第 3 章

Windows 应用程序

在本章中，要学习使用 Visual C++ 开发 Windows 应用程序的相关基本概念。将通过开发一个动态显示漂亮的万花筒的示例，来介绍直接使用 Windows 系统 API 编写 Windows 程序的过程。

3.1 Windows 编程基础

Microsoft Windows 是一个应用于微型计算机上的图形化用户界面的操作系统。它为应用程序提供了一个由一致的窗口和菜单结构构成的多任务环境。

Windows 的程序设计语言，包括 Visual C++、Visual Basic、Visual Java 等，都称为“面向对象(Object Oriented)”的程序设计语言。在 Windows 编程中，“对象(Object)”是指 Windows 的规范部件，包括各种窗口、菜单、按钮、对话框及程序模块等，这些多样化的“对象”能够充分满足构成应用软件操作界面的需要，因此编写 Windows 程序的相当一部分工作是在创建对象和为对象属性赋值。对象在具有规范形态的基础上还具有规范的操作模式，如能够对鼠标或键盘的规范操作产生规范的程序分支响应。

用户可以采用传统的源程序代码编写方法编写程序，也可以采用 Windows 特色的交互式操作方法构造程序。采用交互式方法时，可视化开发平台给出了许多选用的对象，程序员可以选择所需要的对象并为对象的属性确定参数值，由此搭建起应用程序的“大框架”。在这个“大框架”中，程序员根据需要进一步编写必要的细节代码段，最后构成完整的应用程序。

使用 Visual C++ 2008 开发 Windows 应用程序时，通常有 3 种基本的方法：

- 使用 Windows 提供的 Windows API 函数。
- 使用 Microsoft 提供的 MFC 类库。
- 使用 Windows Forms。

API 是应用程序编程接口(Application Programming Interface)的缩写，Windows API 是 Windows 系统和 Windows 应用程序间的标准程序接口。API 为应用程序提供系统的各种特殊函数及数据结构定义，Windows 应用程序可以利用上千个标准 API 函数调用系统功能。使用 Windows API，构成应用程序 GUI 的所有元素都必须以编写代码的方式创建。

MFC 类库集成了大量已经预先定义好的类，用户可以根据编程的需要调用相应的类，或根据需要自定义有关的类。在 MFC 应用程序中，能够以图形的方式布置窗体的控

件，在构建 GUI 方面得到很大的方便。但是仍然需要对程序和用户之间的交互作用编写大量的代码。

Windows Forms 是一种基于窗体的开发机制，用于创建在 CLR 中执行的应用程序。在 Windows Forms 应用程序中，能够以图形的方式布置与用户交互的控件，只需按自己的意思将控件放在窗体适当的位置即可，创建这些控件的代码将自动生成。使用 Windows Forms 是目前最快、最简单的生成 Windows 应用程序的机制，相对其他两种方法而言，程序员需要编写的代码量大为减少。

本章将重点讲述与这 3 种 Windows 应用程序开发方法有关的基本概念，在后面几章将更加详细地学习使用 MFC，并通过一些实例来加深对它们的理解。

编写 Windows 应用程序与编写 DOS 应用程序有很大区别，掌握 Windows 编程方法必须首先了解以下内容：

- 窗口的概念。
- 事件驱动的概念。
- 消息及其在编程中的应用。

3.1.1　窗口的元素

窗口是 Windows 应用程序基本的操作单元，是应用程序与用户之间交互的接口环境，也是系统管理应用程序的基本单位。一个基本的 Windows 应用程序窗口的组成如图 3-1 所示。编写一个 Windows 应用程序首先应创建一个或多个窗口，应用程序的运行过程即是窗口内部、窗口与窗口之间、窗口与系统之间进行数据处理与数据交换的过程。

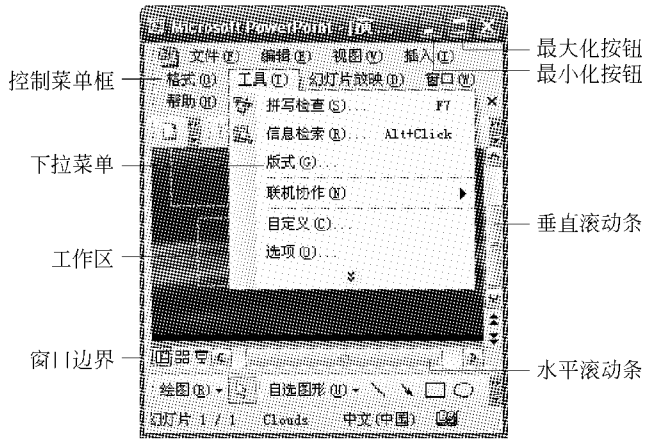

图 3-1　Windows 应用程序窗口的组成

3.1.2　事件驱动

基于 DOS 的应用程序主要使用顺序的、过程驱动的程序设计方法。顺序的、过程驱动的程序有一个明显的开始、明显的过程和一个明显的结束，因此，程序能直接控制各个过程执行的顺序。

基于 Windows 的应用程序设计方法与 DOS 程序设计方法的不同就在于 Windows 程序是事件驱动的。事件驱动的程序不是由程序的顺序来控制，而是由事件的发生来控制。

假设有这样一个应用程序，该程序的功能是计算一个学期中进行了 3 次测验后一个班的平均成绩。在一个顺序的、过程驱动的程序中，可以设想用下面的步骤来实现该应用程序要求实现的功能：

(1) 输入学生姓名。

(2) 输入第一次测验成绩。

(3) 输入第二次测验成绩。

(4) 输入第三次测验成绩。

(5) 计算并显示平均成绩。

其逻辑流程图可以用图 3-2 来表示。这种设计是基于过程驱动的，它使用户只能按照程序规定好的步骤进行操作。尽管在顺序的、过程驱动的程序中也有许多处理异常的方法，但是这些异常处理也是顺序的、过程驱动的结构。

事件驱动型程序的设计是围绕着消息的产生与处理而展开的。消息是由操作系统产生的关于发生的事件的信息。Windows 程序员的工作就是对正开发的应用程序所要发出或要接收的消息进行排序和管理。由于 Windows 消息是随着事件产生的，因此消息是不会以任何预定义顺序出现的。图 3-3 所示是基于事件驱动的计算学生平均成绩的程序流程示意图。

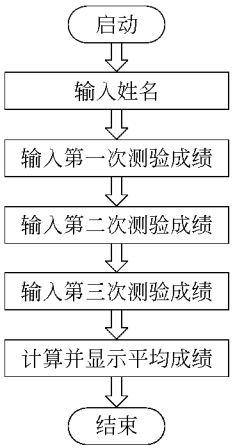

图 3-2 用过程驱动的方法来计算平均成绩

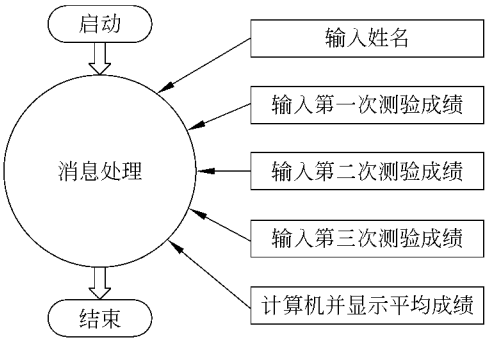

图 3-3 用事件驱动的方法来计算平均成绩

按图 3-3 所示流程设计的基于事件驱动的程序实现的功能，和用图 3-2 所示的流程所设计的基于过程的程序实现的功能是相同的。但是，用户可以在不同的窗口中来回切换，并不需要按顺序按步骤地进行数据的输入。例如，可以直接进入输入第三次测验成绩的窗口输入第三次测验成绩，而不必先输入第一、二次测验成绩。

事件驱动程序方法提供了许多便利，对于那些需要大范围用户干预的应用程序来说，更显其优越性。

3.1.3　Windows 消息

Windows 应用程序中的事件指的是单击鼠标、按下键盘的某个按键等动作。Windows 操作系统将每个事件记录在一条消息中，放入目标程序的消息队列。通过发送消息，Windows 可以告诉程序某件事情需要完成，某种信息已经可用，或者某个事件已经发生，程序将以适当的方式响应这些消息。

Windows 编程中常用的消息有窗口管理消息、初始化消息、输入消息、系统消息、剪贴板消息、控件处理消息、控件通知消息、非用户区消息、MDI(多文档界面)消息、DDE(动态数据交换)消息以及应用程序自定义的消息等。

3.2　Windows 应用程序的结构

Windows 的应用程序具有相对固定的基本结构，其中由入口函数 WinMain、消息处理函数 WindowsProc(有时也称窗口处理函数)构成基本框架，并包含各种数据类型、数据结构与函数等。入口函数 WinMain 和消息处理函数 WindowsProc 是 Windows 应用程序的主体。

下面通过创建一个仅使用 Windows API 的应用程序来进一步了解 Windows 程序的结构。首先新建一个使用“Win32 项目”模板的项目，将它命名为“MyWindow”，如图 3-4 所示，单击“确定”按钮。

图 3-4　新建一个 Win32 项目

在弹出的应用程序设置对话框中单击“下一步”按钮，然后选中“空项目”，最后单击“完成”按钮即可。

在解决方案资源管理器中的“源文件”图标上右击，选择“添加”|“新建项”(如图 3-5 所示)，将弹出“添加新项”对话框。用户可以选择“C++ 文件(.CPP)”模板，在“名称”文本框中输入“winmain.cpp”，单击“添加”按钮，这样就新建了一个 C++ 文件，如图 3-6 所示。该源文件将包含 WinMain 和 WindowsProc 两个函数。下面分别讲述构成这两个函数的各个部件的功能以及如何编写它们。

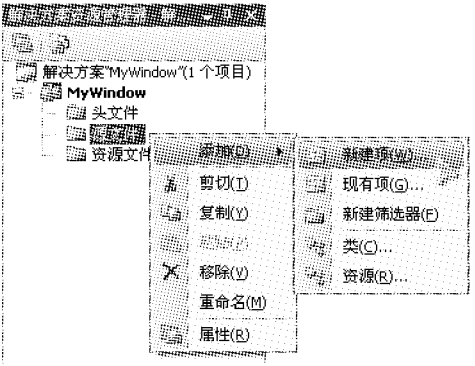

图 3-5 新建一个源文件

图 3-6 新建一个 C++ 文件

3.2.1 WinMain 函数

WinMain 函数是所有 Windows 应用程序的入口，类似控制台应用程序中的 main 函数，其功能是完成一系列的定义和初始化工作，并产生消息循环。WinMain 函数有 4 个形参和 1 个 int 类型的返回值，其声明形式如下：

```
int WINAPI WinMain
(
    HINSTANCE hInstance,          //应用程序当前实例句柄
    HINSTANCe hPrevInst,          //应用程序其他实例句柄
    LPSTR lpszCmdLine,            //指向程序命令行参数的指针
    Int nCmdShow                  //应用程序开始执行时窗口显示方式的整数值标识
);
```

其中：

- hInstance 是指向当前程序实例的句柄，每个实例都有自己独特的 hThisInst 值。
- hPrevInst 是从 16 位版本的 Windows 操作系统继承下来的，在 32 位的 Windows

下面已经完全失去意义，该参数的值始终是 NULL。

- lpszCmdLine 是指向某个字符串的指针，该字符串包含程序的命令行参数。LPSTR 是一种 Windows 类型，用来指定 32 位的字符串指针。
- nCmdShow 用来选择窗口的外观，例如可以选择窗口启动时正常显示，也可以选择启动时最小化显示。

该函数在返回类型说明符 int 后面有一个函数说明符 WINAPI，这是一个 Windows 定义的说明符，将使系统以某种特殊的方式处理函数名和实参。

Windows 应用程序可能并行地多次执行，因而可能出现同一个应用程序的多个窗口同时存在的情况，这也是 Windows 操作系统能进行多任务管理的特点。Windows 系统将应用程序的每一次执行称为该应用程序的一个实例(instance)，并使用一个实例句柄来唯一地标识它。

WinMain 函数实现以下 4 个功能：

- 定义窗口类，设置窗口的各种属性。
- 注册窗口类，建立窗口及执行其他必要的初始化工作。
- 进入消息循环，根据从应用程序消息队列接受的消息，调用相应的处理过程。
- 当消息循环检索到 WM_QUIT 消息时终止程序运行。

下面依次看一下这几个部分，然后创建一个完整的 WinMain 函数。

(1) 定义窗口类。

Windows 定义了一个名为 WNDCLASSEX 的 struct 类型，存储在该结构中的数据用来设置窗口的形式与功能。定义窗口类实际上就是创建一个 WNDCLASSEX 类型的变量，并给该变量的每个成员赋值。

WNDClASSEX 结构的定义如下所示：

```
struct WNDCLASSEX
{
    UINT cbSize;                //本结构体的大小(所占的字节数)
    UINT style;                 //窗口类型风格
    WNDPROC lpfnWndProc;        //指向消息处理函数
    int cbClsExtra;             //额外数据，赋值为 0 即可
    int cbWndExtra;             //额外数据，赋值为 0 即可
    HINSTANCE hInstance;        //当前应用程序实例的句柄
    HICON hIcon;                //最小化时应用程序的图标
    HCURSOR hCursor;            //窗口中光标的图标
    HBRUSH hbrBackground;       //窗口背景色
    LPCWSTR lpszMenuName;       //菜单名，可以设置为 NULL
    LPCWSTR lpszClassName;      //窗口类名称，必须指定
    HICON hIconSm;              //与窗口类相关的小图标，可以设置为 NULL
}
```

定义窗口类首先要做的是声明一个 WNDCLASSEX 类型的变量：

```
WNDCLASSEX wndclass;
```

接下来需要做的全部工作就是填写 wndclass 各个成员的数值。

- cbSize：这里需要用到 sizeof 运算符，其赋值语句如下：

```
wndclass.cbSize=sizeof(WNDCLASSEX);
```

- style：该成员决定窗口行为的各个方面，可以从许多选项值中进行选择（请参见 MSDN）。通常将窗口类型设为默认类型：

```
wndclass.style=0;
```

- lpfnWndProc：这是指向消息处理函数的指针。前面已经介绍过，这里的消息处理函数是 WindowProc，那么初始化该变量的语句如下：

```
wndclass.lpfnWndProc=WindowProc;
```

- cbClsExtra 和 cbWndExtra：这两个成员通常设置为 0 即可。
- hInstance：这是当前应用程序实例的句柄，应将它赋值为 Windows 传递给 WinMain 函数的值：

```
wndclass.hInstance=hInstance;
```

- hIcon、hCursor、hbrBackground：这 3 个成员分别定义了最小化窗口时的图标、窗口使用的光标、窗口的背景色，在此将它们设置为标准的 Windows 值：

```
wndclass.hIcon=LoadIcon( NULL, IDI_APPLICATION);
wndclass.hCursor=LoadCursor( NULL, IDC_ARROW);
wndclass.hbrBackground= (HBRUSH)GetStockObject(WHITE_BRUSH);
```

在上面的 3 条语句中，用到了 3 个 Windows API 函数：LoadIcon、LoadCursor 和 GetStockObject，即图标设为 Windows 的默认图标，光标设为标准的箭头光标，窗口的背景设为白色。

函数 LoadIcon 的作用是在应用程序中加载一个窗口图标。其原型为：

```
HICON LoadIcon
(
    HINSTANCE hInstance,      //图标资源所在的模块句柄，为 NULL，则使用系统预定义图标
    LPCTSTR lpIconName        //图标资源名或系统预定义图标标识名
)
```

函数 LoadCursor 的作用是在应用程序中加载一个窗口光标。其原型为：

```
HCURSOR LoadCursor
(
  HINSTANCE hInstance,        //光标资源所在的模块句柄，为 NULL，则使用系统预定义光标
  LPCTSTR lpCursorName        //光标资源名或系统预定义光标标识名
)
```

函数 GetStockObject 的作用是获取系统提供的背景刷，其原型为：

```
HBRUSH GetStockObject(int nBrush); //nBrush 为系统提供的背景刷的标识名
```

- lpszMenuName：该成员指向定义窗口菜单的资源。在本例中，窗口没有菜单资源，所以将它设置为空：

```
wndclass.lpszMenuName=NULL;
```

- lpszClassName：该成员存储了窗口类的名称，在创建窗口时会用到它，其设置语句如下：

```
static LPCWSTR lpszClassName=L"窗口";
wndclass.lpszClassName=lpszClassName;
```

- hIconSm：该成员表示某个与该窗口类相联系的小图标，在此将它设置为空，Windows 将使用与 hIcon 成员相关的小图标替换。

```
wndclass.hIconSm=NULL;
```

到这里，wndclass 各个成员的初始化就完成了。

(2) 注册窗口类。

Windows 系统本身提供部分预定义的窗口类，程序员也可以自定义窗口类，窗口类必须先注册后使用。窗口类的注册由函数 RegisterClassEx()实现，RegisterClassEx 函数返回一个布尔值，注册成功则返回值为真。相应的语句为：

```
If ( !RegisterClassEx( &wndclass))          //wndclass 为窗口类结构
{
  MessageBeep(0);
  return FALSE;
}
```

(3) 创建窗口。

在注册窗口类之后，就可以创建该窗口了，这一部分由函数 CreateWindow()来实现。该函数的原型为：

```
HWND CreateWindow
(
    LPCTSTR lpszClassName,              //窗口类名
    LPCTSTR lpszTitle,                  //窗口标题名
    DWORD dwStyle,                      //创建窗口的样式，常用窗口样式如表 3-1 所示
    int x,                              //窗口左上角坐标
    int y,                              //窗口左上角坐标
    int nWidth,                         //窗口宽度
    int nHeight,                        //窗口度高
    HWND hwndParent,                    //该窗口的父窗口句柄
    HWENU hMenu,                        //窗口主菜单句柄
    HINSTANCE hInstance,                //创建窗口的应用程序当前句柄
    LPVOID lpParam                      //指向一个传递给窗口的参数值的指针
)
```

表 3-1 常用窗口样式

标　识	说　明
WS_BORDER	创建一个带边框的窗口
WS_CAPTION	创建一个带标题栏的窗口
WS_CHILD	创建一个子窗口，它不能与 WS_POPUP 样式一起用
WS_HSCROLL	创建一个带水平滚动条的窗口
WS_MAXIMIZEBOX	创建一个带有最大化按钮的窗口
WS_MAXIMIZE	创建一个最大化的窗口
WS_MINIMIZEBOX	创建一个带有最小化按钮的窗口
WS_MINIMIZE	创建一个最小化的窗口
WS_OVERLAPPED	创建一个带边框和标题的窗口
WS_OVERLAPPEDWINDOW	创建一个带边框、标题栏、系统菜单及最大、最小化按钮的窗口
WS_POPUP	创建一个弹出式窗口，它不能与 WS_CHILD 一起使用
WS_POPUPWINDOW	创建一个带边框和系统菜单的弹出式窗口
S_SYSMENU	创建一个带系统菜单的窗口
S_VSCROLL	创建一个带垂直滚动条的菜单
WS_VISIBLE	创建一个初始化为“可见”的窗口，该样式可被函数 ShowWindow 或 SetWindowPos 打开或关闭

在本例中，相应的代码为：

```
HWND hWnd;
static LPCWSTR lpszTitle=L"万花筒";
⋮
hWnd=CreateWindow
(
    lpszClassName,
    lpszTitle,
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT,                    //使用默认值
    CW_USEDEFAULT,                    //使用默认值
    480,                              //使用默认值
    360,                              //使用默认值
    0,                                //没有父窗口
    0,                                //不需要菜单
    hInstance,
    0
)
```

(4) 显示窗口。

在创建窗口之后，需要调用 ShowWindow()函数将窗口在屏幕上显示出来：

```
ShowWindow(hWnd, nCmdShow);          //显示窗口
```

其中，hWnd 为窗口句柄，nCmdShow 指出了在屏幕上显示窗口的方式，表 3-2 列出了常用的显示方式。

表 3-2　常用窗口显示标识及其说明

标　识	说　明	标　识	说　明
SW_HIDE	隐藏窗口	SW_SHOWNA	按当前的状态显示窗口
SW_SHOW	按当前的位置和大小激活窗口	SW_SHOWNORMAL	显示并激活窗口

显示窗口后，应用程序常常调用 UpdateWindow()函数更新并绘制用户区，并发出 WM_PAINT 消息。相应的代码为：

```
UpdateWindow(hwnd);
```

(5) 消息循环。

最后一个需要 WinMain 函数完成的任务是处理 Windows 为应用程序排好的消息队列。Windows 将产生的消息放入应用程序的消息队列中，而应用程序的 WinMain 函数从消息循环提取队列中的消息，并将其传递给消息处理函数 WindowsProc 的相应过程处理。消息循环的代码如下：

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

其中，函数 GetMessage()的作用是从消息队列中读取一条消息，并将消息放在一个 MSG 结构中。其原型为：

```
GetMessage
(
    lpMSG,                    //指向 MSG 结构的指针
    hwnd,
    nMsgFilterMin,            //用于消息过滤的最小消息号值
    nMsgFilterMax             //用于消息过滤的最大消息号值
)
```

通过设置参数 nMsgFilterMin 和 nMsgFilterMax 可实现消息的过滤，即仅处理所确定的消息号范围内的消息。如果两个参数都为 0，则不过滤消息。

TranslateMessage()函数负责将消息的虚拟键转换为字符信息。DispatchMessgae()函数将参数 lpMSG 指向的消息传送到指定窗口函数。

当 GetMessage()函数返回零值，即检索到 WM_QUIT 消息时，程序将结束循环并