG_MAP(CLinkckView)
        ON_BN_CLICKED(IDC_SCAN, OnScan)
        ON_COMMAND(ID_FILE_PRINT_PREVIEW, OnFilePrintPreview)
        ON_COMMAND(ID_FILE_SCAN, OnScan)
        ON_LBN_DBLCLK(IDC_LB, OnScan)
        ON_UPDATE_COMMAND_UI(ID_FILE_PRINT, OnUpdateFilePrint)
        ON_UPDATE_COMMAND_UI(ID_FILE_PRINT_PREVIEW, OnUpdateFilePrint)
        //}}AFX_MSG_MAP
// Put your extra macros here
        // Standard printing commands
        ON_COMMAND(ID_FILE_PRINT, CFormView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_DIRECT, CFormView::OnFilePrint)
END_MESSAGE_MAP()
```

位于AFX_MSG_MAP行之间的宏来自Class Wizard 。在第二个AFX_MSG_MAP注释行之后，你可以放入任何你想放入的东西，Class Wizard并不知道它的存在。已经存在的两个打印宏来自App Wizard。它把这两个宏放在那儿，所以在Class Wizard 中它们便不再出现。

从前面的消息映射例子可以看出，两个基本的宏是BEGIN_MESSAGE_MAP和END_MESSAGE_MAP。这两个简单的宏用于开始消息映射表，制作一些基本的入口，然后再关闭该表。两者之间便是重要的宏(见表1-2)。当然，也有针对每一个特定消息的宏(例如，ON_WM_CLOSE是用于WM_CLOSE消息的，而ON_WM_PAINT是用于WM_PAINT的)。

表1-2 有用的消息映射宏

名称/参数	描　　述
ON_COMMAND	处理WM_COMMAND消息
ID	WM_COMMAND消息带的控件ID

（续）

名称/参数	描　　述
func	调用的成员函数[void func(void)]
ON_COMMAND_RANGE	为WM_COMMAND消息处理ID的范围
ID	范围内的第一个ID
IDLast	范围内的最后一个ID
func	调用的成员函数[void func(WORD id)]
ON_COMMAND_EX	同ON_COMMAND，但处理函数以ID为参数并返回BOOL值
ID	控件ID
func	调用的成员函数[BOOL func(WORD id)]
ON_COMMAND_EX_RANGE	同ON_COMMAND_EX，但用于ID范围
ID	范围内的第一个ID
IDLast	范围内的最后一个ID
func	调用的成员函数[BOOL func(UNIT id)]
ON_UPDATE_COMMAND_UI	处理MFC的请求，用于声明用户界面项目的状态
ID	控件ID
func	调用的成员函数[void func(CCmdUI *pCmdUI)]
ON_UPDATE_COMMAND_UI_RANGE	同ON_UPDATE_COMMAND_UI，但用于ID范围
ID	第一个ID
IDLast	最后一个ID
func	调用的成员函数[viod func(CComdUI *pCmdUI)]
ON_NOTIFY	处理来自新风格控件的WM_NOTIFY消息(例如，公共(common)控件)
code	指示编码
ID	控件ID
func	调用的函数[void func(NMHDR *hdr,LRESULT *result)]
ON_NOTIFY_RANGE	同ON_NOTIFY，但用于范围
code	指示编码
ID	控件ID
IDLast	最后一个ID
func	调用的函数[void func(UINT id，NMHDR *hdr,LRESULT *result)]
ON_NOTIFY_EX	同ON_NOTIFY，但调用的函数返回BOOL
code	指示编码
ID	控件ID
func	调用的函数[BOOL func(NMHDR *hdr,LRESULT *result)]
ON_NOTIFY_EX_RANGE	同ON_NOTIFY_EX，但用于ID范围
code	指示编码
ID	控件ID
IDLast	最后一个ID
func	调用的函数[BOOL func(UINT id，NMHDR *hdr,LRESULT *result)]
ON_CONTROL	处理控件指示的WM_COMMAND消息(如，EN_和BN_消息
code	指示编码
ID	控件ID
func	调用的函数[void func(void)]