信息科学与技术丛书

程序设计系列

Visual C++

通信编程工程实例精解

胡志坤 主编

秦 业 鄢 锋 等编著

- 精解各通信实例的开发过程
- 涵盖所有通信编程的核心代码
- 串口通信编程
- Socket网络通信
- 远程数据库访问
- OPC通信技术
- 有线Modem与GPRS Modem的应用编程

附赠光盘

机械工业出版社
CHINA MACHINE PRESS

信息科学与技术丛书

程序设计系列

Visual C++通信编程工程实例精解

胡志坤　主编

秦　业　鄢　锋　等编著

机械工业出版社

本书介绍利用 Visual C++进行通信程序开发。书中精选了大量来自工程实践的应用实例，涵盖了串口通信、Socket 网络通信、远程数据库访问、应用于工业上的 OPC 通信、Modem 通信以及 SMS 和 GPRS 移动通信编程。

书中的每个应用实例都是在简单介绍必备的背景知识后，重点剖析了应用实例的源代码，并对源代码进行总结、延伸和扩展，以便让读者能举一反三，进行快速的二次开发和工程应用。本书第 1 章是进行 Visual C++高级编程的基础，其余章节的内容均具有一定的独立性，读者可以选择感兴趣的部分来阅读。书中 11 个实例及 2 个扩展实例，均为作者多年的工程实践，所有实例的源代码均在本书的配套光盘中提供。配套光盘中还附有作者多年来收集的大量实用源代码和技术资料。

本书可作为具有一定 Visual C++基础的读者进行通信程序开发的参考书，也可以作为科研单位、高校相关专业人员的参考书籍。无论是对 C++的初学者，还是 Visual C++的高级设计人员，本书均具有很高的参考价值。

图书在版编目（CIP）数据

Visual C++通信编程工程实例精解 / 胡志坤主编. —北京：机械工业出版社，2007.1
（信息科学与技术丛书. 程序设计系列）
ISBN 7-111-20659-2

Ⅰ. V… Ⅱ. 胡… Ⅲ. C 语言—程序设计 Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2006）第 162620 号

机械工业出版社（北京市百万庄大街 22 号 邮政编码 100037）
策　　划：胡毓坚
责任编辑：李利健
责任印制：杨　曦

高等教育出版社印刷厂印刷

2007 年 1 月第 1 版 · 第 1 次印刷
184mm × 260mm · 17.25 印张 · 424 千字
0001 - 5000 册
定价：31.00 元（含 1CD）

出 版 说 明

随着信息科学与技术的迅速发展，人类每时每刻都会面对层出不穷的新技术、新概念。毫无疑问，在节奏越来越快的工作和生活中，人们需要通过阅读和学习大量信息丰富、具备实践指导意义的图书，来获取新知识和新技能，从而不断提高自身素质，紧跟信息化时代发展的步伐。

众所周知，在计算机硬件方面，高性价比的解决方案和新型技术的应用一直备受青睐；在软件技术方面，随着计算机软件的规模和复杂性与日俱增，软件技术受到不断挑战，人们一直在为寻求更先进的软件技术而奋斗不止。目前，计算机在社会生活中日益普及，随着因特网延伸到人类世界的层层面面，掌握计算机网络技术和理论已成为大众的文化需求。由于信息科学与技术在电工、电子、通信、工业控制、智能建筑、工业产品设计与制造等专业领域中已经得到充分、广泛的应用，所以这些专业领域中的研究人员和工程技术人员越来越迫切需要汲取自身领域信息化所带来的新理念和新方法。

针对人们对了解和掌握新知识、新技能的热切期待，以及由此促成的人们对语言简洁、内容充实、融合实践经验的图书迫切需要的现状，机械工业出版社适时推出了“信息科学与技术丛书”。这套丛书涉及计算机软件、硬件、网络、工程应用等内容，注重理论与实践相结合，内容实用，层次分明，语言流畅，是信息科学与技术领域专业人员不可或缺的图书。

现今，信息科学与技术的发展可谓一日千里，机械工业出版社欢迎从事信息科学与技术方面工作的科研人员、工程技术人员积极参与我们的工作，为推进我国的信息化建设作出贡献。

机械工业出版社

前　言

信息科学与技术的发展，使得我们能获得和共享大量的数据，这些都必须依赖数据的通信。在我们进行工控、电力、交通等项目的开发时，可靠、安全、有效的数据采集与传输往往成为项目研发成败的关键。这个时候我们就急需一些成功的工程应用实例来快速熟悉某个通信编程的知识要点。实际上，Visual C++不仅仅是一个编程工具，而且还是一个 Windows 应用程序的开发环境。这是因为它有一个几乎封装了 Windows 所有编程资源的 MFC 类库，而且能无缝地集成 Windows 的所有通信相关的资源，例如 Socket、串口控件、串口 API 以及 OPC 的接口等。

本书作者及所在的开发团队长期利用 Visual C++从事过程工业控制、电力系统自动化方面的软件开发，积累了大量的 Visual C++开发经验，尤其是在通信编程方面积累了大量的实际工程实例和源代码。为了满足从事工控、电力、交通等领域的非通信专业的开发人员从事通信程序开发的需要，本书精选了 11 个实例以及 2 个扩展实例，几乎囊括了工程应用中所有的与通信相关的编程知识。在讲解这些实例时，我们尽量避免深奥的理论，重点剖析了应用实例的编程知识要点，并对这些实例进行了总结、延伸和扩展，以便让读者能举一反三，进行快速的二次开发和工程应用。

本书的编写小组是由一批年富力强、工作在工程第一线的博士组成的。本书由中南大学的胡志坤担任主编，全书的构思、选题、编写和最后的统稿均由胡志坤、鄢锋和北京航空航天大学的秦业共同完成。孔玲爽、彭勃、张定华参加了本书部分章节的编写及相应程序的调试，文杰、张作良、范俊杰、徐飞、林勇、郭爽、徐赞、徐丹也参加了部分编写工作，在此一并对他们表示感谢。

本书可以作为高等学校自动化专业、电气工程专业和计算机专业的辅导教材，也可以作为从事工控、电力系统自动化、交通运输等行业的专业人员从事通信程序开发的参考书。

本书的读者及本书相关领域的专家、高手如果想同作者联系，可发送电子邮件至 jsjfw@mail.machineinfo.gov.cn，读者可以通过电子邮件向作者索要最新的源代码、书中源代码的详细注解，以及其他内容。同时，也期待广大读者通过电子邮件与作者交流自己在通信编程中的心得及相关源代码。

由于作者的水平有限，书中错误及不妥之处，请广大同行和读者批评指正。

编　者

目　录

第1章 概 述

Visual C++是在 Microsoft C 的基础上发展起来的，拥有功能强大的基础类库 MFC，已成为集编辑、编译、运行、调试为一体的功能强大的集成编程环境。本章首先概述了 Windows 的编程基础，然后重点讲解了 Visual C++ 6.0 的集成编程环境，最后简述了 Visual C++编程的基础类库 MFC。本章可以帮助初学者迅速熟悉编程环境，有一定编程基础的开发人员也可以通过本章简单回顾一下相关的知识。

1.1 Windows 应用程序编程基础

1.1.1 Windows 应用程序的特点

Windows 是一种基于图形界面的多任务操作系统，要求以一种全新的思维方式进行程序设计，主要表现为以下几点。

1．事件驱动的程序设计

事件驱动程序设计是一种全新的程序设计方法，它不是由事件的顺序来控制，而是由事件的发生来控制。而这种事件的发生是随机的、不确定的，并没有预定的顺序，这样就允许程序的用户使用各种合理的顺序来安排程序的流程。对于需要用户交互的应用程序来说，事件驱动的程序设计有着过程驱动方法无法替代的优点。首先，它是一种面向用户的程序设计方法，在程序设计过程中除了完成所需功能之外，更多的考虑了用户可能的各种输入，并针对性地设计相应的处理程序；其次，它是一种“被动”式程序设计方法，程序开始运行时，处于等待用户输入事件状态，然后取得事件并做出相应反应，处理完毕又返回并处于等待事件状态。

2．消息循环

消息是一种报告有关事件发生的通知。事件驱动围绕着消息的产生与处理展开，一条消息是关于某一个已经发生了的事件的通知。事件驱动是靠消息循环机制来实现的。

Windows 应用程序的消息来源有以下 4 种。

1）输入消息。输入消息包括键盘和鼠标的输入，该类消息首先放在系统消息队列中，然后由 Windows 将它们送入应用程序消息队列中，由应用程序来处理消息。

2）控制消息。控制消息用来与 Windows 的控制对象，如列表框、按钮、检查框等进行双向通信。当用户在列表框中改动当前选择或改变了检查框的状态时发出此类消息。这类消息一般不经过应用程序消息队列，而是直接发送到控制对象上去。

3）系统消息。系统消息对程序化的事件或系统时钟中断做出反应。一些系统消息，像 DDE 消息（动态数据交换消息）要通过 Windows 的系统消息队列，而有的则不通过系统消息队列，直接送入应用程序的消息队列，如创建窗口消息。

4）用户消息。用户消息是程序员自己定义并在应用程序中主动发出的，一般由应用程

序的某一部分内部处理。

在 Windows 下，由于允许多个任务同时运行，应用程序的输入输出是由 Windows 来统一管理的。Windows 操作系统包括 3 个内核基本元件：GDI、KERNEL、USER。其中，GDI（图形设备接口）负责在屏幕上绘制图形、打印输出，绘制用户界面包括窗口、菜单、对话框等；系统内核 KERNEL 支持与操作系统密切相关的功能，如进程加载、文本切换、文件 I/O、内存管理、线程管理；USER 为所有的用户界面对象提供支持，它用于接收和管理所有输入消息、系统消息并把它们发给相应的窗口的消息队列。消息队列是一个系统定义的内存块，用于临时存储消息，或是把消息直接发给窗口过程。每个窗口维护自己的消息队列，并从中取出消息，利用窗口函数进行处理。

3. 图形输出

丰富的图形输出也是 Windows 操作系统的一个特色，它包括如下 3 点含义：

1）Windows 的每一个应用程序对屏幕的一部分进行处理。Windows 是一个多窗口的操作系统，由操作系统来统一管理屏幕输出；每个窗口要输出内容时，必须首先向操作系统发出请求（GDI 请求），由操作系统完成实际的屏幕输出工作。

2）Windows 程序的所有输出都是图形。Windows 提供了丰富的图形函数用于图形输出，这对输出图形是相当方便的。但是，由于字符也被作为图形来处理，输出时的定位要比 DOS 复杂得多。因为 Windows 输出是基于图形的，它输出文本时不会像 DOS 那样自动换行，而必须以像素为单位精确定位每一行的输出位置。另外，由于 Windows 提供了丰富的字体，所以在计算坐标偏移量时还必须知道当前所用字体的高度和宽度。

3）Windows 下的输出是设备无关的。Windows 下的应用程序使用图形设备接口（GDI）来进行图形输出。GDI 屏蔽了不同设备的差异，提供了设备无关的图形输出能力。Windows 应用程序只要发出设备无关的 GDI 请求，如调用 Rectangle 画一个矩形，均由 GDI 去完成实际的图形输出操作。

4. 用户界面对象

Windows 支持丰富的用户接口对象，包括窗口、边框、系统菜单框、标题栏、菜单栏、工具条、客户区、滚动条、状态栏、图标、光标、插入符、对话框、控件和消息框等。程序员只需用简单的几十行代码，就可以设计出一个非常漂亮的图形用户界面。以下简单介绍几种典型的用户界面对象。

1）窗口。窗口是用户界面中最重要的部分，是屏幕上与一个应用程序相对应的矩形区域，也是用户与产生该窗口的应用程序之间的可视界面。每当用户开始运行一个应用程序时，应用程序就创建并显示一个窗口；当用户操作窗口中的对象时，程序会做出相应反应。用户通过关闭一个窗口来终止一个程序的运行，通过选择相应的应用程序窗口来选择相应的应用程序。

2）边框。绝大多数窗口都有一个边框，用于指示窗口的边界，同时也用来指明该窗口是否为活动窗口。当窗口活动时，边框的标题栏部分呈高亮显示，用户可以用鼠标拖动边框来调整窗口的大小。

3）系统菜单框。系统菜单框位于窗口左上角，以当前窗口的图标方式显示，用鼠标单击该图标（或按〈ALT+空格〉键）就弹出系统菜单。系统菜单提供标准的应用程序选项，包括 Restore（还原窗口原有的大小）、Move（使窗口可以通过键盘上的光标键来移动其位置）、

Size（使用光标键调整窗口大小）、Minimize（将窗口缩成图标）、Maximize（窗口最大化）和 Close（关闭窗口）。

4）标题栏。标题栏位于窗口的顶部，其中显示的文本信息用于标注应用程序，一般是应用程序的名字，以便让用户了解哪个应用程序正在运行。标题栏的颜色反映该窗口是否是一个活动窗口，当为活动窗口时，标题栏呈现醒目颜色。鼠标双击标题栏可以使窗口在正常大小和最大化状态之间切换。在标题栏上按住鼠标左键可以拖动并移动该窗口，单击右键可弹出窗口系统菜单。

5. 资源共享

Windows 是一个多任务的操作系统，各个应用程序共享系统提供的资源。常见的资源包括：设备上下文、画刷、画笔、字体、对话框控制、对话框、图标、定时器、插入符号、通信端口、电话线等。

Windows 要求应用程序必须以一种能允许它共享 Windows 资源的方式进行设计，它的基本模式是这样的：首先向 Windows 系统请求资源，然后使用该资源，需要注意的是，使用完后应释放该资源给 Windows 以供别的程序使用。即使最有经验的 Windows 程序员也常常会忽略释放资源，如果忽略了这一步，当时可能不会出错，但过随即将出现程序运行异常，或其他程序的正常运行受到干扰等情况。这是因为资源没有释放，导致内存泄漏。

在 Windows 应用程序设计中，CPU 也是一种非常重要的资源，因此，应用程序应当避免长时间地占用 CPU 资源（如一个特别长的循环），如果确实需要这样做，也应当采取一些措施，使程序能够响应用户的输入以退出循环。主存也是一个共享资源，要防止同时运行的多个应用程序因协调不好而耗尽内存资源。应用程序一般不要直接访问内存或其他硬件设备，如键盘、鼠标、计数器、屏幕、串口、并口等。Windows 系统要求绝对控制这些资源，以保证向所有的应用程序提供公平的、不中断的运行。如果确实要访问串并口，应当使用通过 Windows 提供的函数来安全地访问。

1.1.2 Windows 的数据类型

1. 句柄

Windows 应用程序中存在许多对象，例如菜单、窗口、图标、内存对象、位图、刷子、设备对象和程序实例等。在 Windows 中，对象使用句柄进行标识。通过使用一个句柄，应用程序可以访问一个对象。

在 Windows 软件开发工具中，句柄被定义为一种新的数据类型。在应用程序中，对句柄的操作一般有如下 3 类：

1）赋值。句柄可以被赋以初始值，可以被改变为用于标识同类对象中的另一个对象，也可以被用作函数的参数。

2）与 NULL 进行相等比较。用来判定一个句柄是否为一个有效的句柄。

3）和与标识同类对象的另一个句柄进行相等比较。用来判定两个句柄是否标识同一个对象。

句柄并不一定是一个 16 位的整数，它的长度将会随着不同的计算机平台和 Windows 的发展而有所变化，如在 32 位 Windows 中，句柄将是一个 32 位的数据，并且不是整数类型。

一种通用句柄类型为 HANDLE，从 HANDLE 类型又派生出了一些新的句柄数据类型，

每种类型的句柄用于标识一种类型的对象。表 1-1 是一些常见的句柄类型。

表 1-1　常见的句柄类型

类　型	说　明	类　型	说　明
HANDLE	通用句柄类型	HBRUSH	标识一个刷子对象
HWND	标识一个窗口对象	HPEN	标识一个笔对象
HDC	标识一个设备对象	HFONT	标识一个字体对象
HMENU	标识一个选单对象	HINSTANCE	标识一个应用程序模块的一个实例
HICON	标识一个图标对象	HLOCAL	标识一个局部内存对象
HCURSOR	标识一个光标对象	HGLOBAL	标识一个全局内存对象

2. 数据类型

为便于开发 Windows 应用程序，Windows 的开发者新定义了一些数据类型。这些数据类型或是与 C/C++中已有的数据类型同义，或是一些新的结构数据类型。引入这些类型的主要目的是为了便于程序员开发 Windows 应用程序，同时也是为了增强程序的可读性；另一个目的是便于程序将来能被移植到其他种类的计算机平台上，或适应 Windows 将来的新版本的变化。大部分的数据类型在 Windows.h 中定义，下面是在这个文件中定义的部分类型：

```
#define PASCAL pascal
#define NEAR near
#define FAR far
typedef unsigned char BYTE
typedef unsigned short WORD
typedef unsigned long DWORD
typedef long LONG
typedef char *PSTR
typedef char NEAR *NPSTR
typedef char FAR *LPSTR
typedef void VOID
typedefint *LPINT
typedef LONG (PASCAL FAR * FARPROC)();
```

在 Windows.h 中，还使用 typedef 定义了一些新的结构类型。这些结构类型的名字也使用大写形式的标识符，如表 1-2 所示。

表 1-2　Windows 新的数据结构

类　型	说　明
MSG	消息结构
WNDCLASS	窗口的类的结构
PAINTSTRUCT	绘图结构
POINT	点的坐标的结构
RECT	矩形结构

这里以类型 MSG 为例来说明类型的定义方法。类型 MSG 是一个消息结构，它的定义方式及其各域的含义如下：

```
typedef struct tagMSG {
    HWND hWnd;          //窗口对象的标识符，该条消息传递到它所标识的窗口上
    UINT   message;     //消息标识符，标识某个特定的消息
    WPARAM wParam;      //随同消息传递的 16 位参数
    LPARAM lParam;      // 随同消息传递的 32 位参数
    DWORD time;         //消息产生的时间
    POINT pt;           //产生消息时光标在屏幕上的坐标
} MSG;
typedef MSG FAR *LPMSG;
```

其中的 POINT 类型的定义如下：

```
typedef struct tagPOINT {
int x;      /* X 坐标 */
int y;      /* Y 坐标 */
} POINT;
typedef POINT FAR *LPPOINT;
```

Windows.h 在定义大部分类型的同时，还定义了该类型的指针类型。例如，上例中的 LPPOINT 和 LPMSG 等，其中字母前缀 LP 表示远指针类型；若使用 NP 作为一个类型的前缀，则表示近指针类型；若使用 P 作为一个类型的前缀时，则表示一般的指针类型，这时由编译程序时所使用的内存模块决定这种指针是远指针或是近指针。在 Windows.h 中说明的大部分指针类型都采用这里介绍的方法进行说明。例如，LPRECT 表示一个 RECT 类型的远指针。

在 Windows.h 中说明的大部分指针类型使用了 C/C++的关键字 const。如果一个指针类型的名字前缀为 LPC、NPC 或 PC，则其中的字母 C 表示这种类型的指针变量所指向的变量不能通过该指针变量来修改，这种指针类型一般采用下述方法进行说明：

```
typedef const POINT FAR * LPCPOINT;
typedef const RECFAR * LPCRECT;
```

一个使用 const 修饰的指针（称其为 const 指针）可以指向没有使用 const 修饰的变量，但没有使用 const 修饰的指针不能指向 const 修饰的变量，例如：

```
const   POINT   pt;
LPCPOINT   lpPoint = &pt;       // 正确
LPPOINT    lpPoint = &pt;       // 错误
```

在变量名的表示方法方面，Windows 推荐使用一种称为“匈牙利表示法”的方法。每个变量名用小写字母或描述了变量的数据类型的字母作为前缀，变量的名字紧跟其后，且用大写字母开始的单词（一个或多个单词）表示其含义，这样每个变量都能附加上其数据类型的助记符。例如：

```
WORD wOffset ;          /* w 表示 WORD 类型 */
DWORD dwValue ;         /* dw 表示 DWORD 类型*/
```

Windows 中常使用的一些字母前缀和它们所代表的数据类型如表 1-3 所示。

表 1-3 Windows 中常使用的一些字母前缀和它们代表的数据类型

类 型	说 明	类 型	说 明
B	BOOL，布尔类型	n	短整型
by	BYTE 类型	np	近（短）指针（near pointer）
c	char 类型	p	指针
dw	DWORD 类型	s	字符串
fn	函数类型	sz	以'\0'结尾的字符串
i	整型	w	WORD 类型
l	LONG 类型	x	short，用于表示 X 坐标时
lp	远（长）指针（long pointer）	y	short，用于表示 Y 坐标时

Windows 程序员也可以根据上述思想和使用目的，发明一些其他的前缀。但要注意，对这些前缀的使用必须保持前后一致。在 Windows 中，所有的函数均根据其用途来命名，它们一般由 2 到 3 个英文单词组成，每个单词的第一个字母大写，例如，函数 CreateWindow()，由该函数的名字可以知道它的用途是创建一个窗口。

3. Windows 的常量

在 Windows.h 中，大多数语句是用于定义一个常量，例如：

```
#define WM_QUIT 0X0012
```

该语句用标识符 WM_QUIT 来表示编号为 0X0012 的消息。每个常量由一个前缀和表示其含义的单词组成的标识符组成，两者之间用下划线隔开。前缀表明这些常量所属的一般范畴。表 1-4 说明了一些前缀和它们所属的范畴的说明。

表 1-4 前缀表示常量所属的范畴

类 型	说 明
CS	窗口类的风格（Class Style）
IDI	预定义的图标对象的标识符（IDentity of Icon）
IDC	预定义的光标对象的标识符（IDentity of Cursor）
WS	窗口的风格（Windows Style）
CW	创建窗口（Create Windows）
WM	窗口消息（Windows Message）
DT	绘制文本（Drawing Text）

1.1.3 Windows 的函数、消息和窗口

1. 事件和消息

在 Windows 中，用户或系统中所发生的任何活动均被当作事件来处理。例如，用户单

击鼠标按钮，就产生了一个鼠标事件。对于所发生的每一个事件，Windows 将其转换成消息的形式放在一个称为消息队列的内存区中，然后由 Windows 的消息发送程序选择适合的对象，将消息队列中的消息发送到欲接收消息的对象上。

Windows 应用程序通过执行一段称为消息循环的代码来轮询应用程序的消息队列，从中检索出该程序要处理的消息，并立即将检索到的消息发送到有关的对象上。典型的 Windows 应用程序的消息循环形式为：

```
MSGmsg;
while (GetMessage(&msg, NULL, 0, 0L))
{
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
```

函数 GetMessage 从应用程序队列中检索出一条消息，并将它存于具有 MSG 类型的一个变量中，然后交由函数 TranslateMessage 对该消息进行翻译，然后由函数 DispatchMessage 将消息发送到适当的对象上。

2. 窗口

对 Windows 用户和程序员而言，窗口对象（简称窗口）是一类非常重要的对象。尤其对程序员，窗口的定义和创建以及对窗口的处理过程最能直观地反映出 Windows 中面向对象的程序设计的四个基本机制（类、对象、方法、消息）。

在 Windows 中，窗口类是在类型为 WNDCLASS 的结构变量中定义的。在 Windows.h 中，结构类型 WNDCLASS 的说明为：

```
typedef struct tagWNDCLASS{
    DWORD style;                    /* 窗口风格 */
    WNDPROC *lpfnWndProc;           /* 窗口函数 */
    int cbClsExtra;                 /* 类变量占用的存储空间 */
    int cbWndExtra;                 /* 实例变量占用的存储空间 */
    HINSTANCE hinstance;            /* 定义该类的应用程序实例的句柄 */
    HICON hIcon;                    /* 图标对象的句柄 */
    HCURSOR hCursor;                /* 光标对象的句柄 */
    HBRUSH hbrBackground;           /* 用于擦除用户区的刷子对象的句柄 */
    LPCSTR lpszMenuName;            /* 标识选单对象的字符串 */
    LPCSTR lpszClassName;           /* 标识该类的名字的字符串 */
} WNDCLASS;
```

WNDCLASS 类型有 10 个域，它描述了该类的窗口对象所具有的公共特征和方法。在程序中可以定义任意多的窗口类，每个类的窗口对象可以具有不同的特征。lpszClassName 是类的名字，在创建窗口对象时用于标识该窗口对象属于哪个类。lpfnWndProc 是指向函数的一个指针，所指向的函数应具有下述的函数原型：

```
LRESULT CALLBACK WndProc(HWND hWnd,UINT message,WPARAM wParam,LPARAM lParam);
```

该函数被称为窗口函数，其中定义了处理发送到该类的窗口对象的消息的方法。窗口函数是一个回调函数，所以在定义窗口函数时要使用 CALLLBACK 类型进行说明。参数 hWnd 是一个窗口对象的句柄，一个窗口函数可以通过该句柄检测出当前正在处理哪个窗口对象的消息。参数 message 是消息标识符。参数 wParam 和 lParam 是随同消息一起传送来的参数，随着消息的不同，这两个参数所表示的含义也不大相同，在定义消息时对这两个参数的含义同时进行定义。

当程序员设置了 WNDCLASS 变量的各个域之后，使用函数 RegisterClass 向 Windows 注册这个类。至此，完成了定义一个窗口类的过程。函数 RegisterClass 的原型为：

```
BOOL RegisterClass(LPWNDCLASS lpWndClass);
```

该函数惟一的一个参数是指向 WNDCLASS 类型的变量的指针。函数返回非零，表示注册成功，否则注册失败。不能向 Windows 注册具有相同名字（lpszClassName 域指向相同的两个字符串）的两个类，否则第二次注册失败并被忽略。

窗口的某些特征（如窗口的颜色等）属于窗口类中定义的，并由该窗口类的所有实例共享。在注册了窗口类之后，程序员使用函数 CreateWindow 创建窗口，得到窗口类的一个实例（一个窗口对象）的句柄。一个窗口可以是一个重叠式窗口，或是一个弹出式窗口，或是一个隶属窗口，或是一个子窗口，这也是在使用 CreateWindow函数时指定的。每一个子窗口都有一个父窗口，每一个隶属窗口都有一个拥有者，这个拥有者是另一个窗口对象，而弹出式窗口是一种特殊的窗口。

一个窗口对象所接受到的消息的响应是由该对象的方法决定的，这些方法被定义在窗口函数中。同一类的所有对象共用同一个窗口函数。窗口函数决定着对象如何用内部方法对消息作出响应，例如，如何在屏幕上画出窗口自身。

一个最简单的窗口函数为：

```
LRESULT CALLBACK WndProc(HWND hwnd, UNIT message, WPARAM wParam, LPARAM lParam)
{
    return DefWindowProc (hwnd, message, wParam, lParam);
}
```

该窗口函数通过调用 Windows 的函数 DefWindowProc（默认窗口函数），让 Windows 的默认窗口函数来处理所有发送到窗口对象上的消息。

当用户操作屏幕上的一个窗口对象时（例如用户改变了屏幕上窗口对象的位置或大小）或发生其他事件时，该事件的消息被存于应用程序的消息队列中，消息循环首先从该队列中检索出该消息，然后将消息发送到某个对象上。发送过程由 Windows 来控制，Windows 根据消息结构中的 hWnd 域所指示的消息发送的目标对象，调用该对象所在类的窗口函数完成消息的发送工作。窗口函数根据消息的种类，选择执行一段代码（方法），对消息进行处理，并通过 return 语句回送一个处理结果或状态。消息循环、Windows 和窗口函数协同配合，完成一条消息的发送和处理。在处理完一条消息之后，如果应用程序队列中还有其他消息，则继续进行上述处理过程。否则，应用程序产生的消息就在消息处理队列中进行等待。

在窗口函数中，使用 switch 语句来判断窗口函数接收到什么消息，通过执行相应的语句

对消息进行处理。当处理完一条消息时，窗口函数要返回一个值，表示消息的处理结果，许多消息返回 0 值，也有些消息要求返回其他的值，这由具体的消息决定。窗口函数不打算处理的消息必须交由 DefWindowProc()进行处理，并且函数必须返回 DefWindowProc()的返回值。

1.1.4 Windows 程序的组织

本节介绍一个完整的用 API 完成的 Windows 程序。一个 Windows 程序必须有一个名为 WinMain 的主函数。

```
#include <windows.h>
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
int PASCAL WinMain(
    HINSTANCE hInstance,              // 应用程序的实例句柄
    HINSTANCE hPrevInstance,          // 该应用程序前一个实例的句柄
    LPSTR lpszCmdLine,                // 命令行参数串
    int nCmdShow)                     // 程序在初始化时如何显示窗口
{
    char szAppName[] = "Window";
    HWND hwnd;
    MSG  msg;
    WNDCLASS wndclass;
    if (!hPrevInstance) {
        // 该实例是程序的第一个实例，注册窗口类
        wndclass.style = CS_VREDRAW | CS_HREDRAW;
        wndclass.lpfnWndProc = WndProc;
        wndclass.cbClsExtra = 0;
        wndclass.cbWndExtra = 0;
        wndclass.hInstance = hInstance;
        wndclass.hIcon = LoadIcon(hInstance, IDI_APPLICATION);
        wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
        wndclass.hbrBackground = (HBRUSH)GetStockObject(BLACK_BRUSH);
        wndclass.lpszMenuName = NULL;
        wndclass.lpszClassName = szAppName;

        if (!RegisterClass(&wndclass))    //如果注册失败
          return FALSE;
    }
    // 对每个实例，创建一个窗口对象
    hwnd = CreateWindow(
        szAppName,
        "Sample Program",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL,
```

```
            NULL,
            hInstance,
            NULL);
        ShowWindow(hwnd, nCmdShow);
        UpdateWindow(hwnd);

        while(GetMessage(&msg, NULL, 0, 0)) {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
        return msg.wParam;
    }
```

WinMain 函数是 Windows 应用程序开始执行时的入口点，它的返回类型为 int。WinMain 函数的作用十分类似于 MS-DOS 中的 C 应用程序的 main 函数。WinMain 带有四个参数。参数 hInstance 和 hPrevInstance 是程序的实例句柄。在 Windows 环境下，可以运行同一个程序的多个实例，每一个实例都是该应用程序的一个句柄，每个实例使用一个实例句柄进行标识。hInstance 是标识当前程序的实例的句柄，它的值不会为 NULL。如果在此之前 Windows 中已经运行了该程序的另一个实例，则这个实例的句柄由参数 hPrevInstace 给出。如果在运行该程序时，Windows 环境中不存在该程序的另一个实例，则 hPrevInstance 为 NULL。

对同一个类，不能向 Windows 注册一次以上。在这个程序中，通过判别 hPrevInstance 的值是否为 NULL，来决定是否应向 Windows 注册窗口类。这样的程序逻辑保证了只在该程序的第一个实例中注册窗口类。

参数 lpszCmdLine 中包含有运行程序时传递给程序的命令行参数。例如，若以这样的命令运行该程序：Sample.exe Programming Windows，则 lpszCmdLine 将指向字符串“Programming Windows”。

最后一个参数 nCmdShow 是一个 int 类型的整数，用以说明在程序被装入内存时，Windows 以何种方式显示这个程序的窗口。根据运行程序的方式不同，该参数被设置为 SW_SHOWNORMAL 或 SW_SHOWMINNOACTIVE，SW 的含义是“Show Window”（显示窗口）。

Windows 的主函数都是首先以初始化（注册类、创建对象等）这一步开始，而且紧跟着就是消息循环运行这一步。这些步骤对所有的 Windows 应用程序都大同小异。Windows 应用程序主要的不同点在窗口函数的定义上，由于一个应用程序所解决的任务不同，它的窗口函数对消息的处理方式也就不相同，因而每个应用程序需要定义不同的窗口函数。

```
    LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
    {
        switch (message)
        {
            case WM_DESTROY:
            PostQuitMessage(0):
                return 0;
```