第18章 控 制 位 图

Director提供了许多命令、函数和属性，用以修改舞台上的位图角色的外观。 Director 7实际上增添了许多以前版本中没有的新的功能，如旋转和变形。这些功能就是本章的主要内容。

18.1 角色的变形

目前，我们可以使用loc、locH和locV属性来改变角色的位置。还有许多其他属性可以用来确定位图的外观。

18.1.1 rect属性

rect属性曾在第15章“ 图形界面元素 ”的滑动条和进程条行为中被使用，以重新定义矩形的形状。这个属性也能够用来拉伸压缩角色。

例如，如果我们在舞台上放一个角色，并使用消息窗口，可以查看角色的初始 rect属性。下列是一个例子：

```
put (sprite 1).rect
-- rect(200, 150, 400, 400)
```

然后，我们可以把rect属性设置为其他值。它甚至不需要与原来的长宽比例相同。例如，可以仅改变rect的左边和右边的值，就能够压缩角色的横向尺寸。像下面这样试验一下：

```
(sprite 1).rect = rect(250,150,350,400)
updateStage
```

rect属性实际上控制了loc属性的所有功能，而且它的作用还不仅于此。既然我们控制了角色的四个边的位置，就能够移动、拉伸和压缩角色。该功能可以用于所有位图角色，还可以用于图形和矢量图形。它甚至对将文本和域演员限制在有限的范围内也是有效的，只要演员的帧属性没有设置成“ Adjust To Fit ”，就能够对它们进行调节。

注释 第一个数字表示矩形左边的位置；第二个表示矩形顶边的位置；第三个表示右边位置；最后一个是底边位置。所以依次是左、顶、右、底。

18.1.2 rotation属性

Director 7的一个新的强大功能就是旋转角色的功能。有一整套 Lingo命令能够控制角色的rotation属性。位图和文本能够被旋转，但图形则不能。

rotation值是以度(°)为单位的，范围为0~360，但是我们能够使用超出此范围的数据，因为Director能够把这些值转换到这个范围内。一个新角色的 rotation属性的值总是被设置成0.0。

```
put (sprite 1).rotation
-- 0.0000
```

对rotation属性的设置非常容易：

```
(sprite 1).rotation = 45
```

```
updateStage
```

下面是一个只有一行语句的行为，它使角色绕着自己的中心旋转，每一帧旋转 1度。

```
on exitFrame me
  (sprite 1).rotation = (sprite 1).rotation + 1
end
```

还有一种更复杂的行为，它旋转角色，使之总是指向光标。这种行为设想位图的“指针”一般是指向舞台的右边。下面一段程序为它重新定向，使之指向光标。

```
on exitFrame me
  -- get the mouse location
  p = the mouseLoc
  x1 = p.locH
  y1 = p.locV

  -- get the sprite location
  x2 = (sprite me.spriteNum).locH
  y2 =  (sprite me.spriteNum).locV

  -- use atan to compute the angle
  angle = atan(float(y2-y1)/float(x2-x1))

  -- correct the angle
  if x1 < x2 then angle = pi()+angle

  -- convert the angle to degrees
  angle = angle*360.0/(2.0*pi())

  -- set the sprite
  (sprite me.spriteNum).rotation = angle
end
```

也可以很容易地把该行为用于其他的点。例如，它能让角色向另一个角色看齐。当那个角色在用户的交互式操作下移动或根据影片的设计动作而移动时，被控制的角色也随之变化。

atan函数是该行为的核心。它的作用是把由两点构成的斜线换算成一个角度值。因为要把得到的角度值限制在半圆内，所以做了一个调节，以把角度都处理在圆的左边。这样计算出的角度的单位是弧度，所以还要先将它转换为角度，再应用于 rotation属性。

注释 如果你还记得中学的三角几何知识，就会知道弧度是描述角度大小的另一种方法。一个圆是360°，表示成弧度则是2π或6.2832弧度。

18.1.3 flipH和flipV属性

位图角色也能沿着横轴或纵轴镜像翻转。镜像翻转围绕演员的套准点进行。 Lingo语言通过flipH和flipV属性控制这个功能。它们可以被设置成 TRUE或FALSE。如果设置为TRUE，则角色被镜像翻转显示。

下面是一个例子。此例子沿横向镜像翻转角色：

```
(sprite 1).flipH = TRUE
updateStage
```

然而，如果角色已经镜像翻转，上面这个例子就不起作用了。下面是一个将角色的横向镜像翻转反向的例子：

```
(sprite 1).flipH = not (sprite 1).flipH
updateStage
```

下面的行为根据光标与角色的相对位置控制角色的镜像翻转属性。无论何时光标跨过角色的某个轴，该角色就发生镜像翻转：

```
on exitFrame me
  -- get the cursor location
  cLoc = the mouseLoc

  -- get the sprite location
  sLoc = (sprite me.spriteNum).loc

  -- flip horizontally if needed
  if cLoc.locH > sLoc.locH then
    (sprite me.spriteNum).flipH = TRUE
  else
    (sprite me.spriteNum).flipH = FALSE
  end if

  -- flip vertically if needed
  if cLoc.locV > sLoc.locV then
    (sprite me.spriteNum).flipV = TRUE
  else
    (sprite me.spriteNum) .flipV = FALSE
  end if
end
```

18.1.4 skew属性

skew属性也能用Lingo语言设置。其结果是在角色矩形的垂直边发生某个角度的变化。从0到90°的角度使垂直边向右边倾斜，从 0°到-90°的角度向左边倾斜。角度在 90°~180°~-90°~ -180°范围时显示镜像翻转角色的效果。 360°的变化将扭曲复原到角色初始的状态，0°旋转也是一样。

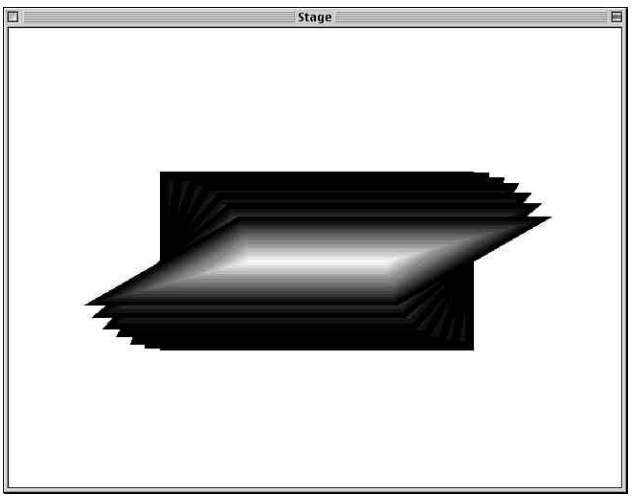

图18-1 以10°的间隔多次扭曲角色的效果，trails属性已被打开，这样就记录了每一步扭曲的过程

下面是一个将角色重复地扭曲10°的简单行为：

```
on exitFrame me
  (sprite 1).skew = (sprite 1).skew + 10
end
```

该行为的结果显示在图18-1中。

18.1.5 quad属性

最后一个并且也是功能最强的角色处理属性是角色的 quad属性。它像rect一样，是一个包含四项的数据列表，但它包含的是点的坐标，而不是数字。其中每一个点表示角色的一个角，下列是一个例子：

```
put (sprite 1).quad
-- [point(184.0000, 126.0000), point(463.0000, 126.0000), point(463.0000, ¬
302.0000), point(184.0000, 302.0000)]
```

quad属性的便利之处在于无论我们想怎样设定四个点都可以。我们基本上能够使用 Lingo语言“拖动”任一个角。

例如，如果我们有一个矩形图像，然后将角色的左上角向右下方推进；并将角色的右上角向左下方推进。这样我们能够在舞台上模仿透视感并创建 3D效果。图18-2显示了这个效果。

注释 quad和skew之间的不同在于用quad时我们可以完全控制一个角色的四个角。而用skew时，我们只能控制角色的矩形边的倾斜角度。

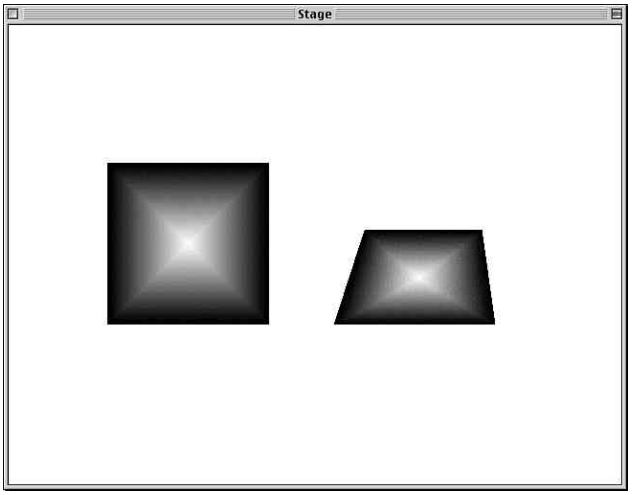

图18-2 舞台显示了两个角色，它们拥有同样的位图演员。然而，其中一个角色的quad属性已经被改变以模仿3D的透视效果

我们没有办法在舞台上改变角色的 quad属性，但是当影片播放时，下面的这个处理程序帮助我们能够达到这个目的。它使我们能够点击并抓住角色的任一角，并拖动该角：

```
property pCorner

on beginSprite me
  pCorner = 0
end
```

```
on mouseDown me
  pCorner = 0
  repeat with i = 1 to 4
    if distance(me,the clickLoc,sprite(me.spriteNum).quad[i]) < 10 then
      pCorner = i
      exit repeat
    end if
  end repeat
  put pCorner
end

on mouseUp me
  pCorner = 0
end

on mouseUpOutside me
  pCorner = 0
end

on exitFrame me
  if pCorner > 0 then
    q = sprite(me.spriteNum).quad
    q[pCorner] = the mouseLoc
    sprite(me.spriteNum).quad = q
  end if
end

on distance me, p1, p2
  return sqrt (power(p1.locH-p2.locH,2)+power(p1.locV-p2.locV,2))
end
```

18.1.6 其他变形属性

其他许多属性没有前面讨论的属性对位图角色的影响那么大。许多属性是 Director 7中出现的新内容。下面是一个完整的清单：

■ useAlpha —— 确定在显示演员时是否使用 32-bit位图的alpha通道里的信息。当该属性为TRUE时，alpha通道将用来改变位图的透明度。

■ alphaThreshold —— 如果有一个alpha通道存在，此属性能被设置成 0~255。0表示角色里的所有点都能够对鼠标点击有所反应。其他的值表示像素对点击有所反应所需要的透明度。

■ blend —— 与剪辑室中的blend属性相符。我们能够将数值设置在 0~100之间，但是角色的油墨必须是支持混色效果的那种油墨。

■ color —— 角色的前景颜色。它能被设置成 paletteIndex结构，如paletteIndex(255)，或一个color结构，如color(255,255,255)或color("#FFFFFF")。当1-bit演员的所有已使用的像素都使用color 属性时，差异非常明显。它也用于被设置为 Lighten和Darken油墨的角色。

■ bgColor —— 角色的背景色。它能被设置成 paletteIndex结构，例如paletteIndex(255)，或一个color结构，例如color(255,255,255)或color("#FFFFFF")。当1-bit演员的所有未使用的像素都使用color 属性时，差异非常明显。它也用于被设置为 Lighten和Darken油墨的角色。

参见第3章“ 位图演员 ”里的3.4节“ 位图演员的属性 ”，以得到关于位图的信息。

参见第10章“ 角色和帧的属性 ”里的 10.4节“ 角色的油墨 ”，以得到更多关于油墨的信

息。

参见第10章里的10.5节“角色的混色”，以得到更多关于角色混合的信息。
参见第10章里的10.6节“角色的颜色”，以得到更多关于角色颜色的信息。
参见第10章里的10.7节“角色的形状”，以得到更多关于角色形状的信息。
参见第10章里的10.8节“角色的其他属性”，以得到更多关于角色属性的信息。

18.2 添加3D效果

Director不是3D编辑环境，但实际上，计算机 3D程序也不是真正的3D环境。因为计算机屏幕本身是平面的。从本质上说， 3D计算机图形只是让用户在平面的屏幕上得到 3D的错觉。这一节讨论如何使用Lingo语言使这些错觉尽可能逼真。

18.2.1 缩小角色

缩小角色的错觉是 Director和Lingo语言十分容易处理的一种特殊效果。基本的想法是物体离眼睛越远，看上去就越小。所以，如果我们有两个相同尺寸的物体，但是其中一个离我们有两倍远，那么远处的那个看上去只有一半大。

图18-3显示有两个角色的舞台。左边的角色显示为 100%，而右边的角色为50%显示。效果是其中一个看上去比另一个近。

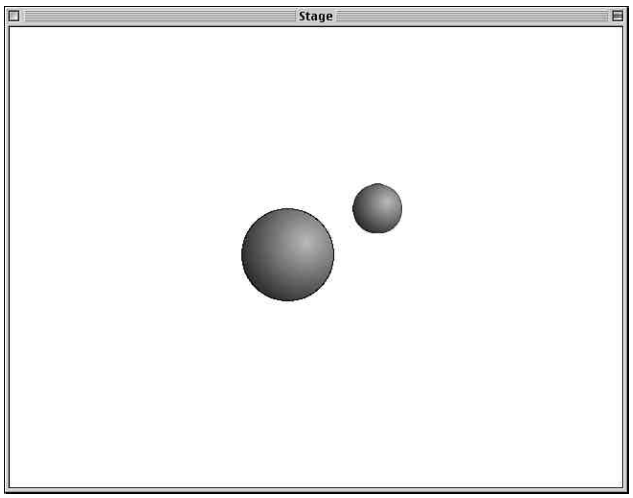

图18-3 设置角色的大小能够创造深度错觉

当用在动画里时，错觉更加强烈。例如，想象一下一个物体首先显示成一个小点。然后增大到完全的尺寸。它看起来就像物体正向观察者移动。

下面的程序就是实现这个过程的行为。它设定屏幕中心为 320，240，这也是角色的原始位置。它获得该角色的 quad属性，并按比例并相对于屏幕的中心来对这四个点进行调整。像下面这样，当比例达到1.0，就停止：

```
property pOrigQuad, pScale

on beginSprite me
  pOrigQuad = sprite(me.spriteNum).quad
```

```
  pScale = 0.0
end

on exitFrame me
  if pScale < 1.0 then
    -- increase scale 1%
    pScale = pScale + .01

    -- copy the quad
    newQuad = duplicate(pOrigQuad)

    -- set new quad points
    repeat with i = 1 to 4
      newQuad[i] = newPoint(me,newQuad[i],pScale)
    end repeat

    -- set the sprite
    sprite(me.spriteNum).quad = newQuad
  end if
end

-- this handler will take a point and a scale
-- and return a new point based on the
-- center of 320, 240
on newPoint me, p, scale
  centerPoint = point(320,240)

  -- find relative location
  p = p - centerPoint

  -- multiply by the scale
  p = p*scale

  -- add back center point
  p = p + centerPoint

  return p
end
```

像这样的剧本能用于制作物体在屏幕上从一定距离移到它们本身位置的运动。在创建向用户飞来的文本时，它也能产生一个好的效果，而不只是向左、右、上或下移动。

18.2.2 在移动过程中缩小角色

创建深度感的另一方法是当角色在舞台上移动时，改变它的尺寸。例如，假定舞台的左边比右边离我们更远，我们根据角色离舞台左边的距离来缩小这个物体。

下面的程序显示了角色的中心紧跟鼠标变化的行为。用户越把角色引向舞台的左边，角色变得越小：

```
property pWidth, pHeight

on beginSprite me
  pWidth = sprite(me.spriteNum).rect.width
  pHeight = sprite(me.spriteNum).rect.height
```

```
end

on exitFrame me
  -- get the new location
  x = the mouseH
  y = the mouseV

  -- figure the size as a number between 0 and 1
  if x = 0 then  percent = the maxInteger
  else percent = 640.0/x

  -- figure the new height and width
  w = pWidth/percent
  h = pHeight/percent

  -- make a rectangle with the point at the center
  newRect = rect(x-w/2,y-h/2,x+w/2,y+w/2)

  -- set the sprite
  sprite(me.spriteNum).rect = newRect
end
```

像这样的错觉，在背景图形中制造一些暗示会更容易产生。在前面的例子中，我们可以增加一个向左收缩的墙的背景图形。

18.2.3 使用quad属性创建错觉

然而，创建3D错觉的另一种方法，是使用角色的 quad属性，用2D图形表示3D效果。立方体是一个很好的例子。图 18-4显示了舞台上的一个立方体。它实际由 6个角色绘制而成，每个角色表示一个面。它用3D三角学方法和quad来决定每个面向哪个方向变形。

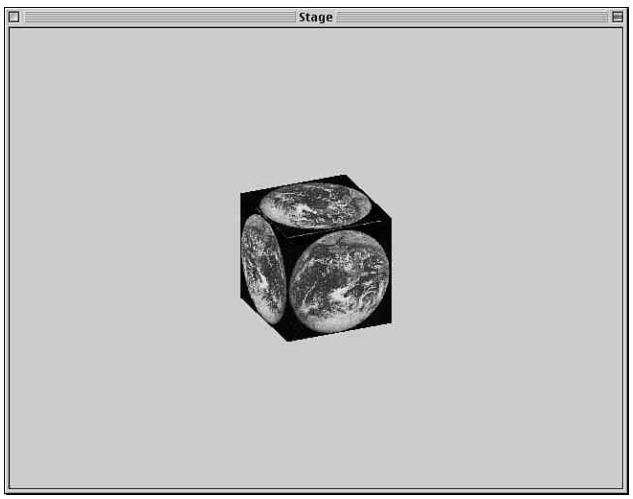

图18-4 用于绘制3D立方体的6个角色

下面是构造3D立方体的代码。实际上有许多不同的方法能够实现这种效果，但这个例子使用的是下面的影片剧本：

注释 你可能认为这些代码十分复杂。如果不能正确理解它到底在做些什么，别着急。试试CD-ROM上的演示文件，可以更好地理解这个例子。

```
global gCorners, gCenter, gRectList, gRotate, gPlane

on startMovie
  -- initialize lists for a cube
  initBox
end

on frameScript
  -- add to rotation based on mouse location
  gRotate = gRotate - (float(the mouseH-320)/30)*pi()/100

  -- calc plane tilt based on mouse location
  gPlane = - (float(the mouseV-240)/30)*pi()/20
  drawSides
end

on initBox
  -- list of corners with x, y and z coordinates
  gCorners =[[-60,-60,-60], [60,-60,-60], [60,60,-60],¬
    [-60,60,-60], [-60,-60,60],[60,-60,60],[60,60,60],[-60,60,60]]

  -- the screen center
  gCenter = point(320,240)

  -- list of sides
  -- each side has four corners
  gRectList = [[1,2,3,4],[1,2,6,5],[3,4,8,7],[2,3,7,6],¬
    [1,4,8,5],[5,6,7,8]]
end

on drawSides
  -- generate a list of screen points and depths based on the
  -- gCorners list and the transformation to 2d screen coordinates
  list = []
  repeat with i = 1 to count(gCorners)
    temp = plotPoint(gCorners[i])
    add list,temp
  end repeat

  -- create a quad list that takes 4 points to display side of cube
  quadList = [:]
  repeat with i = 1 to count(gRectList)

    -- get the four corners that make this side
    thisRect = gRectList[i]

    -- get the four screen points to draw
    q = [list[thisRect[1]][2],list[thisRect[2]][2],¬
        list[thisRect[3]][2],list[thisRect[4]][2]]

    -- get the closest (depth) screen point
    z = min(list[thisRect[1]][1],list[thisRect[2]][1],¬
```

```
            list[thisRect[3]][1],list[thisRect[4]][1])

    -- add to list, with closest screen point as the property name
    addProp quadList, z, q
  end repeat

  -- sort list by property name so closest
  -- sides get drawn on top of farther ones
  sort quadList

  -- draw each side
  repeat with i = 1 to count(gRectList)
    sprite(i).quad = quadList[i]
  end repeat
end

on plotPoint objectInfo
  -- get x, y, and z from objectInfo list
  x = getAt(objectInfo,1)
  y = getAt(objectInfo,2)
  z = getAt(objectInfo,3)

  -- TRANSFORM BY ROTATION AROUND Z

  -- compute the radius
  radius = sqrt(x*x+y*y)

  -- compute the angle
  if x = 0.0 then angle = atan(the maxInteger)
  else angle = atan(float(y)/x)
  if y < 0 then angle = angle + pi()

  -- rotate
  set angle = angle+gRotate

  -- compute new x, y, and z
  realX = radius*cos(angle)
  realZ = radius*sin(angle)
  realY = Z

  -- TRANSFORM BY ROTATION AROUND X

  -- compute then radius
  radius = sqrt(realY*realY+realZ*realZ)

  -- compute the angle
  if realZ = 0 then angle = atan(the maxInteger)
  else angle = (atan(realY/realZ))
  if realZ < 0 then angle = angle + pi()

  -- rotate
  angle = angle - gPlane

  -- compute then new x, y and z
  screenX = realX
```

```
  screenY = radius*sin(angle)
  screenZ = radius*cos(angle)

  -- return both z, and the x and y point
  return  [screenZ,point(screenX,screenY)+gCenter]
end
```

为了让这个影片剧本产生真正的效果，它还需要 6个正方形的位图作为前6个角色，并另加一个帧剧本：

```
on exitFrame
  frameScript
  go to the frame
end
```

此程序根据鼠标位置确定各个平面，并旋转立方体。然而，我们能够删除其中的部分代码，再设计能够让用户移动立方体的按钮。我们甚至可以让它以固定的速度旋转。

18.2.4 角色的映射

当我们使用quad属性变形角色时， Director仍然保持初始演员的外观。实际上，我们能使用Lingo语言中的两个特殊函数在屏幕位置和初始位图的像素的相对位置间转换。

图18-5显示了一个很好的例子，以说明我们为什么需要做此转换。图中的棋盘已用 quad属性对它做了变形，以让它有深度感。实际上，该棋盘是有64格的160×160的简单正方形。

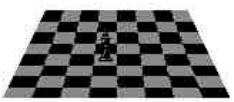

图18-5 此棋盘位图原本是正方形，它已用quad属性进行了改变，因此显得有深度效果

下面是on beginSprite处理程序，它将这个普通的方形的棋盘转换成我们在图 18-5中看见的3D形式。它只是变动了四边形的两个顶点。其中没有使用特殊的数学方法，为了简化程序，只使用了粗略估计的硬代码。

```
on beginSprite me
  q = sprite(me.spriteNum).quad
  q[1] = q[1]+point(-30,100)
  q[2] = q[2]+point(30,100)
  sprite(me.spriteNum).quad = q
end
```

当影片播放时，此角色看上去就像图 18-5所示的那样。当它只是个普通的方形时，要得到鼠标点击的行或列的位置是十分容易的。该棋盘上的每个格是 20×20，所以只要将横向位置或纵向位置除以20，再加1，以得到1~8间的一个数字。然而，随着棋盘发生如图 18-5中所示的变形，我们需要用Lingo语言映射点击位置。

用于映射点击位置的函数是mapStageToMember。它取一个角色编号和一个舞台位置为参数，然后将这两个量转换成相对于位图演员的位置。

所以，如果用户点击了变形角色的左上角，该函数返回点(0,0)。下面是on mouseUp处理程序使用mapStageToMember函数计算用户点击位置的例子：

```
on mouseUp me
  p = mapStageToMember(sprite 1, the clickLoc)
  x = 1+p.locH/20
```

```
  y = 1+p.locV/20
  put "Row: "&&x&&"Column: "&&y
end
```

与mapStageToMember函数功能相反的是mapMemberToStage函数。用它可以基于演员中某一位置计算出相应的舞台位置。下面的处理程序的功能是：当给定棋盘上的某一格时，将返回它的舞台位置。

```
on getPos me, x, y
  p = point(x*20-10,y*20-10)
  return mapMemberToStage(sprite 1, p)
end
```

为了像图18-5那样放置“王后”，可以使用下面这个行为。它假设棋盘在角色通道1中。

```
on beginSprite me
  p = getPos(sprite 1, 4, 5)
  sprite(me.spriteNum).loc = p
end
```

用这种方法映射角色，可以使某一角色产生变形，然后相对这个变形角色定位其他的角色。它能够与立方体的例子一起使用，以在立方体表面放置一个物体。它也能与压缩角色的例子一起使用，以在某个角色上映射另一个角色，或当角色的尺寸变化时，提供在角色上的有效点击。

参见第31章“Shockwave 短程序”里的31.3节“创建广告”，以观看使用3D立方体剧本的例子。

18.3 处理位图演员

我们不能用位图演员做太多的事情。它们没有像文本演员或域所具有的可编辑成分。然而我们可以用crop命令剪裁演员。

这个命令以一个演员名和一个rect属性作参数，并将剪裁后的图像放回同一个演员位置。所以，如果有一个100×100的位图，我们只要知道它的右下角坐标，就可以使用下面这个命令：

```
crop(member "mybitmap", rect(75,75,100,100))
```

rect属性中的数字定义了我们想保留的位图演员内的矩形，剩下的部分被去掉。

我们有时也可能想保留原始位图演员。做法是创建一个新的演员并复制原始位图演员，然后将图像拷贝进新的演员中。

new命令用于创建任何类型的新演员。对位图来说，只要以下面方法使用它就可以了。

```
myNewMember = new(#bitmap)
```

变量myNewMember现在是一个新演员的引用变量。我们能够用这个引用变量设置它的名字：

```
myNewMember.name = "New Bitmap"
```

如果想在演员表库中创建这个演员，而不让它作为内部演员存在，则要使用另一个参数：

```
myNewMember = new(#bitmap, castLib "pictures")
```

或者，我们可以像下面一样，用或不用某个castLib引用变量都可以确定演员的精确位置：

```
myNewMember = new(#bitmap, member 18)
```

```
myNewMember = new(#bitmap, member 18 of castLib "pictures")
```

在我们有了一个新演员之后，我们能够通过使用演员的主要属性来将信息放置其中。例如：对于域，我们可以用到text属性。

```
myNewMember.text = "Some text"
```

对文本演员，我们可以使用text、rtf或html属性。然而，对位图，我们必须使用picture属性。但是我们不能用Lingo语言创建一幅图像；所以必须从另一演员那儿获得。要从一个演员拷贝一个位图到另一个演员，像下面这样做：

```
myNewMember.picture = member("my picture").picture
```

我们也能使用media属性。此属性就像picture用于位图演员，html用于使用HTML的文本演员，rtf用于使用RTF的文本演员，sound用于声音演员等等。它仅表示演员包含的素材类型。

提示 文本演员也包含一个picture属性。这个属性表示在舞台上显示的文本位图。我们可以抓取这个图像，并将它放在位图演员中。

包含位图演员的舞台，具有一个picture属性。我们能够捕获该舞台的一个场景，并像下面一样将它存储在一个位图中：

```
member("myBitmap").picture = (the Stage).picture
```

我们也能使用crop剪裁图像。下面是一个捕获舞台图像，然后剪裁它以匹配一个矩形的简短处理程序：

```
on grabArea intoMember, screenArea
  member(intoMember).picture = (the stage).picture
  crop(member(intoMember),screenArea)
end
```

注释 如果从左到右或从上到下都是白色场景，前面的例子的结果会与想象的不同。当图像被捕获并放入演员中时，任何白色场景都被忽略。所以我们捕获rect时，必须考虑舞台rect和图像rect是不同的。

通过把舞台上的某个角色(如一个图形)的rect作为剪裁rect，我们能够很容易地得到这个矩形角色含有的部分舞台。请看下面的程序：

```
grabArea(member "myMember", sprite(1).rect)
```

本章介绍的这些属性能为我们提供对影片中位图演员和角色的极大的控制权。

18.4 位图的故障排除

■ 没有什么事做起来是不受限制的。当我们旋转、变形或拉伸角色时，要受影片的播放速度的影响。我们不能期望在舞台上有大量一直旋转的角色和其他用quad拉伸的角色时，影片仍然像没有这些东西时运行得一样快。所以，在设计时请务必记住这一点。

■ 如果一个32位的图像没有正确显示，或是在Paint窗口中和在舞台上看上去不同，试着关闭Use Alpha属性。有时，内容不正确的Alpha通道会对32位图像有影响。

■ 直接使用带有alpha通道的Photoshop文件会产生“预叠加”的alpha能道。这将在图像周围出现一个白色的光晕。校正这种效果的简单方法之一是首先在Macromedia Fireworks中打开该文件，将它存成PNG文件，然后导入那个文件。

18.5 你知道吗

■ 我们可以使用未记入正式文件的函数 getPixel，用位图中的某个点的x和y坐标，以得到位图中的一个彩色像素的整数值。例如：

```
getPixel(member("myBitmap"), 50, 100)
```

我们也能使用未记入正式文件的 setPixel改变其颜色。

```
setPixel("myBitmap"),50,100,65535)
```

因为这些函数不被支持，它们不是十分可靠，所以确定这些数字表示什么十分困难。这些我们需要使用的数字依赖于演员的位深和 alpha通道的存在。

■ 未记入正式文件的一个属性是 the useFastQuads。将它设置成TRUE并且用quad属性拉伸角色，角色绘制起来会更快一些，但拉伸效果不太相同。效果不像模拟 3D那样好。

■ 为了将角度数值转成弧度，用此数值除 360.0并乘以2.0π。将弧度数值转成角度时，用它除2.0π，然后乘360.0。要确保原始数据是浮点数，而非整数。