第13章　重要的Lingo句法

尽管Lingo关键词有几百个，但是日常使用的只有很少一部分。其中大多数是针对演员或事件的特定属性的。只要了解了其中的重要句法，就可以完成大多数任务了。

这些重要的命令可以使程序做决定、重复动作和操纵变量。当你掌握了它们，你就掌握了Lingo。其余的只是一些细节问题。

本章简要地介绍一些重要的命令、功能和属性，其中带有一些简单的实例。你读完本章后就可以独立制作实用的Lingo剧本了。

13.1　使用数字变量

在上一章我们看到了Lingo可以进行简单的数学运算。不过，使用数字变量还有许多其他方法。同其他任何编程语言一样，Lingo也有一整套处理数字的命令和函数。

13.1.1　执行运算

可以对数字和变量进行加、减、乘和除运算，所使用的符号为“ + ”、“ - ”、“ * ”和“ / ”。加号是唯一名符其实的符号；连字符代替了减号；星号用作乘号，在大多数编程语言中它都是这样使用的；斜杠是除号，因为标准键盘上没有除号。

使用这些符号的方法同我们平时在纸上写算式一样。以下是在消息窗口内的一些实例：

```
put 4+5
-- 9
put 7-3
-- 4
put 6*9
-- 54
put 8/2
-- 4
put 9/2
-- 4
```

13.1.2　整数和浮点数

请注意上面最后一个算式，9除以2，返回的值是4。这是因为Director在这里只考虑整数。整数可以是正数或负数，但没有小数部分。4是一个整数，而4.5不是。

带有小数部分的数字称为浮点数。这个名称指的是计算机存储这些数值的方法，它比存储整数的方法复杂得多。4.5即为一个浮点数，简称为浮点。

请注意，浮点数包含整数。4是整数，也是浮点数。但4.5是浮点数而不是整数，因为它有小数部分。

在Lingo中，浮点数总是带有小数点。这样，Director才能区分这两种数字。

如果让Director做9/2这样的运算，它将查看所有参与运算的数字，并决定计算结果应该是整数还是浮点数。在本例中，因为我们让它做的是两个整数间的运算，因此它返回一个整数。

为了实现这一点，该数字的小数部分被舍弃，而只保留了整数部分。因此9/2返回4，而不是4.5。

注释 当Director做这类除法时，不会四舍五入。小数部分直接被舍弃。因此7/4将返回1，而不是2。

我们可以通过把参与计算的某个数字写为浮点数，而强迫 Director返回浮点数。在消息窗口内试做以下运算：

```
put 7/4
-- 1
put 7.0/4
-- 1.7500
put 7/4.0
-- 1.7500
put 7.0/4.0
-- 1.7500
```

我们还可以使用两个函数把整数和浮点数相互转换。函数（ funtion）是一个Lingo词汇，它通过对一个或多个数值进行计算，最终返回一个数值。在消息窗口内试做以下运算：

```
put 7
-- 7
put float(7)
-- 7.0000
put 7.75
-- 7.7500
put integer(7.25)
-- 7
put integer(7)
-- 7
put integer(7.75)
-- 8
```

integer和float函数可以把任意数值(无论是整数还是浮点数)转换成我们想要的类型。可以注意到，对整数使用integer函数将不会改变它。也可以注意到， integer函数对数字进行四舍五入处理，因此7.75将得到整数8。

在运算内也可以使用函数。在消息窗口内试做以下运算：

```
put 8/5
-- 1
put float(8)/5
-- 1.6000
put float(8/5)
-- 1.0000
put integer(float(8)/5)
-- 2
```

请注意put float(8)/5与put 8.0/5是等价的。还可以注意到第三个例子的运算结果似乎错了。它说8/5的浮点数结果是1.000。这是因为在转换为浮点数之前，我们先让它计算 (8/5)。从第一个例子里可以看出， 8/5值为1。因此该行只是把1转换成为1.000。

最后一个例子演示了如何同时使用两个函数执行运算，并得到四舍五入的结果。除法运算得到了一个浮点数，随后又用 integer函数进行了四舍五入运算。

13.1.3 优先权

最后两个例子引出了一个有趣的话题：运算的顺序 (也称优先权)。当执行多种运算，或运算与函数结合时，先执行哪种运算呢？在消息窗口内试做以下运算：

```
put 5+3
-- 8
put 8*6
-- 48
put 5+3*6
-- 23
```

5加3是8，8乘以6是48。但如果把这些都写一行里，却得到 23，为什么呢？这是因为乘、除运算优先于加、减运算。因此在运算式 5+3*6里，先做乘法，然后做加法。大多数编程语言都是这样的。

但是，我们可以通过指定先进行哪种运算，而改变原来的优先次序。用括号把运算组合起来就可以了。在消息窗口内试做以下运算：

```
put 5+3*6
-- 23
put (5+3)*6
-- 48
```

使用括号后可以先做加法，再用和值乘以 6。如果需要，还可以使用多层括号。

```
put ((5+3)*6+(7-3)*4)*8
-- 512
```

13.1.4 运算和容量

现在我们了解了数学运算的所有基本规则，下一步将了解如何使用变量。在数学运算中对变量的解释与在前面的例子中对数字的解释是一样的。

```
myNumber = 5
put myNumber
-- 5
put myNumber+1
-- 6
put myNumber*7
-- 35
put (myNumber+2)*3
-- 21
```

也可以让变量与某个算式的运算结果相同。

```
myNumber = 7+4
put myNumber
-- 11
myNumber = (7+4)*2
put myNumber
-- 22
myNumber = 5
myOtherNumber = 6
mySum = myNumber + myOtherNumber
put mySum
-- 11
```

在最后一个例子中，一个变量等于另外两个变量的运算结果。对变量进行运算，并把结果存储在其他变量里，这在Lingo和其他编程语言中是很常见的。在本例中，变量里所包含的内容显而易见，似乎只用数字就可以了，而且更合理。但在实际编程过程中，变量的值总是在变化的，如游戏中的得分或用户的年龄。

13.1.5 函数

除了integer和float函数外，还有很多对数字进行处理的函数。表 13-1列出了这些函数及其功能。

表13-1 Lingo数学函数

函 数	描 述	实 例
abs	返回数字的绝对值。它只是去掉小于 0的数字的负号	abs(-7) = 7
atan	返回数字的反正切值。它使用弧度系统，而不使用角度	atan(1.0) = 0.7854
cos	返回数字的余弦值。它使用弧度系统，而不使用角度	cos(3.14) = -1.000
exp	返回自然对数的底数(2.7183)的n次方(n为给定的数值)	exp(3) = 20.0855
float	把数字转换为浮点数	float(4) = 4.000
integer	用四舍五入法把浮点数变成整数。对整数使用该函数，整数将保持原状	integer(7.8) = 8
mod	用来把数字限制在一个范围内。该限制值总是从 0起，到这个指定的数字	4 mod 3 = 1.
sqrt	返回数字的平方根。如果给定的是整数，它将对结果进行四舍五入，以得到最接近的整数。如果给定的是浮点数，将返回一个浮点数	sqrt(4) = 2
sin	返回数字的正弦值。它使用弧度系统，而不是角度	sin(3.14/2) = 1.000

这些函数中的多个都是三角函数。在用 Lingo定义曲线或环形的动画路径时，三角函数非常有用，但水平较高的Lingo程序员才会使用它。不必担心你已忘记了高中的数学知识，因为在大部分程序中用不到它们。

参见第18章“控制位图”中的18.1.2节“Rotation属性”，可以获得更多有关三角函数的信息。

13.2 使用字符串变量

另外一种基本类型的变量是字符串，它存储的是一系列字符。字符串可以是简单的单个字符，也可以是复杂一些的单词、一行文本或一页文本，甚至可以没有字符。以下的例子给出了消息窗口中的不同类型的字符串：

```
myString = "A"
put myString
-- "A"
myString = "Hello"
put myString
-- "Hello"
myString = "Hello World. "
put myString
-- "Hello World. "
myString = " "
put myString
-- " "
```

13.2.1 子字符串表达式

用Lingo可以把字符串拆开，从中得到单个字符或多个字符，即子字符串。要得到子字符串，可以使用char和word关键词。在消息窗口内试做以下运算：

```
myString = "Hello World. "
put char 1 of myString
-- "H"
put word 1 of myString
-- "Hello"
put char 2 to 4 of myString
-- "ell"
```

这种用char和关键词of再加上字符串的子字符串表达式其实是旧的 Lingo句法。Director 7允许我们使用新的“点句法”来实现同样的目的。它之所以称为“点句法”，是因为表达式中有点(“.”，即西文的句号)的存在。

```
myString = "Hello World. "
put myString
-- "Hello World. "
put myString.char[1]
-- "H"
put myString.word[1]
-- "Hello"
put myString.char[2..4]
-- "ell"
```

除了char和word以外，还有更多的子字符串表达式。首先是 line和paragraph。paragraph实际上是Director的新添内容，但它与line是一样的。word根据字符串里的空格的位置返回结果，而line或paragraph则根据字符串里的回车符返回整行字符。

此外，还有item子字符串表达式。它根据字符串里的逗号的位置返回字符串的某些片断。在消息窗口内试做如下运算：

```
myString = "apples,oranges,pears,peaches,bananas"
put item 2 of myString
-- "oranges"
put myString.item[3]
-- "pears"
```

可以看到，item表达式可以使用旧的句法，也可以使用新的“点句法”。此外，我们还可以改变 item表达式里做为分界符号的字符。为了实现这一点，需要为一个叫做 the itemDelimiter的特殊变量赋值。关键词 the表示这是Director中的一个特殊属性。在消息窗口内试做以下运算：

```
myString = "walnuts;peanuts;sunflower seeds"
the itemDelimiter = ";"
put item 2 of myString
-- "peanuts"
put myString.item[3]
-- "sunflower seeds"
```

各种子字符串表达式的最大优点在于它们可以一起使用。例如，可以得到某一行里的某个单词里的某个字符。在消息窗口内试做以下运算：

```
the itemDelimiter = ","
```

```
myString = "red,yellow,blue,green,light brown"
put myString.item[5]
-- "light brown"
put myString.item[5].word[2]
-- "brown"
put myString.item[5].word[2].char[4]
-- "w"
put char 4 of word 2 of item 5 of myString
-- "w"
```

13.2.2 修改字符串

除了可以获得字符串的片断外，我们还可以为字符串添加或减去某些片断。连接字符串的方法是使用“ & ”字符。在消息窗口内试做以下运算：

```
myString = "Hello"
put myString&"World. "
-- "HelloWorld. "
myOtherString = "World. "
put myString&myOtherString
-- "HelloWorld. "
put myString&&myOtherString
-- "Hello World. "
```

第一个例子使用了一个字符串变量和一个真实字符串。当我们用“ & ”连接内容为Hello的字符串变量和真实字符串 World时，就得到了HelloWorld。当我们使用两个字符串时，得到的是同样的结果。最后，当我们使用“ && ”时，它在新字符串中插入了一个空格。 &&的用途的确如此：用一个空格连接两个字符串。

还可以用连接法创建新的字符串，并把它存储在一个变量里。在消息窗口内试做以下运算：

```
myString = "Hello"
myOtherString = "World. "
myNewString = myString&&myOtherString
put myNewString
-- "Hello World. "
```

我们还可以用另一种方法改变字符串的内容，这涉及到 put命令的使用。当我们单独使用put命令时，它的作用是把一些信息放到消息窗口内。但是，如果把 put命令与关键词before、after或into放在一起，它就变成了另外一个命令。它实际上是为变量赋予新的值。在消息窗口内试做以下运算：

```
myString = "Hello"
put "World. " after myString
put myString
-- "HelloWorld. "
put "I said " before myString
put myString
-- "I said HelloWorld. "
```

我们看到，可以用put命令向字符串中添加文本。通过连接几个字符串，并把它们赋予新的变量(或同一个变量)，也能得到同样的结果。

使用该命令的一种特殊方法是结合使用 put命令和子字符串表达式。实际上可以向字符串

里插入文本。

```
myString = "HelloWorld. "
put " " after myString.char[5]
put myString
-- "Hello World. "
```

还可以把into与put命令一起使用。它将替换原字符串。尽管这与用“ = ”为变量赋一个新值相同，但put和into共同使用，可以修改字符串的局部内容。

```
myString = "Hello World. "
put "Earth" into myString.char[7..11]
put myString
-- "Hello Earth. "
```

除put命令外，还有一个delete命令。它可以从一个字符串变量中删去一个子字符串。在消息窗口内试做以下运算：

```
myString = "Hello World. "
delete myString.char[2]
put myString
-- "Hllo World. "
```

delete也能够用于字符、单词、文本行和其他项目。

参见第16章“控制文本”里的16.1节“使用字符串和子字符串”，可以获得有关字符串的更多信息。

13.3 比较变量

我们想让Lingo完成的任务之一是做决定。要让计算机做决定，它需要有信息。信息对于计算机来说就是二进制数：1或0，开或关，TRUE(真)或FALSE(假)。

当需要让Lingo做决定时，它使用“真”和“假”的概念。当我们比较变量时，例如，当我们问：“它们相等吗？”，答案可以为“是”或“不是”，而计算机则称为“真”或“假”。

在消息窗口里试做一些练习。操作符“ = ”是最常用的比较操作符。练习以下内容：

```
put 1 = 1
-- 1
put 1 = 2
-- 0
put "abc" = "def"
-- 0
put "abc" = "abc"
-- 1
```

可以看到，如果比较的结果是“真”，Director返回1；结果为“假”，则返回0。用1和0代表“真”和“假”是非常普遍的，Director甚至把TRUE和FALSE这两个单词直接认作那两个数字。

```
put TRUE
-- 1
put FALSE
-- 0
```

注释 在前面的例子里，TRUE和FALSE是大写的，因为它们是常量。常量是Lingo术语，它所定义的内容是不变的，如TRUE就是1。由于Lingo对大小写不敏感，我们本不需要

大写这些词汇，但由于这是一个习惯用法，因此本书也照样执行。

除了使用“ = ”操作符外，变量和真实值还可以用其他许多方法进行比较。表 13-2是所有的Lingo比较操作符。

表13-2 Lingo比较操作符

操作符	比较数字	比较字符串
=	两个数字是否相等？	两个字符串是否相同？
<	第一个数字是否小于第二个数字？	如果按字母顺序排列，第一个字符串是否排在第二个字符串的前面？
>	第一个数字是否大于第二个数字？	如果按字母顺序排列，第一个字符串是否排在第二个字符串的后面？
<=	第一个数字是否小于或等于第二个数字？	如果按字母顺序排列，第一个字符串是否排在第二个字符串的前面，或与之完全相同？
>=	第一个数字是否大于或等于第二个数字？	如果按字母顺序排列，第一个字符串是否排在第二个字符串的后面，或与之完全相同？
<>	第一个数字与第二个数字是否不同？换句话说，它们不相等吗？	两个字符串是否不同？

以下是消息窗口里的一些实例：

```
put 1 < 2
-- 1
put 2 < 1
-- 0
put 2 <= 1
-- 0
put 2 <= 2
-- 1
put 2 <=3
-- 1
put 2 >= 2
-- 1
put 2 >= 1
-- 1
put 2 <> 1
-- 1
put "this" > "that"
-- 1
```

有一点要注意的是，当比较字符串时，Director不考虑大小写。因此abc与Abc被视为相等。此外还要注意不要把比较操作符放在引号内。

13.4 使用处理程序

在上一章我们已看到了如何创建处理程序 (handler)。处理程序是一些Lingo命令的集合，用来完成某项任务。处理程序可以用特定的关键词命名，对 Director的事件做出反应。处理程序也可以有自定义的名称，并可以由我们编写的其他处理程序调用。有时，处理程序还可以返回一个值。

参见第12章“ 学习Lingo ”里的12.6.5节“ 编写程序 ”，可以获得更多有关创建处理程序的信息。

13.4.1 事件处理程序

on mouseUp或on exitFrame等处理程序就属于事件处理程序。它的用途是回应 Director里的特定事件所发出的消息。第 12章列出了很多种事件，并列举了一些实例，如“用户点击鼠标”或“某一帧播放完毕”。

了解什么类型的剧本使用什么类型的处理程序是很重要的。 on mouseUp处理程序通常位于行为剧本里。这个行为剧本是附属于用户所点击的角色的。 on exitFrame处理程序通常用于帧剧本。这个帧剧本的任务是在该帧被显示出来后执行某些动作。在 Lingo编程的初学者所写的剧本里，这些都是这些处理程序的典型位置。

事实上大多数处理程序都可以被放置在任何地方。 on mouseUp处理程序可以放在帧剧本或影片剧本里，只要我们希望每次用户点击鼠标时就调用该处理程序，且没有其他对象接收该消息。on exitFrame处理程序可以用于角色行为剧本上，只要在该角色演出，每一帧结束时就运行该处理程序。

有些处理程序不能用在某些位置。当用户按下键盘里的某个键时， on keyDown处理程序就得到一个消息。它可以用于可编辑的文本演员或域，可以用于帧剧本，甚至可以用于影片剧本。但是，如果on keyDown被用于属于某个位图角色的行为，它将永远不会被调用。这是由于位图不同于可编辑的文本演员或域，它不接受按键信息，也不会做出反应。

所有Lingo代码必须由事件处理程序来调用。即使我们编写了自定义的、用于完成某种任务的处理程序，它也必须由某些事件处理程序来调用，否则自定义的处理程序就处于空闲状态，等待调用。但也存在少数例外，如当我们在消息窗口内键入自定义处理器的名称时，即使此时没有事件处理程序的参与，该处理程序也将运行。

13.4.2 自定义处理程序

当我们编写自己的处理程序时，可以为它取任何名称，只要不是事件的名称就可以。通常我们把它放在影片剧本里。这样，该处理程序可以被任何行为或其他影片剧本调用。自定义的处理程序可以被放在行为里，但它只能被该行为的其他处理程序调用。当我们学会编写复杂的行为后，可以向其中加入自定义处理程序，但这些处理程序只能被那个行为里的其他事件处理程序调用。

下面的例子是一个影片剧本。当影片开始时，on startMovie处理程序被一个事件消息调用。而它又调用另一个名为 initScore的自定义处理器。在这里， score指的是游戏的得分，而不是Director的剪辑室(Score)。这个处理程序设置了几个全局变量：

```
on startMovie
  initScore
  go to frame "intro"
end

on initScore
  global gScore, gLevel
  set gScore = 0
  set gLevel = 0
end
```

我们原本可以把 initScore处理程序的所有程序行都写在 on startMovie处理程序里。但是，

创建自定义的处理程序有几个好处。首先，它使程序层次清晰。 initScore处理程序只承担一项任务。第二，它使得initScore处理程序能在后来被再次调用。在本例里，当用户开始一个新游戏时，我们也许需要把得分复位。如果我们把这些程序行放在了 on startMovie里，则还需要再执行一些本不该执行的程序行，如 go to frame "intro"等，尽管在下一次我们想要把得分复位时并不需要执行那些操作。

13.4.3 函数

有一类自定义的处理程序有时被称作函数。这类处理程序的特别之处在于它们能返回数值。它们与本章在前面所介绍的数学函数的工作方式是相同的。不同之处当然在于我们可以定义该函数的内容。

与简单的处理程序相比，函数有两个元素与之不同：输入和输出。函数处理程序通常把一个或多个数值作为输入接受，并返回一个数值。输入可称为参数，而输出则称为返回值。

为了使处理器能够接受参数，唯一需要做的就是把变量的名称添加到声明行。这是指用on开头的那一行。以下是一个例子：

```
on myHandler myNumber
  put myNumber
end
```

这个例子并不是一个真正的函数，因为它没有返回任何数值，但是它是一个有效的处理程序。当你开始编写自己的程序后，写一些只接受一个或多个参数，却不返数值的处理程序的现象是很常见的。

通过把变量名myNumber放在一个处理程序的声明行，我们已准备好让该处理程序在被调用时，就接受该数值。如果把该处理程序放在影片剧本里，然后在消息窗口内输入以下信息，该处理程序将会运行：

```
myHandler(7)
-- 7
```

可以看到，数字7被送到消息窗口， put myNumber这一行就负责把它放在那里。当用 7作为参数调用myHandler时，这个处理程序就运行。在执行任何 Lingo命令之前，它把数字7放在局部变量myNumber里。

把多个变量用作参数也是很容易的，只要在声明行用逗号把它们分开即可。在调用该处理程序时也是如此。请看下面的例子：

```
on myHandler myNumber, myOtherNumber
  mySum = myNumber + myOtherNumber
  put mySum
end
```

从消息窗口调用该处理程序时，在处理程序名称后面用括号把两个数字括起来。该处理程序把这两个数字赋给声明行里的两个变量。这个处理程序立即把这两个变量相加，创建一个新变量，并把它的值显示在消息窗口里。

```
myHandler(7,4)
-- 11
```

这时距离把这个例子变成真正的函数只有一步之遥，只要让该处理器返回一个数值就可以了。用return命令就可以实现这一目的。下面是同一个处理程序，只是没有 put命令，而添

加了return命令：

```
on myHandler myNumber, myOtherNumber
  mySum = myNumber + myOtherNumber
  return mySum
end
```

现在，可以像在本章前面的部分里调用 integer函数那样在消息窗口调用这个处理程序了。在消息窗口里试做以下运算：

```
set myNumber = myHandler(7,4)
put myNumber
-- 11
```

on myHandler处理程序现在就是一个独立的个体了。它使用两个数字，并能返回它们的和。现在可以为处理程序和变量指定更符合其功能的名称：

```
on addTwoNumbers num1, num2
  sum = num1 + num2
  return sum
end
```

该函数现在就可以在消息窗口里使用了，也可以被其他处理程序调用了。

```
put addTwoNumbers(5,24)
-- 29
put addTwoNumbers(-7,2)
-- -5
put addTwoNumbers(4.5,2.1)
-- 6.6000
```

我们还可以使用函数来比较变量。除了返回一个新数值外，它还可以返回“真”或“假”：

```
on isOneGreaterThan num1, num2
  return (num1 - 1) = num2
end
```

该函数对两个数字进行比较。它实际上是把第一个数减 1后再与第二个数比较。如果是“真”就返回1，否则返回0。

```
put isOneGreaterThan(5,4)
-- 1
put isOneGreaterThan(7,2)
-- 0
```

13.5 使用if...then

现在我们已经了解了如何比较变量以及得到“真”或“假”的值。再学一些句法将会更有助于我们处理这些数值。

13.5.1 简单的if语句

关键词if和then可以用来处理比较结果。请看下面的例子：

```
on testIf num
  if num = 7 then
    put "You entered the number 7"
  end if
```

```
end
```

在消息窗口试做以下运算。你可能事先已猜到了结果：

```
testIf(7)
-- "You entered the number 7"
```

如果在if和then之间的语句的值是“真”，则在if行与end if行之间的所有命令都将被执行。

注释 也可以把整个if语句写在同一行内：if num=7 then put "You entered the number 7"。但只有当if语句里只有单行命令时才能这样做。

if语句的自然延伸是else关键词。可以用这个关键词指定当 if语句的判断为假时所要执行的命令。请看下面的例子：

```
on testElse num
   if num = 7 then
   put "You entered the number 7"
 else
   put "You entered a number that is not 7"
 end if
end
```

以下是在消息窗口里测试该函数时所得到的结果：

```
put testElse(7)
-- "You entered the number 7"
put testElse(9)
-- "You entered a number that is not 7"
```

13.5.2 Case语句

还可以把if语句变得更复杂。可以用else关键词寻找其他特定的情况。例如：

```
on testElse2 num
  if num = 7 then
     put "You entered the number 7"
  else if num = 9 then
    put "You entered the number 9"
else
     put "You entered another number"
  end if
end
```

现在，我们的处理程序已经可以处理各种情况了。实际上， Director有一些特殊句法，可以处理像上面这个处理程序所遇到的多重条件语句。下面的处理程序的功能与上面那个处理程序的功能完全相同：

```
on testCase num
 case num of
   7:
     put "You entered the number 7"
   9:
     put "You entered the number 9"
   otherwise:
     put "You entered another number"
 end case
end
```

case语句以更简洁的方式书写了多重条件语句。它的功能没有超出 if语句。

在case语句里，我们把条件放在第一行的 case与of之间。接着，在每个数值后面列出相应的命令，数值后面有一个冒号。关键词 otherwise与if语句里的最后一个else的作用相同。

13.5.3 嵌套的if语句

理解Lingo命令的动态性是非常重要的。例如，一个 if语句可以存在于另一个 if语句之内。请仔细看下面这个例子：

```
on nestedIf num
  if num < 0 then
    if num = -1 then
      put "You entered a -1"
    else
      put "You entered a negative number other than -1"
    end if
  else
    if num = 7 then
      put "You entered a 7"
    else
      put "You entered a positive number other than 7"
    end if
  end if
end
```

前面的例子首先判断数字是否小于 0，如果是，它又根据该数字是 -1还是另一个负数而做两件事情。如果该数字不小于 0，它又会做另外两件事，即如果该数字是 7，则做第一件事，否则就做第二件。

尽管不一定要这样嵌套才能达到所需要的结果，但它演示了 if语句的嵌套使用法。使用这种方法可以使程序更好地顺应我们的想法，也可能由于逻辑的需要而必须嵌套使用 if语句。当我们有了更多编程经验后，就会遇到这类情况。

还可以进一步嵌套。可以根据需要，多层嵌套，其层次的深度没有限制。也可以把 if语句放在case语句之内，反之亦可。

13.5.4 逻辑操作符

如果要同时测试两个项目又怎样呢？假设一个函数要检测两个数字，而不是一个，这个函数可能是这个样子：

```
on testTwoNumbers num1, num2
  if num1 = 7 then
    if num2 = 7 then
      put "You entered two 7s"
    end if
  end if
end
```

实现这一目的的简单方法是使用逻辑操作符 and。它能够在一个if语句时检查多个条件：

```
on testTwoNumbers2 num1, num2
  if num1 = 7 and num2 = 7 then
    put "You entered two 7s"
  end if
```

```
end
```

第二个处理程序更紧凑，可读性更好。我们还可以使用操作符 or来检查两个条件是否有一个为“真”，而不必是两个同时为“真”。

```
on testEither num1, num2
  if num1 = 7 or num2 = 7 then
    put "You entered at least one 7"
  end if
end
```

我们应该掌握的另外一个逻辑操作符是 not。该操作符能够反转比较的结果。真值变成了假，假值变成了真。下面这个简单的例子演示了它的用法：

```
on testNot num
  if not (num = 7) then
    put "You did not enter a 7"
  end if
end
```

这几个简单的例子只涉及了这几个操作符的皮毛。使用 and、or、not和括号，可以构筑非常复杂的逻辑语句。

13.6 使用repeat循环

计算机非常善于做重复性的工作。让一组 Lingo命令重复执行的方法是使用 repeat命令。我们可以让某个命令重复一定的次数，直至满足某个条件，或者永远重复下去。

13.6.1 repeat with

如果想让一个Lingo命令计数到100，只需要几行简单的程序。下面是一个例子：

```
on countTo100
  repeat with i = 1 to 100
    put i
  end repeat
end
```

repeat with循环创建一个新变量，即 i，并告诉它从哪里开始、到哪里结束。在 repeat和end repeat之间的所有命令都将执行那么多次。此外，变量 i中包含当前循环的次数。

这个处理程序的结果是从1计数到100，并把每一个数都显示在消息窗口内。

提示 使用down而不是to，可以让repeat循环从某个数字开始倒着数。

13.6.2 repeat while

另外一类重复循环是repeat while循环。该操作将重复进行，直至某一个条件为“真”。下面这个处理程序与刚才的例子的功能是完全相同的。

```
on repeatTo100
  i = 1
  repeat while i <= 100
    put i
    i = i + 1
  end repeat
end
```

该处理器令变量i的初始值是1，然后开始循环，每循环一次就把该变量的值输出到消息窗口内一次，再把该变量的值加1。在每次循环开始之前，要检查i<=100这个语句是否为真。当它不再为真时，循环即告结束。

当然，这个例子是很简单的。如果真需要实现类似的功能，还是应该使用前面例子里的repeat with。repeat with句法用于计数，而repeat while则适合做其他许多事情，如下例中所遇到的情况。在下面的例子里，处理程序将一直计数，直至用户按住 Shift键使之停止：

```
on countWhileNoShift
  i = 1
  repeat while not the shiftDown
    put i
    i = i + 1
  end repeat
end
```

这个例子使用了一个名为 shiftDown的新属性。当 Shift键被按住时，它返回 1；当它没有被按下时，则返回0。当我们在消息窗口内运行该处理程序时，变量 i开始计数。按住Shift键，它即会停止。消息窗口里显示着计数的过程。

13.6.3 其他repeat命令

假设我们想让一个处理程序计数到 100，但也可以用Shift键中断它。下面的例子是实现这一目的的一种方法：

```
on countTo100orShift1
  i = 1
  repeat while (i <= 100) and (not the shiftDown)
    put i
    i = i + 1
  end repeat
end
```

该处理程序使用了 repeat while循环，检查两个条件是否同时为真： i小于等于100，且Shift键没有被按下。实现这一目的的另一种方法是使用 exit repeat命令：

```
on countTo100orShift2
  repeat with i = 1 to 100
    put i
    if the shiftDown then exit repeat
  end repeat
end
```

这第二个处理程序显得更加简炼。它使用了更适合于计数的 repeat with循环。循环内有一条语句，它检查Shift键是否被按下。如果是，则exit repeat命令使Director跳出该循环。

exit repeat可以使用在repeat with和repeat while循环里。它作为第二个令循环结束的方法而存在。我们现在写的循环只有几行，但高级 Lingo程序员则会写出相当长的循环。有时，exit repeat命令是在必要时跳出循环的最好方法。

还有一个next repeat命令，它的作用不像exit repeat那么突然。它不是终止循环，而只是阻止执行该行以后的所有命令行，而让循环立即跳到下一次循环的起点。请看下面的例子：

```
on countTo100WithShift
  repeat with i = 1 to 100
    put "Counting... "
```

```
    if the shiftDown then next repeat
    put i
  end repeat
end
```

同以前的许多例子一样， 这个处理程序也是由1计数到100。每循环一次，它就在消息窗口内显示Counting...，然后再显示一个数字。但如果在循环过程中 Shift键被按下， next repeat命令就阻止Director继续执行put i这一行。结果是只有Counting...显示了足够多次。

注释 Lingo的速度非常快，以至于用户都没有机会在这个处理程序结束运行之前用Shift键中断它。你可能需要用1000或更大的数字来试验，而不能使用100。

13.6.4 永远循环

有时，我们需要构筑一个循环，希望它永远继续直至执行了一个 exit repeat命令。在这种情况下，我们不想使用repeat with，因为该命令会使循环只重复特定的次数。使用 repeat while也需要设置一个能够使循环终止的条件。

但是，也有一个小窍门，使得我们可以使用 repeat while，而不必设置条件，实际上也就是让它永远循环。请看下面的例子：

```
on repeatForever
  repeat while TRUE
    put "repeating... "
    if the shiftDown then exit repeat
  end repeat
end
```

这个处理程序实际上并没有永远循环，只是在 Shift键被按下之前一直循环。但是，也只有exit repeat才能终止这个循环，因为我们把TRUE用作了repeat while循环的条件。TRUE永远都是真，因此该循环会永远执行，直至被 exit repeat打断。

在我们不想把循环条件直接放在 repeat while后面时，也许就会需要使用这种循环。当条件很长很复杂时，或当终止循环的条件有很多个却又相互都是平等的关系时，可能需要使用这种循环。

提示 如果你设立了一个将永远执行的循环，Director也无法令它停止，你随时都可以按组合键Command+.(西文的句号)(Mac)或Ctrl+.(Windows)，中止Lingo和Director。如果不这样做，Director会陷入死循环。

13.7 使用Lingo的浏览命令

循环、if语句和处理程序都是组织和控制 Lingo命令的方法。但那些真正“干活”的命令又怎样呢？

Director里最简单的命令是那些能够在剪辑室里来回移动播放头的命令。我们可以全面控制播放头的走向。可以令它跳到某个带编号或标签的帧，甚至跳到另一部影片。

13.7.1 go

只要对它说go，就可以让影片跳到某个编号的帧。打开剪辑室窗口和消息窗口。播放头

现在应该在第1帧。现在，在消息窗口内键入 go 5。

播放头应前进到第5帧。该命令的更正式的形式是使用完整的语句 go to frame 5。这个语句的可读性更好一些。

也可以让影片前进到下一标志。为第 7帧设定一个标志叫做 intro，然后在消息窗口内键入 go to frame "intro"。

表13-3给出了go命令的所有变种。

表13-3 GO命令的多种形式

形 式	描 述	实 例
go to frame X	影片走到编号为X的那一帧	go to frame 7
go to frame "X"	影片走到标志为X的那一帧	go to frame "credits"
go to the frame	影片重复播放同一帧	go to the frame
go to the frame + X	影片向前跳X帧	go to the frame + 1
go to the frame - X	影片向后跳X帧	go to the frame - 1
go next	影片跳到下一个带有标志的帧	go next
go previous	影片跳到当前带标志的帧的前面一个带有标志的帧	go previous
go loop	影片跳回当前带标志的帧	go loop
go marker(X)	影片向前跳到第X个带有标志的帧	go marker(2)
go marker(-X)	影片向后跳到第X个带有标志的帧	Go marker(-2)

最容易令人混淆的是当前带标志的帧与前一个带标志的帧。图 13-1是含有几个带标志的帧的剪辑室。播放头在第9帧。第7帧的标志为that，第3帧为this，第12帧为other。如果此时执行go loop，影片将跳回到that帧，因为它是在播放头前面距播放头最近的带标志的帧。但是，go previous却把影片带回到this帧，因为它被视为前一帧。

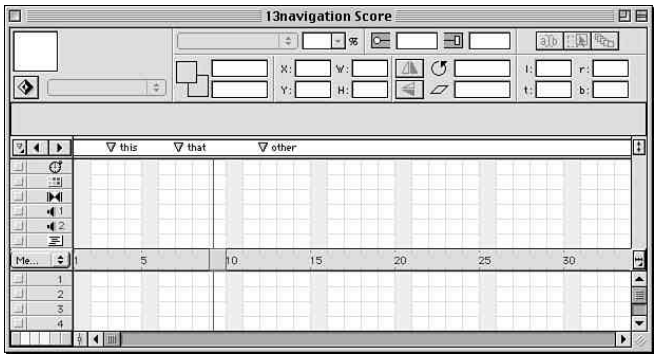

图13-1 播放头位于两个带标志的帧之间。that被视为当前帧，this被视为前一帧

表13-3还介绍了Lingo的其他一些元素。第一个是属性 the frame。该属性返回当前帧的编号。如果把剪辑室设置图13-1的样子，可以在消息窗口内做如下操作：

```
put the frame
-- 7
```

marker函数也可以从剪辑室里得到信息。它以一个数字作为参数。如果该数字是 0，它返回当前标志的名称；如果数字是 -1，它返回前一个标志的名称；如果是 1，它返回下一个标志的名称。当剪辑室与图13-1相同时，在消息窗口内试做以下操作：

```
put marker(0)
-- "that"
```

```
put marker(1)
-- "other"
put marker(-1)
-- "this"
```

因此，Lingo命令go previous 与go to marker(-1)是相同的。go next与go loop分别与前面这个例子里的另外两项相对应。

我们还可以用marker函数得到(或走到)在当前标志帧之前或之后的某个标志帧，而不是仅仅只能走到与当前标志帧相邻的标志帧。

13.7.2 使用go

要使用go命令真正完成某些任务，只需要把它放到一个简单的行为剧本里。请看下面的例子：

```
on mouseUp
  go to the frame 8
end
```

要确认这里所创建的是行为剧本，而不是影片剧本。用 Script Cast Member Properties对话框可以检查这个选项。用剧本窗口里的 Info(信息)按钮，可以调出该对话框。

当我们有了这个行为剧本后，可以把它拖动到任何一个角色上，以创建一个简单的按钮。该剧本的作用与用行为监察窗创建的行为的作用是一样的。

更有用处的行为是按标志走，而不是按编号走。我们还可以使用 go next和go previous来创建前进和后退按钮。

另外一种使用go命令的方法是用在帧剧本里。下面这个例子可能是用 Lingo写的最常用的剧本：

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

这个帧剧本所做的事情就是让影片暂停在当前帧。每当该帧播放结束，就再次播放它。Hold on Current Frame库行为也是这个作用。我们用这个剧本使影片暂停在当前帧，以便用户能够选择某个按钮或执行某种动作。

同样地，也可以用go loop命令让影片循环至当前标志帧。go to the frame是把一帧循环播放，而go loop则循环播放多帧。这使我们能够上演循环的动画，同时又保持一些按钮以供用户使用。

13.7.3 play

在影片中浏览的第二种方法是使用 Play命令。这个基本命令与 go相似，但有一点重大的差别。play done命令可以用来把播放头返回到原来那一帧。

假设在剪辑室里有三个标志： menu、chapter 1和chapter 2。在有标志menu那一帧的一个按钮可以发出play frame “ chapter 1 ”命令，在有标志chapter 1的那一帧里的另一个按钮可以发出play done命令，使播放头回到 menu帧。同样的play done按钮和行为还可以用在有标志chapter 2的那一帧中。

事实上，我们可以把 play命令放在影片中的任意地方， Director可以记忆那个位置，当接到play done命令时，就会返回到那一帧。

13.7.4 离开当前影片

用go和play不仅能跳到当前影片的某个帧，也可以跳到另一部影片去。其句法只是对前面所介绍的句法的简单扩展。

下面这个行为可以打开另一部影片，并从它的第 1帧开始播放：

```
on mouseUp
  go to movie "nextMovie.dir"
end
```

注释 如果我们忘记了为影片文件选择正确的扩展名， Director却不会忘记，因为它够聪明。因此，我们也不必担心 Windows文件有.dir扩展名，而Mac没有。当我们需要制作保密版本或压缩版本而需要使用带有 .dxr和.dcr扩展名的文件时，也不必担心这个问题。

如果不想让影片从第1帧开始播放，可以把该命令再加以扩展而指定某一帧：

```
on mouseUp
  go to frame "intro" of movie "nextMovie.dir"
end
```

用play命令时也使用同样的格式。这时就到了 play命令发挥威力的时候了。我们可以设置一个按钮，把用户带到另一个影片里，让他们在那里浏览。当他们浏览完了以后，可以按一个带有play done命令的按钮，再回到原先的影片里来。 play done命令不但把用户送回原先的影片，而且送到了正确的那一帧。

参见第2章“用Director演示”里的2.3节“非线性演示”，可以得到更多有关影片和浏览的信息。

13.8 控制角色的属性

除浏览外 ，Lingo命令能够完成的另一个重要的任务是改变角色的属性。属性描述了对象的所有特征，包括从它们的位置到它们的油墨种类。改变角色的属性是我们用 Lingo让角色活动起来的方法。

通常，我们都在附属于角色的行为剧本里设置角色的属性，但现在我们可以在消息窗口里查看角色的一些属性。

13.8.1 角色的位置

先为下面的实例创建一个位图演员，把它放到舞台上，同时也把它放到剪辑室的通道 1里。然后打开消息窗口，试键入以下信息：

```
sprite(1).locH = 50
updateStage
```

可以看到该角色将跳动一下, 它的套准点(通常为它的中心)现在距舞台的左边框是50像素。这是上面例子里的第一行语句完成的动作。它把角色 1的属性locH设置为50。locH属性是角色的横向位置。

但是，在第二个命令 updateStage执行前，我们在舞台上看不到任何变化。这是在没有播放影片的情况下为了显示舞台上所发生的变化而存在的特殊命令。如果影片正在播放，而且

对属性的修改已写进了行为剧本，则每播放一帧，舞台将自动更新。但在我们这些例子里，还需要使用updateStage，以观察角色属性的改变。

句法sprite(1).locH是Director 7的新添内容。如果你曾用 Director 6.5或更早的版本编过程序，可能使用的是这样的命令：

```
set the locH of sprite 1 = 50
updateStage
```

在这种简单的例子里，还可以使用这种旧的句法。但最好要习惯使用这种新的“点”句法。这不仅因为它的结构更简单，而且因为 Director的新功能要求这样做。

也可以设置角色的纵向位置。要实现这一点，应使用 locV属性：

```
sprite(1).locV = 100
updateStage
```

还有方法同时设置角色的横向和纵向位置。该属性称为角色的 loc。但它不是直接使用数字，而是使用 point。point是一个特殊的Lingo对象，它是这样使用的： point(x,y)。括号里的两个数字代表横向和纵向坐标。下面的例子就使用了它：

```
sprite(1).loc = point(200,225)
updateStage
```

我们还可以设定角色的许多其他属性。表 13-4列出了其中的一部分。

表13-4 角色的部分属性

属 性	描 述	取值的例子
member	角色所使用的演员	member "myButton"
locH	角色的横向位置	50
locV	角色的纵向位置	50
loc	角色的位置	point(50,50)
rect	角色的矩形	rect(25,25,75,75)
ink	角色的油墨	8
blend	角色的混色	100
trails	轨迹属性是开还是关	FALSE
color	角色的前景色	rgb("#000000")
bgcolor	角色的背景色	rgb("#FFFFFF")

13.8.2 角色的演员

把角色的演员重新设置，使之与原来的演员不同，这是很常见的。这样，鼠标在按钮上掠过时和按钮被按下时可以采用与按钮的正常状态所不同的演员，而且在动画里可以使用不同的图像。

要改变角色所使用的演员，只要设置那个角色的 member属性。不能只把它设置成某个演员的名称或编号，而要设置成真实的演员对象。演员对象只是 Lingo能够理解的指代演员的一种方式。如果某个演员叫做“ button down state ”，就要用member (" button down state ")指代它；如果该演员是7号演员，就应该用member (7)指代它。

创建一个含有两个位图的影片。可以选用任何位图，甚至可以是在 Paint窗口里随意涂抹的图像。为它们命名为bitmap1 和bitmap 2。把第1个演员放到舞台上，成为角色 1。在消息窗口内试键入以下信息：

```
sprite(1).member = member("bitmap2")
updateStage
sprite(1).member = member("bitmap1")
updateStage
```

可以看到角色改变了，但随即又变回原状，每次变化都是由于执行了 updateStage命令。也可以用演员的编号指代它们。

13.8.3 角色的矩形

rect属性也有特殊的 Lingo对象与之关联。在表 13-4里，使用了这样的例子：rect(25,25,75,75)。这几个数字分别表示了角色的左边、顶边、右边和底边的位置。

我们可以把一个角色的rect设置成任意尺寸，而不必顾及这与角色实际尺寸的对应关系。如果演员能够被拉伸(如位图)，它就会在舞台上变形，以对应该角色的矩形。

13.8.4 角色的油墨

ink属性取一些数字作为它的值。不幸的是，我们不能把它的值设为 Background Transparent等内容。请参考表13-5，查看油墨与编号的对应关系。

表13-5 Lingo油墨编号

编 号	对应的油墨
0	Copy
1	Transparent
2	Reverse
3	Ghost
4	Not Copy
5	Not Transparent
6	Not Reverse
7	Not Ghost
8	Matte
9	Mask
32	Blend
33	Add Pin
34	Add
35	Subtract Pin
36	Background Transparent
37	Lightest
38	Subtract
39	Darkest
40	Lighten
41	Darken

13.8.5 角色的颜色

改变角色的color和bgColor属性只是在有些时候才起作用。这要取决于角色的类型以及它所使用的油墨。 1-bit的位图用这两种颜色设置其黑色和白色像素的颜色。使用 Lighten(变浅)和Darken(变深)油墨的彩色位图可以用这些颜色来改变位图的外观。

定义颜色的方法有几种。第一种是使用颜色的红、绿和蓝的数值。要实现这一点，需要使用rgb结构。用point和rect一样，rgb能帮助我们定义无法用单个数字表现的项目。

先为下面的例子创建一个1-bit位图，并把它放在舞台上。它应该是角色 1，以缺省颜色黑色显示在舞台上。在消息窗口内试键入以下信息：

```
sprite(1).color = rgb(0,0,255)
updateStage
```

位图里的黑色像素应该变成蓝色。rgb结构里的三个数字分别代表红、绿和蓝色。把它们设置为0、0和255，结果将得到蓝色。当像这样定义rgb值时，要记住最小值是0，最大值是255。

用rgb定义颜色的另一种方法是使用颜色的十六进制值字符串。这与 HTML中使用的颜色值是一致的。请看下例：

```
sprite(1).color = rgb("#0000FF")
updateStage
```

同前面的例子一样，这个例子也把颜色设为蓝色。字符串里的 6个数字代表三种颜色：“ 00 ”是0，“ FF ”是255。如果你熟悉十六进制数，可以使用这个系统。如果你习惯于使用HTML颜色，也可以使用它。否则，还是使用前一种方法为好。

注释 在Director 7以前，我们只能根据影片的调色板定义颜色。这些256色调色板用于foreColor和backColor属性。这两个属性现在已经废弃不用了，但它们在Director 7里仍然有效，如果你的确想用也可以。

若想使用影片的调色板里的颜色，可以使用 paletteIndex结构。这使我们可以使用调色板的颜色编号，而不使用rgb数值：

```
sprite(1).color = paletteIndex(35)
updateStage
```

在Mac系统调色板和Windows系统调色板里， 35号颜色都是红色。

参见第14章“ 创建行为 ”里的14.1节“ 控制单个角色 ”，可以获得更多有关角色属性的信息。

13.9 控制演员的属性

除设置角色的属性外，我们也可以设置演员的属性。这些属性取决于演员的类型。

其中一个例子是图形演员。它的一个属性是 shapeType。可以把它设置为 #rect、#roundRect、#oval或#line。在舞台上创建一个图形演员，并确认它是演员 1。然后在消息窗口内试键入以下内容：

```
member(1).shapeType = #rect
member(1).shapeType = #oval
member(1).shapeType = #roundRect
member(1).shapeType = #line
```

可以注意到，这里没有使用 updateStage命令，演员的形状就发生了改变。这是由于该命令可以更新角色的属性，但演员的属性一旦被改变，就立即在舞台上发生变化。但当我们掌握了更多高级Lingo技巧后，还会发现也有例外。

13.10 使用列表变量

每种编程语言都有存储一组数据的能力。在有些语言里，这叫做数组；在 Lingo里，叫做

列表(List)。它可以分为两类：线性列表(linear list)和属性列表（property list)。

13.10.1 线性列表

线性列表是指包含在单个变量里的一串数字、字符串或数据。在消息窗口内试做以下运算：

```
myList = [4,7,8,42,245]
put myList
-- [4, 7, 8, 42, 245]
```

在创建了列表后，要有办法取出其中的单个元素。这需要一些特殊句法：

```
put myList[1]
-- 4
put myList[4]
-- 42
```

注释 在Director 6和以前的版本里，需要用函数getAt来实现这个目的。该函数仍然有效，但有了新句法就没有必要使用它们了。

列表里也可以存储字符串。实际上，它可以混合存储数字和字符串。它们甚至可以存储point、rect和rgb结构的数值。以下是一些合法的列表：

```
myList = ["apples", "oranges", "peaches"]
myList = [1, 2, 3, "other"]
myList = [point(50,50), point(100,100), point(100,125)]
myList = [[1,2,3,4,5], [1,2,3,5,7,9], [345,725]]
```

最后一个例子里的列表里又包含了一个列表。在高级 Lingo里，这是会用到的。下面这个例子里的列表包含一个小型的人名-电话号码数据库：

```
myList = [["Gary", "555-1234"], ["William", "555-9876"], ["John", "555-1928"]]
```

下面的处理程序表现了列表的实用之处。列表里含有几个演员的名称。当该处理程序被调用时，它用该列表快速地更换角色1的演员：

```
on animateWithList
  myList = ["arrow1", "arrow2", "arrow3"]
  repeat with i = 1 to 3
    sprite(1).member = member myList[1]
    updateStage
  end repeat
end
```

该处理程序用repeat循环使变量i从1递增至3，然后把角色1的演员分别设为列表里的每个元素。updateStage用来使这些变化显示在舞台上。

创建列表的另一个方法是使用一些命令向列表中添加元素或从列表中删去元素，而不是一次就创建好列表。 add命令可以向已经存在的列表里添加元素， deletaAt命令则从列表中删去元素。在消息窗口内试做以下运算：

```
myList = []
add myList, 5
add myList, 7
add myList, 9
put myList
-- [5, 7, 9]
```

```
add myList, 242
put myList
-- [5, 7, 9, 242]
deleteAt myList, 3
put myList
-- [5, 7, 242]
```

本例的第一行创建了一个空的列表，然后向里面添加了 3个元素。查看了一下其中的内容后，又添加了第4个元素。最后，又从中删去第3个元素，即9。

我们应该了解的另一个命令是列表的 count属性。它告诉我们列表里当前有几个元素。在消息窗口内试做以下运算：

```
myList = [5,7,9,12]
put myList.count
-- 4
```

在前面那个处理程序的例子里，若不令 i从1递增到3，可以让i从1递增到myList.count。这样，将来可以为列表添加或删除元素，而不必考虑要在剧本里修改那个硬编码数字“ 3 ”。

13.10.2 属性列表

线性列表存在的一个问题是我们只能通过元素的位置去指定元素。另一种列表 —— 属性列表 —— 允许我们为列表里的每个元素分别命名。下面是一个典型的属性列表：

```
myList = [#name: "Gary", #phone: "555-1234", #employedSince: 1996]
```

属性列表里的每个元素都包含一个属性名称和一个属性值。例如，前面那个列表里的第一个元素是属性名称#name和它的值“ Gary ”。属性名称和属性值由冒号分隔。

要引用这样一个列表里的属性，可以使用属性名称而不是位置。在消息窗口内试做以下运算：

```
myList = [#name: "Gary", #phone: "555-1234", #employedSince: 1996]
put myList[#name]
-- "Gary"
```

注释 该列表里的属性称为“ 符号(Symbol) ”。在Lingo里，任何由“ # ”引导的元素都是符号，任何通过符号来引用的元素，其读取速度都非常快。因此，符号用于属性列表是最理想的。在属性列表里，还可以用数字和字符串来做属性名称。

可以用addProp命令向属性列表里添加元素，也可以用 deleteProp命令从属性列表里删除元素。

```
myList = [:]
addProp myList, #name, "Gary"
addProp myList, #phone, "555-1234"
put myList
-- [#name: "Gary", #phone: "555-1234"]
deleteProp myList, #phone
put myList
-- [#name: "Gary"]
```

可以注意到，在创建空的属性列表时，使用了［ :］而不是［］。对线性列表不能使用addProp，对属性列表则不能使用add。

13.10.3 帧剧本

帧剧本是在剪辑室里占据着某一帧的帧剧本通道的行为 。本章已经出现过一个帧剧本的例子。它大致是这样的：

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

这个简单的帧剧本是十分重要的。它可以把影片暂停在某一帧，使其他剧本能够接收prepareFrame、enterFrame和exitFrame等消息，同时用户可以有时间分析该帧的内容。

如果没有这类剧本，影片将立即前进到剪辑室里的下一帧。对于连续播放的动画片可以这样，但对于应用程序来说，就必须让影片在当前帧循环。

从现在起，本书的所有剧本几乎都要依赖 on exitFrame处理程序循环当前帧。

13.11 Lingo句法的故障排除

■ 放在引号里的内容被视为字符串，因此，“ 4+2 ”仍旧是4+2，而4+2是6。

■ 请记住Lingo处理四舍五入的两种方式。如果对整数进行计算，如 3/4，则小数部分被舍弃，得到3/4=0。但如果用integer函数，运算结果会被四舍五入。因此 integer(.75)=1。

■ 如果某个函数在结尾处没有使用 return命令，而使用了 exit命令退出处理程序，它将返回一个VOID。Lingo把它解释为FALSE。

13.12 你知道吗

■ 用floatPrecision系统属性，可以设置Lingo的浮点数的缺省小数位数。该值至少为 4。

■ 函数chars是从字符串中截取子字符串的另一种方法。如果试用 chars("abcdefdgh",3,5)，将得到“ cde ”。如果你打算制作 Java短程序， chars是唯一一个能用来从字符串或域里截取子字符串的办法。

■ 如果在单行内使用if语句， Director则认定在紧接着的下一行里应该有一个 else语句。如果没有，或该行是一个空行， Director则知道没有与if语句配对的else语句。但是，我们平时很容易不经意地在带有 else语句的if语句里嵌套一个单行 if语句。这时， Director会认为else语句与第二个if配对，而不是与第一个 if配对。下面的例子告诉我们如何用一个空行来纠正这个错误：

```
if a = 1 then
  if b = 1 then c = 1
  -- need comment or blank line here
else
  c = 2
end if
```

■ 可以设置多个可能的数值作为 case语句的条件，方法是在冒号前面放置多个数值，其间用逗号分开。

■ 矩形表达式可以写为4个数字一组，如rect(10, 20, 50, 60)，也可以写为2个点一组，如rect (point(10, 20)， point(50, 60))。

■ 也可以使用list函数创建列表。因此， list(4, 7, 8)即是［4, 7, 8］。