第一部分

WINDOWS、VISUAL C++和应用框架基础

第一章

Microsoft Windows 和 Visual C++

Microsoft Windows 已经广泛被人们所接受，人们越来越多地体会到了图形用户界面(GUI)给人们带来的好处，关于这点已经有许多的书籍都专门作了介绍。本章总结了 Windows 编程模式，并且介绍了 Visual C++的各组成部分是如何相互配合来帮助人们编写 Windows 应用的。同时，读者也可以在本章中学到有关 Windows 的一些新的东西，回顾过去是为了放眼未来。

1.1 Windows 的编程模式

不管人们使用什么样的开发工具，Windows 程序设计同以往已经过时的批命令或面向事务程序设计已经有了根本的不同。首先读者必须了解一些有关 Windows 的基础知识，我们将参照大家熟知的 MS-DOS 编程模式来进行介绍。尽管读者现在可能已不再编写纯粹的 MS-DOS 应用，但相信读者对 MS-DOS 编程还是比较熟悉的。

1.1.1 消息处理

当用 C 来编写 MS-DOS 应用时，最起码要有 main 函数。当用户运行该应用时操作系统会自动调用 main，然后人们就可以使用任何所需要的程序结构。如果程序需要得到用户键盘输入，或者需要使用操作系统所提供的功能，它就可以调用适当的函数，如 getchar，或者使用基于字符的窗口库。

当 Windows 操作系统运行程序时，它首先调用程序中的 WinMain 函数。因此，在 Windows 应用程序中一定要有 WinMain 函数，该函数一般用来完成某些特殊的任务，其中最重要的任务之一就是要创建应用的“主窗口”。“主窗口”中必须包含用来处理 Windows 所发送的消息的代码。基于 Windows 的程序和基于 MS-DOS 的程序之间最基本的一个差别就在于 MS-DOS 程序通过操作系统来获得用户输入，而 Windows 程序则是通过操作系统发送的消息来处理用户输入。

说明：许多 Windows 开发环境，包括使用 Microsoft 基本类库 2.0 版的 Microsoft Visual C++，都通过隐藏 WinMain 函数及构造消息-控制机制来使编程得到简化。虽然使用类库编程不再需要 WinMain 函数，但是弄清楚操作系统和程序之间的这种联系是非常有用的。

许多 Windows 消息都经过了严格的定义，并且会发送给所有的应用。例如，当窗口被

创建时就会自动发送 WM_CREATE 消息，当用户按下鼠标的左按钮时就会自动发送 WM_LBUTTONDOWN 消息，当用户敲了一个字符键时就会自动发送 WM_CHAR 消息，而当用户关闭窗口时系统又会自动发送 WM_CLOSE 消息。当用户进行菜单选择时，系统又会相应地发送一些其它消息("命令"消息)，而这些消息则完全依赖于应用的菜单设计。当然，程序员还可以定义一些其它消息，这些消息被称作"用户消息"。

读者完全用不着担心如何用代码来管理这些消息，因为这些都是应用框架的事情。不过需要注意的是，Windows 的这种消息处理机制同时也强加给了用户程序许多固定的结构。因此，不要强迫 Windows 程序看起来象老式的 MS-DOS 程序，只要仔细研究一下本书中的例子，读者就不难发现这些程序看起来的确与以往的程序有所不同。

1.1.2 Windows 的图形设备接口(GDI)

许多 MS-DOS 程序都直接往视频存贮区或打印机口输送数据，这种做法的不利之处在于需要对每种显示卡或打印机类型提供相应的驱动程序。Windows 则提供了一个抽象的接口，称作图形设备接口(GUI)。Windows 已经为人们提供了各种显示卡及打印机的驱动程序，这样人们就可以不必关心与系统相连的显示卡及打印机类型。人们可以通过调用 GDI 函数来和硬件打交道，而各种 GDI 函数会自动参考被称为设备环境的数据结构。Windows 会自动将设备环境结构映射到相应的物理设备，并且会提供正确的输入/输出指令。GDI 在处理速度上几乎和直接进行视频访问一样快，并且允许 Windows 的不同应用共同享用显示器。

1.1.3 基于资源的程序设计

在 MS-DOS 下，为了实现数据驱动的程序设计，必须将数据定义为初始化常量，或者提供单独的数据文件供程序读取。而当进行 Windows 程序设计时，可以用一些特定的格式将这些数据存贮在资源文件中，Windows 通过"联编"过程将资源文件嵌入到连接程序中。资源文件可以包含位图、标象、菜单定义、对话框设计和字符串，甚至可以包含用户自定义格式。

人们通常通过文本编辑器来编辑程序，而对各种资源则通过"所见即所得"(wysiwyg)风格的工具来加以编辑。例如，人们在设计对话框时，首先从控制模板标象组中选取适当的元素，然后再对按钮、列表框等通过鼠标来进行定位和设置尺寸。利用 Visual C++的 App Studio 资源编辑器，人们可以对大部分的资源进行有效地编辑。(说明：在 Microsoft Visual Basic 和 Microsoft Access 中，控制模板又被称作工具箱(tool box))

1.1.4 内存管理

在过去，MS-DOS 的传统内存被限制在 640KB 之内，这就限制了程序的大小。尽管人们可以使用各种覆盖管理技术和扩展/扩充内存管理器来弥补这一缺陷，但都存在不尽人意之处。基于 80386SX 的(或更好的)计算机通常具有 4MB 或更多的内存，并且它的 CPU 自带内嵌内存管理硬件。Windows 以及 Visual C++的编译器将会为人们提供更加完备

的内存管理机制，使人们完全用不着再为内存而担忧。

第 9 章详细介绍了目前对 Windows 的内存管理技术。如果读者觉得锁住内存把柄、形式替换程序(thunk)、以及内存申请管理太复杂的话，不用担心，那已成为过去。现在人们可以非常简单地申请到所需要的内存，至于细节问题完全由 Windows 来控制。程序中的有些部分，包括各种资源，可以随心所欲地在硬盘和内存之间调来调去，只要用户计算机具备了必要的内存，那么用户的程序就完全可以在这些内存中腾挪施展得开。

1.1.5 动态连接库(DLLs)

在 MS-DOS 环境下，所有程序的目标组件在创建过程中都被静态地连接起来，而 Windows 则允许动态地连接，即一些特定结构的库可以在运行过程中被装入和连接，并且多个应用可以共享同一个动态连接库，这可大大节省内存和磁盘空间，同时动态连接还可以大大增强对程序的调试灵活性，因为人们可将动态连接库单独编译和调试。

设计者起初创建 DLL 时完全是为 C 语言而设计的，C++则使这一问题变得有些复杂。经过不懈的努力，Microsoft 基本类库的开发者终于将应用框架的所有类组合进了动态连接库中，并且人们还可以创建自己的基于 Microsoft 基本类库的 DLL。第二十六章详细介绍了如何创建 DLL。

1.1.6 新的 Windows 特性——OLE 和 TrueType

Windows 操作系统已经发展演化了数年，最近新增的两个特性就是对象的连接与嵌入(OLE)和 TrueType 字体。借助于 OLE，人们可以将不同应用所创建的对象组合起来，从而大大增强程序的功能。例如，人们可以将图表、声音和图示等包含进同一个文档中。当人们激活某个 OLE 对象时，创建该对象的 Windows 程序允许人们对该对象进行操作。使用 Windows 的 SDK 来进行 OLE 程序设计是非常困难的，然而正如读者在第二十五章将要看到的，Microsoft 基本类库大大简化了这一工作。

TrueType 字体彻底改变了文本在屏幕或打印机上的输出质量。由于人们可以将这些字体变倍到任意尺寸，并且可以将它们在任何打印机上进行输出，这就真正消除了将显示字体与打印字体相匹配这一负担。

1.1.7 Windows NT

正当 Microsoft 公司推出 Visual C++的时候，Windows NT 已进入了最后的 beta 测试阶段。这种新的操作系统包含了一个更高级的文件系统，它具有保密性、多线性、真正的先入为主的多任务性、更高级的网络性能、以及可以在 RISC 体系结构的机器上运行等特点。在 Windows NT 环境下，既可以运行现存的基于 Windows 的 16 位应用，也可以运行新的增强的基于 Windows 的 32 位应用。

如何开发 Windows 的 32 位应用呢？早期的 beta 测试者不得不借助于 Win32 SDK，它包含了一个新的 C 语言应用程序接口(API)。由于需要处理 32 位参数，因此许多 Win32 函数原型同相应的 16 位函数原型相比都有所不同，并且还新增了不少函数，尤其在有关磁盘 I/O 方面新增了不少函数。Windows 的 32 位应用通过 Win32 API 来访问文

件，而不再通过 MS-DOS API。

Win32 SDK 包含了一个扩充部分，称作 Win32s，利用它所开发的 32 位应用既可以在 Windows 3.1 上运行，也可以在 Windows NT 上运行。然而这些应用无法支持一些高级的 NT 特性，因此 Win32s 只能算是 Win32 API 的一个子集。

要想将现存的用 C 语言编写的 Windows 16 位应用转化为真正的 32 位应用需要做大量的转化工作，然而对于 Microsoft 基本类库应用只须重新编辑一遍即可做到这点，因为 Microsoft 基本类库在设计时完全考虑到了 Win32 API。当 Windows NT 正式推出之时，Visual C++的 Win32 版本也将同时出台，人们可以用 32 位版的 Visual C++来产生既可以在 Windows NT 上运行又可以在 Win32s 上运行的应用。关于 Visual C++的 Windows NT 版本的有关情况详见附录 D。

1.2 Visual C++的组成

Visual C++包含了两套完整的 Windows 应用开发系统。如果人们愿意，仍然可以使用 Windows SDK 所提供的 API 来开发用 C 语言编写的 Windows 应用。Windows 的 SDK 程序设计技术已经广为人知，并且已经有许多书籍对此专门作了介绍，其中包括 Charles Petzold 的《Programming Windows 3.1》(Microsoft 出版局 1992)。人们可以利用 Visual C++所提供的一些新的工具使编写 Windows SDK 风格的程序更加方便。

然而本书并不是关于如何用 Windows 的 SDK 来进行 Windows 程序设计的，而是关于如何用 Visual C++所提供的 Microsoft 基本类库应用框架来进行 C++程序设计。在这里，人们将使用《Class Library Reference》一书中所介绍的 C++类，并且将会使用应用框架的专用 Visual C++工具，如 AppWizard 及 ClassWizard 等。

说明：使用 Microsoft 基本类库程序设计接口并不意味着人们不能使用 Windows 的 SDK 函数，实际上人们几乎总是需要在类库程序中直接调用 Windows 的 SDK 函数。

迅速地浏览一下 Visual C++的各组成元素将有助于人们了解应用框架。图 1-1 描述了 Visual C++应用的大致创建过程。

1.2.1 Visual 工作平台和创建过程

Visual 工作平台是一个运行于 Windows 上的交互式开发环境，它是直接从 Microsoft QuickC for Windows 演化而来的。如果读者已经习惯于在命令行上运行编译器这一工作方式，那么不妨试试 Visual 工作平台。请读者相信我，Visual 工作平台确实不错。我从来就不使用旧的字符状态程序员工作平台，但现在我利用 Visual 工作平台来创建我的所有的工程，本书中所有的例子都是在 Visual 工作平台上完成的。

两个 Visual C++版本：
标准版和专业版

市面上可以买到的 Visual C++有两个版本：标准版和专业版。标准版提供了

开发 Windows SDK 应用和基于应用框架的 Windows 应用所有必须的软件配置。专业版还包括一些附加的可打印文档材料、编译器的优化版本和一些附加工具，同时还具有产生 MS-DOS 应用及 P-code Windows 应用的能力。读者可要看清楚了，如果需要编写 MS-DOS 应用的话，必须购买价钱贵一些的专业版。

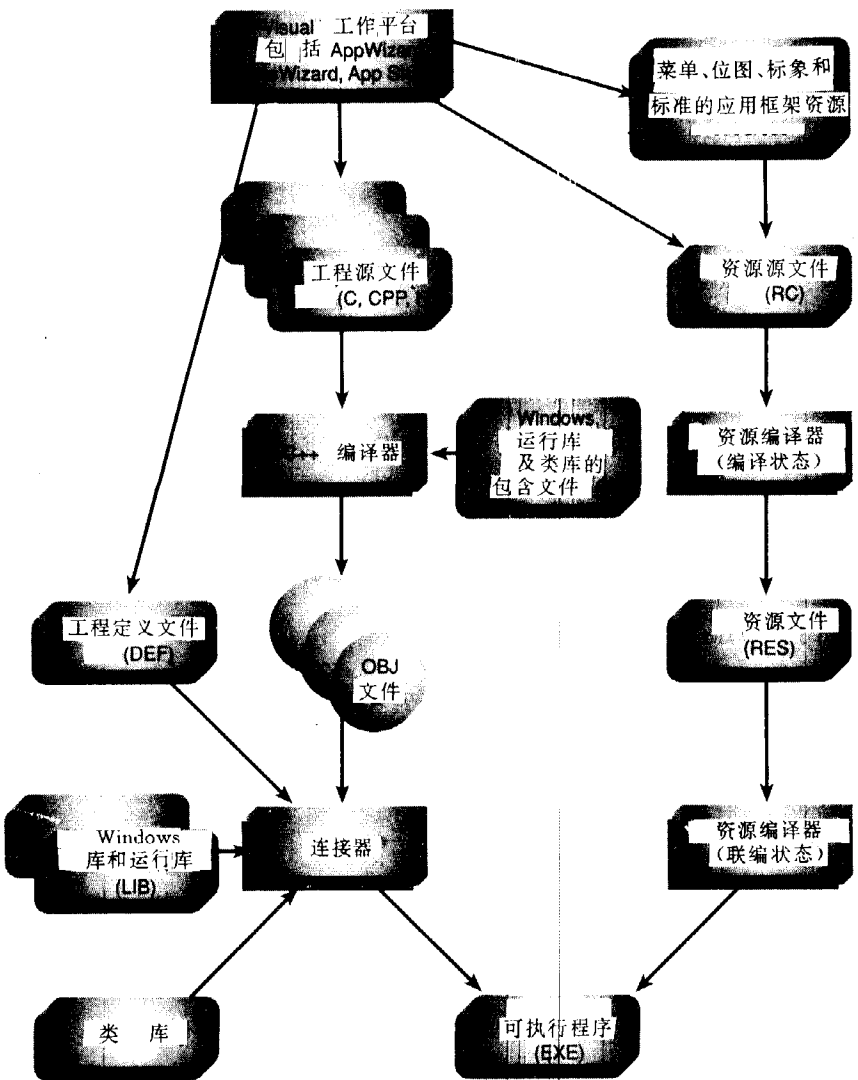

图 1-1 Visual C++应用的创建过程

如果读者使用过 QuickC for Windows 程序员工作平台，或者 Borland IDE，那么就已经知道了 Visual 工作平台是如何工作的了，但如果读者对集成开发环境还比较陌生的话，那么首先必须弄清楚什么是工程(project)。“工程”是一些相互关联的源文件的集合，这些源文件经过编译、连接，然后被组合在一起形成可执行的 Windows 应用程序。工程源

文件一般被存储在一个单独的子目录中，工程也需要依赖于该子目录之外的许多文件，如包含文件和库文件。

熟练的程序员对于 make 文件不会感到陌生，它表述了所有源文件之间的关系（源代码文件需要的特定包含文件，可执行文件要求包含的目标文件模块及库等等）。创建程序首先读取 make 文件，然后再激活编译器、汇编器、连接器和资源编译器来产生最后的输出，最后输出并生成的通常是可执行文件。创建程序利用内在推理规则来激活编译器，以便通过对特定 CPP 文件的编译来产生特定的 OBJ 文件。

在命令行工作方式下，人们必须手工编写 make 文件，而 Visual 工作平台则能够自动生成 make 文件，不过此时称为“工程文件”(project file)。在大多数情况下，人们可能希望工程文件中所包含的所有源文件都被放在工程子目录下，但人们也可以不这么做，而把一些源文件放到其它子目录下。一旦创建了某个工程，人们就可以在不同的子窗口中来分别编辑源文件。Visual 工作平台能够“记住”当前人们正处理的源代码文件，并且还会保存一份最近使用过的工程列表。作为工程的一部分，人们还可以通过一系列对话框来保存所指定的编译器和连接器的开关设置。人们只要从 Visual 工作平台 Project 菜单中选择 Build 命令，系统就会自动生成可执行程序。

Visual 工作平台还包含了一个完全符合 Windows 界面标准的文本编辑器，该编辑器会用颜色来加亮 C++文法。不幸的是，人们或许还未能完全适应该编辑器，也不能安装自己的编辑器。如果人们真的决定使用自己的编辑器，那么必然会失去集成开发环境所提供的连续性。例如，当创建某个工程发现源代码文件中出现错误时，Visual 工作平台会自动加亮出现错误的文本行，同时允许人们设置追踪断点。

1.2.2 App Studio 资源编辑器

最初的 Windows SDK 使用不同的工具来分别对对话框、位图和字体等进行编辑，而在 Visual C++中，人们可以使用 App Studio 来编辑大部分的资源，第三章列出了一些 App Studio 的窗口（详见第三章 3.6.2 节）。App Studio 不但包含了一个所见即所得菜单编辑器，而且还包含了一个功能非常强大的对话框编辑器，该对话框编辑器比旧的 Windows SDK DIALOG 程序要优越得多。人们也可以将 App Studio 作为进行 Windows SDK 程序设计的资源编译器，但当将 App Studio 用于类库程序设计时，可以交互式地在对话框中插入 Visual Basic 控制，以便以后和 C++代码相连接。

App Studio 的内部文件格式和 ASCII Windows 资源(RC)文件格式相同，并且每个工程通常都有一个 RC 文件，在该文件中通过 #include 说明来将位于其它子目录下的资源包含进来。我们不主张在 App Studio之外对 RC 文件进行编辑。App Studio 同样也可以处理 EXE 和 DLL 文件，因此，我们可以通过裁剪板来从其它 Windows 应用中“偷”一些资源，如位图、标象等。

App Studio 堪称是最出色的 Windows 应用，同时它本身就是通过使用 Visual C++工具及类库编写而成的，这一点具有非常重要的意义。App Studio 甚至可以对自身的资源进行编辑！不信读者可以试试。（不过读者需要将 App Studio 拷贝一份，然后将其拷贝装载进来。）

1.2.3 C/C++编译器

Visual C++的编译器可以处理C和C++源代码，它通过源代码文件名后缀来识别代码本身所使用的语言。C后缀代表C源代码，CPP或CXX则代表C++源代码。该编译器完全遵从ANSI 2.1版，并且增加了一些Microsoft扩充内容。Visual C++版本1.0不支持模板和异常文法(template and exception syntax)。编译器通过Visual工作平台的Use Microsoft Foundation Classes选项(该选项在Project Option对话框中)来决定是否使用Microsoft基本类库包含文件。

1.2.4 连接器

为了生成EXE文件，Visual C++的连接器需要对编译器所生成的OBJ文件进行处理。如果用户在Visual工作平台选项中指定了Use Microsoft Foundation Classes选项，那么连接器就会使用相应内存模式的Microsoft基本类库文件。

1.2.5 资源编译器

在编译状态和联编状态都要用到资源编译器。在编译状态，资源编译器会对App Studio所生成的ASCII资源(RC)文件进行编译，生成相应的二进制RES文件；在联编状态，RES文件被嵌入到了可执行(EXE)文件中。如果RES文件被更新，那么只须将它重新联编进它的EXE文件中，而无须重新进行连接。

1.2.6 调试器

如果程序被第一次运行，那么用不着使用调试器，可是以后就少不了一次又一次地使用调试器了。Visual C++调试器首次提供了一个Windows上的C++调试环境。调试器和Visual工作平台紧密配合以便保证断点被保存在磁盘上。通过工具条按钮可以设置断点，控制单步执行。图1-2显示出了正在运行的Visual C++调试器，注意Local窗口是如何对对象指针进行扩展以便显示出派生类和基类的所有数据成员的。为了能够对程序进行调试，在创建程序时必须设置编译器和连接器相应的选项以便产生相应的可调试信息。

专业版的Visual C++在提供了基于Windows上调试器的同时，还提供了字符状态CodeView。基于Windows的CodeView可以调试p-code，同时它还具有一些其它非常有用的特性。

1.2.7 AppWizard

AppWizard是一个代码发生器，它会按照用户通过对话框指定的特性、类名及源代码文件名来产生Windows应用的工作构架。读者在学习本书中所提供的例子时需要经常使用AppWizard。不过，不要把AppWizard同传统的代码发生器如Caseworks CASE:W和Blue Sky WindowsMAKER等混淆起来。AppWizard所产生的代码只是一些最基本的代码，它所完成的功能完全由应用框架的基类所决定，其目的在于使人们能够尽快开始新

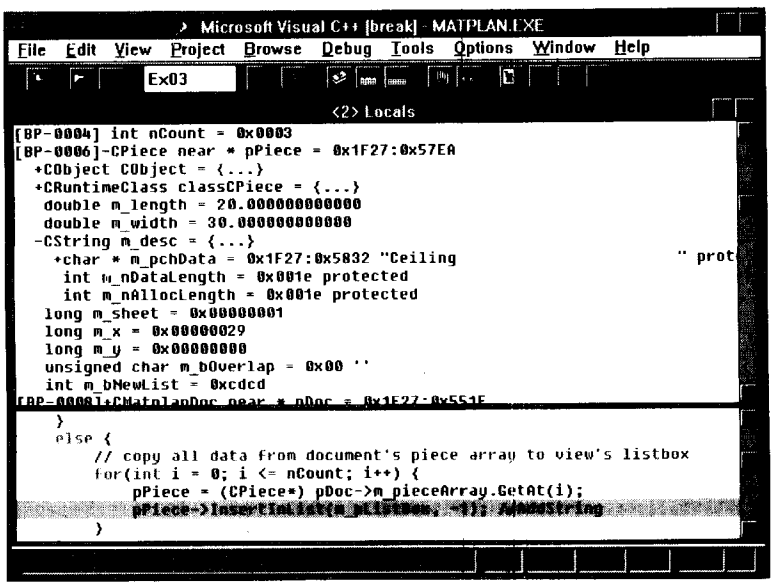

图 1-2　Visual C++调试窗口

的应用。

1.2.8　ClassWizard

ClassWizard(它是通过 DLL 来实现的)既可以在 Visual 工作平台中被运行,也可以在 App Studio 中被运行,它极大地方便了 C++类代码的编写工作。想增加新的类或函数来控制 Windows 消息吗?那么 ClassWizard 能够给出原型、函数体以及将消息同应用框架相联系的相应代码。ClassWizard 还可以被用来对用户自己编写的类的代码进行修改,这就解决了令一般代码发生器感到头疼的维护问题。

1.2.9　源程序浏览器

如果某个程序是读者自己从头到尾编写的,那么您一定会对其中的源代码文件、类及成员函数了如指掌,但如果遇到别人编写的程序,那么您若想搞清楚就需要一些帮助了。Visual C++的源程序浏览器(简称"浏览器")能够使人们从类或函数的角度来了解(和编辑)程序,而不是直接从文件入手,这有点象其它的面向对象库(如 Smalltalk)的"检查器"。浏览器有如下几种观察状态:

- Definitions and References——如果人们选择任何函数、变量、类型、宏定义或者类,都会马上看到它在工程中是在什么地方定义的,并且在哪些地方用到它。
- Call Graph/Caller Graph——对于所选择的函数,给出它的调用与被调用函数图示。
- Derive Class Graph/Base Class Graph——给出类层次关系的图形表示。对于所选择的类,人们可以观察它的派生类或者基类,并且可以利用鼠标来控制层次结构

的扩展。

第三章第三节“不做任何事情的应用”中的浏览应用部分给出了一个典型的浏览器窗口。

说明：如果人们重新安排了源代码文件中的代码行，那么必须重建浏览器数据库。

1.2.10 联机帮助

有关 Windows SDK 参考手册及类库参考手册的全部内容都包含在 Visual C++的联机帮助中，当然 App Studio、AppWizard 和 ClassWizard 也有相应的帮助。读者千万别小看了这些帮助，连 Microsoft 的许多程序员都要经常使用它。如果人们想要查阅有关某个函数的帮助信息，只须在 Visual 工作平台的编辑器中，在该函数出现的地方用鼠标点一下（或将光标移到该函数处），然后按下 F1 键，此时就会出现帮助窗口，如图 1-3 所示。

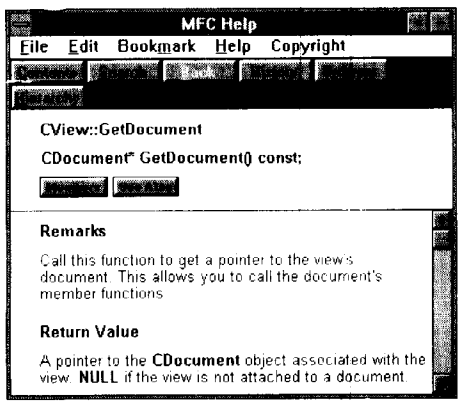

图 1-3 Visual C++的帮助窗口

Visual C++的帮助解决了 Windows SDK 函数和相应的 Microsoft 基本类库函数名之间的冲突问题。如果人们选择的函数名和好几个类的成员函数名相同的话，人们可以从列表框中选择所要的类。如果人们要求查看有关某个类的帮助信息的话，该类的成员函数和数据成员就会按功能被分门别类依次列出。

1.2.11 Windows 诊断工具

专业版 Visual C++中所包含的诊断工具同 Windows 的 SDK 作为单独产品时所包含的诊断完全相同：用于观察 Windows 消息的 SPY，进行内存检查的 HEAPWALK，用于编译帮助文件的 HC31，用于人工限制可使用内存的 STRESS，以及用于反映代码瓶颈的剖析程序。

Visual C++的标准版和专业版中都包含有用来显示诊断结果的 DBWIN，以及用于对手工编制的 make 文件进行处理的 NMAKE 程序。NMAKE 主要用来创建非标准的 Microsoft 基本类库应用。

1.2.12 Microsoft 基本类库 2.0 版

Microsoft 基本类库 2.0 版(简称类库)是本书所要讨论的主题,它定义了马上就要讲到的应用框架。从第二章开始将给读者介绍一些实际代码及一些重要概念。

第二章

Microsoft 基本类库应用框架

本章介绍了 Microsoft 基本类库 2.0 版(简称类库)应用框架,并着重阐述了其优点。首先读者将会看到一个只具基本骨架但却完全能够运行的 Windows 类库程序,它可以帮助读者理解应用框架程序设计究竟是怎么一回事。本章力求少介绍一些理论,但有关消息映射及文档——视部分所介绍的内容对于理解和掌握以后几章所提供的例子是非常有帮助的。

2.1 为什么要使用应用框架

如果要开发 Windows 应用,首先得挑选一种开发环境。假设读者已经放弃了交互式开发环境,如 Microsoft Visual Basic,那么就必须在下面的选择中进行挑选:

- 久经考验的 Windows SDK(Software Development Kit)
- 全新的 Microsoft 基本类库应用框架
- 其它的基于 Windows 的应用框架,如 Borland 公司的对象 Windows 库(Object Windows Library, OWL)

如果读者对以上几种选择都不了解,那么选择任何一种都需要很长的学习过程,但读者如果已经是一位 Windows SDK 程序员,对于类库仍然需要有一个学习和掌握的过程。那么付出这些努力会得到哪些好处呢?

Microsoft 出版局的编辑们建议我不要使这部分听起来象广告,但我实在是忍不住。我确实为该产品感到欢欣鼓舞,要知道它是我十年来梦寐以求的应用开发环境。

我将我所体会到的类库的好处介绍如下。

类库版本 2.0 是 C++的 Microsoft Windows API。读者是否同意这点取决于您对 C++的接受程度。如果 C++取代了 C,正如许多人所期望的那样,那么 Windows 自然就应该有一个 C++程序设计接口。不过,和 Windows 所不同的是,这种 C++接口标准可以由任何公司来制定,而不一定非得由 Windows 本身的开发者 Microsoft 公司来决定。

我坚信 C++必将取代 C,这不仅是因为它是唯一的被全球广泛接受的面向对象语言,而且还在于对于那些希望由可重用及模块化组件构成的大的软件项目来说,面向对象程序设计已是必然的选择。随着人们所需软件复杂性的日益增加,C 程序越来越显得力不从心了!

应用框架所产生的应用使用了标准化的结构。任何程序员在考虑大的项目时都会首先设计自己的程序结构,问题在于不同的程序员所设计的程序结构是截然不同的,这对于

该项目的新增成员来说要想学习和掌握它的结构就比较困难了。类库应用框架则采用了它自己的应用结构，并且这种结构在许多软件环境及项目中都得到了证实。因此，读者可尽管放心地去使用类库来进行 Windows 程序设计。

用不着担心类库的这种结构会降低程序的灵活性。借助于类库，人们可以做任何 Windows SDK 程序能做的工作。这也就意味着人们可以最大限度地使用 Windows。

应用框架所产生的应用代码短而运行速度快。函数套函数，类库应用程序几乎和 Windows SDK 程序一样短。类库"Hello，world!"例程序将全部类库函数静态连接起来以后才只有 80KB，使用了类库动态连接库之后该程序仅为 23KB。至于运行速度，在有些情况下类库应用实际上要比相应的 Windows SDK 应用还要快一些。

类库应用框架的功能非常丰富。Microsoft C/C++ 7.0 所提供的类库 1.0 版奠定了 C++的 Windows 程序设计接口的基础，虽然类库 2.0 版新增了一些新的特性，然而却保持了以下特性：

- 一般类(非面向 Windows)，其中包括：
  - 有关列表、数组及映射的集合类
  - 用途广泛并且功能强大的字符串类
  - 时间、时间间隔及日期类
  - 独立于操作系统的文件访问类
  - 支持对象有组织地在磁盘中进行传递
  - "常规根对象"类层次结构
  - 支持精简了的多文档接口(MDI)应用
  - 有效地支持 OLE(对象连接与嵌入)

类库版本 2.0 对 1.0 版进行了进一步的扩充，能够支持更多的在现有的基于 Windows 应用中能够见到的用户接口特性。应用框架除了采用了一种新的程序结构，它还增加了一些新的特性，这可总结如下：

- 全面支持 File Open、Save 和 Save As 菜单项，并且采用了最近才使用的文件列表形式。
- 打印预显和支持各种打印机
- 滚动窗口(scrolling window)和切分窗口(splitter window)
- 工具条(toolbar)和状态条(status bar)
- 可以处理 Microsoft Visual Basic 控制
- 上下文相关帮助(context-sentive help)
- 自动处理进入对话框中的数据
- 简捷的 OLE 接口
- 支持 DLL

读者会见到反映所有这些特性的例子，为了欣赏和体会类库新增的这些特性，还是让我们首先来看看"打印预显和支持各种打印机"这一特性。为了支持各种打印机，Charles Petzold 在《Programming Windows 3.1》一书中共用了 60 页的篇幅来阐述这一问题，其它关于 Windows 的书干脆对此问题避而不谈。类库在内部共用了好几千行的代码来使得

它能够自动支持打印预显和各种打印机，仅此特性就足以说明我们为什么要使用类库。

Visual C++所提供的一些工具极大地减小了编程时的繁杂程度，App Studio、AppWizard和ClassWizard又极大地缩短了编程所花费的时间。例如，App Studio可以被用来创建包含#define常量在内的头文件，AppWizard可以被用来产生整个应用的代码框架，而ClassWizard则可以被用来产生控制各种消息的函数原型及函数体。

2.1.1 学习曲线

以上所讲述的所有这些好处听起来非常不错，对吗？读者或许在想“一定不会白学”，是的，确实是这样。为了能用效地使用应用框架，读者必须从头到尾仔细地进行学习，当然这需要时间。如果读者不得不连C++、Windows和类库一块学，那么至少需要学习半年的时间，然后才能动手进行实际程序设计。有趣的是，这和单独学习Windows的SDK所用的时间接近。

为什么学习类库只需要这么少的时间呢？原因之一就在于人们可以避免Windows SDK程序员必须学习和掌握的许多程序设计的细节问题。依我看，只要理解和掌握了面向对象程序设计，那么面向对象应用框架就会使得对Windows程序设计的学习和掌握变得相对容易一些。

类库并不能使真正的Windows程序设计大众化。Windows程序员通常比其它程序员的薪水要高，这种状况仍然会延续下去。类库的易学性再加上应用框架所具有的强大功能，决定了社会对类库程序员的需求将有增无减。

2.2 什么是应用框架？

应用框架的定义之一就是“为了生成一般的应用所必须的各种软组件的集成集合”。这么定义实际上并没有多大用处，对吗？如果读者真的想了解什么是应用框架，那么您就得把这本书读完，您很快将会在本章中看到一个应用框架例子，这将是学习应用框架的一个很好的开端。

2.2.1 应用框架和类库

C++之所以是一种倍受欢迎的语言，原因之一就在于它可以通过类库来进行扩展。有一些类库是随编译器一起提供的，还有一些是由其它软件公司销售的，还有一些则是由用户自己开发的。类库是一个可以在应用中使用的相互关联的C++类的集合。例如，矩阵类库可以用来完成一般的涉及矩阵的数学运算，通讯类库可以用来完成串联方式的数据交换。有些情况下人们需要创建一些所提供类的对象，而在有些情况下人们又可能需要派生出自己的类，所有这些都完全依赖于特定类库的设计。

应用框架是一种类库的超集。一般的类库只是一种可以用来嵌入任何程序中的孤立的类的集合，但应用框架却定义了程序的结构。这听起来似乎反映了两者之间的差别，事实也的确如此。大多数Windows类库，包括Microsoft基本类库版本1.0，Borland OWL，以及Microsoft基本类库版本2.0，都被认为是一种应用框架，然而Microsoft基本类库版

本 2.0 却比其它类库提供了更多的重要特性。

2.2.2 应用框架例子

前面内容全是泛泛而谈，现在来看一些实际的代码——不是伪码，而是真正的可以编译并运行的使用了类库的程序代码。能猜到吗？这个例子是一个非常好的老例子“Hello, world!”，只不过又补充了一些新的东西(如果读者使用过类库版本 1.0 的话，那么相信您一定不会对此例感到陌生，当然，主窗口基类除外)。该例子是一个具有最短代码同时又可以运行的类库应用实例。最好将它同相应的 Windows SDK 应用作一下比较！读者现在还不需要弄懂每一个程序行，不过，最好别怕麻烦，把它敲进去实际试试。等到了下一章，读者就可以使用真正的应用框架了。

说明：按照惯例，类库类名用大写字母“C”打头。

下面给出的是 MYAPP 应用的头文件和源代码文件。其中包含了两个类，CMyApp 和 CMyFrame，它们都是从类库基类中派生出来的。首先让我们来看看 MYAPP 应用的 MYAPP.H 头文件：

```
// application class
class CMyApp : public CWinApp
{
public:
    virtual BOOL InitInstance();
};

// frame window class
class CMyFrame : public CFrameWnd
{
public:
    CMyFrame();
protected:
    // 'afx_msg' indicates that the next two functions are part of the
    //  class library message dispatch system
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    afx_msg void OnPaint();
    DECLARE_MESSAGE_MAP()
};
```

现在让我们再来看看 MYAPP 应用的 MYAPP.CPP 源文件：

```
#include <afxwin.h> // class library header file declares base classes
#include "myapp.h"

CMyApp NEAR theApp; // The one and only CMyApp object

BOOL CMyApp::InitInstance()
{
    m_pMainWnd = new CMyFrame();
    m_pMainWnd->ShowWindow(m_nCmdShow);
    m_pMainWnd->UpdateWindow();
    return TRUE;
```

```
}

BEGIN_MESSAGE_MAP(CMyFrame, CFrameWnd)
    ON_WM_LBUTTONDOWN()
    ON_WM_PAINT()
END_MESSAGE_MAP()

CMyFrame::CMyFrame()
{
    Create("AfxFrameOrView", "MYAPP Application");
}

void CMyFrame::OnLButtonDown(UINT nFlags, CPoint point)
{
    TRACE("Entering CMyFrame::OnLButtonDown - %lx, %d, %d\n",
          (long) nFlags, point.x, point.y);
}

void CMyFrame::OnPaint()
{
    CPaintDC dc(this);
    dc.TextOut(0, 0, "Hello, world!");
}
```

下面我们来介绍一下程序中的一些元素：

WinMain 函数 记住 Windows 总是要求每个应用都要有 WinMain 函数，读者之所以在此见不到此函数，是因为它被隐藏在应用框架内部了。

CMyApp 类 类 CMyApp 的对象就代表了一个应用。该程序定义了一个单独的全程 CMyApp 对象 the App，CWinApp 基类决定了 the App 的大部分行为。

应用的开始 当人们运行该应用时，Windows 会自动调用应用框架内部的 WinMain 函数，WinMain 函数会去查找该应用的全程构造对象，该对象是由 CWinApp 所派生出的类的对象。读者不要忘了，在 C++中全程对象在主程序被运行之前就已经被构造好了。

CMyApp 的 InitInstance 成员函数 当 WinMain 发现了应用对象时，它会自动调用 InitInstance 成员函数，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。在派生出的应用类中人们必须重新设计 InitInstance 函数，因为 CWinApp 基类根本无法知道具体应用需要什么样的主框架窗口。

CWinApp 的 Run 成员函数 Run 函数被隐藏在基类中，它被用来负责传递应用的消息，从而维护着应用的运转。WinMain 在调用 InitInstance 之后紧接着调用 Run。

CMyFrame 类 类 CMyFrame 对象代表着应用的主框架窗口。当构造函数调用基类 CFrameWnd 的 Create 成员函数时，Windows 将创建具体的窗口结构，应用框架会将所创建的窗口结构连接到 C++对象中，为了显示所创建的窗口，必须调用基类中的 ShowWindow 和 UpdateWindow 成员函数。

CMyFrame 的 OnLButtonDown 函数 在这里我们可以提前看到类库的消息处理机制。我们已经将鼠标的左按钮被按下这一事件映射到了 CMyFrame 的一个成员函数，读者将会在第四章中学习到有关类库消息映射的细节。以后读者就会明白，当用户按下鼠标的左按钮时，OnLButtonDown 函数就会自动被调用。该函数将会激活类库 TRACE 宏，