北京科海培训中心 SAMS

Borland C++ 3.1
开发 Windows 应用程序

[美] James W. McCord

钟向群 龙旭东 薛 安 陈 刚 译

施振川 文 极 校

清华大学出版社

目　录

第一部分　用 Borland C++ 进行 Windows 程序设计

第二部分 用 ObjectWindows 进行 Windows 程序设计

第三部分 参考信息

第一部分

用 Borland C ++进行 Windows 程序设计

第 1 章　Windows 入门

Microsoft Windows 是一个用于 MS-DOS 计算机上的图形用户界面，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。由于应用程序之间的窗口和菜单是一致的，因而对于用户来说，Windows 应用程序相对于用惯了的基于 DOS 的程序来说是很容易学习和使用的。多任务 Windows 系统的好处就在于允许你同时运行几个应用程序，特别是那些专为 Windows 环境而创建的应用程序。当然，Windows 也提供了运行非 Windows 的 MS-DOS 应用程序。Windows 3.1 环境见图 1.1 所示。

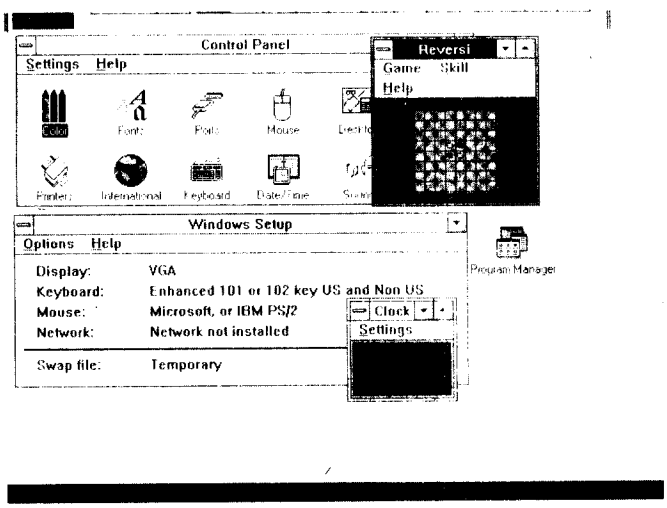

图 1.1　多任务的 Windows 环境

本章介绍 Windows 的图形用户界面，也涉及 Windows 开发过程的一些简史，关于 Windows 新的特征及其使用上的优点的讨论，并概述了一个窗口的基本构成。Microsoft Windows 是基于 MS-DOS 的机器上的一个颇为优良的图形用户界面环境。尽管还有其它的图形用户界面环境，但没有一个能在普及性上和 Windows 相比。图形用户界面使用位图(bitmap)显示来传达信息，可以显示的也不象传统的基于文本方式的 MS-DOS 程序那样局限于 ASCII 字符集，而且显示的图像准确地反映了打印机或绘图仪等输出设备上产生的图像，这种屏幕图像与输出图像的关系就叫做“所见即所得”(WYSIWYG)，这也是象 Windows

这样的图形用户界面如此普及的原因之一。Windows 提供给用户和程序员许多的优越性，这在本章将会讨论到。

1.1 Windows 的历史

Microsoft 在 1983 年就开始了 Windows 第一版的工作，并于 1985 年发行了 Windows 的 1.1 版。此版本是为带有 256K 的 RAM、双软驱及 8088 CPU 的 IBM PC 机系统设计的。Windows 1.1 支持自动的拼接式应用程序窗口及弹出窗口。

1987 年，Microsoft 公司公布了 Windows 的第二个主要版本 2.0 版。此版本包含了能支持重叠式窗口的新的用户界面。Windows 2.0 主要的优点是改进了扩充内存的使用。然而，因为 Windows 2.0 只能在实模式下运行，因此能够访问的内存总量仍然限制在 1M 之内。

1990 年，Microsoft 公司公布了 Windows 3.0 版本。此版本的 Windows 增加了针对用户自绘菜单、用户自绘列表框及用户自绘按钮等功能的支持，并尽可能地改进了其内存管理。于是，Windows 3.0 成为 MS-DOS 机器的标准的图形用户界面。

1992 年 4 月，Microsoft 公司发布了 Windows 3.1，增强了许多 Windows 3.0 所没有的新特性，并增加了新的功能，包括含有“文件拖动及放置”功能的新的文件管理器、对象的连接和嵌入(OLE)以及多媒体扩展等。

Windows 3.1 的新特性使 Windows 环境更适合于使用者了。而为了利用这些特性，程序员就必须懂得这些新的功能以及添加于 Windows 3.1 应用程序编程界面(API)中的数以百计的函数和消息。本章我们只是略览一下 Windows 3.1 的一些新特性，本书的第三部分增添了 Windows 3.1 的新函数、新消息、打印机 ESC 序列及数据结构的更为详细的参考信息。

1.2 Windows 3.1 的新内容

Windows 3.0 支持三种操作方式：实模式、标准模式和 386 增强模式。Windows 3.1 只支持两种操作方式：标准模式和 386 增强模式。Windows 3.1 以 386 增强模式运行时，应在至少 640K 基本内存和 1024K 扩展内存的 386 机器或更高档的机器上运行；当Windows 3.1 处于标准模式时，则应在至少 640K 基本内存外加 256K 扩展内存的 286 机器或更高档机器上运行。Windows 3.1 可以在少于 2M 内存的 386(或更高)机器上运行标准模式。

对象的连接和嵌入(OLE)是 Windows 3.1 的新特性，有了 OLE，应用程序可以通过嵌入对象或连接对象来共享数据。对象嵌入是指将诸如声音文件或图画之类的数据插入另一文件中的能力。例如，将一幅位图图像嵌入你的文件中而你想编辑此位图时，仅仅只要从字处理器中选择这个位图，则用来生成该嵌入位图的位图编辑器将自动打开，允许你去编辑此位图。

“对象连接”允许在应用程序之间共享数据。例如，你可以将一幅图连入一个字处理文件中，或是一个已存在的图表中，或是一个扩展的数据表中。你可以从包含这幅图的任何一个应用程序中来编辑这幅图。有了对象连接，只需一个连接数据存在，任何对该数据的修改，都会影响所有连入了此数据的文件。Windows 3.1 API 包含了许多 OLE 函数，这些函数编入第 13 章中，你很容易找到它们，因为它们均以字母 Ole 开头。

Windows 3.1 的另一个新特性是文件管理器中对“文件拖动和放置”的支持。此特性允

许用户只用拖着一个文件便可去移动、打印或拷贝。Windows 3.1 API 包含了几个可以用来实现这些新特性的新函数。

Windows 3.1 支持 TrueType 可变字体。TrueType 字体是可标度的字体，也就是说它们是可以改变为你所想要的精确的字体大小。Windows 3.1 API 包含了能在你的应用程序中实现可变字体的新函数和数据结构。

Windows 3.1 的多媒体特性允许用户存取不同的多媒体设备，可将声音、图形、动画、视频带入你的 Windows 环境之中。Windows 3.1 的多媒体特性是很激动人心的，但由于这些新特性是多媒体 API 的一部分，它们没有编入本书中。

Windows 3.1 给 Windows 用户提供了许多新的特性，同时向 Windows 程序员提供了更多新的挑战。Borland C ++包含了用 C 或 C ++设计和开发 Windows 3.1 应用程序的特性。

1.3 为什么要用 Windows?

使用 Windows 和 Windows 应用程序能给用户及程序员带来许多好处。诸如 Point—and—Click 功能及多任务功能对于用户来说是很有利的，而诸如独立于设备的图形及扩展的内存管理等特性对于程序员来说是很有帮助的。下面几段描述了 Windows 对于用户及程序员的一些优越性。

1.3.1 Windows 对于用户的优越性

对 Windows 和 Windows 应用程序的用户来说，一个优点就是一致的用户界面。因为大多数用户都在几个不同的软件包上工作，而每一个软件包都有它自己的用户界面，因而用户界面一致性的设计便非常重要了。Windows 环境中的每一个窗口都包含了相同的基本特性，由于这些基本特性在应用程序中是一致的，因而用户便能很容易地适应新的应用程序。

对用户的另一个好处便是使用了基于图形的图像来代表应用程序和数据。因为 Windows 是一个图形用户界面，因而采用了图形图像来代表诸如文件、应用程序、窗口和目录等物理数据结构。用户可以通过对鼠标的选择、连按两次、拖动等操作来管理这些物理结构。对用户来说，对一个代表某一个操作的图标连按两次比进入相应的目录再键入文件名去执行要容易得多。

“所见即所得”也是一个主要的好处。例如，大多数基于文本的字处理软件包使用只读存储器，屏幕显示是使用基本的输入输出系统(ROM BIOS)字符集，而 ROM BIOS 字符集并不是成比例的，与某一个操作所使用的字体没有直接关系，屏幕上显示的文本与打印机上输出的正文并不吻合。然而，由于有了“WYSIWYG”，Windows 可以把正文看作是一串图像，在屏幕上画出的图像，在打印的时候将以同样形式出现。

Windows 给用户提供了多任务功能。多任务对于用户是很重要的，因为它允许同时运行几个应用程序，因而你就无须节约工作文件，也无须退出一个操作再进入另一个操作；只要简单地在各个操作之间切换就可以了。由于 Windows 的存储管理功能，多任务功能大大提高了，Windows 3.0 支持用户存取扩展内存，可访问 16M 字节，由于 Windows 支持对所有可用内存资源的存取，用户就能更容易地优化系统了。

1.3.2 Windows 对于程序员的优越性

Windows 的许多对用户的优点也是对程序员的优点。例如，一致性的用户界面对用户是一个优点，因为应用程序的用户界面是相同的；而程序员也同样受益于一致的用户界面，因为基本的界面设计及工具都已建立了，由于界面对每一个应用程序是基本相同的，因而程序员可以花更多的时间去进行应用程序的实际功能设计，而在界面设计上花的时间就较少。

Windows 中的一些图形用户界面的基本设计对于程序员来说是另一个优点。程序员可以更容易地设计出代表诸如文件和目录等物理结构的图形表示，并能提供那些很容易使用的特性，例如弹出菜单、对话框等等。Windows 也提供了对鼠标和键盘的直接支持，这可以大大减少应用程序的开发时间。

Windows 对存储器的管理特性也是程序员的一个很明显的优点。这些特性允许你比使用传统的 MS-DOS 操作能存取更多的内存，从而能使你能使用绝大部分的系统内存资源，同时能保持系统设计中的灵活性。第 8 章“存储管理和 Windows”中将对 Windows 的存储管理特性提供更为详细的讨论。

Windows 也提供了开发独立于设备的图形能力。因为一个良好设计的 Windows 应用程序并不直接存取图形硬件(屏幕和打印机)，它只是和一个视频与系统或带有 Windows 驱动程序的打印机打交道。对于程序员来说，独立于设备的图形意味着代码并不依赖于某一确定的系统配置，而且，每一个应用程序员并不需要为所有可能的视频显示、适配器和打印机开发设备驱动程序。

1.4 一个窗口的标准的组成成分

一个 Windows 应用程序使用一个窗口进行对屏幕的输入和输出。Windows 应用程序创建一个窗口并对窗口有一些主要的存取功能。应用程序和 Windows 共同分担管理应用程序窗口的责任。Windows 负责管理大小、位置及应用程序窗口的组成成分；应用程序则主要负责管理应用程序窗口的工作区域的大小。应用程序窗口包括了下列部分或全部成分：

- 窗口边界
- 工作区大小
- 控制菜单框
- 控制菜单
- 水平滚动条
- 最大框
- 菜单条
- 最小框
- 标题栏
- 垂直滚动条

图 1.2 所示为一个应用程序笔记本窗口，它包括了上述所列的所有成分。下面将对一典型的应用程序窗口的每一个成分进行说明。

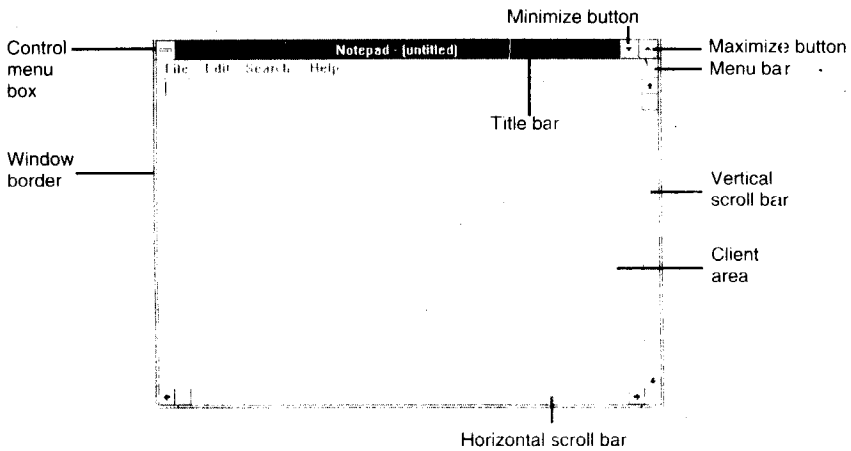

图 1.2　包含窗口标准成分的 Notepad 窗口

1.4.1　窗口边界

窗口边界包围着一个应用程序窗口的外沿，它包括三个基本元素集。第一个元素集是窗口的四个角，这四个窗口角可使你同时变化窗口的水平和垂直方向的大小；第二个元素集是边界的垂直边，它使你能改变水平方向的大小；第三个元素集是边界的水平边，它能使你改变窗口的垂直方向的大小。

1.4.2　用户区

用户区(工作区)是没有被菜单条、滚动条、边界或其它成分所占据的窗口的实际部分。应用程序使用用户区域作为它的工作区。应用程序维护用户区域，而 Windows 则维护窗口的位置、大小及应用程序窗口元素。

1.4.3　控制菜单

控制菜单框位于应用程序窗口的左上角，它提供了对控制菜单的存取。

通过控制菜单选项，可以恢复、移动改变大小、窗口最大化、窗口最小化或关闭一个应用程序窗口。控制菜单也叫系统菜单，当用户不选择使用鼠标或者没有鼠标时，系统菜单提供了对应用程序窗口的基本操作。图 1.3 是 Notepad 窗口的控制菜单。

1.4.4　水平滚动条

有了水平滚动条，可以移过那些比用户区域更大的文档或图像。水平滚动条包含三个元素，其一是左箭头，位于滚动条的左端，按动它，就可显示边界左边的文档或图像部分；其二是右箭头，位于滚动条的右端，按动它，就可显示边界右边的文档或图像部分；第三个元素则指示了当前可视区域的位置，滚动条的这个方块标识了相对于文件或图像的最左方或最右方的当前视区位置。

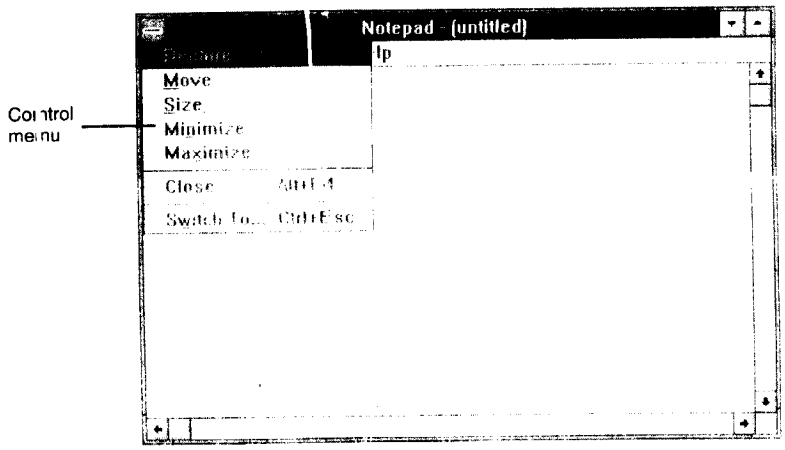

图 1.3 Notepad 窗口的控制菜单

1.4.5 最大化按钮

最大化按钮可将应用程序窗口充满整个屏幕。当按下最大化按钮，它即由恢复钮所替代。可以按动此恢复钮将窗口恢复成原来的大小。当窗口恢复成原来的大小时，恢复钮又为最大化按钮所替代。

1.4.6 菜单条

菜单条列出了应用程序提供给用户的菜单。文件、编辑及帮助菜单对于大多数应用程序来说是共同的，应用程序可以为用户定制可使用的菜单的数目及类型。

1.4.7 最小化按钮

最小化按钮将应用程序窗口缩小为一个图标。当你按动此钮时，应用程序窗口最小化(缩小为一个图标)，对此图标连按两下，窗口将恢复为原先的大小。

1.4.8 标题条

标题条列出了应用程序的名字。标题条可以通过颜色或亮度来区分是活动窗口的标题条呢，还是不活动窗口(暂不使用的窗口)的标题条。

1.4.9 垂直滚动条

通过使用垂直滚动条，可以移过那些比用户区域大的文件或图像。垂直滚动条也包括三个元素。其一是位于滚动条上部的向上的箭头，按动它，可以显示窗口边界上部的文件或图像；其二是位于滚动条下部的向下的箭头，按动它，可以显示窗口边界下部的文件或图像；其三便是指示出当前视区的位置的翻动块，滚动条中的此方块可以指示出相对于文件或图像的最下方和最上方的当前视区的相对位置。

1.5 Windows 函数

Windows 函数是 Windows 应用程序的核心。大多数用 C 或 C ++写的 MS-DOS 应用程序都使用可与 MS-DOS 接口的编译器的运行库函数，而这些函数主要是为面向运行于 80x86 体系上的 MS-DOS 应用程序开发的。使用 Windows 函数能利用 Windows 的功能以及它的设备独立性，这对开发 Windows 应用程序是很有好处的。

Windows 函数的范围是很广的。第 13 章和附录 A 提供了关于每一个函数的附加参考信息。你应略览一下第 13 章以获得关于 Windows 函数数目的大致印象。当然，不用担心，在这许多的程序设计库中，你会很快发现你需要熟悉并将反复使用的一个函数子集。

1.6 Windows 消息

Windows 应用程序利用 Windows 消息来与其它 Windows 应用程序及 Windows 系统进行通信。由于 Windows 应用程序是消息或事件驱动的，因此懂得 Windows 消息如何工作是重要的。第 14 章和附录 B 提供了对每一个 Windows 消息的参考信息并解释了 Windows 消息的使用。可略览一下第 14 章以获得关于 Windows 消息的大致数目的印象。

1.7 匈牙利记法

匈牙利记法是一种创建变量名的方法，由 Windows 程序员共同使用。匈牙利记法以 Microsoft 公司的程序员 Charles Simonyi 的国籍命名，并为 Microsoft 公司广泛使用于应用程序和操作系统软件中。

匈牙利记法的变量名中，以小写字母作前缀用来表示变量的数据类型，变量名的其余部分则描述了变量的功能。例如：

```
nCharacterCounter
```

说明此变量是一个整数(n)，并表示一个字符计数(CharacterCounter)。

第 13 和第 14 章所列出的 Windows 函数都在变量和参数名字中使用了表 1.1 中列出的前缀。

表 1.1 Windows 变量的公用前缀

前 缀	含 义
b	布尔型(非零为真；0 为假)
c	字符型(一个字节值)
dw	32 位无符号长整数
f	合并成一个 16 位整数的标志位
h	16 位句柄
l	32 位长整数
lp	32 位长指针
n	16 位短整数

（续表）

前　缀	含　义
p	16 位短指针
pt	组装成一个无符号 32 位整数的坐标对(x 和 y)
rgb	组装成一个 32 位整数的 RGB 颜色值
w	16 位无符号短整数

1.8　句柄(Handles)

句柄(Handles)是整个 Windows 编程的基础，因此，必须准确地理解句柄指的是什么。一个句柄是指 Windows 使用的一个唯一的整数值，用于标识应用程序中的一个对象，诸如一个窗口、实例、菜单、存储器、输出设备、控制或文件等等。例如，在模块定义文件中(这将在后面解释)，将定义菜单资源，并将资源和句柄相联系。比如，一个应用程序的菜单条中的第一个菜单的第一个菜单项可能被赋以 100 作为句柄，第二个菜单项被赋以 101 作为句柄。在应用程序的源代码中，这些菜单项便由句柄 100 和 101 来标识，Windows 应用程序往往只能存取句柄，而不能存取句柄所代表的实际数据。Windows 控制可存取数据因而能在多任务环境中保护数据。

1.9　包含文件

Borland C ++提供了 WINDOWS. H 这个包含文件，因而可以在自己的 C 或 C ++程序中存取不同的 Windows 函数和消息。WINDOWS. H 包含了 Windows 常量、变量、数据结构和函数定义，所有的 Windows 应用程序均应在源代码中包含 WINDOWS. H 文件。

下一章介绍使用传统的 Windows 编程技术进行 Windows 应用程序的编程的基础。你可以懂得一些事件驱动技术的概念，并领略一个标准窗口的开发过程。而本书的第一部分的余下各章将阐述 Windows 应用程序的各种编程技术。

第 2 章　Windows 编程基础

Windows 编程对于初学者来说，确实是件令人头痛的事，即使对于大多数能熟练用 C 语言进行传统的顺序程序设计的程序员来说，对 Windows 程序设计所带来的挑战也并不一定有足够的精神准备。许多程序员在进行 Windows 编程时往往会有如下的态度：

- “Windows 程序与一个 C 程序并没有多大区别”
- “我用 C 编程已好几年了，因此我所需做的只是熟悉 Windows 函数即可！”
- “Windows？C？都是一回事！”

遗憾的是，正是这些态度，使 Windows 初学者认为 Windows 编程很困难。

学习 Windows 编程本身并不比学懂任何结构化程序语言的来龙去脉困难多少——你只需学习基本的代码结构和程序设计方法。对大多数程序员来说，学习 Windows 程序设计的最大障碍在于必须打破传统的程序设计方法并全心地接受 Windows 程序设计思想。如果以一种开放的思维去学习 Windows 编程，将会很快地适应这个新的程序设计的概念和方法。

本章将带你对 Windows 编程进行一次旋风式旅行。本章以事件驱动程序设计入门作为开始，介绍了用 Borland C++进行 Windows 应用程序编程的基本概念和方法，并以这些概念和方法在一个基本的 Windows 程序中的实证作为结束。

2.1　事件驱动程序设计

MS-DOS 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、明显的过程及一个明显的结束，因此，程序能直接控制程序事件或过程的顺序。

Windows 程序设计方法与 MS-DOS 程序设计方法的不同就在于 Windows 程序是事件驱动的。事件驱动的程序不由事件的顺序来控制，而是由事件的发生来控制。下面的例子能较快地将顺序的、过程驱动的程序和事件驱动的程序的差别体现出来。

假设你要写一个程序来计算一个学期中进行了三次测验后一个班的平均成绩。你也许会按如下的顺序来决定产生这个平均数的步骤：

1. 输入学生姓名；
2. 输入第一次测验的分数；
3. 输入第二次测验的分数；
4. 输入第三次测验的分数；
5. 计算并显示平均分。

在一个顺序的、过程驱动的程序中，逻辑流程图可能如图 2.1 所示。程序在屏幕上显示一个提示符，请你输入姓名，然后下一屏幕又提示你输入第一次测验的成绩，再下一屏提示你输入第二次、第三次测验成绩。你输入了三次测验成绩后，则会出现一屏以显示计算出的平均成绩。这个步骤是逻辑的，而且遵循事件的顺序结构。然而，使用该程序，你必须遵循能

设计的过程，而不可能让你有额外的空间在进入第 5 步以后再去改变第 3 步的成绩。

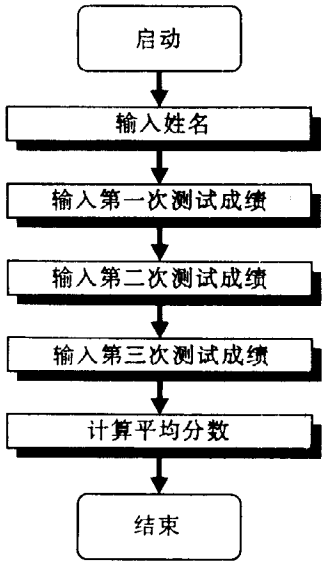

图 2.1　顺序的、过程驱动的方法

虽然在顺序的、过程驱动的程序中也有许多处理异常的方法，但是，即使这样的异常处理也仍然是顺序的、过程驱动的结构。事件驱动程序的设计则以一种非顺序的方式处理事件，从而回避了顺序的、过程驱动的方法。

事件驱动程序设计是围绕着消息的产生与处理而展开的。一条消息是关于发生的事件的信息。例如，无论何时，一个键或鼠标按钮被按下，就发出了一个消息，而当松开此钮时，另一个消息又发出了。作为一个 Windows 程序员，你工作的最大限度，也不过是对你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 消息是事件驱动的，消息是不会以任何预定义顺序出现的。

上述顺序的、过程驱动的例子以事件驱动方法可很容易地实现。图 2.2 展示了以一个事

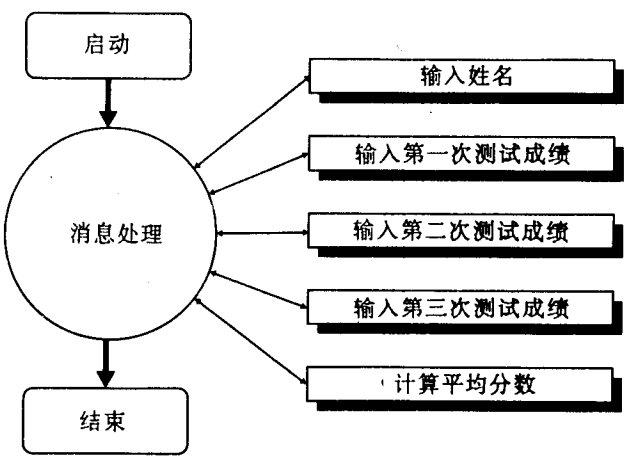

图 2.2　事件驱动方法

件驱动的结构实现该应用程序的方法。顺序的、过程驱动的程序中的全部功能仍然存在；然而，用户并不需要按顺序按步骤地计算成绩。用户可以在不同的屏幕上跳来跳去，在每一级上增加或修改数据。例如，在事件驱动应用程序中，你可以进入第三次测验的成绩区而无需先进入第一和第二次测验的成绩区。而在过程驱动的例子中，你是不能这样做的。事件驱动程序设计方法提供了许多的便利，对于那些需要大范围的用户干预的应用程序来说，更是很有用处的。

2.2 WinMain 函数

WinMain 函数是所有 Windows 应用程序的入口点，WinMain 函数有三个基本部分：过程说明，程序初始化及消息循环。WinMain 函数的一般形式如下：

```
int PASCAL WinMain (HANDLE hInstance,HANDLE hPrevInstance,
                    LPSTR lpszCmdParam,int nCmdShow)

{
HWND hWnd;
MSG Message;
WNDCLASS WndClass;

if (! hPrevInstance)
      {
      WndClass.cbClsExtra=0;
      WndClass.cbWndExtra=0;
      WndClass.hbrBackground=GetStockObject(WHITE_BRUSH);
      WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
      WndClass.hIcon=LoadIcon(NULL,"END");
      WndClass.hInstance=hInstance;
      WndClass.lpfnWndProc=WndProc;
      WndClass.lpszClassName="WIN_ONE";
      WndClass.lpszMenuName=NULL;
      WndClass.style=CS_HREDRAW|CS_VREDRAW;

      RegisterClass(&WndClass);
      }

hWnd=CreateWindow("WIN_ONE",          /* class name */
        "Fundamental Window",         /* Caption */
        WS_OVERLAPPEDWINDOW,          /* Style */
        CW_USEDEFAULT,                /* x position */
        0,                            /* y position */
        CW_USEDEFAULT,                /* cx_size */
```

```
        0,                              /* cy_size */
        NULL,                           /* Parent window */
        NULL,                           /* Menu */
        hInstance,                      /* Program Instance */
        NULL);                          /* Parameters */

    ShowWindow(hWnd,nCmdShow);
    while (GetMessage (&Message,0,0,0))
        {
        TranslateMessage(&Message);
        DispatchMessage(&Message);
        }
    return Message.wParam;
```

WinMain 函数的过程说明如下：

```
int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTRlpszCmdLine, int cmdShow);
```

hInstance 是一程序的句柄，hPrevInstance 是另一个与 hInstance 相关的程序句柄。如果程序共享同一个模块名，则称程序是相关的。lpszCmdLine 定义程序命令行参数。cmdShow 定义第一次打开的主窗口的状态。

程序初始化是 WinMain 函数的一部分。在本章后面的例子中，WinMain 函数的初始化部分包含了一个窗口的创建。在创建窗口过程中，必须定义一个窗口类。WinMain 函数初始化部分的第一部分定义了各种 WndClass 数据结构。在初始化代码中，你定义用于窗口的光标、填充窗口背景的刷子、窗口过程名、窗口类风格、窗口主菜单以及类名。下面是这个例子的初始化部分：

```
if (! hPrevInstance)
    {
    WndClass.cbClsExtra=0;
    WndClass.cbWndExtra=0;
    WndClass.hbrBackground=GetStockObject(WHITE_BRUSH);
    WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
    WndClass.hIcon=LoadIcon(NULL,"END");
    WndClass.hInstance=hInstance;
    WndClass.lpfnWndProc=WndProc;
    WndClass.lpszClassName="WIN_ONE";
    WndClass.lpszMenuName=NULL;
    WndClass.style=CS_HREDRAW|CS_VREDRAW;

    RegisterClass(&WndClass);
    }

hWnd=CreateWindow("WIN_ONE",        /* class name */
      "Fundamental Window",         /* Caption */
      WS_OVERLAPPEDWINDOW,          /* Style */
      CW_USEDEFAULT,                /* x position */
      0,                            /* y position */
      CW_USEDEFAULT,                /* cx size */
      0,                            /* cy size */
```