AppleScript

简明基础教程

Ver 0.9，未校对版
2010年8月7日

iDoraemon Nathan 编著

写在前面

关于AppleScript：AppleScript很简单——保证比VB还简单，但很实用！学起来很轻松！学会之后你会发现你节约了很多时间，摆脱了很多机械性的无聊琐事。

关于创作目的：国内目前有关AppleScript的资料还非常少，我希望以最简单最简洁的形式来介绍尽可能多的AppleScript知识。简明是我的最终目的！

关于截图：本教程中所有截图都是依据Mac OS X 10.6简体中文界面进行的，在10.5系统中通常不会有明显区别，如果存在区别会在出现图片的地方加以说明的。特别声明：教程中图片均由我亲自制作，有特别说明的除外。

关于制作：使用Pages排版，Photoshop制图。

关于教程：本教程基本均为原创，以本人经验和国外相关网站作为参考（如developer.apple.com和macscripter.com），主要的权威参考书目为苹果官方的《AppleScript Language Guide》2008年3月版（可在官方网站上免费下载）。本人才学粗浅，涉水AppleScript时日也不长，如有疏漏和错误，请不吝指正。电子邮件：nocturn21st@gmail.com，有任何问题也欢迎在Macidea.com上发帖讨论。

关于发布：本教程目前暂时更新完毕，当然如果有需要，我会继续补充和更改。本人也非常希望其他网友能参与进来。

其他：AppleScript中译名应当为“苹果脚本”，但为了和系统统一，本教程中将不予翻译。

本教程为MacIdea而作！

iDoraemon Nathan

本版教程未经严格校对

可能存在大量文字录入错误，请谅解

目录

第一章 AppleScript入门

本章将初步介绍AppleScript的概念和基础知识，之后将简单讲解Automator的使用。前三节是讲AppleScript的一些套话——比如吹捧它的功能，你可以略看，不过仔细阅读也不会有任何损失。从第四节开始我就要求你必须认真阅读了。

第一节 什么是AppleScript

AppleScript的概念可以大致可以用下面几个词来描述:

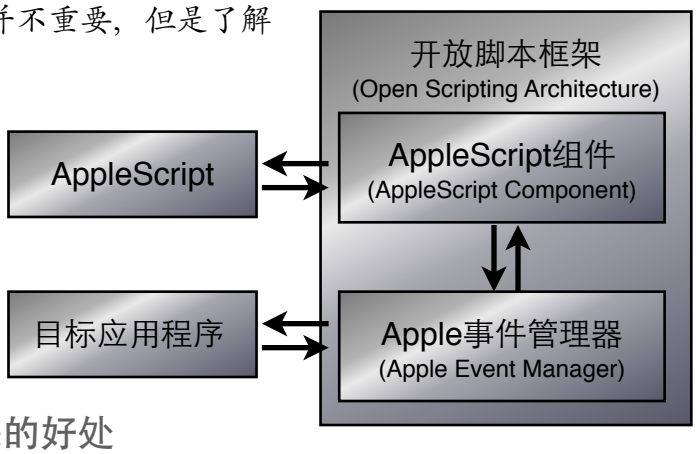
- 一种脚本语言
  和我们所知道的VBScript和JavaScript类似
- 内建于Mac OS
- 用来控制现有的应用程序
  请特别注意这一点！
- 使繁琐重复的机械操作自动化

第二节 AppleScript的工作机制

对于初学者初学者来说，工作机制并不重要，但是了解它终会有好处的。

如右图所示，AppleScript的工作机制中的四个部分均能实现双向交互。对于脚本编写者来说，只需要了解AppleScript和目标应用程序部分。

关于开放脚本框架(Open Scripting Architecture)我们暂时不需要了解。坦率地率，我认为不必要去了解。



第三节 AppleScript的用途和它带来的好处

AppleScript的用途举例:

- 批量图片处理
- 网站日常维护
- 文件和文件夹维护
- 包括Adobe系列软件和Microsoft Office在内的很多软件都提供了AppleScript支持
- 还有很多很多。。。

AppleScript带来的好处:

- 高效率
- 低出错率
- 更高的统一性
- 更高的精确度
- 免去你的操心

什么时候用ApplesScript？

- 当需要做重复机械性的且耗时的工作时
- 当你需要在未来某个时刻还要做一样的事情时
- 当写一个脚本比实际上做那件事更快时

第四节 和AppleScript有关的程序和设置

如果你没有接触过AppleScript，请务必仔细阅读本节。

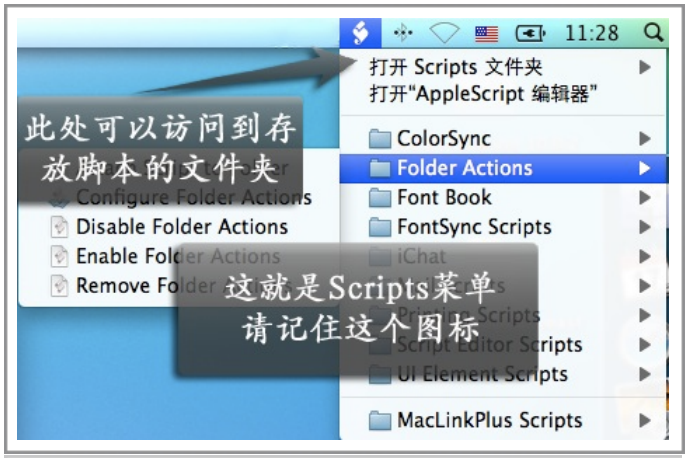
特别说明：AppleScript编辑器和设置工具在Mac OS 10.5 Leopard以及先前版本和10.6 Snow Leopard中有不同！请根据操作系统不同来调整。本节标题中括号内为Leopard和Tiger系统中的名称

AppleScript编辑器（脚本编辑器）

在10.5 Leopard和10.4 Tiger下：脚本编辑器和AppleScript实用工具位于“应用程序/AppleScript”文件夹中！

在10.6 Snow Leopard下：AppleScript编辑器位于“应用程序/实用工具”中；AppleScript实用工具已经不存在了，其功能合并入AppleScript编辑器，作为其偏好设置的一部分。

这个编辑器是我们用来编辑、调试乃至运行AppleScript脚本所必需的[1]。程序界面和其基本介绍如左下图。



在偏好设置（AppleScript实用工具）中打开“脚本菜单”

关于编辑器偏好设置（Leopard和Tiger中为单独的“AppleScript实用工具”），我只想提一样东西——“在菜单栏显示脚本菜单”（位于“通用”设置里）。

右侧的图片和下面几个问题是关于“脚本菜单的”介绍

- 脚本菜单是什么？

  就是预装的脚本——包括系统自带的和第三方提供的。它显示在菜单栏右侧（输入法的附近，如右图）。

- 脚本菜单用来做什么？

  快速打开已经编辑好的脚本。

- 如何添加自己的脚本到这个菜单？

  通过菜单第一项“打开Scripts文件夹”（本机的可被所有用户访问，用户的只能被当前用户访问到。），拷贝自己的脚本到这个文件。建议建立文件夹以保持整洁。

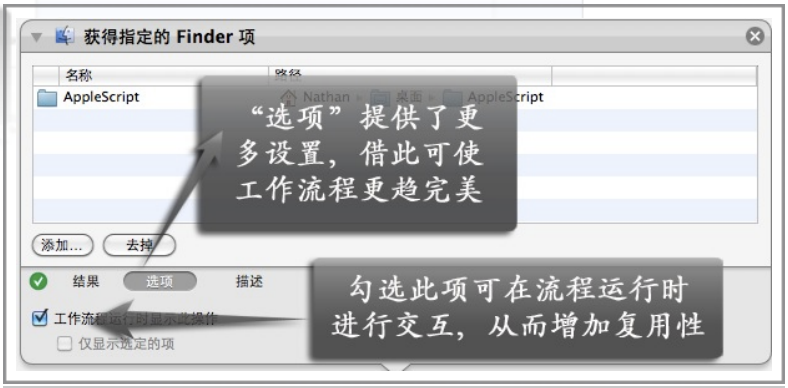
[1] 说是“必须”其实也未必，目前有第三方的AppleScript编辑开发软件，此外Xcode也提供了AppleScript的开发环境，同时Xcode也是所谓带有GUI的AppleScript Studio程序开发和编译所必需的。

第五节 Automator和AppleScript

Automator也是Mac OS自带的程序之一，他是一个“阉割版”AppleScript编辑工具，提供了直观的视图和简单的拖曳操作，但是功能上比AppleScript少很多（举例：Automator不支持循环）

下面图片中的流程可以实现对文件夹中所有图片进行批量格式转换。如果你有兴趣，请自己尝试下。（其实还是挺实用的噢）

优化工作流程：每个“指令”都有各自的选项，修改它们可以获得不同的结果，如右图



Automator虽然功能局限，但是并不代表它是鸡肋，它仍然具有很多的实用价值。本教程重点在于AppleScript的学习，故Automator将不作深入介绍。Automator程序作为AppleScript的简化版，大家有空不妨多多动手尝试。你一定会觉得Mac还真的好用！

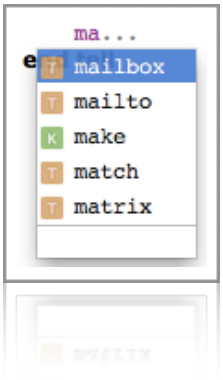
第二章 快速上手AppleScript编辑器[2]

本章将介绍AppleScript编辑器的一些“隐藏”功能，脚本的存储格式，以及应用程序对AppleScript脚本的支持情况，此外将通过一个实例来介绍并评价“录制脚本”功能。本章节对更好的理解AppleScript有一定的帮助。

第一节 挖掘实用的功能

在默认情况下，或者第一次使用AppleScript编辑器，你会发现这个编辑器和其他编程工具比起来非常不好用。但做一下几个简单的设置就可以让它迅速成为一个你的好助手。这里举两个例子，其他更多设置请读者按照自己喜好设置。

打开脚本助理



位于AppleScript编辑器的“偏好设置...”－“编辑”中。

此助理在默认情况下未打开，但是强烈强烈建议你勾选它！将会给你带来大大的方便。下面是主要的两个功能，第一个用处不大，但是第二个是非常有用的。

助理使用方法之一：输入代码时会以灰色字母或点来提示接下来应该输入的字母（有多个备选时以“..”表示，点的个数和字母数一致）

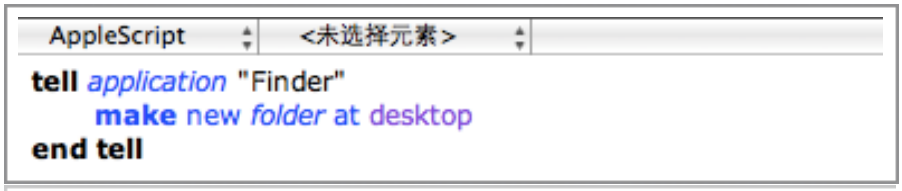
助理使用方法之二：只需输入代码的开头几个字母，按下F5键（根据你的设置，非常有可能是Fn+F5键），便会出现如右图所示的备选框。相信有编程经验的同学们不会陌生。

显示“tell application”弹出式菜单（仅限Snow Leopard）[3]

位于AppleScript编辑器的“偏好设置...”－“编辑”中。

为了直观说明这个特性，请对比下面两段代码（其实现的功能是一致的，都是在桌面上建立新文件夹），请留心图片上方的工具条。

未打开此功能时，需要输入三句：



打开此功能后，先在上方第二栏“tell current application”菜单中选择“Finder”，只需输入一句话：

第二节 脚本的存储格式

在“存储为...”对话框中，提供了四种备选方式来保存我们编写的脚本，即脚本、脚本包、应用程序和文本。这四种方式都比较常用。下面我将依次介绍。

---

[2] 在Mac OS 10.5以及10.4中，该名称为“脚本编辑器”。本书中所有“AppleScript编辑器”都是如此。请读者根据自己的操作系统进行修正。

[3] 此功能为“AppleScript编辑器” 2.3 (118)版本中提供的新特性，该版本随着Mac OS X 10.6提供，因此之前版本的用户将可能无法使用此功能。但是，旧系统用户下载2.3版的编辑器即可使用此功能。

脚本

这中保存方式直接将编辑的脚本保存为可运行（也许可以被编辑）的脚本，扩展名为.scpt。其不具有应用程序架构或者包结构。

在保存为脚本时，如勾选“仅运行”，将使得脚本不可被编辑，并且作为可执行文件打开，直接运行代码。

若未勾选“仅运行”，则其默认打开方式为“AppleScript编辑器”，代码可被编写并可更新该脚本文件。

脚本包

除了它具有包结构，扩展名为.scptd，其他和保存为“脚本”都一样。

所谓包结构，即在Finder中如右击（或者ctrl单击）该文件，会有“显示包内容”这个命令，其包中含有rtfd介绍文件、plist配置文件和sctp脚本。

此外，AppleScript编辑器窗口右上方的“包内容”按钮也将可用，并可修改其中内容。

应用程序

保存为扩展名为.app的应用程序，它将具有标准Cocoa程序的架构。它包内容含量比脚本包更多，进一步包含了图标，包简介(PkgInfo)，Unix可执行文件等等。

保存为此格式时，有三个选项：

- 仅运行：使应用程序包中的脚本不可编辑。
- 启动屏幕：使程序运行开始前显示一个对话框（包含description.rtfd的内容）。
- 保持打开：针对那些拖曳应用程序，使它始终处于可用状态。

文本

保存为扩展名为.applescript的纯文本文件。

第三节 支持AppleScript的应用程序

在第一章第一节中就已经提到了AppleScript是用来控制现有应用程序的，因此有必要了解哪些应用程序提供了AppleScript支持。

其实提供AppleScript支持的应用程序非常多，包括Finder、Adobe Photoshop、Microsoft Office[4]等等，这里不一一列举。查看程序是否支持AppleScript控制，请查阅对应程序的帮助文档。请注意，本教程将着重介绍Mac OS自带的程序（如Finder）。

应用程序对AppleScript的支持类型

即使支持AppleScript，不同的应用程序对它的支持方式不同，有下面三种类型：

- 可编程：通过输入脚本来控制应用程序
- 可录制：通过AppleScript编辑器的录制功能（将在下一节介绍）
- 可嵌入：应用程序支持AppleScript脚本交互式运行，应用程序在其菜单中有专门的脚本菜单（或者子菜单）。如Photoshop CS4中的“文件(File)-脚本(Scripts)”菜单

一个应用程序可以支持多种类型，并且很多程序都同时支持三种类型。

应用程序脚本的特点

- 应用程序通常都提供了自己的脚本控制指南
- 部分应用程序提供了示例脚本
- 不同应用程序的脚本编写难度不同
- 不同应用程序的脚本可进行的操作不同

---

[4] 此处应用程序均指Mac版本的对应程序。Adobe®，Microsoft®是各自公司的注册商标。

第四节 AppleScript的录制功能

“录制”允许用户通过最简单的方式来“输入”代码——记录你的每一步操作。Automator程序同样也支持“录制”。只需按下AppleScript编辑器左上角（Automator右上角）（如右图）的按钮就行了。

这个功能听起来很诱人，因为它简单易用并且可以帮助学习脚本语言。不过事实上它还是有很大的局限性的，主要体现在三方面：一是无法实现循环；二是将错误操作也记录了下来；三是代码质量低下，可读性差。

应用实例1：建立100个子文件夹

功能说明：这段脚本用于在桌面下建立一个名为“Test”的文件夹，并在其下建立100个子文件夹。

目的：现在你不需要理解这段代码！举这个例子的目的在于让你明白“录制”功能的局限性。你可是试试看录制出这段脚本（试想下，你需要确确实实地建立起100个文件夹）

以下为标准的代码[5]

```
tell application "Finder"
	make new folder at desktop with properties {name:"Test"}
	repeat with a from 1 to 100
		make new folder at folder "Test" of desktop with properties {name:a as string}
	end repeat
end tell
```

以下为录制的代码（部分节选）

```
tell application "Finder"
	activate
	make new folder at folder "Desktop" of folder "Nathan" of folder "Users" of startup disk
		with properties {name:"未命名文件夹"}
	set name of folder "未命名文件夹" of folder "Desktop" of folder "Nathan" of folder "Users"
		of startup disk to "Test"
	make new folder at folder "Test" of folder "Desktop" of folder "Nathan" of folder "Users"
		of startup disk with properties {name:"未命名文件夹"}
	set name of folder "未命名文件夹" of folder "Test" of folder "Desktop" of folder "Nathan"
		of folder "Users" of startup disk to "1"
	make new folder at folder "Test" of folder "Desktop" of folder "Nathan" of folder "Users"
		of startup disk with properties {name:"未命名文件夹"}
	set name of folder "未命名文件夹" of folder "Test" of folder "Desktop" of folder "Nathan"
		of folder "Users" of startup disk to "2"
	...

	--还有196句代码
	...
end tell
```

对比之下，我们很容易就发现录制出来的代码很糟糕。

[5] 根据本章第一节的介绍，可以通过“tell application”菜单来进一步精简代码。但是本教程考虑到与10.5和10.4系统的统一性，使用了最为标准的代码。

第三章 AppleScrip语言初步

本章内容是AppleScript的基础，内容比较多而且都非常重要，请读者仔细阅读。友情提示：看完后自己操作是最好的学习和复习的方法。

第一节 对象、属性和命令

AppleScript是一种面向对象（Object-Oriented，简称OO）的脚本语言，和现在主流的面向对象程序语言一样，它拥有三个重要的OO术语：对象（Object）、属性（Property）和命令（Command）[6]。

考虑到新手的学习，这里将通过一个实际生活例子——汽车来简单介绍下：

对象：又称“类”（Class），在生活中即一个具体的物体——汽车本身，当然也可以是一个部件，如轮子、车门、引擎——这三者作为汽车的一个组成部分，在OO概念里被称为子类。请记住，对象（类）是有可以拥有层次关系的。

属性：是对象的特性。如汽车的颜色、车身长度、重量都是对象汽车的属性。一个对象如果没有具体的属性，那么它将是抽象的或者是笼统的。

命令：又称“方法”（Method）。顾名思义，命令就是指令，它告诉脚本/程序应该去做什么。在非面向对象（面向过程）的语言中，命令或者方法可被成为“函数”。

第二节 标识符和关键字

首先值得说明的是：AppleScript的采用Unicode文字编码，并且不区分大小写。

标识符（Identifier）

标识符就是对象、属性、常量、变量等等的名称。就像人的姓名一样。

AppleScript规定：标识符必须以英语字母开头，可使用26个英语字母、10个阿拉伯数字以及下划线（_）。

特殊规则：如果标识符以“|”开头并结尾，则标识符可以使用任何Unicode字符，但是标识符名称本身是不包括“|”。

例：　abcd，ABC_91，a0abc，|a&b*c|，|中文名称|　　都是合法的标识符

　　91abc，_abc，中文名称，a&b*c　　不合法的标识符，请读者思考原因

建议：无论你是新手还是编程老手，强烈建议使用有意义的标识符，既方便自己也方便别人阅读。同时，避免使用符合“|”这一特殊规则的标识符。

关键字（Keyword）

关键字就是AppleScript保留的标识符，这些词通常拥有特殊含义（如被用于AppleScript语法），它们不能被用户定义为自己的标识符。

请注意部分关键字由两个词组成，AppleScript的关键字列表请参见本教程附录。

第三节 数据类型

数据类型（Data Type或者Value Class）可以由AppleScript语言本身定义，也可以是由应用程序定义的。在AppleScript常见的有以下几种：

Boolean（布尔型）

仅仅包含两个值：True和False

Number（数字型）、Integer（整型）和Real（实型）

如：1，2，1.0，1.1，3.14，-1.56

---

[6] 关于面向对象的三个重要术语，英语术语及其翻译和其他程序语言可能存在不同。请注意，这里采用了苹果官方《AppleScript Language Guide》（可在苹果开发者网站developer.apple.com上找到）中的说法。

Number类可进一步分为Integer（整数型）和Real（实数型）。

Text（文本型）和 String（字符串型）

如："This is a text"。 请注意引号为英文引号，以后都是这样。

在目前的AppleScript中，Text和String两个类型是一致的[7]。

Date（日期型）

如：date "2009年8月30日星期日 下午12:31:34"。

此格式的具体形式由“系统偏好设置-语言与文本”的相关设置决定。

Constant（常量型）

如：yes，no，ask

这些常量可以是已经被AppleScript预定义的，也可以是用户定义的不可变变量。这种类型的数据一经确定不可更改。此外可以认为所有关键字都是常量型的数据

List（列表型）

如：{1,2,3}，{{1,2},{a,b,c},}，{1,1.9, "text"}

列表型数据由{}包裹，一个列表中可以再包含列表，形成多维列表，列表里的具体数据可以是同类型的。

Record（记录型）

如：{firstName:"iDoraemon", lastName:"Nathan"}

记录就是带有名称的列表。记录中的每一项都有名称（标识符）。我们可以认为List是每个数据都是匿名的Record。Record也可以进一步包含另一个Record。

此例中，包含两个Text型数据 "iDoraemon"和 "Nathan"，它们的标识符分别是firstName和lastName。通过of关键字可以得到想要的数据：

```
firstName of {firstName:"iDoraemon", lastName:"Nathan"}
```

上面这行代码返回 "iDoraemon"。

确定数据类型-Class of

要确定一个数据到底是什么类型的，使用Class of语句

```
class of "string"
```

此代码得到结果text。

第四节 强制数据类型转换

有时默认的数据类型并不是我们想要的，比如"1"的数据类型为Text，而我们想要的是Integer；另一个例子是命令要求的参数类型与实际类型不一致时，我们就需要强制类型转换（Coercion）。

as关键字

用as关键字即可实现强制类型转换。用法：数据 as 类型

请注意，强制类型转换有时会丢失精度。

```
--文本类型转数字类型
"1.99" as real          --得到Real类型的1.99，而原来的数据是Text（因为带有引号）。
"1.99" as integer       --得到Integer类型的2，精度丢失！
"1" as real             --得到Real类型的1.0，自动提升精度！

--转换成List类型
"text" as list          --得到{"text"}
```

[7] String类和未提及的Unicode Text类是在AppleScript 2.0时代的东西，现在Text类已经完全代替了它们，但是Apple®为了方便起见，保留了这两个类。根据《AppleScript Language Guide》2008年3月版，“text, string, and Unicode text will all compare as equal”，三者完全一致。

```
1.99 as list          --得到{1.99}
{a:1, b:2} as list    --得到{1, 2}，精度丢失（标识符丢失）！

--错误举例
"text2" as number     --错误！Text中包含了无法转换成数字的字符。
1 as record           --错误！无法获得标识符。
{1, 2} as record      --错误！无法获得标识符。
```

第五节 运算符[8]

数学运算符（常见七种）

运算符	含义	举例	运算符	含义	举例
+	数学+	1+1 返回2	^	指数运算a^b即a^b	3^2 返回9.0
-	数学-	2-1 返回1	div	整除，舍弃余数	5 div 2 返回2
*	数学 ×	2*3 返回6	mod	除法取余数	5 mod 2 返回1
/或 ÷	数学 ÷	3/2返回1.5			

请注意：/和^两个运算结果在任何情况下均为Real类型的数值。div和mod运算符在任何情况下结果都为Integer。其他运算符根据输入的数值，结果可以为Real也可为Integer。

这里给编程达人一个提醒（请初学者忽略），AppleScript对除法的运算过程和代数学一样，不会出现其他编程语言（如C和Java）中3/2=1的情况[9]，AppleScript更贴近实际生活，请不要和其他编程语言混淆。

比较运算符

下表中所有运算符均返回Boolean（布尔型）数值。同一个格子的运算符中是同一功能的不同表达方式。从表格中不难看出，AppleScript在设计的时候的确很用心，语言很接近英语。

运算符	含义	运算符	含义
= equals is equal [is] equal to	比较两端是否相同 支持的数据类型非常多。	≠ is not isn’ t isn’ t equal [to] is not equal [to] doesn’ t equal does not equal	比较两段是否不同 支持的数据类型非常多。

[8] 本节中特殊符号通过按住Option键加符号来输入。如÷用option和/组合键输入，≥用option和>输入。

[9] 在Java和C中，两个整型变量相除得到的还是整数，这就产生了3/2=1的情况。而在AppleScript中，即使是整型变量相除，仍会得到小数。如：(3 as integer) / (2 as integer)结果仍然是1.5。

运算符	含义	运算符	含义
> [is] greater than comes after is not less than or equal [to] isn’ t less than or equal [to]	左边是否数值大于/文本长于/时间晚于右边 支持类型: integer, real, date, text	⩾ >= [is] greater than or equal is not less than isn’ t less than does not come before doesn’ t come before	左边是否数值大于/文本长于/时间晚于或等于右边 支持类型: integer, real, date, text
< [is] less than comes before is not greater than or equal [to] isn’ t greater than or equal [to]	左边是否数值小于/文本短于/时间早于右边 支持类型: integer, real, date, text	⩽ <= [is] less than or equal is not greater than isn’ t greater than does not come after doesn’ t come after	左边是否数值小于/文本短于/时间早于或者等于右边 支持类型: integer, real, date, text
start[s] with begin[s] with	前者是否以后者为开始 支持text和list	end[s] with	前者是否以后者为结束 支持text和list
contain[s]	前者中是否包含后者这一元素 支持text, list和record类型	does not contain doesn’ t contain	前者中是否不含后者这一元素 支持text, list和record类型
is in is contained by	后者中是否包含前者这一元素 支持text, list和record类型	is not in is not contained by isn’ t contained by	后者中是否不含前者这一元素 支持text, list和record类型

比较运算符表达方式非常多样，读者应该根据自己的实际情况选择一种或几种。记得要亲自试一试！

逻辑运算符

and（逻辑与），or（逻辑或）和not（逻辑非）。

三个对参与运算的数据要求均为Boolean型，not为单目运算符。

&运算符

简单来说这个是合并运算符。支持任何类型数据。

记住“&”运算符的三个规则:

- “&”左边的数据类型为Text（文本型）时，结果为Text；存在报错可能
- “&”左边的数据类型为Record（记录型）时，结果为Record；存在报错可能
- “&”左边的数据类型为其他时，结果为List类型

```
"Text" & 1                  --结果：“Text1” (Text类型)
1 & "Text"                  --结果：{1, "Text"} （List类型）

{name:"a"} & "b"            --结果：{name:"a", «class ktxt»:"b"} (Record类型，第二个元素匿名)
3 & {name:"a"}              --结果：{3, "a"} (List类型，且丢失标识符)

{1, 2, 3} & {4, 5, 6}       --结果：{1,2,3,4,5,6} （List类型-一维的）
{1, 2, 3} & 4 & 5 & 6       --结果：同上
{a:1, b:2} & {c:3}          --结果：{a:1, b:2, c:3} （Record类型-一维的）

--错误举例
"Text" & {name:"a"}         --错误！无法将Record类型数据转为文本
{name:"a"} & 3              --错误！无法将Integer转换为Record
```

还是那句话，请亲手试试看“&”，这个运算符非常重要。

第六节 提取对象中的元素

提取字符串中的字母或单词

从简单的开始：提取字符串中全部字母（单个Unicode字符），结果为List类型

```
every character of "一个字符串"        --结果：{"一", "个", "字", "符", "串"}每个字符
characters of "A String"               --结果：{"A", " ", "S", "t", "r", "i", "n", "g"}每个字符
```

请注意：every character of 和 characters of 两个命令的功能完全一致。

类似地，提取字符串中全部单词[10]，结果为List类型

```
words of "A string"                --结果：{"A", "string"}
every word of "我的名字叫张三"     --结果：{"我", "的", "名字", "叫", "张三"}
```

其中：every word of 和 words of 功能也完全一致

下面是进阶一点的：提取指定未知的几个字符或单词，结果为List类型；提取特定位置的一个字符或单词，结果为Text。

```
characters 3 through 5 of "A String"        --结果：{"S", "t", "r"}          (几个字符-List)
words 3 through 5 of "我的名字叫张三"       --结果：{"名字", "叫", "张三"}   (几个单词-List)

word 2 of "This is a text"                  --结果："is"                     (一个单词-Text)
character 3 of "This is a text"             --结果："i"                      (一个字符-Text)

ninth character of "This is a text"         --结果："a"，说明AppleScript还能认识序数词!
```

特别注意：

- 第一个字符的位置是1，而不是0！
- “through”可以缩写为“thru”
- 不推荐用序数词方法获得一个单词或字符
- 不推荐对于中文引用“word”相关语句

提取Finder文件列表

用提取字符串中字母的思想，现在来提取Finder文件列表。直接看代码：

```
tell application "Finder"
    every file of desktop       --获得桌面上所有文件（List类型），其内容很详细
    files of desktop            --同上

    every folder of desktop     --获得桌面上所有文件夹（List类型），其内容很详细
    folders of desktop          --同上
```

---

[10] 对中文竟然也有效，它的断词依据系统词典（个人猜测可能是输入法的词库），看起来断词非常符合中文语言规范。不过建议不要依赖这个功能。

```
    name of every file of desktop    --结果：获得桌面上所有文件名称（List类型）
end tell
```

说明：2-5行代码返回的List中每一项都非常详细，形如“document file "AppleScript.pages" of desktop of application "Finder"”；相反倒数第二行代码仅仅包含Text类型的名称列表。

继续前进！现在提取符合指定条件的Finder文件列表

```
tell application "Finder"
    every file of desktop whose name begins with "a"    --仅获得文件名以a开头的文件列表
    every file of desktop where its name contains "a"    --仅获得文件名包含a的文件列表
end tell
```

这段代码中，使用到了上一节中介绍的比较运算符，代码中的“begins with”“contains”可以用其他比较运算符代替。

新知识：whose和where its：两者功能基本一致，用于限定条件

第七节 添加注释和括号

所谓注释就是写在代码里却不会被执行的文字，通常用于辅助说明代码功能。学过编程的人都听说过一句号“没有注释的代码永远不会是优秀的代码”。对于注释的重要性想必大家都能理解。

行尾注释（End-of -line comment）

以“--”（两个连字符，不含引号）开头[11]；--之后的内容全部为注释，而之前的内容仍然为会被执行的代码，仅对当前一行有效！在前面的代码示例中，我就采用了这种注释方法。

块注释（Block comment）

以“(*”开头，并以“*)”结尾（括号加上星号，均不含引号）；包含在里面的所有文字均视为注释，可以跨行，也可以在一行中间位置。

给代码添加括号

这里只是给大家一个提醒，由于代码执行的优先级问题，可能导致代码未按照预期执行，那么此时加上括号来强制控制运算优先级将很必要。更重要的是，对于复杂的代码的，不管默认优先级如何，总加上括号来明确是个好习惯！

第八节 代码缩写

为了节省时间，将冗长的关键字缩写为几个字母也是非常有用的。此处举几例：

- application 简写为 app（编译时会补全）
- end tell/repeat/try 简写为 end（编译时会补全）
- through 简写为 thru
- if语句 可省略 then（将在下一章节介绍）

[11] 行尾注释还有另一种方法：以“#”开始（井号，不含引号），这种注释在2.0版AppleScript才出现，目的是为了Unix的一些目的，比较复杂，在此不赘述同时也不推荐此方法。

第四章 读懂AppleScript字典

本章节将介绍如何寻找并读懂AppleScript的帮助文件——被称作“字典”。对于任何一种程序或者脚本语言，读懂帮助文件都是非常重要的。

第一节 打开特定应用程序的AppleScript字典

这里介绍两个简单方法来打开可脚本控制的应用程序的AppleScript字典。

打开“AppleScript编辑器”。

- 单击“文件”菜单下的“打开字典...”，将会弹出对话框列出可脚本控制的程序。
- 直接将应用程序（app文件）拖曳到Dock中的“AppleScript编辑器”图标上。

我个人推崇第二种方法。

此外先前已经说过并不是所有程序都是Scriptable（可脚本控制）的，关于是否可脚本控制，请记住一点——只要能够用上面介绍的方法成功打开字典的应用程序都是可以脚本控制的，反之亦然。如果试图打开某应用程序的AppleScript字典时得到如右图所示的错误，则说明该程序不是Scriptable的。

第二节 读懂AppleScript字典

AppleScript字典窗口如右图所示，您可以根据自己的喜好来决定它的样式（默认情况下你的窗口应该没有左侧的索引栏，需要通过拖右侧中间的分割线来调出）。特别强调：一旦你确定喜欢某种字典布局，请点击“窗口”菜单中的“存储为默认”。

关于符号

AppleScript使用主要下面几个图标来区别关键字。

=Suite 套装

=Command 命令

=Class 类（对象）

=Property 属性

理解字典的内容

为了简要起见，这里仅仅介绍最重要的部分——理解命令。

首先来概要下如何阅读：黑体字是关键字，即必须一字不差的照打，如果带有[]则是可以省略的。在黑体字后面、冒号前面的正常体文字是必要要添加的参数类型，冒号后面告诉您参数具体应该是什么。

接着，以make命令为例，逐行分析（灰色底纹的文字为字典中所查询到的文字）：

make *v* : Make a new element
第一行说明词性（v表示动词，和英文字典一样）和具体解释“创建一个新元素”
以下开始就是完整的命令语句所应该（或者可以）含有的内容

make
首先是关键字make，必须先打make（看起来是废话）

new type : the class of the new element
必须要有关键字new，并且后接参数type：所要创建元素的类型。冒号后面是对前面type的具体解释。

at location specifier : the location at which to insert the element
必须要有关键字at，并且后接location specifier类型的参数：元素要创建的位置。

[**to** specifier] : when creating an alias file, the original item to create an alias to or when creating a file viewer window, the target of the window
可选关键字to，如用此关键字必须后接参数specifier。

[**with properties** record] : the initial values for the properties of the element
→ specifier : to the new object(s)
可选关键字with properties，后接Record类型的参数。

最后我们看一下make命令的实战演练

```
tell application "Finder"
	make new folder at desktop
	--最精简的方式，不能比这个还少任何一项
	--其中make为命令，new和at为必备关键字参数，folder属于type，desktop属于location specifier
	--此语句在桌面上建立了一个未命名的文件夹
	make new folder at desktop with properties {name:"AppleScript"}
	--加上了可选参数，是Record类型的。
	--此语句在桌面上建立了一个名称为“AppleScript”的文件夹
end tell
```

再次提醒：务必记得加上“Tell application xxx”和“End tell”

在学会了如何查询AppleScript字典之后，其实很多内容都可以自己探索了！

第五章 变量和属性

本章将更进一步介绍变量和属性，包括它们的生存空间（全局变量、局部变量和预定义变量）、有效性和稳固性，随后将会对变量和属性这两个概念进行区分。本章内容是第三章的延续，因此回顾第三章内容也许是必要的。

第一节 变量的概念

所谓变量（Variable）就是一个拥有名称（标识符，其概念参见第三章第二节）的数据，并且这个数据是可以通过它的名称被引用或者是修改的[12]。使用变量的好处是增强程序的可读性、可拓展性等等。在AppleScript中使用下面语句来声明一个变量。

```
set name to value as type          --as type可省略
```

其中：name为变量名称，其命名必须符合第三章第二节所述的规则；
value为其初始值；
as type（可省）用来强制指定类型，须为第三章第三节所列的类型之一。

此外，很多命令本身带有返回值，因此类似下面这样的语句也是有意义的，并且非常有用。

```
set myResult to the result of (make new folder at desktop)        --需要tell app "finder"
```

其中：myResult为自定义的名称；
the result of语句用于获得命令的结果。请注意并不是所有命令都有返回结果。

第二节 全局变量和局部变量

说到全局变量（Global Variables）和局部变量（Local Variables），不得不再次提到面向对象的一些概念（第三章第一节中有简单介绍）。用最简单的话来说：

- 局部变量：只在一个对象内部有效（可访问和修改），在其他对象里没法访问
- 全局变量：只要在同一个脚本（脚本包）内，任何对象都能访问（和/或修改）的。
- 在不指定全局或局部时，变量默认都为局部变量

通过事件处理器和脚本对象来理解

上面这样的讲法并不严谨，也很难理解。下面将简单介绍AppleScript中的两个新内容——事件处理器（Handler）和脚本对象（Script Object）。

- 事件处理器：相当于其他语言中的方法或者函数。它可以拥有形式参数。
  以“on handlerName”开始，以“end handlerName”结束。
- 脚本对象：一个脚本中的子脚本（或者说是子空间）。
  以“script name”开始，以“end script”结束
  这两个内容将在以后具体讲解，此处仅仅举例。

现在我们可以进一步把局部变量的含义具体化：局部变量的生存空间仅仅在一个事件处理器或脚本内，一旦离开这两个对象，局部变量将无法被访问。对于事件处理器来说，传入的参数（即“形式参数”）是最典型的局部变量。

我们来看这个较为复杂的例子，包含了脚本对象、事件处理器和变量重载。

```
set message to "最先定义的"                    --定义局部变量message

run aNewScript                                --运行脚本对象

displayA("作为形式参数定义的")                  --激活事件处理器displayA，并传递参数

------------------------------
--脚本对象定义开始--
script aNewScript                             --aNewScript是脚本对象的名称
```

[12] 这个说法非常的不严谨，仅仅是为了方便新手理解，请编程高手无视。

```
	set message to "在脚本对象里定义的"	--重定义局部变量message（重载）
	display dialog message	--显示一个包含内容message的对话框
end script
--脚本对象定义结束--

--事件处理器定义开始--
on displayA(message)	--displayA是事件处理器的名称
	--message是形式参数（一个局部变量）
	display dialog message	--显示一个包含内容message的对话框
end displayA
--事件处理器定义结束--
```

试着运行它，将依次显示下面两个对话框:

如果你能够实现预料到运行结果，那么说明你已经体会到了局部变量的“真谛”。如果不行，请再次体会，并试着自己编写代码来试验。

强制全局或者局部

在定义变量之前，增加“Global name”（全局）或者“Local name”（局部）语句可以强制控制变量的生存空间，其中name为变量名称。

第三节 数据共享机制

所谓数据共享机制，就是在复制和修改时——尤其是通过语句“set b to a”（把变量b赋值为变量a的值）方式时，AppleScript对于原数据和新数据内容的处理方式。AppleScript的数据处理方式和其他主流编程语言是基本一致的。

基本数据类型的共享机制（不含Record和List）

对于非Record和List数据，都是采用“拷贝”（pass-by-value，俗称“传值”）。通过实例来说明:

```
set a to 1	--给变量a赋值1
set b to a	--给变量b赋值为a的值
display dialog "赋值的结果：a=" & a & "; b=" & b	--显示a和b的值

set b to 0	--修改变量b的值为0
display dialog "修改变量b之后：a=" & a & "; b=" & b	--再次显示a和b的值
```

运行后，两对话框分别显示:

赋值的结果：a=1; b=1

修改变量b之后：a=1; b=0

结果证明：对于整数型数据，通过“set b to a”方式进行的赋值，b将得到a中数据的拷贝，而不是共享一个数据。以后修改a和b的值，将互不影响。对于除了Record和List之外的基本数据类型，用得都是这样的“拷贝”方式

Record和List的数据共享机制

Record和List型数据，如用“set b to a”语句，将采用“共享数据，只增加引用参考（Reference）”的方式（pass-by-reference，俗称“传引用”或者“传参考”）。

```
set a to {1, 2, 3, 4, 5}	--定义a为List型数据
set b to a	--给变量b赋值为a的值（List型）
```

```
display dialog "赋值的结果：a=" & a & "; b=" & b        --显示a和b的值

set item 1 of b to 0                                    --修改List b中的第一个值为0
display dialog "修改变量b之后：a=" & a & "; b=" & b     --再次显示a和b的值
```

运行后，两对话框分别显示：

赋值的结果：a=12345; b=12345

修改变量b之后：a=02345; b=02345

这次第二个对话框的结果似乎有点出乎意料，的确，对于List（包括Record）通过“set b to a”方式实现的是数据共享，而不是拷贝！实际上a和b指向的是内存中的同一个位置，a和b只是同一内存位置的不同引用！

copy关键字

如果你想要List和Record数据交换方式和其他数据一样，那么您需要将“set”改为“copy”，需要注意，copy和set中的源数据和目标数据的位置是相反的，并且copy的目标数据必须要事先定义。

```
set a to {1, 2, 3, 4, 5}                                --定义a为List型数据
set b to 1                                              --需要提醒的是，使用copy之前必须先定义
copy a to b                                             --给变量b赋值为a的值（List型）
display dialog "赋值的结果：a=" & a & "; b=" & b        --显示a和b的值

set item 1 of b to 0                                    --修改List b中的第一个值为0
display dialog "修改变量b之后：a=" & a & "; b=" & b     --再次显示a和b的值
```

运行后，两对话框分别显示：

赋值的结果：a=12345; b=12345

修改变量b之后：a=12345; b=02345

第四节 属性

对于用电脑的人来说，属性（Property）是一个非常熟悉的名字。比如文件的名称、大小、修改时间、类型都是属性。在第三章中已经提到，属性是用来描述对象的。在AppleScript中你可以自己定义属性，它的用途就好像变量一样（当然它和变量是有区别的）。

定义属性的语句如下：

```
property Label : value
```

其中，Label为属性的标签（相当于变量的名称），value为属性值。

请注意，这个语句不是用来赋值的，而是用来初始化一个属性的！

属性的引用和修改方法和变量是一致的，比如set用来赋值。

属性和变量的区别

属性和变量的根本区别在于其稳固性（Consistence）。所谓稳固性，即脚本退出后其值是否保持不变。属性在脚本退出运行后，仍然记录下它最后的值，并且下一次运行时可以被调出[13]。因此，属性的一个用途就是记录一个脚本运行了多少次，代码如下：

```
property countTimes : 0
set countTimes to countTimes + 1
display dialog "这是第" & countTimes & "次运行本脚本"
```

不断运行这个脚本（不要重新编译），你会得到一个对话框精确地告诉你这是第几次运行。如果将countTimes定义为变量，那么将无法实现功能。

属性的另一个特点是没有全局和局部之分，所有属性都是全局的。

---

[13] 本质上，属性值最终是写入文件，而变量是在内存中的。

第五节 预定义变量

有些变量是预定义的，它包含预设值（常数）或者可以实现特殊功能。一下是几个常用的预定义变量:

- result: 记录最近一个命令执行的结果，如果命令没有结果，那么将会得到错误
- it: 指代最近的一个tell对象
- me: 这指代段脚本。用法举例path to me返回本脚本所在绝对路径
- tab: 用于string，一个制表位
- return: 用于string，一个换行

第六章 流程控制语句

本章节将介绍AppleScript中丰富的流程控制语句。需要事先说明的是，本章节中没有包含有关“异常处理”的控制语句。

第一节 Tell语句

Tell语句是最早接触且最重要的流程控制语句。它的作用是指明脚本要控制的程序对象。Tell语句具有两种语法。

- 简单形式：**tell** referenceToObject **to** statement
- 复合形式：以**tell** referenceToObject开始，**end tell**结尾。中间包含命令语句。

其中：refenrenceToObject为需要控制的对象，如一个应用程序、一个窗口。
statement为命令语句，包含至少一个命令

下面实例将关闭Finder程序中位于最前的窗口，将使用了两种形式的Tell语句，两种形式作用结果相同。

```
tell front window of application "Finder" to close
--这是简单形式

tell application "Finder"
	close front window
end tell
--这是复合形式
```

通常情况下，使用Tell语句的复合形式较为常见，也便于阅读

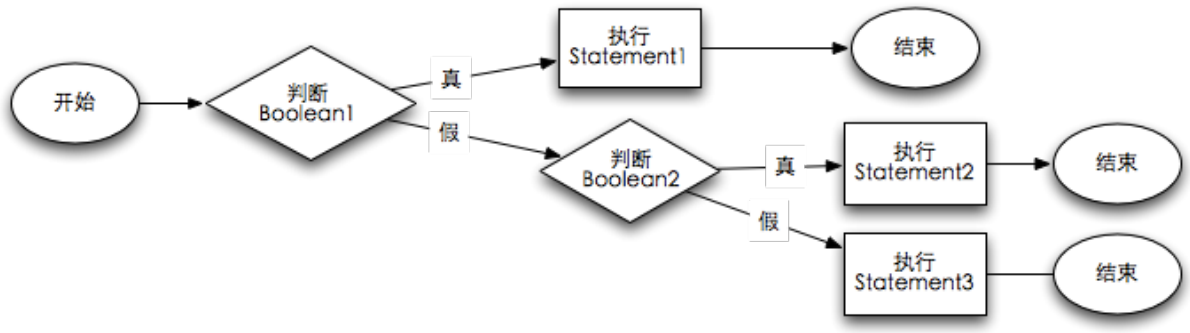
第二节 条件语句If

If语句用于在满足一定条件的情况下执行语句，或者依据条件实现程序流程跳转。

- 简单形式：**if** *boolean* **then** statement
- 复合形式：

```
if boolean1 then
	statement1
else if boolean2 then
	statement2
else
	statement3
end if
```

该符合形式程序流程如下：



其中：boolean为Boolean数据，或者判断语句（返回true或false）
if语句中，可以有多个else if，但至多只能有一个esle（可以没有）。

下面实例将根据变量mark的值，来显示相应的对话框。

```
set mark to 99 --修改这里的数据来观察结果
```

```
if mark ≥ 60 and mark < 80 then
	set response to "You passed the exam"
else if mark ≥ 80 then
	set response to "Congratulations! You're smart"
else
	set response to "Sorry. You failed"
end if

display dialog response
```

第三节 循环语句

在AppleScript中，循环的功能非常强大，提供了至少六种循环方式。在学习循环语句语法之前，有必要先学习如何退出循环！

退出循环的命令

退出循环的命令非常简单，四个字母：exit（完整的为exit repeat）。请牢记

除了exit可以强制退出循环外，return也可以起到退出循环的作用，但是return的实际意义是退出当前的事件处理器，返回脚本流程。

无限循环

这是最简单的循环，但是请慎重使用！语法如下

```
repeat
	--do something
end repeat
```

请务必在无限循环语句中包含exit命令（当然我想exit包含在If语句中才是正确的）

限定次数循环

语法中的n指定了循环多少次。n必须为整数或者整型数据，且大于等于1

```
repeat n times
	--do something
end repeat
```

“直到”循环

```
repeat until boolean
	--do something
end repeat
```

循环将反复执行，直到boolean为真！也就是说，boolean为假的时候循环执行，一旦boolean为真，退出循环。

“当”循环

```
repeat while boolean
	--do something
end repeat
```

在boolean为真时，循环反复执行，一旦boolean为假退出循环。

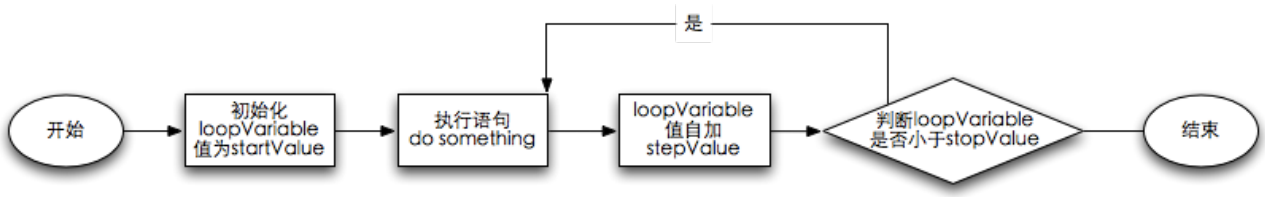
当循环和直到循环在一定意义上是正好相反的。

变量循环

```
repeat with loopVariable from startValue to stopValue by stepValue
	--do something
end repeat
```

其中stepValue可省略，省略时为默认为1；loopVariable无需事先定义。

其执行流程如下图，请注意各个变量的用途



这种循环用于精确控制循环次数，或者是和循环变量本身有关的循环。

*List类型数据循环

这个循环较为复杂，需要读者自己进一步探索，请焦急的读者略过此部分

```
repeat with loopVariable in list
	--do something
end repeat
```

其中loopVariable无需事先定义，list是List型或者Record型数据。

在循环体中，loopVariable将依次得到item 1 of list, item 2 of list....这样的指针（指向list中的第几项）。请特别注意是指针！如果要得到list中的具体内容，使用contents of loopVariable来获得。

在这个循环里，循环体将执行和list项目数量一样的次数。

为了具体说明，请看下面的例子。

```
set myList to {"Hello", "Hi", "Hey", "Goodbye"}	--定义一个List
--循环开始
repeat with i in myList
	display dialog (contents of i)		--显示一个包含List中当前指向项目的内容的对话框
	set (contents of i) to "oh"		--修改List中当前指向项目为"oh"
end repeat
```

试着自己预料循环结束后myList的内容，并输入代码来测试结果。

第四节 Considering/Ignoring语句（用于文本比较）

此语句可以在比较文本时，指定忽略或考虑某一属性（如大小写，空格等等）。

```
considering attribute1 but ignoring attribute2
	--compare texts
end considering
```

上面代码含义是考虑attribute1但忽略attribute2

其中 but ignoring attribute2可以省略;

considering和ignoring位置可以互换，但是end considering也要相应改成end ignoring，当然你可以选择最简略的方式——只输入end，让编译器自己补上considering/ignoring。

attribute应该为下面列表中的任意一个:

case	大小写
diacriticals	字母变调符号（如e和 é ）
hyphens	连字符（-）
numeric strings	数字化字符串（默认是忽略的），用于比较版本号时启用它[14]。
punctuation	标点符号（,.?!等等，包括中文标点）
white space	空格

[14] 这一属性的具体例子是：”1.10.1”>”1.9.4”在默认情况下（忽略此属性）结果是false，而启用(considering numeric strings)时结果为true。所以说这个属性用于比较软件版本号时非常有用。

第七章 基本用户交互

本章将介绍AppleScript中基本的用户交互——主要是对话框。在开始具体内容之前，有必要说明一点：常规的AppleScript脚本并不包含用户界面（UI），因此本章介绍的用户交互其实都是简单的对话框或者系统面板[15]。AppleScript自带的用户交互功能全部包括于AppleScript字典中Standard Addition下User Interaction类。本章节不会对每个命令的具体参数进行全面讲解，所以还请读者自己打开AppleScript字典进行细致研究。

第一节 简单对话框和输入框

Display Dialog命令

对于这个基本的对话框，在前面的章节中已经有所提及。它最简单的格式是：

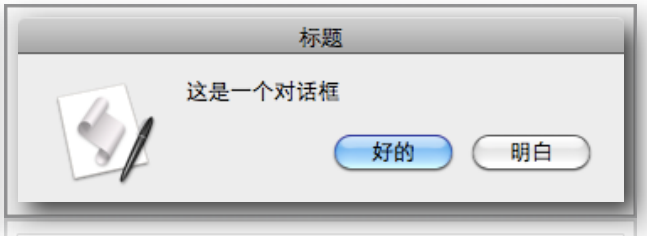
```
display dialog "对话框包含的内容"
```

此命令将显示一个指定文字的、拥有确认和取消按钮的对话框，并且确认按钮是默认按钮——响应按下回车键（Enter/Return键）。

接下来看看一个较为完整的对话框：

```
display dialog "这是一个对话框" buttons {"好的", "明白"} default button "好的" with title "标题"
    with icon note giving up after 5
```

显示对话框效果如右下图，可以和左下图（最简形式做一个比较）：



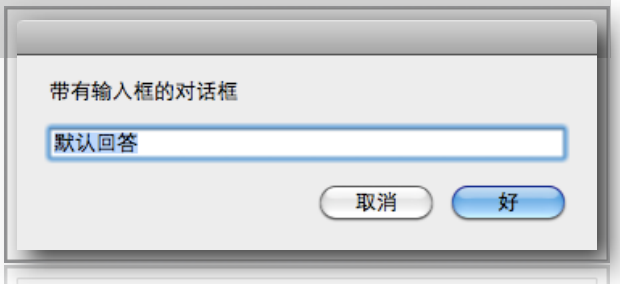
具体的参数介绍如下：

- buttons　紧跟List型参数，指定对话框拥有的按钮名称，注意最多为三个
- default button　紧跟text型参数的某一个按钮名称，设定默认按钮
- with title　紧跟text，指定对话框的标题（省略时无标题）
- with icon　紧跟stop/note/caution中的一个或者file类型的路径，指定显示的图标
- giving up after　紧跟number型的整数，指定在number秒后自动消失对话框。

带有输入框的对话框

要想实现输入功能，同样可以用Display Dialog命令，效果如右下图：

```
display dialog "带有输入框的对话框" default
    answer "默认回答"
```



只需要添加default answer (+text)就可使普通对话框升级为输入框。其中text可以用空文本（直接键入两个引号）；添加hidden answer true命令，可以隐藏输入文本（输入密码时用）。此外，前面介绍的关于简单对话框的几个参数仍然可用。

对话框的返回值

为了确定用户输入的文本和按下的按钮，我们需要通过对话框的返回值来实现。

display dialog命令的返回值总是一个Record，通常包含下面几项的一个或多个：

---

[15] AppleScript Studio完整支持用户界面设计，它需要通过Xcode开发工具进行开发。

- text returned　　用户输入的文本
- button returned　　用户按下的按钮的名称（即显示的名称）
- gave up　　是否自动超时消失（和giving up after命令相应）

第二节 警告对话框

警告对话框一定程度上非常接近简单对话框，来看一下它的特色语法:

```
display alert "这是一个警告" message "警告的信息" as warning
```

结果如左下图。

其中:　　message参数指定了补充信息（在对话框中以小字显示），as warning/critical/informational指定了对话框的重要性（表面上看起来就是图标不同）。此外可用简单对话框中的buttons（指定按钮）, give up after（自动超时消失）参数。

第三节 列表选择对话框

这是一个特殊对话框，提供了一个列表供用户选择。语法如下:　（结果如上右图）

```
choose from list {"备选一", "备选二", "备选三"} with title "这是一个列表选择框" with prompt "请做出选择！" default items {"备选二"} with empty selection allowed and multiple selections allowed
```

参数说明:

- 直接参数　　紧跟List类型参数，包含所有备选项
- title　　紧跟text，指定选择框的标题
- prompt　　紧跟text，指定提示信息
- default items　　紧跟List，指定默认选择的项目
- empty selection allowed　后紧跟true表示允许不选
- multiple selections allowed　　后紧跟true表示允许多选

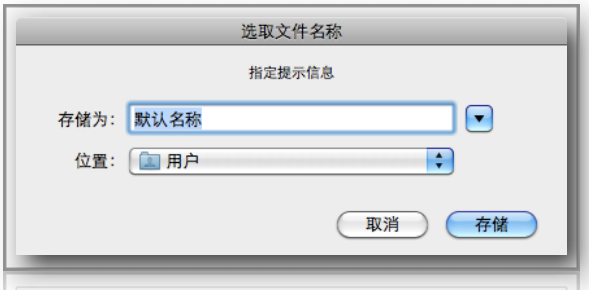
第四节 文件选择对话框

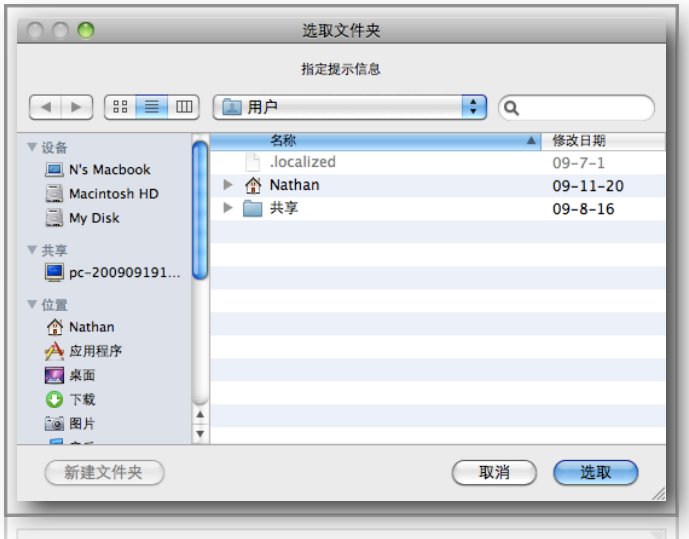
选取文件名称Choose File Name

这个对话框很类似于一般软件的另存为对话框。要求用户指定一个将来用于保存信息的文件，请注意，Choose file name命令并不会创建文件，它的返回值是file类型的，包含完整路径。完整语法如下

```
choose file name with prompt "指定提示信息" default name "默认名称" default location file "Macintosh HD:Users"
```

其中prompt指定提示信息，default name指定默认名称，default location指定默认存储位置，需要file类型的参数，三个参数均可以省略。上面这段代码执行结果如下左图





选取文件夹Choose Folder

选取文件夹对话框，效果如上右图，完整语法如下：

```
choose folder with prompt "指定提示信息" default location file "Macintosh HD:Users" with
    invisibles, multiple selections allowed and showing package contents
```

其中prompt和default location参数同Choose File Name；另外invisibles指定显示隐藏文件，multiple selections allowed可以多选，showing package contents显示包内容，省略时则不显示隐藏文件/不可多选/不显示包内容。

choose folder命令返回值为alias或者是List（由alias构成，当允许多选时）

选取文件Choose File

选取文件对话框几乎和选取文件夹对话框一样，拥有几乎相同的参数、语法和返回值类型，只是多了一个of type可选参数（后加List，list中应包含一个或多个text，指定允许选择的文件类型）

```
choose file of type {"txt"}
```

注意：Choose Floder所有的参数，Choose File命令也可用，这里不再列举。

第五节 其他用户交互

除了上述所说的几种常用对话框外，AppleScript还提供了Choose color用于选择颜色，返回包含RGB信息的List，这个对话框（更准确的说是面板）和我们在其他软件，如Pages中的颜色选择面板是完全一样的。另外还有一些不常用的对话框，有兴趣的读者可自己阅读AppleScript字典。

关于用户交互，还有几个特殊的命令：beep,delay,say。这三个命令用起来都非常简单：

beep后接一个整数，让机器蜂鸣n下，如果缺省整数，将会发出一声。
delay后接一个整数，让代码暂停n秒，如果缺省整数，将会没有意义。
say后接一个text，让电脑读给你听这个单词或句子。注意，Mac OS只会念英文。

第八章 错误处理

错误处理（很多计算机语言中叫做“异常处理”即Exception Handling，但在AppleScript中却叫做“Error Handling”）是所有脚本或程序设计所要面对的重要课题。在脚本运行时，时常会遇到意外的错误，如文件不存在。本章节将讲解AppleScript中有关错误处理、错误排除的内容。涉及的命令仍然是Standard Addition类中的。

第一节 基本的Try语句

首先，先要了解一个原则：就是脚本执行过程中如果遇到错误[16]时，将立即退出脚本运行，之后的代码将得不到执行。

基本try语句的便是告诉Mac，如果遭遇错误就忽略掉，不执行这条代码，转而执行try语句之后的代码。

```
try
	--statement
end try
```

举一个简单的例子来说明try的用途

```
tell application "Finder"
	make new folder at desktop with properties {name:"AppleScript"}
end tell
display dialog "Success！"
```

执行上面代码第一次将在桌面上建立名为“AppleScript”的文件夹，并会看到显示有“Success！”的对话框，但是如果你继续执行第二次便会得到错误，并且不会看到对话框（原因是已经存在了同名文件夹）。现在将前三行代码（tell块），用try-end try包裹起来，再执行它，你会发现无论执行多少次这个脚本，都能看到Success对话框。

第二节 带有错误处理的Try语句

所谓带有错误处理，即我们不再像前一节那样直接忽略掉引起错误的代码，而是要对发生的错误进行负责任地处理。

```
try
	--do something at risk 做一些可能导致错误的事情
on error errText number errNum
	-- do something to handle error 做一些处理错误的事情
end try
```

比起基本Try语句，多了on error那一行（注意 on error在这里是必须的，errText number errNum是可省略的）。有关解释：errText是自定义的变量名称，它将包含遇到错误的信息，是text格式的；number为关键字，后跟一个变量名（例中为errNum），这个变量将包含错误号码。

习惯上，在on error和end try之间，一般都会包含一个if语句，来检查错误类型，并对不同的错误进行不一样的处理。在第三节中将举一个综合例子。

第三节 AppleScript中的错误（Error）

抛出一个自定义错误

```
error "自定义一个错误" number 999
```

[16] 指没有“捕获”的错误，即没有放在try语句内。

没错，上面这行代码非常脑残！因为它什么有意义的事也没做，只是告诉AppleScript运行器“我遇到错误了！”[17]。error后面紧跟一个text用于说明错误内容，number关键字后跟一个数字指定错误代码——注意通常不要让这个数字和系统预定义的错误代码相同。

系统预定义的错误

在AppleScript中，预定义了很多错误：如error number −128是“User Cancelled.”（用户取消）；error number −43是“File <name> wasn’ t found.”（未找到xx文件）。想要了解更多的错误，请参阅附录。

错误处理实例

```
try
	display dialog "您确定要在桌面上创建名为“AppleScript Folder”的文件夹嘛？"
	tell application "Finder"
		make new folder at desktop with properties {name:"AppleScript Folder"}
	end tell
on error eText number eNum
	if (eNum = -48) then
		display dialog "发生文件错误，错误内容为：\n" & eText
	else if (eNum = -128) then
		display dialog "您按下了取消按钮，错误内容为：\n" & eText
	end if
end try
```

例子仍然非常简单，请先预测各种脚本执行情况，然后自己执行它看看结果。

第四节 超时（Timeout）

AppleScript经常要于用户或者其他引用程序交互，当AppleScript发送出了一条命令后，它将等待接受用户或者应用程序的响应。默认情况下AppleScript将等待120秒，如果在120秒内没有得到响应，将会抛出 "AppleEvent 已超时。" number −1712错误。不过，120秒不是适合所有情况的，如果需要自己确定一个等待的时间，那么将要用到TimeOut语句,其语法如下:

```
with timeout of x seconds		--x必须为整数
	--wait for something
end timeout
```

上面的语句告诉脚本，在wait for something中最多只等待x秒，如果x秒过后，仍然没有得到预期响应就会抛出超时错误。需要说明的是：x是可以大于120的！这样就突破了AppleScript默认的“耐心”，有时候也是有用的。

下面以举例说明timeout语句具体用法:

```
with timeout of 3 seconds
	display dialog "我最多等你三秒"
end timeout
display dialog "你的确在三秒内做出了响应"
```

如果你在三秒内的确按下了对话框中的按钮，那么脚本随后跳出下一个对话框。如果你没有在三秒内按下对话框中任意按钮，脚本抛出错误。请自己尝试一下。

---

[17] 这句话并不是说error是没用的，只是说明如果不用在合适的地方，那error命令将是累赘。在进阶的编程中这个命令非常有用，很多错误处理都需要它。

第九章 文件操作

之前已经介绍过利用Finder程序来实现文件操作，而本章节介绍的文件操作将是AppleScript原生的，也就是不再需要Tell application “Finder”了。

第一节 Alias类型

如果你对Mac OS X很熟悉，那么你不会对Alias（替身）感到陌生：Alias就是文件的一个指针，它指向的某个实际的文件，但Alias本身不是文件（因为它只记录文件的ID），请特别注意它不同于windows的快捷方式——记录文件的路径！Mac OS的Alias是记录文件的唯一识别码[18]，即使文件被移动了，Alias替身仍然可以准确指向本来所指的文件。

请注意：AppleScript也提供了传统的file类型，但是在一般情况下，我们总是使用alias而不是file来实现文件操作——file类型多用于操作尚不存在的文件。

在AppleScript中创建Alias类型的数据是非常容易的：只需要在alias关键词后加上路径——text类型的，以冒号为分隔符的完整路径[19]，举例如下：

```
set myAlias1 to alias "Macintosh HD:System:Library:CoreServices:Finder.app:"
set myAlias2 to alias "Macintosh HD:Users:Nathan:Desktop:example.txt"
```

第二节 相对路径和POSIX路径

当你编写的一段脚本需要放到其他电脑上运行时，往往会因为系统盘名称或者是用户名的不同而让你不知所措——AppleScript无法找到你想要的文件。绝对路径的缺点就在于此。

相对路径 path to命令

在Mac OS X中，有很多文件夹具有特殊地位，如用户的文档文件夹（Documents）、系统的资源库（Library）、应用程序文件夹（Application）等等，这些文件夹经常会被使用到，而用绝对路径来表达它们的位置显然会产生可移植性问题。AppleScript中提供的path to命令就是用来解决这个问题的：请看下面的示例代码。

```
path to documents folder --返回当前用户的“文档”文件夹绝对路径alias
path to library folder from system domain --返回系统的“资源库”绝对路径alias
```

可以使用path to命令来获得的文件夹非常多，如application support、applications folder、desktop、documents folder、downloads folder、system folder等等（请参阅AppleScript字典）。from指定了文件夹所属的域，之所以要指定域是因为类似资源库（Library）文件夹在系统中存在不只一个（用户的、系统的、本地的等）。常用user domain（默认缺省值）、system domain和local domain。

POSIX路径和POSIX file类型

POSIX路径全称Portable Operating System Interface of Unix，是IEEE的标准之一。POSIX路径（path）只是POSIX中一个很小的部分，该路径用斜杠（/）分隔层次——注意不是windows的反斜杠（\）。它具有简明性和相对性的特点，也可以用来避免绝对路径的致命缺陷。AppleScript提供了POSIX路径的支持：

```
POSIX path of alias "Macintosh HD:System:Library:CoreServices:Finder.app:"
--返回"/System/Library/CoreServices/Finder.app/"
POSIX file "/Users/Nathan/Desktop/example.txt"
--返回file "Macintosh HD:Users:Nathan:Desktop:example.txt"
```

---

[18] 请不要深究Alias到底记录了什么，唯一识别码这个说法并不精确，您只需要记住：Mac OS的替身不是记录路径的！和windows的快捷方式不一样！

[19] Mac系统中，常用的文件路径有两种，一种是完整路径，即以冒号分隔的；另一种是POSIX路径，以斜杠（/）分隔。AppleScript默认使用前者

POSIX path of 命令用于获得POSIX格式的路径。
POSIX file是一个类型，和alias相似，只是它是用POSIX的方式来操作文件。

POSIX路径是一个比较麻烦的问题，实际编写脚本时尽量使用alias结合path to命令来实现功能，尤其要避免POSIX file和alias混用的情况。

第三节 文件读取

AppleScript的文件读取简单到难以想象，只要一个命令：read就行了。[20]

```
set myFile to alias "Macintosh HD:Users:Nathan:Desktop:example.txt"
read myFile
```

这个最简单的格式将会返回包含文件中所有内容的text类型数据[21]
只需要加上几个限定，我们就可以让read命令变得更听话些：

from 整数	指定从哪个位置开始读取（位置指字节数）
for 整数	指定读取多少个字节
to 整数	指定读取到哪个位置为止
before 文本	指定读取到文本所指定的关键字为止（不含本身）
until 文本	同上，但含本身
using delimiter 文本	指定分隔符读取成list类型的数据，这里参数也可是由文本组成的list
as 类型	指定读取成何种数据类型，如text,list

在这里有必要提一下文件结尾（end of file），如果文件是空的，在尝试读取时，会让AppleScript抛出文件结尾错误，因此，在读取文件之前，应该养成先确定文件长度的习惯，AppleScript中使用get eof命令（后直接跟alias类型的参数）。

第四节 文件写入

文件写入相比读取总要麻烦许多，AppleScript尽管已经为用户尽量减少了很多工作，但是仍然需要我们记住一下基本的读取流程：即打开文件，写入数据，关闭文件。

```
set aFile to alias "Macintosh HD:Users:Nathan:Desktop:example.txt"
set fp to open for access aFile with write permission --打开文件
write "abc" to fp --写入数据
close access fp --关闭文件
```

简单来说，要写入一个文件，首先需要open for access with write permission，然后使用write to命令，最后还要close access。此外，write命令可以指定staring at参数（开始写入的位置）。

关于文件的打开和关闭：在上一节介绍文件读取时，并没有提到打开和关闭，但事实上我们应该在读取之前先打开文件（open for access），并在读取完毕之后正确关闭它（close access）。如果你读取文件之后没有关闭它，当你之后需要写入数据时，会发现根本没法打开写入权限，AppleScript会告知文件已经打开却没有给你写入的权限！总之一句话：对于任何文件操作，必须要先打开再操作，最后要关闭！

---

[20] 事实上，文件读取和写入永远不会像说得那样简单，在下一节就将介绍常规文件读写流程。

[21] 在指定读取长度时请特别小心，包括中文在内的很多语言可能会因为不合适的指定长度导致不和预期的返回结果，这和文字编码有关。

第十章 事件处理器

事件处理器（Handler）是AppleScript的专有名词，可以理解为其他程序语言中的函数（Function，如C语言）、方法（Method，如Java）或者是子程序（Subroutines）。简单来说，事件处理器在AppleScript中的价值就是减少大段类似代码，提高代码可复用性。然后深入学习之后，你可以借用事件处理器来编写超酷的“Drag & Drop App”以及“文件夹脚本”

第一节 基本的事件处理器

事件处理器是一段代码的集合，通过呼叫它的名称（即“调用”）来执行其中的具体代码。事件处理器如果没有被调用，那么其中的代码将永远不可能被执行。最基本的事件处理器定义和调用过程如下:

```
on HelloWorld()                          --定义开始
      display dialog "Hello, world"      --事件处理器中具体要执行的代码
end HelloWorld                           --定义结束

HelloWorld()                             --调用
```

其中HelloWorld是这个事件处理器的名称，请特别留心名称后面的括号，正是这对括号才把变量名称和事件处理器名称区分开来。

又一个非常重要的问题需要强调，任何事件处理器无法在Tell语句块内直接被调用，如果确实需要调用，应该使用 HelloWorld() of me。请记住“of me”。

第二节 带参数的事件处理器

所谓事件处理器，应当是智能的——它应可以根据不同的场景根据同样的规则作出不同的响应。我们可以通过“参数”把场景告诉事件处理器。AppleScript提供了两种参数格式，位置参数（Positional Parameters）和标签参数（Labeled Parameters）。前者是和其他语言类似的常规参数格式，而后者是为了体现AppleScript贴近生活语言而设计的。不过，如果你没有处理好，标签参数将会非常的悲剧，因此个人建议少使用标签参数，即使它看起来非常棒。

位置参数根据参数出现的前后顺序来区分。要使用位置参数，应当将变量名称（不应和现有名称重复，应该取一个新名称，仅供事件处理器内部使用，体现“局部变量”概念）置于括号内。至于参数个数没有限制，多个参数以逗号分隔。如:

```
on Hello(somebody, howLong)                                --定义开始
      display dialog somebody giving up after howLong      --该对话框将在指定时间后自动取消
end Hello                                                  --定义结束

Hello("Apple", 2)                                          --调用，必须指定两个参数
Hello("Microsoft", 1)
Hello("User", 3)
```

上面的代码以somebody和t作为参数传递给名为“Hello”的事件处理器。对于有参数的时间处理器，调用时必须给出设定数量参数。值得一提的是，不同于严谨的高级语言，AS中的参数不需要指出参数类型，直接写变量名称即可。

标签参数对于英语好的同学非常有帮助，因为它让代码看起来就象是日常对话。此外使用标签参数可以不按照参数本来的顺序，即使颠倒顺序，AS也能正确运行。实现与上面位置参数示例相同功能，标签参数如下:

```
on Hello to somebody for howLong                           --定义开始
      display dialog somebody giving up after howLong
end Hello                                                  --定义结束

Hello to "Apple" for 2                                     --调用
```

```
Hello for 1 to "Microsoft"                                    --颠倒顺序也可以正确执行
Hello to "User" for 3
```

很美的代码，尽管很让人头疼。AppleScript允许使用很多介词来“有意义”地区分各个参数。包括about, above, against, apart from, around, aside from, at, below, beneath, beside, between, by, for, from, given, instead of, into, on, onto, out of, over, since, thru (or through), under等。但是，每个介词只能出现一次。即使这段代码如此诱人如此亲切，但是我还是强调踏踏实实用位置参数，不要炫耀英语水平。

第三节 返回值

事件处理器不仅可以处理，也可以告诉你它作了什么。和其他编程语言类似，AS也使用return关键字。

```
on add(x, y)
	set answer to (x + y)
	return answer
end add

display dialog add(1, 2) --获取add处理器的返回值
```

不仅可以return一个变量（代码：return aVarName）也可单独用return来结束事件处理器运行。

第四节 run和open事件处理器

AS为脚本应用程序（Script Application）提供了一些特殊的事件处理器。包括run, open, quit和idle。使用起来和自定义的事件处理器一样，“on run”和“end run”。请注意，如果你直接运行脚本它们通常不会正常工作，只有你将脚本保存为“应用程序”后它们才能够正确运行。

run

run是一个默认的事件处理器，它是脚本运行的入口，任何不在其他事件处理器里的代码都隶属于run事件处理器。通常我们不需要、也没有必要显式声明run事件处理器。请记住，如果你显式声明了run，那么不能有任何代码位于事件处理器外，即所有代码必须位于你的事件处理器中或者是run事件处理器中。

open

open对于AS开发者来说是非常有用的，它可以实现超酷的“Drag & Drop App”（使用拖放实现某些功能的程序），它仅仅对于保存为“应用程序”的脚本有效。

open其实是一个Mac OS的系统命令，一旦用户在Finder中把文件或文件夹拖放到某程序或脚本的图标上，它就立刻收到open这个系统命令，如果你的AS脚本中包含了open事件处理器，那么它将会处理这个情况。

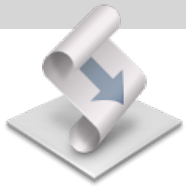
open事件处理器可以、通常也一定会添加参数（标签参数），如“on open aFile”，aFile是参数，告诉脚本什么东西被拖放到了它的图表上。

下面示例代码[22]的功能是：如果用户将文件或者文件夹拖放到脚本程序图标上，脚本会打开“文本编辑”（TextEdit）软件，并列出所拖放的文件的完整路径列表。

```
on open names
	set pathNamesString to "" -- 首先设定一个空的文本

	repeat with i in names
		-- 在这个循环中，处理了每个被拖放到程序图标上的文件
		-- 所做的工作就是记录下文件路径并加上回车，添加到本来的文本中
		set iPath to (i as text)
```

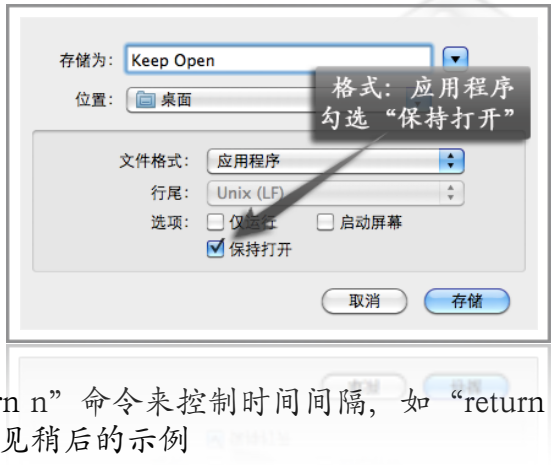
[22] 本段代码摘录自苹果官方的《AppleScript Language Guide》，作者对注释部分进行了翻译

```
        set pathNamesString to pathNamesString & iPath & return
    end repeat

    -- 将记录下来的文件路径列表写入到文件
    tell application "TextEdit"
        set paragraph 1 of front document to pathNamesString
    end tell

    return
end open
```



再次强调，为了让脚本按照预想地工作，必须将它保存为“应用程序”，然后把文件拖放到该图标上。留意一下它的图标，看起来和一般的脚本没什么区别，但是它有个小箭头！显示它是一个很牛的“Drag & Drop App”

第五节 保持打开的脚本应用程序



要实现本节所讨论的功能，请确保你将您的脚本存储为应用程序，并勾选“保持打开”，如右图。为实现脚本“保持打开”，我们经常用到两个预设事件处理器：idle和quit。

idle

这个事件处理器用于处理应用程序空闲时的后台任务，它具有循环的功能，只要不退出或者有其他任务，它就会隔一段时间自动执行其中的代码一次。其默认时间间隔是30秒，我们可以通过“return n”命令来控制时间间隔，如“return 10”，告诉AS每隔10秒执行一次Idle中的任务，请参见稍后的示例

quit

它用于处理用户手动退出保持运行的程序时要执行的任务。其中，必须包含有“continue quit”命令，否则程序将不可能正常退出。也就是说，即使触发了quit事件处理器，该保持打开的应用程序也不一定会退出。应用程序将在执行“continue quit”命令时退出。

关于保持打开的脚本应用程序，示例如下：

```
on idle
    beep 2                                          --蜂鸣两声
    display dialog "程序正在运行" giving up after 1
    return 5                                        --设定每隔5秒执行一次
end idle

on quit
    display dialog "真的要退出? " buttons {"是的", "不"}    --询问是否退出
    if button returned of result = "是的" then
        continue quit                               --只有这个命令才会让它真正退出
    end if
end quit
```

再次提示，如果直接从AppleScript编辑器中运行将看不到结果，请保存后双击该程序运行。

第六节 文件夹操作

您是否曾经注意到Finder右键菜单中的“文件夹操作”（Folder Action）命令，它很酷很强大。当文件或文件夹发生更改时，“文件夹操作”就会被激活，这里的的“文件夹操

作”其实就是AS脚本，它们存储在“资源库\Scripts\Folder Action Scripts\”（或者用户资源库下相应的）文件夹中。你需要在相应文件夹的右键菜单中激活文件夹操作。

关于编写自己的文件夹操作脚本，我们可以在AppleScript字典Standard Addition-Fold Actions中找到，里面有5个命令，通过事件处理器的方式我们可以编辑对应命令。这五个命令都有很多参数可选，请阅读AppleScript字典。示例:

```
on adding folder items to theFolder after receiving theItemList
	--发现文件添加文件就弹出对话框，显示有几个文件添加了
	display dialog ((count theItemList) & "个项目被添加到了该文件夹)
		as text buttons {"好", "立即查看"}
	--选择立即查看则打开文件夹
	if button returned of result = "立即查看" then
		tell application "Finder"
			open theFolder
		end tell
	end if

end adding folder items to
```

要使用这个脚本，首先需将脚本保存到上文所述的指定位置，然后右击想要执行操作的文件夹，选择“文件夹操作设置”，按照屏幕提示选择你所编写的脚本。

文件夹操作的缺点是运行速度较慢，好处是不影响系统使用，你可以编写一个脚本来监视你的重要文件夹，将各种操作写入到文件，而用户（包括你自己）全然不知有个脚本正在记录你的一举一动。

第十一章 脚本对象

在本教程开端，就提出AS是一个OO（object-oriented）语言。面向对象的特性让代码更加地简洁易读，本章将介绍除了事件处理器之外的另一个重要OO部分——脚本对象，相当于其他语言的的“类”（Class）。

第一节 me关键字

me关键字在AS中指代当前脚本。两个常见用途是:

path to me　返回当前脚本的路径，如果在编写脚本时运行，返回结果将是AppleScript编辑器的路径;

class of me　返回“我”的类型，毫无疑问一般情况下就是script（脚本）

实践中“path to me”非常有用，尤其在需要调用其他脚本文件时。

第二节 编写和使用基本的script对象

使用script对象的目的是将同样功能代码归类（专业术语为“封装”）。你可以通过script和end script关键字来显式声明一个script。示例如下:

```
property dialogText：“ShowMe脚本中的代码”

script ShowMe
	display dialog dialogText

	on showSomething()　　--在脚本对象内部建立一个事件处理器
		display dialog "这是脚本对象中的事件处理器"
	end showSomething
end script
```

如果你尝试运行它，会发现没有任何结果！没错，脚本对象本身就像一个工人，它不会自说自话地去完成任何工作，除非老板明确指示他要做。但是通过脚本对象内部的代码，这个工人能够做些什么事情已经被确定了。脚本对象就是建立了这样一个“工人”。

接下来我们的任务自然是学习如何“指示这个工人去工作”。要执行脚本对象中的代码，应当使用run命令:（请注意，本节中以下代码和前一个示例的代码存在于同一文件）

```
run ShowMe
```

注意:正如普通的事件处理器，位于脚本对象中的事件处理器，在没有得到明确指令时是不会执行其中代码的。使用工人的例子来说就是，工人也各种能力，老板让他用这个能力，他一定不会去用。

接下来的代码是如何运行位于脚本对象中的事件处理器。

```
showSomething() of ShowMe
```

此代码将仅仅执行ShowMe对象中的showSomething()事件处理器内的代码。

这里推荐另一种调用脚本对象的方法，即tell语句，适用于调用任何类型的脚本对象内部代码，这个方法提高了程序易读性和整洁性。

```
tell ShowMe
	run --执行脚本对象中的代码，但不包括事件处理器的代码
	showSomething() --执行对应事件处理器内的代码，不执行其他代码
end tell
```

第三节 载入和调用外部script对象

要载入并运行来自其他脚本文件中的脚本对象，首先要求存在一个脚本文件，然后我们应该获得它的路径位置，其次载入它，最后运行它。

下面示例要求我们先将第二节中的第一段代码保存到桌面，并命名“Script to Load”，然后新建一个脚本文件，输入如下代码:

```
set thePath to ((path to desktop) as text) & "Script to Load.scpt"
```

```
--获得目标脚本对象的位置
set theScript to load script file thePath
--载入该脚本，记得一定要给它“取名”

run ShowMe of theScript
--运行脚本对象
showSomething() of ShowMe of theScript
--运行脚本对象中的事件处理器
```

你会发现用了好多个of，确实，为了让傻傻的AppleScript编译器知道我在跟谁说话，必须一次次明确告诉他我想和谁说话。当然你会想到又一个简单明了的方法：那就是用tell语句模块。

特别说明：其实载入外部脚本，并不要求该外部脚本拥有script对象，任何脚本都是可以被加载的。开发者通常把一堆事件处理器放在一个文件中，这样就建立了一个“库”。

第四节 修改外部script对象中的属性变量

AS不仅允许载入外部script对象，还允许修改外部script对象中的属性。本节示例的要求和第三节一样，要求合适地保存第二节中第一段代码。

```
set thePath to ((path to desktop) as text) & "Script to Load.scpt"
set theScript to load script file thePath

run ShowMe of theScript                                    --初次运行

set dialogText of theScript to "哈哈，我修改了你！"
run ShowMe of theScript                                    --修改之后再运行
```

两个同样的run ShowMe of theScript显示了不同的对话框，证明外部脚本中的属性可以被修改。不过当你再次运行脚本时，发现第三行的“run ShowMe”结果仍然是本来外部脚本文件中所设定的。没错，这里“set dialogText”命令只是传递给了内存中的外部脚本，并没有写入脚本文件。所以这里的set仅仅是“一次性”的，它不会影响外部脚本本身。

当然，如果你确实真需要修改外部脚本中的属性，也是可以的，AS提供了“store script”命令。在上面示例中添加一行新代码：

```
store script theScript in file thePath with replacing
```

那么第二次运行当前脚本时，第一个“showMe”仍然是“哈哈，我修改了你！”。需要指出的是，with replacing属性告诉store script命令“别怕，放心地去替换”，如果省略，AS将会询问是否确认替换，with replacing本质上是一个危险的操作，因为你可以随意修改外部脚本。

附录一：AppleScript保留关键字

about	contain	front	on	tenth
above	contains	get	onto	that
after	contains	given	or	the
against	continue	global	out of	then
and	copy	if	over	third
apart from	div	ignoring	prop	through
around	does	in	property	thru
as	eighth	instead of	put	timeout
aside from	else	into	ref	times
at	end	is	reference	to
back	equal	it	repeat	transaction
before	equals	its	return	true
beginning	error	last	returning	try
behind	every	local	script	until
below	exit	me	second	where
beneath	false	middle	set	while
beside	fifth	mod	seventh	whose
between	first	my	since	with
but	for	ninth	sixth	without
by	fourth	not	some	
considering	from	of	tell	

附录二：预定义的错误代码和错误信息

AppleScript错误：

错误代码	错误信息
-2700	Unknown error.
-2701	Can’ t divide <number> by zero.
-2702	The result of a numeric operation was too large.
-2703	<reference> can’t be launched because it is not an application.
-2704	<reference> isn’t scriptable.
-2705	The application has a corrupted dictionary.
-2706	Stack overflow.
-2707	Internal table overflow.
-2708	Attempt to create a value larger than the allowable size.
-2709	Can’t get the event dictionary.
-2720	Can’t both consider and ignore <attribute>.
-2721	Can’t perform operation on text longer than 32K bytes.
-2729	Message size too large for the 7.0 Finder.
-2740	A <language element> can’t go after this <language element>.
-2741	Expected <language element> but found <language element>.
-2750	The <name> parameter is specified more than once.
-2751	The <name> property is specified more than once.
-2752	The <name> handler is specified more than once.

-2753	The variable <name> is not defined.
-2754	Can't declare <name> as both a local and global variable.
-2755	Exit statement was not in a repeat loop.
-2760	Tell statements are nested too deeply.
-2761	<name> is illegal as a formal parameter.
-2762	<name> is not a parameter name for the event <event>.
-2763	No result was returned for some argument of this expression.

Mac OS系统错误

错误代码	错误信息
0	No error.
-34	Disk <name> full.
-35	Disk <name> wasn’ t found.
-37	Bad name for file
-38	File <name> wasn’ t open.
-39	End of file error.
-42	Too many files open.
-43	File <name> wasn’ t found.
-44	Disk <name> is write protected.
-45	File <name> is locked.
-46	Disk <name> is locked.
-47	File <name> is busy.
-48	Duplicate file name.
-49	File <name> is already open.
-50	Parameter error.
-51	File reference number error.
-61	File not open with write permission.
-108	Out of memory.
-120	Folder <name> wasn’ t found.
-124	Disk <name> is disconnected.
-128	User cancelled.
-192	A resource wasn’ t found.
-600	Application isn’ t running
-601	Not enough room to launch application with special requirements.
-602	Application is not 32-bit clean.
-605	More memory needed than is specified in the size resource.
-606	Application is background-only.
-607	Buffer is too small.
-608	No outstanding high-level event.
-609	Connection is invalid.
-904	Not enough system memory to connect to remote application.
-905	Remote access is not allowed.
-906	<name> isn’ t running or program linking isn’ t enabled.
-915	Can’ t find remote machine.
-30720	Invalid date and time <date string>.

后记

编写这本简明教程的过程中，我深深感到让所有读者都能迅速上手AppleScript是非常困难的。学会脚本语言，说到底就是学会如何阅读帮助文件（就像Java的Documentation），那就是AppleScript Dictionary（字典）。因此看似非常简单实则保罗万象的第四章也许是最重要的。当你充分学会了如何阅读AppleScript字典，那么你就可以从AS中获得更多便利，比如说批量图片处理（利用Image Events）。和学习任何其他程序语言一样，没有人可以教会你如何去编写你想要的程序。一切教程的目的就是让你学会如何通过技术文档去学习一种编程语言。