第29章　图形图像软件

在Director影片里，随处都可以看到图形图像，有时它们的地位已经超出了“影片的组成部分”这个范围。在很多情况下，它们几乎就是影片的全部。

例如，幻灯片把一组图像作为整部影片的中心焦点。有时，影片只显示一幅地图或绘图等图像，但这幅图像的尺寸超出了屏幕的尺寸。这时，就需要使用移动视窗、滚动条或对图像变焦缩放等手段。

另外一种可能是影片的图像像蒙太奇那样不断地变化。 Director通过运用油墨和颜色，可以轻易地实现这种效果。

29.1　制作幻灯片

制作第2章“用Director制作演示”里的幻灯片是很容易的。简单一些的影片甚至不需要任何Lingo。可以把图像输入到演员表里，然后在剪辑室里设置显示每幅图像的各个帧。

不过，如果我们还不知道将要在幻灯片里播放哪些图像，应当怎么制作幻灯片呢？或者，如果图像可能需要替换成其他图像时又怎么办呢？有时，需要添加图像的人不会使用 Director，甚至根本没有Director时又怎么办呢？

我们可以制作一个智能的Director影片，它可以把一个文件夹里的图像编辑成一个幻灯片。用户可以随意增减该文件夹里的图像，从而得到完全不同的幻灯片。

这样的影片看上去就像图 29-1。这幅图像实际上是一个链接的演员，而不是输入的演员。要实现这一点，应当使标准的输入方法，但在输入之前，在 Import对话框里要选择Link to File。

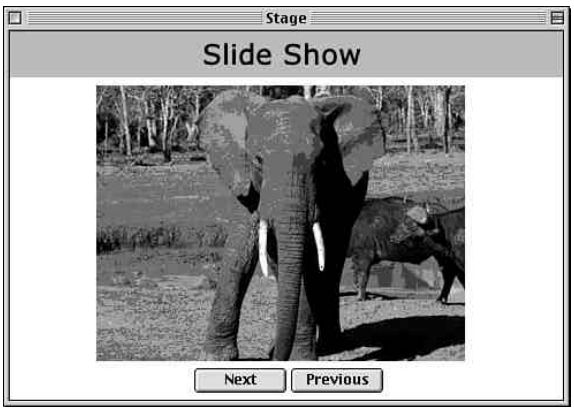

图29-1　简单的幻灯片可以逐个地输入图像，而不需要它们一开始就全出现在影片里

所有用于幻灯的图像都应当放在名为 Images的文件夹里。这个文件夹应当与影片在同一个文件夹里，将来也与放映机在同一个文件夹里。为了简化程序，应当把第一个图像演员与这个文件夹里的幻灯片的第一幅图像链接起来。

影片首先需要读取 Images文件来里的内容，并把文件的名称存储在一个列表里。

getNthFileNameInFolder函数将执行这个任务。

注释 getNthFileNameInFolder函数得到路径名和编号这两个参数。它返回特定文件夹里的与编号相对应的文件名。如果给定的数字比那个文件夹里的文件总数还大，则返回一个空字符串。根据以往经验，文件夹里的文件总数少于255时，该函数的功能比较可靠。

文件的列表存储在一个全局变量里。同时，该文件夹的路径也存储在一个全局变量里。当输入每一幅图像时，还将需要它。文件夹的路径由影片或放映机的位置决定，外加 Image这个词。此外，在其尾部还有一个额外的文件分隔符。在 Mac上，这个分隔符是“ :”；在Windows上，这个分隔符是“ \”。

下面是所需要的全局变量声明、 on startMovie处理程序和编辑图像文件列表的 on getImageList处理程序：

```
global gImageFolder -- where the images are located
global gImageList -- a list of all image file names
global gImageNum -- the number of the current image

on startMovie
  -- determine the location of the images
  gImageFolder = the pathname & "Images"
  if the platform contains "mac" then
    gImageFolder = gImageFolder & ":"
  else
    gImageFolder = gImageFolder & "\"
  end if

  -- set other globals
  getImageList
  gImageNum = 1

  -- show first image
  showImage
end

-- this handler looks at each file in the folder and gets the
-- filename of that image
on getImageList
  gImageList = []
  repeat with i = 1 to 255
    filename = getNthFileNameinFolder(gImageFolder,i)
    if filename = " " then exit repeat
    add gImageList, filename
  end repeat
end
```

当文件名被存储到列表里后，把文件输入到演员表的任务就很简单了，只需要设置演员的filename属性。程序把它设置为图像的完整路径，包括前面经过编辑的 gImageFolder全局变量。

```
-- this handler resets the filename of the image to a new file
on showImage
  member("Image").filename = gImageFolder & gImageList[gImageNum]
```

```
end
```

可以注意到，不需要改变角色或剪辑室。剪辑室并没有变化，还是显示着相同的演员。是演员本身变化了。

在图29-1的例子里，有Next和Previous两个按钮。为这两个按钮配剧本并不困难，只要确保用户向后不要超过结尾，向前不要超过开头就可以了。

```
on nextImage
  gImageNum = gImageNum + 1
  if gImageNum > gImageList.count then
    gImageNum = 1
  end if
  showImage
end

on previousImage
  gImageNum = gImageNum - 1
  if gImageNum < 1 then
    gImageNum = gImageList.count
  end if
  showImage
end
```

CD-ROM里的影片样例里的两个按钮使用的是第 14章“ 创建行为 ”里所介绍的复杂的按钮行为。此外，在on exitFrame处理程序里，还放置有一个简单的go to the frame。

注释 如果需要调入的文件的数据量很大，把它从硬盘里读出来就要花一些时间，这样在显示图像时会造成一些延迟。要想把这种延迟减到最短，可以把文件的数据量减小，方法是使用GIF或JPEG等数据量较小的格式。

图像是按照字母顺序先后显示的。 Director并没有干预这件事， getNthFileNameInFolder只是忠实地按操作系统对文件的排序方式返回这些文件的。

提示 让图像按照我们所希望的顺序出现的一种简单方法是在每个文件名前面加上数字等一些字符。例如，文件0010zebra.jpg将出现在文件0020elephant.jpg的前面。文件名开头的几个0使我们能够编排至少1000幅图像，并把它们正确地排序；文件名结尾的一个0使我们能够在某些图像之间轻易地安插进去几幅图像，如0015gorilla。

getNthFileNameInFolder技术仅限于在Director和放映机里使用。在Shockwave里不能像这样使用getNthFileNameInFolder，因为Shockwave不支持这个函数。不过，我们仍旧可以通过重新设置链接的位图演员的filename属性来输入图像。这个属性不但可以接受本地磁盘的路径，而且可以接受因特网的网址。

问题就转化成确定哪些文件存在。可以在服务器上的一个文本文件里放置一个由图像的文件名构成的列表。可以用 getNetText系列命令读取这个文件。只有在删除或添加了某个图像后才需要更新该文件。

有些服务器甚至可以把某个文件目录内的文件的清单作为一个简单的 HTML页面返回。这个HTML页面叫做Index。如果服务器有这个功能，我们可以用 getNetText与文件夹的路径得到一个及时更新的清单。不过，由于返回的文本并不是一个简单的列表，因此从中提取出文件名是比较困难的。

29.2 移动观察大幅图像

Director作品的创作者们迟早都会遇到图像尺寸大于舞台尺寸的情况。这些大幅图像经常是地图、绘图甚至是细节丰富的照片。

如果需要保留图像里的细节，就不能选用把图像缩小的方法。唯一的解决方案是每次只显示图像的某一部分并允许用户移动观察整幅图像，从而能够看到图像的任意部分。

要建立这种功能可以只是在剪辑室窗口里把角色的 movable属性设置为TRUE。不过，如果直接这样操作，会把图像完全移出窗口之外，而且该属性也没有提供抓住并拖动图像等特殊功能。

图29-2的影片只显示了部分图像。整幅图像其实都存在于它自己的角色通道里，但只有位于舞台中心的那一部分可见。屏幕的四周还有一些灰色的区域，可以用来放置按钮、说明和注释等元素。

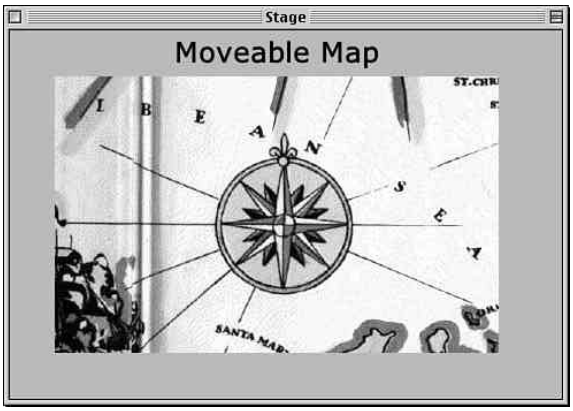

图29-2 在这个影片里只能看到一幅大图像的一部分。不过，用户可以点击和拖动图像，从而看到图像的任何部分

这个影片所需要的功能是使用户能够点击并拖动图像，却不至于把图像的边缘拖到画面内部。为了避免这一点，程序需要知道可见区域的坐标。通过创建一个隐藏在图像背后的矩形图形角色可以知道这个坐标值。图形的坐标与图像的坐标是相同的。它的行为要控制该角色的rect，使该角色不会被拖过它的极限位置。

on beginSprite处理程序取得这个图像背后的角色的 rect，并用它计算出pBounds属性。这个rect代表图像的套准点允许被移动的极限值，使得图像始终出现在可见区域内。这种计算是很简单的。试着运行CD-ROM里的例子，以分析它的作用。

```
property pDrag, pOffset, pBounds, pCursor

on beginSprite me
  pDrag = FALSE

  -- get rectangle of display area
  spr = sprite(me.spriteNum)
  mem = sprite(me.spriteNum).member
  screen = sprite(me.spriteNum-1).rect
  pBounds = rect(0,0,0,0)
  pBounds.left = screen.right - (mem.width - mem.regPoint.locH)
```

```
  pBounds.right = screen.left + (mem.regPoint.locH)
  pBounds.top = screen.bottom - (mem.height - mem.regPoint.locV)
  pBounds.bottom = screen.top + (mem.regPoint.locV)
end
```

当用户点击这个角色时，pDrag就会标明正在发生拖动操作。此外，还要记录点击处的坐标与套准点之间的距离，使用户能够从点击处移动角色，而不是从中心移动角色。

```
-- when the image is clicked, enable dragging
on mouseDown me
  pDrag = TRUE
  pOffset = the clickLoc - sprite(me.spriteNum).loc
end
```

除了拖动外，这个行为还使用光标告诉用户正在发生的事情。在每一帧都调用 on mouseWithin处理程序。它先查看正在进行什么动作，然后相应地把光标显示成攥紧的拳头或伸开的手掌。为了确保cursor命令不至于在每一帧都被执行，on changeCursor处理程序首先把所需要的光标与上次改动后的光标进行比较，只有当需要再次改动光标时才调用 cursor命令。

```
-- when the cursor is over the image, see if a cursor is needed
on mouseWithin me
  -- check to make sure that the cursor is directly over sprite
  if the rollover = me.spriteNum then
    if pDrag = TRUE then
      -- closed hand for dragging
      changeCursor(me,290)
    else
      -- open hand if not dragging
      changeCursor(me,260)
    end if
  else
    -- normal cursor if not directly over sprite
    changeCursor(me,0)
  end if
end

-- if cursor leaves image area, reset it
on mouseLeave me
  changeCursor(me,0)
end

-- change the cursor only if not the same as last time
on changeCursor me, cursorNum
  if cursorNum <> pCursor then
    pCursor = cursorNum
    cursor(pCursor)
  end if
end
```

当用户释放鼠标按钮时，拖动就停止了：

```
-- stop dragging
on mouseUp me
  pDrag = FALSE
end
on mouseUpOutside me
  pDrag = FALSE
```

```
end
```

实际动作发生在on exitFrame程序里。根据鼠标的位置计算出了一个新的位置，然后要检查图像的边缘是否被拖到了规定的区域之外。如果是，将把图像的位置限制在规定的区域的边缘。

```
-- set the new location if dragging
on exitFrame me
  if pDrag then
    newloc = the mouseLoc - pOffset

    if newloc.locH < pBounds.left then newloc.locH = pBounds.left
    if newloc.locH > pBounds.right then newloc.locH = pBounds.right
    if newloc.locV < pBounds.top then newloc.locV = pBounds.top
    if newloc.locV > pBounds.bottom then newloc.locV = pBounds.bottom
    sprite(me.spriteNum).loc = newloc
  end if
end
```

通过把这个行为拖动到任何角色上，可以很轻易地使用这个行为。不过，在该角色通道的下面还需要一个矩形来确定可见区域。查看 CD-ROM里的例子，可以了解这些究竟是如何实现的。

用这个行为可以显示很宽或很高的图像。如果图像的高度或宽度中只有一个尺寸与屏幕恰好吻合，背后的那个矩形的对应尺寸也与之完全相同。这样用户在这个方向的根本无法移动图像，因为的确没有其他内容可供显示了。不过，用户仍旧能够在另一方向上拖动图像。

29.3 为大幅图像制作滚动条

尽管用移动视窗的方法使用户能够看到图像的任意部分，但却不能告诉他们现在看到的是图像的哪一个部位。而且，拖动鼠标有时会让人觉得厌烦。如果图像很大，用户需要点击和拖动很多次才能到达图像的另一端。

滚动条可以解决这个问题。很多计算机程序里都有滚动条，即使是对程序不熟悉的人也能够自如地使用它。

滚动条与第15章“图形界面元素”所描述的滑动条界面元素相似。那里有一个滑动条、一幅背景图像和两个箭头按钮。不过，现在不需要使用那个滑动条例子里的阴影了。

滚动条行为应当既用于横向滚动条，又用于纵向滚动条。行为控制横向滚动条还是纵向滚动条，是由参数决定的。此外，还需要一个链接滑动条与背景角色的参数，和一个链接滑动条与图像的角色的参数：

```
property pPressed -- whether the sprite is being pressed
property pBounds -- the rect of the shadow sprite at start
property pBGSprite -- the number of the shadow sprite
property pImageSprite -- the image to manipulate
property pValue -- actual value of the slider
property pOrientation -- horiz or vert scroll bar

on getPropertyDescriptionList me
  list = [:]
  addProp list, #pBGSprite, [#comment: "Scroll Background Sprite",¬
    #format: #integer, #default: 0]
```

```
  addProp list, #pOrientation, [#comment: "Orientation",¬
    #format: #symbol, #default: #horizontal, ¬
    #range: [#horizontal, #vertical]]
  addProp list, #pImageSprite, [#comment: "Image Sprite",¬
    #format: #integer, #default: 0]
  return list
end
```

在这个角色的开始，它要了解一些情况。首先要用背景角色确定滑动条的运动范围，因此背景角色的特征与第15章所描述的shadow角色相似。它还把滚动条滑动条的初始位置发送给图像角色，而名为on newImagePos的处理程序则相应地调整图像的位置：

```
on beginSprite me
  -- get the bounds on all four sides
  pBounds = sprite(pBGSprite).rect
  pBounds.right = pBounds.right - sprite(me.spriteNum).member.width/2
  pBounds.left = pBounds.left + sprite(me.spriteNum).member.width/2
  pBounds.bottom = pBounds.bottom - sprite(me.spriteNum).member.height/2
  pBounds.top = pBounds.top + sprite(me.spriteNum).member.height/2

  -- get initial value and send to image sprite also
  if pOrientation = #horizontal then
    pValue = sprite(me.spriteNum).locH - pBounds.left
    sendSprite(sprite pImageSprite,#newImagePos,getValue(me),VOID)
  else if pOrientation = #vertical then
    pValue = sprite(me.spriteNum).locV - pBounds.top
    sendSprite(sprite pImageSprite,#newImagePos,VOID,getValue(me))
  end if

  setMarker(me)
end
```

这个滑动条的运动是用拖动行为的典型处理程序实现的。

```
on mouseDown me
  pPressed = TRUE
end

on mouseUp me
  pPressed = FALSE
end

on mouseUpOutside me
  pPressed = FALSE
end

on exitFrame me
  if pPressed then
    moveMarker(me)
  end if
end
```

当滑动条被拖动时， on moveMarker处理程序使滑动条能够与鼠标一起运动，并随时检测滚动条的边界，以确保滑动条不会被拖出特定的范围。图像角色的行为里的 on newImagePos处理程序在每帧也将被调用，使得图像的位置随着滑动条的移动不断被更新。

```
on moveMarker me
```

```
  if pOrientation = #horizontal then
    pValue = the mouseH - pBounds.left
    -- check to make sure it is within bounds
    if pValue > pBounds.width then
      pValue = pBounds.width
    else if pValue < 0 then
      pValue = 0
    end if
    sendSprite(sprite pImageSprite,#newImagePos,getValue(me),VOID)
  else if pOrientation = #vertical then
    pValue = the mouseV - pBounds.top
    -- check to make sure it is within bounds
    if pValue > pBounds.height then
      pValue = pBounds.height
    else if pValue < 0 then
      pValue = 0
    end if
    sendSprite(sprite pImageSprite,#newImagePos,VOID,getValue(me))
  end if

  setMarker(me)
end
```

同第15章的滑动条行为一样，箭头按钮调用滑动条行为里的一个处理程序。这个处理程序每次把滚动条移动1像素。同前面的处理程序一样，它也执行边界检测命令，并向图像角色发送一个消息，从而更新它。

```
-- this handler moves the marker one value left or right
on moveMarkerOne me, direction
  if direction = #left or direction = #up then
    pValue = pValue - 1
  else if direction = #right or direction = #down then
    pValue = pValue + 1
  end if

  -- check to make sure it is within bounds
  if pOrientation = #horizontal then
    if pValue > pBounds.width then
      pValue = pBounds.width
    else if pValue < 0 then
      pValue = 0
    end if
    sendSprite(sprite pImageSprite,#newImagePos,getValue(me),VOID)
  else
    if pValue > pBounds.height then
      pValue = pBounds.height
    else if pValue < 0 then
      pValue = 0
    end if
    sendSprite(sprite pImageSprite,#newImagePos,VOID,getValue(me))
  end if

  setMarker(me)
end
```

这个行为还需要一个辅助处理程序 on setMarker，它可以根据滚动条的数值设置滑动条。

它是由on moveMarkerOne处理程序所使用的，这样当用户点击箭头按钮时，滑动条会移动。

```
-- this sets the marker sprite
on setMarker me
  if pOrientation = #horizontal then
    sprite(me.spriteNum).locH = pBounds.left + pValue
  else
    sprite(me.spriteNum).locV = pBounds.top + pValue
  end if
end
```

on getValue处理程序返回滑动条的值。这个处理程序可以被某个影片处理程序调用，也可以被另外一个行为调用。它返回 0～1间的一个数值来表示滑动条在滚动条中的位置，而不是返回滑动条的以像素为单位的位置值。

```
-- this handler returns the value of the slider between 0 and 1
on getValue me
  if pOrientation = #horizontal then
    return float(pValue)/float(pBounds.width)
  else
    return float(pValue)/float(pBounds.height)
  end if
end
```

最后，滚动条滑动条行为里的最后一个处理程序允许一个外部处理程序设置滑动条的位置。它使用0～1之间的一个数字计算出滑动条的以像素为单位的位置值。这个处理程序使用户能够让图像跟随光标移动，并在图像移动的同时更新滑动条。

```
-- this sets the marker according to a number between 0 and 1
on setValue me, percent
  if pOrientation = #horizontal then
    pValue = percent*pBounds.width
  else
    pValue = percent*pBounds.height
  end if
  setMarker(me)
end
```

滚动条的行为到此就结束了，但却还没有直接移动图像的能力。它几次调用位于图像角色行为里的on newImagePos处理程序。下面这个处理程序可以被放在刚才最后使用的同一个图像角色行为里，它基于0～1间的数值重新定位图像。

```
-- this handler receives messages from the scroll bars telling it
-- to change the position of the image
on newImagePos me, hPercent, vPercent
  if not voidP(hPercent) then
    -- message from horizontal scroll bar
    newH = pBounds.left + pBounds.width*(1.0-hPercent)
    sprite(me.spriteNum).locH = newH
  end if
  if not voidP(vPercent) then
    -- message from vertical scroll bar
    newV = pBounds.top + pBounds.height*(1.0-vPercent)
    sprite(me.spriteNum).locV = newV
  end if
end
```

在用户移动图像时，图像角色的行为也需要反过来向滚动条发送信息。它用 on sendPosition处理程序完成这个任务。这个处理程序需要在 on exitFrame程序的尾部——即刚刚设置角色的位置后——被调用。

```
-- this handler sends the new position of the image to the scroll bars
-- so they can adjust their markers accordingly
on sendPosition me
  horizScrollSprite = 18
  vertScrollSprite = 25
  hPercent = 1.0-float(sprite(me.spriteNum).locV-pBounds.top)/pBounds.height
  sendSprite(sprite horizScrollSprite,#setValue,hPercent)
  vPercent = 1.0-float(sprite(me.spriteNum).locH-pBounds.left)/pBounds.width
  sendSprite(sprite vertScrollSprite,#setValue,vPercent)
end
```

下面要处理的是滚动条按钮。它们使用与第 15章所描述的滑动条的箭头按钮相似的程序。这个行为需要知道每个按钮代表的是哪个方向，以及与哪个滑动条相关联。每个按钮还需要一种按下状态。

```
property pDownMember, pOrigMember -- down and normal states
property pPressed -- whether the sprite is being pressed
property pMarkerSprite -- the number of the marker sprite
property pArrowDirection -- 1 or -1 to add to slider

on getPropertyDescriptionList me
  list = [:]
  addProp list, #pMarkerSprite, [#comment: "Marker Sprite",¬
    #format: #integer, #default: 0]
  addProp list, #pDownMember, [#comment: "Arrow Button Down Member",¬
    #format: #bitmap, #default: " "]
  addProp list, #pArrowDirection, [#comment: "Arrow Direction",¬
    #format: #symbol, #range: [#left,#right,#up,#down], #default: #right]
  return list
end

on beginSprite me
  pOrigMember = (sprite me.spriteNum).member
end

on mouseDown me
  pPressed = TRUE
  (sprite me.spriteNum).member = member pDownMember
end

on mouseUp me
  liftUp(me)
end

on mouseUpOutside me
  liftUp(me)
end

on liftUp me
  pPressed = FALSE
  (sprite me.spriteNum).member = member pOrigMember
```

```
end

on exitFrame me
  if pPressed then
    sendSprite(sprite pMarkerSprite, #moveMarkerOne, ¬
     pArrowDirection)
  end if
end
```

这就是滚动条程序的全部内容。这其中有很多复杂的处理程序。大部分复杂情况与分析滚动条滑动条和图像本身的移动界限有关。

我们可以很容易地舍弃两个滚动条中的一个，而只在单方向上滚动图像。也可以禁止用户拖动图像，只允许用户使用滚动条这一种手段。

还要记住的是滚动条的图形完全是由我们自己确定的。本例所使用的滚动条很像标准的滚动条，不过，我们可以制作更适合特定界面风格的更新奇的滚动条。

29.4 对大幅图像变焦缩放

让用户能够看到大幅图像的各个局部的第三种方法是让他们先看到全部图像，然后再点击图像上的某一点，把图像放大观察。另外还可以设置一个缩小的内容完整的图像。这种方法对地图来说尤其有用。

在这类程序里，用户首先看到的是图 29-3这样的画面。在屏幕上能看到整幅图像。当用户在某处点击后，将看到放大至真实尺寸的画面。这时我们需要使用一些额外的光标，以便让用户能够了解当前正处于什么状态。

图29-3 当用户点击图像时，它被放大至真实尺寸

当用户在图像里的某处点击后，图像被放大至真实尺寸。用户点击的地方落在屏幕的中心(如果点击位置靠近图像边缘，则需要做一些调整)。然后，用户还可以像从前一样点击和拖动鼠标。要返回图像的变焦缩小状态，可以按着 Shift键再点击鼠标。

实现这种功能的行为是建立在本章的 29.2节“ 移动观察大幅图像 ”里所介绍的移动观察图像行为的基础之上的。在这里，行为需要顾及到两种模式。一是变焦缩小后的完整图像，用户点击它就可以把它变焦放大；二是真实尺寸的图像，用户可以点击并拖动它，或按下 Shift键时点击鼠标，把图像变焦缩小。

```
property pMode, pDrag, pOffset, pBounds, pScreen, pOrigRect, pCursor

on beginSprite me
  pDrag = FALSE
  pMode = #zoom

  -- get rectangle of display area
  spr = sprite(me.spriteNum)
  mem = sprite(me.spriteNum).member
  pScreen = sprite(me.spriteNum-1).rect
  pBounds = rect(0,0,0,0)
  pBounds.left = pScreen.right - (mem.width - mem.regPoint.locH)
  pBounds.right = pScreen.left + (mem.regPoint.locH)
  pBounds.top = pScreen.bottom - (mem.height - mem.regPoint.locV)
  pBounds.bottom = pScreen.top + (mem.regPoint.locV)

  -- remember original "zoomed out" rect
  pOrigRect = sprite(me.spriteNum).rect
end
```

程序需要使用全局变量gOrigRect来记录在使用该行为前角色在舞台上的状态。这个原始状态就是变焦缩小后的图像。我们需要在舞台上把该角色缩小直至与显示区域的尺寸相适配。如果需要，横向与纵向的缩放倍率可以不同。这两种效果都可以用该行为实现。

下面一个处理程序处理所有点击鼠标的操作。如果 pMode属性是#zoom，它使用mapStageToMember函数来确定图像里被点击的确切的一点。然后把这点放在舞台的中心，即坐标为(240, 160)的地方。然后它做一些检测，以确定图像处于可拖动的边界之内，并把pMode改为#drag。

如果pMode开始就是#drag，pDrag属性将被设置为TRUE，而且将计算pOffset属性。on exitFrame处理程序负责处理拖动操作，直至鼠标被释放。

当该处理程序遇到pMode设置为#drag，且用户按着Shift键点击鼠标的情况时，它将把图像恢复成原始尺寸。

```
-- when the user clicks, perform one of three actions
on mouseDown me

  -- zoom in on location
  if pMode = #zoom then

    -- get real point clicked
    clickPoint = ¬
      mapStageToMember(sprite(me.spriteNum), the clickLoc)

    -- set sprite to full size
    mem = sprite(me.spriteNum).member
    sprite(me.spriteNum).rect = mem.rect

    -- center on the point clicked
    newloc = point(240,160) + ¬
       sprite(me.spriteNum).member.regPoint - clickPoint
    if newloc.locH < pBounds.left then newloc.locH = pBounds.left
    if newloc.locH > pBounds.right then newloc.locH = pBounds.right
    if newloc.locV < pBounds.top then newloc.locV = pBounds.top
```

```
    if newloc.locV > pBounds.bottom then newloc.locV = pBounds.bottom
    sprite(me.spriteNum).loc = newloc

    -- now enable dragging
    pMode = #drag

    -- zoom out to original view
  else if pMode = #drag and the shiftDown then

    -- set to original rect
    sprite(me.spriteNum).rect = pOrigRect

    -- ready for next zoom
    pMode = #zoom

    -- drag
  else if pMode = #drag then

    -- start drag
    pDrag = TRUE
    pOffset = the clickLoc - sprite(me.spriteNum).loc

  end if
end
```

on mouseWithin处理程序和辅助性的on changeCursor处理程序与前面两节里的内容相同。不过，还需要另外两个光标。当图像为变焦缩小状态时，光标应当是带“ + ”符号的放大镜，这意味着用户点击图像就可以变焦放大；当图像为真实尺寸，且用户正按着 Shift键时，光标应当是带“ - ”符号的放大镜，这意味着用户可以变焦缩小图像。下面是完成这个任务的程序：

```
-- when the cursor is over the image, see if a cursor is needed
on mouseWithin me
  if the rollover = me.spriteNum then
    if pMode = #zoom then
      changeCursor(me,302) -- magnifying glass with +
    else if pMode = #drag and the shiftDown then
      changeCursor(me,303) -- magnifying glass with -
    else if pMode = #drag and pDrag = TRUE then
      changeCursor(me,290) -- closed hand
    else if pMode = #drag then
      changeCursor(me,260) -- opened hand
    end if
  else
    changeCursor(me,0)
  end if
end

-- if cursor leaves image area, reset it
on mouseLeave me
  changeCursor(me,0)
end

-- change the cursor only if not the same as last time
on changeCursor me, cursorNum
```

```
  if cursorNum <> pCursor then
    pCursor = cursorNum
    cursor(pCursor)
  end if
end
```

on mouseUp、on mouseUpOutside和on exitFrame处理程序与前面几节所使用的处理程序是相同的。

```
-- end dragging
on mouseUp me
  pDrag = FALSE
end
on mouseUpOutside me
  pDrag = FALSE
end

-- set the new location if dragging
on exitFrame me
  if pDrag then
    newloc = the mouseLoc - pOffset

    if newloc.locH < pBounds.left then newloc.locH = pBounds.left
    if newloc.locH > pBounds.right then newloc.locH = pBounds.right
    if newloc.locV < pBounds.top then newloc.locV = pBounds.top
    if newloc.locV > pBounds.bottom then newloc.locV = pBounds.bottom
    sprite(me.spriteNum).loc = newloc
  end if
end
```

可以删除这个“变焦缩放”和“移动观察”行为里的“移动观察”部分，让用户只能点击鼠标变焦放大观察，再点击鼠标变焦缩小。也可以用 on changeCursor把一些说明性的文字放在舞台上的文本演员里。例如，当用户处于 #zoom模式时，可以显示“Click on the image to zoom in (在图像上点击从而变焦放大)”；当用户处于#drag模式时，可以显示“Click and drag the image to pan; Shift+click to zoom out (点击并拖动图像可以移动观察图像；按着 Shift键点击图像可以变焦缩小)”。

参见第15章里的15.5节“创建滑动条”，可以获得更多有关制作滑动条的信息。

29.5 使用油墨和颜色控制

在大多数情况下，图像在 Director里显示的是它的原始效果。不过，由于能够使用各种油墨以及改变前景色和背景色，我们可以用各种方法改变图像的效果。

这种方法的实用性不是一句话就能说清的。例如， Lighten和Darken油墨可以把图像改变为各种颜色。为什么要制作这种效果呢？原因之一是使我们能够多次使用该图像，而它每次的效果都有所不同。

当我们处理图像的颜色时，有时很难找到想要的那种颜色。此时就可以使用如图 29-4所示的程序了。它使我们能够改变角色的 color和bgColor属性里的红、绿和蓝成份了。这些颜色变化对使用Lighten和Darken油墨的图像有很大影响。

这个程序从本书前面的内容里借用了大量程序。滑动条使用了第 15章的滑动条行为，两个单选按钮也使用了第15章的行为。

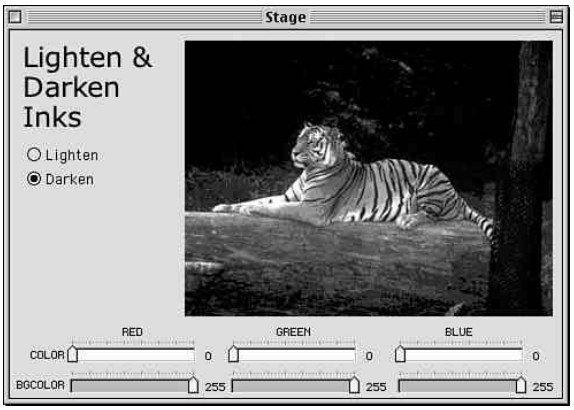

图29-4 该程序使我们能够调节用于Lighten和Darken油墨的颜色

图像本身有一个简单的行为。它可以读取单选按钮和滑动条的值，并相应地设置角色的油墨和颜色。它频繁地执行这个操作，因此在我们进行调节时，就能看到变化。

```
on exitFrame me
  -- read radio buttons to set ink
  if selected(sprite 3) = 3 then
    ink = 40
  else
    ink = 41
  end if
  sprite(me.spriteNum).ink = ink

  -- read first set of sliders to set color
  c = rgb(0,0,0)
  c.red = sprite(7).pValue
  c.green = sprite(12).pValue
  c.blue = sprite(17).pValue
  sprite(me.spriteNum).color = c

  -- read second set of sliders to set bgcolor
  c = rgb(255,255,255)
  c.red = sprite(23).pValue
  c.green = sprite(28).pValue
  c.blue = sprite(33).pValue
  sprite(me.spriteNum).bgColor = c
end
```

尽管颜色的变化对其他大多数油墨的影响不很显著，但仍旧可以通过试用更多种油墨，把这类程序进一步扩展。

还可以设置一个按钮，它使我们能够为正在显示的演员输入一幅新图像。可以使用 FileIO Xtra或MUI Xtra得到图像的文件名，再设置该图像的filename属性，从而把它输入进来。但是，这个演员起初就应当属于链接的图像，而不是内部图像。

对这个程序的另一种处理是用它创建我们自己的 Xtra。把这个影片放到Director的Xtras文件夹里可以实现这个目的。可以向其中添加一些有趣的功能，如让它使用 selection of castLib属性，看看用户是否选择了一个位图，并用它的 picture属性替换Xtra里的图像的picture属性。这样，我们可以在演员表里选择一幅图像，然后运行这个 Xtra，它将显示所选定的图像的一

个拷贝，并允许我们调节它的油墨和颜色。

参见第10章“角色和帧的属性”里的10.4节“角色的油墨”，可以获得关于角色的颜色的背景知识。

参见第10章里的10.6节“角色的颜色”，可以获得关于角色的颜色的背景信息。

参见第10章里的10.5节“角色的混色”，可以获得关于角色的混色的背景信息。

参见第15章里的15.5节“创建滑动条”，可以获得关于制作滑动条的更多信息。

参见第15章里的15.3节“使用单选按钮”。

29.6 图形图像软件的故障排除

■ 如果幻灯片找不到图像文件，可能是由于路径名没有被正确编辑。试着在每次设置filename属性时用put命令把路径名放在消息窗口里。这样就能够检查它了。

■ 在幻灯片里，需要让图像演员开始就与某幅图像链接。在发行最终产品时，要确保该图像存在，否则当运行影片时，将出现一个错误信息对话框。把文件名链接至幻灯片的第一幅图像是一种很好的方法，也可以让它链接到某个很小的占位用的图像文件。

■ 当为滚动图像行为创建滚动条的图形时，背景图像的放置将决定滑动条的运动范围。因此对它的最大值和最小值都要进行测试。

■ 并不是所有的油墨都能被角色的color和bgColor属性所影响。如果图像的位深大于或等于8-bit，对于大多数油墨来说，这些设置对图像的影响都不明显。

29.7 你知道吗

■ 制作幻灯片的另外一种方法是把所有图像都放在一个外部演员表里。然后可以用更换演员表的方法得到不同的幻灯片。在 Web和放映机上都可以使用这种方法。这种方法还可以使他人很难剽窃其中的图像，因为它们并不是普通的 JPEG或GIF等格式的图像。

■ 要把滚动条的行为做得更完整，应当允许用户能点击滚动条本身。如果用户在滑动条的前面点击，则向前翻页；如果用户在滑动条的后面点击，则向后翻页。滚动条通常都具有这种特性。