



**AppleScript for Absolute Starters**

# 苹果脚本跟我学

 AS4AS

作者 / Bert Altenburg

翻译 / 刘 珏

校对 / 杜志佳

## 目录

目录	2
前言	3
第0章 写在开始之前	5
第1章 脚本就是一系列指令	6
第2章 执行和存储一段脚本	10
第3章 快速编写脚本 (I)	13
第4章 处理数字	15
第5章 处理文本	17
第6章 列表—— <i>list</i>	20
第7章 记录—— <i>record</i>	27
第8章 快速编写脚本 (II)	32
第9章 没有注释？那可不行！	34
第10章 条件语句	36
第11章 避免错误	44
第12章 路径、文件夹和应用程序	46
第13章 重复	51
第14章 处理程序—— <i>handler</i>	57
第15章 信息资源	62
译者后记	63

## 前言

苹果脚本 (AppleScript) 是一项具有划时代意义的技术, 它使计算机程序之间的沟通成为可能。例如, 通过AppleScript你可以

- 查阅电子邮件的同时转存它们;
- 让图片编辑程序批量修改图片的分辨率和尺寸, 之后把修改好的图片发送到另外的计算机上或者发布到网络上;
- 许多其它事情。

所谓的苹果脚本, 或者直接叫成脚本, 是通过脚本语言AppleScript编写一系列的指令。这种语言和英语十分类似, 因此易读、易写、易于理解。

AppleScript功能强大, 重点应用的领域有两个。一是印刷行业, 它们利用AppleScript实现流程自动化 (常用软件有Photoshop、QuarkExpres、InDesign)。再就是Filemaker Pro开发者将AppleScript用于安装了Mac微机的自助服务亭 (Mac-based kiosk), 你通常都能在大型商场或者博物馆中见到这种自助服务亭 (常用软件k-Builder)。除了上面提到的软件, 许多Mac微机上主流非主流的应用软件像GraphiConverter、BBEdit、Word等都支持脚本功能。也就是说可以利用AppleScript操控这些软件。哪些应用程序支持脚本操作并不是本书的重点, 市面上其它的书应该会对这个问题有所涉及。但即使那些书提到了AppleScript的内容, 也往往粗略简单甚至一笔带过, 要读懂那些内容必须有较好的AppleScript的基础知识。本书的主旨就是与你一同学习这些基础知识。

如果你希望进一步拓展自己的知识, 你可能需要查找更多的内容 (参见第15张)。其它的关于编写脚本的书也许会很有帮助。本书是一本免费读物, 我同时非常欢迎你把这本书推荐给其他的Mac微机用户。作为回报, 请你认真阅读第0章关于如何促进Mac微机发展的内容。

进入到了AppleScript的世界, 你会注意到“AppleScript”这个术语宽泛地用于三个概念。

- AppleScript语言: 一种与英语类似的脚本语言, 用来编写针对Mac微机的脚本;
- 一段AppleScript脚本: 或者直接叫一段脚本, 是使用AppleScript语言编写的一系列指令;
- Mac微机操作系统 (Mac OS X) 的重要组成部分, 这个程序帮助操作系统读取AppleScript脚本并执行其中包含的指令。

本书今后再涉及到这三个概念的区分时, 会使用下面的术语:

- AppleScript语言;
- 一段AppleScript脚本, 或者直接叫做脚本;
- 组成Mac OS X的AppleScript脚本程序。

学习使用AppleScript编写脚本是初学编程的一个理想途径。例如Java这类的计算机编程语言, 它的程序员在能够完成最简单的编程任务前必须学习大量的知识, AppleScript则摒弃了这个繁杂的过程。它的简单到10岁的孩童都学得会, 但强大的功能却另专业人士亲睐。它为你的发展留下了巨

大的空间。你甚至可以通过AppleScript写出无论外观还是功能都像商业软件一样出色的东西来，有按钮、有菜单、有滚动条和一切你希望有的东西。这要用到苹果公司免费向用户提供一种名为AppleScript Studio的软件，它不是本书涉及的话题。

编写脚本和编写程序有什么不同？我想这很好回答：编写脚本简单，编写程序复杂。当然，Javascript对我们来说也不是容易的东西。所以这个定义可能靠不住。

## 如何使用本书

正如你所见到的，书中一些文字呈现绿色。我们建议你至少阅读本书两遍。第一次阅读时跳过这些绿色的文字（如果你阅读的是黑白或灰度的版本，这些文字是被放置于两个井号“#...#”之间的内容）。再读的时候连同这些绿色的文字一同阅读。这样你可以有效的复习所学，同时弥补第一遍阅读的不足，学到一些新的技巧。这样使用本书将会减缓你的记忆曲线的坡度。

书中还包含了许多脚本的实例。为了保证你能将文字说明和脚本内容准确的联系起来，每个脚本都用放在中括号里的数字标出标签，例如：[4]。如果脚本包含两行以上的内容，那么中括号里的第二个数字则指明是第几行。例如[4.3]表示第四个脚本中的第三行。

学习骑术靠纸上谈兵是不行的。同样的道理，你不亲自实践，AppleScript也难以学的好。这是本电子版的手册。你没有理由不随时切换到脚本编辑程序（Script Editor）边学习边操作（参见第2章）。

## 第0章 写在开始之前

我为读者写了这本读物。因为是免费的，作为回报，请允许我在进入正题前谈谈关于促进Mac微机发展的话题。每一位Mac微机的使用者都可以为促进自己钟爱的电脑平台的发展尽进绵薄。这里将告诉你如何去做。

1、Mac微机功能越强大，越容易令别人关注它。所以要及时浏览关于Mac微机的原创网站，阅读Mac微机的杂志。当然还要学好 AppleScript 并使其发挥作用。在工作上，AppleScript 能为你节约大量的时间和金钱。

2、通过视觉展示告诉世界并不是人人都用PC。在公共场合穿着一件以Mac微机为印花的T恤是个办法，但还有许多其他途径。如果运行“活动监视器（Activity Monitor）”（位于“应用程序”文件夹下的“实用程序”文件夹里面），你会注意到你的Mac微机只是偶尔才会满负荷运行。

科研人员正在推动几项“分散计算计划”（distributed computing projects, 简称DC），比如 Folding@home 和 SETI@home，就是利用Mac微机空闲的处理能力来为公众服务。你只需要下载一个被称做DC客户端（DC client）的免费小程序并开始处理工作。这些DC客户端（DC client）占用很少的系统资源。如果你运行一个占用资源很大的程序，DC客户端（DC client）将自动中止，因此你大可不必在意它的运行。这项工作如何帮助Mac微机？通常这种DC项目的网站上会对各个团队的工作进度进行排名。如果你加入了一个Mac微机团队（你可以从他们的名字中区别来），你就可以帮助你的团队提升排名。其他平台的用户会看到Mac微机是如此之棒！DC项目的内容很丰富，有关于数学的，也有关于医疗的等等。你可以通过以下网址找到一个你感兴趣的DC项目：

**<http://distributedcomputing.info/projects.html>**

对于这个提议，唯一的问题是它可能会令你上瘾！

3、确保Mac微机拥有最好的软件。你自己不必写软件。要作的是把向开发人员（礼貌的）反馈使用意见变成一种习惯。你对试用的某一款软件并不感冒，那么告诉它的开发者为什么你不喜欢这个软件。发现了bug也要及时报告，最好在报告中精确的描述一下你当时的操作过程。访问下面的网站，里面的多媒体教程会告诉你如何作：

**[www.macinstruct.com/tutorials/crash/index.html](http://www.macinstruct.com/tutorials/crash/index.html)**

4、为你使用的软件付费。只要Mac微机的软件市场能够生存下去，开发人员就会坚持不懈的提供优秀的软件。

5、请向至少3位对编写程序感兴趣的Mac微机用户推荐本书，并告诉他们哪里能够找到本书。或者建议他们履行以上4点建议。

好了，在后台下载DC客户端（DC client）的同时我们开始学习AppleScript。

## 第1章 脚本就是一系列指令

AppleScript作为Mac微机操作系统的一部分能够执行的任务是有限的。例如，它能够产生“咚”的一声。一起来看看脚本 [1]，它能让你的Mac微机发出“咚”的一声。

```
[1]
beep
```

这个一定是世界上最短脚本了，仅仅包含了一个单词组成的指令。一行指令叫做一个语句行，即使一行只有一个单词，也是一行语句。如果上面的脚本被执行，那么你的Mac会发出“咚”的一声。

如果想多响几声，只要在命令beep后面加上数字，这个数字表示你希望发出声响的次数。见例 [2]。

```
[2]
beep 2
```

比较例 [1] 和例 [2] 可知，后面的数字信息是可选的。如果不提供数字，AppleScript假定你只要响一次。所以，1是一个默认值。

如果你觉得“咚”声是PC机上的动静，为什么不让AppleScript以Mac微机的方式与你交流，见例 [3] 这个语句：

```
[3]
say "This is a spoken sentence."
```

你甚至可以选择朗读语音，比如使用“Fred”、“Trinoids”、“Cellos”或者“Zarvox”。例 [4] 就使用了“Victoria”代替了默认语音。

```
[4]
say "This is a spoken sentence." using "Zarvox"
```

**#注意：**通常来说，AppleScript不要求大小写。也就是说你使用大小写都没有关系。尽管这些语音名称比如“Victoria”和“Zarvox”必须以大写开头。#

如你所见，AppleScript的指令与英语十分相似，这就另脚本易读易懂，甚至没有脚本编写经验的人也可以看懂。尽管例 [1] 至例 [4] 很有趣，但是并不实用。AppleScript语言还有许多指令，但也许没有给你太深印象。AppleScript的长处在于它可以让你和其它程序沟通。只要对方程序支持脚本操作就可以实现。幸运的是，大多数Mac微机的程序都可以实现脚本操作。因此你不仅仅要处理组成Mac OS X的AppleScript脚本程序提供的有限的指令，而且要掌握大量其它应用程序提供的脚本指令。

一些Mac微机程序十分流行。“Finder”就是人人都使用的一个。是的，Finder是一个应用程序。

当你打开Mac微机的电源，它也随之启动并一直运行。它允许你移动文件，在硬盘上查找，创建文件夹，复制或者改名。尽管你可以使用键盘或鼠标来清空废纸篓，你同样可以使用AppleScript完成这项操作。

```
[5]
tell application "Finder"
    empty the trash
end tell
```

就像老板，你必须告知

- 谁来执行某项任务，还有
- 执行什么任务。

如果你告诉Photoshop清空废纸篓就不会达到预期效果。因为Photoshop不知道如何完成这项任务。所以，清空废纸篓这样的命令必须交给Finder执行。

现实世界，老板不一定把每件事情都交待明白。你的Mac微机却是一个很笨的雇员，只完成你交待的工作。如果废纸篓里有你一个十分重要的文件，一旦你执行了例[5]给出的脚本，你将永远的失去这份文件。

语句行[5.1]是“tell”语句，通过它我们要求组成Mac OS X的AppleScript脚本程序传送一个或多个指令到另外的一个应用程序，这里是到Finder。组成Mac OS X的AppleScript脚本程序持续执行各行语句，直到遇到第[5.3]行的语句“end tell”。在上面的例[5]我们要求AppleScript向Finder发送了清空废纸篓的命令，之后停止继续指挥Finder。合在一起，我们把这样的部分

```
tell application "xyz"

end tell
```

叫做“tell模块”。在模块中的指令将由程序“xyz”执行。顺便提一下，AppleScript语言在符号使用上并不像其它脚本语言特别是编程语言那样看重细节。它也有一些规则，其中一条就是必须使用双引号将应用程序的名称引起来，就像第[5.1]行那样。

我们也可以给Finder其它指令。在下面的例[6]中，第[6.2, 6.3]行都是发送到Finder指令。因为他们双双能够被Finder执行，记得要把它们放到指向Finder的tell模块里面。

```
[6]
tell application "Finder"
    empty the trash
    open the startup disk
end tell
```

清空了废纸篓，Finder又打开了一个窗口显示硬盘上的内容。

如你所见，我们可以命令Finder作任何我们想要做的事情。我们甚至可以让Finder缩放窗口，将窗口放到指定的位置。后面你将学到这些内容。

我们创建的脚本可以包含针对Finder的指令和针对组成Mac OS X的AppleScript脚本程序本身的指令。参见例 [ 7 ]。

```
[7]
tell application "Finder"
    empty the trash
    open the startup disk
end tell
beep
```

首先，Finder接收了第 [ 7.2, 7.3 ] 行的指令。之后第 [ 7.5 ] 行的beep指令被AppleScript执行。

有趣的是beep指令的位置可以随意（其它针对组成Mac OS X的AppleScript脚本程序本身的指令都有这个性质），可以在tell模块外面，也可以在里面。参见例 [ 8 ]。

```
[8]
tell application "Finder"
    empty the trash
    beep
    open the startup disk
end tell
```

尽管Finder不知道beep指令什么含义，但是组成Mac OS X的AppleScript脚本程序知道如何处理。这样写让脚本更加简单易懂。否则，你就不得不在第一个tell模块写上第一个要求Finder执行的命令；中间的语句行写beep命令；最后又写一个tell模块，里面写第二个要求Finder执行的命令。

值得引起注意的是，针对组成Mac OS X的AppleScript脚本程序本身的指令可以放在一个脚本中任何位置，但要针对某一个特定程序的指令就必须放在一个指向这个程序的tell模块中。例 [ 9 ] 中就隐含了一个致命的错误（在第 [ 9.5 ] 行）。

```
[9]
tell application "Finder"
    empty the trash
    beep
end tell
open the startup disk
```

组成Mac OS X的AppleScript脚本程序不知道怎么去打开启动盘窗口，也不会去查找那个可以执行这个指令的应用程序。脚本的前半部分（第 [ 9.2 ] 至 [ 9.3 ] 行）可以被漂亮的执行，但到了第 [ 9.5 ] 行就不能执行了。

执行脚本时，一旦发现错误，后面的内容就不再被执行。参见例 [ 10 ]。



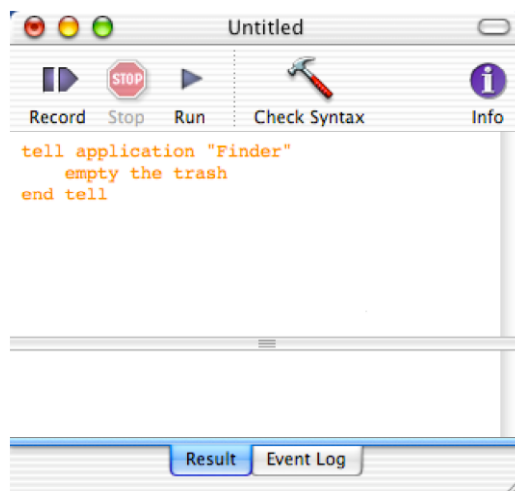
```
[10]
tell application "Finder"
    empty the trash
end tell
open the startup disk
say "I emptied the trash and opened the startup disk for you" using "Victoria"
```

在清空废纸篓后，组成Mac OS X的AppleScript脚本程序将会在第 [ 10.4 ] 行终止运行，因为这行指令本应该指向Finder。但你也不会听到 [ 10.5 ] 的语音了，尽管它没有错误。

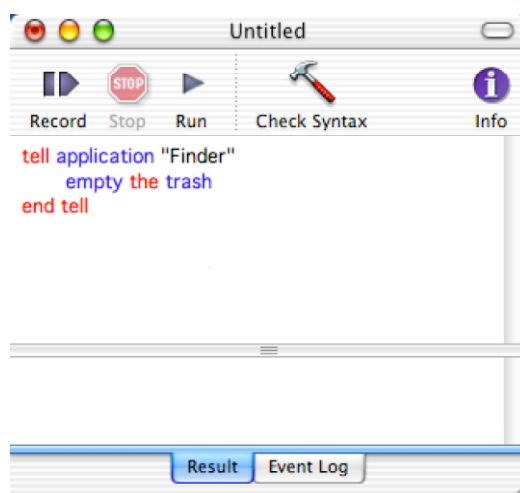
## 第2章 执行和存储一段脚本

我们已经见过了一些脚本的例子，不可否认它们和英语非常的相似，因此易读易懂。你可以在脚本中执行一些指令——比如清空废纸篓——尽管你自己可以使用键盘或鼠标完成。现在看看Mac微机是如何为你执行这个指令的。

脚本编辑程序（Script Editor）是一个用于输入和执行脚本的应用程序。你可以在“应用程序”文件夹下的“AppleScript”文件夹中找到它。启动脚本编辑程序后你会看到窗口被分为两个部分，上面的是区域是输入脚本的地方。



在工具栏中部有一个标签为“Check Syntax”的按钮。尽管AppleScript与英语很相似，但毕竟不同于我们日常所说的英语。“Yo Finder! Dump my garbage.”或者“Hey Finder, clean out the bin.”（两句话都大体都可以译作：“嘿，Finder！把废纸篓倒掉！”——译者按）可不是Finder希望得到的指令。通过检查脚本的语法（check syntax），实际上是编译（compile），组成Mac OS X的AppleScript脚本程序完成了这样一个过程，即检查一下它是否可以理解你的脚本。如果可以，那么就会漂亮的执行你的指令。未被编译的部分以橙色显示，编译后脚本中AppleScript的保留字以红色和蓝色显示。



如果由于人为的错误导致脚本不能被编译，你会看到一条含义模糊的提示信息告诉你脚本中有错误。试试在输入例 [ 2 ] 的时候不带双引号，那么组成 Mac OS X 的 AppleScript 脚本程序就对你的输入“不知所云”了。

[2]

```
say "I'm learning AppleScript the easy way!" using "Zarvox"
```

如果一切顺利你就可以按下“Run”按钮来执行脚本。现在启动脚本编辑程序来实践一下吧。

#Enter键是启动语法检查的键盘迅捷。Enter键的位置在笔记本键盘空格键的右侧、台式机键盘的数字小键盘上。Return键（在右侧Shift键上面）则是用于换行。你不能用Return键启动语法检查功能。

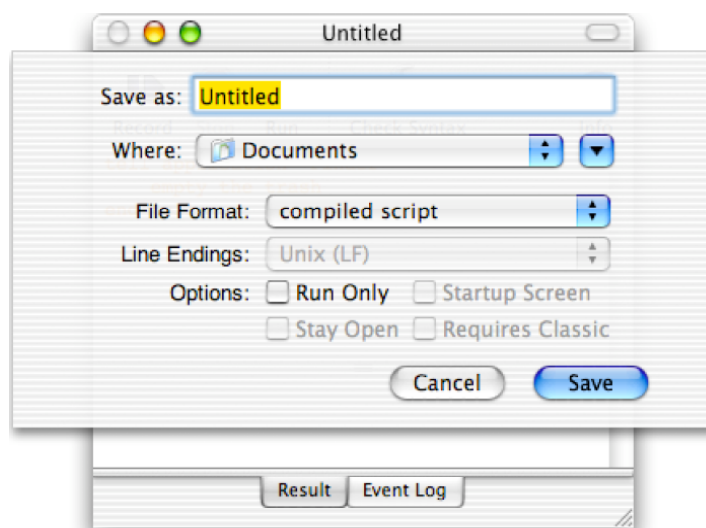
其实并没有必要在执行脚本前先按“Check Syntax”按钮。因为当你按下“Run”按钮，就会先检查脚本的语法，语法正确脚本才会被执行。

Command + R键是“Run”按钮的键盘迅捷。#

## 存储你的脚本

存储脚本有很多方法。但是如果脚本没有成功通过语法检查，那么你能只能把它存储为文本。

如果语法检查没有任何问题，就会出现下面的对话框，询问你将写好的脚本存储为“已编译的脚本”还是“应用程序”。



已编译的脚本（compiled script）：如果你双击它的图标，那么脚本编辑程序就会启动，你可以通过点击“Run”按钮来运行脚本。



应用程序（application）：如果双击它的图标，脚本编辑程序不启动，而是直接运行脚本。被保存为应用程序的脚本可以直接加载到“登陆项目”中（在“系统预置”面板）。账户登陆时你的Mac微机就会自动执行你编写的脚本内容。如果你要编辑一个已经被保存为应用程序的脚本，则需要启动脚本编辑程序，再从“File”菜单中的“Open”项打开这个脚本。



**#注意：要慎重使用存储对话框中“Run-only”一项。在选择这一项前，一定要对你的脚本进行备份，因为以“Run-only”存储方式的脚本不能再被编辑。#**

## 第3章 快速编写脚本（I）

在第1章中我们见到了这个脚本：

```
[1]
tell application "Finder"
    empty the trash
end tell
```

下面我们一起来看看AppleScript脚本编译程序如何使你方便迅速的编写脚本。

写tell模块的第一行时，你不必把单词“application”全部输入，你只需要键入：

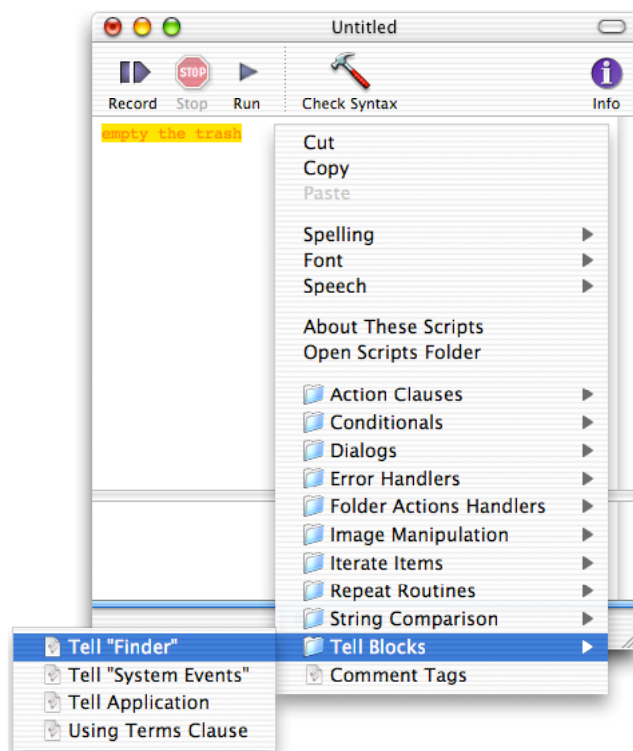
```
tell app "xyz"
```

编译时，脚本编辑程序会把“app”补全为“application”。你甚至不用键入或者不用知道脚本指向的应用程序的名称。随便键入几个字母（不能是某个特定应用程序的名称），比如“pqr”。编译脚本时，AppleScript会列出你的Mac微机上所有支持脚本操作的应用程序，你只需要选择适合的程序，AppleScript就会用正确程序名称代替“pqr”，实际上，就是帮你完成了语句的编写。

事实上，脚本编辑程序还可以允许你通过上下文菜单，不用键盘输入就创建出tell模块。所谓上下文菜单是你按住ctrl键再单击时出现的菜单。这个小技巧分两种情况：

1) 按住ctrl键，在脚本编辑程序窗口上半部分的窗格中单击鼠标。上下文菜单就会出现，在菜单底部你会看到名为“Tell Blocks”的菜单项，在这一项的子菜单中你可以直接选择“Tell ‘Finder’”。

2) 如果事先写好了一些语句行——比如“empty the trash”——但这些语句行并没有被包含在tell模块中，那么先选中这些语句行，之后按照1)来操作。如下图所示。这些语句行将会被自动的包含到tell模块中。



## 第4章 处理数字

小学的时候我们作过这样的填空题：

$$2 + 6 = ( \quad )$$

$$( \quad ) = 3 * 4$$

到了中学，填空过时了，我们改用x、y这样的变量（variables）来代替。回头看看，也许你想知道为什么这样微小的记号的变化会令那么多人感到恐慌。

$$2 + 6 = x$$

$$y = 3 * 4$$

AppleScript也使用变量。变量并不神秘，它是用来代替特定数据的名称，比如代替一个数。变量名通常被看作是一种标识，因为它们可以用来区分数据。例 [ 1 ] 中有两行AppleScript语句，其中的变量分别被赋值，赋值使用的是set指令。

```
[1]
set x to 25
set y to 4321.234
```

尽管对AppleScript来说，变量名本身并没有特殊意义，但是描述性变量可以令脚本易于理解。特别是你在脚本中寻找错误（错误（error）在脚本和程序中习惯被成为“bug”）的时候它就显得特别有用。所以避免使用像x这样的非描述性的变量名。比如代表图片宽度的变量名可以叫做pictureWidth。参见例 [ 2 ] 。

```
[2]
set pictureWidth to 8
```

请注意，变量名由一个词组成（必要时是一个字母），即各个单词中间不要留有空格。

在进行语法检查以后，变量名被以绿色显示，所以你可以立即把它同以红色和蓝色显示的AppleScript保留字区分开。同时注意一下数据（比如例 [ 2 ] 中的数字8）是以黑色显示的。

# 尽管你在使用变量名上有充分的自由，但依然要遵循一些规则。也许这些规则令人讨厌。最重要的一条规则是你不能使用AppleScript中的指令或者其它保留字。比如set、say和beep等单词对AppleScript来说都是有特殊含义的。使用简明的单词组成变量名，比如pictureWidth，通常是安全的。为了保证变量名的可读性，推荐不同单词之间以首字母大写为标志。

下一条规则是一个变量名不能以数字开头，但数字可以出现在变量名中。另外，使用下划线“\_”也是可以的。#

现在我们知道了如何给变量名赋值，就可以进行计算了。AppleScript可以进行基本的数学计算，所以没有必要指向一个专门的计算程序来计算图片的面积。例 [ 3 ] 这段脚本就可以完成。

[3]

```
set pictureWidth to 8
set pictureHeight to 6
set pictureSurfaceArea to pictureWidth * pictureHeight
```

你可以使用下面这些符号，术语叫做运算符，进行基本的数学运算：

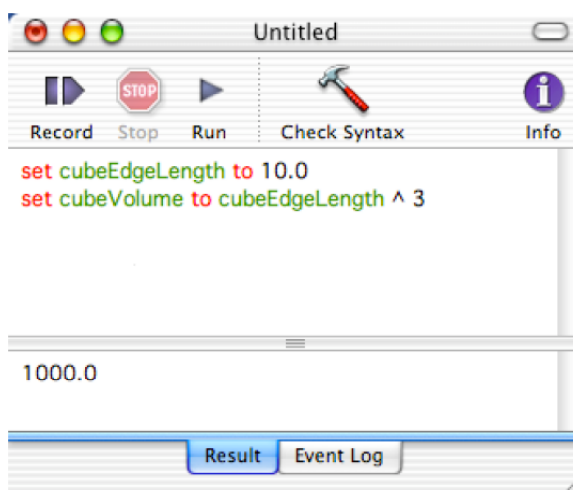
+ 加法   - 减法   / 除法   \* 乘法

乘方运算使用乘方符号“^”。例 [4] 给出的是立方的计算。

[4]

```
set cubeEdgeLength to 10.0
set cubeVolume to cubeEdgeLength ^ 3
```

在脚本编辑程序中运行例 [4]，结果会显示在下半部分的窗格中。如果你没有看到结果，则点击下部的水平栏上的标签切换到“Result”标签页。“Result”标签页的窗格中显示最后一行语句的执行结果。如果脚本只有第 [4.1] 行，那么结果应当是10.0。运行完整的脚本 [4] 将得到结果1000.0。这是因为表达式“cubeEdgeLength^3”进行了立方运算。



数字基本上分为两类：整数（integers）和分数（fractional numbers）。正如语句行 [1.1] 和 [1.2] 分别给出的。整数用来计数，比如重复执行若干次某一特定指令（见第13章）。分数或者称作实数（real numbers，简写作reals）用来计算例如棒球的击中率。顺便提一下，整数和实数都可以是负数，比如你银行帐户里的数字。



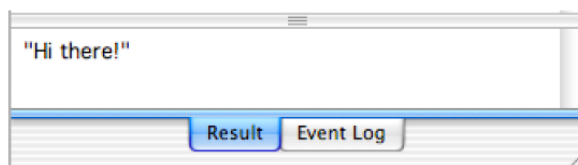
## 第5章 处理文本

变量不仅可以用来指代数字，也可以用来指代文本。一段文本，即使没有字符或者只有一个字符，它也被称为一个字符串。字符串必须被放置在双引号中。例 [ 1 ] 中有三个例子：每个变量名都对应着一个与它的描述相对应的字符串。

[1]

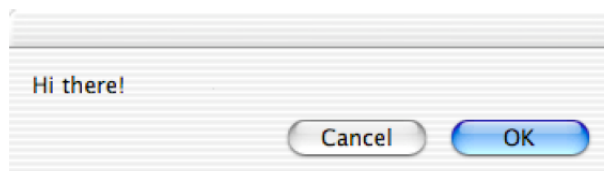
```
set emptyString to ""  
set notEmptyContainsASpace to " "  
set greeting to "Hi there!"
```

运行例 [ 1 ]，结果区显示的字符串也带有双引号。这说明结果区不仅显示结果的值，而且显示出结果的数据类型（数字不带双引号；字符串带双引号）。



由于结果区只能显示最后一行语句的执行结果，所以只有变量greeting所在的第 [ 1.3 ] 行的结果显示了出来。

除了结果区以外，AppleScript提供另一种方便的文本传达方式：对话窗口。如下图所示：



你可以使用这个指令调用对话窗口：“display dialog”，其后面还要加上你希望显示的数据（数字或字符串）。上面的对话窗口是使用例 [ 2 ] 给出的语句创建的。

[2]

```
display dialog "Hi there!"
```

为什么字符串要用双引号引起来？AppleScript由有限的词汇构成，阅读你的脚本时，判别哪些是指令哪些不是，对Mac微机来说是一件很复杂的事情。所以AppleScript需要依赖一些线索帮助其读懂脚本中语句行的各个元素的含义。这就是为什么要把字符串放到双引号里面。否则AppleScript可能将字符串误当作变量。看看例 [ 6 ]：

[6]

```
set stringToBeDisplayed to "Hi there!"  
display dialog "stringToBeDisplayed"  
display dialog stringToBeDisplayed
```

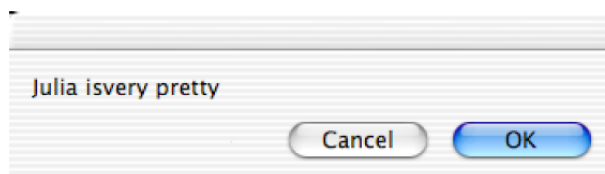
运行脚本例 [6] 试试，结果应当是，第 [6.2] 行语句显示字符串 “stringToBeDisplayed”；而第 [6.3] 行语句显示文本 “Hi there!” 脚本编辑程序将编译后的脚本分色显示。我们很容易看出在第 [6.3] 行的 “stringToBeDisplayed” 是一个变量名，因为它以绿色显示的；而第 [6.2] 行的 “stringToBeDisplayed” 是黑色的，表明它是一个数据（这里是一个字符串）。这种分色显示格式可以帮助你更快的捕捉到错误。

前面说到AppleScript需要线索帮助其将类似英语的脚本解释成Mac微机能够读懂的东西。下面还有一个例子来说明为什么这种线索很重要：如果我们用阿拉伯数字在双引号中写了一个三十，也就是 “30”，那么它就不再表示一个数字而是一个字符串。你要意识到数据类型是十分重要的，因为一些运算只能用于特定的数据类型。比如你可以对两个数做除法，但是不能对两个字符串做除法。下面看到的一些操作则只能用于字符串。

字符串可以像数字一样相加。你可以通过被称做 “连结” 的操作把字符串粘连在一起，这个操作要用到符号 “&”。参见例 [7]。

```
[7]
set nameOfActress to "Julia"
set actressRating to "very pretty"
set resultingString to nameOfActress & " is" & actressRating
display dialog resultingString
```

在第 [7.3] 行，我们连结了三个字符串，其中两个是用变量代替的。



请注意，字符串和 “&” 符号之间的空格数并不影响最终的显示结果，即变量resultingString中各个单词之间的空格数。脚本编辑程序在编译时自动将各个单词间的空格调整为一个。如果你需要在显示句子时单词间有多个空格，你需要把空格加到双引号中。在第 [7.3] 行，除了 “is” 左边的一个空格，字母 “s” 的右边还应该加一个空格。

还有很多指令可以用来操作字符串。其中一些指令要配合后面讲到的内容使用，所以我们到后面的章节再讨论。现在我们再来看另外一个与字符串有关的指令：查看字符串长度。参见例 [8]。

```
[8]
set theLength to the length of "I am"
```

运行这个脚本，结果区将显示结果4。所以请记住，空格也会占字符串的长度。

因为双引号被用作字符串开始和结束的标志，你可能会认为字符串中不能包含双引号。其实，AppleScript提供了一个解决方案——使用 “转义字符”（escaping）。将反斜杠 “\” 放到双引号的前面，AppleScript就不会把反斜杠后面的双引号当作是字符串结束的标记了。参见例 [9]。

[9]

```
set exampleString to "She said: \"Hi, I'm Julia.\""
```

#如果你细想一下，字符串里包含一对前面带反斜杠的双引号有时可能带来一些问题。比如，你想显示下面这段文本

```
blah blah \" blah
```

首先，你要在反斜杠前面加一个作为转义符号的反斜杠，这样AppleScript就会忽略后一个反斜杠的特殊含义。当然，你还要使双引号发生转义，否则AppleScript就会认为我们的字符串在这里结束。因此，还是你前面看到的，双引号前面带上一个反斜杠。把所有内容写在一起，我们得到了下面的例 [10]。

[10]

```
display dialog "blah blah \\\" blah"
```

为了让你看清楚，我用黑体标出了例 [10] 中的转义符号“\”，但是脚本编辑程序不会这样做。你务必留意反斜杠的使用，因为它放在特定字符前会产生特定含义。比如\n表示另起一行（换行），\t表示移动一个制表位。#

通过上面的学习我们已经知道了数字和字符串是不同的数据类型。你不能像例 [11] 那样从一个字符串中减去3。

[11]

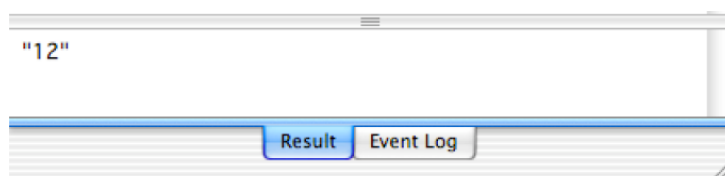
```
set nonsensical to "fifteen" - 3
```

如果试图运行例 [11] 给出的脚本，你就会发现AppleScript是多么友好的一个脚本语言了。它会试图将字符串内容转换成数字。如果字符串是“15”而不是“fifteen”，那么这种转化就可以生效。将一种数据类型转为另外一种的转化叫做“类型转换”（coercion）。你也可以像例 [12] 那样强制发生这样的转换。

[12]

```
set coercedToNumber to "15" as number  
set coercedToString to 12 as string
```

脚本编辑程序的结果区将显示出最后一行（第 [12.2] 行）语句的运行结果。变量coercedToString指代的是一个字符串，双引号已经清楚的说明了这点。



如果单运行脚本的第一行（第 [12.1] 行）语句，结果区将显示一个不带双引号的15，表明这是一个数字而不是字符串。

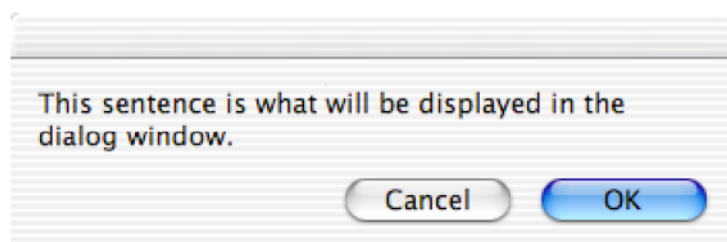
## 第6章 列表——list

在前面的章节中，我们已经学习了如何编写简单的脚本来进行基本的数学计算和字符串操作。运算的结果通常使用下面例 [1] 中给出 “display dialog” 指令反馈给脚本的用户。

[1]

```
display dialog "This sentence is what will be displayed in the dialog window."
```

运行例 [1]，你会看到一个对话框，这个窗口有两个默认的按钮：“Cancel”（“取消”按钮）和“OK”（“好”按钮）。

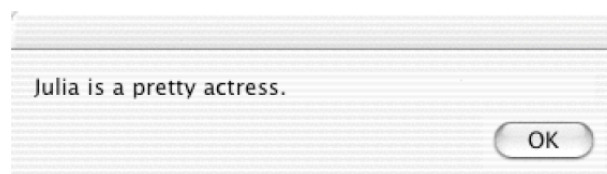


“Cancel”按钮通常用来阻止对脚本的进一步操作。因为上面的脚本不会有后续的操作，所以“Cancel”按钮是多余的。我们可以通过自定义对话框的按钮去掉它。“display dialog”指令允许你通过一个列表 (list) 来定义按钮。在我们这个例子中，实际上只需要一个“OK”按钮，见例 [2.2]。

[2]

```
set stringToBeDisplayed to "Julia is a pretty actress."  
display dialog stringToBeDisplayed buttons {"OK"}
```

运行上面的脚本，你会看到“Cancel”按钮已经被去掉了。



注意第 [2.2] 行，定义按钮的列表里仅仅包含一个字符串“OK”（OK带双引号），列表被大括号括起。为什么放在大括号里面呢？前面说过了，AppleScript需要一些线索帮助它理解语句中的每个元素的含义。大括号就是表明列表的线索。

第 [2.2] 行的列表中只包含了字符串“OK”一个元素。如果列表中包含多个元素，那么要用逗号将它们分割开，见例 [3]。

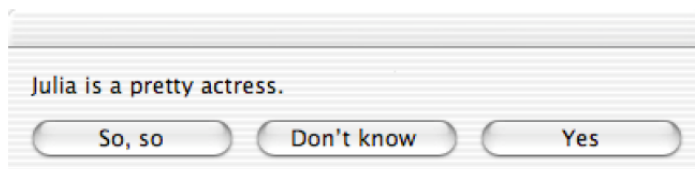
[3]

```
set exampleList to {213.1, 6, "Julia, the actress", 8.5}
```

上面例 [3] 中的列表包含4个元素：一个字符串和三个数字。现在再回到“display dialog”指

令的讨论上来，我们要创建一个带有多个按钮的对话框。AppleScript允许你通过“display dialog”指令创建带有1~3个按钮的对话框，你还可以定义每个按钮的名称。创建一个带有三个按钮的对话框，你需要在列表中写入三个项目，就像第[4.2]行那样。

```
[4]
set stringToBeDisplayed to "Julia is a pretty actress."
display dialog stringToBeDisplayed buttons {"So, so", "Don't know ", "Yes"}
```



# 你可能已经注意到了，通过第[2.2,4.2]行那样自定义按钮的时候没有按钮被高亮显示。也就是说运行脚本的时候你 cannot 通过按下回车键来使对话框消失。因为我们都很欣赏Mac微机界面的友好，所以我们下面要怎样设定高亮按钮。

```
[5]
set stringToBeDisplayed to "Julia is a pretty actress."
display dialog stringToBeDisplayed buttons {"So, so", "Don't know ", "Yes"}
                                         default button "Yes"
```

在指定高亮按钮时，如果不想写按钮的名称，也可以通过些按钮的序号来指定按钮。例[6]中“default button”后面的数字“3”表示被设定高亮的按钮是列表中的第3个按钮。

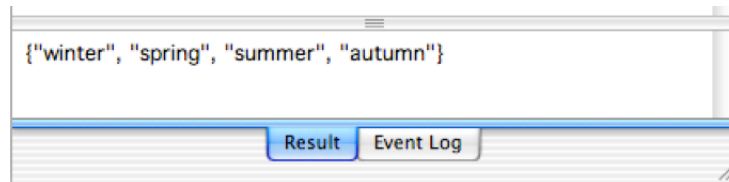
```
[6]
set stringToBeDisplayed to "Julia is a pretty actress."
display dialog stringToBeDisplayed buttons {"So, so", "Don't know ", "Yes"}
                                         default button 3

#
```

下面一章我们将讨论如何知道哪个按钮曾被按下。现在我们继续学习关于列表的知识。列表可以被用来存储一系列数据。所以你需要知道如何编辑列表以及如何取回列表中的数据。向列表的开头或结尾追加数据十分简单，我们使用“&”符号，就像操作字符串那样。

```
[7]
set addToBegin to {"winter"}
set addToEnd to {"summer", "autumn"}
set currentList to {"spring"}
set modifiedList to addToBegin & currentList & addToEnd
```

在第[7.4]行我们实际上创建了一个包含4个元素的列表。结果区显示的结果带有大括号，表明其数据类型是“列表”。



请注意第 [ 7.1, 7.2 ] 行的 “addToBegin” 和 “addToEnd” 是变量名，这样给变量取名是为了使你更好的理解上面讲的脚本，除了指代数据并没有其它什么作用。使用绿色显示已经很明白的说明了它们是变量的名称。

你可以使用元素的序号来指代元素。最左边的元素是元素1，其次是元素2，等等。你可以使用这种方式去从列表中调出具体值，也可以修改列表中元素的值（比如数值或数字）。请看例 [ 8 ]。

```
[8]
set myList to {"winter", "summer"}
set item 2 of myList to "spring"
get myList
```

上面第 [ 8.3 ] 行的 “get” 命令的作用是在结果区显示变量myList的值。根据脚本例 [ 8 ]，结果去应当显示的结果是： {“winter”, ”spring”}。

现在把注意力转移到第 [ 8.2 ] 行，用例 [ 9 ] 中的任何一个语句替换第 [ 8.2 ] 行都将收到同原有语句同样的效果。

```
[9]
set the second item of myList to "spring"
set the 2nd item of myList to "spring"
```

第 [ 9.1 ] 行语句体现优雅的AppleScript语言与英语语言类似的特点。但是这种一字母拼写的序数词最多只能使用到 “tenth” 。之后，就要使用 “item 11” 的形式，或者像第 [ 9.2 ] 行那样，写成 “11th item” 的形式。除了使用序数词，还可以使用 “last item” 指代列表中最后项目。参见例 [ 10 ]。

```
[10]
set myList to {"winter", "summer"}
set valueOfLastItem to the last item of myList
```

所以，当你只操作列表中最后一个值的时候，你不必知道列表具体有多少项目。

AppleScript允许你以相反的方向来指代元素，也就是可以从右向左数。这需要你使用负数， -1 指代最后一个元素， -2指代倒数第2个元素。例 [ 11 ] 可以获得和例 [ 10 ] 一样的结果。

```
[11]
set myList to {"winter", "summer"}
set valueOfLastItem to item -1 of myList
```

你已经知道了如何创建一个列表，如何在列表中添加元素、改变元素。你也学习了如何一次取回列表中的一个元素。现在你可能想知道如何一次取回列表中的若干元素。参见例 [ 12 ]。

[12]

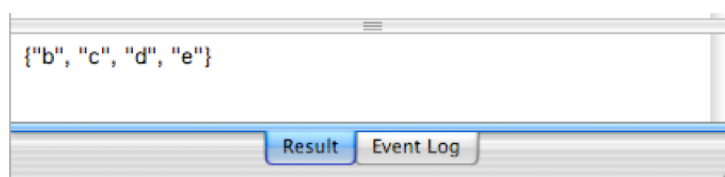
```
set myList to {"a", "b", "c", "d", "e", "f", "g", "h"}  
set shortList to items 2 through 5 of myList
```

如果你打字时喜欢用“ru”代替“you are”，你也可以用“thru”代替第 [12.2] 行的“through”。但如果你像第 [13.2] 行那样反向定义范围，你并不会反向的取出列表中的值。

[13]

```
set myList to {"a", "b", "c", "d", "e", "f", "g", "h"}  
set shortList to items 5 through 2 of myList
```

所以，脚本例 [13] 的运行结果和例 [12] 是一样的。



如果需要使列表中的元素反向，我们可以使用例 [14] 中提供的“reverse”指令。

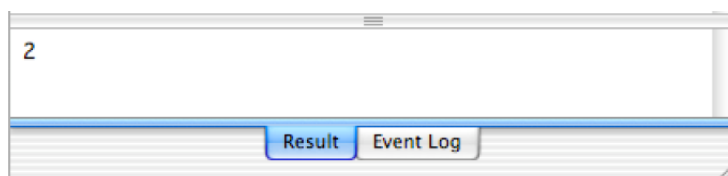
[14]

```
set reversedList to reverse of {3, 2, 1}
```

有些时候，你不得不知道列表中有多少元素，通过使用例 [15] 中的指令很容易得到答案。

[15]

```
set theListLength to the length of {"first", "last"}  
set theListLength to the count of {"first", "last"}
```



最后，你还可以随即指代元素，见例 [16]。

[16]

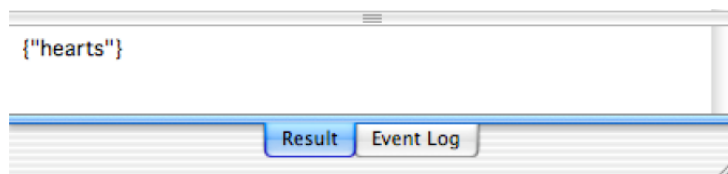
```
set x to some item of {"hearts", "clubs", "spades", "diamonds"}
```

我知道这是一个比较长的章节，但是我们不得不使用较多的篇幅同时讨论列表和字符串的问题。有些关于字符串的问题我不能放到第5章中，而必须留在这一章，因为第5章还没有讲到列表的概念。坚持到底！

前面的章节我们讨论了可以将一种数据类型转换成另外一种类型。现在我会教你如何将字符串或者数字转换成列表。见例 [17]。

```
[17]
set cardType to "hearts"
set stringAsList to cardType as list
```

将一个字符串转换成列表的结果是生成了一个列表，列表中包含这个字符串作为其一个元素。



# 当混合处理列表和字符串的时候，类型转换是一种十分安全的方式，下面将予以说明。

你的记忆中符号“&”可以用来连结字符串。但如果你将“&”用来联系一个字符串和一个列表会是什么结果？

```
[18]
set myList to {"a"}
set myString to "b"
set theResult to myList & myString
```

变量theResult的数据类型取决于第 [ 18.3 ] 行中运算表达式最先遇到的值的数据类型。因为表达式以变量myList开始，而myList指代的是列表，所以结果也将是一个列表。你来亲自试试，结果区显示的结果一定带有一个大括号。如果我们像例 [ 13 ] 那样把变量myList和myString调换一下位置，结果的数据类型会是一个字符串。

```
[19]
set myList to {"a"}
set myString to "b"
set theResult to myString & myList
```

毫无疑问，上面的例子告诉我们，如果稍不留心就很容易在编写脚本时产生错误。为了组织不必要的麻烦，你要主动进行转换，见例 [ 20 ]。

```
[20]
set myList to {"a"}

set myString to "b"
set theResult to (myString as list) & myList
```

在第 [ 20.3 ] 行中指代字符串“b”的变量myString被转换成列表 { “b” }，因为变量myString已经变成了列表，第 [ 20.3 ] 行结果变量theResult的数据类型类型自然也是列表。

在列表末尾添加元素，而不是使用连结的方法，你可以使用一下指令：

```
set mylist to {1, 2, 3, 4}
set the end of mylist to 5
get mylist
```



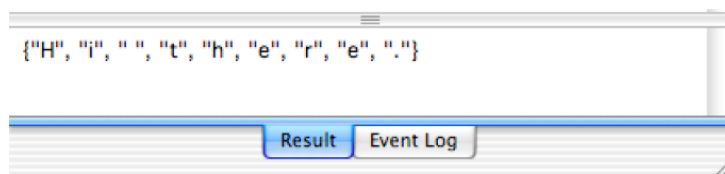
这个方法我认为更便捷。#

除了通过类型转换将一个字符串变成一个列表，你还可以创建一个列表，列表的元素是组成字符串的每一个字母。， 见例 [21]。

[21]

```
set itemized to every character of "Hi there."
```

在结果区将显示如下结果:



相比单个字母，你可能更想把一个整句按单词断开。你可以通过苹果脚本文本去限器（AppleScript's text item delimiters）实现。首先定义一个字符作为分割文本的标记，以这个标记分割出来的元素将被包含在列表里。要把句子分割成一个个单词，你需要以空格为分割标记。看看例 [22]。优秀的脚本编写要求如果苹果脚本文本去限器的值被更改了（第 [22.3] 行），一旦完成任务还要将它改回原来的值（第 [22.5] 行）。

[22]

```
set myString to "Hi there."
set oldDelimiters to AppleScript's text item delimiters
set AppleScript's text item delimiters to " "
set myList to every text item of myString
set AppleScript's text item delimiters to oldDelimiters
get myList
```

仔细看第 [22.4] 行，在什么位置出现了“text item”，不能省略成“item”。这是一个成见的错误。一定写“text item”、“text item”、“text item”，记住了么？

把列表转成字符串也很容易，见例 [23]。

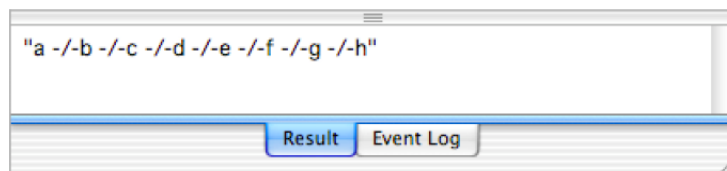
[23]

```
set myList to {"a", "b", "c", "d", "e", "f", "g", "h"}
set myList to myList as string
```

如果你需要一个或若干个字符加在字符串中来分隔原来列表中的元素，你只需要照着例 [24] 那样，记得要正确设置苹果脚本文本去限器。

[24]

```
set myList to {"a", "b", "c", "d", "e", "f", "g", "h"}
set oldDelimiters to AppleScript's text item delimiters
set AppleScript's text item delimiters to " -/-"
set myList to myList as string
set AppleScript's text item delimiters to oldDelimiters
get myList
```



将例 [ 22 ] 、 [ 24 ] 联合起来你就学会了如何对字符串进行查找和替换操作。

# 之所以要求必须将苹果脚本文本去限制器改回原来的值，是因为其他的脚本编写人员可能比较粗心大意，他写的脚本时可能认为苹果脚本文本去限制器就是默认值。而事实上，一旦发生了改变组成Mac OS X的AppleScript脚本程序会把这种改变保持到整个脚本运行结束。#

最后我们再来一起回忆一下重点：数据类型有很多种：数字、字符串、列表等等。每一种数据类型都有与之相适应的运算符对这种数据进行操作。许多的运算符也可以用来操作列表。本章我们学习了列表的知识，同时补充了上一章没有讲完的字符串的知识。

## 第7章 记录——record

前面的章节我们通过“display dialog”指令引出了列表（list）这种数据类型。现在我们还要使用这个指令引出另外一个数据类型。

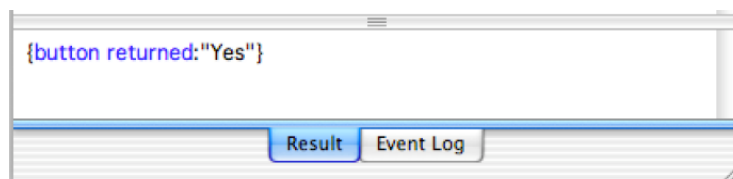
“display dialog”指令允许你通过一个包含不超过三个字符串的列表定义按钮和按钮的个数，见例 [ 1.2 ]。

```
[1]
set stringToBeDisplayed to "Julia is a pretty actress."
display dialog stringToBeDisplayed buttons {"So, so", "Don't know", "Yes"}
```

但是我们如何知道是哪个按钮被按下了呢？请接着看例 [ 2 ]。

```
[2]
set stringToBeDisplayed to "Julia is a pretty actress."
set tempVar to display dialog stringToBeDisplayed buttons {"So, so", "Don't
                                                         know", "Yes"}
```

运行例 [ 2 ]，你可以看到结果框里出现如下结果，这个结果取决于你按过了哪个按钮。



这里就出现了另外一种新的数据类型，叫做“记录”（record）。和列表类似，它也带有一个大括号；但不同的是记录由两部分组成，中间以冒号分隔。组成一个记录的要素通常是“标签 / 值”这样形式的一组属性。属性的前半部分是一个标签（或者叫做名称，这里是“button returned”）；后半部分是这个属性的具体值（这里这个值是“Yes”）。因为后半部分是一个值，脚本编辑程序用黑颜色显示它。

为了找出是哪个按钮被按下，我们要找出“button returned”标签下的具体值，见语句行 [ 3.3 ]。

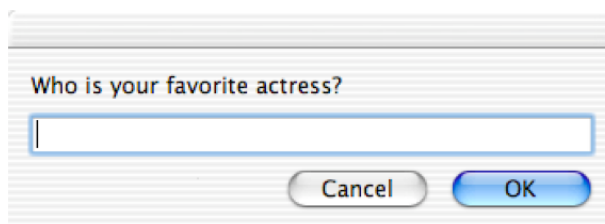
```
[3]
set stringToBeDisplayed to "Julia is a pretty actress."
set tempVar to display dialog stringToBeDisplayed buttons {"So, so", "Don't
                                                         know", "Yes"}

set theButtonPressed to button returned of tempVar
display dialog "You pressed the following button " & theButtonPressed
```

在第 [ 3.3 ] 行我们将记录tempVar中标签为button pressed的值赋给变量theButtonPressed。这样我们就可以知道对话框中的哪个值曾经被按下了。

对话框的局限性在于它只能显示数字和（较短的）字符串。它不能显示列表和记录。相比之

下，结果窗口可以显示所有的数据类型。尽管如此，对话框还有其它有用的功能：它能够接收用户输入的文本或着数字供脚本使用。

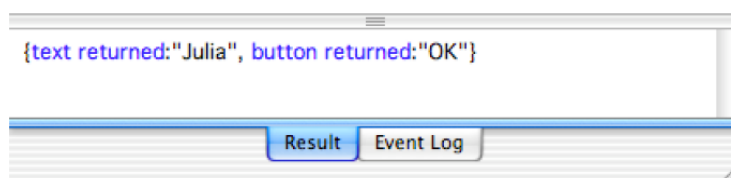


要使用输入框，你还必须提供一个字符串作为默认输入内容，比如一个空字符串“”，参见例 [4]。

[4]

```
set temp to display dialog "Who is your favorite actress?" default answer ""
```

运行例 [4]，结果是一个带有两组属性（即两组“标签 / 值”）的记录。你的结果框可能显示与下面类似的结果：



注意，就像列表中的元素一样，你需要用逗号分隔记录中各组属性。AppleScript不会把记录误认为是列表，因为记录中使用了冒号。你可能需要记住列表中每个元素和其对应的序号，相比之下，读取记录中的数据十分简便，你只要将操作指向属性的标签即可。比如要读取记录中女演员的名字，你只要向属性的标签“text returned”取值，见 [5.2]。

[5]

```
set temp to display dialog "Who is your favorite actress?" default answer ""
set textEntered to text returned of temp
```

# 请注意，标签“text returned”的值是一个字符串，也就是一组文本，即使用户输入的是数字。比如用户输入30，那么变量textEntered的值是字符串“30”而不是数字30。如果你想利用输入的数字进行计算，那么你很幸运，AppleScript会在第 [6.3] 行自动转换字符串的数据类型，使之成为数字。所以，在 [6.2] 行的后面加上第 [7.1] 行给出的类型转换语句。

[6]

```
set temp to display dialog "What is your age?" default answer ""
set ageEntered to text returned of temp
set ageInMonths to ageEntered * 12
display dialog "Your age in months is" & ageInMonths
```

[7]

```
set ageEntered to ageEntered as number
```

上面脚本的类型转换在用户输入数字后，比如30，会自动起作用。但是如果你输入的是thirty或者30 years，AppleScript将不会进行数据类型的转换，脚本运行将会出现错误。因为你不能保证所有用户按照你的意愿进行操作，所以编写脚本时必须考虑到用户的行为。在第10章我们将讨论如何处理这类问题。#

你可以通过将变量赋予“标签/值”这一组属性来创建自己的记录。

[8]

```
set personalData to {age:30}
```

注意，上面的记录中属性的标签“age”是以绿色显示的，表明是自定义的标签；属性中的值通常用黑色显示。

# 你可能注意到了在例 [3] 和例 [5] 中，属性的标签“button returned”和“text returned”是蓝色显示的，而且还是两个单词组成。这是因为它们是AppleScript中特殊的保留字。如果你创建一条记录，你不能使用两个或更多的单词来定义标签，标签（或者名称）只能是单个单词。

[9]

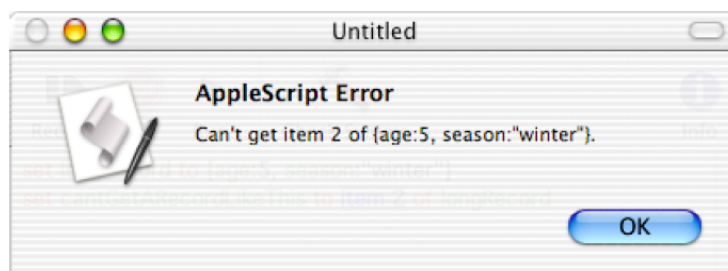
```
set improperlyNamedProperty to {my property: "This is not correct"}
set properlyNamedProperty to {myproperty:"This is correct"}
#
```

你可以想连结列表那样连结记录，但是要注意，如果被连结的记录中的属性包含相同的标签，那么会发生你不期望的结果。

不要将记录中的属性（property）叫成元素（item），因为即使下面的例 [10] 可以通过语法检查，它仍然会在运行时出错。AppleScript不允许将记录中的“标签/值”的形式叫做“元素”。

[10]

```
set longRecord to {age:5, season:"winter"}
set cantGetARecordLikeThis to item 2 of longRecord
```



使用例 [ 11 ] 中的指令，你可以知道一条记录中包含有多少个属性：

```
[11]
set longRecord to {age:5, season:"winter"}
set theNoOfProperties to the count of longRecord
```

在新创建的记录中包含原有记录的属性，你可以遵照例 [ 12 ] 的做法。

```
[12]
set longRecord to {age:5, season:"winter"}
set temp to the age of longRecord
set newRecord to {age:temp}
```

结果区显示的新记录是 {age:5}。上面的例 [ 12 ] 可以简单写成例 [ 13 ] 的形式，但是你可能觉得读起来不太容易。

```
[13]
set longRecord to {age:5, season:"winter"}
set newRecord to {age:age of longRecord}
```

但不幸的是你不能操作已经出现在记录里面的“标签 / 值”属性。也就是你不能创建属性标签的列表。同样的，你不能修改记录中的标签。如果你想使用一个新的标签，你只能创建新的记录，想例 [ 13 ] 那样。

# 最后，我们以一个另人头痛的陷阱结束本章。下面的例 [ 14 ] 还不会给你什么惊喜。

```
[14]
set firstValue to 30
set rememberFirstValue to firstValue -- A copy is made and stored by
                                         'refToFirstValue'.

set firstValue to 73 --Change the value of original variable.
get rememberFirstValue -- We ask for the value of 'refToFirstValue'.
```

例 [ 14 ] 的结果是30。但如果是记录或者列表，结果将完全不同，这样的错误还会很难被发现。请看下面例 [ 15 ]。

[15]

```
set personalData to {age:30}  
set rememberPersonalData to personalData  
set age of personalData to 73  
get rememberPersonalData
```

例 [15] 的结果是 {age:73}！在包含记录或列表时set指令没有复制数据。为了确保数据被复制，你应当使用copy指令，见例 [16]。

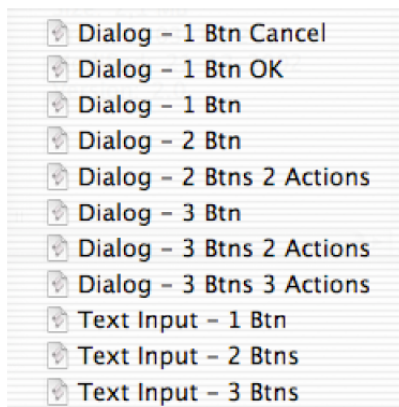
[16]

```
set personalData to {age:30}  
copy personalData to rememberPersonalData  
set age of personalData to 73  
get rememberPersonalData
```

#

## 第8章 快速编写脚本（II）

在前面的章节我们已经学习了几种显示对话框的方法。如果你觉得你记住了所有的内容，欢迎你到我家来作客，和我交流。如果你更喜欢简单的方法，那么只需要你记住按住ctrl键单击脚本编辑程序的上半部分窗格调出上下文菜单。其中有一项名为“Dialogs”的菜单项，单击展开其下的子菜单。



这里的“Btn”是按钮（button）的简写。它前面的数字代表按钮的个数。现在先不要管那些名字中带有“Actions”的菜单项，到了第10章我们再讨论那些项目。

子菜单上最后三个菜单项允许用户输入文本。在脚本编辑程序中输入下面的内容：

```
set temp to
```

确保在“to”后面留有一个空格。现在按住ctrl键单击那个空格后面，并且选择“Text Input - 2 Btns”一项。这样就自动生成了语句用于创建一个可以允许用户输入数据的对话框，就是我们在第7章中讨论“文本接收”属性时谈到的那种对话框。

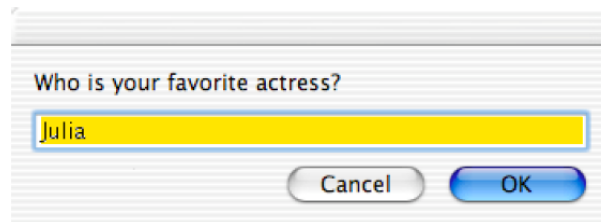
如果你想在为文本接收对话框提供一个默认的输入，用户在必要时还可以修改。如果你提供的默认回答很恰当，会使你的脚本显得十分友好，比如你提供了像例[1]那样的默认回答。如你所知，Mac微机的用户都喜欢友好的用户界面。但即使默认的回答不是很恰当，它依然给用户提供了一个范例，告诉用户应当回答什么内容。

[1]

```
set temp to display dialog "Who is your favorite actress?" default answer  
"Julia"
```

如果你运行例[1]，你会看到下面这个对话框。如果你认可默认的回答，只需要按下回车键；否则你可以输入其他人的名字。





简而言之，没有必要去精确的记忆那些生成对话框的指令代码。脚本编辑程序已经为你准备好了，你不用动键盘就可以把它们加入到你的脚本。

## 第9章 没有注释？那可不行！

事实证明AppleScript的很多特性使其脚本易读易写并且易于维护。这里有些特性是在你的掌控之外的，比如使用和英语相似的脚本语言；但又有些特性可以让事情按照你的意愿而定的，比如使用哪些单词组成变量名称。在这一章，我们还会讨论另外的重要特性。

我马上要为大家引出只有数行之长的例子，但在AppleScript中你能写的东西却很长。很重要的一点，你在编写脚本的时候不要仅仅急于让你的脚本完成你指定的任务，而且要适当的对脚本加以说明。过后，你在没有通读脚本就要做出修改时，注释可以帮助你了解某段脚本是做什么的、为什么写在这里。我们强烈建议你花些时间为脚本编写注释，而且敢保证，你一定会从这项工作中受益。当你和别人分享脚本时，别人也可以从你的注释中迅速获得关于这段脚本的信息。

创建一段注释要以两道连字号“--”开头。

```
-- This is a comment
```

经过编译（语法检查），注释被以灰色显示。

```
-- This is a comment  
-- This is a comment spanning more than  
   one line
```

过去，多行的注释被放在这两半符号之间(\* \*)。

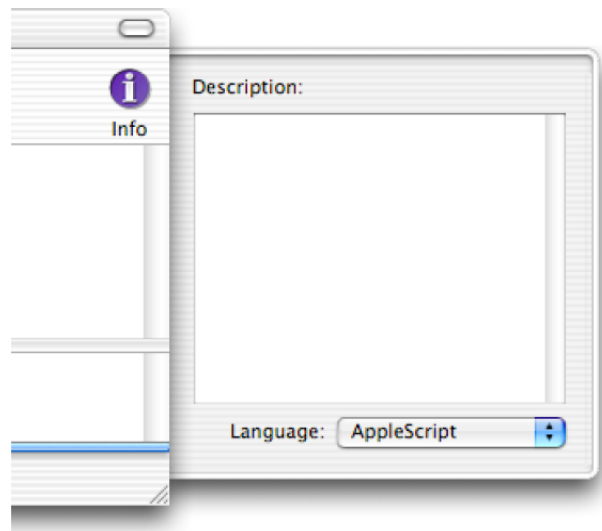
```
(* This is a comment  
   extending over two lines.*)
```

随着上下文菜单的出现，这个符号已经很少被拿出来做介绍了。你只要使用两道连字号就够了。这还要从一种叫做“放入注释法”（outcomment）的操作实践说起。

如果把脚本的某一部分放入符号(\* \*)之中，你将使这部分脚本暂时失效，这样就可以查看剩余部分的脚本是否可以正常执行，以此检查错误，这查错种方法叫做“放入注释法”。假如你放入注释的脚本正好要给某个变量赋值，你可以临时加上一行语句，为变量临时赋一个合适值用来测试脚本的其余的部分。

脚本编辑程序中上下文菜单的应用使“放入注释”更加简便。如果你希望把某段脚本放入注释，只需要选中这个部分（比如使用鼠标拖拽），之后按住ctrl键单击上半部分窗格，选择“Comment Tags”一项。你可以通过选择这个菜单项添加或删除注释标记。如果要删除注释标记，你需要保证带有注释标记的行被选中，选取不需要十分精确，然后仍然选择上下文菜单中“Comment Tags”一项。但是这个操作对于前面有两道连字号的注释不起作用。

在脚本编辑程序中，单击“Info”按钮可以展开一个抽屉，里面可以编写你对脚本内容的解释说明。此外希望你编写脚本以注释开始（使用两道连字号）。



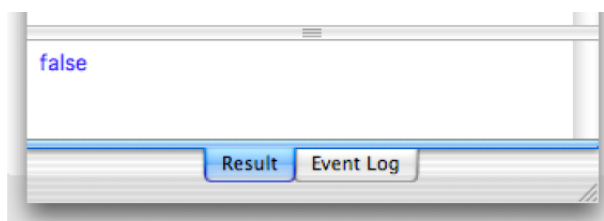
本书给出的例子并没有附加很多注释，因为它们已经被上下文的说明内容包围了。

## 第10章 条件语句

有时候，你希望只有特定条件满足时脚本才会执行动作。在脚本编辑程序中输入下面的例 [1] 并单击“Run”按钮。

```
[1]
73 = 30
```

你会看到这个结果。



AppleScript对上面例 [1] 的比较进行判别并得出“假”（false）这个结论，也就是“非真”（not true）。如果你输入的是比如“30 = 30”，那么得到的结果将是“真”（true）。

AppleScript比较两个值的功能用于“if...then”指令中可以达到满足条件才执行后续语句的目的，如例 [2]。我们把“if...then”指令叫做“条件语句”（conditional statement）。请注意，条件语句还需要一个“end if”收尾。后面我们继续深入讨论这个条件语句。

```
[2]
if true then
    -- actions performed
end if

if false then
    -- these actions are not performed
end if
```

我们要做的不过是用我们的比较条件替换第 [2.1] 行的“true”。如果比较条件为“真”，那么第 [2.2] 的语句就会被执行。

```
[3]
set ageEntered to 73
set myAge to 30
if ageEntered is myAge then
    display dialog "You are as old as I am."
end if
```

如果第 [3.3] 行的比较条件“ageEntered is myAge”是“真”，那么对话框就会被显示。但是从脚本的前两行我们可以知道，你不会看到这个对话框的。

如果条件满足后要执行的是若干指令，那么这些指令要全部被包含在“if...then...end if”模块中，见例 [ 4 ]。

```
[4]
set ageEntered to 73
set myAge to 30
if ageEntered is myAge then
    display dialog "You are as old as I am."
    beep
end if
say "This sentence is spoken anyway."
```

你可能发现了在tell模块和“if...then”模块之间有一些相似。他们都以一个包含单词“end”的语句结尾。“end if”帮助AppleScript识别哪些语句是满足条件后才能被执行的。在脚本例 [ 4 ] 中，不论条件是否满足，你都会听到第 [ 4.7 ] 行发出的声音。

当条件没有被满足的时候，我们还可以使用例 [ 5 ] 中的“if...then...else”指令进行选择。

```
[5]
set ageEntered to 73
set myAge to 30
if ageEntered = myAge then
    display dialog "You are as old as I am."
else
    display dialog "You are not as old as I am." -- [5.6]
end if
```

因为通过 [ 5.3 ] 行的比较，结果为“假”，所以第 [ 5.6 ] 行语句的对话框将会被显示。下面给出的是用于不同数据类型的比较运算符。

除了第 [ 5.3 ] 行出现的等号，下面的这些比较运算符都可以用于数的比较。

#### 用于数的比较运算符

=	is (or, is equal to)	等于
>	is greater than	大于
<	is greater than	小于
>=	is greater than or equal to	大于等于
<=	is less than or equal to	小于等于

上面的第二栏（英文部分）不仅仅是对符号的解释而且也可以被用在脚本中，见例 [ 6 ]。

```
[6]
if a is greater than b then
    display dialog "a is larger"
end if
```

如果你键入的是“>=”，语法检查后，AppleScript会自动将比较运算符转换成标准形式“≥”。

同样的，小于等于“<=”也会被“<”替代。你必须按照正确的顺序输入符号，否则AppleScript会报错，看来它并不是所有时候都很友好。

反义符号也可以在这里使用，比如“is not greater than”（不大于）。如果你输入“/=” ，会被自动转换成“≠”，它表示“is not”（不等于）。

第7章中我们已经用过了下面的例 [ 7 ] ，这段脚本用于反馈对话框哪个按钮被按下。

[7]

```
set stringToBeDisplayed to "Julia is a pretty actress."
set tempVar to display dialog stringToBeDisplayed buttons {"So, so", "Yes"}
set theButtonPressed to button returned of tempVar
```

使用“if...then”指令我们可以让脚本继续执行我们想要的动作，见例 [ 8 ] 。

[8]

```
set stringToBeDisplayed to "Julia is a pretty actress."
set tempVar to display dialog stringToBeDisplayed buttons {"So, so", "Yes"}
set theButtonPressed to button returned of tempVar
if theButtonPressed is "Yes" then
    say "I agree entirely!"
else
    say "Didn't you see the movie 'Pretty Woman'?"
end if
beep
```

如果用户按下了“Yes”按钮，那么第 [ 8.5 ] 句将会被执行。但在任何情况下，脚本都会以“咚”的一声结束运行。

实际上，AppleScript可以针对字符串进行复杂的比较运算。例 [ 9 ] 中有4个例子。你可能注意到了，如果“if...then”指令和要执行的语句写在同一行，那么就不再需要“end if”。

[9]

```
set textString to "Julia is a beautiful actress."
if textString begins with "Julia" then display dialog "The first word is Julia"
if textString does not start with "Julia" then beep
if textString contains "beau" then set myVar to 5
```

下面给出的是用于字符串的比较运算符。

#### 用于字符串的比较运算符

begins with (or, starts with)	以……开头
ends with	以……结尾
is equal to	一致
comes before	在……之前
comes after	在……之后
is in	在……之中

contains    包含

你还可以使用下面的反义运算符:

does not start with      不以……开头

does not contain        不以……结尾

is not in                 不在……之内

等等。

如果你写的是“doesn't”，它将被自动转换为“does not”。

如果你写的是“does not begin with”，它将被自动转换为“does not start with”。

比较运算符“comes before”和“comes after”对字符串逐字母比较。所以运行例 [10] 会发出“咚”的响声。

```
[10]
if "Steve" comes after "Jobs" then
    beep
end if
```

# 当进行字符串比较时，大小写至关重要。必须告诉AppleScript是大写或是小写。试试 运行例

[11] 的结果。

```
[11]
set string1 to "j"
set string2 to "Steve Jobs"
considering case
    if string1 is in string2 then
        display dialog "String2 contains a \"j\""
    else
        display dialog "String2 does not contain a \"j\""
    end if
end considering
```

默认情况下，空白（white space）（包括空格、换行和制表位）也被记入字符串的字数之内。

如果你不希望这样，可以使用例 [12] 中使用的“ignoring white space”指令忽略空白。但是请注意，用完要以“end ignoring”或者“end considering”结尾。

```
[12]
set string1 to "Stev e Jobs"
set string2 to "Steve Jobs"
ignoring white space
    if string1 = string2 then beep
end ignoring
```

你也可以要求AppleScript忽略标点符号、发音记号等等。#

对于列表来说，用于它的比较运算符要比用于字符串的比较运算符少，但是这些运算符的写法是一致的。

#### 用于列表的比较运算符

begins with 以……开头

ends with 以……结尾

is equal to 一致

is in 在……之中

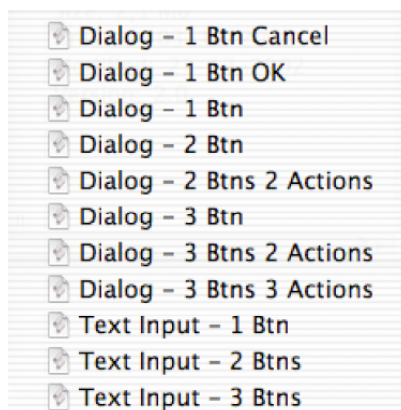
contains 包含

通常我们比较同一列表中的（或者不同列表中的）元素个体。我们来一起看一个例子。这个例子源自例 [8]，改成了带有三个按钮的对话框，并嵌套了“if...then”模块，使用户所有可能的操作都被考虑进去了。

```
[13]
set stringToBeDisplayed to "Julia is a pretty actress."
set tempVar to display dialog stringToBeDisplayed buttons {"So, so", "Who?",
"Yes"}
set theButtonPressed to button returned of tempVar
if theButtonPressed is "Yes" then
    say "I agree entirely!"
    beep
else
    if theButtonPressed is "Who?" then
        say "Didn't you see the movie 'Pretty Woman'?" -- [4.8]
    else
        say "I don't agree with you."
    end if
end if
```

# 如你所见，脚本中语句行的缩进错落有致，但仍然不太方便阅读。另外，大量的输入使你很容易忘记这句“end if”或者将它放错地方，从而导致脚本不能被编译。脚本编辑程序的上下菜单可以帮你大忙！还记得在Dialog子菜单下那些带有“Actions”字样的菜单项么？





选择带有3个按钮、执行3个动作的那项，所有你需要的语句将被显示出来。你要做的就是语句中填充上列表的具体值（第 [ 14.1 ] 行）并输入具体的动作。

```
[14]
display dialog "" buttons {"", "", ""} default button 3
set the button_pressed to the button returned of the result
if the button_pressed is "" then
    -- action for 1st button goes here
else if the button_pressed is "" then
    -- action for 2nd button goes here

else
    -- action for 3rd button goes here
end if
```

前两行需要加以解释。如你所见，第 [ 15.1 ] 行与第 [ 8.2 ] 行的不同之处在于我们没有把包含“display dialog”指令结果的记录赋给一个变量。虽然结果区总是自动显示最后执行的语句的结果，如果你使用关键字“result”指代了它们，前面语句的结果依然是可以使用的。如果“if”指令写在了和“else”所在行（即“else if”的形式），那么则不必为这个“if”指令单加一个“end if”，见 [ 14.5 ] 行。总的来说，上面的脚本还是比较好懂的。 #

对记录来说，比较运算符的个数更为有限，比适合于列表的比较运算符还要少。

#### 适合记录的比较运算符

is equal to (也可以使用 =) 一致

contains 包含

```
[15]
set x to {name:"Julia", occupation:"actress"}
if x contains {name:"Julia"} then display dialog "Yes"
```

# 你注意到了么？在第 [ 15.1 ] 行中，一个标签的颜色是蓝色的，另一个则是绿色的。这是告诉你其中一个标签是一个保留字。尽管AppleScript不阻止你使用和保留字同名的词作为属性标签，

比如“set”、“to”、“string”等，但是这可能给你带来麻烦。所以要尽量避免使用保留字，标签命名规则可以参考第4章关于变量名的命名规则。#

有限的适用于记录的比较运算符并不是大问题，因为很少对整个记录作比较，而是比较记录中属性的值，参见例 [16]。

```
[16]
set aRecord to {name:"Julia", occupation:"actress"}
if name of aRecord is "Julia" then display dialog "OK"
```

本章的最开始我们已经看到了，比较两个值（不论何种数据类型），你实际上是在判断比较表达式的真假。它是另外一种数据类型——布尔型数据（Boolean）。布尔型变量只有两种值：“真”或者“假”。对于数来说，你有“+”和“-”这样的运算符，你使用这些运算符得出的结果也将是数。对布尔型数据，我们有运算符“and”（逻辑与运算）、“or”（逻辑或运算）和“not”（逻辑否运算），这类运算的结果还是布尔型数据。

“not”是三个运算符中最简单的一个，“not true”就是“false”，“not false”就是“true”。

```
[17]
set x to not true -- So, x is false
```

下面的例 [18] 中使用了“and”运算，只有x和y同时为真，z才能为真。

```
[18]
set x to true
set y to true
set z to (x and y) -- z is true
```

相比较，使用“or”运算，x或y任何一个为真，z就是真。

```
[19]
set x to true
set y to false
set z to (x or y) -- z is true
```

为什么要讲这些内容？在有些情况下，你可能要求多个条件被满足时才执行某些指令。看看下面的例 [20]，只有最后一句定义的对话框会被显示。

```
[20]
set x to 5
set y to 7
set z to "Julia"
if x = 5 and y = 6 then display dialog "Both conditions met."
if x = 5 or z = "actress" then display dialog "At least one condition met."
```

在第 [20.4] 行，第一个比较是真，第二个比较是假。因为“and”运算要求两个都为真时才为真，所以对话框不能被显示。

在第 [ 20.5 ] 行是 “or” 运算，有一个条件满足真即可。

你可以类似第 [ 20.4 ] 和 [ 20.5 ] 行的语句中使用小括号，以提高脚本的可读性（和可靠度）。

## 第11章 避免错误

我们之前讨论的所有脚本，一旦出现错误AppleScript就会终止它的执行。

```
[1]
beep
set x to 1 / 0
say "You will never hear this!"
```

对上面的例 [1] 进行语法检查时，AppleScript不会提示任何错误。但一旦运行它，你并不会听到第 [1.3] 行被读出的声音，尽管第 [1.3] 没有错误，但第 [1.2] 行的错误会令后面的运行全部终止。

当然，意外的终止是我们所不希望的。比如，你的脚本需要打开一个文件夹处理其中的文件，但是这个文件夹已经被删除了，你会希望脚本允许用户选择其它合适的文件夹，而不是意外退出。

编写脚本时，你要学会辨识那些在运行时容易引起错误的语句。你可以把这些可能引起运行错误的语句放入“try...end try”模块中，参见下面例 [2]。

```
[2]
try
    beep
    set x to 1 / 0
    say "You will never hear this!"
end try
say "The error does not stop this sentence being spoken"
```

现在我们运行这个脚本，你就能听到第 [2.6] 行被读出的声音了，因为AppleScript会继续运行“end try”（第 [2.4] 行）指令之后的语句。

在第7章讨论记录的时候，我们已经见到过例 [3] 中给出的这个脚本。

```
[3]
set temp to display dialog "What is your age in years?" default answer ""
set ageEntered to text returned of temp
set ageInMonths to ageEntered * 12
display dialog "Your age in months is " & ageInMonths
```

这段脚本存在的问题是如果用户输入了其它内容而不是一个数字的时候，脚本就会崩溃。其实我们可以避免这种崩溃并且给出用户相关的提示信息，见 [4]。

```
[4]
set temp to display dialog "What is your age in years?" default answer ""
set ageEntered to text returned of temp
```

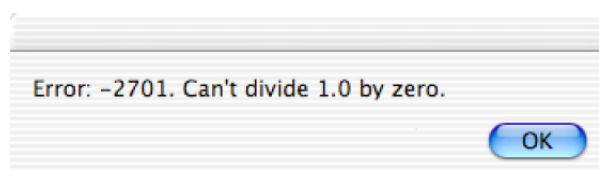
```
try
  -- First we check if the user entered a number
  set ageEntered to ageEntered as number
  set ageInMonths to ageEntered * 12
  display dialog "Your age in months is " & ageInMonths
on error
  -- If it is not a number, the entry must have been text."
  display dialog "Instead of a number, like 30 , you entered text."
end try
```

现在，如果用户输入的不是一个数字，他将得到一条提示。但脚本还有欠缺，那就是每次输入前用户必须重新运行它。在第13章我们会解决这个问题。

# 在脚本编辑程序中将语句行放入“try...end try”模块十分简单。选中要放入模块的语句行，激活上下文菜单，选择相应的选项，就像例 [ 17 ] 那样。

```
[17]
try
  set x to 1 / 0
on error the error_message number the error_number
  display dialog "Error: " & the error_number & ". " & the error_message
buttons {"OK"} default button 1
end try
```

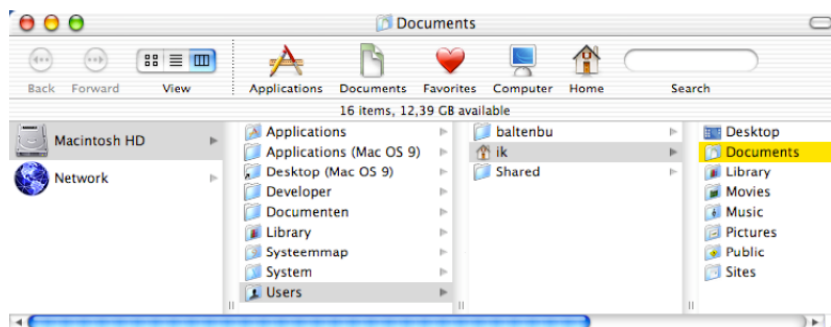
如果你在“on error”指令后面放上一个变量名，那么错误描述信息将被赋给这个变量。如果你在变量名前面加上“number”字样，那么错误代码将被赋给变量。我们让这两者同时在第 [ 17.3 ] 行被使用，你会看到类似下图的对话框。



#

## 第12章 路径、文件夹和应用程序

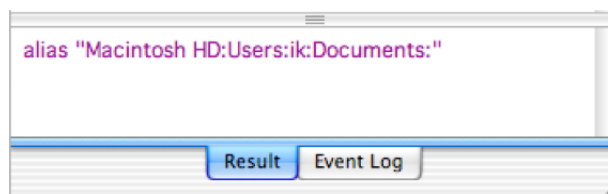
本章我们将一起学习组织管理硬盘上文件夹和文件的知识。如果打开一个以分栏方式（column view）显示的Finder窗口，你会看到类似下图的屏幕显示：



先是硬盘，硬盘下面包含文件夹、应用程序和文件（上图中没有显示出文件和应用程序）。所有这些元素按照一定的层次组织起来。这样我们就可以通过路径（path）这个概念来确定一个文件的位置。我们通过例 [1] 来看看所谓的路径是什么。

[1]  
choose folder

运行脚本 [1] 时如果我选择了我的“Documents”文件夹，那么会看到结果区如下显示：



你可以发现，路径一般的都符合这样的格式：

硬盘：文件夹：子文件夹：子文件夹：

上面截图中的显示就是按照这样的踪迹直到文件夹“Documents”的。现在把你的注意力转移到上面的截图中。请注意其中的冒号，它们是用来作分隔符的。这也就是为什么文件和文件夹的名称中不能使用冒号的原因。尝试在桌面上新建一个文件夹并用带有冒号的文字命名它，你将得到一个错误提示。最后，这个路径以一个冒号结尾，以此来表明这里指向一个文件夹。

我们可以使用路径让Finder打开指定的文件夹，见例 [2]。

[2]  
tell application "Finder"  
    open folder "Macintosh HD:users:ik:Documents"  
end tell

运行例 [2] 时（不要忘记将“ik”换成你自己的登录名），AppleScript并不介意你的路径是否

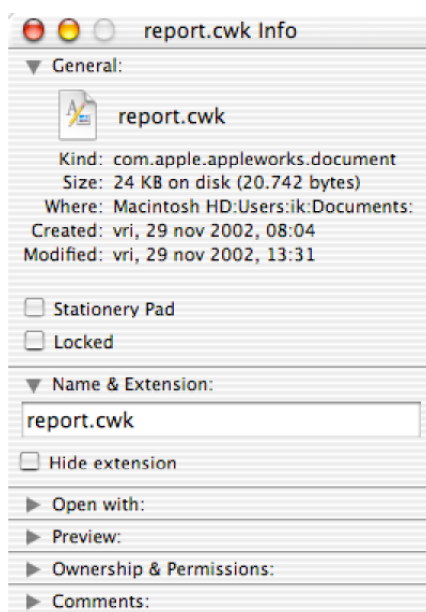
用冒号结尾。但是你一定记得对于硬盘的名称要正确使用其中的大写字母。

我的文件夹“Documents”中包含一个AppleWorks文件叫做“report”。我们使用例 [ 3 ] 来打开它。

```
[3]
tell application "Finder"
    open file "Macintosh HD:users:ik:Documents:report.cwk"
end tell
```

要注意的第一点是在第 [ 3.2 ] 行中要使用“open file”而不是“folder”，因为我们指向的是一个文件。第二点是文件名后面要带有扩展名。这里扩展名是“.cwk”，它是AppleWorks后继版ClarisWorks产生的文件。

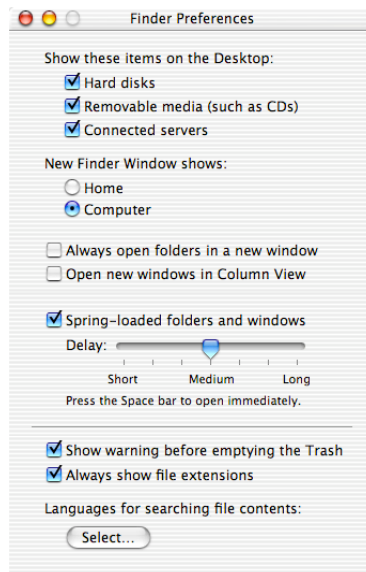
# 在Mac OS X系统中，默认隐藏扩展名。你可以选择文件，按command-i键调出“简介”窗口，在这个窗口里面你可以找到文件的扩展名。



你也可以通过修改Finder的设置让电脑总是显示扩展名。单击Finder菜单条上“Finder”项，选择“预置”（Preferences）一项。



在“预置”窗口中在“显示所有文件扩展名”（Always show file extensions）一项前面打勾。



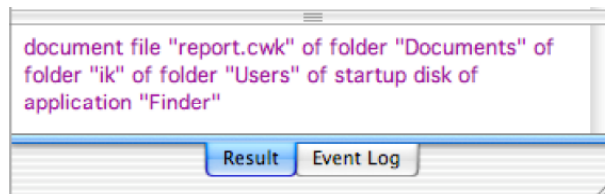
#

如果想用一个变量来指代文件“report.cwk”的路径，你可以参考例 [ 4 ] 的做法。

[4]

```
tell application "Finder"
    set thePath to file "Macintosh HD:Users:ik:Documents:report.cwk"
end tell
```

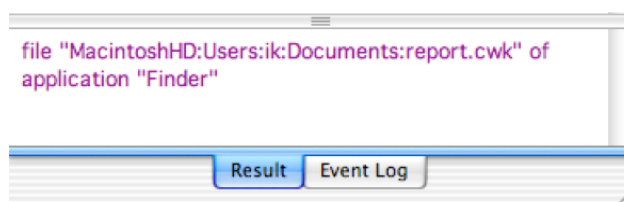
但是这样个脚本的结果并不是我们前面所看到的路径格式，而是如下图所示：



这个显示方法实在不方便阅读，特别是对于那些较长的路径，更糟糕的是，这种方式只能被 Finder 识别。你可能更喜欢，或者是确实需要使用有冒号作分隔符的路径显示方式。为此你需要强制 Finder 使用这种路径显示方式，你要做的是像例 [ 5 ] 那样，加上“a reference to”指令。

[5]

```
tell application "Finder"
    set thePath to a reference to file "Macintosh HD:Users:
                                     ik:Documents:report.cwk"
end tell
```



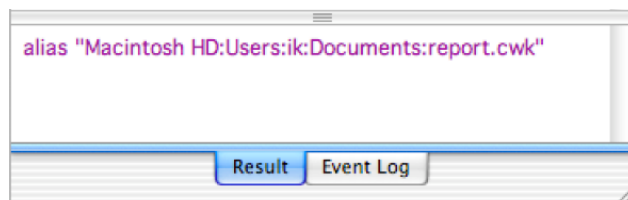
注意！运行上面的脚本，结果区说我们有一个 file（文件）。请现在运行脚本 [ 6 ] 并通过鼠标



选择同一个文件“report.cwk”。

[6]  
`choose file`

结果如下图:



这里出现的是“alias”（替身）而不是“file”（文件）！为了解释两者在AppleScript中的不同，我们首先来讨论什么是“替身”。作为Finder的使用者，你应该对这个名词比较熟悉。

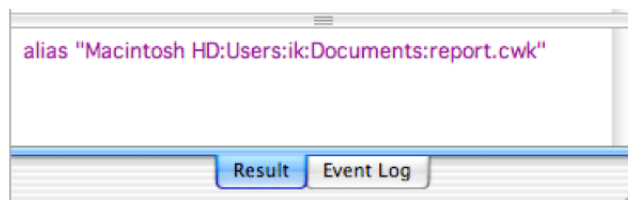
假如我在桌面上为“Documents”文件夹中的文件“report.cwk”创建了一个替身。如果今后我移动“report.cwk”到其它位置，或者把它重命名为“funny\_story.cwk”，双击替身依然能够打开这个文件。这是因为替身并不以“Macintosh HD:users:Documents:report.cwk”的方式记录文件“report.cwk”的存储位置（和名字），而是记录这个文件的ID。Finder有专门的数据库，里面保存着代表具体文件的ID和当前文件所在的位置。当我移动文件“report.cwk”时，代表这个文件的ID不变，Finder只是更新它的数据库，重新映射文件的新地址。当双击替身图标，Finder就通过替身提供的文件ID找到并打开具体的文件。

为了避免因为文件被移动或改名造成的脚本运行中断，我们应当让脚本记录文件的ID而不是“符号链接”的路径。为此我们可以像例[7]那样编写脚本。

[7]  
`set thePath to alias "Macintosh HD:Users:ik:Documents:report.cwk"`

很重要的一点，脚本例[7]中的语句指向的是“Documents”文件夹中的原始文件，而不是用户创建的文件替身，就像我前面假设的在桌面上创建的那个。其中“alias”是一个保留字，它表明经过编译（也就是语法检查），脚本将记录下文件的ID，并且在运行时也不会按照显示的出的路径向Finder查找文件位置，而是以文件的ID作为依据。

运行例[7]将在结果区得到如下结果:



尽管你看不到文件ID本身，但是路径前面的“alias”一词已经告诉你，脚本的内部是使用ID处理文件链接问题的，而不是“符号链接”的路径。现在，脚本编译后，即使你移动文件“report.cwk”到其它位置，脚本依然正常运行。我把文件移动到“Documents”下的“Miscellaneous”文件夹中，

没有改动例 [ 7 ] 中的脚本，运行结果也会随之改变。

```
alias "Macintosh HD:Users:ik:Documents:Miscellaneous:report.cwk"
```

现在你自己来试试。（再次提醒：你的路径与我的不同，因为你的登录名不是“ik”，而且，你也可能使用的是别的文件夹。）如果你将脚本存储为“已编译的脚本”或者“应用程序”，文件ID也将一同被存储，即使你将原文件改名或者移动，下次脚本依然可以运行。但是如果你将原文件删除，脚本在运行时崩溃就一点也不奇怪。

总结一下，你有两种在脚本中处理文件路径的方式。可以描述文件的位置（符号链接），也可以使用保留字“alias”使编译后的脚本可以忽略文件的移动和重命名。

如果在脚本中使用“alias”处理文件路径，当编译脚本时要确保原始文件是存在的。并且，这样的脚本不能保存成应用程序给别人使用，因为即使那人的Mac微机上相似的位置有同名的文件，但文件的ID是不同的。

```
# 一点关于例 [ 7 ] 的说明。过去我个人很惊讶：像例 [ 7 ] 这样的脚本没有使用指向Finder的tell模块就可以被执行，毕竟保存着文件ID和位置的内部数据库是属于Finder的。这是因为Finder是唯一能够提供这些信息的应用程序，而作为Mac OS X组成部分的AppleScript是知道这一点的，当AppleScript需要文件的ID时，表面在请求Finder，实际上是从后台直接访问数据库。这种动作是一个例外。在前面例 [ 3 ] 中打开一个文件就使用了tell模块，是因为并非只有Finder可以打开那个文件，别的程序，比如创建文件的程序也可以用来打开它。比如，一个jpeg格式的图片可以用Photoshop打开，也可以用浏览器打开。#
```

现在已经知道如何表示一个文件的位置了，我们就可以移动或者复制它们了，见例 [ 8 ]。

```
[8]
```

```
tell application "Finder"
    move file "Macintosh HD:Users:ik:Documents:report.cwk" to the trash
end tell
```

## 第13章 重复

前面见到的所有脚本中几乎每条语句都只执行了一次。但有些时候你可能会需要多次执行某一条或某一些语句。AppleScript对此问题提供一些解决方案。

如果你知道需要重复执行的次数，你就可以使用例 [ 1 ] 中的repeat指令。重复的次数必须是一个整数，因为使一个操作重复2.7次是令人匪夷所思的。

```
[1]
repeat 2 times
    say "Julia is a beautiful actress"
end repeat
say "This sentence is spoken only once"
```

与之前的tell、try、if...then指令相似，必须有一句“end repeat”指令作为重复的结束。

你也可以使用变量来表达重复的次数，见例 [ 2 ] 。

```
[2]
set repetitions to 2
repeat repetitions times
    -- Statements to repeat here
end repeat
```

例 [ 3 ] 是例 [ 2 ] 的一个逼真的应用实例。执行例 [ 3 ] 时，脚本的用户被允许在对话框窗口输入一个数值。因为输入的任何内容以字符串的形式存放到“text returned”标签下，所以我们必须将这个输入的值转换成整数。但如果用户输入了文本或者实数（分数）那么将不能转换。我们还要提前采取对策。

```
[3]
-- The user is allowed to determine how often the text string is spoken
set textToDisplay to "How often has the sentence to be repeated?"
-- The number '2' is displayed to give the user a hint of the type of
    answer expected
display dialog textToDisplay default answer "2"
set valueEntered to text returned of the result
try
    -- valueEntered is a string (not an integer), which may look like this: "2"
    -- Here we try to coerce the value entered by the user into an integer. If
        that doesn't work, the try statement prevents the script
        from being aborted
    set valueEntered to valueEntered as integer
```

```
end try
-- If valueEntered is of the appropriate class (i.e. integer), we can perform
  the repeat block. If not, we provide a dialog.
if class of valueEntered is integer then
  -- The repeat block is executed if the coercion did not fail
  repeat valueEntered times
    say "Julia is a beautiful actress"
  end repeat
else
  display dialog "You did not enter a (valid) number."
end if
```

在第 [ 3.21 ] 行的else指令后, 脚本的用户将得到一个错误提示, 告诉他应当如何做才是正确的。之后, 用户要重新运行脚本。这一点会让你的脚本在Mac社区中难以得到好的反响。例 [ 4 ] 、例 [ 5 ] 利用repeat指令, 直到条件满足时再进行下一步操作。

```
[4]
set conditionMet to false
repeat while conditionMet is false
  -- if (some test is passed) then execute the following statement
  --   set conditionMet to true
end repeat
```

```
[5]
set conditionMet to false
repeat until conditionMet is true
  -- if (some test is passed) then execute the following statement
  --   set conditionMet to true
end repeat
```

下面我们用例 [ 4 ] 中的语句改写例 [ 3 ] 。我们会重复要求用户输入, 直到他输入的内容可以被转换成整数。一旦类型转换成功, 脚本变量correctEntry的赋值将为true, 脚本就会跳出重复, 运行其余的部分, 见例 [ 6 ] 。如果用户输入的内容不能被转换为整数, 就会给出一条反馈。

[6]

```

set correctEntry to false
repeat while correctEntry is false
    -- The user is allowed to determine how often the text string is spoken
    set textToDisplay to "How often has the sentence to be repeated?"
    -- The number '2' is displayed to give the user a hint of the type
    answer expected
    display dialog textToDisplay default answer "2"
    set valueEntered to text returned of the result
    try
        -- The valueEntered is always string.
        -- Here we try to coerce the value entered by the user

        into an integer. If that doesn't work, we jump to the
        on error section
        set valueEntered to valueEntered as integer
        -- Setting correctEntry to true will end the loop
        set correctEntry to true
    on error
        -- We will give detailed feedback.
        try
            -- First we check if the user entered a fractional number
            set valueEntered to valueEntered as number
            display dialog "You entered a fractional number instead
                of an integer."

        on error
            -- If it is not a number, the entry must have been text."
            display dialog "Instead of an integer, like 9 , you
                entered text."

        end try
        -- Because the value of correctEntry is still false, the loop continues.
    end try
end repeat

-- The script can make it here only if correctEntry is true
CorrectEntry is only true if valueEntered is successfully coerced
into an integer
repeat valueEntered times
    say "Julia is a beautiful actress"
end repeat

```

请注意，“set correctEntry to true”一句非常重要。它必须是

- 在（第一个）try模块中；而且

- 在这句之后可能导致一个错误（这里是试图进行类型转换）。

否则，变量correctEntry赋予true值可以忽略条件是否已经达到（是否成功转换成整数）。

# 尽管例 [3] 也可以完成和例 [6] 一样的设计任务（按指定次数说出句子），但是它的界面不够友好，脚本也不够完善，也就是说它会在用户输入错误的数字时发生崩溃，并且一旦用户操作错误不能给出正确的提示信息。例 [6] 同样有待完善，应当增加变量valueEntered的输入上限（比如是用例 [7] 提供的指令）可以避免句子被说上1000次，否则例 [6] 可能被你的欲望扼杀。

```
[7]
if valueEntered > 5 then
    set valueEntered to 5
end if
```

写出完善的脚本需要你认真广泛的测试。试试输入文本、分数、整数、极端的数值等等来验证脚本可以正常的工作。例 [6] 中仍然有一个问题没有涉及到，那就是如果用户输入负数会发生什么情况。你不能把它转变成正数或者仅仅提示用户只能输入正数。有趣的是，例 [6] 不会因为输入负数而崩溃。你可以通过修改例 [1] 自己试试输入负数会发生什么情况。#

例 [4] 和例 [5] 中的repeat指令可以被用于任何用途。你可以生成一个循环来确认

- 用户选中了文件或者文件夹，
- 一个词汇出现在特定文档中，等等。

相比例 [4] 和例 [5] 中repeat指令的一般用途，例 [1] 和例 [2] 中使用这个指令以数字作为条件。这里还有一些这方面的例子。

```
[8]
repeat with counter from 1 to 5
    say "I drank " & counter & " bottles of coke."
end repeat
```

如你所见，你第 [8.1] 行在repeat指令中使用了变量，也就是“counter”。但是在repeat模块中你不能更改变量的值。

```
[9]
repeat with counter from 1 to 5
    say "I drank " & counter & " bottles of coke."
    set counter to counter + 1
end repeat
```

运行上面的脚本，没有一个句子会被跳过去，依然会从1到5读5次。

在第 [9.1] 行中，步长（stepsize）默认为1，如果需要不同的步长值，你可以参考第 [10.2] 行的语句。

```
[10]
repeat with counter from 1 to 5 by 2
    say "I drank " & counter & " bottles of coke."
end repeat
```

如果存在一个列表，且列表中所有的元素被用于某一个操作，你可以数出列表中元素的个数，并应用repeat指令进行循环操作，见例 [11]。

```
[11]
tell application "Finder"
    set refToParentFolder to alias "Macintosh HD:Users:ik:Documents:"
    set listOfFolders to every folder of refToParentFolder
    set noOfFolders to the count of y
    repeat with counter from 1 to noOfFolders
        -- actions here
    end repeat
end tell
```

AppleScript还有更好的替代方案，下面就给出。例 [12] 允许你决定一个被选中的文件夹内的子文件夹数。之后使用repeat指令创建一个列表，列表中的元素是所有可见文件夹的名字。

```
[12]
set folderSelected to choose folder "Select a folder"
-- To find out which folders are present in the selected folder, we have to ask
  the Finder to give us the answer.
-- Note: "every folder" does NOT include folders inside other folders. It is just
  the folders you would see if you'd open the folder in the Finder.
tell application "Finder"
    set listOfFolders to every folder of folderSelected
end tell
-- The result is a list of folder references (paths), which can be processed
  outside the Finder tell block
-- Outcomment all the following statements and use the result field to see
  that the list 'listOfFolders' contains items that look like this:
  folder "reports" of folder "Documents" of folder "ik" of folder "Users"
  of startup disk of application "Finder".
-- Only the Finder and the AppleScript component of Mac OS X can deal with
  such references.

-- The folder names are to be stored in a new list, which is created here
set theList to {}
repeat with aFolder in listOfFolders
```

```
-- We have references to folders, and because a reference contains the
    name of the folder, the AppleScript component of Mac OS X can
    obtain the name without having to rely on the Finder
-- If you want to find out another property of the folders, e.g., the size
    (in bytes) of the folders, a trip to the Finder is required (i.e. a
    tell block for the next statement)
set temp to the name of aFolder
-- Here we add the name to the list
set theList to theList & temp
end repeat
```



## 第14章 处理程序——handler

AppleScript与英语类似的特点使脚本易读易写。但要使这个特点发挥出来你也有一份责任：比如，你应该为变量选择具有描述性的变量名，适当编写注释等等。AppleScript中提供的“处理程序”（handler）能更加优化你的脚本的可读性。试想，如果你在脚本的不同地方使用了相同的语句行，不巧的是其中发现了错误，你就不得不一处处的修改这些错误。AppleScript提供了这样的功能，允许你将在脚本不同位置使用的相同语句行编成组，并给这个组命名，需要调用这些语句行的时候只要调用这个组的名字就可以执行这些语句行。

下面的例 [ 1 ] 是定义一个处理程序。

```
[1]
on warning()
    display dialog "Don't do that!" buttons {"OK"} default button "OK"
end warning
```

今后要使用这个处理程序，你只要在脚本中直接调用即可，见例 [ 2 ]。

```
[2]
warning()
```

脚本中调用语句可以出现在对处理程序的定义之前，也可以在之后。

脚本 [ 1 ] 中的处理程序显示的文本是固定的。当然，你也可以在调用的同时告诉处理程序显示什么样的文本内容。

```
[3]
on warning(textToDisplay)
    display dialog textToDisplay buttons {"OK"} default button "OK"
end warning
warning("Don't do that!")
warning("Go fishing!")
```

第 [ 3.1 ] 行中的变量textToDisplay用于接收处理程序被调用时传送来的值。（在第 [ 3.4 ] 和 [ 3.5 ] 行，每个括号里都有一个值，这些值被传送到处理程序。）当例 [ 3 ] 被执行，两个对话窗口会连续显示出来。

调用处理程序的语句的小括号中除了可以是具体值也可以是一个变量，见例 [ 4 ]。

```
[4]
on warning(textToDisplay)
    display dialog textToDisplay buttons {"OK"} default button "OK"
end warning
set someText to some item of {"Don't do that!", "Go fishing!"}
warning(someText)
```

注意，第 [4.5] 行中调用处理程序的语句小括号中变量名与第 [4.1] 行定义处理程序使用的变量名不同。所以，你不需要知道处理程序中使用的变量。当然，你要知道被调用的处理程序中使用的数据的数据类型。

你不仅可以将信息传送到处理程序，还能将处理程序处理过的数据取回。

```
[5]
on circleArea(radius)
    set area to pi * (radius ^ 2)
end circleArea
set areaCalculated to circleArea(3)
```

处理程序“circleArea()”计算 $\pi * 3^2$ 的值并将结果返回。尽管与get指令类似，但处理程序是将最后一句语句运行得出的值作为结果自动返回。为了确保你得到想要的结果，可以使用return指令（见例 [6]），但这需要多写出一些像第 [6.3] 行这样的语句。

```
[6]
on older(a)
    if a > 30 then
        return "older"
    end if
    return "not older"
end older
set theAge to older(73)
```

如果第 [6.2] 行的比较结果是真，那么第 [6.3] 行而不是最后一行的运行后的值将被作为结果返回。

向处理程序传送数值不仅仅限于单个值传送，见例 [7]。

```
[7]
on largest(a, b)
    if a > b then
        return a
    end if
    return b
end largest
set theLargest to largest(5, 3)
```

上面的脚本最多可以回传两个值。注意第 [7.1] 行语句中有两个变量用于接收传送来的值。相

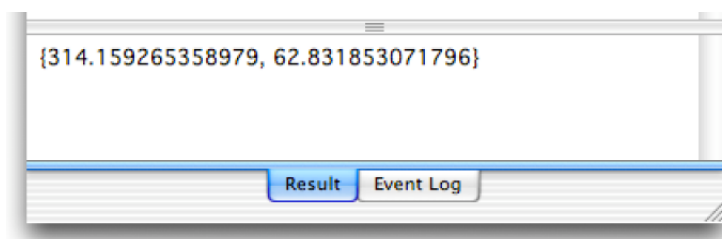
应的，在第 [ 7.7 ] 行调用这个处理程序的时候也要提供两个可以被传送的值。在现实使用的脚本中，被传送的两个值中至少有一个是变量的形式。

当然返回的结果也可以不只一个。下面脚本中第 [ 8.4 ] 行将结果数据以列表的形式返回。

```
[8]
on circleCalculations(radius)
    set area to pi * (radius ^ 2)
    set circumference to 2 * pi * radius

    return {area, circumference}
    set testVar to 3
end circleCalculations
set circleProperties to circleCalculations(10)
```

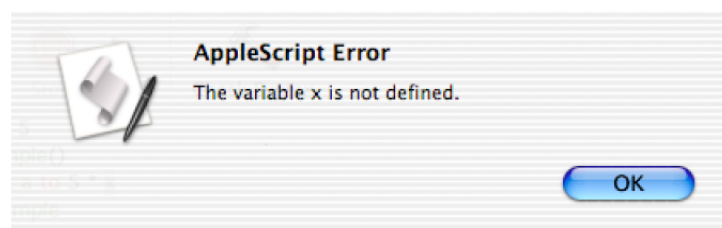
运行上面的脚本会返回一个包含两个值的列表，其中一个值是给定半径的圆的面积，另一个则是圆周的周长。



因为处理程序的特性，在脚本中使用它可以减少麻烦、节约时间。但这里还有一个问题（我将替你回答）：如果你的脚本中的一个变量和一段处理程序中某个作用完全不同的变量重名会怎样？不要担心，二者之间没有冲突。例 [ 9 ] 可以证明这一点。

```
[9]
set x to 5
on example()
    set a to 5 * x
end example
set y to example()
```

运行例 [ 9 ] 你将得到下面的错误提示：



在处理程序内部，变量x没有被定义。如果你想将第 [ 9.1 ] 行的变量x的值用于处理程序，你需要像例 [ 4 ] 中讲的那样向这个x传送数值。

此外，更改处理程序内部变量的值，并不影响处理程序外部同名的变量。例 [10] 可以证明这个性质。

```
[10]
set x to 5
on example()
    set x to 6
end example
example()
get x
```

这个脚本的运行结果是x的值5。所以，除了将值传入，将结果返回，处理程序和脚本没有可交互性。我要说的是倒是有一种办法，可以让第 [9.1] 和 [10.1] 行的变量x不通过传值就在一段处理程序中发挥作用。只是这种方法会令脚本难于阅读和调试。此外，还会使下面讨论到的处理程序的优点丧失掉。

除了前面说过的重要优点（比如增强了脚本的可读性等等），另外一个很大的优点就是你可以将处理程序用在其它脚本中。因为一段处理程序已经在某个脚本中测试并使用过了，所以你可以放心的将它应用于其它脚本。这样会为你的脚本编写工作节约许多时间。不要以为非处理程序形式的语句行可以被轻易的从一段脚本复制到另外一段脚本。你必须浏览全部脚本内容来确定哪些语句需要被复制，这会花费很多时间。此外，你还要承担脚本复制不全或不能正常运行的风险。相信我，使用处理程序是一条捷径。

# 好了，现在你可以享受在其它脚本中重复有效的利用处理程序带来的便利了。但是如果发现处理程序中有错误，或者今后还想增强功能该怎么半？你将不得不重写你的脚本。AppleScript为此类问题提供了解决方案——“load script”指令。你要做的只是：

- 1) 将一段或多段处理程序存储为已编译的脚本
- 2) 将使用“load script”指令的语句用于要调用处理程序的脚本

```
set aVariableName to (load script "path here")
```

为了使用被存储为已编译的脚本的处理程序，你要使用tell模块。

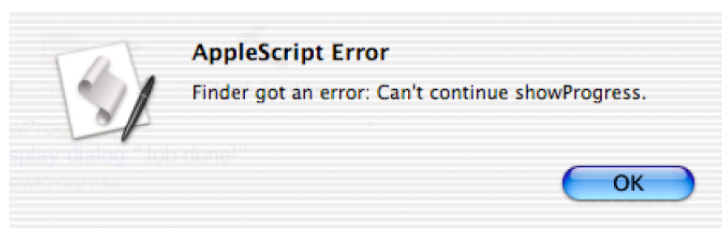
```
tell aVariableName
    handlerName()
end tell
```

```
#
```

处理程序还有很特殊的一点需要引起注意。当使用tell模块的时候，如果不加上一个范围，那么处理程序就不能在tell模块中被正确定义。运行脚本例 [11] 就会出现下面的错误提示。

```
[11]
on showProgress()
    display dialog "Job done!"
end showProgress

tell application "Finder"
    empty the trash
    showProgress()
end tell
```



对于这个错误解决方法很简单，只要告诉处理程序的范围就在脚本之内即可，见第 [ 12.7 ] 行。

```
[12]
on showProgress()
    display dialog "Job done!"
end showProgress

tell application "Finder"
    empty the trash
    showProgress() of me
end tell
```

这只对处理程序是必要的，而对于变量则不需要这样做，运行例 [ 13 ] 。

```
[13]
set x to 4
tell application "Finder"
    set x to 5
end tell
get x
```

你得到的结果是5。

## 第15章 信息资源

本书的目的是教会你AppleScript的基础知识。如果这本书你已经读过了两遍以上，并且也尝试了书中的例子，甚至自己改动了例子，那么你已经可以学习使用脚本编程了。

给那些要动手编写脚本的人一个重要的忠告：先别着手做！可能已经有人写好了你需要的东西。找到到它，稍作修改变成你需要的脚本，这样可以节省大量的时间。上哪里去找这些脚本？当然是因特网上。你可以浏览苹果的站点，里面可能会找到有价值的链接：

**<http://www.apple.com/applescript>**

我建议你下面的站点加入书签中：

**<http://macscripter.net>**

Macscripter网站中的论坛允许你发布你的问题。热心的网友和网站的工作人员会尽己所能帮助你。是的，我们说的是那里的Macintosh社区。这个网站还提供大量的链接和信息资源。

我衷心的希望你能喜欢这本书，并且能继续坚持学习AppleScript。噢，对了，请不要忘记了第0章。

作者：Bert

## 译者后记

这是我翻译的Bert Altenburg的关于苹果电脑的第二本手册。有了之前翻译《Xcode开发人员入门导引》（*Become an Xcoder*）的经验，这一本译起来相对容易一些。但仍然常常通宵工作到凌晨两三点。

如果说翻译前面一本手册的初衷是打发无聊的暑假，那么现在翻译这本《苹果脚本跟我学》（*AppleScript for Absolute Starters*）则是因为我深深为Altenburg对苹果电脑的热爱和挚着所感动。Altenburg总是在书的最开始用一整章的篇幅来呼吁苹果的用户要做些事情来“告诉世界并不是人人都用PC”。在他的影响下，我希望作为一位苹果用户的我，能以我有限的行动让更多的人关注苹果。这也就成了我翻译第二本手册的原因和动力。

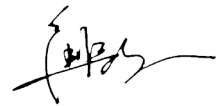
回过头来说AppleScript，它是苹果公司自行开发的一项脚本语言技术，具有简单易学易用的特点。但实事求是的讲，我对AppleScript的了解也不过是一知半解的水平。专业知识的欠缺成为了我翻译的最主要瓶颈。当我想找些材料佐证或参考时又发现手头可以看到的资料少得可怜。这就更坚定了我的心——一定要完成这次翻译的，因为我想，写出来起码可以给大家作作参考。

死党小杜，翻译上本书时教我C编程的家伙，南京一所著名高校的高才生，处处比我优秀，直教人心生嫉妒。好在他这次竟然肯“屈尊”为我的译本做校对，心理着实平衡不少。

谢谢网友张大鹏（音，网名loewez）对上一本译书中第65页图片制作失误的指正。这本书我会特别注意。

最后还要向弟弟小博道歉，因为赶翻译进度，害他自己一个人补作业补得昏天暗地。

不过还好，所有的工作到今天为止总算可以告一段落了。但我不能说是结束，因为手册中的错误和不足还期待您向我指正。真诚的给您道谢了。



二零零六年八月，与河北唐山寓所