目录

第 13 章　Win32 窗口程序设计

在 Windows 操作系统下运行的程序有无界面的后台进程，但更多的是有界面的应用程序，称为窗口程序或者窗口应用程序。Win32 窗口程序与前面介绍的控制台应用程序的工作过程有很大差别。本章主要介绍 Win32 窗口程序运行的基本过程、窗口程序的基本结构、以及用汇编语言编写窗口应用程序的基本方法。当然，在开发实际的窗口应用程序时需要读者花比较大的精力去了解 Windows API 函数、各种事件产生的消息等知识。与采用 MFC（Microsoft Foundation Class）等开发的程序相比，基于汇编语言的程序能够更直观地展现 Windows 窗口程序的核心结构和执行过程。掌握窗口程序运行的基本原理有助于使用高级工具开发更复杂的应用程序。

13．1 Win32 窗口程序设计基础

13.1.1 窗口程序运行的基本过程

Windows 窗口程序是 Windows 环境下最常见的一种程序样式。例如 WORD 文档编辑软件、PPT 幻灯片制作软件、Excel 表格制作软件、浏览器软件等等都是窗口程序。

窗口一般是屏幕上的矩形区域，其大小和在屏幕上的显示位置均可调整。窗口通常包含标题栏、菜单、工具栏、状态栏、图标、最小化/最大化/关闭按钮、滚动条等，窗口中间还有用来显示信息和与用户交互的客户区。常见的对话框，如文件打开和保存对话框、字体选择对话框、查找和替换对话框、颜色选择对话框等等，也是一种窗口，它中间可以含有按钮、输入框、单选按钮、复选框、组合框、列表框、静态文本框、滚动条等等子窗口控件。

窗口应用程序运行的最大特点是事件驱动，没有固定的流程。用户点击菜单、工具栏、按钮、按键、移动鼠标、单击鼠标、拖动窗口、改变窗口大小等等操作都是事件，程序根据用户的操作进行响应，也就是调用对应的处理函数，完成程序所规定的功能。

应用程序的运行过程高度依赖 Windows 操作系统，其中很多功能是由操作系统来完成的。图 13.1 给出了一个窗口程序运行的基本过程。

(1) 获取应用程序的句柄(GetModuleHandle)

所谓句柄(Handle)实际上一个双字类型(32 位系统中)的整数值，它是操作系统管理的各种资源的唯一编号。应用程序、窗口、按钮、图标、文件等等都有各自的句柄。用户程序必须先通过 Windows API 来获得指定程序、窗口、按钮、图标、文件等的句柄，然后才能通过该句柄对所代表的对象进行操作，如发送消息、读写文件等。

(2) 注册窗口(RegisterClassEx)

注册窗口就是向操作系统报告窗口的一些信息，由操作系统进行登记管理。其中比较重要的信息包括窗口类名、窗口上图标、菜单、窗口消息处理函数等。

(3) 创建窗口(CreateWindowEx)

创建窗口时的信息包括窗口类名（必须与注册窗口时所使用的窗口类名相同）、父窗口或者

主窗口句柄、窗口的宽度和高度、窗口左上角点的坐标、窗口标题栏的名称、窗口风格等。创建窗口后函数的返回值为窗口句柄。

(4) 显示窗口（ShowWindow）

(5) 刷新窗口客户区（UpdateWindow）

(6) 消息循环

消息循环是一个循环执行的程序片段。该程序段的主要功能是：取消息(GetMessage)、翻译消息(TranslateMessage)、派发消息(DispatchMessage)。

这里先要解决一个问题，即消息放在哪，从何处取消息。

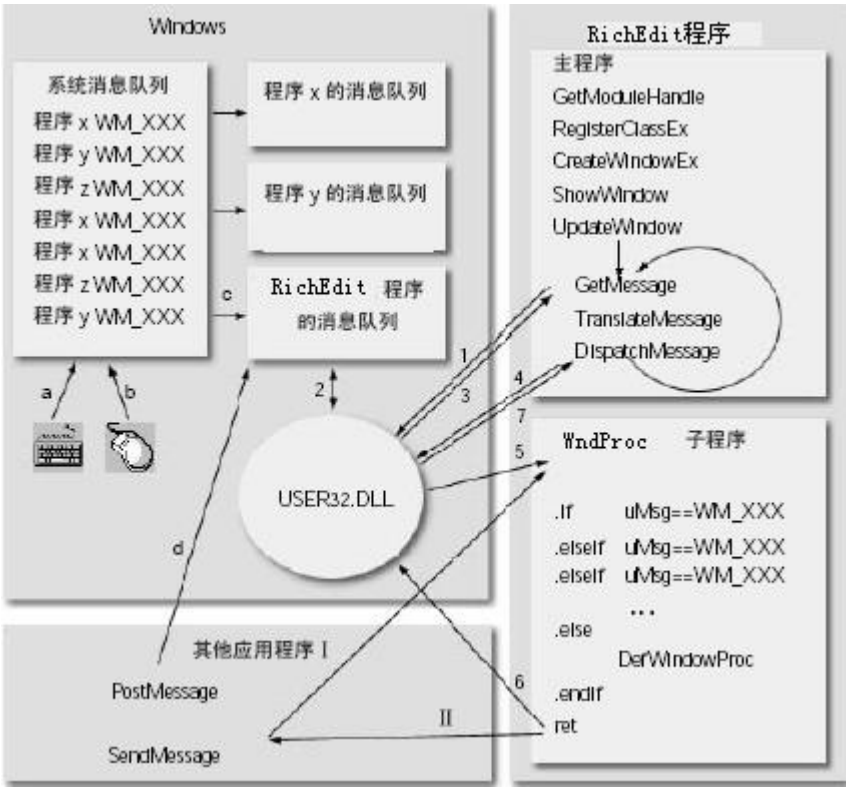

图 13.1 窗口程序的运行过程

如图 13.1 所示，其右半部分是一个窗口应用程序，左上部分是 Windows 操作系统进程，左下部分是其他应用程序。当应用程序在用户面前展现一个界面后，用户按动键盘、移动鼠标、按下或放开鼠标左键或右键等，都会产生相应的记录放在 Windows 系统消息队列里（图 13.1 的 a,b 箭头所示）。换句话说，用户操作键盘和鼠标等产生的消息并没有直接发送给应用程序。这也比较容易理解。Windows 环境下有多个程序在运行，它们共享一个键盘和一个鼠标。那么，按下键盘或鼠标后，产生的消息会发送给哪个应用程序呢？这显然不是应用程序所能决定的事情，而要由操作系统来管理。操作系统作为一个大管家，管理着这些外部设备，同时也管理着当前运行的各个程序。操作接收消息后，先放在系统消息队列中。

Windows 为每个程序（严格地说是每个线程）维护一个消息队列。Windows 操作系统检查系统消息队列里各个消息的发生位置或内容，根据窗口在屏幕上的层叠关系、当前哪个窗口处于激活状态等信息，判断消息所属的应用程序，并把该消息移送到应用程序的消息队列里，如图 13.1 中的箭头 c 所示。当然，其他程序也可以使用 PostMessage 向应用程序发出消息，如图 13.1 中的箭头 d 所示。操作系统不管应用程序是否来取消息，都会将消息送到应用程序消息队列中。

当应用程序中的消息循环执行到 GetMessage 的时候，控制权转移到 GetMessage 所在的USER32.DLL 中（箭头 1）。 USER32.DLL 从程序消息队列中摘下队首一条消息（箭头 2），然后把这条消息返回应用程序（箭头 3）。当队列为空的时候,这个函数会被挂起。如果执行应用程序的某段代码耗时长，则在此期间产生的消息就会积压在应用程序消息队列中。应用程序只能一条条依次取回消息、对消息进行处理，此时程序运行显得很“卡”，跟不上用户操作的步伐。

应用程序可以对取回来的消息进行预处理，例如，可以用 TranslateMessage 把基于键盘扫描码（位置编码）的虚拟按键消息转换成基于 ASCII 码的字符消息；使用 TranslateAccelerator 把键盘快捷键转换成命令消息等。

在对消息进行翻译等预处理后，应用程序将执行该消息对应的响应函数。但是应用程序中不会直接调用窗口过程(即图 13.1 中的 WndProc 消息处理函数)，而是通过 DispatchMessage 间接调用窗口过程。当控制权转移到 USER32.DLL 中的 DispatchMessage 时，DispatchMessage 会找出消息对应窗口的窗口过程（之前在操作系统注册窗口时，已登记了窗口消息处理函数的入口地址），把消息的具体内容当作调用该窗口过程的参数，并调用该窗口过程（箭头 5）。窗口过程根据消息找到对应的需要执行的分支语言片段，窗口过程执行完后返回时（箭头 6），控制权回到 DispatchMessage。最后 DispatchMessage 函数返回应用程序（箭头 7）。这样，一个循环就结束了，程序又开始新一轮的 GetMessage。当取到退出程序的消息(WM_QUIT)后，整个消息循环处理程序结束，进而退出程序，返回操作系统。

这里有一个问题，在取回消息和翻译消息后，为什么不直接调用窗口消息处理程序，而要用 DispatchMessage 来派发消息？原因有几方面。一是一个应用程序可以有多个窗口，取回的消息是属于程序的消息，虽然消息结构中指明了它属于哪个窗口，但是需要根据窗口句柄来调用相应的消息处理函数，若让应用程序来控制，无疑会增加程序的工作量。二是其他的程序可以用 SendMessage 通过 Windows 直接调用某个窗口消息处理函数，图 13.1 中左下部分的 I、II 给出了它的工作过程。第三个原因是 Windows 没有将所有的消息都放在消息队列中，对于实时性要求很强的消息如 WM_SETCURSOR，操作系统立即调用窗口消息处理函数进行处理。

13.1.2 Windows 消息

“消息”是指发给应用程序的各种命令，例如，键盘和鼠标的输入命令、系统或其它软硬件产生的命令等。操作系统将这些不同的命令转换成统一的消息格式，按照命令产生的时间顺序存放到消息队列中。存放消息的结构 MSG 定义如下：

```
MSG STRUCT
  hwnd      DD      ?     ;窗口句柄，指明该消息属于哪个窗口
  message   DD      ?     ;消息号，指明消息的种类
  wParam    DD      ?     ;消息的附加信息一，其含义依赖于消息号
  lParam    DD      ?     ;消息的附加信息二，其含义依赖于消息号
  time      DD      ?     ;指明消息产生的时间
  pt        POINT   <>    ;指明消息产生时，光标相对屏幕坐标的位置
                          ;（POINT 是由 X、Y 坐标值组成的结构）
MSG ENDS
```

注意，“MSG STRUCT …MSG ENDS”是采用汇编语言形式定义的结构体。有关结构体定义的

语法可参见第 10 章的介绍。用 C 语言定义的同名结构如下（参见 WinUser.h）：

```
typedef struct tagMSG {
    HWND        hwnd;
    UINT        message;
    WPARAM      wParam;
    LPARAM      lParam;
    DWORD       time;
    POINT       pt;
#ifdef _MAC
    DWORD       lPrivate;
#endif
} MSG;
```

比较用 C 语言和用汇编语言定义的 MSG 结构，两者本质是相同的，即组成的字段相同、字段顺序相同、各个字段的长度相同。注意，在 WinUser.h 定义的 MSG 结构中，有一个条件编译语句，“#ifdef _MAC DWORD lPrivate; #endif”，在给出编译开关 _MAC 时（在 Mac 操作系统的环境下），会多一个字段，但在 Windows 环境下，编译后不会有该项。

Windows 系统中的消息很多，编写程序时需要了解各种操作会产生一些什么消息。下面介绍一些常用的消息。

1、 键盘消息

涉及到键盘操作的消息包括 WM_KEYDOWN、WM_SYSKEYDOWN、WM_KEYUP、WM_SYSKEYUP。当按了键盘上的任何一个键时，Windows 都会收到一个击键消息。对于那些产生可显示字符的操作，Windows 还会收到字符消息。例如，当按了字符 A 键时，依次会产生 WM_KEYDOWN、WM_CHAR、WM_KEYUP 消息；当按了 Shift 键时，将产生 WM_KEYDOWN 和 WM_KEYUP 消息。字符消息 WM_CHAR 不是由硬件产生的，而是由可产生显示字符的击键消息转换而来的。对于一些特殊键，如 F10 键、含有 Alt 键的击键组合，它们用于快速激活菜单及菜单中的选项、切换当前窗口，会产生 WM_SYSKEYDOWN、WM_SYSKEYUP 消息。注意，WM_KEYDOWN、WM_KEYUP 等都是符号常量，它们对应一个数值编号。例如，WM_KEYDOWN 对应 100H，但是在程序中使用 WM_KEYDOWN 显然比使用 100H 具有更好的可读性。消息编号存放在 MSG 结构的 message 字段。

在按键时，仅有一个消息编号是不够的，还需要知道是被击键的信息，这些信息存放在 MSG 结构的 wParam 和 lParam 中。例如，当按下“←”时，产生的消息号 message=WM_KEYDOWN，wParam 中存放该键的虚拟键码 VK_LEFT（参见 WinUser.h），lParam 中存放指定重复按键次数（如按住该键不放）（双字中的第 0-15 位）、扫描码（16-23 位）、扩展键标识（第 24 位）、保留位（25-28 位）、关联码（第 29 位）、键的先前状态（第 30 位）、转换状态标识位（第 31 位）。

2、 鼠标消息

与鼠标有关的消息包括：鼠标移动（WM_MOUSEMOVE）、单击鼠标左键（WM_LBUTTONDOWN）、松开鼠标左键（WM_LBUTTONUP）、双击鼠标左键（WM_LBUTTONDBLCLK）、单击鼠标右键（WM_RBUTTONDOWN）、松开鼠标右键（WM_RBUTTONUP）、双击鼠标右键（WM_RBUTTONDBLCLK）、单击鼠标中键（WM_MBUTTONDOWN）、松开鼠标中键（WM_MBUTTONUP）、双击鼠标中键（WM_MBUTTONDBLCLK）、鼠标滚轮消息（WM_MOLSEWHEEL）。

在鼠标消息中，用 lParam 表明鼠标光标的位置，其参数值分为高位字与低位字，低位字中

存储鼠标光标的 X 坐标值，高位字存储 Y 坐标值；用 wParam 记录鼠标、Ctrl、Shift 的按键情况以及滚轮的滚动方向，它中间有 5 个二进制位分别来记录 5 个键的按键状态，这 5 位分别对应是 MK_LBUTTON（0x0001）、MK_RBUTTON（0x0002）、MK_SHIFT（0x0004）、MK_CONTROL（0x0008）、MK_MBUTTON（0x0010）。

3、 命令消息

当单击菜单、工具栏按钮、加速键、子窗口按钮的时候，都会产生 WM_COMMAND 消息。

WM_COMMAND 消息中，wParam 的高字为通知码（菜单的通知码为 0，加速键的通知码为 1），wParam 的低字(两字节)为命令 ID。lParam 为发送命令消息的子窗体句柄，对于菜单和加速键来说，lParam 为 0，只有子窗体中的控件被点击时，此项表示子窗口的句柄。命令 ID 是资源脚本中定义的菜单项的命令 ID 或者加速键的命令 ID，相当于它们的身份标识。

单击 Windows 菜单中菜单项和加速键，会产生 WM_SYSCOMMAND 而不是 WM_COMMAND 消息。注意，WINDOWS 菜单是系统菜单，也就是在标题栏单击鼠标左键的时候弹出的菜单。

4、 退出程序运行的消息

一般退出程序时，会按退出程序的菜单项，或者界面上的关闭图标(即窗口右上角的关闭按钮)，此时将产生一个 WM_CLOSE 的消息。WM_CLOSE 消息通常采用缺省的（即 Windows 操作系统自带的）处理方式，在该消息处理中调用 DestroyWindow 函数，销毁窗口（在屏幕上的窗口消失），并发出 WM_DESTROY 消息。对于有多个窗口的程序，一个子窗口的关闭并不代表程序结束，主窗口仍正常显示在屏幕上，可继续接收并处理消息，也即子窗口的处理程序可以不理 WM_DESTROY 消息。但是，对于主窗口而言，编程者应该对 WM_DESTROY 消息进行处理，调用 PostQuitMessage，发出 WM_QUIT 消息。在消息循环中收到 WM_QUIT 消息后，就退出消息循环。如果在主窗口的消息处理程序中不处理 WM_DESTROY 消息，虽然在屏幕上看不到窗口了，但在后台进程依然看得到该程序，程序并未真正的结束。

Windows 的消息非常多，一个操作能够可能引发多个消息。除了从有关资料学习事件引发的消息外，也可以采用捕获窗口收到的消息的方法，来了解各个事件会触发的消息。

13.1.3 Win32 窗口程序的开发环境

前面介绍的控制台程序开发时采用了 Microsoft Visual Studio 2019 开发平台，该平台同样可以用开发 Win32 窗口程序，操作步骤也是类似的。

在 Windows 窗口程序中，要大量使用 Windows 操作系统提供应用程序开发接口函数。为了让编译器能正确的开展编译工作，必须对函数进行说明。例如，前面提到的一些函数的原型说明如下。

```
GetModuleHandle PROTO STDCALL :DWORD
RegisterClassEx PROTO STDCALL :DWORD
CreateWindowEx PROTO STDCALL :DWORD,:DWORD,:DWORD,:DWORD,:DWORD, :DWORD
                        :DWORD,:DWORD,:DWORD,:DWORD,:DWORD, :DWORD
ShowWindow   PROTO STDCALL :DWORD,:DWORD
UpdateWindow PROTO STDCALL :DWORD
GetMessage PROTO STDCALL :DWORD,:DWORD,:DWORD,:DWORD
TranslateMessage PROTO STDCALL :DWORD
```

```
DispatchMessage  PROTO STDCALL :DWORD
SendMessage PROTO STDCALL :DWORD,:DWORD,:DWORD,:DWORD
PostMessage PROTO STDCALL :DWORD,:DWORD,:DWORD,:DWORD
PostQuitMessage PROTO STDCALL :DWORD
ExitProcess PROTO STDCALL :DWORD
```

后面的例子中，还将看到使用更多的 API 函数。另外，在程序中也会使用大量的符号常量，例如在介绍常见的 Windows 消息时出现如下消息种类。

```
WM_KEYDOWN              equ 100h
WM_KEYUP                equ 101h
WM_CHAR                 equ 102h
WM_COMMAND              equ 111h
WM_MOUSEMOVE            equ 200h
WM_LBUTTONDOWN          equ 201h
WM_LBUTTONUP            equ 202h
WM_LBUTTONDBLCLK        equ 203h
```

在程序中使用符号常量而不是后面的数值，优点是非常显著的。人们容易记住有规律的、有一定含义的名字，而记数值就非常困难了。为了使用这些符号常量，在程序中也必须进行说明。

在 C 语言程序设计中，将数据结构的定义、符号常量的定义、函数原型说明放在一些头文件中，用户开发程序时使用#include 包含相应的头文件即可。尽管在汇编语言程序中也要使用这些信息，但在汇编语言程序中不能直接使用这些文件，原因在于编译器不支持 C 语言的语法格式。

感谢Steve Hutchesson为汇编程序员所做的工作，提供了一个汇编语言开发软件包MASM32，其中包含了开发需要的规模庞大的头文件、导入库文件等。该软件包可以从 http://www.masm32.com/ index.htm 下载，最新版本为 masm32v11r.zip。安装该软件包后有include 目录，直接使用该文件夹下的有关头文件即可。例如，消息结构 MSG 定义在 windows.inc 中，因而在程序的开头增加“include windows.inc”后，就可以直接使用 MSG 结构了。

在窗口应用程序中，经常会用到各种资源，如菜单(Menu)、对话框(Dialog)、工具栏(Toolbar)、位图（Bitmap）等等。可以使用 Visual Studio 平台提供的资源编辑器来生成程序需要的资源文件。当然，也可以使用文本编辑器来生成资源脚本文件。

13．2 Win32 窗口应用程序的结构

基于窗口的应用程序一般分为四个部分组成：主程序、窗口主程序、窗口消息处理程序以及用户处理程序。这四部分的基本关系为：操作系统首先执行主程序，主程序调用窗口主程序。窗口主程序创建窗口、向操作系统注册窗口、然后不断地从程序的消息队列中取回消息、翻译消息和派发消息。窗口主程序并不直接调用窗口消息处理程序，而是通过操作系统来调用窗口消息处理程序。窗口消息处理程序接收操作系统转发过来的消息，判断收到的消息种类，并调用用户处理程序完成相应的功能。当然，在程序简单的情况下，可以将主程序与窗口主程序合成一个函数，将窗口消息处理程序以及用户处理程序合成一个函数。下面将更详细的介绍各部分的处理流程。

13.2.1 主程序

主程序一般用于完成初始化工作，主要包括：获取本程序在主存中的地址（也称句柄）、获取命令行参数的地址、调用窗口主程序等，如图 13.2 所示。从该图中可以看出，如果“主程序”没有调用“窗口主程序”，则主程序就会在完成了前面几步操作后直接退出，不会创建基于窗口的应用程序。

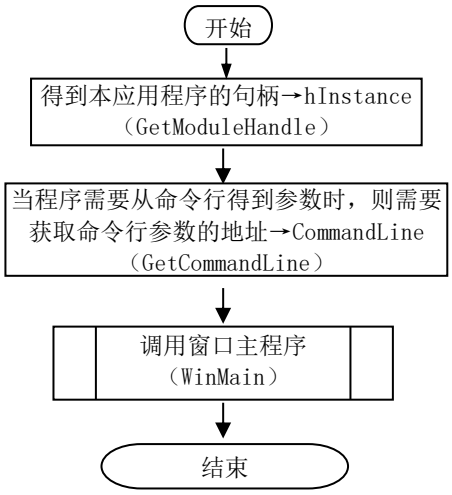

图 13.2　主程序流程图

13.2.2 窗口主程序

窗口主程序是一个函数体，函数的名字和参数并没有特殊的要求，但为了与高级语言一致，一般把名字定为 WinMain。WinMain 首先完成窗口注册、窗口创建、程序所需资源的装载等操作，然后不断地从操作系统中获取消息，并分发到窗口消息处理程序。其流程如图 13.3 所示。

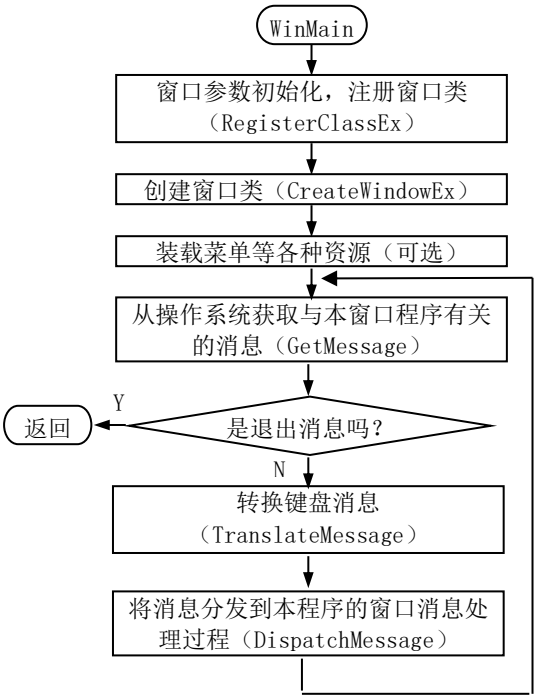

图 13.3　窗口主程序流程图

一般而言，窗口主程序用采用以下的原型说明。

```
WinMain PROTO  hInst: DWORD,          ;应用程序的实例句柄
               hPrevInst: DWORD,      ;前一个实例句柄（参数值恒为 NULL）
               lpCmdLine: DWORD,      ;命令行指针，指向以 0 结束的命令行字符串
               nCmdShow :DWORD        ; 指出如何显示窗口
```

由于窗口主程序是在主程序中调用的，因而参数的个数和顺序可以自定。

13.2.3 窗口消息处理程序

窗口消息处理程序（或称窗口过程，也是一个函数体形式）的主要功能是对接收到的消息进行判断，调用对应的处理函数。其程序结构是一个典型的分支程序（switch…case），每一个支路可以是直接完成某项功能的语句序列，也可以调用 Windows API 函数和“用户处理程序”中的函数。对于未处理的消息，交给操作系统做缺省处理，也即分支程序的最后出口要么是处理了对应消息后直接返回（一般在 0→eax 后返回），要么是在调用缺省处理的 Windows API 函数 DefWindowProc 后直接返回(将 DefWindowProc 的返回值作为此部分的返回值),如图 13.4 所示。

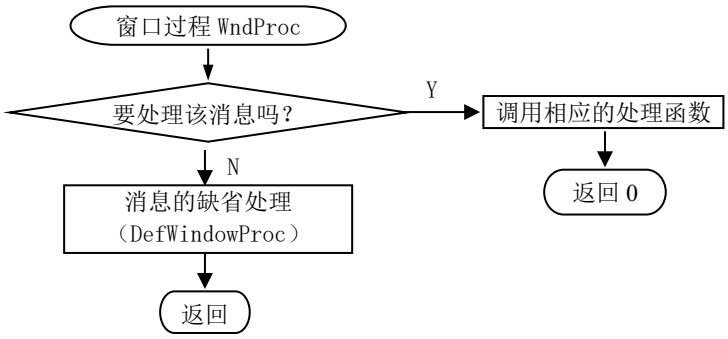

图 13.4 窗口消息处理流程图

窗口消息处理程序的函数名可以自行定义，一般取名为 WndProc。与窗口主程序不同，窗口消息处理程序的原型说明必须满足如下的形式,因为该函数不是应用程序直接调用的，而是由操作系统间接调用的。

```
WndProc PROTO hWin   : DWORD,     ;窗口句柄
              uMsg   : DWORD,     ;消息编号，是分支判断的主要依据
              wParam : DWORD,     ; 消息附加信息一。
                                  ; 若是子消息号，则是嵌套分支判断的依据
              lParam : DWORD      ; 消息的附加信息二
```

在基于窗口的应用程序运行中，会产生各种各样的消息。如果所有的消息都要自己写响应函数，无疑工作量是巨大的。实际上，大部分窗口的行为都差不多，可遵循共同的规范。这样就可以用一种默认的方式来处理消息。Windows API 提供了默认的消息处理函数 DefWindowProc，可处理几百种消息。正是利用了该函数，使得移动窗口、改变窗口大小、将鼠标移到窗口边框（鼠标形状发生变化）、最大化、最小化窗口时，窗口都能作出正确的响应。

当然，如果用户不想使用 Windows 默认的行为，调用自己的编写的消息处理方法即可。

用户处理程序主要是根据程序功能需要、按照模块化的程序设计思想，编写的完成某个消息

响应任务的函数。

13．3 窗口应用程序开发实例

13.3.1 不含资源的窗口程序

【例 13.1】 编写一个窗口应用程序，在屏幕上创建一个窗口，窗口的标题为：“First Window”。当按下一个键时，在窗口中以 16 进制形式显示该键的 ASCII 码。

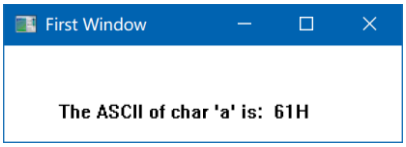

图 13.5 程序运行界面示例

本例主要是为了展现 Windows 窗口程序的基本结构，程序中不含有菜单、对话框、工具栏等资源。在窗口消息处理程序中只对 WM_CHAR 和 WM_DESTROY 进行了处理，其他的消息都采用默认的消息处理方式，调用 DefWindowProc 来完成。在运行程序时，窗口可移动、可改变大小、可最大化、最小化和关闭，点击程序图标，还可以弹出一个菜单，并进行响应。该程序可以同时运行多个实例，即打开多个如图 13.5 所示的窗口。源程序如下。

```
.686P
.model flat,stdcall
OPTION CASEMAP:NONE        ; 大小写敏感，程序中的一些变量名与结构名同名
                           ; 区分大小写。若不区分，可修改变量的名字
WinMain proto :DWORD       ; 简化了窗口主程序，未用的信息未做参数
WndProc proto :DWORD,:DWORD,:DWORD,:DWORD ; 窗口消息处理程序
Convert proto :BYTE,:DWORD  ; 数值（一个 BYTE）转换为 ASCII 串（串的首地址 DWROD）
include windows.inc          ; 头文件说明
include user32.inc
include kernel32.inc
include gdi32.inc
.data
szClassName  db "TryWinClass",0          ; 窗口类名
szTitle      db " First Window",0        ; 标题栏显示的信息
hInstance    dd  0                       ; 应用程序的句柄
szDisplayStr db  "The ASCII of char '?' is:  "  ; 窗口中显示的串
szASCII      db  "00H",20H,20H                   ; 20H 为空格符
dwDisplayLength   dd   $-szDisplayStr            ; 窗口显示串的长度
.code
start:
    invoke GetModuleHandle,NULL        ; 获得并保存本程序的句柄
    mov    hInstance,eax
```

```
    invoke WinMain,hInstance            ; 调用窗口主程序
    invoke ExitProcess,eax              ; 退出本程序，返回 Windows

;-------------- 窗口主程序 WinMain --------------
WinMain proc hInst:DWORD
    LOCAL wc:WNDCLASSEX     ; 创建窗口时需要的信息用结构 WNDCLASSEX 说明
    LOCAL msg:MSG           ; 获取的消息用消息结构 MSG 存放
    LOCAL hwnd:HWND         ; 本窗口的句柄
    invoke RtlZeroMemory,addr wc,SIZEOF wc  ; 将结构变量中的所有内容置 0
    mov   wc.cbSize,SIZEOF WNDCLASSEX       ; WNDCLASSE 结构的大小（字节数）
    mov   wc.style, CS_HREDRAW or CS_VREDRAW
                                ; 本窗口风格(当窗口高度和宽度变化时则重画窗口)
    mov   wc.lpfnWndProc, offset WndProc  ; 本窗口消息处理程序的的入口地址
    push  hInst
    pop   wc.hInstance                    ; 本窗口所属的应用程序句柄
    mov   wc.hbrBackground,COLOR_WINDOW+1   ; 窗口的背景颜色为白色
    mov   wc.lpszMenuName,NULL              ; 窗口上无菜单
    mov   wc.lpszClassName,offset szClassName  ; 窗口的类名
    invoke LoadIcon,NULL,IDI_APPLICATION     ; 装入系统默认的图标
    mov   wc.hIcon,eax                      ; 窗口上的图标
    invoke LoadCursor,NULL,IDC_ARROW        ; 装入系统默认的光标
    mov   wc.hCursor,eax                    ; 窗口上的光标
    invoke RegisterClassEx, addr wc         ; 注册窗口类
    invoke CreateWindowEx,NULL,addr szClassName,addr szTitle,\
           WS_OVERLAPPEDWINDOW,\
           CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,\
           NULL,NULL,hInst,NULL
    mov   hwnd,eax                  ; 创建窗口，获得窗口句柄
    invoke ShowWindow, hwnd,SW_SHOWNORMAL  ; 显示窗口
    invoke UpdateWindow, hwnd
StartLoop:                                  ; 消息循环
    invoke GetMessage,addr msg,NULL,0,0     ; 取消息
    cmp  eax,0
    je   ExitLoop
    invoke TranslateMessage, addr msg     ;  翻译消息
    invoke DispatchMessage, addr msg      ;  分配消息
    jmp StartLoop
ExitLoop:
     mov  eax,msg.wParam
     ret
```

```
WinMain ENDP

; -------------- 窗口消息处理程序 WndProc --------------
;   hWnd    窗口句柄
;   uMsg    消息号，指明消息的种类
;   wParam  该消息的附加信息。若是子消息号，则是嵌套分支判断的依据
;  lParam   该消息的附加信息
WndProc proc hWnd:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD
  LOCAL hdc:HDC            ; 存放设备上下文句柄
 .IF uMsg==WM_DESTROY
     invoke PostQuitMessage,NULL
 .ELSEIF uMsg==WM_CHAR
      invoke GetDC,hWnd     ; 根据窗口句柄确定设备句柄
      mov    hdc,eax
      mov    eax,wParam        ; 将按键的 ASCII 码送到 AL 中
      mov    szASCII-8,al      ; 取代显示串中的‘？’
      invoke  Convert,al,addr szASCII    ; 数串转换
      invoke TextOut,hdc,40,40,addr szDisplayStr,dwDisplayLength   ;显示串
 .ELSE
      invoke DefWindowProc,hWnd,uMsg,wParam,lParam
              ;  未在本函数中处理的消息皆调用缺省处理
      ret
 .ENDIF
  xor    eax,eax
  ret
WndProc endp

; -------------- 数串转换程序 Convert--------------
; lpStr : 存放转换结果串的首地址
; bChar : 待转换的数值
Convert  proc bChar:BYTE,lpStr:DWORD
         push esi
         push eax
         mov  al,bChar
         mov  esi,lpStr
         shr  al,4
         cmp  al,10
         jb   L1
         add  al,7
L1:      add  al,30H
```

```
        mov   [esi],al
        mov   al,bChar
        and   al,0FH
        cmp   al,10
        jb    L2
        add   al,7
L2:     add   al,30H
        mov   [esi+1],al
        pop   eax
        pop   esi
        ret
Convert endp
end   start
```

从本例中可以看到使用了多个 Windows API 函数，要编写好一个 Windows 窗口应用程序，就要熟悉 API 函数，也要熟悉 Windows 的消息机制。

13.3.2 含菜单和对话框的窗口程序

下面介绍一个含有菜单和对话框的简单窗口应用程序的开发过程，并给出了程序的源代码。

1、程序功能

在屏幕上创建一个窗口，窗口的标题为：“Menu Dialog Test Window”，它的菜单项如图 13.6 所示。

图 13.6 程序的运行界面

点击帮助菜单下的版本信息、作者信息，都会出现一个消息框，列出有关信息。

点击操作菜单下的“打开一个对话框”，在框中输入信息后，按“确定”按钮，则会在主窗口中显示在对话框中输入的信息，如图 13.7 所示，在框中输入了 Welcome，在主窗口中也显示该串；若在对话框单击“取消”按钮、或者单击对话框右上角的“关闭”按钮，则在主窗口中显示不同的信息。

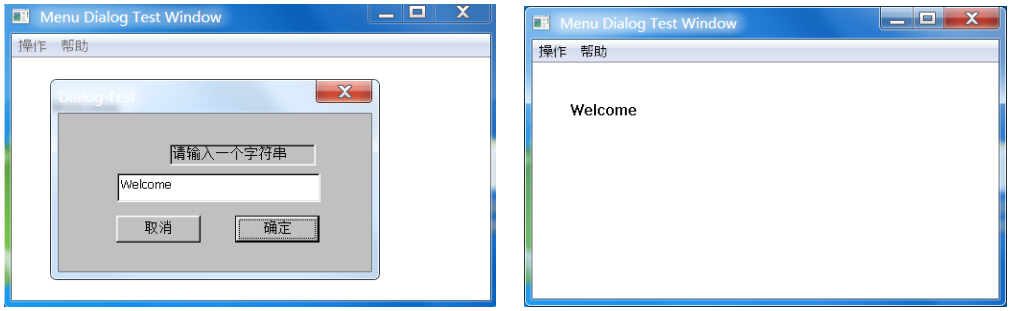

图 13.7 执行打开对话的运行界面

2. 资源编辑

在本程序中，要使用菜单资源（MENU）、对话框资源（DIALOG），还要用到 Windows 系统提供的标准化的资源，如应用程序标题栏左边的程序图标（ICON）、鼠标形状（CURSOR）。编程者完全可以使用 Visual Studio 自带的资源编辑器来编辑资源文件。这是一种所见即所得的资源编辑器，与 C++程序开发所采用的资源编辑形式完全相同。资源编辑器会根据用户的操作自动生成资源脚本文件*.rc 和资源文件*.res。资源脚本文件可以用文本编辑器打开。

（1） 菜单的定义

在本例中，菜单和对话框资源的脚本描述如下：

```
#include "resource.h"
IDM_MYMENU MENU
BEGIN
    POPUP "操作"
    BEGIN
        MENUITEM "打开一个对话框\tCtrl+O",     ID_OP_OPEN
        MENUITEM "退出",                       ID_OP_EXIT
    END
    POPUP "帮助"
    BEGIN
        MENUITEM "版本信息",                   ID_HELP_VERSION
        MENUITEM "作者信息",                   ID_HELP_AUTHOR
    END
END
```

菜单的定义格式为：

```
菜单 ID  MENU  [DISCARDABLE]
BEGIN
   菜单项定义 ……
END
```

首先，整个菜单有一个编号，取值为 1～65535。但是写程序时用数值编号易错，因而一般都用一个符号常量来代替。例如，在本例中取名为 IDM_MYMENU,在包含的头文件 resource.h 中以 C 语言的格式定义该符号常量：“#define IDM_MYMENU 101”。当然可以用其他符合规定的标识符和常量值。DISCARDABLE 是菜单的一个可选内存属性，含有该选项，表示菜单不再使用时可以从内存中暂时释放它。

菜单项的定义有三种形式：

① MENUITEM　菜单文字, 命令 ID　[, 选项列表]

② MENUITEM　SEPARATOR

③ POPUP　菜单文字 [, 选项列表]

```
BEGIN
    菜单项定义 ……
END
```

第三种形式定义了一个弹出式菜单，在其菜单项定义中还可以包含弹出式菜单，形成一个层次结构。第二种形式只是在菜单项之间增加了一个分隔线。第一种形式定义的菜单项可被单击，它有命令 ID。当单击一个有命令 ID 的菜单项时，Windows 系统就会向窗口消息处理程序发送 WM_COMMAND 消息，消息的参数 wParam 就是该 ID。与整个菜单的 ID 一样，取值范围在 1～65535。为了便于写程序，也采用符号常量代替数值量。在取名时，要注意名称的可读性。一般情况下，不同的菜单项有不同的 ID。若两个菜单项有相同的 ID，它们被单击的消息参数 wParam 就相同，会执行相同的功能。

菜单项的选项列表用来定义菜单的各种属性。如 CHECKED 表示菜单项名的前面打对勾；GRAYED 表示菜单项是灰化的；INACTIVE 表示菜单项禁用。

（2） 对话框的定义

对话框定义的格式为：

```
对话框 ID　DIALOGEX [DISCARDABLE] X 坐标, Y 坐标, 宽度, 高度
[可选属性]
BEGIN
    子窗口控件……
END
```

可选属性包括标题文字、窗口风格、扩展风格、字体、菜单、窗口类等，各项的格式如下。

```
标题文字：CAPTION 窗口标题栏上的文字
窗口风格：STYTLE 风格组合
扩展风格：EXSTYTLE 风格组合
字　　体：FONT 大小，“字体名”
菜　　单：MENU 菜单 ID
窗 口 类：CLASS “类名”
```

子窗口控件的一般格式为：

```
控件名称 文本, ID, 类, 风格, X 坐标, Y 坐标, 宽度, 高度 [,扩展风格]
```

控件名称包括 PUSHBUTTON（按钮）、DEFPUSHBUTTON（默认按钮）、EDITTEXT（文本编辑框）、CHECKBOX（复选框）、RADIOBUTTON（单选按钮）、LISTBOX（列表框）、COMBOBOX（组合框）、LTEXT、CTEXT/RTEXT（左/居中/右对齐的静态文本框）、SCROLLBAR（滚动条）等等。

下面给出了本程序的对话框的脚本描述。

```
IDD_MY_DIALOG DIALOGEX 0, 0, 186, 82
STYLE  DS_SETFONT | DS_MODALFRAME | DS_FIXEDSYS | WS_POPUP
       | WS_CAPTION | WS_SYSMENU
CAPTION "Dialog Test"
```

```
FONT 8, "MS Shell Dlg", 400, 0, 0x1
BEGIN
    DEFPUSHBUTTON   "确定",IDOK,105,53,50,14
    PUSHBUTTON      "取消",IDCANCEL,34,53,50,14
    LTEXT           "请输入一个字符串",IDC_STATIC,66,16,87,11,WS_BORDER
    EDITTEXT        IDC_MYDIALOG_EDIT,35,31,121,15,ES_AUTOHSCROLL
END
```

在资源脚本文件中，出现了一系列的资源名称，这些名称是在编辑资源时取的名字，它们都对应了一个资源编号。在使用 VS 编辑器时，将它们的定义放在了 resource.h 中。

```
#define IDM_MYMENU                      101
#define IDD_MY_DIALOG                   102
#define IDC_MYDIALOG_EDIT               1001
#define ID_Menu                         40004
#define ID_OP_OPEN                      40005
#define ID_OP_EXIT                      40006
#define ID_HELP_VERSION                 40007
#define ID_HELP_AUTHOR                  40008
```

在汇编源程序中，也要使用这些资源编号，可以增加一个头文件，如 resource.inc，用汇编语言的格式来定义这些符号常量。

```
IDM_MYMENU          EQU     101
IDD_MY_DIALOG       EQU     102
IDC_MYDIALOG_EDIT   EQU     1001
ID_Menu             EQU     40004
ID_OP_OPEN          EQU     40005
ID_OP_EXIT          EQU     40006
ID_HELP_VERSION     EQU     40007
ID_HELP_AUTHOR      EQU     40008
```

3、主程序

在本例中，有两个窗口，一个是对话框窗口，一个是主窗口，两个都有各自独立的窗口消息处理函数。发生在各自框上的消息，将有由操作系统的消息派发到对应的消息处理函数中。源程序如下。

```
.686P
.model flat,stdcall
OPTION CASEMAP:NONE
WinMain proto :DWORD
WndProc proto :DWORD,:DWORD,:DWORD,:DWORD       ; 主窗口的消息处理程序
DialogProc proto :DWORD,:DWORD,:DWORD,:DWORD    ; 对话框窗口的消息处理程序
include windows.inc
include user32.inc
include kernel32.inc
```

```
include gdi32.inc
include resource.inc
.data
szClassName       db "MenuDialogTest",0
szTitle           db " Menu Dialog Test Window",0
hInstance         dd  0
szMessageBoxTitle db "Message Box Title",0
szMessageConfirm db "Are you sure to close the window ?",0
szAuthorInfo      db "Author : Xu Xiang Yang",0
szVersionInfo     db "Version : 1.0",0
szInputText       db  100 dup(0)
dwInputLength dd  0
szCancelPress     db "Cancel Button is pressed ",0
dwCancelPressLength = $ - szCancelPress -1
szClosePress      db  " Close Window is pressed. ",0
dwClosePressLength = $ - szClosePress -1
szClearText       db  100 dup(' ')
.code
start:
    invoke GetModuleHandle,NULL
    mov    hInstance,eax
    invoke WinMain,hInstance
    invoke ExitProcess,eax

;-------------- 窗口主程序 WinMain --------------
WinMain proc hInst:DWORD
    LOCAL wc:WNDCLASSEX
    LOCAL msg:MSG
    LOCAL hwnd:HWND
    LOCAL hMenu:HMENU
    invoke RtlZeroMemory,addr wc,SIZEOF wc
    mov   wc.cbSize,SIZEOF WNDCLASSEX
    mov   wc.style, CS_HREDRAW or CS_VREDRAW
    mov   wc.lpfnWndProc, offset WndProc
    push  hInst
    pop   wc.hInstance
    mov   wc.hbrBackground,COLOR_WINDOW+1
    mov   wc.lpszClassName,offset szClassName
    invoke LoadIcon,NULL,IDI_APPLICATION
    mov   wc.hIcon,eax
```

```
    invoke LoadCursor,NULL,IDC_ARROW
    mov   wc.hCursor,eax
    invoke RegisterClassEx, addr wc
    invoke LoadMenu, hInst, IDM_MYMENU    ; 装载菜单
    mov    hMenu,eax                      ; 在创建窗口时带上菜单
    invoke CreateWindowEx,NULL,addr szClassName,addr szTitle,\
           WS_OVERLAPPEDWINDOW, \
           CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,\
           NULL, hMenu, hInst, NULL
    mov   hwnd,eax
    invoke ShowWindow, hwnd,SW_SHOWNORMAL
    invoke UpdateWindow, hwnd
StartLoop:
    invoke GetMessage,addr msg,NULL,0,0
    cmp  eax,0
    je   ExitLoop
    invoke TranslateMessage, addr msg
    invoke DispatchMessage, addr msg
    jmp StartLoop
ExitLoop:
     mov  eax,msg.wParam
     ret
WinMain ENDP

; -------------- 主窗口的消息处理程序 WndProc --------------
;   hWnd    主窗口的窗口句柄
;   uMsg    消息号，指明消息的种类
;   wParam  该消息的附加信息。若是子消息号，则是嵌套分支判断的依据
;  lParam   该消息的附加信息
WndProc proc hWnd:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD
  LOCAL hdc:HDC
  LOCAL x:DWORD
.IF uMsg == WM_COMMAND        ; 菜单上的消息，
   .IF wParam == ID_OP_OPEN
       invoke DialogBoxParam,NULL,IDD_MY_DIALOG,hWnd,offset DialogProc,NULL
              ; 创建对话框窗口，指明了该窗口的处理函数是 DialogProc
       invoke GetDC,hWnd
       mov    hdc,eax
       invoke TextOut,hdc,40,40,addr szClearText,100
       invoke TextOut,hdc,40,40,addr szInputText,dwInputLength
```

```
                ; 显示信息，信息在对话框的消息处理函数中设置
   .ELSEIF  wParam == ID_OP_EXIT
        invoke DestroyWindow, hWnd
;       invoke PostQuitMessage,NULL
   .ELSEIF  wParam == ID_HELP_VERSION
        invoke MessageBox, hWnd,addr szVersionInfo, addr szMessageBoxTitle,MB_OK
   .ELSEIF  wParam == ID_HELP_AUTHOR
        invoke MessageBox, hWnd,addr szAuthorInfo, addr szMessageBoxTitle,MB_OK
   .ENDIF   ; 菜单消息处理结束
.ELSEIF uMsg==WM_CLOSE      ; 关闭窗口消息
      invoke MessageBox, hWnd,addr szMessageConfirm,
                         addr szMessageBoxTitle,MB_YESNO
      .if  eax == IDYES  ; 上面是询问是否要关闭，点击 YES，才关闭，否则不做处理
           invoke DestroyWindow, hWnd
      .endif
.ELSEIF uMsg==WM_DESTROY
      invoke PostQuitMessage,NULL
 .ELSE
      invoke DefWindowProc,hWnd,uMsg,wParam,lParam
      ret
 .ENDIF
  xor    eax,eax
  ret
WndProc endp

; -------------- 对话框上的窗口的消息处理程序 DialogProc --------------
;   hWnd    对话框窗口的窗口句柄
DialogProc proc hWnd:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD
LOCAL  hEdit:DWORD
.IF uMsg == WM_CLOSE
     invoke wsprintf, addr szInputText, addr szClosePress
     mov  dwInputLength, dwClosePressLength
     invoke  EndDialog, hWnd, NULL
.ELSEIF uMsg == WM_COMMAND
   .IF wParam == IDOK
        invoke GetDlgItem, hWnd,IDC_MYDIALOG_EDIT
        mov hEdit, eax
        invoke GetWindowTextLength,hEdit
        mov dwInputLength, eax
        invoke GetDlgItemText, hWnd,IDC_MYDIALOG_EDIT, offset szInputText,100
```

```
            invoke EndDialog,hWnd,NULL
        .ELSEIF  wParam == IDCANCEL
            invoke    wsprintf, addr szInputText, addr szCancelPress
            mov dwInputLength, dwCancelPressLength
            invoke EndDialog,hWnd,NULL
        .ENDIF
    .ENDIF
      xor     eax,eax
      ret
DialogProc endp
end  start
```

在编译上述文件时，还需要实现所用到的 API 函数的库文件。有于 Visual Studio 一般已缺省包含了 kernel32.lib、user32.lib、gdi32.lib 等，因而在程序中无需使用 includelib 包含这些库文件。它们的用法与 C 语言程序开发没有差别。

由于 Windows 操作系统功能强大，相应的编程环境的内容是非常丰富的。上面仅介绍了最基本的内容。更全面和详细的内容请参考相关的编程指南及 SDK 应用指南。

13．4 与 C 语言开发的窗口程序比较

根据本章第 13.2 节的介绍，窗口应用程序有比较固定的结构。VS2019 中定制了多种类型的程序的模板，使用模板可以自动生成程序的框架。自动生成的程序可以直接编译、运行。

下面给出了使用“Windows 桌面向导”自动生成的应用程序。程序运行界面如图 13.8 所示。

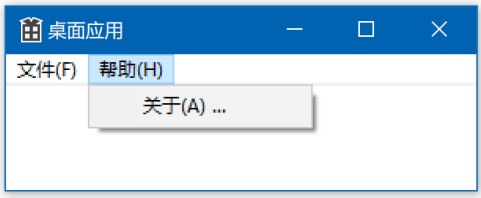

图 13.8 自动生成的桌面应用程序的运行界面

自动生成的程序如下。

```
#include "framework.h"
#include "桌面应用.h"
#define MAX_LOADSTRING 100
HINSTANCE hInst;                                // 当前实例
WCHAR szTitle[MAX_LOADSTRING];                  // 标题栏文本
WCHAR szWindowClass[MAX_LOADSTRING];            // 主窗口类名
// 此代码模块中包含的函数的前向声明:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);
int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
```

```
                        _In_opt_ HINSTANCE hPrevInstance,
                        _In_ LPWSTR        lpCmdLine,
                        _In_ int           nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);
     // 初始化全局字符串
    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDC_MY, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);
    // 执行应用程序初始化:
    if (!InitInstance (hInstance, nCmdShow))
    {
        return FALSE;
    }
    HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_MY));
    MSG msg;
    // 主消息循环:
    while (GetMessage(&msg, nullptr, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }
    return (int) msg.wParam;
}

//  函数: MyRegisterClass()
//  目标: 注册窗口类。
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEXW wcex;
    wcex.cbSize = sizeof(WNDCLASSEX);
    wcex.style          = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc    = WndProc;
    wcex.cbClsExtra     = 0;
    wcex.cbWndExtra     = 0;
    wcex.hInstance      = hInstance;
    wcex.hIcon          = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_MY));
    wcex.hCursor        = LoadCursor(nullptr, IDC_ARROW);
    wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
    wcex.lpszMenuName   = MAKEINTRESOURCEW(IDC_MY);
    wcex.lpszClassName  = szWindowClass;
```

```
    wcex.hIconSm            = LoadIcon(wcex.hInstance,
    MAKEINTRESOURCE(IDI_SMALL));
        return RegisterClassExW(&wcex);
    }

    //    函数: InitInstance(HINSTANCE, int)
    //    目标: 保存实例句柄并创建主窗口
    //          在此函数中，在全局变量中保存实例句柄并创建和显示主程序窗口。
    BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
    {
        hInst = hInstance; // 将实例句柄存储在全局变量中
        HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
            CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);
        if (!hWnd)
        {
            return FALSE;
        }
        ShowWindow(hWnd, nCmdShow);
        UpdateWindow(hWnd);
        return TRUE;
    }

    //  函数: WndProc(HWND, UINT, WPARAM, LPARAM)
    //  目标: 处理主窗口的消息。
    //  WM_COMMAND   - 处理应用程序菜单
    //  WM_PAINT     - 绘制主窗口
    //  WM_DESTROY   - 发送退出消息并返回
    LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
    {
        switch (message)
        {
        case WM_COMMAND:
            {
                int wmId = LOWORD(wParam);
                // 分析菜单选择:
                switch (wmId)
                {
                case IDM_ABOUT:
                    DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
                    break;
                case IDM_EXIT:
                    DestroyWindow(hWnd);
                    break;
```

```
                default:
                    return DefWindowProc(hWnd, message, wParam, lParam);
                }
            }
            break;
        case WM_PAINT:
            {
                PAINTSTRUCT ps;
                HDC hdc = BeginPaint(hWnd, &ps);
                // TODO: 在此处添加使用 hdc 的任何绘图代码...
                EndPaint(hWnd, &ps);
            }
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
    }

    // “关于”框的消息处理程序。
    INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
    {
        UNREFERENCED_PARAMETER(lParam);
        switch (message)
        {
        case WM_INITDIALOG:
            return (INT_PTR)TRUE;
        case WM_COMMAND:
            if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
            {
                EndDialog(hDlg, LOWORD(wParam));
                return (INT_PTR)TRUE;
            }
            break;
        }
        return (INT_PTR)FALSE;
    }
```

有兴趣的读者，可以通过阅读、反汇编调试该 C 语言程序，熟悉 Windows 窗口应用程序的结构、消息处理机制、窗口应用程序的运行过程。

当然，使用 VS2019 可以创建基于 MFC（Microsoft Foundation Classes）的应用程序。MFC 将

Windows API 函数进行封装，形成很多类库，供开发者使用。开发 MFC 应用程序时，需要面向对象（C++）的程序设计知识，以及对 MFC 类库的了解。

总之，开发工具越高级，开发应用程序就越简单，离计算机底层的实现机理就越远。利用所学的汇编语言知识，可以从机器的角度了解程序的运行过程的，掌握其原理，从而更好地开发出程序。

习题 13

13.1　Windows 窗口应用程序由哪些部分组成？各个组成部分完成的主要功能是什么？

13.2　Windows 窗口应用程序工作的基本流程是什么？

13.3　句柄是什么？

13.4 消息是什么？

13.4 单击一个菜单项，会产生什么消息？如何知道单击的是哪一个菜单项？

13.5 鼠标移动会产生什么消息？如何知道鼠标在什么位置？

13.6 编写一个程序，在窗口中显示鼠标的位置信息。当鼠标移动时，显示鼠标的新位置坐标。

上机实践 13

13.1 用 VS2019 创建 Windows 应用程序。阅读自动生成的程序，描述程序的基本结构，消息处理机制。

13.2 在上机实践 13.1 的基础上，增加 13.3.2 节中程序的菜单项和对话框资源。

13.3 用反汇编的方法，比较 C 语言程序的反汇编结果和用汇编语言编写程序的共同点和差异。