序　言

本书的内容是关于用 Microsoft® Foundation Class Library(MFC)编程的。类库就是 C++类集,它封装了为 Microsoft Windows™操作系统而编写的应用程序的函数。这个版本的 MFC 能支持 Win32® 平台的编程,包括 Microsoft Windows NT™。

第 1 部分,请按顺序阅读,是类库的概述,用于帮助理解一个 MFC 应用程序的主要组成部分,以及它们是如何协同工作的。第 1 部分介绍了下述主题:

- MFC 应用程序的关键组成部分:
  - 应用程序对象,它代表的是应用程序
  - 文档模板对象,它创建文档、边框窗口和视等对象
  - 文档对象,用于存储数据和把数据串行化送到永久性存储体中
  - 视对象,用于显示文档的数据并管理用户与数据的交互
  - 边框窗口对象,用于包含视
  - 线程对象,允许用 MFC 类编写多个执行线程
- 对话框、控制和控制条,如工具条和状态条。
- 对象链接和嵌入(OLE)的 Visual Editing 和 OLE Automation。
- 使用 Open Database Connectivity(ODBC)的数据库支持。
- 有用的通用类,如字符串集合、异常和日期/时间等对象。

第 2 部分,可以随机地阅读,是一个百科全书——关于用 MFC 编程的以字母顺序排列的一些条目。利用这些条目可以跟踪不同信息的线索。关于这些条目之间的相互协同,参见"使用百科全书"第 1 条。以特别重要的线索开始的条目包括:

- "MFC"
- "OLE 概述"
- "数据库概述"

文档约定

本书使用了下列印刷上的约定:

例　子	描　述
STDIO.H	大写字母表示文件名、寄存器,以及在操作系统命令级使用的术语。
char, **_setcolor**, **_far**	黑体表示 C 和 C++关键字、操作符、语言特定的字符和库函数。在讨论句法时,黑体表示必须完全按所示的文本输入。 许多常量、函数和关键字都以单或双下划线开头。这些是该名称所必须的。例如,只有当含有前导的双下划线时,编译器才能识别**__cplusplus** 这个说明常量。
expression	斜体字所占的位置表示的是程序员必须提供的信息,如文件名。斜体字偶尔也用在文本中以突出重点。

续表

例　子	描　述
[[*option*]]	双方括号内的项表示是可选的。
#pragma pack {1|2}	花括号和竖条表示在两个或多个项中选择一个。除非双方括号([[ ]])把花括号括了起来，否则必须选择这些项中的一个。
#include <io.h>	这种字体用于在文本中表示例子、用户的输入、程序的输出和错误消息。
CL [[*option*...]]*file*...	一个项后面有三点(省略号)表示可能有具有相同格式的多个项出现。

while()
{
⋮
}
	三点排成一列或一行表示实例程序有部分被故意省略掉了。

CTRL+ENTER	小号的大写字母用于表示键盘上的键名。如果两个键名之间有一个加号(+)，则在按第二个键时应该按住第一个键。 回行键叫做 ENTER，有些时候在键盘标成了弯曲的箭头。
"argument"	引号括起来的内容是文本中第一次定义的新术语。
"C string"	有些 C 构件，如字符串，要求有引号。语言所要求的引号采用的形式是" "和' '，而不是“”和‘’。
Dynamic-Link Library (DLL)	第一次遇到缩写字时一般都将把它拼出来。
Microsoft Specific	本书中所列的某些特性有特殊的使用限制。标识异常特性的引导词，后面跟一个箭头，表示这些异常特性的开始。
END Microsoft Specific	**END** 后面跟异常特性的引导词，表示关于有使用限制的特性的文本结束。
△CEnterDlg;	代码边上的箭头表示它与以前的例子有变化，通常是因为要求对它进行编辑。

第1部分

MFC库的概述

第1章　用类编写 Windows 下的应用程序

Microsoft Foundation Class Library(MFC)中的各种类结合起来构成了“应用程序框架”——建立 Windows 下的应用程序是基于这个框架而进行的。从总体上讲,这个框架定义了应用程序的轮廓,并提供了用户接口的标准实现方法。程序员的工作就是向这个轮廓中填入剩余的一些部分,即具体应用程序特有的一些东西。可以用 AppWizard 创建一个初级应用程序的所有文件,以此作为一个良好的开端,用 Microsoft Visual C++ 资源编辑器直观地设计用户接口部分,再用 ClassWizard 把这些部分联接到代码中,最后用类库实现应用程序特定的逻辑。

MFC 框架的 3.0 版本能够支持 Win32 平台的 32 位编程,包括 Microsoft Windows NT 3.5 及以后的版本。MFC Win32 还支持多线程。

本章将泛泛地描述一下应用程序框架,还将探讨构成应用程序的各种主要对象,以及它们是如何创建的。本章将讨论下述问题:

- 运行中的应用程序的主要对象
- 框架和代码之间的任务分工
- 应用程序类,它封装了应用程序级的一些功能
- 文档模板是如何创建和管理文档及其相关视和边框窗口的
- **CWnd**,所有窗口的基类
- 图形对象,如笔和刷子

后面的几章将继续讨论框架,包括:

- 消息和命令,第 2 章“使用消息和命令”
- 文档、视和边框窗口,第 3 章“使用边框窗口、文档和视”
- 第 4 章“使用对话框、控制框和控制条”
- 第 5 章“使用对象链接和嵌入”

除了为编写 Windows 下的应用程序提供一个非常好的开端外,MFC 还使得编写使用对象链接和嵌入(OLE)的应用程序简单了许多。可以把应用程序做成 OLE Visual Editing 封装器、OLE Visual Editing 服务器,或既是封装器也是服务器,这样就可以向应用程序中增加 OLE Automation,使得其它应用程序可以使用其中的对象,甚至对其进行远程驱动。

- 第 6 章,“使用数据库”

MFC 还提供了一组数据库类,简化访问数据库应用程序的编程。使用数据库类就可以通过一个 Open Database Connectivity(ODBC)驱动程序联接到数据库中,从表中选择记录,在屏幕上显示记录信息。

- 第 7 章,“使用通用类”

另外，MFC 完全可以用于编写需要使用 Unicode™和多字节字符集(MBCS)的应用程序，尤其是双字节字符集(DBCS)。

《Visual C++入门》的第 6 至 15 章中有一个教程，它可以一步一步地演示如何用框架建立一个应用程序。下表所示的是其它一些文档：

从这里得到进一步的信息

标　题	书　名	章　节
本书中提到的类	类库参考	按字母顺序参考
AppWizard	*Visual C++用户指南*	第 1 章
ClassWizard	*Visual C++用户指南*	第 12 章
开展环境	*Visual C++用户指南*	第 1－21 章
教程	*Visual C++入门*	第 5－27 章
诊断、例外	用 *Microswoft Foundation Class Library* 编程(本书)	第 2 部分(见“诊断”和“例外”)
宏和全局变量	类库参考	按字母顺序
资源	*Visual C++用户指南*	第 4－11 章

1.1 框　架

使用框架时大部分是基于几种主要的类和几个 Visual C++工具。有些类封装的大部分是 Win32 的应用程序编程接口(API)，而有些类封装的则是应用程序本身的概念，如文档、视和应用程序本身，还有一些类封装的是 OLE 特性和 ODBC 数据访问功能。

1.1.1 SDI 和 MDI

MFC 不仅简化了单文档接口(SDI)，也简化了多文档接口(MDI)的应用程序。

SDI 应用程序一次只允许打开一个文档边框窗口，而 MDI 应用程序允许在应用程序的同一个实例中打开多个文档边框窗口。MDI 应用程序有一个窗口，在其中可以打开多个 MDI 子窗口，而这些子窗口本身就是边框窗口，每个窗口中都有一个独立的文档。有些应用程序中，子窗口可以是不同类型的，如条图窗口和数据表窗口。在这种情况下，激活不同类型的 MDI 子窗口将改变菜单条。

1.1.2 文档、视和框架

框架的中心是文档和视的概念。一个文档是一个数据对象，用户在编辑过程中与这个数据对象进行交互。它是由 File 菜单中的 New 或 Open 命令创建的，一般都保存在一个文件中。一个视是一个窗口对象，用户通过这个窗口对象与一个文档进行交互。

正在运行的应用程序的几个主要对象是：

■ 文档

文档类(由 **CDocument** 派生而来)指定了应用程序的数据。

如果想要应用程序中具有 OLE 功能，则文档类要从 **COleDocument** 或其派生类之一中

派生出来,这要依据所需功能的类型而定。

■ 视

视类(从 **CView** 派生而来)就用户的"数据窗口"。视类指定了用户以什么方式见到文档的数据,以及如何与其进行交互。在有些情况下一个文档可能要有多个视。

如果需要滚动,则视类要从 **CScrollView** 派生。如果视中有一个以对话框模板资源的形式出现的用户接口,则视类要从 **CFormView** 派生。对于简单的文本数据,可以使用 **CEditView** 或从其派生类。对于基于格式的数据访问应用程序,如数据输入程序,视类要从 **CRecordView** 派生。

■ 边框窗口

视是显示在"文档边框窗口"中的。在 SDI 应用程序中,文档边框窗口也是该应用程序的"主边框窗口"。在 MDI 应用程序中,文档窗口是显示在主边框窗口中的子窗口,所派生的主边框窗口类指定了包含视的边框窗口的格式及其它特性。要定制 SDI 应用程序的文档边框窗口,就要从 **CFrameWnd** 派生;而要定制 **MDI** 应用程序的主边框窗口,就要从 CMDIFrameWnd 派生。如果要定制应用程序支持的不同类型的 MDI 文档边框窗口,则每种类型都要从 **CMDIChildWnd** 派生出一个类来。

■ 文档模板

文档模板使得所创建的文档、视和边框窗口和谐地结合起来。一种特定的文档模板类创建并管理着已打开的同种类型的所有文档。支持多种类型文档的应用程序要有多个文档模板。SDI 应用程序要使用 CSingleDocTemplate 类,如果是 MDI 应用程序则使用 **CMultiDocTemplate** 类。

■ 应用程序对象

应用程序类(从 **CWinApp** 派生而来)控制着上述的所有对象,并指定了应用程序的活动,如初始化和清除工作。应用程序的一个且是唯一的一个应用程序对象将创建并管理该应用程序所支持的所有文档类型的文档模板。

■ 线程对象

如果应用程序创建了独立的执行线程,例如,在后台进行计算,则要使用从 **CWinThread** 派生的类。**CWinApp** 本身就是从 **CwinThread** 派生而来的,表示的是应用程序中的主执行线程(或主进程)。在辅线程中也可以使用 MFC。

在运行着的应用程序中,这些对象,用命令和其它消息结合在一块,共同对用户动作进行响应。单个应用程序对象管理着一个或多个文档模板,每个文档模板创建并管理一个或多个文档(要看该应用程序是 SDI,还是 MDI)。用户通过包含在边框窗口中的一个视观察和处理文档。图 1.1 说明了一个 SDI 应用程序中各对象之间的相互关系。

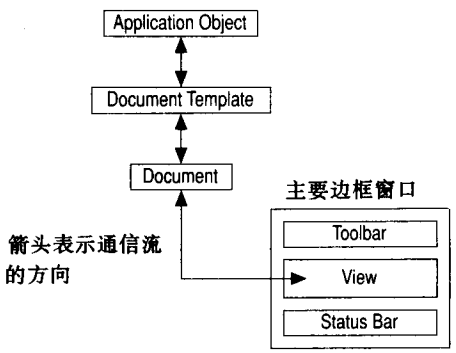

图 1.1 一个运行着的 SDI 应用程序中的各种对象

本章的剩余部分将讨论框架工具,**AppWizard, ClassWizard** 及资源编辑器等是如何创建这些对象的,这些对象是如何协同工作的,

以及如何在编程过程中使用这些对象。文档、视和边框窗口将在第 3 章,“使用边框窗口、文档和视”中详细描述。

1.1.3 AppWizard, ClassWizard 和资源编辑器

在 MFC 编程过程中, Visual C++ 有两个绝好的工具和许多集成的资源编辑器。尽管在编写 MFC 应用程序时也可以不使用这些工具,但使用这些工具将大大地简化编程工作,提高编程速度。

用 AppWizard 创建 MFC 应用程序

利用 AppWizard 创建 Visual C++ 的一个 MFC 项目——它可以包含对 OLE 和数据库的支持。项目中的文件包含了应用程序、文档、视和边框窗口等类;标准资源,包括菜单和可选的工具条;其它必须的 Windows 文件;和可选的含有 Windows Help 主题的.RTF 文件等。

用 ClassWizard 管理类和 Windows 消息

ClassWizard 有助于创建 Windows 消息和命令的处理函数;创建和管理类;创建类成员变量;创建 OLE Automation 的方法和属性;创建数据库类;以及其它一些工作。

提示 ClassWizard 也有助于覆盖 MFC 类中的虚函数。先选类,再选择需要覆盖的虚函数。该过程的其余部分与消息处理是类似的,消息处理将在下面几段中进行描述。

运行在 Windows 下的应用程序都是“消息驱动的”。发生在运行着的程序中的用户动作和其它事件将使 Windows 给该程序的窗口发送一些消息。例如,用户在一个窗口中单击了鼠标器,如果按下的是鼠标器的左按钮,则 Windows 就发送 **WM_LBUTTONDOWN** 消息,松开该按钮时发送 **WM_LBUTTONUP**。如果用户从菜单条中选择了命令,则 Windows 还将发送 **WM_COMMAND** 消息。

在框架中,各种各样的对象——文档、视、边框窗口、文档模板和应用程序对象——都可以“处理”消息。类对象都提供了一个“处理函数”作为其成员函数之一,而框架则把进来的消息都映射给其处理函数。

编程过程的大部分工作就是要选择把哪些消息映射给哪些对象,然后实现这些映射。可以用 ClassWizard 工具来做这些事。

ClassWizard 将创建空的消息处理成员函数,编程人员再用源码编辑器来完成该函数的函数体。

用资源编辑器创建并编辑各种资源

利用 Visual C++ 的资源编辑器创建并编辑菜单、对话框、定制的控制框、加速键、位图、图标、光标、字符串和版本等资源。可以把 ClassWizard 与该编辑器一起使用,例如,创建了对话模板资源以后,可以运行 ClassWizard 把该资源与对话类联接起来。

为了进一步帮助程序员编程, Microsoft Foundation Class Library 还提供了一个叫做 COMMON.RC 的文件,其中含有一些“剪辑艺术”资源,程序员可以从中复制一些资源,再粘贴到自己的资源文件中。COMMON.RC 中含有工具条按钮、通用光标、图标等等。在应用程序中可以使用、修改、重新分发这些资源。关于 COMMON.RC 的进一步信息,参见第 2 部分中的第 52 条“COMMON.RC 实例资源”。

关于各种工具及它们如何协同工作的进一步信息,参见第 2 部分中的第 246 条“MFC 编程工具”。

1.2 基于框架构造应用程序

在用框架配置应用程序时，程序员的任务就是提供应用程序特定的源代码，并通过定义各部分所需要响应的消息和命令把各个组成部分联接起来，利用 C++ 语言和标准 C++ 技术从类库所提供的那些类中派生出应用程序特定的自己的类，并覆盖和增加基类的性能。

表 1.1 说明的是相对于框架所做的工作和程序员所要做的工作。表 1.2 说明的是在创建 OLE 应用程序时，程序员和框架各自所扮演的角色。尽管其中有些步骤有其它选择，例如，应用程序都是从几种类型的视类中选用其中的一种类型，但大多数情况下都可以按照这些步骤创建 MFC 应用程序。

表 1.1 利用框架构造应用程序的顺序

要做的工作	程序员所做的	框架所做的
创建轮廓应用程序	运行 AppWizard。在可选项页中指定所要的选项。可选项包括使应用程序成为一个 OLE 服务器，还是一个封装器，或既是服务器又是封装器；增加 OLE Automation，使用应用程序能识别数据库。	AppWizard 为轮廓应用程序创建各种文件，包括应用程序、文档、视和边框窗口的源文件；创建项目文件(.MAK)或其它文件。所有文件都是按照指定的规格定制的。
看看在不增加程序员自己的代码的情况下，框架和 AppWizard 都提供了什么	构造轮廓应用程序，并在 Visual C++ 中运行这个程序。	运行中的轮廓应用程序将从框架中派生出许多标准的 File, Edit, View 和 Help 菜单命令。对于 MDI 应用程序，还有一个完全可用的 Window 菜单，而且框架将管理 MDI 子窗口的创建、排列和清除。
构造应用程序的用户接口	利用 Visual C++ 的资源编辑器直观地编辑应用程序的用户接口： ■ 创建菜单。 ■ 定义加速键。 ■ 创建对话框。 ■ 创建并编辑位图、图标和光标。 ■ 编辑 AppWizard 创建的工具条位图。 ■ 创建并编辑其它资源。 在对话编辑器中还可以对对话框进行测试。	由 AppWizard 创建的默认资源文件提供了所需的许多资源。Visual C++ 允许很容易地直观地编辑现有资源和增加新资源。
把菜单映射成处理函数。	利用 ClassWizard 把菜单和加速键联接到代码中的处理函数中。	ClassWizard 在程序员指定的源文件中插入消息映射条目和空的函数模板，并管理许多的人工编码工作。

续表

要做的工作	程序员所做的	框架所做的
编写处理函数代码	用 ClassWizard 在源码编辑器中直接跳到代码部分，填入处理函数的代码	ClassWizard 将带出源码编辑器，滚动到空的函数模板，并定位光标。
把工具条按钮映射成命令	通过为按钮分配适当的命令 ID，把工具条上的每个按钮映射成菜单或加速键命令。	框架控制着工具条按钮的绘制、有效、无效、选中和其它视觉上的效果。
测试处理函数	重新建立程序，并用内置的调试工具测试处理函数能否正确运行。	可以单步执行或跟踪代码，看看处理函数是如何被调用的。如果填完了处理函数的代码，则处理函数将执行一些命令。框架将自动使不被处理到的菜单项和工具条按钮无效。
增加对话框	用对话编辑器设计对话模板资源，然后再用 ClassWizard 创建一个对话类和处理该对话框的代码。	框架将管理该对话框，并允许捡取用户输入的信息。
初始化、验证和捡取对话框数据	也可以定义对话框的控制如何被初始化和验证。用 ClassWizard 向对话类中增加成员变量，并把它们映射成对话控制。指定当用户输入数据时适用于各个控制的验证规则。如果希望的话，可以提供自己定制的验证规则。	框架将管理对话框的初始化和验证。如果用户输入了不正确的信息，则框架将显示一个消息框，让用户重新输入数据。
创建附加的类	利用 ClassWizard 创建除由 AppWizard 自动创建的类以外的文档类、视类和边框窗口类。可以创建附加的数据库记录集类、对话类等等	ClassWizard 将把这些类增加到源文件中，并帮助联接到其所处理的命令上。
实现文档类	实现应用程序特定的文档类。增加用于存放数据结构的成员变量，增加成员函数以提供与数据的接口。	框架已经知道了如何与文档数据文件进行交互。它可以打开和关闭文档文件，读写文档的数据，以及处理其它用户接口。程序员可以集中精力确定如何对文档的数据进行操作。
实现 Open、Save 和 Save As 命令	编写文档的串行化成员函数代码。	框架将为 File 菜单中的 Open，Save 和 Save As 命令打开对话框。它将利用在串行化成员函数中所指定的数据格式对文档进行读写。
实现视类	根据文档的需要实现一个或多个视类。实现该视的利用 ClassWizard 映射成用户接口的成员函数	框架管理着一个文档与其视之间的大部分关系。视的成员函数将访问该视的文档，提供用于屏幕显示或用于打印的图象，并根据用户的编辑命令对文档的数据结构进行更新。

续表

要做的工作	程序员所做的	框架所做的
增强默认的打印功能	如果要支持多页打印，则要覆盖视的成员函数。	框架能够支持 File 菜单中的 Print, Print Setup 和 Print Preview 命令。程序员必须告诉它如何把文档分成多页。
增加滚动功能	如果需要支持滚动，则视类要从 **CScrollView** 派生。	当视窗变得太小时，该视将自动增加滚动条。
创建格式视	如果想把视建立在对话模板资源的基础上，则视类要从 **CFormView** 派生	视将利用对话模板资源显示其控制。用户可以用 tab 键在视的各控制之间移动。
创建数据库格式	如果想要一个基于格式的数据访问应用程序，则视类要从 **CRecordView** 派生。	该视与格式视相似，但其控制却是与代表着一个数据库表的 **CRecordset** 对象的各个字段联接在一起的，MFC 在控制和记录集之间传送数据。
创建简单的文本编辑器	如果想要视成为一个简单的文本编辑器，则视类要从 **CEditView** 派生。	该视提供了编辑功能、对 Clipboard 的支持和文件的输入/输出。
增加分割窗口	如果想支持窗口分割，可以向 SDI 边框窗口或 MDI 子窗口中增加一个 **CSplitterWnd** 对象，并把它挂到该窗口的 **OnCreateClient** 成员函数中。	框架在紧挨着滚动条的地方提供了分割框控制，并对把视分割成多个视窗进行管理。如果用户分割了一个窗口，则框架将给文档创建并增加附加的视对象。
建立、测试和调试应用程序	利用 Visual C++ 提供的工具建立、测试及调试应用程序	Visual C++ 允许调整编译、链接和其它可选项，也允许浏览源码和类结构。

表 1.2 说明的是在创建 OLE 应用程序时，程序和框架各自所扮演的角色。这些只表示可能的选择，而不是执行的顺序。

表 1.2 创建 OLE 应用程序

要做的工作	程序员所做的	框架所做的
创建 OLE Visual Editing 服务	运行 AppWizard。从 OLE 选项中选择 OLE Full Server 或 OLE Mini_server。(在已有的应用程序中可以模仿 Scribble 教程中的第 7 步)。	框架将创建一个具有 OLE 服务器能力的轮廓应用程序。只要稍做改动，所有 OLE 功能都可以转到现有的应用程序中。
由草图创建 OLE Visual Editing 封装器应用程序	运行 AppWizard。从 OLE 选项中选择 OLE Container。在 ClassWizard 中，跳到源码编辑器。填入 OLE 处理函数的代码	框架将创建一个轮廓应用程序，在该程序中可以插入由 OLE 服务器应用程序创建的 OLE 对象。

续表

要做的工作	程序员所做的	框架所做的
由草图创建可支持 OLE Automation 的应用程序	运行 AppWizard。从 OLE 选项中选择 Automation Support。用 ClassWizard 列出应用程序中用于实现自动化的方法和属性。	框架将产生一个轮廓应用程序，该应用程序可以被其它应用程序激活并自动运行。

从上可以看出，AppWizard, Visual C++ 的资源编辑器、ClassWizard 和框架为程序员做了大量的工作，而且使得对代码的管理变得容易多了。应用程序特定的代码的主体放在文档和视类中。要探讨真实应用程序的这种过程可以参见《*Visual C++ 入门*》一书：

- 第 6 到 15 章说明了如何使用 MFC 的基础框架。
- 第 16 到 23 章讲述了 OLE 编程方法。
- 第 23 到 27 章讨论数据库编程方法。

尽管用手工或其它工具也可以完成这些工作，但使用这些工具和框架可以节省时间和精力，少出错误。这些都说明使用这些工具和框架是非常有益的。

1.3 框架如何调用代码

理解源码与框架中代码之间的关系是非常重要的。应用程序在运行时，控制流的大部分都在框架代码的内部。当用户选择命令以及编辑视中的数据时，框架管理着用于从 Windows 获取消息的消息循环。框架自身能够处理的事件一点都不依赖于程序员的代码。例如，在响应用户命令时，框架知道如何关闭窗口以及如何退出应用程序。当框架处理这些工作时，它利用消息处理函数和 C++ 虚拟函数也给程序员提供了对这些事件进行响应的机会，但程序员的代码不处在主宰的位置，框架才是主宰者。

框架将调用程序员的代码来处理应用程序特定的事件。例如，用户选择了一条菜单命令后，框架将把这条命令引导到一系列 C++ 对象中：当前视和边框窗口、与该视相关联的文档、该文档的文档模板和应用程序对象。如果这些对象中的一个能处理这条命令，则该对象就调用适当的消息处理函数来处理这条命令。对于任一给定的命令而言，所调用的代码可能是程序员编写的，也可能是框架自己的。

对于具有传统 Windows 编程或事件驱动编程经验的程序员来说，这种安排是比较熟悉的。

在下面的几节中可以看出，在初始化和运行应用程序时，以及在应用程序结束时，框架都做了哪些工作，还可以更清楚地看到所编写的应用程序都适用于什么地方。

1.4 CWinApp：应用程序类

主应用程序类包含了 Windows 下应用程序的初始化、运行和结束。基于框架而建立的应用程序必须有一个(且只有一个)从 CWinApp 派生的类对象。该对象在创建窗口之前构造。

注意 应用程序类组成了应用程序的主执行线程。利用 Win32 的 API 函数还可以创建辅执行线程。这些线程都可以使用 MFC 库。

进一步的信息可以参见第 2 部分中的第 156 条“多线程”。

与所有 Windows 程序一样，框架应用程序也有一个 **WinMain** 函数。但是，在框架应用程序中不用编写 **WinMain** 的代码，它是由类库提供的，在应用程序启动时调用这个函数。**WinMain** 所完成的是诸如注册窗口类等标准服务，然后它再调用应用程序对象中的成员函数来初始化和运行该应用程序。

初始化应用程序时，**WinMain** 将调用应用程序对象的 InitApplication 和 InitInstance 成员函数。要运行应用程序的消息循环时，**WinMain** 将调用 **Run** 成员函数。在程序结束时，**WinMain** 将调用应用程序对象的 ExitInstance 成员函数。图 1.2 说明了框架应用程序中的执行顺序。

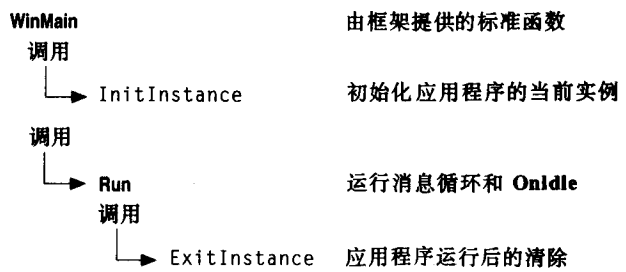

图 1.2 执行顺序

注意 以黑体字印刷的名称表示的是 Microsoft Foundation Class Library 和 Visual C++ 提供的元素。以单间隔字体印刷的名称表示的是程序员创建或覆盖的元素。

CWinApp 和 AppWizard

在创建轮廓应用程序时，AppWizard 将说明一个从 **CWinApp** 派生的应用程序类。AppWizard 还将产生一个含有下述项的实现文件：

- 应用程序类的消息
- 一个空的类构造函数
- 说明该唯一一个对象的一个变量
- InitInstance 成员函数的实现方法

应用程序类放在项目头和主源文件中。类名和文件名都是基于在 AppWizard 中所提供的项目名而创建的。

所提供的标准实现方法和消息映射在许多情况下都是充分可用的，但需要的时候也可以修改。这些实现方法中最有趣的就是 InitInstance 成员函数。一般情况下都需要在 InitInstance 实现方法的轮廓中加代码。

可覆盖的 CWinApp 成员函数

CWinApp 提供了几个关键的可覆盖成员函数：

- **InitInstance**
- **Run**

■ **ExitInstance**

■ **OnIdle**

唯一一个必须覆盖的 **CWinApp** 成员函数是 **InitInstance**。

InitInstance

Windows 允许同一个应用程序有多个副本，或叫做“实例”。每次启动应用程序的一个新的实例时，**WinMain** 都要调用 **InitInstance**。

AppWizard 创建的 InitInstance 的标准实现方法可以完成下述工作：

■ 创建文档模板，这是其中心工作，从而创建文档、视和边框窗口。关于这一过程的描述，参见 1.6 节“文档模板”。

■ 从 .INI 文件中载入标准的文件选，包括最近使用的一些文件的文件名。

■ 注册一个或多个文档模板。

■ 对于 MDI 应用程序，创建一个主边框窗口。

■ 处理命令行，以打开命令行中指定的文档，或打开一个新的空文档。

Run

框架应用程序的大部分时间都花在 CWinApp 类的 Run 成员函数上。初始化完成后，WinMain 将调用 Run 来处理消息循环。

Run 不断执行消息循环，检查消息队列中有没有消息。如果有消息，Run 将把它分发下去，以便采取动作。如果没有消息——经常是这种情况，Run 将调用 OnIdle 来做用户或框

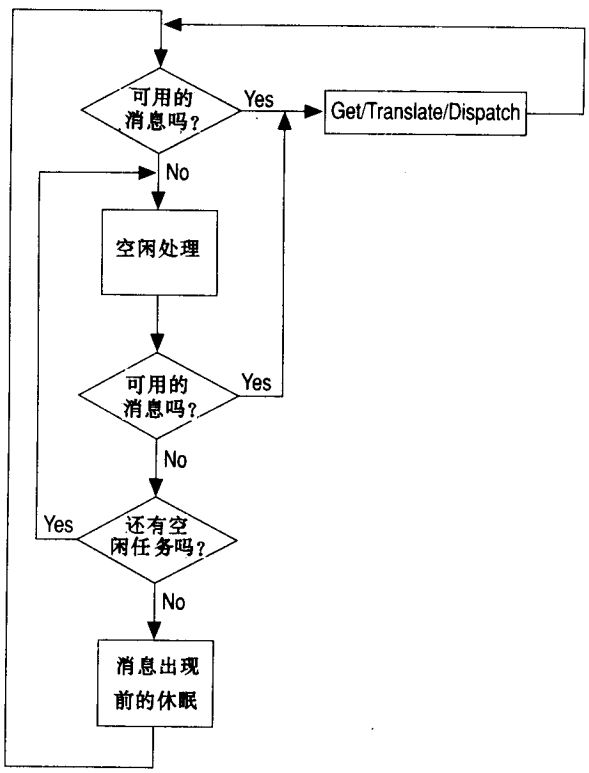

图 1.3　消息循环

架可能需要做的空闲时的处理工作。如果既没有消息要处理，也没有空闲时的处理工作要做，则应用程序将一直等待，直到有事情发生。当应用程序结束时，Run 将调用 ExitInstance。图 1.3 说明了消息循环中的动作顺序。

消息的分发要依赖于消息的类型。进一步的信息可参见第 2 章“使用消息和命令”。

ExitInstance

每次应用程序的一个副本结束，通常是用户要退出该应用程序时，都要调用 **CWinApp** 类的 **ExitInstance** 成员函数。如果需要做特殊的清理工作，如释放图形设备接口(GDI)资源或释放程序执行期间占用的内存，则可以覆盖 **ExitInstance**。然而，框架提供了对一些标准工作的清理工作，如文档和视，而对那些需要做特殊清理工作的对象，则提供了其它方法：覆盖的函数。

OnIdle

如果没有 **Windows** 消息要处理，则框架将调用 **CWinApp** 的 **OnIdle** 成员函数(在《类库参考》一书中作了描述)。如果要执行后台任务，就需要覆盖 **OnIdle**。该成员函数的默认版本将更新用户接口对象的状态，如工具条按钮，并完成框架在其运行过程中创建的临时对象的清理工作。图 1.3 说明了在队列中没有消息的情况下消息循环是如何调用 **OnIdle** 的。

1.5 特殊的 CWinApp 服务

CWinApp 除了运行消息循环，提供初始化应用程序的机会，及在程序结束后做一些清理工作外，还提供了其它几种服务。

Shell 的注册

在默认的情况下，对于由应用程序创建的数据文件而言，AppWizard 允许用户在 Windows File Manager 中双击这些文件，以打开它们。如果应用程序是一个 MDI 应用程序，并在应用程序创建文件时指定了文件扩展名，AppWizard 将在 InitInstance 中增加对 **CWinApp** 的 **EnableShellOpen** 和 **RegisterShellFileTypes** 成员函数的调用，覆盖掉 **AppWizard** 为程序员编写的代码。

RegisterShellFileTypes 将向 File Manager 注册应用程序的文档类型。该函数将向由 Windows 维护的注册数据库中增加一些条目。这些条目注册了所有文档类型，把文件扩展名与文件类型关联起来，指定打开应用程序所用的命令，并指定了打开某类文件所用的动态交换(DDE)命令。

EnableShellOpen 允许应用程序接收来自 File Manager 的 DDE 命令，以打开用户选中的文件，从而完成了全过程。

由于 **CWinApp** 能支持这种自动注册，所以就有必要把一个.REG 文件与应用程序一起发行，或做特殊的安装工作。

File Manager 的拖放

Windows 3.1 及以后的版本允许用户从 File Manager 的文件视窗中拖动一些文件名，把它们放到应用程序的一个窗口中。例如，程序可以允许用户把多个文件名拖到 MDI 应用程序的主窗口中，这样，应用程序就可以检取这些文件，并为这些文件打开 MDI 子窗口。

为了在应用程序中能够使用文件拖放，在 InitInstance 中，AppWizard 为主边框窗口写

进了调用 **CWnd** 的 **DragAcceptFiles** 成员函数的代码。如果不想实现拖放特性，可以去掉这个函数调用。

注意 还可以用 OLE 实现更通用的拖放功能——在文档内部或文档之间拖动数据。关于这方面的信息，参见第 2 部分 91 条拖放(OLE)。

跟踪最近使用的文档

随着用户打开和关闭一些文件，应用程序对象可以跟踪四个最近使用的文件，这些文件的名称放到了 File 菜单中，且随文件名的变化而不断更新。框架把这些文件名存入与项目名同名的.INI 文件中，当应用程序启动时再从这个文件中把它们读出来。AppWizard 创建的用于覆盖 InitInstance 的代码，包含了对 **CWinApp** 的 **LoadStdProfileSettings** 成员函数的调用，这个函数调用将从.INI 文件中载入信息，包括最近使用的文件名。

1.6 文档模板

为了管理创建与视和边框窗口相关的文档这一复杂过程，框架使用了两种文档模板类：用于 SDI 应用程序的 **CSingleDocTemplate** 和用于 MDI 应用程序的 **CMultiDocTemplate**。**CSingleDocTemplate** 一次可以创建和存储一种类型的文件。**CMultiDocTemplate** 保存了一种类型的多个打开文档的一个列表。

有些应用程序能支持多种文档类型。例如，应用程序可以支持文本文档和图形文档。在这样的应用程序中，当用户在 File 菜单中选择了 New 命令时，对话框中将显示要打开的可能的新文件类型的一个列表。对于所支持的各种文档类型，应用程序将使用不同的文档模板对象。图 1.4 说明了一个能支持两种文档类型的 MDI 应用程序的配置。该图显示了几个打开的文档。

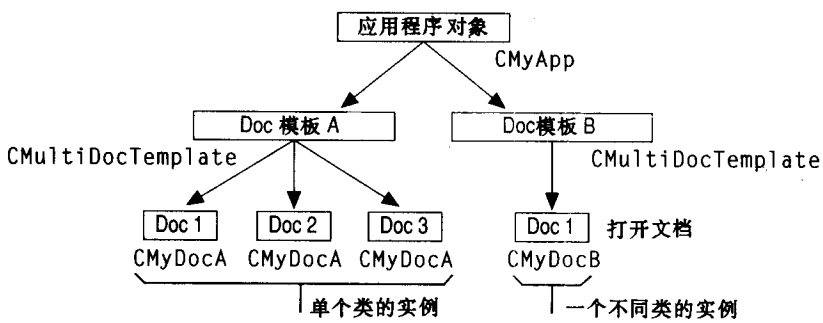

图 1.4 一个有两种文档类型的 MDI 应用程序

文档模板由应用程序对象创建和维护。在应用程序 InitInstance 的执行期间所完成的关键任务之一，就是构造一个或多个适当类型的文档模板。这一特性将在下面的"文档模板的创建"中讲述。应用程序对象在其模板列表中保存了一个指向其模板列表中各个文档模板的指针，并提供了增加文档模板的接口。

如果要支持两种或多种文档类型，则对每种文档类型都要增加一次调用 **AddDocTemplate**。

1.6.1 文档模板的创建

在响应 File 菜单中的 New 或 Open 命令创建一个新文档时，文档模板也要创建一个新的边框窗口，以便察看文档。

文档模板的构造函数指定了模板可以创建的文档、窗口和视等的类型。这是由传给文档模板构造函数的变量决定的。下面的实例应用程序代码说明的是创建 **CMultiDocTemplate** 的情况：

```
AddDocTemplate(new CMultiDocTemplate(IDR_SCRIBTYPE,
        RUNTIME_CLASS(CScribDoc),
        RUNTIME_CLASS(CMDIChildWnd),
        RUNTIME_CLASS(CScribView)));
```

指向新的 **CMultiDocTemplate** 对象的指针被用作 **AddDocTemplate** 的指针。传给 **CMultiDocTemplate** 构造函数的变量包括与文档类型的菜单和加速键相关联的资源 ID，以及 **RUNTIME_CLASS** 宏的三次利用。**RUNTIME_CLASS** 返回的是以其变量为名称的 C++类的 **CRuntimeClass** 对象。传给文档模板构造函数的三个 **CRuntimeClass** 对象，提供了在文档创建过程中创建指定类的新对象所需的信息。上面的例子说明的是创建带有 CScribView 对象的 CScribDoc 对象所要求创建的文档模板。各个视都由标准的 MDI 子边框窗口分开。

1.6.2 文档/视的创建

框架提供了在 File 菜单中实现 New 和 Open 命令(还有其它命令)的方法。一个新文档及与其相关的视和边框窗口的创建工作，是应用程序对象、文档模板、新创建的文档和新创建的边框窗口等共同合作的结果。表 1.3 总结出了哪个对象创建什么东西。

表 1.3 对象的创建者

创建者	所创建的东西
应用程序对象	文档模板
文档模板	文档
文档模板	边框窗口
边框窗口	视

1.6.3 MFC 各对象之间的关系

为了帮助读者在头脑中显现文档/视的创建过程，先想一想一个运行着的程序都有什么东西：文档、用于包含视的边框窗口，以及与文档相关联的视。

- 文档中有该文档的视的列表，以及一个指针指向创建该文档的文档模板。
- 视中有一个指向其文档的指针，视是其父边框窗口的一个子窗口。
- 文档边框窗口有一个指向其当前活动视的指针。
- 文档模板有其已打文档的一个列表。
- 应用程序中有其文档模板的一个列表。