第28章 商 用 软 件

尽管商用软件不如教学软件或游戏那样生动有趣，但也可以用 Director制作。第2章“用Director制作演示”里的幻灯片演示等大多数演示作品可以归类到商用软件里。

一些高档的影片也可以处理和显示数据。本章将讲解如何用 Director制作简单的数据库程序、如何制作图表、如何制作调查问卷以及如何制作培训软件。

28.1 制作数据库软件

用Director制作数据库是很简单的。不过，要制作一个非编程人员也能够使用的交互性的数据库界面却是个棘手的问题。你可能已经注意到，属性列表就像是一个小小的数据库。如果创建一个含有几个属性列表的线性列表，实际上就是建立了一个数据库。请看下面的例子：

```
[[#name: "Gary", #title: "Chief Engineer", #company: "CleverMedia"] ┐
, [#name: "Bill", #title: "CEO", #company: "Microsoft"]]
```

列表里的属性名称代表项目名称，属性的值代表项目本身。线性列表里的一项为一个记录。这个数据库例子里有两个记录，每个记录里有三个项目。

28.1.1 主菜单屏幕

制作数据库程序也就是创建一些画面，用它们可以处理前面的例子那样的列表数据库。这种程序需要用一个全局变量来存储数据库。 gDatabase是一个不错的名字。还需要知道当前正在编辑的是哪一个数据库记录。这个信息可以存储在名为 gCurrentRecord的全局变量里。

主菜单屏幕使用户能够执行各种任务。图 28-1是一个主菜单屏幕。

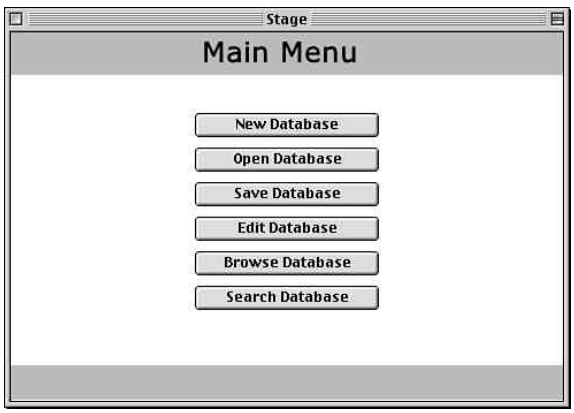

图28-1 主菜单屏幕里包含用于数据库的功能的所有按钮

在程序的开始，用户需要创建一个新数据库。点击 New Database可以实现这个目的。这个处理程序将把全局变量gDatabase清零，然后进入数据条目(entry）屏幕：

```
on newDatabase
```

```
  gDatabase = [newRecord()] -- database with one record
  go to frame "Entry"
  gCurrentRecord = 1
  showRecord
end
```

on newRecord函数返回一个新的、空白的记录。要想向数据库里添加记录或替换记录时，使用这个函数很方便。

```
on newRecord
  record = [:]
  addProp record, #item, " "
  addProp record, #id, " "
  addProp record, #category, " "
  addProp record, #number, " "
  addProp record, #price, 0.0
  addProp record, #description, " "
  return record
end
```

on newRecord函数定义了数据库所包含的项目。可以看到，这个数据库包含 6个项目。

28.1.2 数据库条目屏幕

接下去，我们需要一个数据库条目屏幕，这样用户能够输入新记录和修改旧记录。 on newDatabase处理程序以调用 on showRecord处理程序为结尾。这个处理程序把数据库条目里的各项内容显示在屏幕上：

```
on showRecord
  -- get current record
  record = gDatabase[gCurrentRecord]

  -- place fields into members on stage
  member("Entry - Item").text = record.item
  member("Entry - ID").text = record.id
  member("Entry - Category").text = record.category
  member("Entry - Number").text = record.number
  member("Entry - Price").text = string(record.price)
  member("Entry - Description").text = record.description

  -- show the record number
  member("Record Number").text = "Record"&&gCurrentRecord
end
```

图28-2就是数据库条目屏幕。

在这个数据库条目屏幕里，六个可编辑栏是处于激活状态的元素。屏幕里的其余的大部分元素都只是静态的文本。左上角是记录的编号，它是不可编辑的，它显示的是当前条目的编号。

用户可以向任何栏内输入任意的内容。这几栏都被设置为可编辑的，而且被设置为“ Tab to Next Editable Item ”，这样，它们可以像其他程序的输入栏那样对用户按 Tab键有所反应。

屏幕底部的按钮使用了第 14章“ 创建行为 ”里所描述的复杂的按钮行为。 Next和Previous按钮可以调出下一个或前一个记录； Done按钮调出主菜单屏幕；而 New按钮则在数据库的尾部创建一个新记录，并使它成为当前条目。

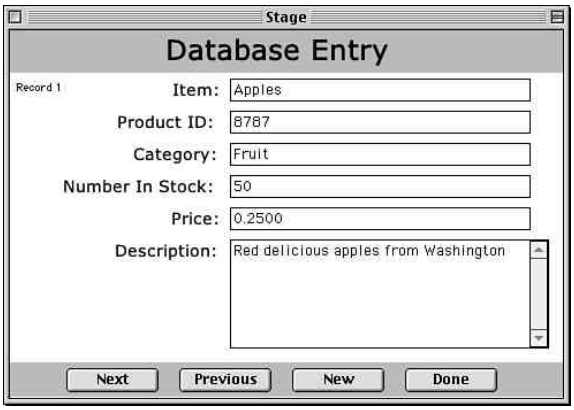

图28-2 超市的库存数据库的数据库条目屏幕

无论按下哪个按钮，都需要先把这些栏里的信息存储在数据库里，然后再执行相应的动作。

on recordRecord处理程序把所有栏里的文本放在一个新的空记录里，然后用它取代数据库全局变量里的旧记录：

```
on recordRecord
  -- get empty record to use
  record = newRecord()

  -- change all fields of record to reflect screen data
  record.item = member("Entry - Item").text
  record.id = member("Entry - ID").text
  record.category = member("Entry - Category").text
  record.number = member("Entry - Number").text
  record.price = value(member("Entry - Price").text)
  record.description = member("Entry - Description").text

  -- replace record in database
  gDatabase[gCurrentRecord] = record
end
```

28.1.3 浏览数据库

除了建立新数据库外，用户还应当能够返回数据库条目屏幕里去编辑记录。 Edit Database按钮可以调用editDatabase处理程序。这个处理程序在允许用户编辑数据库前，要先确认数据库存在。如果没有任何数据库，它将创建一个，而不是给出错误信息：

```
on editDatabase
  -- check to see if any database exists
  if not listP(gDatabase) then
    nextDatabase
  else
    go to frame "Entry"
    gCurrentRecord = 1
    showRecord
  end if
end
```

Next和Previous按钮也带有影片处理程序，负责记录当前条目的改变，并在数据库里向前

或向后移动：

```
-- move forward in the database
on nextRecord
  -- accept all changes in current record
  recordRecord

  -- go to the next record
  gCurrentRecord = gCurrentRecord + 1

  -- if past the end of the database, loop around
  if gCurrentRecord > gDatabase.count then
    gCurrentRecord = 1
  end if

  -- display the new record
  showRecord
end

-- move backward in the database
on previousRecord
  -- accept all changes in current record
  recordRecord

  -- go to the previous record
  gCurrentRecord = gCurrentRecord - 1

  -- if user tries to move back past 1, loop around
  if gCurrentRecord < 1 then
    gCurrentRecord = gDatabase.count
  end if

  -- display the new record
  showRecord
end
```

当用户点击New按钮时，必须有一个处理程序去找到数据库里的最后一个条目，并在它的后面再添加一个条目：

```
-- create a new record in the database
on createRecord
  -- accept all changes in current record
  recordRecord

  -- go to the record one past the last in database
  gCurrentRecord = gDatabase.count + 1

  -- set this new record to a blank record
  gDatabase[gCurrentRecord] = newRecord()

  -- display the new record
  showRecord
end
```

Done按钮使用户能够退出数据库条目屏幕，返回到主菜单屏幕。同这一部分的其他处理程序一样，这个按钮必须先存储对当前条目的改动，然后再执行动作：

```
-- finished entering data, return to main menu
on doneEntry
  recordRecord
  go to frame "Main"
end
```

接下去需要讨论的是Save Database和Open Database按钮。这两个按钮都调用使用 FileIO Xtra的处理程序，向文件中存储或从文件中调出数据库全局变量。

on saveDatabase处理程序创建一个文件，并把列表以字符串的形式存入其中。在更高级的程序里，这就是Save As...功能。在那种情况下，还需要另外一个执行 Save动作的处理程序，即把数据库存储在同一个文件里，而不必用户指定文件的位置。为了简单起见，本程序只使用了一个处理程序：

```
-- save the current database to a text file
on saveDatabase
  -- ask user for a file name
  fileObj = new(Xtra "FileIO")
  filename = displaySave(fileObj, "Save Database", "database.txt")
  if filename = " " then exit

  -- create file and write to it
  createFile(fileObj,filename)
  openFile(fileObj,filename, 2)
  writeString(fileObj, string(gDatabase))
  closeFile(fileObj)
end
```

要打开一个文件，需要编写一个与存储文件相似的处理程序。同时，要验证一下用户所选定的文件的类型。在本例里，用户可以打开任何类型的文件，但要读取其中的文本，以确认它是有效的Lingo列表。如果是，它就被认为是有效的数据库文件。如果你觉得有必要，还可以进行更严格的检查，以确认它是一个线性列表，并且只包含相似的属性：

```
-- open an existing database file
on openDatabase
  -- ask user for a file
  fileObj = new(Xtra "FileIO")
  filename = displayOpen(fileObj)
  if filename = " " then exit

  -- open the file and read the text
  openFile(fileObj, filename, 1)
  text = readFile(fileObj)
  closeFile(fileObj)

  -- try to convert the text to a list
  database = value(text)
  if not listP(database) then
    -- not a list
    alert "Not a valid database file. "
  else
    -- is a list, set database
    gDatabase = database
  end if
end
```

现在，我们就可以创建数据库、向其中添加条目、编辑条目、把它存储到硬盘上和从硬盘上读出了。既然建成了数据库，就可以利用它做一些有实际意义的事情了。

28.1.4 列出数据库里的记录

下面的处理程序创建一个带滚动条的文本区域，显示数据库里的全部项目。它在一个文本演员里用HTML的格式制作一个表格。图28-3是带有这个表格的屏幕。

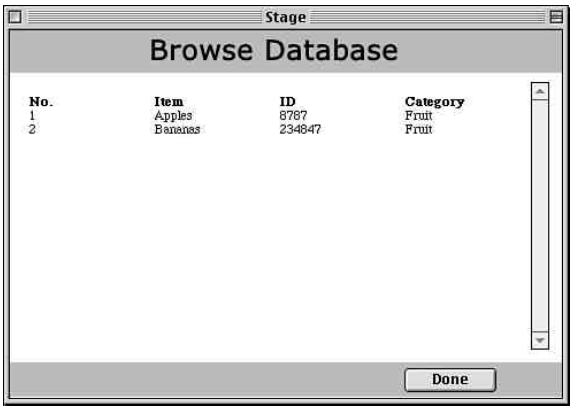

图28-3 在Browse Database屏幕里，用HTML格式的文本演员列出了数据库里的全部项目

```
-- display a list of all the records using HTML
on browseDatabase

  -- html header
  htext = "<HTML><BODY BGCOLOR=#FFFFFF>"&RETURN

  -- put the table headings
  put "<TABLE><TR><TH>No.</TH><TH>Item</TH><TH>ID</TH><TH>Category</TH></TR>"┐
     &RETURN after htext

  -- loop through database and create table rows
  repeat with i = 1 to gDatabase.count
    put "<TR>" after htext
    put "<TD>"&i&"</TD>" after htext
    put "<TD>"&gDatabase[i].item&"</TD>" after htext
    put "<TD>"&gDatabase[i].id&"</TD>" after htext
    put "<TD>"&gDatabase[i].category&"</TD>" after htext
    put "</TR>"&RETURN after htext
  end repeat

  -- close out table and HTML
  put "</TABLE></BODY></HTML>" after htext

  -- place HTML in text member
  member("Database List").html = htext

  go to frame "Browse"
end
```

尽管能够在一个清单里看到数据库的全部条目是一件很好的事情，但如果能按某种分类

原则把显示的范围缩小就更好了。下面的处理程序与前面的处理程序相似，但可以执行对数据库的简单检索。结果得到HTML格式的清单。

这个处理程序把要检索的关键词放在一个文本演员里。首先把用户带到一个检索屏幕里，并请他输入检索关键词：

```
-- display a list of records that are found in search
on performSearch

  -- get search term from field
  searchText = member("Search Text").text

  -- HTML header
  htext = "<HTML><BODY BGCOLOR=#FFFFFF>"&RETURN

  -- put the table headings
  put "<TABLE><TR><TH>No.</TH><TH>Item</TH><TH>ID</TH><TH>Category</TH></TR>"¬
    &RETURN after htext

  -- loop through all records
  repeat with i = 1 to gDatabase.count
    record = gDatabase[i]

    -- see if the record contains the search text
    -- search all properties for it
    if (record.item contains searchText) or¬
      (record.id contains searchText) or¬
      (record.category contains searchText) or¬
      (record.number contains searchText) or¬
      (record.price contains searchText) or¬
      (record.description contains searchText) then

      -- a match was found, add a row to table
      put "<TR>" after htext
      put "<TD>"&i&"</TD>" after htext
      put "<TD>"&record.item&"</TD>" after htext
      put "<TD>"&record.id&"</TD>" after htext
      put "<TD>"&record.category&"</TD>" after htext
      put "</TR>"&RETURN after htext
    end if
  end repeat

  -- close out table and HTML
  put "</TABLE></BODY></HTML>" after htext

  -- put HTML into text member
  member("Database List").html = htext

  go to frame "Search Results"
end
```

上面的程序是在数据库的所有栏内检索某个关键词，不过，我们也许想要把检索的范围再加以限制。例如，可以只检索项目的描述和名称。也可以检索类别栏内的信息，以找到性质相似的项目，如所有水果或所有肉类。甚至可以检索价格栏，找到所有低于或高于某个价格的项目，而不只是找一些相同的信息。

正是这些检索功能才使得数据库变得更有用。数据库的另一个常用功能是计算。可以编写一个处理程序，让它读取所有项目的价格，再分别乘以它们的数量，得到库存商品的总价值。

参见第13章“重要的Lingo句法”里的13.10节“使用列表变量”，可以获得关于列表的信息。

参见第16章“控制文本”里的16.9节“使用文本文件和FileIO Xtra”，可以获得更多有关FileIO的信息。

参见第16章里的16.7节“使用HTML和表格”，可以获得更多关于在文本演员里使用HTML的信息。

28.2 制作图表

图表通常就是另外一种位图。我们可以在Microsoft Excel或Lotus里轻易地制作这类图表，然后把这些图表拷贝、粘贴到Director里作为位图演员。

不过，制作这些图表的另外一种方法是用Director的图形、文本演员和线条等各种元素。

28.2.1 柱图

图28-4是在Director里制作的一个柱图。柱和x、y轴使用了不同的图形角色， x、y轴的刻度使用了文本演员。

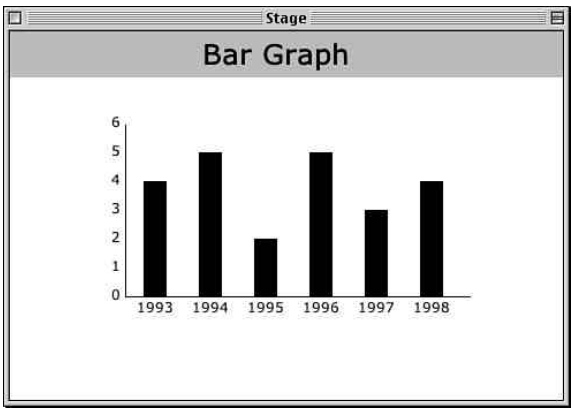

图28-4 这个图里使用了几种不同的Director元素，并用Lingo命令正确地调节了每个柱的高度

尽管我们可以手工布置图28-4里的所有元素，但这里使用了一个处理程序把柱调节至适当的高度。下面就是这个处理程序：

```
-- adjust a 6-bar graph
on adjustGraph valueList
  repeat with i = 1 to 6

    -- bar sprites start at 6
    sNum = 5+i

    -- get the rectangle of the sprite
    r = sprite(sNum).rect
```

```
    -- reset the rectangle top
    r.top = r.bottom - valueList[i]*25

    -- set the sprite
    sprite(sNum).rect = r
  end repeat
end
```

这个处理程序假定这些柱是角色 6～11。它以25像素为一个刻度与另一个刻度间的距离。可以用如下方式调用该处理程序：

```
on startMovie
  adjustGraph([4,5,2,5,3,4])
end
```

这个处理程序把柱的高度调节成图 28-4的样子。我们可以再次调用这个处理程序，但使用不同的参数，从而以另外一种方式设置柱图。可以制作一个包含多个柱图的影片，但要用Lingo制作那些柱图，而不是每次都使用不同的位图。

例如，左边的数字可以表示销售额，其单位为“百万美元”。底部的数字当然是年份。然后，可以用几个不同的柱表示不同公司的销售额，或同一个公司里的不同部门的数据。这些柱使用的都是相同的角色，但每次用不同的参数调用 on adjustGraph函数。

还可以编写更复杂的处理程序，使它能调节柱的数量、底部和左侧的文字甚至柱的颜色。要实现这些是相当复杂的，因为创建刻度并把它们按适当的间距分开，是相当困难的。实际上我们可以把每种刻度放在独立的文本演员里，并适当地放置它们。

28.2.2 饼图

另外一种形象的图形是饼图。在 Director 7之前，饼图是很难制作的，因为把饼切开的效果要逐个像素地画出来。不过，由于现在有了矢量图形演员，我们就可以用单个演员制作饼图了。我们应当还记得，可以用曲线制作这类演员，并根据需要填充单一颜色或渐变色。

在图28-5里，舞台上就有这样一个演员。这个图是由一个矢量图形演员和一个行为产生的。在on beginSprite处理程序里，行为把演员做成饼图的形状。

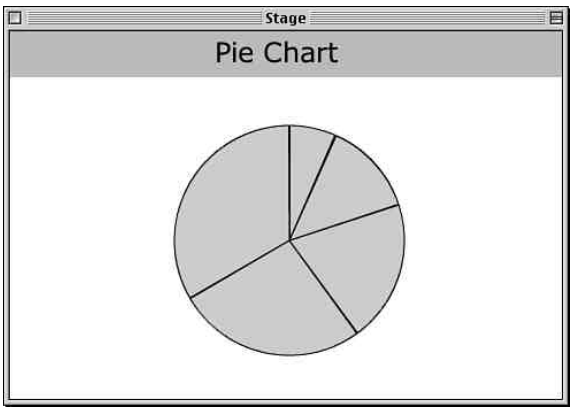

图28-5 由单个矢量图形演员构成的饼图

这个行为使用两个参数：饼块所用的值和饼的半径。然后它就通过画一个圆和把多条直线的一端粘在一个恰当的位置从而一块一块地画出这个饼图。

```
property pValues, pRadius

on getPropertyDescriptionList me
  list = [:]
  addProp list, #pValues, [#comment: "Value List", ¬
    #default: [], #format: #list]
  addProp list, #pRadius, [#comment: "Radius", ¬
    #default: 100, #format: #integer]
  return list
end

on beginSprite me

  -- determine total value of pie
  total = 0.0
  repeat with i = 1 to pValues.count
    total = total + pValues[i]
  end repeat

  -- start new vertextList
  vlist = []

  -- start 1/4 circle, back
  oldAngle = -pi()/2

  -- start at 0
  swing = 0.0

  repeat with i = 1 to pValues.count

    -- begin slice at center
    add vlist, [#vertex: point(0,0)]

    -- calculate angle for end of piece
    swing = swing + pValues[i]
    newAngle = 2.0*pi()*(swing/total)-pi()/2

    -- move from start to end of piece
    repeat with a = 100*oldAngle to 100*newAngle

      -- add point on circumference
      x = pRadius*cos(a/100)
      y = pRadius*sin(a/100)
      add vlist, [#vertex: point(x,y)]
    end repeat

    -- set start for next piece
    oldAngle = newAngle
  end repeat

  -- use vertexList on this sprite
  sprite(me.spriteNum).member.vertexList = vlist
```

```
end
```

把这个行为赋给任意一个矢量图形演员，就可以很容易地快速画出饼图。不过，现在它还不像常规的饼图那样完美，因为整个饼只有一种颜色。由于矢量图形演员只能有一种填充色，因此只用一个矢量图形是无法制作出多色的饼图的。不过，只要使用多个矢量图形就可以了。

下面这个行为与刚才的那一个相似，但它只画出了饼里的某一块。可以把这个行为赋给多个矢量图形演员，它们都将被放置在舞台上相同的位置。不过，要确保每一块饼使用的矢量图形演员各不相同。如果不同角色使用了相同的矢量图形演员，程序将不能正常运行。

这个行为里有一个新参数pPiece。该参数决定角色应当代表饼里的哪一块。在使用这些行为时，我们可以同时把它拖到六个饼角色里，并同时设置 pValues和pRadius属性，然后再返回头用行为监察窗为每块饼重新设置 pPiece属性。请看下面的程序：

```
property pValues, pRadius, pPiece

on getPropertyDescriptionList me
  list = [:]
  addProp list, #pValues, [#comment: "Value List",¬
    #default: [], #format: #list]
  addProp list, #pRadius, [#comment: "Radius", ¬
    #default: 100, #format: #integer]
  addProp list, #pPiece, [#comment: "Piece Number", ¬
    #default: 1, #format: #integer]
  return list
end

on beginSprite me

  -- determine total value of pie
  total = 0.0
  repeat with i = 1 to pValues.count
    total = total + pValues[i]
  end repeat

  -- start new vertextList
  vlist = []

  -- start 1/4 circle, back
  oldAngle = -pi()/2

  -- start at 0
  swing = 0.0

  repeat with i = 1 to pValues.count

    -- begin slice at center
    add vlist, [#vertex: point(0,0)]

    -- calculate angle for end of piece
    swing = swing + pValues[i]
    newAngle = 2.0*pi()*(swing/total)-pi()/2

    -- only draw my own piece
```

```
  if pPiece = i then
    -- move from start to end of piece
    repeat with a = 100*oldAngle to 100*newAngle

      -- add point on circumference
      x = pRadius*cos(a/100)
      y = pRadius*sin(a/100)
      add vlist, [#vertex: point(x,y)]
    end repeat
  end if

  -- set start for next piece
  oldAngle = newAngle
 end repeat

 -- use vertexList on this sprite
 sprite(me.spriteNum).member.centerRegPoint = FALSE
 sprite(me.spriteNum).member.originMode = #center
 sprite(me.spriteNum).member.vertexList = vlist
end
```

当饼图做好后，还可以编辑每一个演员，改变它们的颜色。最终将得到如图 28-6所示的饼图。我们还可以向行为里添加一个pPieceColor参数，让它设置演员的fillColor属性。

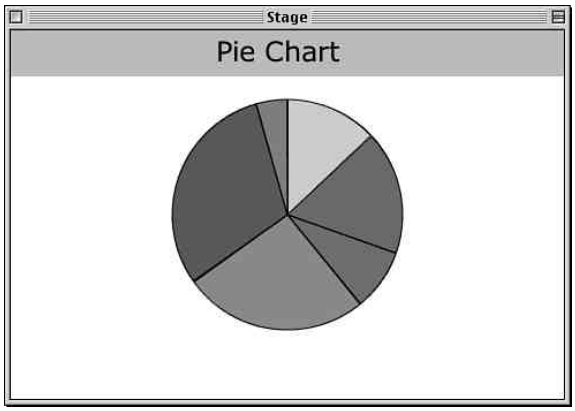

图28-6 这个多色饼图是由六个矢量图形角色组成的，它们使用的是同一个行为

同柱图一样，这个饼图的行为只是半自动的。我们可以不必手工调整剪辑室和舞台，就能容易地改变每一块饼的值。不过，如果想要添加第七块饼，就需要再添加一个角色，并改动行为的所有属性。

还可以制作更复杂一些的行为。它可以通过使用new命令创建新演员，然后用这些演员创建空的角色，从而创建各块饼。这样，我们就可以不需要现成的角色或演员而动态地制作饼图了。

参见第20章“控制矢量图形”里的“用 Lingo建立矢量图形”，可以获得更多有关建立矢量图形演员的信息。

28.3 制作调查问卷

现在已很少使用纸张形式的调查问卷了。最常用的调查方法是电话调查。当然，电话哪

一端的人有一台计算机，把我们给他的答案输入进去。调查还可以用 Web上的独立的计算机信息站进行。

在Director里制作调查问卷与第27章“ 教学软件 ”里的标准化考试非常相似。唯一的区别在于调查问卷还可以使用除单选按钮以外的其他方法。复选框和弹出菜单都是很好的选择。此外，空白的文本区域还可以让用户随意输入一些内容。考试与调查的区别还在于后者没有核对答案的过程，因为其答案没有“ 对 ”与“ 错 ”之分。

在CD-ROM的样例里，给出了含有四个问题的调查问卷。用户由按 Begin按钮开始。这时将为全局变量gAnswers清零，并把演员表里的所有单选按钮和复选框演员复位：

```
global gAnswers -- list to store all responses

on beginQuestions
  -- clear list
  gAnswers = []

  -- remove previous selections
  repeat with i = 1 to the number of members
    if member(i).type = #button then
      member(i).hilite = FALSE
    end if
  end repeat

  -- start
  go to frame "question 1"
end
```

像第27章里的标准化考试程序一样，第一个问题使用了五个单选按钮。图 28-7就是这个问题。

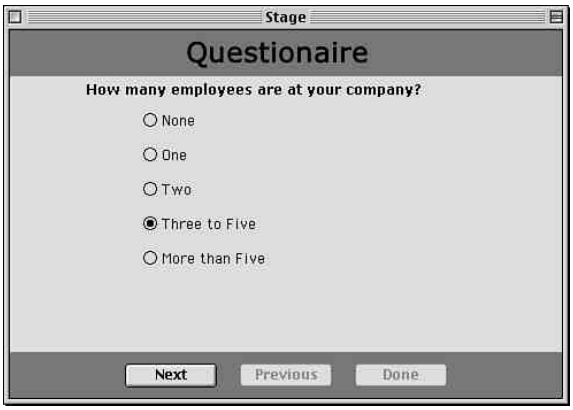

图28-7 调查问卷里的第一个问题使用的是单选按钮，让用户选择一个而且只能选择一个答案

控制这些单选按钮的程序与第 15章“ 图形界面元素 ”里介绍的单选按钮行为相似。不过，在每当一个单选按钮被点击时，该行为就调用一个名为 on addAnswer的处理程序。下面就是这个处理程序：

```
property pState, pGroupList

on getPropertyDescriptionList me
```

```
  list = [:]
  addProp list, #pState, [#comment: "Initial State",¬
    #format: #boolean, #default: FALSE]
  addProp list, #pGroupList, [#comment: "Group List",¬
    #format: #list, #default: []]
  return list
end

on beginSprite me
  if pState then turnMeOn(me)
end

on turnMeOn me
  pState = TRUE
  sprite(me.spriteNum).member.hilite = TRUE
  repeat with i in pGroupList
    if i <> me.spriteNum then
      sendSprite(sprite i,#turnMeOff)
    end if
  end repeat
  addAnswer(sprite(me.spriteNum).member.text)
end

on turnMeOff me
  pState = FALSE
  sprite(me.spriteNum).member.hilite = FALSE
end

on mouseUp me
  turnMeOn(me)
end
```

在这个影片的影片剧本里， on addAnswer把一个字符串添加到全局变量 gAnswers里。它从帧标签里得到问题的编号：

```
on addAnswer text
  -- get question number from frame label
  question = value((the frameLabel).word[2])

  -- set the item in the list
  setAt(gAnswers, question, text)
end
```

影片里的第二个问题使用的是复选框。它允许用户不选择屏幕里的答案，或选择一个、几个或所有答案。图28-8是这个屏幕。

复选框是相互独立的，当用户点击它们时，它们可以分别为“开”或“关”。不过，还需要一个行为来改变全局变量gAnswer。这个行为调用的是单选按钮所使用的那个 on addAnswer 处理程序。

```
on mouseUp me
  text = " "
  repeat with i = 11 to 15
    if sprite(i).member.hilite = TRUE then
      if text <> " " then put "," after text
      put sprite(i).member.text after text
```

```
    end if
  end repeat
  addAnswer(text)
end
```

第三个界面使用了另一种界面元素。这是一个弹出菜单，用户可以从菜单里选择某一项。单选按钮的功能与之相同，但弹出菜单里可以容下更多项内容。图 28-9就是这个屏幕。

图28-8 当允许多个答案存在时，就用复选框来提问

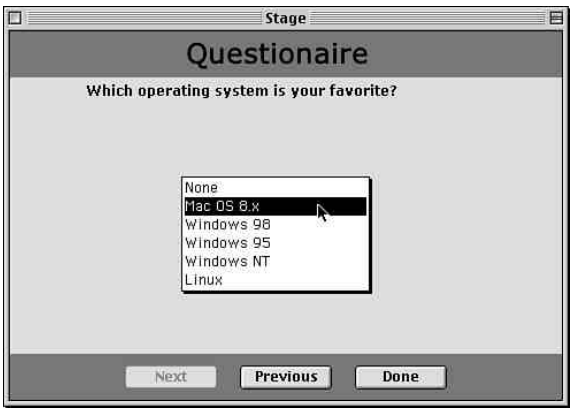

图28-9 弹出菜单与单选按钮的功能相同

弹出菜单可以使用第16章所介绍的程序。这段程序调用 on addAnswer，并把pSeleted行为属性作为参数。

最后，调查问卷还可以使用如图 28-10所示的可编辑的文本区域。

可编辑的文本区域也需要一个行为。下面的简单行为可以在该角色里出现时清除里面的文本，然后在离开该帧时把用户输入的文本送到 on addAnswer处理程序里。

```
on beginSprite me
  sprite(me.spriteNum).member.text = " "
end

on endSprite me
  addAnswer(sprite(me.spriteNum).member.text)
end
```

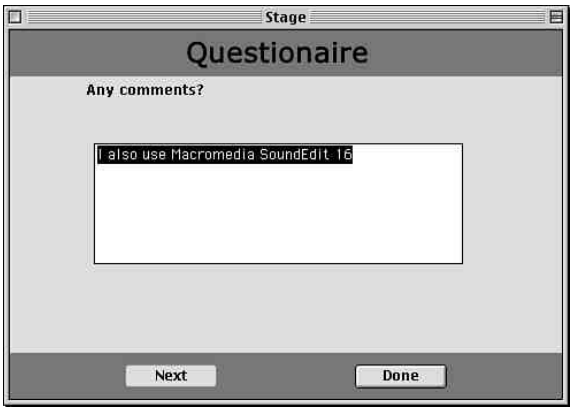

图28-10 可编辑文本区域允许用户输入评论或其他信息

最后，gAnswer列表可以被转换为一个字符串，并存储在一个文件里。我们甚至可以把它追加到某个现有文件的结尾，从而能够编辑多个问卷的结果。如果这是一个 Web上的短程序，你也许想要取得这些数据，并用 postNetText把它们送到服务器上的CGI程序里。

参见第15章里的15.3节“ 使用单选按钮 ”，可以获得更多有关使用单选按钮的信息。

参见第15章里的15.2节“ 使用复选框 ”，可以获得更多有关使用复选框的信息。

参见第15章里的15.7节“ 制作图形化的弹出菜单 ”，可以获得更多有关制作弹出菜单的信息。

参见第16章里的16.5节“ 使用键盘输入 ”，可以获得更多有关接收键盘输入命令的信息。

28.4 制作基于计算机的培训程序

对于检验用户掌握知识的水平来说，测验或考试是很好的手段。但如果适当地设置，它们也可以变成教学程序。基于计算机的培训程序是指用计算机讲解概念的程序。它们使用人—机界面来强调某些概念或操作方法。

举例来说，基于计算机的培训程序可以教一个公司里的职员如何使用一种新的电话系统。如果这个系统用很多代码执行很多命令，职员们开始接触它时总是需要翻阅说明书去查找某种代码的功能。如果职员们能够记住这些代码，就可以十分高效地使用这个系统了。

基于计算机的培训程序可以显示给用户一幅图像，然后问一些这样的问题：“ 如何转接电话？ ”用户就必须按下屏幕上的电话图像里的某些键。如果按错了，就得重新开始。如果正确了，就接着练习下一个功能。图 28-11就是这种程序的一个画面。

设计这个程序的意图是让人们在每天把它使用几次后，就可以记住这些代码了。此外，他们可以在真正需要使用某些代码前就已了解它们了。

这个影片的程序与本章的调查问卷的程序相似。它需要三个全局变量：当前问题编号、当前问题的正确答案和用户目前给出的答案。

```
global gQuestionNum -- which question number
global gCorrectAnswer -- correct for current question
global gAnswer -- the user's answer, so far
```

影片开始时，把gQuestionNum设置为1，然后调用on askQuestion。这个处理程序从一个文本区域内取得问题以及正确的答案，然后把问题显示在舞台上，把正确答案放在gCorrectAnswer里。

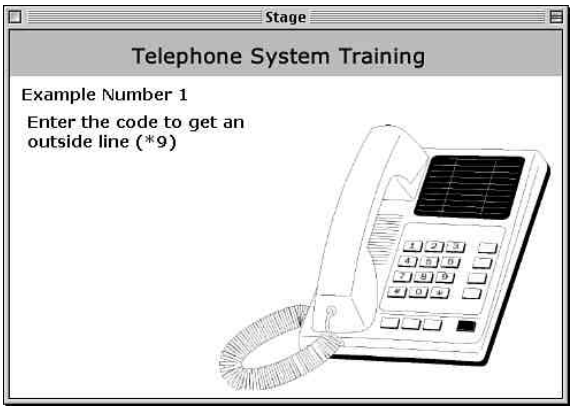

图28-11 基于计算机的培训程序屏幕，教用户如何使用不同的电话代码

```
-- reset questions and ask first
on startMovie
  gQuestionNum = 1
  askQuestion
end

-- get question info and ask it
on askQuestion
  -- data uses ; as separator
  the itemDelimiter = ";"

  -- get the data from a field
  text = member("Questions").text.line[gQuestionNum]

  -- set the question
  member("Question").text = text.item[1]

  -- get the answer
  gCorrectAnswer = text.item[2]

  -- reset user's answer
  gAnswer = " "

  -- set message
  member("Message").text = "Example Number"&&gQuestionNum
end
```

应当仔细地设置含有问题及答案的文本区域。它使用分号作为各个项目之间的分隔符：

```
Enter the code to get an outside line (*9);*9;
Enter the code to forward the current call (22#);22#;
Enter the code to turn auto answer on (#7);#7;
Enter the code to turn auto answer off (*3);*3;
Enter the code to get an operator (*072952#4);*072952#4;
```

正确答案可以使用10个数字以及“ # ”和“ * ”字符。这些字符与电话机上的按键相对应。图28-11里的电话图像的键盘上的确有一些小按钮。每一个按钮都有一个被恰当命名的演员，例如，在“ 1 ”键上的演员的名称为“ 1 ”。当其中的某一个按钮被按下时，它就向影片处理程序发送一个消息。

```
on mouseUp me
  -- gets the character from the member name
  k = sprite(me.spriteNum).member.name

  -- send along to movie script
  addToAnswer(k)
end
```

addToAnswer影片处理程序里含有 gAnswer字符串。它把新字符放在现有的字符的后面。然后把整个字符串与正确答案对比。如果二者相同，程序则前进到下一个问题；如果不同，但与正确答案的前几个字符相对应，表示用户还有输入正确答案的希望。例如，如果用户输入了“ 6 ”，而正确答案是“ 69 ”。

不过，只要检查出用户的输入与标准答案有一点不同，就显示一则信息，让用户重新试一次。

```
on addToAnswer k

  -- add to answer
  put k after gAnswer

  put gAnswer, gCorrectAnswer

  if gAnswer = gCorrectAnswer then
    gotItRight
  else if gCorrectAnswer starts gAnswer then
    -- one step closer, nothing to do here
  else
    gotItWrong
  end if
end
```

on gotItRight和on gotItWrong处理程序负责显示信息和其他回答完问题之后的事情。

```
-- move on to next question
on gotItRight

  -- next question
  gQuestionNum = gQuestionNum + 1

  -- all done?
  if gQuestionNum > member("Questions").line.count then
    member("Message").text = "Finished. "
    beep(3)
  else
    beep()
    askQuestion
  end if
end


-- wrong answer message
on gotItWrong
  member("Message").text =  "Wrong. Try again. "
  gAnswer = " "
end
```

可以注意到在这个程序里，提问时已经泄露了答案，这是为了调试的方便。实际的程序当然不会把这个信息告诉给用户。

这个程序是相当简单的。真正的基于计算机的培训软件可能相当复杂。例如，可以随机地提问题。我们还可以记录用户把哪些题答错了，并把这些题拿出来再重新考一考用户。

28.5 商用软件的故障排除

■ 在数据库和调查问卷软件里，有允许用户输入文本的区域。在用户输入文本时，不要让用户把回车符输入进去。尽管这样并不会造成程序错误，但我们就不能把数据列表转换成字符串然后再转换回来了。解决这个问题的方法是使用一个 on keyDown剧本，它或者不允许回车符通过，或者在存储前把回车符转换成其他字符。

■ 当制作饼图或其他基于矢量图形的图像时，应当确认在演员的属性对话框里使用 Auto-Size选项，并把centerRegPoint设置为FALSE，把originMode设置为#center。否则在舞台上就不能把演员对齐。

28.6 你知道吗

■ 在制作柱图时，图形演员可以使用某种颜色，也可以使用某种图案。在舞台上选择该演员，然后再使用工具面板即可。

■ 当在Mac上存储数据库文件时，可以用 FileIO setFinderInfo为数据库文件设置一种文件类型。在Mac和Windows上都可以用setFilterMask限制用户，使他们只能打开某些文件。

■ 如果我们能够写一个短程序，把调查问卷的数据存储成用制表符分隔的列表，而不是Director的列表，就可以把它输入到某个电子表格或统计分析程序，从而处理调查的结果。