第一部分 基本方法

第1章 入 门

在多窗口应用程序中，用户可以在他的数据天地中任意漫游，在任何时刻，只需用鼠标点取就可以从一个信息窗口切换到另一个信息窗口。

在整本书中我们都用 Contact Manager（联络管理）应用程序作为范例（参阅引言中的图 I.1～I.16）。用户可以在一个窗口中列出公司清单，选择一家公司然后打开另一个窗口列出联络表，在第三个窗口中列出另一家公司的商务活动时间表。

用户可以在那个窗口内编辑当前正在查看的记录，也可以在两个或更多个窗口中编辑公司职员的有关记录以及日程安排时间表的有关记录。用户可以同时实施多项操作。

用户可以在数据库的任何一处执行一系列数据操作——如公司的花名册，通讯录，任务文件，联络团体清单（都称之为事件：讨论会，特殊集会，圣诞卡清单，等等）。

采用事件驱动程序设计，我们可以把控制权转交给应用程序，使它们更易于使用。

在本章中，我们首先讨论多窗口程序设计所带来的特殊问题，并介绍如何在程序模块中解决这些问题。

1.1 入 门

在多窗口环境中，用户可以采用不同于单窗口系统的方法执行各种操作，他可以挂起正在执行的操作，即在这个过程中他可以同时执行多项工作，然后返回被挂起的操作。

被挂起的操作并没有终止，因而可以继续运行（即重新投入运行），这就是在多窗口应用程序中任何操作都好像在同时执行的原因。

因为用户在启动另一个操作时并不需要结束当前操作，所以我们需要采用不同的方法通知应用程序：用户在哪个窗口。ESC 键（终止一个操作）或菜单选择（启动一个操作）仍然有效，但是只使用它们是不够的，因为它们不考虑用户可以切换的并发窗口。这就是多窗口系统和我们以前编写的单窗口系统最大的差别。

另一种差别是：许多任务好像在同时执行，这些并发的操作还可以互相影响。例如，如果我们在窗口中修改记录（见图 1.1 和 1.2），另一个窗口的信息和前面正在修改的记录相关，那么就必须关闭这个窗口（见图 1.3）。

应用程序也可以更新这些相关的窗口，而不关闭它们（我们甚至可以让用户决定他希望我们的应用程序做些什么）。参阅图 1.4 和 1.5。

如果让窗口相互影响（这完全由应用程序设计者决定），那么应用程序必须处理更新一个窗口可能对其他窗口产生的影响。

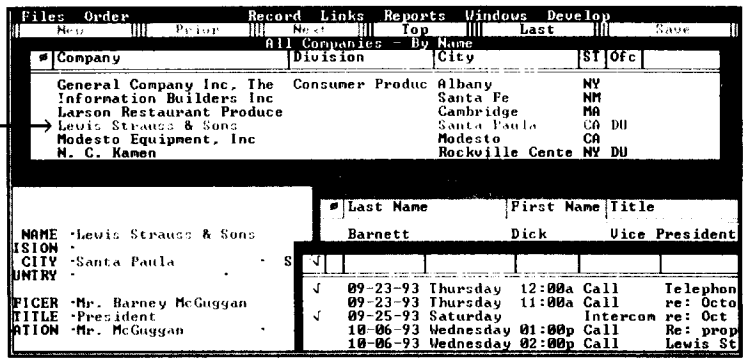

图 1.1 这些窗口均与 Companies 文件的当前记录相关

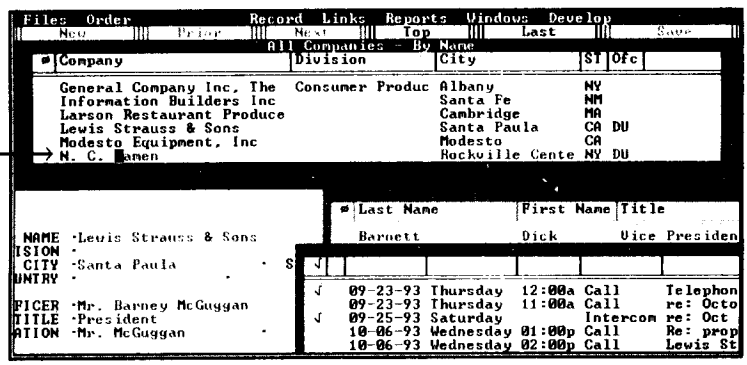

图 1.2 现在对另一个公司记录进行数据处理

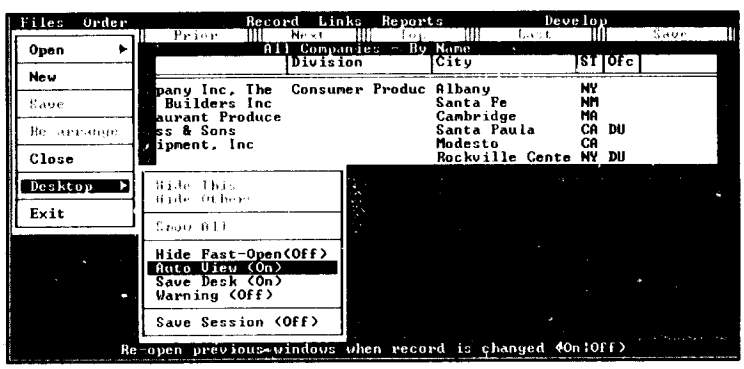

图 1.3 另一个窗口内的信息不再有效

还存在一点差别。如果我们允许以任何次序打开窗口，然后再进入任意一个窗口，那么用户打开窗口的次序就成为一条重要的信息(见图 1.6 和 1.7)。

事件驱动应用程序必须以不同于常规菜单驱动应用程序的方法对处理的状态(即计算机上正在执行的操作)作出响应，常规菜单驱动的应用程序只允许一次处理一部分数据。

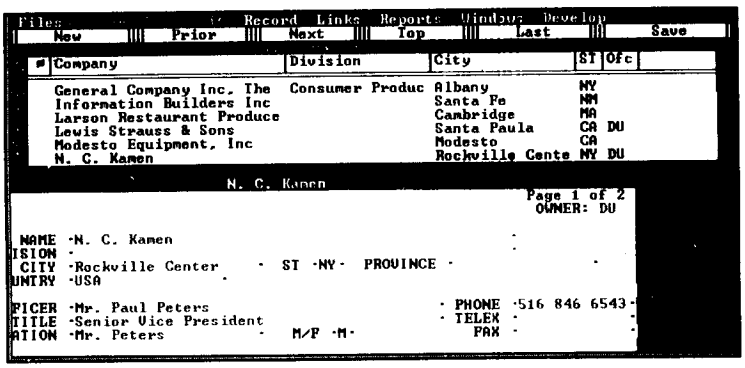

图 1.4 应用程序可以自动打开一个新的编辑窗口

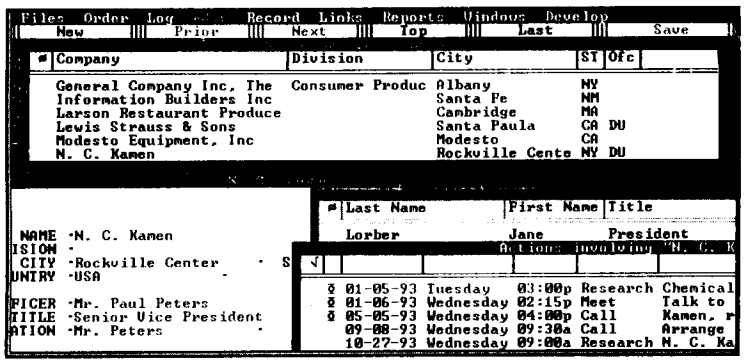

图 1.5 确定要显示的其他记录

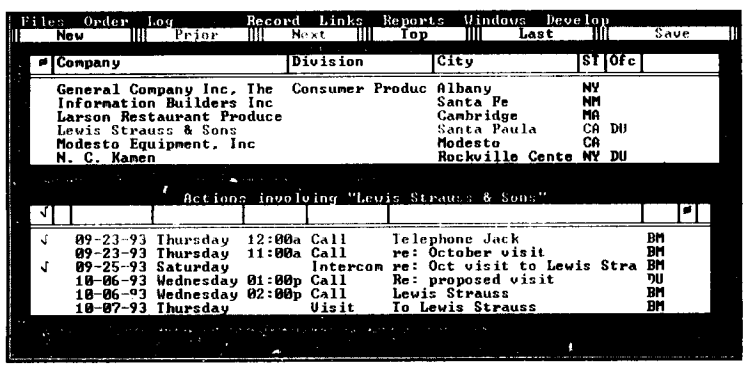

图 1.6 首先查看 Companies 清单，然后查看
与当前公司相关的 Actions 记录

要查看计算机这种动态过程，可观察 FoxPro 的系统菜单子项和子菜单选项的开关状态(ON 和 OFF)，这些选项只在适当的时候提供各种服务，也可以在进行窗口切换时查看窗口标题说明。

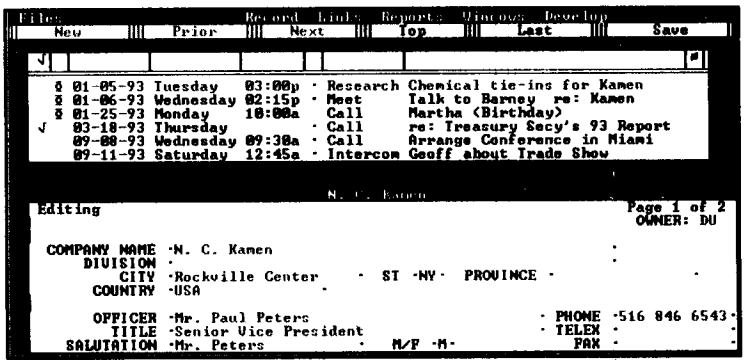

图 1.7 首先打开 Actions 文件；只能看到 Companies 文件中的一条记录(和当前动作有关的记录)

FoxPro 所做的一些工作同我们必须在事件驱动应用程序中实现的工作相同：查询计算机正在进行的操作并确定用户正在进行何种运算，以及用户下一步选择的运算要求(当程序在交互方式下运行时)。

1.1.1 决定事件驱动接口程序的功能

事件驱动接口程序运行的方式很多，这正是必需在连接几个窗口和菜单时做大量工作的原因所在——甚至需要使用 FoxPro 强大的工具来帮助实现这类连接工作。

在开始编写事件驱动系统的程序之前，需要有一个设计思想(design idea)，制定一个规划，以便确定目标程序界面的外观，它将提供何种服务，以及用户将如何与之“交谈”。正如数据库及其应用程序的编制需要一个系统设计，这个事件驱动应用程序的接口程序同样有这种要求。

程序的编制者可能只想让程序的用户做一些简单的操作：例如，在一个窗口内编辑文件中的记录，并显示其他窗口中相关文件内的记录。这就是所谓的设计思想。

然而，也可以让用户做更多的工作。这取决于程序设计者的设计思想，由它决定接口程序(界面)所具有的功能，这有助于接口程序具体实现。

程序员确定了接口程序的工作方式之后，就可以选择最佳的方法来管理信息。

1.1.2 设计思想的一个例子

要了解设计思想的真实含义，我们提供了一个范例。设计思想总的原则是：把数据库应用程序看作是用户处理数据的工具。

首先，信息应以更有意义的方法来存取——把何时显示信息，显示什么信息的控制权交给用户(而不是程序员)。究竟什么方法更有意义呢——即我们看待任何信息组合的观点。我们把那些信息组合起来的方法——也就是我们组织它们的着手点——这种变化通常是时刻发生的。

举一个简单的例子，“Call Rich，Friday，about meeting”，它由四种信息组成：打电话，打电话的对象，时间，什么事情。可以用四种不同的方法考虑它或想起它，这四个不同的方面

是:想起必须打的电话;想起打电话的对象,Rich;想起应该在星期五做什么;或者想起一个特殊的会议。

信息相关的这种方式就是数据库应用程序的实质,也正是数据库文件构造的基础!

然而,这并不是我们通常让用户在信息中查寻的方法。

无论在什么时候设计一个应用程序,为了更有效地存储和检索数据,我们要做的第一件事情是把不同的数据划分到不同的文件中进行管理。应用程序趋向于这种数据分段,如果用户只能处理一类信息,那么一次处理一个数据段(或者一次处理预先确定的一组分段信息)。

我们所面临的问题是我们并不总是一次考虑一个事件。我们思维的方式既不是有层次的,也不是直线式的,以前的系统迫使我们以这种方式进行思考和数据处理:先从这里开始,关闭我们当前进行的数据处理,再进行另一项处理。但是我们的思维方式具有循环特性,一种思维过程嵌套着其他思维过程,双反馈,又进行分支——而不必完成它们。这也正是我们最富有效率的原因:在数据库中同时处理多种数据。

规格化数据对程序员来说很理想,但是对于程序的用户并不总是很方便。我们习惯于用文件和记录进行数据处理,而连续的信息则更自然、更常用。

有一种方法既可以满足程序员的需要,也可以达到用户的要求。多窗口事件驱动接口程序允许我们直观地重构全体规格化信息,我们可以按程序员的理解去构造它们,而我们的应用程序又将现实世界的非规格化数据展现给用户。

多窗口在把规格化信息转换回非规格化信息的过程中"扮演"重要角色,它们使用户毫不费力地重构用户自己的数据形式。而在程序员脑海里浮现的是:同时打开多个窗口,并在它们之间不断移动。

在 FoxPro 中,我们几乎可以像我们联想信息那样方便地访问信息,这种方法允许我们把计算机当作一个焦点区域(field of attention),用户可以用他感兴趣的各种观点和信息改变的组合形式来控制它。

有时,计算机上可能含有有序项目表格,它具有极佳的性能,以至于它成为数据库中必备的工作平台(thinking space)。

当我们设计一个工作平台时,也可能要删去以往应用程序中的一些熟悉的操作,不必在用户启动另一项工作时关闭当前使用的信息,即不必在回到任意一个工作起点前,按 Esc,Esc,Esc(退出键)。

在各个数据处理任务之间进行切换的繁琐工作对程序员来说(即进行编码)易如反掌,但对用户来说就不是这样。程序员可以让应用程序根据用户选用的路径回退或者继续切换过程。Windows,鼠标,以及事件驱动程序也为我们提供了相应的服务。

最后值得一提的是:任何数据库应用程序,无论它的功能如何,都是杰出的列表管理器(或列表生成器),因而我想加入一个 WYSIWYG(所见即所得)的报表功能,它提供用户当前窗口的基本报告。

我使用 FoxPro 的事件驱动程序设计功能的目的,也正是我的设计思想。当我考虑用户如何使用我的程序工作时,我知道接口程序需要一组功能,并知道如何将它们组织在一起:

- 允许用户在需要的地方移动窗口。
- 允许用户在任何时候打开和关闭窗口。
- 允许自动更新窗口或自动关闭它们——我让用户自己决定。

- 允许用户在需要时创建一组窗口。
- 禁止在计算机上出现空窗口。
- 任何时候总是从启动 BROWSE 窗口开始，可以只含有 BROWSE 窗口。
- 限制编辑 READ 窗口(使有效性更易于实现，并使得 BROWSE 更快)。
- 使 READ 窗口变为非 MODAL(模态)状态，允许用户保存已准备好的编辑工作。
- 把按钮加到工具栏菜单上。
- 菜单选项随着计算机处理状态的变化而变化。
- 允许从任意窗口直接打印报表。
- 允许用户保存不同的会话过程。

值得注意的是，上述特点没有一个是事件驱动程序设计所固有的。从本质上来说，上述特点的大多数超出了事件驱动程序设计的基本要求，甚至超过了多窗口程序设计的功能。

但是程序员可以在应用程序中加入上述功能，使之在事件驱动程序的基础上增加一层功能。

1.2 采用一种通用的方法

本书介绍设计事件驱动系统的一般方法。我们需要一种管理许多不同信息窗口的方法，应用程序应该能控制它们——但要求这些窗口协同工作。在设计阶段如果为一些功能预留了空间，删去这些功能比重新设计整个接口程序(以便加入遗漏的功能)容易得多。

任何综合的多窗口程序设计方法至少应该实现下列功能：

- 对应用程序可能遇到的适当内存限制产生响应——还应对遇到的工作区限制产生响应。
- 提供稳定的性能，无论必须管理多少文件和窗口。

最后一项要求为：任何多窗口核心程序(用于管理窗口，菜单和事件)应该可以被应用程序共用，它应该易于理解，易于调试，易于维护，易于扩充。

本书重点讲述程序编写方法，它有助于程序员实现自己的设计思想。

1.2.1 在应用程序中加入事件驱动功能

当读者正在决定自己的设计思想时，即使想编写常规的应用程序，仍可以使用事件驱动程序设计方法进行设计。往这些应用程序中加入必不可少的几种事件驱动功能，就可以使之更易于使用，并使之工作起来更像其他的商业软件。

当读者真的需要编写更复杂的系统时，事件驱动功能也为读者编写优秀的多窗口应用程序提供捷径。

1.2.2 了解用户需求

“用户现在想做什么呢?”

这是事件驱动系统必须回答的关键问题。为用户处理不同的事务提供的替代方法越多——例如，在另一个窗口中处理另一部分数据——为了告诉应用程序计算机上正在进行的

操作，事件驱动程序需要的信息就越多。

我喜欢把事件驱动系统编制的方法视作：程序员对一系列提问的应答。

每个窗口代表了一些窗口进程(BROWSE 或 READ)和正在该窗口处理的文件。进程(process)是程序中正在运行的一段代码。当我们知道用户现在想进行什么操作时——或者他下一步要执行什么运算时，我们就知道需要运行的例程。这就要求我们回答下面的问题：

窗口中正在进行什么处理？

什么文件正在处理？

正在浏览记录——还是在编辑记录？

允许用户执行何种操作？

用户可以修改信息吗？

用户可以使用哪些菜单选项？

这些问题我们是很熟悉的：运行以前的程序——非事件驱动程序：BROWSE；发出一些@SAY/GET 命令，然后执行 READ；显示一列菜单选项。

我们甚至要回答下面两个问题，至少在一些以往的应用程序中，例如一些允许用户在不同文件中移动相关记录的应用程序：

用户如何到达当前工作区？

用户可以到达哪些工作区？

允许用户回绕到进程链中以往的工作点，这是多窗口应用程序的另一个问题：

用户在此处的操作对其他工作区的信息有什么影响？

更新的解决办法是多窗口应用程序让用户重新进入信息处理链中相应的处理过程。这在常规的应用程序中是无法实现的，因为用户回退到较高层的唯一办法只有关闭所有低层处理过程。在多窗口应用程序中，用户可以中止较新的(或相关的)进程，或者切换进程，甚至可以从原来的进程中扩展新的分支进程，而无需在窗口的进程中实现它们。

举例来说明，用户可能正在处理 Action 窗口中的一条记录(这一业务可能涉及一特定公司)，他可以打开公司的 Contact 列表，尽管没有打开 Companies 窗口。这就是我们可以实现的一种信息查询(见图 1.8)。

即使不在 Contact 窗口中，仍可以存取 Event a Contact 列表中包含的信息(见图 1.9 和图 1.10)。

也可以在当前窗口内修改记录，来达到修改另一个窗口信息的目的(见图 1.11 和图 1.12)。

1.2.3 多窗口管理

管理窗口意味着管理各个窗口的有关信息以及窗口中显示的文件，还要管理计算机上发生的一切。

哪种信息是我们必须用之来回答事件驱动程序设计的问题呢？我们又如何解决这些问题呢？

我们必须了解的第一件事情就是用户何时执行其他操作。用户执行下列操作之一都会

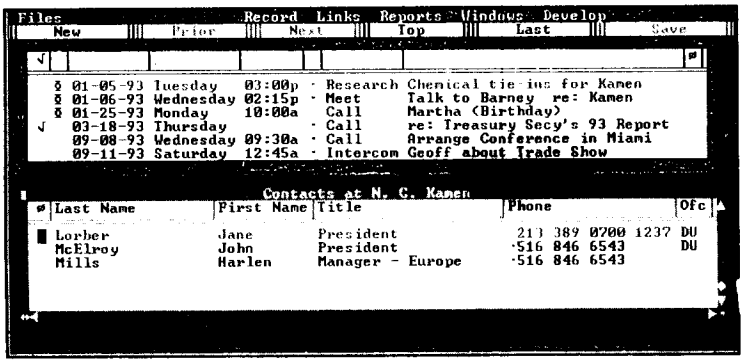

图 1.8 显示和当前业务相关的公司的业务表及其联络表

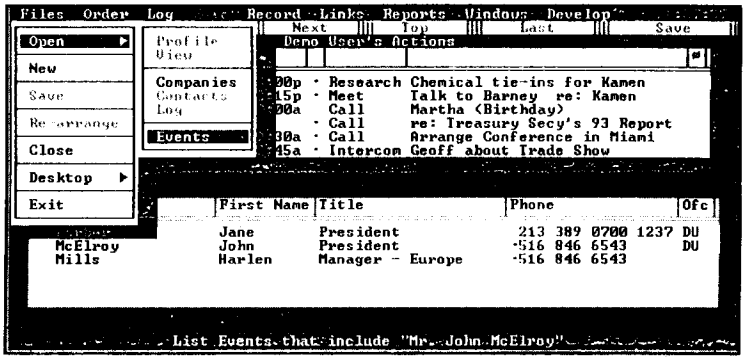

图 1.9 尽管在 Log 窗口中,我们也可以打开与 Contact 相关的窗口

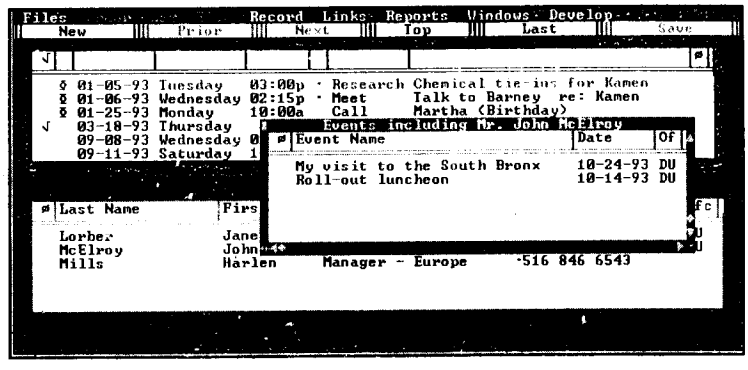

图 1.10 这些 Events(事件)和 Contact(联络)文件中的当前记录相关

出现这个问题:

- 在 BROWSE 窗口中列记录。
- 在 READ 窗口中编辑记录。

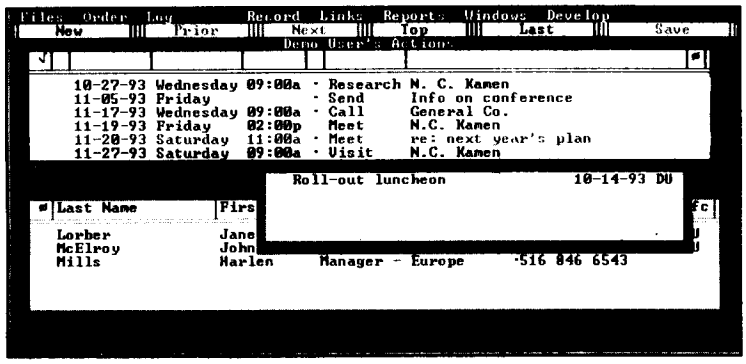

图 1.11 我们已转换到另一条 Action 记录，它和另一家公司相关

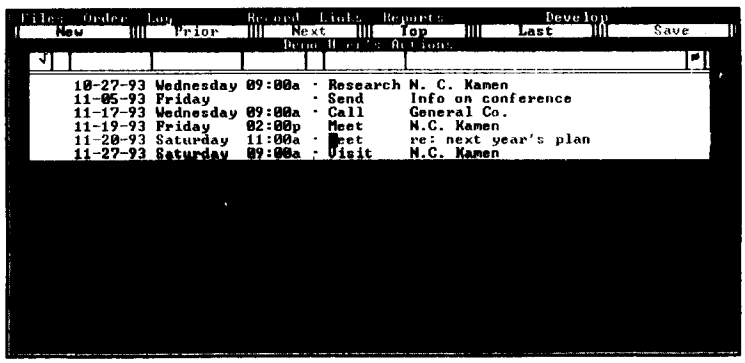

图 1.12 和前面 Action 对应公司相关的全部操作不再有效

- 选择菜单或者弹出子菜单。

当 FoxPro 正在等待键盘输入时，即处于等待状态。

当用户处于任意一种等待状态时，他可以通过如下操作通知事件驱动应用程序他还想执行的操作：

- 关闭当前窗口，或者用鼠标单击另一个窗口。
- 敲入任意一程序定义的“热”键(一个 ON KEY LABEL 键)。
- 选择菜单子项或选择菜单按钮。

这些操作都是事件。我们可以用如下方法确定用户退出等待状态：

- 用户当前在什么窗口？
- 用户如何离开以前的等待状态？
- 用户选择哪个菜单子项或按钮？

上述问题均为事件驱动系统的本源问题提供了部分解答：

用户当前想执行什么操作？

所有其他问题均源于此问题。

1.3 使用基于消息的方法

事件启动之后，我们赋予它一个名字，描述用户执行的操作。我们可以把这些事件名当作消息(message)保存在一个数组中，将来作为引用它们的事件堆栈(Event Stack)。

为了阅读程序，将这个数组命名为：laEvents (Local array Events，局部数组事件)。

```
laEvents[1]="NEW"… or "MENU"… or "SWAPWIN" …
```

无论栈中含有多少条消息，我们都可以把这个栈传送到 Event Loop 过程(事件驱动程序的主管理器)。

事件循环把任何事件的处理分派给专用的 Event-Handler(事件处理)例程。

Event-Handler 例程执行具体的处理过程，如打开和关闭窗口，从一个窗口移至另一个窗口，增加新记录，打印报告。每个事件处理程序只执行它指定的事件。当它结束时，从栈中删去标识它的消息。

在基于消息的多窗口管理方法中——READ 窗口和 BROWSE 窗口——每个用户活动(每个事件)都是中介的，在真正执行事件处理程序之前，要经过一系列的消息传送。当事件循环发现事件堆栈的当前队列中没有消息时，就返回三种基本操作之一：浏览，编辑，或选择。

事件循环的框架如下：

```
DO WHILE .t.
    IF(如果栈中有挂起的事件)
        1.调用事件处理程序处理栈底的事件消息。
        2.执行该事件要求的处理工作。
        3.从事件栈中删去事件消息。
    ELSE(当没有挂起的事件时)
        当用户浏览(BROWSE)当前文件中的记录或者编辑(READ)当前文件中的记录
        时，等待下一个事件发生。
        当事件发生时，标识它，然后压入栈中。
    ENDIF
ENDDO
```

大多数事件发生在用户正在 BROWSE 或 READ 窗口中工作时。有时事件是由菜单发出的，菜单是启动事件应用程序的另一种途径。

事件还可以以另一种方式产生。一些事件处理程序也可能产生新消息(它们把这些消息压入事件栈顶)，但它们是在完成自己的事件处理之后产生这些消息的。

1.4 信息管理

我们需要一种传送事件信息的方法，我们必须尽可能清晰地、高效地、可靠地传送这种

信息。这也正是我们使用明确的消息——专门用来传送这些消息的结构和程序(事件堆栈和正式的事件循环)——来处理用户信息的原因。

事件不是我们需要的唯一信息,消息也不是我们追踪用户活动的唯一途径。我们使用所熟悉的所有方法来传送信息:变量,SET RELATION 命令,以及索引 TAG 命令。这些方法还在提供信息和回答问题中扮演重要角色。

命名约定也提供信息,它们直接向我们说明所蕴含的意思。在这里只举一个例子:我们用文件名定义它窗口的名字,而窗口名则向我们指出处理该窗口的子程序,它们都蕴含了详细的说明。

用户触发事件之后,要执行许多操作。如果我们想开发多窗口系统,在应用程序中我们尚需回答许多问题。这些问题如下(在第 4 章末尾汇总了主要的多窗口程序设计方面的问题):

- 事件管理
  用户如何离开 BROWSE 或 READ 窗口?
  用户当前使用什么窗口?
- 菜单管理
  用户当前选择了什么菜单选项?
  用户已选择了什么菜单选项?
- 窗口管理
  计算机上正在执行什么?
  用户当前的窗口内正在执行什么?
  用户正在执行的操作对其他窗口有何影响?

上述管理程序模块均只是一组例行程序,每组负责管理不同的多窗口平台(它们共同组成事件驱动系统的核心程序)。除了某些窗口管理例程之外,这也是组织常规系统的完善方法。

在令人迷惑的多窗口后面,基于消息的系统其实在运行时同常规系统非常相似。现在每个窗口代表常规应用程序中完全分离和孤立的操作:列表(BROWSE)和编辑(READ)对应不同的进程,而且每个文件也对应彼此分离的进程。在多窗口系统中,它们实际仍是分离的和独立的,但它们可以是同一个过程,我们只不过是让用户以不同的方式去访问它们。因此,我们必须在给定的时间内追踪这些进程的多个副本。

1.5 保持清晰的信息流

如何同时处理几个窗口呢?

我们怎样才能用支持许多多窗口应用程序的设计思想来达到同时处理多窗口的目的呢?——即只需为多窗口系统设计必要的操作,然后在不同的应用程序中重复采用它。

这正是本节讨论的问题。我们采用如下分析的方法解决上述问题:

- 把事件驱动系统当作回答程序员一系列问题并解决这些问题的机制。这种方法对所有这样的系统都通用。

- 把这些问题归为不同的类:事件管理(Event Management),菜单管理(Menu Management),窗口管理(Window Management)。
- 把这些问题融于程序的子系统中(或模块中),由程序来求解所遇到的问题。

这种编码方法采用消息(这种消息是一种命名的事件),并使用了传送消息的事件堆栈和处理消息的事件循环。

事件本身只回答我们的部分问题。它们这样开始:消息通过事件驱动程序传送,"这是用户现在想实现的操作。要实现这一操作所必需进行的处理。"

如果这种分析方法和基于消息的编程方法(它非常近似于分析模型)具有明显的优点,可能只是在于它们采用了事件驱动多窗口系统所固有的方法来更好地观察和求解问题。这两种方法的共同点就是揭示问题的本质,为我们提供更多的控制。

用消息调用指定例程更易于实现如下操作:事件驱动程序正在处理什么事件,在何处(即程序)处理这个事件,这个事件为何发生,我们何时设计应用程序,何时运行(和调试)它们。使用消息使我们更易于控制处理事件的核心程序。程序员获得的事件驱动系统如何工作的控制越多,那么为用户实际使用事件驱动软件所提供的控制也越多。

难道这种消息传送和调用所有处理例程的方法!不会使基于消息的系统真正变慢吗?

不尽然!用这种方法管理多窗口软件不会导致明显的性能下降。它实际上速度奇快,FoxPro 并不只是一个检索快捷的软件。

另外,如果使用这种方法,就不必放弃 FoxPro 强大工具所具有的优点。

1.6 后续章节的内容

在第一部分后三章中,我们将介绍事件管理,菜单管理和窗口管理。更多地讲述每个管理模块,这些模块应该实现的功能,以及用户解决问题所需的各种信息,结构和程序设计技术。

后续章节的每节都包括了一些示例程序,主要用于说明一些最基本的概念。

要点: 本书附带的磁盘上有一个实例程序(SAMPLE.PRG),它是设计基于消息的事件驱动系统的恰当入门程序。这个程序的清单附在附录 A"一个基本系统"中。请参阅那节中有关安装和运行这个基本系统的说明。

第2章 事件管理

在事件驱动系统中用户通过“事件”来告诉系统他想转入其他任务，如切换到另一窗口，选择一个菜单项，或通过热键或按钮执行一个操作等。第1章介绍了如何把事件转换成消息及如何把消息发送到事件循环。在第1章里，消息被发送到了专门负责事件处理的过程中。

本章将介绍应用程序在事件确认过程中存在的问题，还将介绍一个基于消息的系统用来解决这些问题的信息和技术(见图2.1)。下面首先介绍如何编写事件发生程序，如何确定(或捕获)事件，及如何将事件发送到处理程序。

在BROWSE或READ窗口中启动事件由事件捕获函数确认并命名，该事件名存放在一个称为“事件栈”的数组中。菜单启动事件则被直接存放入事件栈，不必由事件捕获函数对其进行预先处理。

事件循环将把事件栈中接收到的事件消息传送到合适的事件处理程序中(在下图中用阴影块示出)。图中，“MENU”块和“WinEvents”块也是事件处理程序。

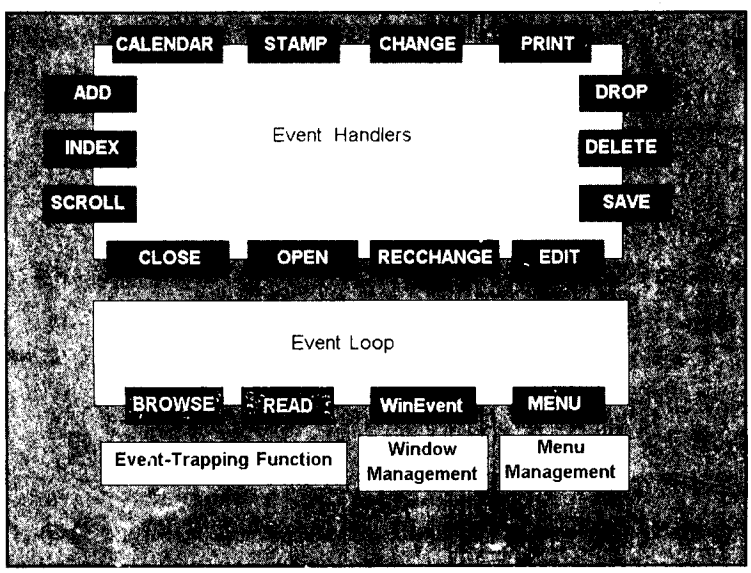

图2.1 基于消息的事件驱动系统的主要结构

下面是加入了第一个控制结构的事件循环：

```
DO WHILE .t.
    IF(如果栈中存在待处理事件)
```

注释

- 调用事件处理程序处理栈顶事件
- 执行该事件的请求操作
- 从栈中删除该事件消息

```
    ELSE(如果栈中无等待事件)
```

在等待下一事件发生的等待期间，用户可以浏览活动窗口中的记录列表或对窗中记录进行编辑(READ)。下一个事件发生时，应调用事件捕获函数确认该事件并将其存入事件栈中，然后将该事件返回发送到事件循环顶部。

```
    ENDIF
ENDDO
```

2.1 产生事件

大多数事件在 BROWSE 或 READ 窗口中发生。在 BROWSE 或 READ 窗口中，FoxPro 处于等待状态(事实上，FoxPro 的空循环正在轮询系统状态)。用户退出等待状态时，需要通过某种方式告知 FoxPro，而用户创建一个新事件时，也要通过另一种方式确认用户目的。

这两件工作可使用同样的技术完成，用户可通过下述几种方式来通知应用程序其意图：

- 关闭当前窗口或单击另一窗口。
- 按下程序中由 ON KEY LABEL 语句定义的热键。
- 选择一个菜单子项或按钮。

在介绍这种技术的工作原理前，先谈谈菜单和子菜单。事件也可以在菜单中或弹出式子菜单产生，但用户以不同方式处理这两种操作。当用户选择的是弹出式子菜单中的选项时，就已明确表述了其操作意图，所以无须再像处理窗口事件那样捕获菜单事件。

应用程序需要捕获的是当用户选择一个菜单子项时(单击一个菜单 PAD，或按下一个菜单热键)，该菜单子项能在屏幕上显示，这种事件称为菜单事件，它就是我们的捕获对象(菜单管理将在第 3 章中论述)。

当 BROWSE 或 READ 窗口事件发生时，应用程序需要掌握以下信息：

- 当前哪一个窗口处于激活状态？
- 用户通过何种方式退出等待状态？
- 用户选择了哪个菜单子项(PAD)或按钮？

这些信息用于确认事件类型与用户意图。

并非所有的用户动作都能被 FoxPro 立即确认。例如，如果在一个事件发生后发现另一个不同的活动窗口，应用程序并不能判定用户在保留前一活动窗口的前提下切换到了这个窗口，还是关闭了前一个活动窗口。FoxPro 需要更多的信息才能对此作出判断。

2.1.1 窗口转换事件

在多窗口应用程序中，许多事件的处理均与窗口转换有关，这包括切换窗口或关闭当前窗口。

FoxPro 通过将另一窗口变成活动窗口来对这两种操作迅速作出反应。这是 FoxPro 的基本功能，应用程序无须越俎代庖。

用户可以进行以下三种改变当前窗口的操作：

动作	事件	键变量
Esc 键	CLOSE	lcTopWin
Close 按钮	CLOSE	lcTopWin
单击窗口	SWAPWIN	lcTopWin

上表中，“动作”代表用户操作，“事件”是程序对用户动作的命名，用户通过这些动作关闭当前窗口或切换到另一个窗口。

如果程序中定义了 Alt+F1 键，则可获得另一种改变当前窗口的方法。改变当前窗口还可以选用窗口系统中的“Next Window”选项。这两种操作都可以产生 SWAPWIN 事件。

通过这些动作可以得到两条信息：对事件的确认信息和新的活动窗口名。后者被存入变量 lcTopWin。

事件捕获函数得到当前活动窗口名后就可以判断用户意图并调用适当的窗口处理过程。

2.1.2 热键事件

如果用户想转入其他任务，可以使用“热键”通知事件驱动应用程序。

可用 FoxPro 的 ON KEY LABEL 命令定义键或组合键（如 Ctrl+N，鼠标右键，Delete 键等）。这些指令称为存储方法，它们的作用是使其成为 FoxPro 的有机组成部分。

只要用 ON KEY LABEL 命令对键进行重新定义，就可以在任何时候改变这些方法。演示应用程序就是用这种方式来对用户按键或系统事件作出反应，以感知系统运行状态的变化。例如，切换到一个 BROWSE 窗口或一个处于受控状态的 READ 窗口。

将热键与等待状态联系起来后，应用程序还要将热键对应的指令转换成事件。一旦用户按下热键，为了了解用户意图，我们必须让应用程序强制用户退出当前等待状态。

如前所述，一旦用户改变了当前窗口，FoxPro 就会终止等待状态。应用程序中可以采用同样的机制，以便在用户按下热键时引起 FoxPro 的注意。用户也可以进行窗口转换操作，但这时将由应用程序启动硬件事件。

下面介绍其工作原理。用同样的方式定义所有的 ON KEY LABEL 键。例如，可用 ON KEY LABEL RIGHTMOUSE 命令告知应用程序用户将打开一个 READ 编辑窗口并对某个特定记录进行操作：

```
ON KEY LABEL RIGHTMOUSE;
    DO MakeEvent;
    WITH "EDWINDOW"
```

所有热键事件都调用同一例程 MakeEvent。下面是其流程：

```
* ! Which ON KEY LABEL key was hit?
PROCEDURE MakeEvent
PARAMETERS pcHotEvent
STORE "" TO lcHKey
IF called during a BROWSE
    ACTIVATE WINDOW wHotKey
ELSE
    KEYBOARD CHR(23) PLAIN
ENDIF
STORE pcHotEvent TO lcHKey
RETURN
```

这个热键处理程序完成两件任务：首先，使 FoxPro 离开当前等待状态。如果这个热键处理程序是由发生在 BROWSE 窗口中的一个动作调用的，则过程 MakeEvent 将激活一个“不可见”窗口。窗口 wHotKey 被定义为 AT ROM-1，系统将其置于屏幕顶行，被激活时不可见。

若在 Read 窗口调用 MakeEvent 过程，则“进入”Ctrl+W(键值为 CHR(23))，并迫使 FoxPro 离开 READ 等待状态。“PLAIN”使 FoxPro 忽略任何可能与 Ctrl+W 键组合相联的 ON KEY LABEL 命令。

FoxPro 采用与用户选择另一窗口或进入 Ctrl+W 键时同样的方式对这些模拟的硬件事件作出反应，即退出等待状态。

由 MakeEvent 执行的另一操作是将 ON KEY LABEL 定义为 PARAMETER 的事件名存放到变量 lcHKey 中。事件捕获函数需要这条信息来回答其关键问题“用户现在想做什么?”

下面是一组可能用到的典型热键：

动作	事件	键变量
Rightmouse	EDWINDOW	lcHKey
PgUp	PGUP	lcHKey
PgDn	PGDN	lcHKey

2.1.3 菜单事件与按钮菜单

用户也可以通过单击菜单子项(PAD)来告诉事件驱动应用程序他想转入其他任务。

菜单与子菜单是一个事件驱动界面的关键部件。它们是通向应用程序所有内部工作的大门。必须注意不能让应用程序中未设计的操作通过该入口(例如，对 Inventory 文件操作时，绝不应该选择 Mail Merge 选项)。

在一个多窗口桌面中，提供了用户能同时进行操作的多个窗口，用户应在对子菜单进行选择之前测试所有 POPUP 子菜单的选择，以查看哪些选项对用户当前的操作适用。为达到这一要求，在允许用户选择菜单之前应作两种准备工作。

首先，要处理在用户单击菜单 PAD 与显示相关 POPUP 子菜单之间可能发生的任何事件。在本章后面的 2.3.7 节中，将讨论如何处理 BROWSE 窗口的记录更换事件。更换当前记录是对系统状态的一个重要改变操作。