JavaScript Application Cookbook

JavaScript 应用程序经典实例

O'REILLY®
中国电力出版社

Jerry Bradenbaugh 著
何健辉 赵声攀 译

JavaScript 应用程序经典实例

Jerry Bradenbaugh 著

何健辉 赵声攀 译

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

O'Reilly & Associates, Inc. 授权中国电力出版社出版

中国电力出版社

图书在版编目（CIP）数据

JavaScript应用程序经典实例/（美）布雷登堡（Bradenbaugh, J.）编著；何健辉，赵声攀译．-北京：中国电力出版社，2001.8

书名原文：JavaScript Application Cookbook

ISBN 7-5083-0660-0

Ⅰ.J… Ⅱ.①布…②何…③赵… Ⅲ.JAVA语言－程序设计 Ⅳ.TP312

中国版本图书馆CIP数据核字（2001）第034250号

北京市版权局著作权合同登记

图字：01-2001-2079号

书　　名/ JavaScript应用程序经典实例

书　　号/ ISBN 7-5083-0660-0

责任编辑/ 刘江

封面设计/ Edie Freedman，张健

出版发行/ 中国电力出版社（www.inforpower.com.cn）

地　　址/ 北京三里河路6号（邮政编码100044）

经　　销/ 全国新华书店

印　　刷/ 北京市地矿印刷厂

开　　本/ 787毫米×1092毫米　16开本　31.5印张　464千字

版　　次/ 2001年7月第一版　2001年7月第一次印刷

印　　数/ 0001-5000册

定　　价/ 59.00元（册）

O'Reilly & Associates 公司介绍

为了满足读者对网络和软件技术知识的迫切需求，世界著名计算机图书出版机构O'Reilly & Associates公司授权中国电力出版社，翻译出版一批该公司久负盛名的英文经典技术专著。

O'Reilly & Associates公司是世界上在UNIX、X、Internet和其他开放系统图书领域具有领导地位的出版公司，同时是联机出版的先锋。

从最畅销的《The Whole Internet Use's Guide & Catalog》（被纽约公共图书馆评为二十世纪最重要的50本书之一）到GNN（最早的Internet门户和商业网站），再到WebSite（第一个桌面PC的Web服务器软件），O'Reilly & Associates一直处于Internet发展的最前沿。

许多书店的反馈表明，O'Reilly & Associates是最稳定的计算机图书出版商——每一本书都一版再版。与大多数计算机图书出版商相比，O'Reilly & Associates公司具有深厚的计算机专业背景，这使得O'Reilly & Associates形成了一个非常不同于其他出版商的出版方针。O'Reilly & Associates所有的编辑人员以前都是程序员，或者是顶尖级的技术专家。O'Reilly & Associates还有许多固定的作者群体——他们本身是相关领域的技术专家、咨询专家，而现在编写著作，O'Reilly & Associates依靠他们及时地推出图书。因为O'Reilly & Associates紧密地与计算机业界联系着，所以O'Reilly & Associates知道市场上真正需要什么图书。

作者简介

Jerry Bradenbaugh 是加州洛杉矶的一位高级Web程序员和技术负责人。他的JavaScript资源网站HotSyte从JavaScript早期就存在了，也是网上资格最老的JavaScript资源。他曾参与开发Netscape公司和美国第一联合银行的企业级程序。

封面介绍

本书封面上的动物是河马（hippopotamus）。河马原产于非洲的某些地区，栖息在河中或河边的草地上。河马名字的原文hippopotamus就是希腊文“河马”。它们有庞大笨重的身躯，整日里优雅地在水中走来走去。夜里河马离开水边去觅食。它们的食物主要是草，每天150磅，还有一些水生植物和落果。河马没有什么天敌，除了人类以外——河马因其象牙般的长牙，以及皮和肉而遭到人类的大肆捕杀。

河马可以活到40岁，高5英尺，长12英尺，体重可达6000~8000磅。它们的身体几乎没有毛发、皮肤呈灰色、能分泌出红色的油性液体（这经常被误认为血），以使皮肤湿润。河马的五官长得都离头顶很近，因此它们在几乎完全沉入水中时仍然可以呼吸、看和听。有趣的是，有些沼泽动物，如鳄鱼、乌龟和犀鸟经常会栖身于水里的河马之上。

目录

英文版编辑的话

欢迎你阅读本书，它是O'Reilly公司“经典实例（Cookbook）”（译注1）系列的第二本。本书和《Perl Cookbook》有很大的不同，对于这一点，大概需要说明一下。在《Perl Cookbook》的前言里，作者Larry Wall写道，本书“不是要把做好的现成东西给你（它也做不到），甚至也不是要教你如何去做（虽然它对此确有帮助），而只是把许多已经证实颇有用处的思想传递给你……”。

《Perl Cookbook》的确可以算是集Perl语言“烹饪”技巧之大成者。“发现第N个出现的匹配”和“如何晒黑黄油”差不多。而“为散列分类”完全可以认为是在“剥烤辣椒皮”。

而从另外一个方面讲，本书是一本纯粹讲述方法的书。你可以把“购物袋：JavaScript购物车”看成“带熏鲑鱼的袖珍葱饼”。每一章都会提供代码和文件，给出一个几乎完全是用JavaScript写的非常有用的Web应用程序。为本书作者所写的每一种方法做好准备，或者只摘取关键的概念，然后就可以把它们放到你自己的作品中去。（Nick Heinle的《Designing with JavaScript》一书中包含一些更小的应用方法，你可以把它们放到一个单独的网页上，相反，本书向你展示如何用JavaScript一唯一一种浏览器生来就能理解的脚本语言、编写完整的客户端Web应用程序。）

译注1：Cookbook的英文原意是“食谱”，所以下文中有很多与烹饪有关的文字。由于中文很难找到能体现原文韵味的对译，这里权且译为国人易接受的“经典实例”。

既然有两种不同的方法，那么“经典实例”是如何定义的呢？经典实例讲述的不是一些拘泥于形式的内容；它应该是一本帮助你“得心应手地构造代码”的书。我们期望着看到更多的“经典实例”系列图书出版，以各种各样的方法来实现这一目标。

Richard Koman，英文版编辑

前言

我总是感觉似乎有什么东西被遗漏了。我曾经参阅了成堆的JavaScript书籍，还有无数网站一个接一个的页面，我沉浸在尽可能多的代码和概念之中。但在从理论指导中摘取一些新的语法和灵活的技巧之后，我不知道在示例之外还能做什么。就好像我拥有一个丰富充实的厨房，但根本不懂烹调。我拥有了所有这些酷极了的JavaScript技术和代码片段，但我完全没有把握用它来解决普通的Web站点问题。当然，有些书中也讲JavaScript应用程序，但它们和Web井水不犯河水。我的意思是，十二点纸牌游戏自然很棒，电子制表也是如此，但眼下我并不准备把它们放到Web上去。

因此，这里应该是一些诀窍和方法。不仅为了检测某个浏览器的特性或是做一个图片翻转，这里还包括一些成熟的应用程序，你会很想把它们用在网站上。这里大部分原装的应用程序都很灵活，将它们拷贝到Web服务器（或本地计算机）中的文件夹里，马上就可以运行。下面的章节讲述各式各样的JavaScript，可用来帮助用户完成普通的Web任务，比如站点搜索、调查信息搜集、创建图片翻转、浏览在线资料、电子购物和其他更多事情。当然，你会想对它们做适当的修改以便更好地为你所用，而它们已经多多少少为此做了准备。另外，每一个应用程序都带有很长的解说文字，这样，你可以了解每一部分的工作原理。

读者须知

这不是一本为初学者准备的书。你无法在这里学习JavaScript基本知识，而要学习如何使用JavaScript。你不一定要是个有三年丰富经验的JavaScript老手，但如果你连 `info.replace(/</g, "<")`, `new Image()`,还有 `var itemArray = []`都搞不大懂的话，我想你在阅览的时候手头上至少需要一本JavaScript语法书。可以试试O'Reilly公司出版的《JavaScript：The Definitive Guide》（译注1）这本书，它的作者是大名鼎鼎的David Flanagan。

排版约定

斜体

用于文件名、目录路径、以及对象、变量、数组和其他实体的名称。

常规（Constant Width）

用于HTML标签、代码示例、代码片段、函数和其他代码引用方面。

常规斜体（*Constant Width Italic*）

用于用户输入的文字和可替代文字。

常规粗体（**Constant Width Bold**）

用于表示显示在屏幕上的文字。

本书的结构

对大多数部分来说，每一章由如下所示四个部分构成。

执行条件

这个短小的部分指出运行此应用程序所必需的环境。这里通常会指出Netscape

译注1：这本经典著作最新第三版的中文版《JavaScript权威指南》已由中国电力出版社出版，详情请访问www.infopower.com.cn。

Navigator或Microsoft IE的哪些版本是兼容的。这个部分还讨论所需的扩展性和分辨率，给出解决方案。

语法细则

待你将应用程序把玩之后，想要看看“盖头下面”的东西时，可以看看这个部分的内容。在这里，你可以看到关于代码的讨论，基本上是逐行给出的。到目前为止，这是一章中最长的部分，所以你在接触它之前最好调整一下心情，轻松一点。

JavaScript 技巧

当我们颇为费劲地查看语法细则的时候，将会有一个美好的终点，一个 JavaScript 技巧将在此闪亮登场，你可以把它放进自己的 Web 技能百宝囊中去。

应用程序扩展

这个部分将给出一些建议性方法，帮助你将每一个应用程序作适当扩展，使其具备更多魅力四射的功能。有时我给出建议，有时还提供代码。有时候我真是忍不住要把代码写给你——它们包括在一个你能下载的压缩文件中。不管怎样，它将让创造的源泉流淌，这样你就不会被骗，说：“天哪，我怎样才能把它弄到我的站点上去？”

关于代码

这本书都是关于应用程序的。因此毫无疑问，你将看到JavaScript 代码——非常多。一些应用程序包括好几百行，大部分都按代码本身的顺序列出来。在某些情况下，代码甚至被再三给出，这样你就不必总是在说明和代码之间跳来跳去了。

把代码放在本书中的一个缺点是……哦，就是把它放在了书中。书的宽度不足以按我们想要的形式排列所有的代码。代码经常要转到下一行，再下一行。为了保持可读性较好，只能省去了注解，但你可以在文件本身中发现足够多的注解。编辑人员为了在页面的局限内整洁地设计代码格式而搜尽脑汁，但在某些地方，你会发现在文本编辑器中阅览文件对你的眼睛要容易得多。

因为我希望你会使用这些代码，而不仅仅是看一看，所以我把所有有用的应用程序都放到了一个压缩文件中，你可以从 O'Reilly 网站中下载它。请到 http://www.oreilly.com/catalog/jscook/index.html 地址下查找“down load”链接。你会在每一章中看到对这个文件的引用。

发展和测试

我已经列出了用于本书代码开发的硬件和软件，没有特定的顺序。绝大部分都针对 Windows 环境进行了测试，但 Unix 和 Mac 的用户也许会遇到一些问题。

硬件：IBM ThinkPad 55/P75/16M, Compaq Presario/P233/100M, IBM Aptiva C23/P120/128M, DELL OptiPlex/P2-266/128M, Sun SPARC 20

操作系统：Win95 及以上，WinNT Workstation 4.0，WinNT Server 4.0 及以上，Solaris 2.5 及以上

浏览器：Netscape Navigator 3.0, 3.04 Gold, 4.0, 4.04, 4.07, 4.08, 4.5 及以上；Microsoft Internet Explorer 3.0, 3.02, 4.0, 4.01, 5.00 及以上

分辨率：640 × 480, 800 × 600, 1024 × 768, 1152 × 900, 1280 × 1024

当然，不是每一个应用程序都在所有这些情况下测试过。但我努力在编程方面做到足够稳固，使绝大部分的用户环境都能适应。

建议与评论

本书的内容都经过测试，尽管我们做了最大的努力，但错误和疏忽仍然是在所难免的。如果你发现有什么错误，或者是对将来的版本有什么建议，请通过下面的地址告诉我们：

美国：

O'Reilly & Associates,Inc.
101 Morris Street
Sebastopol, CA 95472

中国：

100080 北京市海淀区知春路 49 号希格玛公寓 B 座 809 室
奥莱理软件（北京）有限公司

询问技术问题或对本书的评论，请发电子邮件到：

info@mail.oreilly.com.cn

最后，你可以在 WWW 上找到我们：

http://www.oreilly.com
http://www.oreilly.com.cn

感谢

书的封面上写着我的名字，但我将因为把荣誉分给创作此书的其他人而感到莫大的快慰。我向那些使这一切成为可能的人们呈上发自内心的感激。

在技术方面，我要感谢Steve Quint和James Chan, Jim Esten, Bill Anderson, Roland Chow, Rodney Myers, Matthew Mastracci, Giorgio Braga, Brock Beauchamp，还有其他许多人，在我处于困境时，允许我分享他们的JavaScript宝库和其他编程经验。我还必须对 Patrick Clark 表示敬意，他的代码给了我“在线帮助程序”的灵感。感谢 Richard Koman，我的编辑，为他耐心听取我的意见，使我能够完成此书。还要感谢 Tara McGoldrick 和 Rob Romano 所付出的幕后劳动。

在感情方面，我真诚地感激我的妻子，Róndine Bradenbaugh，数月来忍受我一夜又一夜盯着PC显示器，兴奋地敲打键盘。我还要感激我的父母，他们支持着我，鼓励我提高写作技巧。

我还要谢谢那些常常被忽视的人——就是你，我的读者。是你把辛苦得来的钱交给书店，才使我能够做这一切。有那么多可用的JavaScript书籍，而你选择了我的。谢谢，真高兴你给了我机会，请让我证明你所花的钱是值得的。

原书空白页

绪论

这是一本与众不同的书。它的内容是关于用JavaScript编写尽可能大的Web应用程序。大多数人并不认为JavaScript适合用在这一方面。似乎JavaScript通常（至少被用来）只和添加图片翻转效果、访问者计数、浏览器检测等东西有关。

JavaScript的优点

对发展中的Web应用程序而言，没有一种语言或技术能自我宣称是最好的解决方案，从而垄断市场。每一种都有它的赞成者和反对者。但是，JavaScript和其他不断出现的技术（比如DHTML，Java和Macromedia公司的Flash）上的新近发展，使JavaScript能够充分利用这些工具的长处，创建更强大的Web解决方案。下面还有一些用JavaScript开发应用程序的强有力的支持论据。

易学、快速、功能强大

由于JavaScript非常简单易学，你很快就可以开始运用它。要在站点上添加一些快速功能，JavaScript是完美的方法。一旦有了基础知识，接下来创建一个完整的应用程序也不会太远了。

作为一种高级语言，JavaScript是相当强大的。你不能在机器层用它做任何动作，但它却能揭示出浏览器、网页，有时还包括运行浏览器的系统的许多特征。JavaScript

没有必要像Java或C那样非得经过编译，并且浏览器也不需要装载虚拟机来运行代码。你要做的只是编码之后载入就行了。

JavaScript在类似于Java和C++的面向对象体系结构上运行。诸如构造函数(constructor)和基于原型的继承之类的功能为开发架构添加了一个抽象层。这在很大程度上促进了代码的可重用性。

广泛运用

到目前为止，JavaScript是Web上最流行的脚本语言。全世界成千上万，乃至数百万计的网页包含有JavaScript。JavaScript有最流行的Web浏览器的支持（我们指的是IE中的JScript）。Netscape和Microsoft看来都在为扩展语言的功能而不断追求。这种支持意味着JavaScript更有可能得到网站访问者的绝大多数浏览器的支持。

减轻服务器负载

这是Web开发者采用JavaScript的首要原因之一。它可以在客户端执行通常要在服务器端处理的许多功能。对此，最好的例子之一是表单确认。老派的编码者可能还记得，就在几年以前，用来确认HTML表单用户输入的唯一方法，是将用户信息提交到Web服务器，然后将数据交给CGI脚本，以断定用户的输入是否正确。

如果数据没有错误，CGI脚本就按正常状态作处理。如果产生了某些错误，脚本就返回一个消息给用户，指出问题所在。虽然这也不失为一个解决方法，但考虑到花费就不合算了。提交表单需要从服务器要求另外一个HTTP。这个跨越网络的旅行后面仍然需要CGI脚本的再次执行。每一次用户在表单中出了差错，都要重复这个过程。用户只能等待，直到错误消息到达才能知道究竟是哪里出了毛病。

我们来看看JavaScript。现在你可以在用户将表单送回Web服务器之前确认表单元素。这就减少了经由HTTP的处理，并有效地减少了用户在表单输入中出错的机会。JavaScript同样可以读写cookie，这曾经只能由Web服务器的首部（header）设置功能来执行。

发展中的JavaScript

JavaScript 1.1面世之时，产生了极大轰动，因为我们可以用全新的“图片对象”和document.images数组来创建图片翻转了。然后JavaScript 1.2隆重出场。新潮流的闸门大开。DHTML支持、层以及其他大量增进的功能使许多编程人员晕眩不已。它太棒了，甚至让人觉得不像是真的。

到此还远未结束。JavaScript后来成了EMCA262的设计模型，一种标准化的通用脚本语言。至少有一个公司已经开发了从命令行运行JavaScript的环境。Macromedia公司将定制JavaScript调用融合进它自己的Flash技术中。Allaire公司的ColdFusion将JavaScript结合到自己基于XML的技术 —— Web Distributed Data Exchange（Web分布式数据交换，WDDX）中。JavaScript正变得越来越出色。更多的功能，更多的选项，更大的魅力。

也许你别无选择

有的时候，它是唯一的方法。假设你的ISP不允许执行CGI脚本。现在，如果要添加基于表单的Email或者使用有效的cookie技术的话，你该怎么做呢？你只能使用客户端解决方案。可以证明，JavaScript在将服务器端功能添加到“只有客户端”的网站中是最好的选择。

或许还有更多

我还可以想到更多的优点，你完全可以添加到支持论据列表中去。关键的一点是：尽管服务器端技术有其优点，JavaScript应用程序在网络中永远有其一席之地。

JavaScript基本设计策略

不管创建程序时用不用JavaScript，头脑中有一个策略总是有用的。这可以帮助你把自己的想法和代码组织起来，并加速开发和调试的过程。能找到数十份知名的出版物上讲述如何循序渐进地进行程序设计，你可以从中采纳最有利的一种策略。因此我不想在这里花太多的时间。不管怎么样，当你在<SCRIPT></SCRIPT>标签之间

写下你的代码之前、之中、之后时，记住下面几点会减少很多麻烦。它其实非常简单，就是要回答：做什么？谁做？怎样做？

应用程序的功能是什么？

首先，这个应用程序要做什么呢？要尽量具体一点。应用程序不会提供什么？假设你想要在发送 Email 的时候使用一个 HTML 表单。考虑以下这些问题。

- 表单将包括多少域？
- 用户会自己输入 Email 地址还是从一个选择列表中选取？
- 在发送之前你是否想要确认一下表单输入？你打算确认什么内容？消息？Email 地址？还是两个都确认？
- 发送Email之后发生什么事？你要让用户转到另外一个页面还是根本什么也不做？

这些问题当然还可以继续提下去。不过，正所谓种瓜得瓜，种豆得豆，如果你花时间来思考诸如此类的问题的话，你将会得到一个非常好的计划。

访问者是谁？

确定谁将使用这些信息是决定应用程序性能的关键。要确保至少在以下问题上有明确的答案：

- 人们会使用什么浏览器？Netscape Navigator？什么版本：2.x, 3.x, 4.x, 或者更高？
- 应用程序将会用在Internet上，还是企业内联网上，或者是单独的计算机上？
- 你能否确定大多数用户将会使用的显示器分辨率？
- 大部分用户会使用哪种类型的连接？56K 调制解调器？ISDN？T-1？T-3？

除了关于浏览器类型的问题之外，你可能会认为这些问题和 JavaScript 毫不相干。“连接……谁关心这个呀？我根本不需要配置一个路由器来完成这个工作。”你说的

对。你不需要进行Cisco认证考试。不过还是让我们来浏览一下这些问题，一个接一个地来，看看它们为什么会对你很重要。

可以证明，浏览器的问题是最紧迫的。一般而言，在越新的浏览器中，你能使用越新的JavaScript版本。例如，如果你的观众被限制在Navigator 2.x和IE 3.x上（虽然我不知道为什么会出现这种情况），那么你可以取消图片翻转功能。在这两种浏览器中的JavaScript和JScript版本都不支持Image和document.images对象（注1）。

由于大多数人的这些浏览器都升级到至少4.x版本了，图片翻转是可以实现的。但是现在，你必须认真处理对象模型的挑战。这意味着你需要让应用程序实现跨浏览器兼容，或者为每一个版本写不同的应用程序（可能成为徒劳无功的榜样）。

应用程序应该放在哪儿呢？是在Internet上，还是在企业内联网上，或是在什么人的公共空间中的PC机上？这个问题的答案将会为你能解决的东西依次提供更多的线索。例如，如果应用程序将在Internet上运行，你可以确信几乎可以想像到的任何类型的浏览器都会来访问你的站点，并使用（至少试图使用）这个应用程序。如果应用程序被限制在企业内联网或本地机器上，只会出现某些标准类型的浏览器。在写到这里时，我正在为一家公司做咨询工作，它是一个很大的Microsoft产品商店。如果我写的关于企业内联网的代码在Navigator上不能使用，我也无所谓；用户肯定有IE。

分辨率是另外一个重要的问题。如果你已经在页面上用了900像素宽的表，而用户只有800 × 600的分辨率，那么他们将会看不到你辛勤工作的某些部分成果。你能指望有一个对所有访问者都合适的分辨率吗？如果这是对Internet来说的，那么答案是不可能。如果用户在一个企业内联网上，你可能会幸运地实现这个愿望。一些企业会让PC硬件、软件、浏览器，甚至还有分辨率都符合标准。

连接问题也一样。假设你用一个引人注目的图片翻转序列来为斯皮尔伯格制作电影动画，并以此赚钱（如果是这样的话，可能你和我应该……唔……应该合作）。非常酷，但是56K调制解调器的用户可能会在你的代码载入所有这些图片之前跑出去看电影了。大部分的用户都能理解，网络会因为繁重的传输量而陷入困境，但不一会儿，他们就会跑到其他站点去了。带宽是需要考虑的问题。

注1：　有一些Mac机上的IE 3.x支持图片翻转。

如何克服困难？

对所有这些问题的看法听起来是斩钉截铁的，但其实并非这么简单。你可能根本没有办法适应所有的浏览器版本、分辨率，或者连接规格。现在怎么办？怎样才能让所有的人都满意，让他们为你500K的图片翻转鼓掌叫好？

考虑一下我在下面提出的一个或多个方法。把它们全都读一读，这样你就可以做出一个更周全的决定。

尝试跨浏览器方法

跨浏览器的“大多数人优先”的折衷主义方法大概是最常用、也是最好的方法了。按照这一原则，我们可以想到，大部分用户可能都有IE 4.x和Navigator 4.x。如果你只对重要的浏览器进行检测，并且编制应用程序，使它能够利用4.x版本的共同功能同时又容许它们的差异，这样，你就可以获得巨大的可能访问人数。

合理降级或改变性能

这是跨浏览器策略的一个必然结果。例如，如果你的图片翻转脚本被载入到一个不支持它的浏览器（比如IE 3.x）中，你会得到乱七八糟的JavaScript错误。可以使用浏览器检测来使这些浏览器忽略图片翻转。同样的道理，你可能会想按照分辨率来载入不同的页面。

按低标准处理

这个方法假设每一个人都有Navigator 2.0浏览器，640 × 480的屏幕分辨率，14.4K的调制解调器，还有一个Pentium 33 MHz处理器。坏消息是你将不能使用除JavaScript 1.0之外的任何东西。没有翻转，没有层，没有正则表达式，也没有外部技术（谢天谢地你能使用框架）。而好消息是：大多数人都可以使用你的应用程序。实际上，JavaScript的新近变化可能会使这种情况有所改变。无可否认，我的出发点是很低的，针对的并不是一些常见版本，比如Navigator 3.x和IE 3.x。但是低调有它自己的优点。

按高标准处理

如果你的用户没有IE 5.0，可以假定他们不是科技工作者，不适合看你的应用程序，因此也不影响它的使用。现在你可以尽情编写代码，访问IE文档对象模型（DOM），事件模型，数据绑定等等。当然，这会使你的忠心观众急剧减少，并对你的自信产生长期的影响。

为同一个应用程序提供多种版本

如果你是个受虐狂，你可以为应用程序写多个版本，比如说，一个给Navigator，另外一个给IE。这个办法至少有一个好处。让我们回到连接问题。由于确定用户的带宽是什么类型的情况通常是不可能的，因此应该允许他们做出选择。设置两个来自主页的链接，用户既能够使用T-1线路来载入你美丽的图片翻转，也可以使用调制解调器来浏览较低的版本。

应用程序中的 JavaScript 方法

以上都是一些基础。你会看到，我在本书的应用程序中将这些策略结合起来。我还将提到JavaScript方法，或者编码规范。对于我的出发点如何以及这些方法是否能为你所用，它们将给出一个较好的回答。

考虑一个应用程序的时候，我所做的首要事情是决定它对你的（或我的）网站访问者是否有用。每一个应用程序解决一个或多个基础问题。搜索、发邮件、在线帮助、设置个人参数选择、测试或收集信息、创建图片翻转等等都是网络冲浪者（Web Surfer）所喜欢的常见功能。如果一个要编写的应用程序没有通过合理性测试，我就不会为它浪费时间。

接下来我所做的是决定JavaScript是否能实现我所要的功能。这非常简单。如果答案是“是”，那么我就为它努力。如果不是，它只好进JavaScript垃圾场。

一旦选定了一个应用程序，就可以动手到文本编辑器步骤了。下面是一些我使用的编码规范。

尽可能重复使用代码

这是JavaScript源文件大展拳脚的舞台。也就是说，这些应用程序要使用JavaScript源文件，它们用下面的语法来定位：

```
<SCRIPT LANGUAGE="JavaScript1.1" SRC="someJSFile.js"></SCRIPT>
```

someJSFile.js 包含的代码可被多种脚本使用 —— 每一个都用上面的语法。贯穿本书的许多应用程序都使用JavaScript源文件。这非常有意义。为什么不重复使用呢？也可以用JavaScript源文件来隐藏应用程序剩余部分的代码。你会发现在JavaScript源文件中保存一个非常大的JavaScript数组是很有用的。使用JavaScript源文件明显是值得的，因此我们在第六章中对它进行深入的讨论。

有一些应用程序包含的代码可以很容易地剪切下来，从一个地方粘贴到另外一个地方。这些代码一般可以用作源文件。我就是这样做的，因此，你没有必要花费那么多的时间来读“请看前三章库文件中的代码……”这样的话。用这种方法，代码就总是在你的前边，直到你理解了它，把它从讨厌的页面中剪切掉。当应用程序在你的站点上良好地运行以后，可以考虑创建一个JavaScript源文件。

隔离 JavaScript

在 <HEAD></HEAD> 标签之间尽可能只保留一套 <SCRIPT></SCRIPT> 标签。

在接近顶部的地方定义全局变量和数组

即使它们的初始值被设置为空字符串或未定义，在脚本顶端定义全局变量和数组是管理变量的一个好办法，尤其是在它们会被整个脚本使用的时候。这样，你就不必查看一大堆代码来修改一个变量值了。你知道它就在接近开头的某个地方。

在全局变量之后定义构造函数

我通常把用来创建自定义对象的函数放在开头部分。这只是因为我的绝大部分对象都是在脚本前边创建的。

完全按时间顺序定义函数

换句话说，我试图按照它们在应用程序中的调用顺序来定义函数。脚本中定义的第一个函数是最先调用的，接下来调用的是第二个定义的函数，如此类推。有的时候，它的实现可能非常困难，甚至不可能。但这个方法至少改善了组织性，提高了邻近函数被相继调用的机会。

每个函数执行一个单一的操作

我力图限制每一个函数只执行一个明确的操作，比如用户输入确认，设置或获取cookie信息，一个幻灯片放映的自动化操作，显示或隐藏层，等等。这是一个非常好的理论，但是要在每一种情况下都使用它恐怕不是那么容易。实际上，我在第五章中好几次都违反了这一原则。函数只执行一个基本的操作，但却会有几十行那么长。

使用尽可能多的局部变量

我这样做是为了节约存储空间。由于JavaScript局部变量会在函数执行结束后消亡，它们所占据的空间就还给了系统。如果一个变量不需要和应用程序同生共死的话，我就把它作为局部变量，而不是全局变量。

继续

下面的内容可能会为你提供一幅关于如何着手建立自己的JavaScript应用程序，以及我如何建立应用程序的蓝图。现在就让我们来开始有趣的内容。

原书空白页

第一章

客户端搜索引擎

应用程序要点

- 高效的客户端搜索
- 多种多样的搜索算法
- 分种类和分部分的搜索结果
- 可涉及成千上万的记录
- 简单明了的JavaScript 1.0兼容性代码

JavaScript技巧

- 使用定界字符串存储多种记录
- 循环嵌套
- 轻松使用document.write()
- 利用三元算子实现循环

每个站点都会设置搜索引擎功能，但何必把这些任务都交给服务器来处理呢？客户端搜索引擎允许用户完全在客户端搜索网页。每个用户都可以在所需的页面内下载“数据库”，这比把请求送到数据库或者服务器要好多了。这个临时的数据库其实只是一个JavaScript数组，每个记录存储在数组的一个元素中。

这种方法有着许多显著的优点，最重要的是减少了服务器的工作量，提高了响应速度。同时，应该注意的是这种方法受到用户资源的限制，尤其是处理器速度和可用空间。虽然如此，对于站点来说，它仍然有着巨大的效用。你可以在本书网站下载的压缩文件的*ch01*文件中找到这个应用程序的源代码。图1-1给出了它的运行界面。

这个应用程序中还包括两种逻辑搜索方法：AND和OR。你可以通过文件名、描述信息或文档URL来进行搜索。用户可用的功能非常直接简单，你只要输入所要匹配的条目，然后按回车就行了。下面是搜索选项的细目：

- 输入由空格分开的条目，则所返回的记录至少包含其中的某一个条目（逻辑操作OR）。

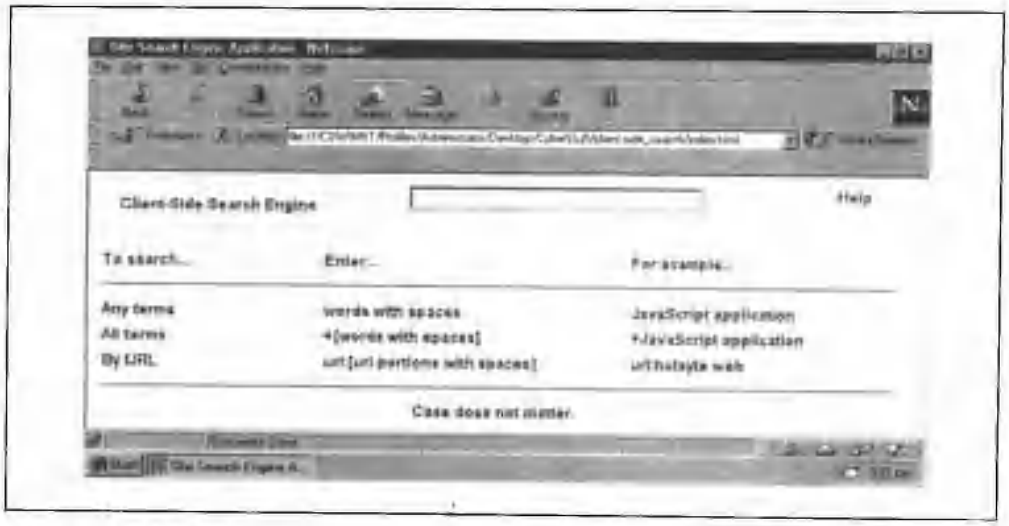


图 1-1 运行界面

- 在搜索字符串前面加“+”标记，则返回的记录应匹配输入的所有条目（逻辑操作 AND）。
- 在一个完整或部分的URL前输入“*url:*”，则返回的记录应匹配某一个在URL中输入的条目。

注意： 可不要忘记了压缩文件。正如前言中所说的一样，O'Reilly 站点中有一个压缩文件，其中可以找到本书所有的代码。要想得到这些压缩文件，请查找 http://www.oreilly.com/catalog/jscook/index.html 站点。

图 1-2 给出了一个简单搜索的结果页面。注意，这个特定的搜索使用了默认的搜索方法（没有前缀），并将javascript作为搜索字符串。每一个搜索在运行中产生一个结果页面，显示最近的搜索结果，后面还有一个回到帮助页面的快速参考链接。

如果能用URL来做搜索也是非常不错的。图 1-3 展示了一个用url:前缀指定的站点，其引擎只进行URL 搜索。在这个例子中输入的是 html 字符串，因此，引擎将返回URL 中含有 html 的所有文档。同样也会显示文件描述，但将 URL 放在前面。URL 搜索方式被限制为单一匹配条件，正如默认方式，但这并不成问题。事实上，没有多少用户喜欢用 URL 执行复杂的搜索运算。

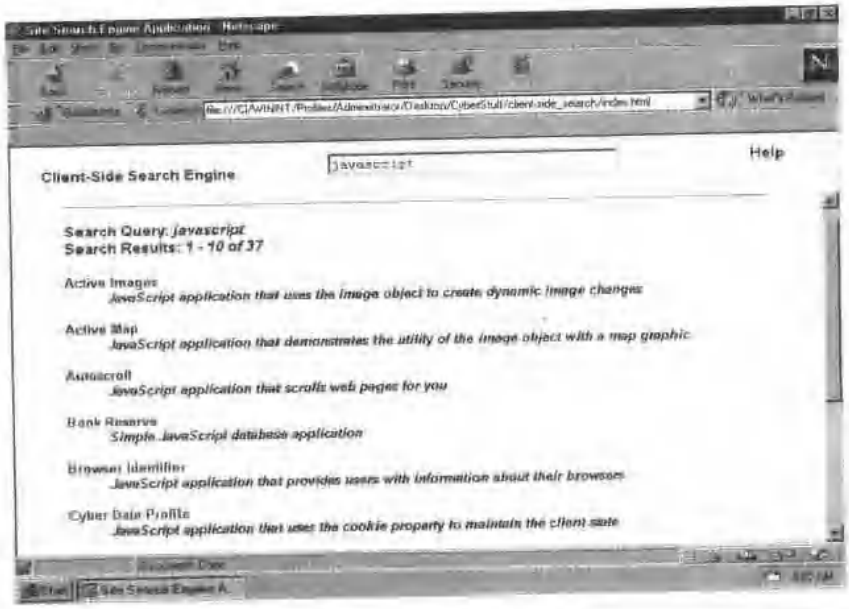


图 1-2　一个典型的搜索结果页面

图 1-3　基于 URL 记录搜索的结果

这个应用程序能限制每页显示的结果数量，并创建浏览上页或下页的按钮，这样用户就不会被长篇累牍的显示记录所累。虽然每页显示数目的默认值为10，但你完全可以按自己所希望的来设置。

执行条件

这里所讨论的应用程序需要浏览器支持JavaScript 1.1。这对使用Netscape公司的Navigator 3以上，或者微软公司的Internet Explorer 4以上的用户来说，是个好消息，而对IE 3的用户则相反。如果你决心要向下兼容的话，先别着急，我会在本章的“应用扩展”部分说明如何兼容IE 3用户（要以损失功能为代价）。

所有的客户端应用程序都要依靠客户端机器上的资源，关于这一点，这里有一个很有说服力的例子：我敢打赌，虽然客户端程序可以获得一定的空间资源来运行它的代码，但如果你传给客户端一个巨大的数据库（大约超过6000或7000个记录），那么，你的执行性能就会变得越来越差，最终死机。

以前，我在IE 4和Navigator 4中使用一个比1万条记录略少的数据库时并没有什么问题。顺便提一句，JavaScript源文件支持1MB以上的记录。我在机器上随处都可以找到24~128MB的RAM，我试图达到和Navigator 3.0 Gold同样的设置，却得到一个堆栈溢出错误，说什么数组中记录过多。最低限度是，用老配置的IBM ThinkPad的IE 3.02来浏览的JavaScript 1.0不允许215条以上的记录，当然再也没有更低的数字让人丧气了。那台笔记本电脑太过时了，你简直可以听见似乎是老鼠在推动CPU运行的声音。绝大多数的用户都希望有比较大的容量。

语法细则

这个应用程序由三个HTML文件（*index.html*,*nav.html*和*main.html*）和一个JavaScript源文件（*records.js*）组成。这三个HTML文件包括一个小的框架设置，一个进入搜索条目时的标题页和一个在显示框架中关于用法说明的默认页面。

nav.html

*nav.html*的头文件可以说是这个应用程序的大脑。事实上，此外能看见JavaScript的地方就是自动创建的结果页面。让我们来看看代码，从例1-1开始。

例1-1：nav.html源文件

```
<HTML>
<HEAD>
<TITLE>Search Nav Page</TITLE>

<SCRIPT LANGUAGE="JavaScript1.1" SRC="records.js"></SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.1">
<!--

var SEARCHANY  = 1;
var SEARCHALL  = 2;
var SEARCHURL  = 4;
var searchType = "";
var showMatches    = 10;
var currentMatch  = 0;
var copyArray     = new Array();
var docObj  = parent.frames[1].document;

function validate(entry) {
   if (entry.charAt(0) == "+") {
    entry = entry.substring(1,entry.length);
    searchType = SEARCHALL;
    }
   else if (entry.substring(0,4) == "url:") {
    entry = entry.substring(5,entry.length);
    searchType = SEARCHURL;
    }
   else { searchType = SEARCHANY; }
   while (entry.charAt(0) == " ") {
    entry = entry.substring(1,entry.length);
    document.forms[0].query.value = entry;
    }
   while (entry.charAt(entry.length - 1) == " ") {
    entry = entry.substring(0,entry.length - 1);
    document.forms[0].query.value = entry;
    }
   if (entry.length < 3) {
    alert("You cannot search strings that small. Elaborate a little.");
    document.forms[0].query.focus();
    return;
    }
   convertString(entry);
   }

function convertString(reentry) {
   var searchArray = reentry.split(" ");
```

```
46    if (searchType == (SEARCHALL)) { requireAll(searchArray); }
47    else { allowAny(searchArray); }
48    }
49
50 function allowAny(t) {
51    var findings = new Array(0);
52    for (i = 0; i < profiles.length; i++) {
53      var compareElement  = profiles[i].toUpperCase();
54      if(searchType == SEARCHANY) {
55        var refineElement = compareElement.substring(0,
56          compareElement.indexOf('|HTTP'));
57          }
58      else {
59        var refineElement =
60          compareElement.substring(compareElement.indexOf('|HTTP'),
61          compareElement.length);
62        }
63      for (j = 0; j < t.length; j++) {
64        var compareString = t[j].toUpperCase();
65        if (refineElement.indexOf(compareString) != -1) {
66          findings[findings.length] = profiles[i];
67          break;
68          }
69        }
70      }
71    verifyManage(findings);
72    }
73
74 function requireAll(t) {
75    var findings = new Array();
76    for (i = 0; i < profiles.length; i++) {
77      var allConfirmation = true;
78      var allString     = profiles[i].toUpperCase();
79      var refineAllString = allString.substring(0,
80        allString.indexOf('|HTTP'));
81      for (j = 0; j < t.length; j++) {
82        var allElement = t[j].toUpperCase();
83        if (refineAllString.indexOf(allElement) == -1) {
84          allConfirmation = false;
85          continue;
86          }
87        }
88      if (allConfirmation) {
89        findings[findings.length] = profiles[i];
90        }
91      }
92    verifyManage(findings);
93    }
94
95 function verifyManage(resultSet) {
96    if (resultSet.length == 0) { noMatch(); }
97    else {
98      copyArray = resultSet.sort();
99      formatResults(copyArray, currentMatch, showMatches);
```

```
100       }
101     }
102
103 function noMatch() {
104     docObj.open();
105     docObj.writeln('<HTML><HEAD><TITLE>Search Results</TITLE></HEAD>' +
106       '<BODY BGCOLOR=WHITE TEXT=BLACK>' +
107       '<TABLE WIDTH=90% BORDER=0 ALIGN=CENTER><TR><TD VALIGN=TOP>' +
108 '<FONT FACE=Arial><B><DL>' +
109       '<HR NOSHADE WIDTH=100%>"' + document.forms[0].query.value +
110       '" returned no results.<HR NOSHADE WIDTH=100%>' +
111       '</TD></TR></TABLE></BODY></HTML>');
112     docObj.close();
113     document.forms[0].query.select();
114     }
115
116 function formatResults(results, reference, offset) {
117     var currentRecord = (results.length < reference + offset ?
118       results.length : reference + offset);
119     docObj.open();
120     docObj.writeln('<HTML><HEAD><TITLE>Search Results</TITLE>\n</HEAD>' +
121       '<BODY BGCOLOR=WHITE TEXT=BLACK>' +
122       '<TABLE WIDTH=90% BORDER=0 ALIGN=CENTER CELLPADDING=3><TR><TD>' +
123       '<HR NOSHADE WIDTH=100%></TD></TR><TR><TD VALIGN=TOP>' +
124       '<FONT FACE=Arial><B>Search Query: <I>' +
125       parent.frames[0].document.forms[0].query.value + '</I><BR>\n' +
126       'Search Results: <I>' + (reference + 1) + ' - ' +
127       currentRecord + ' of ' + results.length + '</I><BR><BR></FONT>' +
128       '<FONT FACE=Arial SIZE=-1><B>' +
129       '\n\n<!-- Begin result set //-->\n\n\t<DL>');
130     if (searchType == SEARCHURL) {
131       for (var i = reference; i < currentRecord; i++) {
132         var divide = results[i].split('|');
133         docObj.writeln('\t<DT>' + '<A HREF="' + divide[2] + '">' +
134           divide[2] + '</A>\t<DD><I>' + divide[1] + '</I><P>\n\n');
135         }
136       }
137     else {
138       for (var i = reference; i < currentRecord; i++) {
139         var divide = results[i].split('|');
140         docObj.writeln('\n\n\t<DT>' + '<A HREF="' + divide[2] + '">' +
141           divide[0] + '</A>' + '\t<DD>' + '<I>' + divide[1] + '</I><P>');
142         }
143       }
144     docObj.writeln('\n\t</DL>\n\n<!-- End result set //-->\n\n');
145     prevNextResults(results.length, reference, offset);
146     docObj.writeln('<HR NOSHADE WIDTH=100%>' +
147       '</TD>\n</TR>\n</TABLE>\n</BODY>\n</HTML>');
148     docObj.close();
149     document.forms[0].query.select();
150     }
151
152 function prevNextResults(ceiling, reference, offset) {
153     docObj.writeln('<CENTER><FORM>');
```

```
154     if(reference > 0) {
155       docObj.writeln('<INPUT TYPE=BUTTON VALUE="Prev ' + offset +
156       ' Results" ' +
157     'onClick="parent.frames[0].formatResults(parent.frames[0].copyArray, ' +
158         (reference - offset) + ', ' + offset + ')">');
159       }
160     if(reference >= 0 && reference + offset < ceiling) {
161       var trueTop = ((ceiling - (offset + reference) < offset) ?
162         ceiling - (reference + offset) : offset);
163       var howMany = (trueTop > 1 ? "s" : "");
164       docObj.writeln('<INPUT TYPE=BUTTON VALUE="Next ' + trueTop +
165         ' Result' + howMany + '" ' +
166     'onClick="parent.frames[0].formatResults(parent.frames[0].copyArray, ' +
167         (reference + offset) + ', ' + offset + ')">');
168       }
169     docObj.writeln('</CENTER>');
170     }
171
172 //-->
173 </SCRIPT>
174 </HEAD>
175 <BODY BGCOLOR="WHITE">
176 <TABLE WIDTH="95%" BORDER="0" ALIGN="CENTER">
177 <TR>
178     <TD VALIGN=MIDDLE>
179     <FONT FACE="Arial">
180     <B>Client-Side Search Engine</B>
181     </TD>
182
183     <TD VALIGN=ABSMIDDLE>
184     <FORM NAME="search"
185       onsubmit="validate(document.forms[0].query.value); return false;">
186     <INPUT TYPE=TEXT NAME="query" SIZE="33">
187     <INPUT TYPE=HIDDEN NAME="standin" VALUE="">
188     </FORM>
189     </TD>
190
191     <TD VALIGN=ABSMIDDLE>
192     <FONT FACE="Arial">
193     <B><A HREF="main.html" TARGET="main">Help</A></B>
194     </TD>
195 </TR>
196 </TABLE>
197 </BODY>
198 </HTML>
```

这是一段很长的代码。读懂它最简单的方法莫过于从头开始，然后逐行往下。不过幸运的是，代码大体上是由一个接一个的函数组成的。

我们将按照以下的顺序来看这段代码：

- *records.js* 源文件
- 全局变量
- 函数
- HTML

records.js

第一个需要注意的是JavaScript源程序*records.js*。我们可以看见它出现在第5行中的<SCRIPT>标签下面。

它包含一个相当长的数组*profiles*。这个文件的内容在本书中已经被忽略，否则它们势必会纠缠在一起。因此，在从压缩文件中获取文件之后，启动文本编辑器，并打开*ch01/records.js*，把它看做你的数据库。每一个元素是一个由三部分组成的字符串。这儿有一个例子：

```
"http://www.serve.com/hotsyte|HotSyte-The JavaScript Resource|The " +
  "HotSyte home page featuring links, tutorials, free scripts, and more"
```

记录部分被“|”符号分隔开来了。在把数据库中的匹配记录显示在屏幕上时，这些符号十分方便易用。记录的第二个部分是文档标题（它和TITLE标签毫不相干）；第三部分是文档描述；而第一部分则是文档的URL。

顺便说一句，没有什么规定禁止使用除“|”以外的符号来分隔记录的几个部分。但要相信，用户大约不喜欢在输入查询字符串（可能是&^或者~[%）的时候把它作为其中的一个部分。别使用反斜杠符号来混淆视听，JavaScript根本不会解析它，这样会产生一个匪夷所思的搜索结果，或者完全终止应用程序。

为什么所有这些功能代码都被包括在JavaScript源文件里呢？原因有二：模块化和可读性。如果站点由数百个网页构成，你可能就会想要一个服务器端程序来产生包括所有记录的代码。如果这些代码产生在JavaScript源文件里，就显得比较有组织性。

同样，你也可以通过在代码中包含*records.js*文件，而在其他搜索应用软件中使用这个数据库。另外，我不得不把所有的代码复制到一个HTML文件里，作为源代码来显示。

JavaScript 技术：使用定长字符串存储多种类型的记录

这种应用依赖于对信息条目的搜索，和数据库的查询非常类似。仿效数据库的查询，JavaScript 可以用同样的格式化数据来解析（搜索）一个数组。

看起来把每一个数组元素赋值为一个数据似乎是一种常识（比如一个URL 或者一个网页的标题）。这是可以实现的，但是你却给自己设置了潜在的危机。

如果用常见符号（如‘|’）连接多个子串作为一个数组元素，可以有效地减少全局数组元素的数量。在解析每一个数组元素时，JavaScript 中用于 *String* 对象的 split()函数将会为每个元素创建一个数组。换句话说，如果能在函数里用局部数组变量，例如：

```
var records = new Array("The Good", "The Bad",
  "and The JavaScript Programmer"),
```

为什么还要用这样的全局数组变量呢：

```
var records = "The Good|TheBad|and The JavaScript Programmer".
split('|');
```

现在你也许在想：“这不是半斤八两吗？它们有什么区别呢？”不同之处在于前者声明了三个全局元素，它们占据一定的内存空间直到删除。而后者仅仅声明了一个全局元素。在搜索的时候，这三个用（‘|’）创建的元素是临时性的，因为它们是局部元素。

对后一种方法而言，JavaScript 在搜索函数运行之后删除记录变量，这就释放了内存空间。并且编码的工作量较小。对我而言，倒宁肯选择后一种方式。等我们看到用于解析的代码时会再次接触这个观点。

全局变量

例 1-1 的第 9～16 行完成全局变量的定义和初始化工作。

```
var SEARCHANY    = 1;
var SEARCHALL    = 2;
var SEARCHURL    = 4;
var searchType    = '';
var showMatches  = 10;
```

```
var currentMatch  = 0;
var copyArray  = new Array();
var docObj    = parent.frames[1].document;
```

接下来将一一说明变量的功能：

SEARCHANY

表示使用任何一个输入条目进行搜索。

SEARCHALL

表示利用所有输入条目进行搜索。

SEARCHURL

表示只搜索URL（利用任何一个输入条目）。

searchType

指明搜索的类型（可以是*SEARCHANY*, *SEARCHALL*或*SEARCHURL*）。

showMatches

确定每页显示的记录数量。

currentMatch

确定当前结果页面上应最先显示哪一条记录。

copyArray

搜索结果临时数组的备份，用来显示上页或下页的内容。

docObj

引用第二个框架中的文档对象的变量。这并不能决定应用程序的好坏，但它可以帮助你更好地管理代码，因为显示结果的时候需要多次访问对象(parent.frames[1].document)。用*docObj*来引用对象，减少了代码数量，可以作为修改操作的集中点。

函数

下面让我们来看看主要函数：

validate()

用户一旦点击回车按钮，位于第18行的validate()函数将会确定用户的搜索目标和搜索方法。我们回忆一下这三个选项：

- 搜索文档标题和描述，只要求匹配一项。
- 搜索文档标题和描述，要求匹配所有项目。
- 搜索URL文档或路径，只要求匹配一项。

validate()函数依靠检测所接收字符串的前几位字符，来确定搜索目标和搜索方法。搜索方法是怎样确立的呢？我们利用的是*searchType*变量。如果用户希望包括所有的项目，那么*searchType*就设置为*SEARCHALL*。如果用户要搜索标题和描述，validate()将把*searchType*设置为*SEARCHALL*（顺便说一句，这也是默认值）。如果用户要搜索URL，*searchType*设置为*SEARCHURL*。下面是设置的过程。

第19行显示了*String*对象的charAt()函数将“+”符号作为第一个符号的查找过程。如果找到，搜索方法将设为第二种（AND逻辑操作）。

```
if (entry.charAt(0) == "+") {
 entry = entry.substring(1,entry.length);
 searchType = SEARCHALL;
 }
```

第23行给出了*String*对象的substring()查找“url:”的过程。如果找到这个字符串，*searchType*就会得到相应设置。

```
if (entry.substring(0,4) == "url:") {
 entry = entry.substring(5,entry.length);
 searchType = SEARCHURL;
 }
```

第20和24行的substring()函数又如何呢？待validate()知晓了搜索目标和搜索方式之后，这些字符标识（“+”和“url:”）就没什么用了。因此，validate()把所需的字符号码从字符串前边开始往下移动。

如果在字符串头没有发现“+”或“url:”符号，validate()将把变量*searchType*设置为*SEARCHANY*，并且在调用convertString()函数之前作一定的清除工作。第28和32行的*while*语句从字符串的开头和结尾处清除多余的空格。

发现用户的优先选择并且清除多余空格后，validate()必须确保有内容保留下来以供搜索时使用。第36行将核实搜索字符串至少应有三个字符。较少字符的搜索并不会减少有效的搜索结果，但是你可以根据个人喜好进行一些修改：

```
if (entry.length < 3) {
 alert("You cannot search strings that small. Elaborate a little.");
  document.forms[0].query.focus();
  return;
  }
```

这一个问题如果顺利，validate()将调用converString()，传递搜索字符串(entry)的一个删节了的副本。

convertString()

convertString()执行两个相关联的操作：将字符串分离为数组元素，然后调用适当的搜索函数。*String* 对象的Split()函数按空格将用户输入的字符串分开，把结果放到数组 *searchArray* 中。这个过程在第45行实现，如下所示：

```
var searchArray = reentry.split(" ");
```

举个例子，如果用户在搜索区域输入字符串“client-side JavaScript development”，数组 *searchArray* 将把client-side，JavaScript和development三个值分别保存在元素0，1和2中。接下来要注意，convertString()将按照 *searchType* 的值调用适当的搜索函数。你可以在第46～47行看到这一点：

```
if (searchType == (SEARCHALL)) { requireAll(searchArray); }
else { allowAny(searchArray); }
```

如你所见，要调用一个或者两个函数。它们执行类似的动作，但彼此并不完全相同。下面是allowAny()和requireAll()两个函数的对照。

allowAny()

顾名思义，当应用程序只有一个最小的单一匹配时，这个函数将从序列中被调用。在第50～68行，你将会看到：

```
function allowAny(t) {
  var findings = new Array(0);
  for (i = 0; i < profiles.length; i++) {
    var compareElement  = profiles[i].toUpperCase();
    if(searchType == SEARCHANY) {
      var refineElement  =
        compareElement.substring(0,compareElement.indexOf('|HTTP'));
      }
```

```
else {
  var refineElement =
    compareElement.substring(compareElement.indexOf('|HTTP'),
    compareElement.length);
  }
for (j = 0; j < t.length; j++) {
  var compareString = t[j].toUpperCase();
  if (refineElement.indexOf(compareString) != -1) {
    findings[findings.length] = profiles[i];
    break;
    }
```

两个搜索函数之后的内容是利用嵌套的*for*循环来比较字符串。你可以参见“JavaScript技巧：循环嵌套”获得更多的相关信息。*for*循环在第52和63行运行。第一个*for*循环负责迭代每一个*profiles*数组元素（从源文件中）。在每一轮循环中，第二个*for*循环迭代从convertString()传递过来的每一个搜索条目。

为了使用户不因为使用大写或小写字母而丢失匹配记录，第53和64行分别定义了*compareElement*和*compareString*两个局部变量，而后将它们分别初始化为记录和搜索条目的大写字母表示。现在，不管用户搜索的是“JavaScript”还是“javascript”，甚至是“jAvasCRIpt”都没有问题了。

AllowAny()仍然需要确定究竟是根据文档标题和描述进行搜索还是根据URL来搜索。因此，局部变量*refineElement*，这个将要和每个搜索条目进行比较的字符串要按照在第55或59行searchType的值进行设置。如果*searchType*的值是*SEARCHANY*，*refineElement*就要设置为包含记录的文档标题和描述的字符串。否则*searchType*必定为*SEARCHURL*，则*refineElement*被设置为包含文档URL的子串。

还记得“|”符号吗？这就是JavaScript能够区分记录不同部分的关键。因此，substring()将返回一个以0开头，以出现在第一个“|HTTP”之前的字符结束的字符串，或者返回一个以第一个“|HTTP”开头，直到元素结束的字符串。现在，我们就有了用来同用户的输入作比较的内容。在第65行可以看到：

```
if (refineElement.indexOf(compareString) != -1) {
  findings[findings.length] = profiles[i];
  break;
  }
```

如果在*refieneElement*中发现*compareString*，我们就得到了一个匹配（这是一个花

时间的工作）。原始记录（不是我们所搜索的删减过的URL形式）被加到第66行的*findings*数组中。我们可以利用*findings.length*作为索引不断为元素赋值。

一旦发现了一个匹配，当然就没有必要再将它和其他搜索字符串作比较。第67行有中断语句来停止*for*循环对当前记录的比较。严格说来这并不是必需的，但它减少了不必要的处理过程。

对所有记录和搜索条目进行迭代后，allowAny()将*findings*数组中的每一个匹配记录传递给位于第95～101行的verifyManage()函数。如果搜索成功，formatResults()将被调用来打印结果。否则，noMatch()函数将搜索失败的消息通报给用户。关于formatResults()和noMatch()，我们会在下一章里作进一步的讨论。让我们以requireAll()来结束其余搜索方法的研究。

requireAll()

在你的搜索条目之前加一个“+”符号，requireAll()就会被调用。除了用户输入的所有搜索条目必须得到匹配外，这个函数和allowAny()非常接近。对于allowAny()，一旦记录匹配，它就被加到结果序列中。而在这个函数里，我们必须耐心等待，直到每一个记录在确定是否

为匹配结果之前同所有条目比较一遍。函数从第74行开始：

```
function requireAll(t) {
  var findings = new Array();
  for (i = 0; i < profiles.length; i++) {
    var allConfirmation = true;
    var allString       = profiles[i].toUpperCase();
    var refineAllString = allString.substring(0,
      allString.indexOf('|HTTP'));
    for (j = 0; j < t.length; j++) {
      var allElement = t[j].toUpperCase();
      if (refineAllString.indexOf(allElement) == -1) {
        allConfirmation = false;
        continue;
        }
      }
    if (allConfirmation) {
      findings[findings.length] = profiles[i];
      }
    }
  verifyManage(findings);
  }
```

JavaScript 技巧：循环嵌套

AllowAny()和requireAll()两个函数都用了*for*循环的嵌套。对与一维数组相对的多维数组的迭代来说，这是一个简单有用的技巧。(JavaScript的数组在理论上说是一维的．但是，JavaScript可以像下面所描述的那样模拟多维数组。) 考虑这个五个元素的一维数组：

```
var numbers =  ("one", "two", "three", "four", "five");
```

如果想把一个字符串同这里的每一个元素作比较，仅仅用一个*for*（*while*）循环就行了，在运行中将每一个数组元素同字符串作比较。就像这样：

```
for (var i = 0; i < numbers.length; i++) {
  if (myString == numbers[i]) { alert("That's the number");
  break;
  }
```

要求可别太高，我们来看一看，多维数组其实也就是数组中的数组，例如：

```
var numbers = new Array(
new Array("one", "two", "three", "four", "five"),
new Array("uno", "dos", "tres", "cuatro", "cinco"),
new Array("won", "too", "tree", "for", "fife")
);
```

单循环对此没有什么用，我们需要更强的功能。第一个*Numbers*数组是个一维（1 × 5）数组。新形式数组是多维的（3 × 5）。迭代所有这15个元素（3 × 5）需要一个特殊的循环：

```
for (var i = 0; i < numbers.length; i++) {          // 1...
  for (var j = 0; j < numbers[i].length; j++) {    // and 2.
    if (myString == numbers[i][j]) {
      alert("Finally found it.");
      break;
      }
    }
  }
```

对每一个元素来说，将面对两种截然不同的命运。我们可以划分得更清楚些。如果我们要在一个表格中制作一个216个网页安全色的调色板，每个单元一种颜色，该怎么办呢？嵌套循环可以帮我们的大忙。但是这次，我们只使用一维数组。

使用十六进制数，网页安全色将变为一组六位数字表示，每两位代表一个颜色

成分，比如FFFFFF,336699和99AACC。构成网页安全色的所有两位数字对为：33，66，99、AA、CC、FF。我们建立这样一个数组：

```
var hexPairs = new Array("33","66","99","AA","CC","FF");
```

“只有一个一维数组，这本书也太不值钱了，我不想要啦。”

先不要跑到书店里去。虽然有三维，但我们在每一维里使用同样的数组。下面是具体的方法：

```
var str = ' ';

//开始做表
document.writeln('<H2>Web Safe Colors</H2>' +
  '<TABLE BORDER=1 CELLSPACING=0>');
for (var i = 0; i < hexPairs.length; i++) {
  // 创建一行
  document.writeln('<TR>');
  for (var j = 0; j < hexPairs.length; j++) {
    for (var k = 0; k < hexPairs.length; k++) {
    // 为带空格的行创建单元格字符串
    // 注意每个背景色用到了三个hexpairs元素
    str += '<TD BGCOLOR="' + hexPairs[i]  + hexPairs[j] + hexPairs[k] +
    '">   </TD>';
      }
    // 写一行数据并重置字符串
    document.writeln(str);
    str ='';
    }
  // 结束一行
  document.writeln('</TR>');
  }
// 结束表格
document.writeln('</TABLE>');
```

在网页文档中输入这段代码（它在压缩文件的*\Ch01\websafe.html*下），你将会得到一个6 × 36的表格，包含了所有216个网页安全色。三维使用三重循环。当然，你可以用无数种方法修改调色板表格，但这仅仅只是让你知道嵌套循环怎么样使你的代码变得简单有效。

初看allowAny()，情况似乎中规中矩。嵌套循环，大写字母，还有变量定义——它们都各就各位。但是在第79～80行，情况发生了变化：

```
var refineAllString = allString.substring(0,allString.indexOf('|HTTP'));
```

注意，变量*searchType*并不像它在allowAny()中第50行那样，用来确定记录的哪个部分要保留下来用于搜索，这是没必要的。requireAll()只当*searchType*的值为*SEARCHALL*时才会调用（见第46行）。URL搜索不包括AND逻辑操作，因此，比较文档标题和描述是明摆着的事儿。

requieAll()函数可就有点难对付了。由于用户所输入的所有条目都要出现在被比较的字符串中，搜索逻辑就会比它在allowAny()中受到更多的限制。看一下第83行～86行：

```
if (refineAllString.indexOf(allElement) == -1) {
  allConfirmation = false;
  continue;
  }
```

如果在一个记录首次同搜索条目不匹配的时候就排除它，将会比用它的号码与匹配的号码相比较，来排除不合格记录要简单得多。因此，当一个记录第一次不包含匹配的时候，continue语句就会告诉JavaScript快点将它忘掉，并着手下一条记录。

如果一个记录已经同所有条目作了比较，并且局部变量*allConfirmation*仍然为true，我们就得到了一个匹配。*allConfirmation*在记录第一次匹配失败时变为false。这样，当前记录就在第89行被加到*findings*临时数组里。这个条件可不是那么容易达到的，但搜索结果可能会因此而详细得多。

一旦所有的记录都用这种方法完成了检测，*findings*就被传递到verifyManage()去检验是否为有效结果。如果存在任何匹配记录，就调用formatResults()。否则，verifyManage()调用noMatch()告诉用户这个不幸的结果。

verifyManage()

你可能已经发现了，这个函数用来确定用户的搜索是否产生了匹配记录，并调用未知的一个或两个函数来打印结果。它从第95行开始：

```
function verifyManage(resultSet) {
if (resultSet.length == 0) {
     noMatch();
      return;
      }
    copyArray = resultSet.sort();
    formatResults(copyArray, currentMatch, showMatches);
    }
```

AllowAny()和requireAll()在运行完各自的进程后都会调用verifyManage()，并把*findings*数组作为参数进行传递。第96行显示，如果*resultSet*数组（*findings*的拷贝）为空，则verifyManage()将调用noMatch()函数。

如果*resultSet*中不论如何至少包含一个匹配记录，那么全局变量*copyArray*将会被设置为*resultSet*中所有元素的排序形式。排序不是必要的，但它是给你的搜索结果排序的重要途径，你不用操心将以何种顺序将记录加到*profiles*数组中去。你可以不断地在尾部添加，同时应该清楚，如果有匹配产生，它们会被排序。

因此，我们何必把已经有的记录再做一次拷贝呢？记住，*findings*是局部的，也就是临时的数组。一旦执行完一次搜索（也就是说应用程序执行了一个搜索函数），*findings*就不存在了，它所占用的存储空间被释放以作将来的使用。这可是件好事。没有理由占据我们可能要在其他地方使用的存储空间，但要确保我们仍然要能够使用那些记录。

由于应用程序设定每页显示10行记录，用户将只能看到匹配结果的一个子集。*copyarray*变量是全局性的，因此，把临时性存储结果分离，赋值到*copyArray*中，这就完整地保存了匹配记录。用户现在一次可以看10条、15条，甚至更多的记录。这个全局变量将保存匹配结果，直到用户提交一个新的搜索。

VerifyManage()所做的最后一件工作是调用formatResult()，传递一个指数（*currentMatch*），指出从哪一个记录开始以及每页显示多少条记录（showMatches）。*CurrentMatch*和*showMatches*都是全局变量，它们在函数执行完后并不会消亡，我们需要它们来继续应用程序的执行。

noMatch()

noMatch()所做的工作和它的名字非常相符。如果你的搜索没有得到任何匹配，这个函数会发送坏消息。它非常短小精炼，但仍然能产生一个定制的结果页面（或者没有结果），声明用户所输入的搜索条目一项匹配记录都没有产生。它从第103行开始：

```
function noMatch() {
  docObj.open();
  docObj.writeln('<HTML><HEAD><TITLE>Search Results</TITLE></HEAD>' +
    '<BODY BGCOLOR=WHITE TEXT=BLACK>' +
```

```
    '<TABLE WIDTH=90% BORDER=0 ALIGN=CENTER><TR><TD VALIGN=TOP>' +
    '<FONT FACE=Arial><B><DL>' +
    '<HR NOSHADE WIDTH=100%>"' + document.forms[0].query.value +
    '" returned no results.<HR NOSHADE WIDTH=100%>' +
    '</TD></TR></TABLE></BODY></HTML>');
  docObj.close();
  document.forms[0].query.select();
  }
```

formatResults()

这个函数所要完成的工作就是完美地把匹配记录显示给用户。虽然不是特别难，但这个函数的确有很广的覆盖面。下面是一个成功的结果显示所应有的成分：

- 一个 HTML 的首部、标题和主体。
- 文档标题，描述，以及每个匹配记录的 URL 及其链接。
- 必要的时候应有用于浏览前后记录的“上页”和“下页”按钮。

HTML 的首部和标题

HTML的首部和标题非常简单易懂。第116～129行给出了首部、标题和主体内容的开始部分。让我们来看一看：

```
function formatResults(results, reference, offset) {
  var currentRecord = (results.length < reference + offset ?
    results.length : reference + offset);
  docObj.open();
  docObj.writeln('<HTML><HEAD><TITLE>Search Results</TITLE>\n</HEAD>' +
    '<BODY BGCOLOR=WHITE TEXT=BLACK>' +
    '<TABLE WIDTH=90% BORDER=0 ALIGN=CENTER CELLPADDING=3><TR><TD>' +
    '<HR NOSHADE WIDTH=100%></TD></TR><TR><TD VALIGN=TOP>' +
    '<FONT FACE=Arial><B>Search Query: <I>' +
    parent.frames[0].document.forms[0].query.value + '</I><BR>\n' +
    'Search Results: <I>' + (reference + 1) + ' - ' + currentRecord +
    ' of ' + results.length + '</I><BR><BR></FONT>' +
    '<FONT FACE=Arial SIZE=-1><B>' +
    '\n\n<!- Begin result set //-->\n\n\t<DL>');
```

在打印首部和标题之前，我们来查一下应该从哪一条记录开始。我们知道第一条要显示的记录从`results[reference]`开始。我们必须显示*offset*条记录，除非*reference*+*offset*大于记录总数。为了得到这个结果，我们用三元算子来确定哪一个更大。在第117行将*currentRecord*变量设为此较大值。我们马上就要用到这个值。

现在，formatResults()在网页上显示HTML的首部和标题。主体部分由一个居中的表格和一条水平线开始。应用程序非常容易地给用户一个搜索条目字符串（125行），它来自表单域值：

```
parent.frames[0].document.forms[0].query.value。
```

但事情在第126行变得有点棘手了。这表示已经到了搜索结果的显示。页面上显示的文本行指出了匹配记录的总数和当前子集，例如：

```
Search Results:  1 - 10 of 38
```

我们需要三个数字来解决这个问题——所显示子集的首条记录数，要显示的记录数目，和在匹配记录存储空间里*copyArray*的长度。让我们按部就班来看看这个过程。记住，这不是记录显示的逻辑。这是让用户知道有多少条记录和从哪一条开始的逻辑。具体步骤如下：

1. 将当前记录的数字赋给变量*reference*，然后打印。
2. 加上名为*offset*的数，它表示每页要显示的记录条数（在本例中为10）。
3. 如果*reference* + *offset*的总数大于匹配记录的总数，就打印匹配记录总数。否则，打印*reference* + *offset*的总数（这个值已经被确定，并且在*currentRecord*中也有）。
4. 打印匹配记录的总数。

第一步和第二步看起来真是太简单了。回忆一下verifyManage()中的代码，特别是第99行：

```
formatResult(copyArray, currentMatch, showMatches);
```

局部变量*results*是*copyArray*的复本。*reference*变量设值为*currentMatch*，因此，*reference* + *offset*的总数就是*currentMatch* + *showResults*的总数。在代码的前边几行（准确的说是第13和14行），*showMatches*设为10，*currentMatch*设为0。因此，*reference*的初值为0，*reference*+*offset*等于10。打印*reference*完成第一步。我们刚得到的数字则由第二步使用。

在第三步里，我们利用三元算子（第117～118行）来确定*reference* + *offset*的总

数是否比匹配记录总数大。换句话说，*offset* 加上 *reference* 所产生的数字会不会比记录总数更大？如果 *reference* 是 20，总共有 38 条匹配记录，在 *reference* 上加 10 我们就得到 30。那么就会有这样的显示结果：

```
Search Results: 20 - 30 of 38
```

如果 *reference* 是 30，但是总共有 38 条匹配记录，在 *reference* 上加 10 就会得到 40。那么就会像这样显示：

```
Search Results: 30 - 40 of 38
```

这是不会发生的。如果只发现 38 条记录，搜索引擎不可能显示第 39 和 40 条记录。这表示已经到了记录的最后。因此，将显示匹配记录总数来代替 *reference* + *offset* 的值。这使我们到了第四步，进程的最后。

```
Search Results: 30 - 38 of 38
```

注意：formatResults()函数里到处都有“\n”和“\t”这样的特殊符号。“/n”是一个换行符，就像你在文本编辑器里输入代码时按回车键一样。“\t”和按 Tab 键相同。本例中这些符号的作用在于，如果你查看搜索结果 HTML 页面的源代码，使用它们会使代码显得相当整洁。我在此处加上它们的目的是为了看得更清楚。要记住它们不必要，也不影响你的应用程序，如果你认为它们把代码弄得一团糟，那完全可以不用。在本书下面的部分我会尽量不浪费它们。

显示文档标题、描述和链接

现在，我们已经得到了所需记录子集，应该在页面上显示了。让我们进入第 130～143 行：

```
if (searchType == SEARCHURL) {
  for (var i = reference; i < currentRecord; i++) {
    var divide = results[i].split('|');
    docObj.writeln('\t<DT>' + '<A HREF="' + divide[2] + '">' +
      divide[2] + '</A>' + '\t<DD>' + '<I>' + divide[1] + '</
I><P>\n\n');
    }
  }
else {
  for (var i = reference; i < currentRecord; i++) {
    var divide = results[i].split('|');
```

```
    docObj.writeln('\n\n\t<DT>' + '<A HREF="' + divide[2] + '">' +
      divide[0] + '</A>' + '\t<DD>' + '<I>' + divide[1] + '</I><P>');
    }
}
```

第131行和138行给出了两个*for*循环，除了打印条目的顺序不同以外，它们对*currentRecord*执行相同的操作。变量*searchType*又出现了。如果它的值为*SEARCHURL*，URL将作为链接文本来显示。否则，*searchType*将等于*SEARCHANY*或*SEARCHALL*。在这两种情况下，文档标题都将作为链接文本来显示。

搜索类型是已经确定了的，但是你要怎样才能很好地显示记录呢？我们只需要利用循环来迭代记录子集，然后把记录分为标题、描述和URL，将它们按照这个顺序放置。下面是两种情况（URL搜索或其他）下都用到的*for*循环：

```
for (var i = reference; i < lastRecord; i++) {
```

现在来看记录部分。回想一下records.js文件。*profiles*的每个元素都是一个字符串，它用来确定某个记录，由“|”符号分成几个部分。这就是我们用来把它们分隔开的方法：

```
var divide = results[i].split('|');
```

对于每一个字符串元素，局部变量*divide*设为一个数组，它的每一个元素是被“|”所分隔开的一个部分。第一个元素(divide[0])是URL，第二个元素(divide[1])是文档标题，而第三个(divide[2])则是文档描述。每一个元素都以适当的HTML语句来显示（我选择<DL>，<DT>和<DD>标签）。如果用户利用URL作搜索，URL将作为链接文本来显示。否则，文档标题就成为链接文本。

添加“上页”和“下页”按钮

唯一还要做的事就是添加按钮，使用户能够浏览前面的和后面的记录子集。确切说来，它是在我们马上就要讨论到的prevNextResults()函数中完成的，但下面是formatResults()的最后几行代码：

```
    docObj.writeln('\n\t</DL>\n\n<!- End result set //-->\n\n');
    prevNextResults(results.length, reference, offset);
    docObj.writeln('<HR NOSHADE WIDTH=100%>' +
      '</TD>\n</TR>\n</TABLE>\n</BODY>\n</HTML>');
    docObj.close();
    }
```

函数在这里调用prevNextResults()，并添加一些HTML的结尾代码，然后将焦点设为查询字符串文本域。

prevNextResults()

如果你已经毫无怨言地看完了这个函数，会发现它并不太长。prevNextResults()如下所示，从第152行开始：

```
function prevNextResults(ceiling, reference, offset) {
  docObj.writeln('<CENTER><FORM>');
  if(reference > 0) {
    docObj.writeln('<INPUT TYPE=BUTTON VALUE="Prev ' + offset +
      ' Results" onClick="' +
      parent.frames[0].formatResults(parent.frames[0].copyArray, ' +
      (reference - offset) + ', ' + offset + ')">');
    }
  if(reference >= 0 && reference + offset < ceiling) {
    var trueTop = ((ceiling - (offset + reference)) < offset) ?
      ceiling - (reference + offset) : offset);
    var howMany = (trueTop > 1 ? "s" : "");
    docObj.writeln('<INPUT TYPE=BUTTON VALUE="Next ' + trueTop +
      ' Result' + howMany  + '" onClick="' +
      parent.frames[0].formatResults(parent.frames[0].copyArray, ' +
      (reference + offset) + ', ' + offset + ')">');
    }
  docObj.writeln('</CENTER>');
  }
```

函数在结果页面上使用了一个居中的HTML表单，它包含一个或两个按钮。第1-3节展示了一个带有“上页”和“下页”按钮的结果页面。有三种可能的按钮组合：

- 显示第一个结果页面时只有“下页”按钮，前边不存在任何记录。
- 除第一个和最后一个结果页面以外的页面，都要同时有“上页”和“下页”按钮。
- 对于最后一个结果页面，只用“上页”按钮，后面没有什么记录了。

三个组合方式。两个按钮。这意味着应用程序必须清楚什么时候该显示某个按钮，什么时候不显示。下面的表描述了各个组合出现的具体情况。

JavaScript 技巧：轻松使用 document.write()

再看一下formatResults()。你会发现我们调用了document.write()或者document.writeln()将HTML代码写到页面上去。这一阶段要处理的代码一般都相当长，跨越数行，彼此之间靠“+”符号来连接。虽然你可能认为，如果每行都调用document.writeln()会增加可读性，但仍有尝试其他方法的必要，这就是我要表达的意思。下面是formatResults()中的一段代码：

```
function formatResults(results, reference, offset) {
    docObj.open();
    docObj.writeln('<HTML>\n<HEAD>\n<TITLE>Search Results</TITLE>\n
      </HEAD>' +
    '<BODY BGCOLOR=WHITE TEXT=BLACK>' +
    '<TABLE WIDTH=90% BORDER=0 ALIGN=CENTER CELLPADDING=3><TR><TD>' +
    '<HR NOSHADE WIDTH=100%></TD></TR><TR><TD VALIGN=TOP>' +
    '<FONT FACE=Arial><B>Search Query: <I>' +
    parent.frames[0].document.forms[0].query.value + '</I><BR>\n' +
    'Search Results: <I>' + (reference + 1) + ' - ' +
    (reference + offset > results.length ? results.length :
      reference + offset) +
    ' of ' + results.length + '</I><BR><BR></FONT>' +
    '<FONT FACE=Arial SIZE=-1><B>' +
    '\n\n<!- Begin result set //-->\n\n\t<DL>');
```

只有唯一的一种调用把文本写到页面上，不是很花哨。一种可供选择的办法是在每行都作调用，并将它们排列整齐。

```
function formatResults(results, reference, offset) {
  docObj.open();
  docObj.writeln('<HTML><HEAD><TITLE>Search Results</TITLE>\n</HEAD>');
  docObj.writeln('<BODY BGCOLOR=WHITE TEXT=BLACK>');
  docObj.writeln('<TABLE WIDTH=90% BORDER=0 ALIGN=CENTER ' +
    'CELLPADDING=3><TR><TD>');
  docObj.writeln('<HR NOSHADE WIDTH=100%></TD></TR><TR><TD VALIGN=TOP   ');
  docObj.writeln('<FONT FACE=Arial><B>' +  'Search Query: <I>' +
    parent.frames[0].document.forms[0].query.value + '</I><BR>\n');
  docObj.writeln('Search Results: <I>' + (reference + 1) + ' - ' );
  docObj.writeln( (reference + offset > results.length ?
    results.length : reference + offset) +
    ' of ' + results.length + '</I><BR><BR></FONT>' +
    '<FONT FACE=Arial SIZE=-1><B>');
  docObj.writeln('\n\n<!- Begin result set //-->\n\n\t<DL>');
```

这样看起来可能会比较整齐，但每一个这种调用都意味着搜索引擎会多费点事。想想吧，什么是你希望做的：上五次商店，每次买一点点东西；或者上一次商店就把东西全买齐？只消用“+”把字符串分开输入，事情就解决了。

只显示“下页”按钮

我们应该在什么地方包括“下页”按钮？回答是：除最后一个外的所有结果页面。或者说，结果页面所显示的最后一个记录数（*reference+offset*）小于记录总数的时候。

现在，我们又该在什么地方排除“上页”按钮呢？回答：在第一个结果页面。也就是 *reference* 为 0（从 *currentMatch* 获得）的时候。

“上页”和“下页”按钮同时出现

什么时候两个都要？除最后一页外其他的页面都应该包括“下页”按钮，同样，除首页外的页面都要“上页”按钮。只要 *reference* 大于 0 时我们都需要“上页”按钮，而 *reference+offset* 小于记录总数时我们都不能不显示“下页”按钮。

只显示“上页”按钮

知道在什么时候显示“上页”按钮而不要“下页”按钮吗？答案是：最后一个结果页面显示的时候。换句话说，就是 *reference* + *offset* 大于或者等于匹配记录总数的时候。

这种说法可能还不够精细，但我们至少知道什么时候要哪个按钮，第 154～160 行的 *if* 语句解决了这个问题。这些语句依靠当前子集和剩余数量来决定选择“上页”和“下页”按钮中的某个或两个都要。

不管用户点击这两个按钮中的哪一个，都会调用 formatResults() 函数。唯一不同的是，它们将传递不同的参数以代表不同的记录子集。两个按钮风格如出一辙，但 VALUE 的值使它们看起来有所区别。下面是第 155～156 行“上页”按钮的开始部分：

```
docObj.writeln('<INPUT TYPE=BUTTON VALUE="Prev ' + offset + ' Results" ' +
```

现在来看第 164～165 行的“下页”按钮代码：

```
docObj.writeln('<INPUT TYPE=BUTTON VALUE="Next ' + trueTop + ' Result' +
  howMany
```

两行代码都包括表单按钮的 TYPE 和 VALUE 属性，并加上一个数字来表示前面或后面有多少条记录。由于前边所显示的记录数（*offset*）总是一定的，“上页”按钮的 VALUE 值就用这个数，比如“前 10 个结果”。但后面所能显示的记录数是可变的。

它要么是*offset*，要么是小于*offset*的最后剩余子集里的记录数。不管是多少，*trueTop*变量将设为这个值来完成这个工作。

我们注意到“上页”按钮的value总是包括“Results”这个词，这是有意义的。在应用程序的执行中，*showMatches*一直保持不变。这样，“上页”按钮的值就是，并且一直是10。这样，用户就总是能浏览前10条记录。但“下页”按钮可就不是这么回事了。设想如果最后一个子集只有一条记录。用户不应该看到标着“Next 1 Results”的按钮，这是不合乎语法的。为了消除这种状况，preNextResults()用了一个局部变量*howMany*，又一次用到了三元算子。

你可以在第163行看见它：

```
var howMany = (trueTop > 1 ? "s" : "");
```

若*trueTop*大于1，*howMany*就设为“s”字符串；若*trueTop*等于，*howMany*就设为空串。就像你在第165行所看到的那样，*howMany*紧接在“Result”后面显示。如果于集中只有一个记录，“Result”保持不变。如果有更多的记录，用户看见的就会是“Results”。

两个按钮的最后一步是告诉它们在点击时应该干什么。我在前边已经提到过，两个按钮的*onClick*事件都调用formatResults()。第157～158行，以及第166～167行清楚地写出了在处理按钮*onClick*事件中对formatResults()的调用。这是第一个设置（document.writeln()的后半部分完成调用）。

```
'onClick="' + parent.frames[0].formatResults(parent.frames[0].copyArray, ' +
 (reference - offset) + ', ' + offset + ')">');
```

三元算子帮助确定了参数，轻轻松松地完成。注意，这三个传递来的参数（一旦产生JavaScript代码）是*copyArray*，*reference-offset*和*offset*。“上页”按钮将一直持有这三个参数。顺便说一句，注意一下formatResults()和*copyArray*是怎样写下来的：

```
parent.frames[0].formatResults(...);
```

以及：

```
parent.frames[0].copyArray
```

开始它可能看起来有点奇怪，但是记住对 formatResults() 的调用不是来自 *nav.html* (parent.frames[0])。它发生在结果框架parent.frames[1]，那里没有叫 formatResults()的函数，也没有叫 *copyArray* 的变量。因此，函数和变量需要这个引用。

“下页”按钮在事件处理方法 *onClick* 中有相似的调用，但是请先等等。我们难道不必处理 *copyArray* 中所剩子集的记录数少于 *offset* 的可能情况，就像我们显示当前记录时在formatResults()中所做的一样？的确不用。formatResults()函数会很好地控制进程，我们要做的一切只是把参数加到 *offset* 上，然后传递进去。再来看看第 166～167 行 document.writeln()后半部分的调用方法：

```
'onClick="parent.frames[0].formatResults(parent.frames[0].copyArray, ' +
(reference + offset) + ', ' + offset + ')">');
```

JavaScript 技巧：三元算子

看完以上的章节之后，你肯定已经注意到了接下来的这个内容。三元算子是非常有用的，因此我必须在这里说说它。三元算子需要三个操作数，整个应用程序到处都要用到它们，比如单行的if-else语句。下面是直接来自Netscape公司的《JavaScript Guide for Communicator 4.0》第九章的语法：

```
(condition) ? val1 : val2
```

这个条件操作符，在恰当的安排下，将在 *condition* 的值为 true 时运行val1部分，否则运行val2的语句。我之所以特别重视它，是因为在很多情况下我发现它使代码变得很少，而且非常易读。如果你在几个嵌套语句内部写代码，这种操作尤其有用。

三元算子并不是万能的。假如你在条件为真或者为假的情况下有很多事情要做，那就用 if-else 好了。不然，你就在代码里试试吧。

HTML

nav.html 有少量的静态 HTML 语句。我们再一次在下面给出，从第 174 行开始：

```
</HEAD>
<BODY BGCOLOR="WHITE">
```

```
<TABLE WIDTH="95%" BORDER="0" ALIGN="CENTER">
<TR>
<TD VALIGN=MIDDLE>
  <FONT FACE="Arial">
  <B>Client-Side Search Engine</B>
  </TD>

  <TD VALIGN=ABSMIDDLE>
  <FORM NAME="search"
    onsubmit="validate(document.forms[0].query.value); return false;">
  <INPUT TYPE=TEXT NAME="query" SIZE="33">
  <INPUT TYPE=HIDDEN NAME="standin" VALUE="">
  </FORM>
  </TD>

  <TD VALIGN=ABSMIDDLE>
  <FONT FACE="Arial">
  <B><A HREF="main.html" TARGET="main">Help</A></B>
  </TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

这实在没什么稀奇的。你有一个嵌入表格的表单。提交表单就会执行我们已经涉及过的代码。你可能遇到的唯一问题是：如果没有按钮，你怎样提交表单？就像HTML 2.0里所说的，大部分的浏览器（包括Navigator和IE）以单一文本域表单作为提交表单的形式。

没有什么规定让你必须这么做。你可以随便添加按钮或图片，让页面更加生动活泼。

创建自己的 JavaScript 数据库

最后，你可以用自己的记录来代替我所给出的。你可以通过三个简单的步骤来实现：

1. 在你的文本编辑器里打开 *records.js* 文件。
2. 删除原来存在的记录，这样，文件就成了这样：

```
var profiles = new Array(
        );
```

3. 用下面的语法添加你要的每一个记录：

```
"Your_Page_Title|Your_Page_Description|http://your_page_url/file_name.html",
```

在小括号中间添加所有这些你想要加的元素。要确定在除最后一个之外的其他记录后面添上逗号。也要注意网页标题、描述和URL都必须用“|”符号划分开。不要在你的标题、描述和URL内部用任何这些符号，那将导致JavaScript错误。记住，如果你想在内部使用双引号的话，一定要用反斜杠符号来转义（举例来说，是用\"来代替"）。

应用程序扩展

搜索引擎的确是非常有用的。更妙的一点是可以进行一些有用的改进或者变化。下面是可能的一些项目：

- 使其与JavaScript 1.0兼容
- 增加稳固性
- 显示广告标幅
- 添加精确搜索性能
- 发展串设置

JavaScript 1.0的兼容性

我们都很清楚，两个最主要的浏览器现在都达到了4.x或5.x以上的版本。两个功能都很不错。但此外仍有许多用户在IE 3.02或Navigator 2.x上挣扎。HotSyte——JavaScript资源网站（http://www.serve.com/hotsyte/）的访问者中这类用户的数量仍然让我吃惊不小。

由于搜索引擎可以说是一个网站的核心，你可以考虑将应用程序向JavaScript 1.0转化。幸运的是，你所要做的所有工作就是先逐行查看代码，找出JavaScript 1.0所不支持的语句，然后把它们全部改掉。

好吧，我已经做了。但是我希望你自己再来一次。事实上，你会在*/ch01/js1.0/*里发现更改后的版本。就像你原来所做的那样，在浏览器中打开*index.html*文件。在这个部分，我们将简单地看看在JavaScript 1.0中是什么在运行应用程序。有三个要修改的地方：

- 没有JavaScript源文件（浏览器所给出）。
- 没有数组种类（及sort()方法）。
- 一个split()工作区

Navigator 2.x和IE 3.x不支持*.js*源文件（注1）。为此设立的工作区（workaround）在*nav.html*中嵌入profiles数组。第二个变化是在第90行取消对resultSet.sort()的调用。这意味着记录结果不会按照字典顺序排序，但可以顺便在*profiles*中按时间排序。最后一个转变是取消split()，JavaScript 1.0不支持它。工作区会安排这个任务，但执行的性能会降低。

TANSTAAFL

这是在佛罗里达州立大学念一年级时，我的经济学教授在黑板上所写的东西。它是"Thar' Ain't No Such Thang As A Free Lunch（没有不要钱的午餐）"的缩写。这些转变将兼容较老版本的浏览器，但是要以功能性和代码管理为代价。

没有对*.js*文件的支持，你不得不把profiles数组放在*nav.html*中。这样，如果想在其他搜索里用到这些记录，就会使可读性变得很差，而且不易管理。

sort()虽然不是操作的决定因素，但它确是一个很突出的特点。人们可能不得不浏览所有的匹配记录，因为这些记录的排列是没有特定顺序的。当然，你可以把结果按字母顺序放在数组里，但它也不是一个好办法。也许可以为JavaScript 1.0编写你自己的sort函数。不管怎样，可以说split()是一个麻烦。应用程序的JavaScript 1.0版本有一个工作区，因此它可以不成问题。

增加稳固性

你可以把"|"作为搜索字符串的一部分进行传递。为什么不增加一个功能来删除搜索字符串中作为分隔符的任何字符呢？这将增加应用程序的稳固性。

注1： 实际上并不完全如此，IE 3.02的早期版本就支持。

显示广告标幅（banner）

如果你的站点有很大的访问量，何不利用它来弄点儿钱呢？怎么做？试试这个。假设你要随便显示五个广告标幅（在此没有特定顺序）。如果在数组里有几个广告图片的URL，你可以随便选出一个来用。数组如下：

```
var adImages = new Array("pcAd.gif", "modemAd.gif", "webDevAd.gif");
```

这样你就可以随意在结果页面上这样显示：

```
document.writeln('<IMG SRC=' + ads[Math.floor(Math.random(ads.length))] +
 '>');
```

添加精确搜索功能

有了这个观念，你可以获得很多规划设计的乐趣。例如，设想用户可以选择数组元素来进行搜索。这样，用户就能精简搜索结果。

考虑在*nav.html*中文本域的下面显示一系列复选框。可能像这样：

```
<INPUT TYPE=CHECKBOX NAME="group" VALUE="97">1997 Records<BR>
<INPUT TYPE=CHECKBOX NAME="group" VALUE="98">1998 Records<BR>
<INPUT TYPE=CHECKBOX NAME="group" VALUE="99">1999 Records<BR>
```

利用这个复选框来确定搜索97类，98类或99类的哪一个。

你可以增加许多东西来提高用户精确搜索的能力。对于用户给出的区分大小写和不区分大小写的搜索，按照目前的状况，不会有什么问题，但是你可以通过加一个允许两种形式的选择来改变。你也可以把布尔搜索从*AND*和*OR*搜索推广到*AND*，*OR*，*NOT*，*ONLY*，甚至*LIKE*，以提高搜索的精确性。下面是普通定义的细则：

AND

记录必须包含AND左边和右边的条目。

OR

记录只需包括OR左右两个条目之一。

NOT

记录不能包括NOT右边的条目。

ONLY

记录必须只包含且仅只包含这个条目。

LIKE

记录应包含拼写相似或读音相似的条目。

这要费点力气（特别是对 *LIKE* 来说），但用户会对你魔术一样的技巧感到吃惊。

簇设置

另一个流行而且行之有效的技巧是建立簇设置（cluster set）。簇设置是预先确定的一组词，它能自动返回预先确定的结果。比如，如果用户在搜索字符串的某处包含了“公共资金”条目，那么描述你们公司金融产品的记录结果就会自动产生。实现这项功能要编制更多的东西，但它在一个搜索应用程序中有很大的特色。

第二章

在线测试

对于几乎所有在网上执行的多项选择测试来说，这个在线测试应用程序可以说是一个样本。在下面所列举出的几个方面，它可以有很强的适应性：

- 确定用户所回答的问题数目。
- 每一次载入应用程序或测试重新开始时，问题和答案都是随机杂乱放置的，这实际上保证了每一个用户所作的测试都具有唯一性。
- 可以在问题设置的任何地方添加或者删减问题；应用程序可以调节打乱操作、管理、评定和分级。
- 可以很简单地从应用程序中删除答案以防止作弊行为，然后将用户的答案送到和你所选等级相对应的服务器端应用程序。

应用程序要点

- 交互式的多项选择测试
- 数千种单一问题的各项测试
- 综合结果的排序和打印
- 网上调查和其他信息收集的出色应用
- 对答错题目的上下文有关说明

JavaScript技巧

- 获取SRC属性
- 打乱和排序操作
- javascript：协议

可以在浏览器中打开*ch02/index.html*载入应用程序。图2-1展示了打开的界面。现在，有谁已经猜到这里的测试问题是关于JavaScript的？试着做一做吧。这是一个有50道题的测试，这些题目对大多数朋友来说都很富有挑战性。题目覆盖了JavaScript的许多内容：JavaScript核心，客户端和服务器端JavaScript，热点链接，已知bug，等等。这可不是简单问题，但非常有意思。我有能解决所有这些问题和答案的文件。但如果你发现有一个或多个题目有什么问题，请给我发Email。

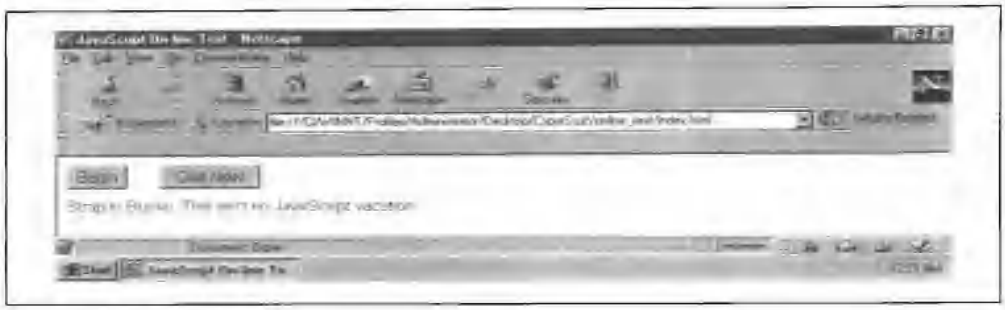


图 2-1 准备好了吗

开始做测试以后，会发现每个问题有四个可选答案。而一旦给出你的选择，应用程序就自动进入下面一题。你不能回过头去更改你的答案，每个问题都只能做一次。图 2-2 给出了这种问－答形式。

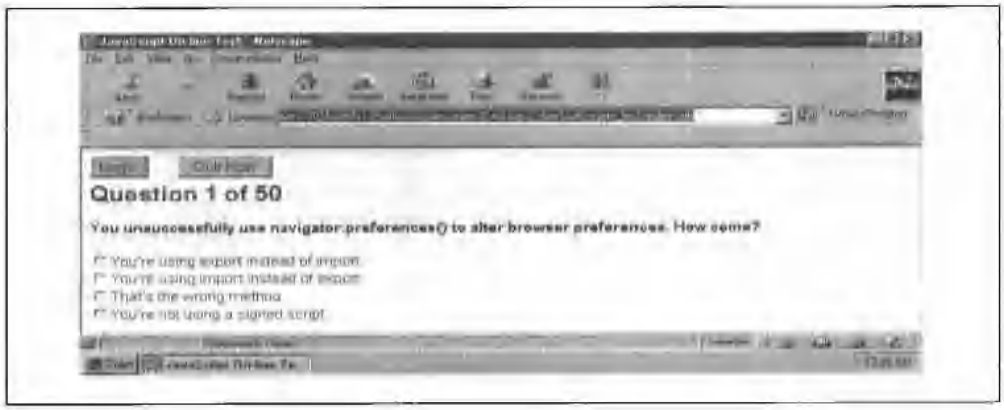


图 2-2 多项选择题

等你做完了最后一道题目的时候，你的选择将会和正确答案相比较，你的成绩、等级还有结果就显示了出来。图 2-3 给出了结果。注意每一个问题和伴随的四个选择，以及你所选的答案都一道写在屏幕上。如果你的答案是正确的，文本就呈绿色，否则为红色。

为了更好地理解回答错误的题目，用户可以把鼠标箭头移到错题的红色文本上，就可以看到题目选项的有关说明。注意，图 2-4 中的说明文字出现在屏幕的右上角。

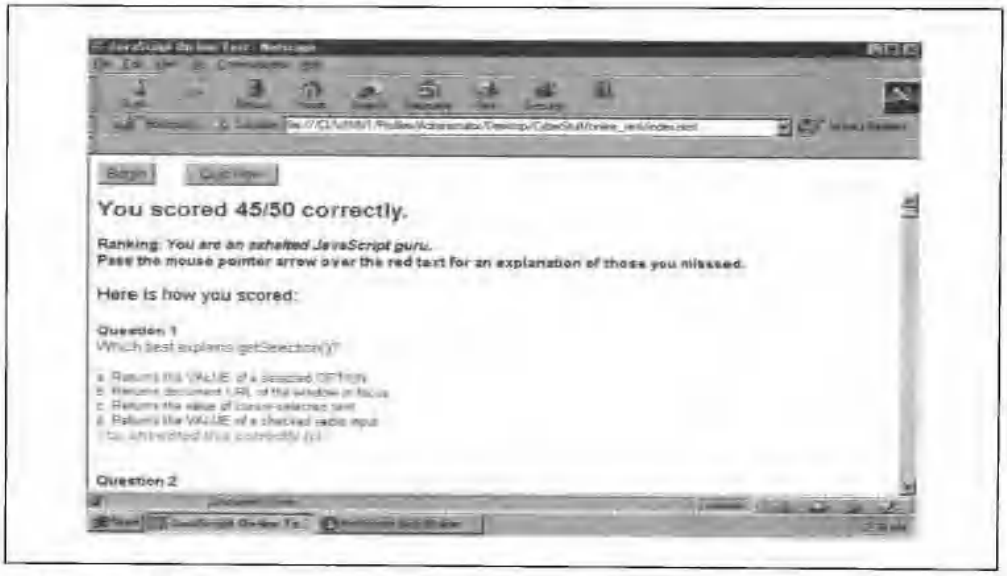


图 2-3 测试结果

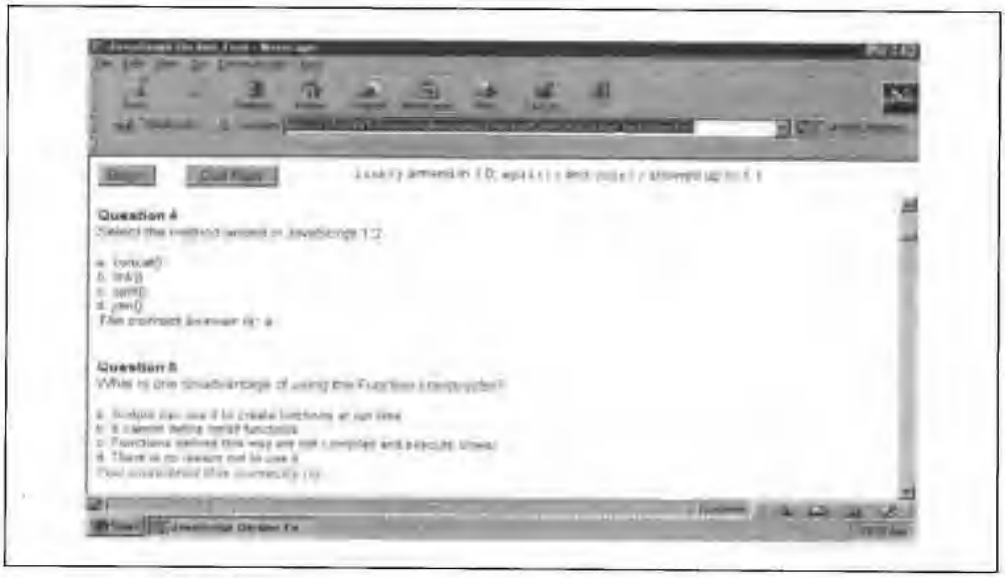


图 2-4 错题的有关说明

好的，这是应用程序的第一部分。其实，它看起来相当简明，不过图 2-5 从用户的角度给出了一个更好的程序流程。虚线代表可选的用户动作或者等待状态。我们来查看一下下面的五个步骤。

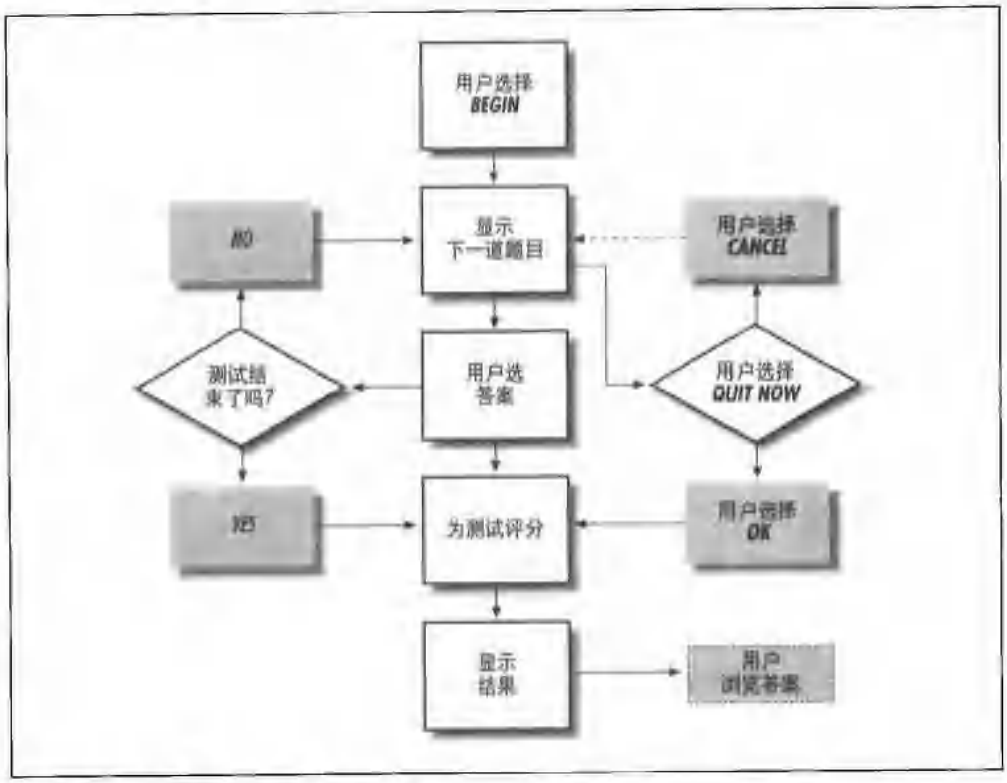


图 2-5　用户角度的应用程序流程图

过程如下所示：

1. 用户选择“开始（BEGIN）”按钮。这个动作将显示第一个问题，并等待用户做出回答或者选择“退出（QUITNOW）”按钮。
2. 如果用户选了一个答案，应用程序就记录这个选择，然后判断测试是否结束，或者应该给出下一道题目。如果测试完成（用户回答完最后一个问题），就跳到第 4 步（为测试评分）。否则，打印下一道题。
3. 但是，如果用户选择了“退出”按钮，将给出是否确定要退出的提示。假如用户选“确定（OK）”，就到第 4 步开始评分（虽然为时过早）。若用户选择“取消（CANCEL）”，则测试继续进行。
4. 测试结束（或者夭折）时，把用户的选择与正确答案作比较，并在屏幕上打印结果。

5. 用户浏览显示结果时，可以将鼠标箭头移到红色的文本（表示回答错误的问题）上，以获得关于题目内容的更多信息。

执行条件

这仓是JavaScript 1.1的东西，因此Navigator 3.x及以上版本和IE 4.x及以上的版本会比较适用。应用程序在容量上一般有75道试题。我在400道题目后就停止试验了。因为没有人会用这么大的题量来安排一场测试或者SAT（Scholastic Assessment Test），我想400道就已经足够了。

语法细则

图2-5给出了一个流程图，表示用户从头至尾是如何使用这个应用程序的。理解这个情况的一种较好方法是从处理JavaScript流程的一个较全面的流程图开始，然后在那里检查文件和相关代码。

图2-6说明了JavaScript流程。虚线框表示测试前后所发生的动作（比如在页面载入期间）。虚线箭头表示用户的可选动作或者是一个等待状态的返回。各过程的相关函数用斜体字列出。

相关函数是用斜体表示的。比较图2-5和图2-6，你很快就明白了。它们基本上是相同的流程，只是在用户做测试的前后有一点不同。

index.html —— 框架设定

应用程序有三个文件：*index.html*, *administer.html*和*questions.js*。由于*index.html*是框架设定，我们就从它开始好了。下面首先出现的是例2-1。

例2-1：index.html源代码

```
1 <HTML>
2 <HEAD>
3 <TITLE>JavaScript On-line Test</TITLE>
4 <SCRIPT LANGUAGE="JavaScript1.1">
5 <!--
6 var dummy1 = '<HTML><BODY BGCOLOR=WHITE></BODY></HTML>';
```

```
 7 var dummy2 = '<HTML><BODY BGCOLOR=WHITE><FONT FACE=Arial>' +
 8   'Strap in Bucko: This ain\'t no JavaScript vacation...</BODY></HTML>';
 9 //-->
10 </SCRIPT>
11 </HEAD>
12 <FRAMESET ROWS="90,*" FRAMEBORDER=0 BORDER=0>
13   <FRAMESET COLS="250,*">
14     <FRAME SRC="administer.html" SCROLLING=NO>
15     <FRAME SRC="javascript: self.dummy1">
16   </FRAMESET>
17   <FRAME NAME="questions" SRC="javascript: self.dummy2">
18 </FRAMESET>
19 </HTML>
```

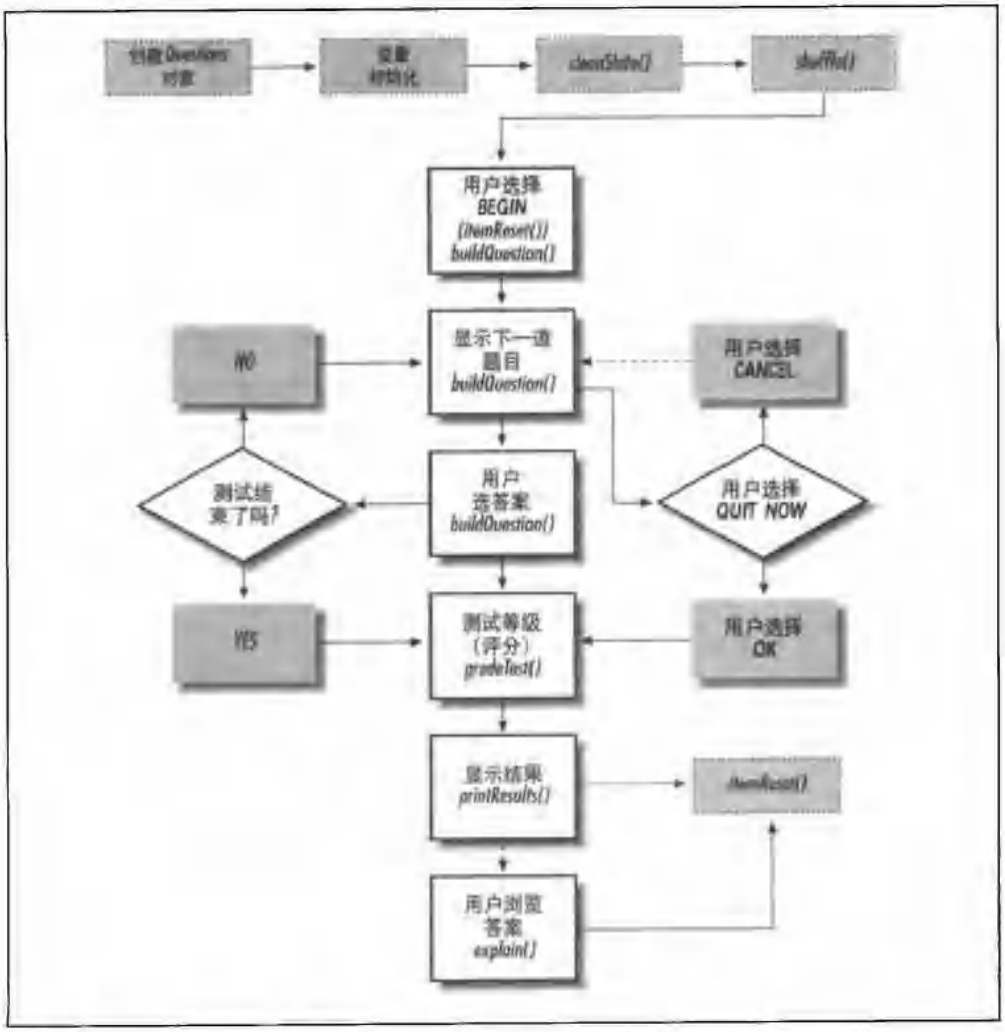


图 2-6 JavaScript 流程

你可能已经注意到了，网页框架设置可不是均分的。首先，它是嵌套的。也就是说，一个框架中包含另一个框架。第12行的外部框架定义了两行 —— 第一行的高为90像素，而第二个占据窗口可用空间的剩余部分。

90像素高的框架内部也包含了一个框架设置，它定义了两个列 —— 第一列有250像素宽，第二个占据剩余的可利用窗口空间。图2-7显示了父窗口的框架划分。也列出了各框架的SRC属性。

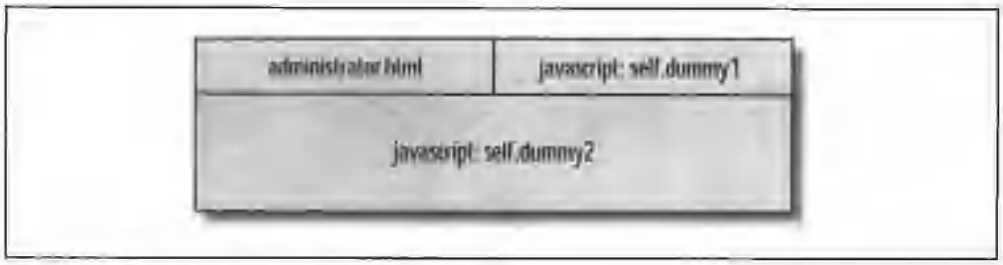


图2-7 index.html中框架设置的嵌套设计

FRAME标签下的*administer.html*作为一个SRC是有意义的，但是另外两个又如何呢？两个虚拟变量潜在定义了HTML页。这意味*dummy1*和*dummy2*不用文件名来指示HTML页。它们都只在应用程序作用域内存在。虚拟变量在第7和第8行得到定义。注意它们每一个都包含有非常少的HTML代码，不多，但很有用。*index.html*利用javascript:协议来求得*dummy1*和*dummy2*中表达式的值，并将其返回，作为SRC属性的URL的内容。在“JavaScript技巧：获取SRC属性”部分可以获得更多的信息。

框架设置现在就完成了。你仅用一个HTML页（*administer.html*）填满了三个框架。这难道不算节约吗？

questions.js —— JavaScript源文件

我们再继续看看如例2-2所示的*questions.js*，它是在*administer.html*中调用的JavaScript源文件。

例 2-2：questions.js 源代码的开始部分

```
 1 function question(answer, support, question, a, b, c, d) {
 2  this.answer = answer;
 3  this.support = support;
 4  this.question = question;
 5  this.a = a;
 6  this.b = b;
 7  this.c = c;
 8  this.d = d;
 9  return this;
10  }
11 var units = new Array(
12  new question("a", "The others are external objects.",
13  "Choose the built-in JavaScript object:",   "Image", "mimeType",
14  "Password", "Area"),
15 // and so on ...
16  }
```

这当然算是一个短小的文件。units数组非常长（总共有75个元素），但这表示units的每一个元素都是在第1～10行的question()函数中定义的题目对象。

应用程序以用户定义的JavaScript对象为基础（你和我可以声明的对象）。如果你对JavaScript对象的概念不清楚，可以查看：http://developer.netscape.com/docs/manuals/communicator/jsguide4/model.htm地址下Netscape的文件。它可以帮助你更好地理解JavaScript对象模式。不过，你也可以将下面的几个段落用作一个应急教程。

对象就是一系列的结构化数据。每个对象可以拥有或者和两种类型的实体相关联——属性和函数。属性包含某些东西，比如数字6，表达式a*b，或者字符串"Jimmy"。函数做某些工作，比如计算圆周或者改变文档背景的颜色。我们来考虑JavaScript文档对象。每个文档包含一些东西（*document.bgColor*, *document.fgColor*及其他），还做一些事情（*document.open()*, *document.write()*, *document.close()*）。属性包含值，而函数做操作，就是这样。

可以通过先创建一个构造函数来创建对象，就像这样：

```
function myFirstConstructor(arg1, arg2, argn) {
  this.property1 = arg1;
  this.property2 = arg2;
  this.propertyn = argn;
  return this;
  }
```

JavaScript技巧：获取SRC属性

将SRC属性设为JavaScript值看起来可能有点莫名其妙。让我们回过头来看看。假设你打开文本编辑器，把下面的代码拷贝到一个新建的空白文档中：

```
<BODY BGCOLOR=WHITE>
<FONT FACE=Arial>
Better strap in, Bucko. This ain't no JavaScript vacation...
</FONT>
</BODY>
```

然后你将此文件命名为*bucko.html*，载入到浏览器中。结果毫无疑问是你可以预先想到的。在*index.html*中基本上发生同样的情况，除了变量*dummy2*被设置为和上面相同的文本，用javascript:协议来求*dummy2*的值。第20行中SRC属性设为javascript:协议所求出的值，也就是*dummy2*的值。有关javascript:协议的更多信息，可以参见本章后面"JavaScript技巧：javascript:协议"部分。

你所得到的是一个匿名的HTML页。我把这个技术叫做获取SRC。在本章后面的部分你会得到有关于这一点的更多说明。

这和所有你可能会创建的函数看起来的确很相似，除了利用this关键字来指示它本身以外。你传递来的任何参数都可以被赋值为属性的值，或者以其他方式进行处理。一旦有了构造函数，所需要的就只是变量名和new操作符了：

```
var myFirstObject   = new myFirstConstructor(6, a*b, "Jimmy"),
var mySecondObject  = new myFirstConstructor(6, a*b, "Jimmy"),
var myThirdObject   = new myFirstConstructor(6, a*b, "Jimmy"),
```

对于我们的脚本来说，对象的执行是如此简单。我们创建的对象，例如question()构造函数，它只包含某些属性。第2～8行确定了每个question()的七个属性：一个答案，一个解释，一个题目（文本），还有四个多项选择的可能选项：a、b、c和d。下面是第1～10行的内容：

```
function question(answer, support, question, a, b, c, d) {
  this.answer = answer;
  this.support = support;
  this.question = question;
  this.a = a;
  this.b = b;
  this.c = c;
```

```
this.d = d;
return this;
}
```

属性和函数通过this符号赋值到对象中。因此，*units*的每一个元素利用new操作符创建一个新的question()实例，它得到七个参数，用以赋值给其中所包含的属性。第9行使用了下面的语法：

```
return this;
```

这一行代码返回一个给示例变量的参数(在本例中也就是units的每个元素)。可以把它看成使操作正规化的环节。现在，*units*中的每个元素都是一个题目。这是对测试题目进行创建、删除和其他一些操作的简便方法。你可以轻松地通过同样的语法来添加题目作为另外一个*units*元素：

```
new question("your_answer_letter", "your_explanation",
"your_question_text", "option_a", "option_b", "option_c", "option_d");
```

我把答案作为函数的第一个参数，可能会使你感到惊讶，这是因为让单字符的字符串代替后面其他参数位于参数列的前边，会使问题变得简单些。有些题目实在是太长了。这只是为了更容易地查找和修改它。

为每一个题目创建一个题目对象看来似乎不可行，但它在处理每一个题目对象的属性中的数据时使问题变得相当简单。等我们讨论*administer.html*时会涉及到这个问题。

注意：如果不在应用程序中使用JavaScript对象，可以考虑改变类型。对象有着相当大的优越性。它们会使代码变得规范而易于管理。对象还有优良的继承性，属性可以从父对象传递到从父对象创建的另一个对象中。你可以下载一个PDF文件或者读一读http://developer.netscape.com:80/docs/manuals/communicator/jsobj/contents.htm地址中的JavaScript文件和对象继承在线文件。

administer.html

对象现在已经建立起来了。要让它们开始为我们工作。这是JavaScript的关键部分，位于上部框架的另外一个应用程序，用下部的框架来完成交互工作。你可以把应用

程序分解为一系列的过程。表2-1给出并描述了这些过程，包括每一过程的相关JavaScript变量和函数。

表2-1 测试过程和相关函数

过程	描述	相关JavaScript
环境设定	定义并初始化全局变量，将问一答设置打乱	变量qIdx, correct, howMany, stopOK, nextQ, results, aFrame, qFrame
测试管理	把每一个问一答设置写到窗口，记录每一个答案	函数buildQuestion(), makeButton(), 可能还有chickenOut()
评定测试	将学生的答案同正确答案作比较	函数gradeTest ()
打印结果	按某个顺序在窗口中打印所有的答案，不管正确与否	函数printResults()
显示说明	为parent.frame[1]打印或清楚说明文字	函数explain()和show()
重置环境	将所有必要的变量设置为它们的初始值	变量qIdx, correct, stopOK, 数组keeper 函数cleanSlate(), shuffle()

我们马上来看一看每个细节。现在，例2-3给出了*administer.html*的代码。

例2-3：The administer.html源代码

```
<HTML>
<HEAD><TITLE>On-line JavaScript Test</TITLE>
<SCRIPT LANGUAGE="JavaScript1.1" SRC="questions.js"></SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.1">
var qIdx      = 0;
var correct   = 0;
var howMany   = 50;
var keeper    = new Array();
var rank      = new Array('No offense, but you need help.',
  'Ummm... Well... Few have done worse.',
  'Ehhh... You know some. Keep at it.',
  'You seem to have a working knowledge.',
  'Better than the average bear.','You are an adequate JavaScripter.',
  'You are a formidable JavaScripter.',
  'You are an excellent JavaScripter.',
  'You are an exhalted JavaScript guru.'
  );
```

```
var stopOK = false;
var nextQ    = '';
var results   = '';
var aFrame = parent.frames[1];
var qFrame = parent.frames[2];
function shuffle() {
  for (var i = 0; i < units.length; i++) {
    var j = Math.floor(Math.random() * units.length);
    var tempUnit = units[i];
    units[i] = units[j];
    units[j] = tempUnit;
    }
  }
function itemReset() {
  qIdx   = 0;
  correct   = 0;
  stopOK   = false;
  keeper   = new Array();
  shuffle();
  }
function buildQuestion() {
  if (qIdx == howMany) {
    gradeTest();
    return;
    }
  nextQ = '<HTML><BODY BGCOLOR=WHITE><FONT FACE=Arial>' +
  '<H2>Question ' + (qIdx + 1) + ' of ' + howMany + '</H2>' +
  '<FORM>' + '<B>' + units[qIdx].question + '</B><BR><BR>' +
  makeButton("a", units[qIdx].a) +
  makeButton("b", units[qIdx].b) +
  makeButton("c", units[qIdx].c) +
  makeButton("d", units[qIdx].d) +
  '</FORM></BODY></HTML>';
  qFrame.location.replace("javascript: parent.frames[0].nextQ");
  qIdx++;
  if(qIdx >= 2 && !stopOK) { stopOK = true; }
  }
function makeButton(optLtr, optAnswer) {
  return   '<INPUT TYPE=RADIO NAME="answer" VALUE="' + optLtr +
    '" onClick="parent.frames[0].keeper[parent.frames[0].qIdx - 1] =
    this.value; parent.frames[0].buildQuestion()">' + optAnswer + '<BR>';
  }
function chickenOut() {
  if(stopOK && confirm('Stopping early? Are you really a ' +
    'JavaScript Chicken?')) {
    gradeTest();
    }
  }
function gradeTest() {
  for (var i = 0; i < qIdx; i++) {
    if (keeper[i] == units[i].answer) {
      correct++;
      }
```

```
71      }
72    var idx = Math.ceil((correct/howMany) * rank.length - 1) < 0 ? 0 :
73      Math.ceil((correct/howMany) * rank.length - 1);
74    printResults(rank[idx]);
75    itemReset();
76    }
77  function printResults(ranking) {
78    results = '<HTML><BODY BGCOLOR=WHITE LINK=RED VLINK=RED ALINK=RED>' +
79      '<FONT FACE=Arial>' +
80      '<H2>You scored ' + correct + '/' + howMany + ' correctly.</H2>' +
81      '<B>Ranking: <I>' + ranking +
82      '</I><BR>Pass the mouse over the red text for an explanation of ' +
83      'those you misssed.</B>' +
84    '<BR><BR><FONT SIZE=4>Here is how you scored: </FONT><BR><BR>';
85    for (var i = 0; i < howMany; i++) {
86      results += '\n\r\n\r\n\r<B>Question ' + (i + 1) + '</B><BR>' +
87        units[i].question + '<BR><BR>\n\r<FONT SIZE= 1>' +
88        'a. ' + units[i].a + '<BR>' +
89        'b. ' + units[i].b + '<BR>' +
90        'c. ' + units[i].c + '<BR>' +
91        'd. ' + units[i].d + '<BR></FONT>';
92      if (keeper[i] == units[i].answer) {
93        results += '<B><I><FONT COLOR=GREEN>' +
94          'You answered this correctly (' + keeper[i] + '). ' +
95          '</FONT></I></B>\n\r<BR><BR><BR>';
96        }
97      else {
98        results += '<FONT FACE=Arial><B><I>' + '
99          '<A HREF=" " onMouseOver="parent.frames[0].show();' +
100         parent.frames[0].explain(\'' + units[i].support + '\'); ' +
101         'return true" onMouseOut="parent.frames[0].explain(\' \');"' +
102         'onClick="return false;">' +
103         'The correct answer is: ' + units[i].answer +
104         '</A></FONT></I></B>\n\r<BR><BR><BR>';
105       }
106     }
107   results += '\n\r</BODY></HTML>';
108   qFrame.location.replace("javascript: parent.frames[0].results");
109   }
110 function show() { parent.status = ''; }
111 function explain(str) {
112   with (aFrame.document) {
113     open();
114     writeln('<HTML><BODY BGCOLOR=WHITE><FONT FACE=Arial>' + str +
115       '</FONT></BODY></HTML>');
116     close();
117     }
118   }
119 function cleanSlate() {
120   aFrame.location.replace('javascript: parent.dummy1');
121   qFrame.location.replace('javascript: parent.dummy2');
122   }
123 </SCRIPT>
```

```
124 </HEAD>
125 <BODY BGCOLOR=WHITE onLoad="cleanSlate();">
126 <FONT FACE="Arial">
127   <FORM>
128   <INPUT TYPE=BUTTON VALUE="Begin"
129     onclick="itemReset(); buildQuestion();">
130        
131   <INPUT TYPE=BUTTON VALUE="Quit Now" onclick="chickenOut();">
132   </FORM>
133 </FONT>
134 </BODY>
135 </HTML>
```

这个长文件可以分为四个部分。首先调用了*questions.js*源文件。接下来是一些全局变量的定义。然后就是函数。最后，还有几行HTML语句。让我们先来看看HTML语句，因为它们非常短小，容易理解。

HTML的主体部分

*adniniser.html*载入完成后，在第125行调用cleanSlate()函数：

```
<BODY BGCOLOR=WHITE onLoad="cleanSlate();">
```

cleanSlate()利用位置对象的replace()函数来替换paren.frames[1]（框架a的别称）和paren.frames[2]（框架q的别称）的当前URL，替换为*index.html*中已经定义过的变量*dummy1,dummy2*的值。看看第119～122行：

```
function cleanSlate() {
  aFrame.location.replace('javascript: parent.dummy1');
  qFrame.location.replace('javascript: parent.dummy2');
  }
```

我们刚在*index.html*中使用过这些语句，是吗？对，我们的确用过。不管怎么说，这可以保证用户由于某种原因载入*administer.html*时，顶部右边的框架总是以一个空白页面开始，而底部框架将总以打开的"bucko"消息开始，页面过去的内容（可能是一个试题）不会再容易产生影响，也不会再发生相关错误。

剩余的HTML语句只是个带有两个按钮的表单，这部分并不长。*adminiser.html*有一个带两个按钮的表单。每一个按钮在被点击时调用不同的函数。下面是第127～132行的代码：

```
<FORM>
  <INPUT TYPE=BUTTON VALUE="Begin"
    onclick="itemReset(); buildQuestion();">
       
  <INPUT TYPE=BUTTON VALUE="Quit Now" onclick="chickenOut();">
</FORM>
```

注意到“开始”按钮调用 itemReset()和buildQuestion()函数，“退出”按钮调用 chickenOut()。我们将在“函数”部分讨论这三个函数。

全局变量

就在第4行嵌入JavaScript源文件 *questions.js* 的代码后面，你可以看见这个应用程序中所用的全局变量。这里是第5～22行的内容：

```
var qIdx     = 0;
var correct   = 0;
var howMany  = 50;
var keeper   = new Array();
var rank     = new Array('No offense, but you need help.',
    'Umm... Well... Few have done worse.',
    'Ehhh... You know some. Keep at it.',
    'You seem to have a working knowledge.',
    'Better than the average bear.',
    'You are an adequate JavaScripter.',
    'You are a formidable JavaScripter.',
    'You are an excellent JavaScripter.',
    'You are an exhalted JavaScript guru.'
    );
var stopOK = false;
var nextQ    = '';
var results   = '';
var aFrame = parent.frames[1];
var qFrame = parent.frames[2];
```

下面的列表描述了每一个全局变量的作用。在关于功能的讨论中你可以更好地认识它们。

qIdx

用来监控显示在屏幕上的试题的变量。

correct

用来在评定测试成绩时追踪正确答案数目的变量。

howMany

用来决定用户要回答的问题数目的静态量（来自 *units* 数组中的可用数字）。

keeper

用来保存用户选择答案数组，初始值为空。

rank

指出各种成绩水平的字符串数组。

stopOK

包含一个布尔值的变量，用来指示是否允许提早结束测试。

nextQ

一个被反复设置为包含某个测试题目的 HTML 语句的空串。

results

一个后来被设置为表示测试结果的 HTML 语句的空串。

aFrame

第二个框架的一个简单引用。

qFrame

第三个框架的一个简单引用。

函数

下面我们开始函数的讨论。先从 `itemReset()` 开始。

itemReset()

应用程序中第一个被调用的函数是 `itemReset()`。它在用户点击“开始”按钮时发生（第 128～129 行）：

```
<INPUT TYPE=BUTTON VALUE="Begin"
onclick="itemReset(); buildQuestion();">
```

`itemReset()` 将全局变量设置为它们的初始值并且将 question 对象数组的元素打乱（马上将会讨论有关于这一点的更多内容）。看一看第 31～37 行：

```
function itemReset() {
  qIdx     = 0;
  correct  = 0;
  stopOK   = false;
  keeper   = new Array();
  shuffle();
  }
```

注意一个情况，用户甚至还没有看见第一道题，而JavaScript已经着手于重置全局变量了。为什么？噢，假设你已经完成了一次测试，但只做对了两道题。你可能会选择“开始”按钮来重做测试题。但是在第一次测试以后，许多全局变量已经不再保持它们的初始值了。ItemReset()会处理这个情况，给你新的值。

注意，这里并没有包括*howMany*变量。它的值在应用程序运行期间保持不变。变量*nextQ*和*results*最初从空串开始，但它们的值没有被重置，因为这是没有必要的。分别看看第43行和第86行，你会看见这些变量在运行中被设置。

随着变量的恰当设置，第36行的shuffle()调用又怎么样呢？

shuffle()

这个小小的函数给了测试管理员很大的弹性。这个函数将题目随机排序，差不多可以保证每一次测试的唯一性。给你一个关于可能组合的计算方法，考虑测试题目的可能组合数目（不同的顺序）是n(n-1)，n是题目的数目。这样一个10道题的测试就有10（10-1），也就是90种可能的组合方式。一个20道题的测试有380个组合。50道题有2450这么多的组合方式。这对于想作弊的家伙来说，简直是一场噩梦。

测试能保持唯一性，还因为即使*units*数组中有75个可用题目，变量*howMany*也只设为50。打乱以后，测试只使用前面50道题目。由于本来有75个题，用户就非常可能在不同的测试中得到不同的50道题目。因此，测试可以给你测试题目的数千种组合。令你感到惊奇的是，这种打乱的方法却相当简单。

这里是第23～30行的代码：

```
function shuffle() {
  for (var i = 0; i < units.length; i++) {
    var j = Math.floor(Math.random() * units.length);
    var tempUnit = units[i];
    units[i] = units[j];
    units[j] = tempUnit;
```

```
    }
  }
```

对于units数组中的每个元素：

1. 选择从0到units.lengh-1之间的一个随机整数。
2. 将局部变量*tempUnit*的值设为当前所指示的元素（units[i]）。
3. 将当前所指示的元素的值设为随机整数所指示元素（units[j]）的值。
4. 将随机整数指示的元素值设为局部变量*tempUnit*的值。

换句话说，系统地迭代数组的所有元素，将每个值同其他某个随机选出的元素值交换，然后将这个随机选出的元素值设为当前处理的元素值。

这样，题目按随机方式被打乱，现在就等着用户了。

buildQuestion( )

这个函数相当于测试的管理员。你可能在最后一个流程图中已经注意到了，buildQuestion()在好几个地方出现。它有很多作用。下面是从第38～54行的代码：

```
function buildQuestion() {
  if (qIdx == howMany) {
    gradeTest();
    return;
    }
  nextQ = '<HTML><BODY BGCOLOR=WHITE><FONT FACE=Arial>' +
  '<H2>Question ' + (qIdx + 1) + ' of ' + howMany + '</H2>' +
  '<FORM>' + '<B>' + units[qIdx].question + '</B><BR><BR>' +
  makeButton("a", units[qIdx].a) +
  makeButton("b", units[qIdx].b) +
  makeButton("c", units[qIdx].c) +
  makeButton("d", units[qIdx].d) +
  '</FORM></BODY></HTML>';
  qFrame.location.replace("javascript: parent.frames[0].nextQ");
  qIdx++;
  if(qIdx >= 2 && !stopOK) { stopOK = true; }
  }
```

在最开始，buildQuestion()检查变量*qIdx*的值是否等于变量*howMany*。如果是，则表示用户已经回答了最后一道题目，可以开始评分了。在第40行调用了gradeTest()函数。

JavaScript 技巧：打乱和排序操作

测试以随机顺序重排数组元素。这是应用程序所需的工作，但是，很多人为控制的打乱方法也很容易实现。下面的函数接收了数组对象的一个复本并将其打乱，其中的整数给出了你所打乱的元素个数：

```
function shakeUp(formObj, stepUp) {
  stepUp = (Math.abs(parseInt(stepUp)) > 0 ?
    Math.abs(parseInt(stepUp)) : 1);
  var nextRound = 1;
  var idx = 0;
  var tempArray = new Array();
  for (var i = 0; i < formObj.length; i++) {
    tempArray[i] = formObj[idx];
    if (idx + stepUp >= formObj.length) {
      idx = nextRound;
      nextRound++;
      }
    else {
      idx += stepUp;
      }
    }
  formObj = tempArray;
  }
```

例如，如果你的数组有10个元素，你想以2的倍数来存储（元素0，2，4，6，8，然后是1，3，5，7，9），然后调用shakeUp(yourArrayObj,2)。如果你传递0，增量默认值则为1。在第六章中你会发现更多这样的函数，构成JavaScript源文件。

如果测试尚未结束，则buildQuestion()产生表示一个完整HTML页面的串，由此进入下一道题目。这发生在第43～50行。如果你查看一下*nextQ*，你会发现将要显示的HTML页包含一个题目号码标识和测试题目的总数。这是第44行代码：

```
'<H2>Question ' + (qIdx + 1) + ' of ' + howMany + '</H2>'
```

接下来你会看见一个FORM的开始标签，后面跟着测试题目文本。试题文本，如果你没有忘记的话，它存储在每一个*units*元素的*question*属性中。这可没什么好惊奇的，现在让我们来看第45行的代码：

```
'<FORM>' + '<B>' + units[qIdx].question
```

在有FORM标签的地方，后面的表单元素不会太长。实际上，这就是建一个HTML页所剩下的工作。这个表单只有四个元素，每一个都是单选按钮。函数makeButton()将产生单选按钮，这比在函数内敲HTML代码产生按钮（几乎每个都是一样的）的方法要好多了。你只需要传递选择的字母和选择的文本就可以了，它们在第46～49行的每一个调用里已经被准备好了。下面是位于第55～59行的makeButton()函数：

```
function makeButton(optLtr, optAnswer) {
  return  '<INPUT TYPE=RADIO NAME="answer" VALUE="' + optLtr +
    '" onClick="parent.frames[0].keeper[parent.frames[0].qIdx - 1] =
    this.value; parent.frames[0].buildQuestion()">' + optAnswer + '<BR>';
    }
```

这个函数返回一个代表单选按钮的字符串，它有一个定制形式的VALUE属性值，可以等于a，b，c或d，供选择的答案显示在其后。VALUE属性来自*optLtr*，选择的文本来自*optAnswer*。

应该记住这种测试是由用户驱动的，因此，一旦用户作了一个选择它就自动进入到下一步。在JavaScript用法中，这意味着通过把表达式和每一个单选按钮中的*onClick*事件管理相连而驱动两个动作的发生。

所发生的第一件事是数组*keeper*保存与用户答案相关的字母。为了确定分配哪一个元素给用户当前选择，我们使用：

```
parent.frames[0].qIdx - 1
```

变量*qIdx*保存了当前题目的号码，因此它对将下一个元素分配给用户当前选择非常有用。

对于JavaScript，所发生的下一件事是调用buildQuestion()来打印下一道题目，或者如果测试结束的话，要对测试评分。注意，两个监控数字和buildQuestion()都从paren.frames[0]开始。由于这个信息将被写到parent.frames[1]中，我们只好从上一级框架中访问他们。

随着表单的建立，所有剩余（只要和HTML有关的）工作就是关闭表单，将所要的内容上载到窗口中。看一看第50和51行：

```
'</FORM></BODY></HTML>';
  qFrame.location.replace("javascript: parent.frames[0].nextQ");
```

这把 *nextQ* 的值载入到了底部框架中。注意，应用程序用位置对象的 replace() 来代替 location.href 实体，或者甚至代替用 document.write() 来把下一道题目打印到页面上。在这个应用程序中，这一点有着重要的区别。replace() 将指定的 URL 载入到浏览器中（在本例中，URL 是一个通过 javascript: 协议求出的 HTML 串），但是在历史记录中代替了当前页。这就使用户不能回头查看前一页的题目或者修改以前的答案。如果用户选择了“返回”按钮，浏览器就会载入以前载入到 *index.html* 中的页面。

在离开 buildQuestion() 前最后一件事是在第 52～53 行作一定的清除的工作。

```
qIdx++;
if(qIdx >= 2 && !stopOK) { stopOK = true; }
```

将 *qIdx* 加 1，准备下一次对 buildQuestion() 的调用。记得在第 39 行，若 *qIdx* 大于测试题目的数量（在 *howMany* 变量中），就应该开始对测试作评分了。第 53 行的 *if* 语句确定用户是否符合结束测试的条件。当前代码要求用户提前选择“退出”按钮时，必须回答至少一道题目才能被允许结束测试。这也许比较符合一般的想法。

gradeTest( )

gradeTest() 执行两个功能。首先，它将用户的答案同正确答案相比较，保留用户正确答案的标记。其次，gradeTest() 计算一系列指数，并基于用户的正确答案数作出响应。下面是 gradeTest() 的全部内容，它位于第 66～76 行：

```
function gradeTest() {
  for (var i = 0; i < qIdx; i++) {
    if (keeper[i] == units[i].answer) {
      correct++;
      }
    }
  var idx = Math.ceil((correct/howMany) * (rank.length - 1)) < 0 ? 0 :
    Math.ceil((correct/howMany) * (rank.length - 1));
  printResults(idx);
  itemReset();
  }
```

数组 *keeper* 包括每题中和用户所选答案相关的每个字符（a，b，c 或 d）。*Units* 数组的每一个元素是一个题目对象，它包含一个 *answer* 属性 —— 同样是 a，b，c 或 d。GradeTest() 迭代每一个 *keeper* 元素，将它的值和相应 *units* 元素的 *answer* 属性作比较。如果匹配，*correct* 变量就加 1。

JavaScript 技巧：javascript:协议

到现在为止，在本书中你已经看到了有关的很多内容，下面你可以看到的更多。javascript:协议允许JavaScript对跟在它后面的任何表达式求值。它有很多种用法。例如，除了在用户点击一个链接后将一个新页面载入到浏览器中以外，如果你还想让别的什么事发生，可以在<A>标签的HREF属性中使用它：

```
<A HREF="javascript: alert('You found the alert dialog!');">Click me</A>
```

你也可以设置其他HTML标签的SRC属性。参见本章前边的“JavaScript技巧：获取SRC属性”可以获得其具体内容。

警告：如果你在所定义的JavaScript函数中使用协议，不要试图用它来求函数中局部变量的值。它对此不起作用。javascript:协议只对全局范围有效，因此只能“看见”和访问全局变量，全局对象，诸如此类的东西。第51行是一个非常典型的例子：

```
qFrame.location.replace("javascript:parent.frames[0].nextQ");
```

可以对变量*nextQ*恰当地作局部定义。毕竟，它只在buildQuestion()中使用。但是，第55行代码由于用了javascript:协议而使下面的代码失效：

```
qFrame.location.replace("javascript: nextQ");
```

如果*nextQ*是局部变量，javascript:协议将不能对它求值。

注意，这个函数不跟踪正确答案的记录。这在此应用程序中确实是不会发生的。函数只确定正确答案的数目，传递基于这个数目的一个序列。等我们一会儿用到printResults()时，还要重新涉及到*keeper*数组。同时也要记住，gradeTest()不关心*howMany*问题。测试中有多少试题并不重要，重要的是用户回答了多少。

结果一旦传入，*correct*变量就存储用户给出的正确答案数目的值。GradeTest()只确定用户的等级，或者他的测试情况。第72和73行是有关于这一点的代码：

```
var idx = Math.ceil((correct/howMany) * (rank.length - 1)) < 0 ? 0 :
  Math.ceil((correct/howMany) * (rank.length - 1));
```

这是它的工作过程。我们要从第9行的*rank*数组元素为某个等级赋值。为了选择一

个元素，我们需要从0到rank.length-1之间的一个整数。函数gradeTest()通过三个步骤选出一个整数：

1. 计算正确答案的百分比（correct/howMany）。
2. 将百分比乘以（rank.length-1）。
3. 将乘积四舍五入为不小于它的最小整数。

这个过程所产生的结果被赋值给局部变量*idx*，它是一个和0到rank.length之间和用户成绩成比例的整数。换句话说，有多少道测试题目并不重要，用户得到的等级基本决定于他回答对了多少道题。下面的例子可以帮助你的理解。假设你已经给*rand*数组作了如下的设置：

```
var rank = new Array( "I've seen better", "So-so", "Good", "Very Good"
"Excellent");
```

rank.length为5，因此如果你的测试有50道题目，等级情况如下：

正确题数	适当整数	等级（rank[int]）
0~9	0	继续努力
10~19	1	一般
20~29	2	还不错
30~39	3	非常好
40~50	4	太出色了

每个等级大约有howMany/rank.lengh种答案（除了最高等级外，比如上表中的40~50等级）。有2个或者200个题目都没有关系，它们以同样的方法运行。

除了相当粗糙的缺点外，这种成绩体制有着行之有效的好处。成绩体制通常要复杂得多。大多数的学校差不多使用这样的成绩体制：以正确答案的百分比，90%以上的为A，80%~89%的为B，70%~79%的为C，60%~69%的为D，60%以下的为F。也许你想用某种曲线图表。在“应用扩展”一节中你可以看到一些新奇的办法。

gradeTest()基本上完成了它的使命。变量rank[*idx*]被传递给函数printResults()用于打印；然后调用itemReset()进行适当的清除。

printResults()

应用程序知道了用户测试结果的好坏；现在可以显示给用户了。函数printResults()显示以下项目完成这个任务：

- 正确答案数与试题总数的百分比
- 基于gradeTes()传递来的计算结果得出的用户等级
- 包括四个选择答案的每一道试题
- 用户的答案（如果正确）或者正确答案（用户回答错误）
- 供用户浏览关于错题附加信息的链接文本

第77～84行是前两个步骤的有关代码：

```
function printResults(ranking) {
  results = '<HTML></HEAD><BODY BGCOLOR=WHITE LINK=RED VLINK=RED
ALINK=RED>'+
    '<FONT FACE=Arial>' +
    '<H2>You scored ' + correct + '/' + howMany + ' correctly.</H2>' +
    '<B>Ranking: <I>' + ranking +
    '</I><BR>Pass the mouse over the red text for an explanation of ' +
    'those you missed.</B>' +
  '<BR><BR><FONT SIZE=4>Here is how you scored: </FONT><BR><BR>';
```

变量*correct*和*howMany*分别代表正确答案数目和试题总数，rank[rankIdx]是表示用户成绩的字符串。至于试题和四个相应选项的显示，发生在第85～91行。它理所当然位于一个*for*循环语句内部：

```
for (var i = 0; i < howMany; i++) {
  results += '<B>Question ' + (i + 1) + '</B><BR>' +
    units[i].question + '<BR><BR><FONT SIZE=-1>' +
    'a. ' + units[i].a + '<BR>' +
    'b. ' + units[i].b + '<BR>' +
    'c. ' + units[i].c + '<BR>' +
    'd. ' + units[i].d + '<BR></FONT>';
```

对于从0到howMany-1的每一次循环，得到的结果是一个字符串，它包含一个问题号码（i+1），题目文本内容（units[I].question），还有相应的四个选择答案（units[i].a,units[i].b,units[i].c,以及units[i].d）。周围的HTML代码使它们真正能够被显示。

最后一个显示方面的难题是，如果用户回答正确，就用绿色文本显示他的答案，否则用红色的链接文本显示。下面是第92～106行的代码：

```
if (keeper[i] == units[i].answer) {
      results += '<B><I><FONT COLOR=GREEN>' +
        'You answered this correctly (' + keeper[i] + '). ' +
        '</FONT></I></B>\n\r<BR><BR><BR>';
      }
    else {
      results += '<FONT FACE=Arial><B><I>' + '
        '<A HREF=" " onMouseOver="parent.frames[0].show();' +
        parent.frames[0].explain(\'' + units[i].support + '\'); ' +
        'return true" onMouseOut="parent.frames[0].explain(\' \');"' +
        'onClick="return false;">' +
        'The correct answer is: ' + units[i].answer +
        '</A></FONT></I></B>\n\r<BR><BR><BR>';
      }
    }
```

对每一个试题来说，用户要么选对，要么选错，没有模棱两可的情况。所以，我们使用if-else语句比较简便。如果keeper[i]等于units[i].answer，则用户做对了。因此用绿色文本表示用户正确，并打印他的答案（keeper[i]）。如果这两个值不等，则用红色文本显示正确答案，并提供用以浏览parent.frames[1]中试题附加信息的选择项。这个框架并没有什么重大的用途，到目前为止，它是最后一个出场。

用户回答正确的题目只以文本形式显示。但是你可以看到，回答错误的题目以链接文本的形式显示。每个链接的*onMouseOver*事件在返回正确之前调用两个函数：show()和explain()。函数show()非常简单。它在状态栏中打印一个空串，以防止*onMouseOver*事件可能引起的任何节外生枝。下面是第110行的代码：

```
function show() { parent.status = ''; }
```

函数explain()接收一个字符串参数，利用document.write()把HTML页写到等了半天的parent.frames[1]上。有关代码在第111～118行：

```
function explain(str) {
  with (aFrame.document) {
    open();
    writeln('<HTML><BODY BGCOLOR=WHITE><FONT FACE=Arial>' + str +
      '</FONT></BODY></HTML>');
    close();
    }
  }
```

即使完成了*onMouseOver*事件的处理，explain()还是有一点工作要做。注意事件处理方法*onMouseOver*中的第101行，explain()被再次调用。但这次传递给explain()传递的是一个空串，因此框架*aFrame*在每个*onMouseOue*事件后出现时已经被清空。

剩下的事是防止在用户点击mouseover功能的链接时发生什么不期望的情况。第102行代码包含了onClick="return false;"。列在HREF属性中的任何文档URL都将被取消。

记住这仍然在*for*循环体内。上述过程发生在每一个从0到howMany-1的answer循环中。*for*循环完成以后，变量*results*变成一个很大的字符串，它包含正确答案数、试题总数、所有的试题文本和四个选项，还有用户答案和正确答案。第107～109行添加了一些HTML的结束语句，将字符串载入底部的框架，并停止函数的执行。

```
results += '\n\r</BODY></HTML>';
  qFrame.location.replace("javascript: parent.frames[0].results");
  }
```

chickenOut()

对于用户提早退出的情况有一个小问题。它当然不是必要的，你可以从任何执行程序中将它删除。我加上它只是为了给用户一个额外的功能。下面是第60～65行的代码：

```
function chickenOut() {
  if(stopOK && confirm('Stopping early? Are you really a ' +
    'JavaScript Chicken?')) {
    gradeTest();
    }
  }
```

如果用户符合条件并且确定了提早退出请求，gradeTest()将被调用。别忘了，用户至少要回答了一个问题之后才会符合提前退出的条件。初始值为false的变量*stopOK*在*qIDx*大于1后被置为true。看一下第53行。

结果是gradeTest()把答案同所有问题作比较，即使用户并没有回答也不管。这会给用户的等级评定带来巨大的损害，但它毕竟是你过早退出要付出的代价。

应用程序扩展

这个应用程序有许多种修改方法。不管怎么说吧，我所了解的两种明显的应用程序扩展是在服务器上处理测试成绩而防止作弊现象，或者修改应用程序，使它执行的是某种调查，而非测试。

防止作弊

在暂时修改应用程序后，你可能会考虑的第一件事是，“用户可以通过下载或者打开JavaScript源文件来窃取答案。”虽然查看每一道题目获取答案是一件痛苦的事，但它毕竟可以做到。

你可以不把答案放在应用程序中一起发送，并且要求用户将他的答案提交给服务器来评分，以防止那些“窥视者”。我们不会在服务器上对测试评分，但它也不会比gradeTes()麻烦多少。可能有一点麻烦，但原理是相同的。

为了删除评分功能以及添加提交给服务器端的功能，你需要完成下面的步骤：

- 从 *question.js* 里的对象和数组中去除任何表示答案的数据。
- 删除gradeTest()，将buildQuestion()中对它的调用代之以printResults()。
- 修改prinResults()，使用户能浏览他的答案，并在HTML表单中嵌入答案数据发送给等待着的服务器。

从数组中删除答案

在 *question.js* 中从题目构造器里将 this.answer 和 this.support 去除。修改：

```
function question(answer, support, question, a, b, c, d) {
  this.answer = answer;
  this.support = support;
  this.question = question;
  this.a = a;
  this.b = b;
  this.c = c;
  this.d = d;
  return this;
  }
```

变为：

```
function question(question, a, b, c, d) {
  this.question = question;
  this.a = a;
  this.b = b;
  this.c = c;
  this.d = d;
  return this;
  }
```

注意*answer*和*support*变量已经被删除了。既然你已经从构造器中删除了这些东西，你就可以从对每一个*units*元素的*new*操作中删除它们。换句话说，就是从每个*units*元素中删除开始两个参数。

删除 gradeTest()并修改 buildQuestion()

由于答案和相关解释不再存在，就没有必要再评定测试或者显示任何结果。这意味着你可以删除 gradeTest()函数。只要在 *administer.html* 中删除第 66～76 行就行了。这也意味着你可以在第 40 行的buildQuestion()中删除对 gradeTest()的调用。实际上，你要用一个对printResults()的调用来代替它，这样用户就可以看见自己的答案，答案可以插入到 HTML 表单中。

在第 39～42 行改变下面的内容：

```
if (qIdx == howMany) {
  gradeTest();
  return;
  }
```

变为：

```
if (qIdx == howMany) {
  printResults();
  return;
  }
```

Modifying printResults( )

printResults()位于大多数new操作出现的地方。*administer.html*中第84行现在看起来是这样的：

```
'<BR><BR><FONT SIZE=4>Here is how you scored: </FONT><BR><BR>';
```

把它变成：

```
'<BR><BR><FONT SIZE=4>Here is how you scored: </FONT><BR><BR>' +
'<FORM ACTION="your_server_script_URL" METHOD=POST>';
```

用下面的代码代替第92～105的内容：

```
results += '<INPUT TYPE=HIDDEN NAME="question' + (i + 1) + '" VALUE="' +
  keeper[i] + '"><B><I><FONT>COLOR=GREEN>You chose ' + keeper[i] +
  '</I></B></FONT><BR><BR><BR>';
```

在确定用户的回答是否正确并显示适当的绿色或红色文本的函数中，做决定的部分被删除了。最后，第107行目前是这样的：

```
results += '\n\r</BODY></HTML>';
```

把它改成：

```
results += '<INPUT TYPE=SUBMIT VALUE="Submit"> </FORM></BODY></HTML>';
```

这些变化被加上了FORM的开始和结束标签，一个带有用户每一个答案值的唯一命名的隐藏域，还有一个SUBMIT按钮。FROM标签和Submit按钮是不变的，但隐藏域还有一点麻烦。

每一个答案以隐藏域的值的形式写出，它按照对应题目的号码来命名。循环变量i用来为每个隐藏域创建唯一的名字，并且把题目号码和相应的用户答案联系起来。每一次变量*i*在*for*循环的更新部分（i++）加1时，一个新的隐藏域就产生了。隐藏域将按照格式*question1,question2,question3*来产生，如此类推。

改变后的printResults()仍然显示题目，四个选项，还有用户答案，但是不再评定测试。用户所要做的是选择“Submit”按钮发送答案。

转变为调查

由于调查在理论上不存在正确或者错误答案，将应用程序转变为管理某项调查，不仅需要进行以上修改，还要加上一个更加简单的东西——修改内容。只要修改units元素，用来表示带有多项选择的调查题目，就可以使用了。感谢上面所作的修改，在递交结果之前，用户就可以浏览用于市场分析的结果了。

第三章

交互式的幻灯片放映

应用程序要点

- 用图表和描述文字制作幻灯片
- 生动的上下文相关幻灯片导航
- 无须手工操作的自动导航形式
- 幻灯片的简单管理及伸缩性

JavaScript技巧

- 迈向跨浏览器DHTML的第一步
- 简单命名协定的优越性
- eval()的功能
- setInterval()和clearInterval()的使用

这个应用程序允许用户浏览成组的幻灯片，可以按照任何顺序，或者依照所选定的时间间隔在自动导航模式下依次浏览。每一张幻灯片都是一个DHTML层，它包含一个图像和适当的描述文字。你的幻灯片可以包含文字、图表、DHTML和诸如此类元素的任何组合。下面的幻灯片展示给用户的是一次虚拟的野生动物界旅行。图3-1中给出了打开的界面示例。

注意屏幕中间的幻灯片和屏幕左上角标注为“Automate”和“<Guide>”的两个图形。图“<Guide>”的箭头（“<”和“>”）允许用户一张接一张浏览幻灯片，按向前或向后的顺序放映。

在放映中用户可以通过点击“Guide”转移到任何一张幻灯片。它将显示一个幻灯片菜单，如果用户将焦点移至所要的幻灯片名字上，就会自动地显示相应的幻灯片。再点击一次“Guide”，幻灯片菜单就会消失。幻灯片菜单如图3-2所示。

在前面两章里，我们将应用程序过程从头到尾讲述了一遍。用户总是从相同的地方开始（输入搜索文字或者回答一个问题），又在相同的地方结束（得到一个匹配结果的页面或者在回答完最后一个问题后得到测试结果）。幻灯片的放映是不同的。用户

几乎可以任意地跳到任何一个地方，利用应用程序的诸多性能。因此，按其特征把应用程序分段讨论比从头到尾说一遍更为合适。我们将按照这种方法来贯穿本章的内容。

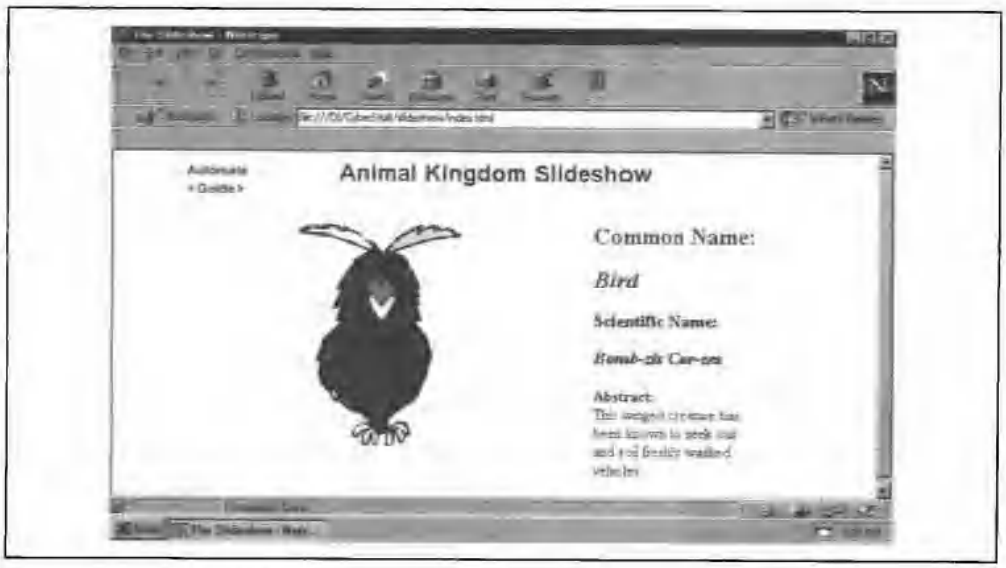


图 3-1 打开的幻灯片

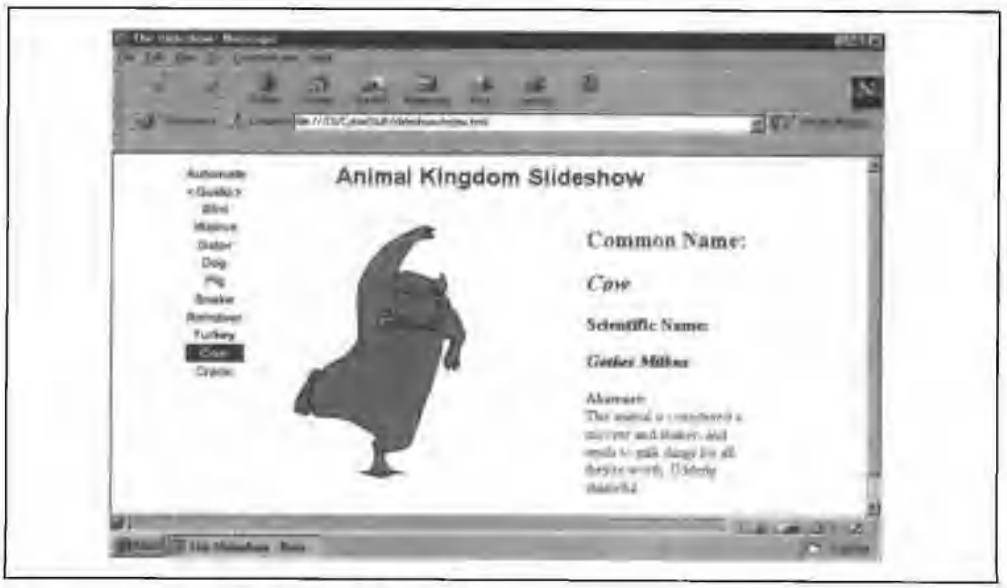


图 3-2 反白名称表示当前浏览的幻灯片

执行条件

当你说到DHTML中的“D”时，可以打赌你正在谈论IE 4.x和Navigator 4.x及以上版本。在这里也是如此。所有的幻灯片都是基于DHTML的实体。就伸缩性来说，你完全可以在在线展示中加载数百张幻灯片。但是，应用程序会预先加载所有图片（除非只有几小张），因此，我敢打赌，你会在数百幅图片的预加载中花费掉许多时间。

语法细则

脚本包含在一个文件 —— *index.html* 中。你会在压缩文件的 *ch03/index.html* 中看到它。例3-1给出了这个文件的代码。

例3-1：index.html

```
<HTML>
<HEAD>
<TITLE>The Slideshow</TITLE>
<STYLE TYPE="text/css">
#menuConstraint { height: 800; }
</STYLE>

<SCRIPT LANGUAGE="JavaScript1.2">
<!--
var dWidLyr    = 450;
var dHgtLyr    = 450;
var curSlide   = 0;
var zIdx       = -1;
var isVis      = false;

var NN         = (document.layers ? true : false);
var sWidPos    = ((NN ? innerWidth  : screen.availWidth)  / 2) -
  (dWidLyr / 2);
var sHgtPos  = ((NN ? innerHeight : screen.availHeight) / 2) -
  (dHgtLyr / 2);
var hideName = (NN ? 'hide' : 'hidden');
var showName = (NN ? 'show' : 'visible');

var img = new Array();
var imgOut = new Array();
var imgOver = new Array();
var imgPath = 'images/';

var showSpeed = 3500;
var tourOn = false;

```

```
33 function genLayer(sName, sLeft, sTop, sWdh, sHgt, sVis, copy) {
34   if (NN) {
35     document.writeln('<LAYER NAME="' + sName + '" LEFT=' + sLeft +
36       ' TOP=' + sTop + ' WIDTH=' + sWdh + ' HEIGHT=' + sHgt +
37       ' VISIBILITY="' + sVis + '"' + ' Z-INDEX=' + (++zIdx)  + '>' +
38       copy + '</LAYER>');
39     }
40   else {
41     document.writeln('<DIV ID="' + sName +
42       '" STYLE="position:absolute; overflow:none;left:' + sLeft +
43       'px; top:' + sTop + 'px; width:' + sWdh + 'px; height:' + sHgt +
44       'px;' + ' visibility:' + sVis + '; z-Index=' + (++zIdx) + '">' +
45       copy + '</DIV>');
46     }
47   }
48
49 function slide(imgStr, scientific, copy) {
50   this.name     = imgStr;
51   imagePreLoad(imgStr);
52   this.copy     = copy;
53   this.structure =
54     '<TABLE WIDTH=500 CELLPADDING=10><TR><TD WIDTH=60% VALIGN=TOP>' +
55     '<IMG SRC=' + imgPath + imgStr + '.gif></TD>' +
56     '<TD WIDTH=40% VALIGN=TOP><H2>Common Name:</H2><H2><I>' +
57     camelCap(imgStr) + '</I></H2><H3>Scientific Name: </H3><H3><I>' +
58     scientific + '</I></H3>' + '<B>Abstract:</B><BR>' + copy +
59     '</TD></TR></TABLE>';
60
61   return this;
62   }
63
64 function imagePreLoad(imgStr) {
65   img[img.length]   = new Image();
66   img[img.length - 1].src = imgPath + imgStr + '.gif';
68   imgOut[imgOut.length] = new Image();
69   imgOut[imgOut.length - 1].src = imgPath + imgStr + 'out.gif';
71   imgOver[imgOver.length] = new Image();
72   imgOver[imgOver.length - 1].src = imgPath + imgStr + 'over.gif';
73   }
74
75 var slideShow  = new Array(
76   new slide('bird', 'Bomb-zis Car-zes', 'This winged creature has been
77     known to seek out and soil freshly-washed vehicles.'),
78   new slide('walrus', 'Verius Clueless', 'These big fellas good fishers,
79     but toothbrushing is another story.'),
80   new slide('gator', 'Couldbeus Luggajus', 'These reptiles often play
81     mascots for large college sporting events.'),
82   new slide('dog', 'Makus Messus', 'Man\'s best friend? Yeah, right.
83     No wonder these mammals get a bad rep.'),
84   new slide('pig', 'Oinkus Lotsus', 'Humans with questionable eating
85     habits are often compared to these farm creatures.'),
86     new slide('snake', 'Groovius Dudis', 'Slick and sly with a
87     watchful eye.'),
88     new slide('reindeer', 'Redius Nosius', 'Though co-workers used to
```

```
    laugh and call him names, he eventually won the respect of the entire
    team.'),
  new slide('turkey', 'Goosius Is Cooktis', 'Celebrated and revered for
    an entire year, then served as dinner shortly after.'),
  new slide('cow', 'Gotius Milkus', 'This animal is considered a moover
    and shaker, and tends to milk things for all they\'re worth. Udderly
    shameful.'),
  new slide('crane', 'Whooping It Upus', 'Not to be confused with a
    piece of heavy construction equipment. Rumored as the source of the
    nickname <I>birdlegs</I>.')
  );

function camelCap(str) {
  return str.substring(0, 1).toUpperCase() + str.substring(1);
  }
function genScreen() {
  var menuStr = '';
  for (var i = 0; i < slideShow.length; i++) {
    genLayer('slide' + i, sWidPos, sHgtPos, dWidLyr, dHgtLyr,
      (i == 0 ? true : false), slideShow[i].structure);
    menuStr += '<A HREF="" onMouseOver="hideStatus(); if(!tourOn)
      { setSlide(' + i + ');' +
      ' imageSwap(\'' + slideShow[i].name + '\', ' + i + ', true)};
          return true;"' +
      ' onMouseOut="if(!tourOn) { setSlide(' + i + ');' +
      ' imageSwap(\'' + slideShow[i].name + '\', ' + i + ', false)};
          return true;"' +
      ' onClick="return false;"><IMG NAME="' + slideShow[i].name +
      '" SRC="' + imgPath + slideShow[i].name +
          'out.gif" BORDER=0></A><BR>';
    }

  genLayer('automation', sWidPos - 100, 11, 100, 200, true,
    '<A HREF="javascript: autoPilot();"  onMouseOver="hideStatus();
        return true;">' +
    '<IMG SRC="' + imgPath + 'automate.gif" BORDER=0></A>'
    );

  genLayer('guide', sWidPos - 100, 30, 100, 200, true,
    '<A HREF="javascript: if(!tourOn) { changeSlide(-1); }"
        onMouseOver="hideStatus(); return true;">' +
    '<IMG SRC="' + imgPath + 'leftout.gif" BORDER=0></A>' +
    '<A HREF="javascript: if(!tourOn) { menuManager(); }"
        onMouseOver="hideStatus(); return true;">' +
    '<IMG SRC="' + imgPath + 'guideout.gif" BORDER=0></A>' +
    '<A HREF="javascript: if(!tourOn) { changeSlide(1); }"
        onMouseOver="hideStatus(); return true;">' +
    '<IMG SRC="' + imgPath + 'rightout.gif" BORDER=0></A>'
    );
  genLayer('menu', sWidPos - 104, 43, 100, 200, false,
    '<DIV ID="menuConstraint"><TABLE><TD>' +
        menuStr + '</TD></TABLE></DIV>'
    );
```

```
144   }
145
146 function refSlide(name) {
147   if (NN) { return document.layers[name]; }
148   else { return eval('document.all.' + name + '.style'); }
149   }
150
151 function hideSlide(name) {
152   refSlide(name).visibility = hideName;
153   }
154
155 function showSlide(name) {
156   refSlide(name).visibility = showName;
157   }
158
159 function menuManager() {
160   if (isVis) { hideSlide('menu'); }
161   else { showSlide('menu'); }
162   isVis = !isVis;
163   }
164
165 function changeSlide(offset) {
166   hideSlide('slide' + curSlide);
167   curSlide = (curSlide + offset < 0 ? slideShow.length - 1 :
168     (curSlide + offset == slideShow.length ? 0 : curSlide + offset));
169   showSlide('slide' + curSlide);
170   }
171
172 function setSlide(ref) {
173   if (tourOn) { return; }
174   hideSlide('slide' + curSlide);
175   curSlide = ref;
176   showSlide('slide' + curSlide);
177   }
178
179 function imageSwap(imagePrefix, imageIndex, isOver) {
180   if (isOver) { document[imagePrefix].src = imgOver[imageIndex].src; }
181   else { document[imagePrefix].src = imgOut[imageIndex].src; }
182   }
183
184 function hideStatus() { window.status = ''; }
185
186 function autoPilot() {
187   if (tourOn) {
188     clearInterval(auto);
189     imageSwap(slideShow[curSlide].name, curSlide, false);
190     }
191   else {
192     auto = setInterval('automate()', showSpeed);
193     imageSwap(slideShow[curSlide].name, curSlide, true);
194     showSlide('menu');
195     visible = true;
196     }
197   tourOn = !tourOn;
```

```
198    }
199
200 function automate() {
201    imageSwap(slideShow[curSlide].name, curSlide, false);
202    changeSlide(1);
203    imageSwap(slideShow[curSlide].name, curSlide, true);
204    }
205
206 //-->
207 </SCRIPT>
208 </HEAD>
209 <BODY BGCOLOR=WHITE>
210 <CENTER>
211 <FONT FACE=Arial>
212 <H2>Animal Kingdom Slideshow</H2>
213 </FONT>
214 </CENTER>
215 <SCRIPT LANGUAGE="JavaScript1.2">
216 <!--
217 genScreen();
218 //-->
219 </SCRIPT>
220 </FONT>
221 </BODY>
222 </HTML>
```

应用程序变量

让我们先大致看一下变量和其他细节；然后再将进入函数的讨论。下面是第5～7行的代码：

```
<STYLE TYPE="text/css">
#menuConstraint { height: 800; }
</STYLE>
```

这里定义了一个style标签，它的名称为*menuConstraint*，有一个单独的属性，表示高度为800像素。它应用于每一张幻灯片，创建这些幻灯片是为了确保用户在浏览幻灯片的时候有足够多的页面。换句话说，如果用户显示器的高度少于800像素，这个样式表将产生竖直方向的滚动条。如果你的图片很多或者有很多复本，这是非常有用的。至少，用户可以拖动滚动条来查看下面的内容。第11～31行给出了变量：

```
var dWidLyr   = 450;
var dHgtLyr   = 450;
var curSlide  = 0;
var zIdx      = -1;
var isVis     = false;
```

```
var NN        = (document.layers ? true : false);
var sWidPos   = ((NN ? innerWidth  : screen.availWidth)  / 2) -
  (dWidLyr / 2);
var sHgtPos   = ((NN ? innerHeight : screen.availHeight) / 2) -
  (dHgtLyr / 2);
var hideName  = (NN ? 'hide' : 'hidden');
var showName  = (NN ? 'show' : 'visible');

var img     = new Array();
var imgOut  = new Array();
var imgOver = new Array();
var imgPath = 'images/';

var showSpeed = 3500;
var tourOn    = false;
```

变量被分为四组：

- DHTML 层的默认值
- 由浏览器决定的变量
- 图片的相关变量
- 自动导航的相关变量

DHTML 层的默认值

dWidLyr 和 *dHgtLyr* 只定义了幻灯片的默认高度和宽度。变量 *curSlide* 总是存储当前显示的幻灯片的数组索引值。变量 *zIdx* 用一个 z 索引值指示每一个创建出来的层，而 *isVis* 变量则保存一个布尔值，用来表示层现在是否可见。

注意： 一般来说，我把幻灯片作为 DHTML 层或者一般的层。不要把它们同 Netcape 中的用来建层的 LAYER 标签混淆起来。换句话说，在 Navigator 中一个层就是一个 LAYER，但在 IE 中却不是这样。

浏览器确定的变量

下面五个变量，*NN*，*sWidPos*，*sHgtPos*，*showName* 和 *hideName* 由载入应用程序的浏览器确定。如果 *document* 对象中存在 *layers* 属性，则第 17 行的变量 *NN* 被设

置为true值。也就是说，这是在Netcape Navigator 4.x中。文档对象模型在Netscape中的当前运行支持 *layers* 对象：

```
var NN  = (document.layers ? true : false);
```

否则，脚本就认为用户所使用的是IE 4.x，并且把 *NN* 设为false。Microsoft的对象模型在 *document.all* 的 *styles* 对象中提到了 *layers*。变量 *sWidPos* 和 *sHgPos* 保存窗口左上角的坐标x和y值，这样，层就可以放置到浏览窗口的中央（不是屏幕中央）。这不仅仅由 *NN* 来确定，还要依赖于变量 *dWidLyr* 和 *dHgLyr* 的值。下面是第18～21行的代码：

```
var sWidPos   = ((NN ? innerWidth  : screen.availWidth)  / 2) -
  (dWidLyr / 2);
var sHgtPos   = ((NN ? innerHeight : screen.availHeight) / 2) -
  (dHgtLyr / 2);
```

怎样才能得到x和y坐标值呢？你可以将浏览窗口宽度除以2得到横向坐标，将浏览窗口高度除以2得到纵向坐标，就很容易地算出了中心点坐标值。换句话说，窗口中点的坐标等于（窗口宽度像素值/2，窗口高度像素值/2）。

现在你知道了窗口中心的坐标值。由于这些坐标也要作为每个层的中心，你可以从中点横向坐标中减去 *dWidLyr* 的1/2，从中点纵向坐标值中减掉 *dHgtLyr* 的1/2，从而得到所需的x和y坐标。

剩余两个由浏览器确定的变量是字符串类型的，它们按照所用DOM（Document Object Model，文档对象模型）来保存层的可见或隐藏状态的正确名称。下面是第22和23行代码：

```
var hideName  = (NN ? 'hide' : 'hidden');
var showName  = (NN ? 'show' : 'visible');
```

依照Netscape的DOM，隐藏层的 *visibility* 属性被设置为hide，而Microsoft DOM中的 *visibility* 属性被设为hidden。相反，Netscape中被显示的层的 *visibility* 属性为show，而Microsoft中设为visible。

同样依照Netscape DOM，*window* 对象的 *innerWidth* 和 *innerHeight* 属性包含了窗口的高和宽，而Microsoft将这些值存储在 *screen* 对象的 *availWidth* 和 *availHeight* 属性中。因为变量 *NN* 已经根据这个目的来设置，JavaScript知道应该使用哪一个属性。

图片的相关变量

下面这组由数组构成的变量和图片的处理相关。第25～28行的代码如下：

```
var img = new Array();
var imgOut = new Array();
var imgOver = new Array();
var imgPath = 'images/';
```

这应该是相当简单易懂的。存储在*img*数组中的图像表示幻灯片图像。存储在*imgOut*数组中的则用于幻灯片菜单图片。数组*imgOver*中的图片用作翻转菜单图片。等一会儿我们讲到函数swapImage()时，你会发现翻转菜单可不是那么简单的。

存储在你的Web服务器中的所有图片路径值将保存在最后一个变量*imgPath*中。这个路径可以是相对的，也可以是绝对的。绝对路径包含完整的文件位置，从主机和域名或者Web服务器的IP地址（比如http://www.oreilly.com/），或者本地驱动器（比如C:\），到文件目录。

下面是两个例子：

```
var imgPath = 'http://www.serve.com/hotsyte/';
var imgPath = 'C:\\Winnt\\Profiles\\Administrator\\Desktop\\';
```

你必须用两个反斜杠符号（\\）来和任何Windows操作系统的单个斜杠区分开。如果你用的是一个单斜杠，JavaScript会认为你的意思是：

```
C:WinntProfilesAdministratorDesktop;
```

这不仅是错的，而且还是一个语法错误。

自动幻灯片放映的变量

最后两个变量*showSpeed*和*tourOn*，一个给出幻灯片的变换速度，另外一个用来指示自动导航是否在运行。它们位于第30和31行：

```
var showSpeed = 3500;
var tourOn = false;
```

变量*showSpeed*是以毫秒来表示的。你可以将变换中间的间隔时间延长，也就是说，幻灯片的信息量比较大时，你可以设置10000来表示10秒的间隔时间。你也可以

试验性地将它设为10来看看幻灯片飞快变换的效果。页面第一次载入的时候，幻灯片的自动操作还没有开始。因此，这时候的 *tourOn* 变量值自然为 `false`。

应用程序的函数

幻灯片放映函数有三种类型：层的创建，图片处理，还有导航/显示。表3-1给出了每一个函数和它所属的类型。

表3-1 幻灯片放映函数及功能描述

函数名	类型	描述
genLayer()	层	产生幻灯片
slide()	层	每一张幻灯片的对象构造器
imagePreLoad()	图片	为幻灯片和导航条预加载图片
camelCap()	层	将幻灯片名称的首字母变成大写
genScreen()	层	调用 genLayer()，放置所有层
hideSlide()	层	隐藏层
showSlide()	层	显示层
refSlide()	层	为基于浏览器的层返回一个参数
menuManager()	层	显示或隐藏幻灯片菜单
changeSlide()	层	通过指示箭头或自动导航变换当前显示的幻灯片
setSlide()	层	通过鼠标事件变换当前显示的幻灯片
imageSwap()	图片	对幻灯片菜单执行图片滚翻转
hideStatus()	导航	将窗口状态栏的值设为“”
autoPilot()	导航	管理自动导航模式
automate()	导航	自动放映幻灯片

层的相关函数

由于绝大多数的幻灯片放映设置都依赖于层的相关函数，我们有必要对它作认真的讨论。

genLayer()

这个函数是跨浏览器DHTML的关键。你在幻灯片放映中所要显示的任何东西，不管有多大、多小、五彩缤纷或者图片纷呈，它们都要经过这一关。看一下第33～47行的代码：

```
function genLayer(sName, sLeft, sTop, sWdh, sHgt, sVis, copy) {
  if (NN) {
    document.writeln('<LAYER NAME="' + sName + '" LEFT=' + sLeft +
      ' TOP=' + sTop + ' WIDTH=' + sWdh + ' HEIGHT=' + sHgt +
      ' VISIBILITY="' + sVis + '"' + ' z-Index=' + (++zIdx) + '>' +
      copy + '</LAYER>');
    }
  else {
    document.writeln('<DIV ID="' + sName +
      '" STYLE="position:absolute; overflow:none; left:' + sLeft +
      'px; top:' + sTop + 'px; width:' + sWdh + 'px; height:' + sHgt +
      'px;' + ' visibility:' + sVis + '; z-Index=' + (++zIdx) + '">' +
      copy + '</DIV>');
    }
  }
```

这个函数包含一个很长的if-else语句。实际上，genLayer()在两部分代码中执行基本相同的操作。不过一个是针对Netscape的，另一个针对IE。在文档对象模型确定之前，这是必须使用的方法。

第34行的代码用*NN*变量来确定用户的浏览器是Netscape Navigator还是Microsoft的Internet Explorer。如果*NN*的值为true，则浏览器为Navigator。否则，可以假设用户使用的是IE。

注意第33行的参数。它们是*sName*，*sLeft*，*sTop*，*sWdh*，*sHgt*，*sVis*和*copy*。不管浏览器是什么，它们都起一样的作用。sName表示给层取的名字。*sLeft*指定层到屏幕左边缘的像素数，*sTop*指定层到屏幕上边缘的像素数。*sWdh*和*sHgt*，和你所想的一样，它们是用来保存层尺寸的像素值。*sVis*保存true或false值，用来确定层是否可见（true）。*copy*包含你想作为层的内容来显示的字符串。层的内容一般是HTML语句，但它总是非常简洁。

不管浏览器是哪一种，函数genLayer()都调用document.writeln()，并且为Navigator构造一个LAYER标签，或者为IE构造一个DIV标签。

JavaScript 技巧：迈向跨浏览器 DHTML 的第一步

在升级到4.x浏览器和DHTML之前，Web开发人员必定受够了关于IE 3.x和JavaScript 1.1不兼容的抱怨。对于其他的东西，则意味着基本上没有图片翻转功能，几乎没有任何对源文件的支持，因此不得不运行工作区。

但是现在，这些麻烦可以解决了，只要给你的页面以跨浏览器的兼容性就行。这是你最有威力的武器之一：*document.all*。

对于简单的应用程序来说，一个 if-else 语句就足够了：

```
if (document.all) {    // 是IE
                       // 使用Jscript
                       // 如document.all.styles等
  }

else {                 // 是Navigator
                       // 使用JavaScript
                       // 如document.layers等
  }
```

slide()

slide()是一个对象构造器。slide()的实例包含每一个幻灯片的重要细节，比如动物名、描述文字、以及HTML内容。看一下第49～62行的代码：

```
function slide(imgStr, scientific, copy) {
  this.name       = imgStr;
  imagePreLoad(imgStr);
  this.copy       = copy;
  this.structure =
    '<TABLE WIDTH=500 CELLPADDING=10><TR><TD WIDTH=60% VALIGN=TOP>' +
    '<IMG SRC=' + imgPath + imgStr + '.gif></TD>' +
    '<TD WIDTH=40% VALIGN=TOP><H2>Common Name:</H2><H2><I>' +
    camelCap(imgStr) + '</I></H2><H3>Scientific Name: </H3><H3><I>' +
    scientific + '</I></H3>' + '<B>Abstract:</B><BR>' + copy +
    '</TD></TR></TABLE>';

  return this;
  }
```

函数 slide()接收了三个参数 —— *imgStr*，*scientific* 和 *copy*。*imgStr* 表示幻灯片中野生动物的名字。*imgStr* 在许多方面都可以算作每一张幻灯片的中枢。现在最好看一看应用程序的命名协定。每一个 *slide* 对象的名称在第 50 行进行设置。

```
this.name  = imgStr;
```

*imgStr*出现了好几次。可以参见第53－59行。下面是幻灯片的*sructure*属性的设置：

```
this.structure =
  '<TABLE WIDTH=500 CELLPADDING=10><TR><TD WIDTH=60% VALIGN=TOP>' +
  '<IMG SRC=' + imgPath + imgStr + '.gif></TD>' +
  '<TD WIDTH=40% VALIGN=TOP><H2>Common Name:</H2><H2><I>' +
  camelCap(imgStr) + '</I></H2><H3>Scientific Name: </H3><H3><I>' +
  scientific + '</I></H3>' + '<B>Abstract:</B><BR>' + copy +
  '</TD></TR></TABLE>';
```

JavaScript 技巧：使用构造合理的命名协定

在这本书里简直到处都是：命名协定（naming convention）。考虑一下幻灯片放映应用程序如何利用诸如*cow*，*bird*，还有*dog*这些简单词汇来发挥重大作用。当然，这些应用中没有一个是为哪些有复杂数据仓库的大公司制作的，但是你会在很短的时间里得到一个令人惊讶的使用量。事实上，这不仅是一个JavaScript技术，你可以在几乎所有的语言中使用它。你看一看就知道了，原来这里使用*imgStr*参数的命名协定是这么简单。

*imgStr*表示一个动物的名字。让我们以猪为例好了。这样，*imgStr*就等于*pig*。这看起来或许过于幼稚，但是字符串毕竟定义了一个动物名，幻灯片图像的基本名称，以及在幻灯片菜单中用于翻转的两张图片（我们马上就要讲到这里）。四个JavaScript成分和一个动物名都基于一个简单的字符串。这可是很合算的呢。下面的表格将说明*pig*和其他*imgStrs*值是如何分别成为各自幻灯片的关键点的。

imgStr	动物名	幻灯片图片	菜单图片	菜单图片翻转
pig	猪	*pig.gif*	*pigout.gif*	*pigover.gif*
cow	牛	*cow.gif*	*cowout.gif*	*cowover.gif*
snake	蛇	*snake.gif*	*snakeout.gif*	*snakeover.gif*

为了有效地创建幻灯片图像，slide()和必需的HTML标签<IMG>连接起来，成为带有两个变量*imgPath*和*imgStr*（都带有'.gif'后缀）的<IMG>标签。如果imgStr等于*pig*，则幻灯片图像的HTML语句就像这样：

```
<IMG SRC='images/pig.gif'>
```

属性*sructure*将幻灯片内容作为HTML表格来定义，每个表格有一个行和两个数据单元。左边的数据单元包含幻灯片图像，而右边那个则为描述文字。第57行又用了*imgStr*来给某种动物赋一个公共名称。

```
camelCap(imgStr)
```

第88～90行的函数camelCap()只把传递给它的任何字符串的首字母变为大写，然后将其作为返回值。这是一个格式化结果，仅仅使界面看起来更悦目一点。也要注意，参数*scientific*是作为动物的学名来设置的。当然，等你看到我给出来的一些学名后，你可能会怀疑（或者嘲笑）科学调查的严谨性了。

等到*imgStr*看起来已经非常精炼准确的时候，slide()将它传递给函数preLoadImages()作另外的工作。看一下第51行代码。这个函数预载入所有的幻灯片图像。我们马上就要讨论到它。

genScreen()

函数genScreen()利用应用程序创建层的能力将物体显示到屏幕上。这是到目前为止代码最多的函数。下面是第105～144行代码。genScreen()不仅管理层的创建和布置工作，还利用动态JavaScript来定义导航性质。

```
function genScreen() {
  var menuStr = '';
  for (var i = 0; i < slideShow.length; i++) {
    genLayer('slide' + i, sWidPos, sHgtPos, dWidLyr, dHgtLyr,
      (i == 0 ? true : false), slideShow[i].structure);
    menuStr += '<A HREF="" onMouseOver="hideStatus();' if(!tourOn)
      { setSlide(' + i + '); +
      ' imageSwap(\'' + slideShow[i].name + '\', ' + i + ', true)};
          return true;"' +
      ' onMouseOut="if(!tourOn) { setSlide(' + i + ');' +
      ' imageSwap(\'' + slideShow[i].name + '\', ' + i + ', false)};
          return true;"  +
      ' onClick="return false;"><IMG NAME="' + slideShow[i].name +
      '" SRC="' + imgPath + slideShow[i].name +
      'out.gif" BORDER=0></A><BR>';
    }

  genLayer('automation', sWidPos - 100, 11, 100, 200, true,
    '<A HREF="javascript: autoPilot();"  onMouseOver="hideStatus();' +
     return true;">'  +
    '<IMG SRC="' + imgPath + 'automate.gif" BORDER=0></A>'
    );
```

```
genLayer('guide', sWidPos - 100, 30, 100, 200, true,
  '<DIV ID="menuConstraint">' +
  '<A HREF="javascript: if(!tourOn) { changeSlide(-1); }" ' +
  'onMouseOver="hideStatus(); return true;">' +
  '<IMG SRC="' + imgPath + 'leftout.gif" BORDER=0></A>' +
  '<A HREF="javascript: if(!tourOn) { menuManager(); }" ' +
  'onMouseOver="hideStatus(); return true;">' +
  '<IMG SRC="' + imgPath + 'guideout.gif" BORDER=0></A>' +
  '<A HREF="javascript: if(!tourOn) { changeSlide(1); }" ' +
  'onMouseOver="hideStatus(); return true;">' +
  '<IMG SRC="' + imgPath + 'rightout.gif" BORDER=0></A></DIV>'
  );

genLayer('menu', sWidPos - 104, 43, 100, 200, false,
  '<DIV ID="menuConstraint"><TABLE><TD>' +
  menuStr + '</TD></TABLE></DIV>'
  );
}
```

这个函数要负责管理所有幻灯片层的创建，每一个幻灯片还要附加三个导航链接（一个给幻灯片菜单，一个针对“<Guide>”图片，还有一个给“自动操作”图片）。第106～120行的*for*循环负责管理各层，并产生幻灯片菜单层的内容：

```
var menuStr = '';
for (var i = 0; i < slideShow.length; i++) {
  genLayer('slide' + i, sWidPos, sHgtPos, dWidLyr, dHgtLyr,
    (i == 0 ? true : false), slideShow[i].structure);
  menuStr += '<A HREF="" onMouseOver="hideStatus();' +
  'if(!tourOn) { setSlide(' + i + '); imageSwap(\'' +
  slideShow[i].name + '\', ' + i + ', true)}; return true;"' +
  ' onMouseOut="if(!tourOn) { setSlide(' + i + ');' +
  ' imageSwap(\'' + slideShow[i].name + '\', ' + i + ', false)}; ' +
  'return true;" onClick="return false;"><IMG NAME="' +
  slideShow[i].name + '" SRC="' + imgPath + slideShow[i].name +
  'out.gif" BORDER=0></A><BR>';
  }
```

在迭代每一个*slideShow*数组元素的过程中，每调用一次genLayer()就创建一个幻灯片层，这是非常简单的。下面是更具体的代码：

```
genLayer('slide' + i, sWidPos, sHgtPos, dWidLyr, dHgtLyr,
     (i == 0 ? true : false), slideShow[i].structure);
```

函数所传递的参数非常多。表3-2具体描述了每一个参数。

表3-2 genLayer()的参数

值	描述
'slide' + i	为每一张幻灯片创建唯一的编入索引的名称，比如*slide0*，*slide1*，等等
sWidPos	从窗口左端到幻灯片的距离的像素值
sHgtPos	从窗口上端到幻灯片的距离的像素值
dWidLyr	幻灯片宽度的默认值，在本例中为450
dHgtLyr	幻灯片高度的默认值，在本例中为450
(i == 0 ? true : false)	确定是隐藏（false）还是显示（true）幻灯片。除了第一张外（*i*等于0），其他所有的幻灯片开始的时候都要隐藏
slideShow[i].structure	这是幻灯片的内容，包括文字、图表，它们嵌在一个表中。它从幻灯片构造器产生。看一下第54～59行的代码

函数genLayer()被调用了slideShow.length次，用来为每一个幻灯片创建一个层。你有6张或106张幻灯片都没有关系：这条生产线会将它们全部处理完。不过可能让你吃惊的是，genScreen()中余下的代码专门负责将三个导航层放到屏幕上，但是在继续后面的内容之前，我们要就这个*for*语句讨论更多的一些用法。看一下剩余的部分：

```
menuStr += '<A HREF="" onMouseOver="hideStatus();' +
  'if(!tourOn) { setSlide(' + i + '); imageSwap(\'' +
  slideShow[i].name + '\', ' + i + ', true)}; return true;"' +
  ' onMouseOut="if(!tourOn) { setSlide(' + i + '); imageSwap(\'' +
  slideShow[i].name + '\', ' + i + ', false)}; return true;"' +
  ' onClick="return false;"><IMG NAME="' + slideShow[i].name +
  '" SRC="' + imgPath + slideShow[i].name +
  'out.gif" BORDER=0></A><BR>';
```

第110行的变量*menuStr*，最初被初始化为一个空串，现在将会被设置为一个HTML字符串，它包含每一张幻灯片的一对翻转图片的相关代码。图3-2显示了幻灯片翻转菜单的结果。

对于每一张幻灯片来说，*menuStr*被设置为它本身加上幻灯片的相关链接图像。在

你要满足那些一个或两个条件之前，先考虑一下每个链接的一对翻转图片究竟需要什么。

1. 一个 <A HREF> 开始标签。
2. 当用户将鼠标箭头从链接图像上移过时的 *onMouseOver* 事件处理代码。
3. 当用户将鼠标箭头从链接图像上移开时的 *onMouseOut* 事件处理代码。
4. 事件处理中用来防止用户点击链接图像（尽管你可以避免）时发生任何以外情况的代码。
5. 包含唯一 NAME 和 SRC 属性的一个 <IMG> 标签。
6. 一个表示结束的 </A> 标签。

第一条很直接明了；直接敲代码就行了。

第二条要稍微麻烦一点。为了去除讨厌的状态条文字，事件处理方法 *onMouseOver* 所做的第一件事就是调用 hideStatus() 将状态条的值设为一个空串。在第 184 行你可以看到这个只有一行代码的函数。

接下来，如果，且仅如果用户没有用自动操作来放映幻灯片，*onMouseOver* 事件需要调用函数 setSlide()（我们马上就要讨论到）。现在嘛，只要记住 *i* 的值被传入就行了。

就好像 *onMouseOver* 没有足够的事情做一样，我们要做的最后一件事是为函数 imagSwap() 添加代码。这个函数管理图片翻转，一会儿也会涉及到。现在，这里的 JavaScript 代码有三个传入的值：slideShow[i].name，*i*，还有布尔值 true。

第三个条目对事件处理方法 *onMouseOut* 有一样的要求，除了不调用 hideStatus() 之外，因为状态条已经被清除了。并且，传给 *imageSwap* 的最后一个布尔值是 false，而不是 true。

第四条非常简单：只要添加 onClick="false"。这会取消用户可能做的所有点击。

下面的代码与如何确保第五条的条件有关。

```
'<IMG NAME="' + slideShow[i].name + '" SRC="' + imgPath +
  slideShow[i].name + 'out.gif" BORDER=0>'
```

<IMG>标签从slideShow[i].name得到唯一的名称。slideShow[i].name也和变量*imgPath*、字符串"out.gif"合用，以创建<IMG>的正确来源。

第六条很简单。在最后添加一个
标签就完成了。

将变量*menuStr*设置为它自身加上从上面*for*语句的每个循环体中得来的字符串。

现在，*menuStr*将会有什么发生呢？由于*menuStr*包含幻灯片菜单的HTML和JavaScript内容，因此它在第140～143行对genLayer()的调用中作为参数来传递。

```
genLayer('menu', sWidPos - 104, 43, 100, 200, false,
  '<DIV ID="menuConstraint"><TABLE><TD>' +
  menuStr + '</TD></TABLE></DIV>'
  );
```

我将这个函数留到最后来讲，是因为创建的其他两个层位于幻灯片菜单的上面，我想，按照这个顺序来讲会显得更有条理一些。注意，ID属性被设为menuConstraint的<DIV>标签。这给幻灯片放映提供了800像素的高度保证。

我们还需要调用两次genLayer()来完成幻灯片放映计划。一次是为了显示一幅链接图片来开始或结束自动放映功能，另一次是创建一幅链接图片，靠导航器的上下箭头来显示或隐藏幻灯片菜单，完成一次一张的幻灯片导航。为自动功能中的链接图片创建层并没有多少工作要做。请看第122～126行。

```
genLayer('automation', sWidPos - 100, 11, 100, 200, true,
  '<A HREF="javascript: autoPilot();"  onMouseOver="hideStatus();' +
  'return true;"><IMG SRC="images/automate.gif" BORDER=0></A>'
  );
```

你已经看到了这里的一切东西。javascript:协议将会被用在HREF属性中，完成对autoPilot()函数的调用，而事件处理方法*onMouseOver*则调用hideStatus()。如果你喜欢富有挑战性的东西，就看一看最后一个层的代码好了。第128～138行对genLayer()的调用创建了最后一个层。它包含三个图片：两个箭头和“Guide”。它看起来就像这样：<Guide>。

```
genLayer('guide', sWidPos - 100, 30, 100, 200, true,
  '<A HREF="javascript: if(!tourOn) { changeSlide(-1); }" ' +
  'onMouseOver="hideStatus(); return true;">' +
  '<IMG SRC="' + imgPath + 'leftout.gif" BORDER=0></A>' +
  '<A HREF="javascript: if(!tourOn) { menuManager(); }" ' +
  'onMouseOver="hideStatus(); return true;">' +
  '<IMG SRC="' + imgPath + 'guideout.gif" BORDER=0></A>' +
```

```
'<A HREF="javascript: if(!tourOn) { changeSlide(1); }" ' +
'onMouseOver="hideStatus(); return true;">' +
'<IMG SRC="' + imgPath + 'rightout.gif" BORDER=0></A>'
);
```

每一个图片的代码看起来都是一样的。你又一次看到了链接图片中的大量代码。点击左或者右箭头图片链接，将会有条件地调用changeSlide()。传入-1值将会使幻灯片放映退到前一张。而值1使幻灯片放映前进到下一张。我们马上就要说到changeSlide()。链接的<Guide>图片所做的所有工作是显示或隐藏幻灯片菜单，这是由menuManager()函数来控制的。

在我们让genScreen()适当休息之前，请注意在页面被载入之前，它需要在<BODY>标签里调用。IE不能在载入页面后创建层，因此，我们需要在此之前搞定这件事。下面是第215～219行的代码：

```
<SCRIPT LANGUAGE="JavaScript1.2">
<!--
genScreen();
//-->
</SCRIPT>
```

SlideShow的元素

你可能已经注意到了*slideShow*数组变量。它的每一个元素包含单个*slide*对象的构件（属性）。下面是第75～98行的*slideShow*数组。这里有10个元素，也就是有10张关于动物的幻灯片：

```
var slideShow  = new Array(
  new slide('bird', 'Bomb-zis Car-zes', 'This winged creature has ' +
     'been known to seek out and soil freshly-washed vehicles.'),
  new slide('walrus', 'Verius Clueless', 'These big fellas ' +
     'good fishers, but toothbrushing is another story.'),
  new slide('gator', 'Couldbeus Luggajus', 'These reptiles ' +
     'often play mascots for large college sporting events.'),
  new slide('dog', 'Makus Messus', 'Man\'s best friend? Yeah, right. ' +
     'No wonder these mammals get a bad rep.'),
  new slide('pig', 'Oinkus Lotsus', 'Humans with questionable eating ' +
     'habits are often compared to these farm creatures.'),
  new slide('snake', 'Groovius Dudis', 'Slick and sly with a ' +
     'watchful eye.'),
  new slide('reindeer', 'Redius Nosius', 'Though co-workers used to ' +
     'laugh and call him names, he eventually won the respect of the ' +
     'entire team.'),
  new slide('turkey', 'Goosius Is Cooktis', 'Celebrated and revered ' +
     'for an entire year, then served as dinner shortly after.'),
```

```
new slide('cow', 'Gotius Milkus', 'This animal is considered a ' +
   'moover and shaker, and tends to milk things for all they\'re ' +
   'worth. Udderly shameful.'),
new slide('crane', 'Whooping It Upus', 'Not to be confused with a ' +
   'piece of heavy construction equipment. Rumored as the source of the ' +
   'nickname <I>birdlegs</I>.')
);
```

把每一次调用幻灯片构造器所传入的值同所要求的参数作一下比较。第一个是动物的名称（也就是图片的名称）；后面一个是“学名”；每一个都由同样的描述性文字来修饰。注意第85行的文字中包括了一些HTML。就算在幻灯片自身中定义层，你也应该能树立这样的概念。参见本章后面的“应用扩展”，你可以发现有更多的应用可能性。

如果你的表单过长，可以考虑将这个数组放到JavaScript源程序中，稍微简化一下这里的代码。由于这里只有10个，我便将它保留在这个文件里了。

与图片相关的函数

就这一阶段的幻灯片函数，让我们来看看怎样管理图片。

preLoadImages()

这个函数所做的工作倒真是名副其实。看第64～73行：

```
function imagePreLoad(imgStr) {
  img[img.length]  = new Image();
  img[img.length - 1].src = imgPath + imgStr + '.gif';

  imgOut[imgOut.length] = new Image();
  imgOut[imgOut.length - 1].src = imgPath + imgStr + 'out.gif';

  imgOver[imgOver.length] = new Image();
  imgOver[imgOver.length - 1].src = imgPath + imgStr + 'over.gif';
  }
```

这个函数创建新的*Image*对象，每次预加载三个图片源文件。由于这增加了应用程序初次加载的时间，恐怕没有多少用户在浏览幻灯片时肯傻傻地等着图片下载。

变量*imgPath*与*imgStr*将分别与*.gif*，*out.gif*还有*over.gif*合并在一起，使每一张幻灯片和所需的图片相关联。例如，名为*cow*的幻灯片有图片*cow.gif*，*cowout.gif*和*cowover.gif*同它关联。

imageSwap()

这个函数执行图片的翻转，不管用户是手动地在链接图片上移过或者移出鼠标来调用它，还是在自动操作模式下运行。虽然不是很多，但第179～182行的代码给出了两者之间的所有区别：

```
function imageSwap(imagePrefix, imageIndex, isOver) {
  if (isOver) { document[imagePrefix].src = imgOver[imageIndex].src; }
  else { document[imagePrefix].src = imgOut[imageIndex].src; }
  }
```

许多关于翻转的脚本，包括在我站点上的一个，都利用两个单独的函数来执行翻转：一个针对*onMouseOver*，另一个针对*onMouseOut*。但是，你可以通过传递几个简单的参数将这两个操作结合在一个函数里。

被称为*imagePrefix*，*imageIndex*和*isOver*的这些参数表示用来给图片（又是*imgStr*）命名的基本字符串，所需图片的索引（这是genScreen()中*for*循环的循环变量*i*值），以及一个用来确定是用*imgOver*还是*imgOut*中的图片的布尔值。

为了让这个情况更清晰一些，易于你的理解，我们看一看genScreen()函数的第105～120行代码。注意第112行所创建的动态JavaScript：

```
imageSwap(\'' + slideShow[i].name + '\', ' + i + ', true)};
```

等它写到文档中去，并且*i*等于0时，就会是这个样子：

```
imageSwap('bird', 0, true);
```

函数一旦被调用，你就可以知道事情是从何处开始的了。由于*isOver*的值为true，所以：

```
document[bird].src = imgOver[0].src;
```

同时，imgOver[0].src为*images/birdover.gif*。如果*isOver*等于false，图片就设置为imgOut[0].src，也就是*images/birdout.gif*。

导航函数

幻灯片函数创建了幻灯片，以及对它们的浏览控制。函数image负责预加载和翻转功能。现在我们来看看究竟是什么使这个幻灯片放映动起来的——导航函数。

refSlide()，hideSlide()，showSlide()和menuManager()

幻灯片已经被创建，图片也已经载入。现在，我们要来对幻灯片做点什么——也就是显示我们所要的某一张幻灯片，而把其他的隐藏起来。在我们能实现此操作之前，我们必须先能得到它们。这通常都很简单，对吗？只要使用层的名称就可以了。不过，你想并不完全正确。你确实必须用到层的名称，但是要记住，Navigator和IE在它们的文档对象模式中引用层的方法是不同的。函数refSlide()在第146～149行对此作了处理：

```
function refSlide(name) {
  if (NN) { return document.layers[name]; }
  else { return eval('document.all.' + name + '.style'); }
  }
```

如果用户使用的是Navigator，refSlide()返回一个参数到document.layers[name]。但如果是IE，refSlide()就用eval('document.all.' + name + '.style')返回一个参数。这允许我们改变层的可见性，而不管用的是哪一种浏览器。没什么好奇怪的，让我们来看第151～157行的这两个函数好了：

它不仅简单，而且所有这些成员以后都非常易于访问。

```
function hideSlide(name) {
  refSlide(name).visibility = hideName;
  }

function showSlide(name) {
  refSlide(name).visibility = showName;
  }
```

两个函数都调用refSlide()并且传递它们所接收的名称参数。这些代码可能初看有点奇怪。如何使refSlide()有*visibility*属性？事实上它是没有的。但是要记住，refSlide()返回一个referece给层，每一个层都有*visibility*属性。如果我们想隐藏特定的层，那么我们可以用refSlide()得到它，然后把返回对象的*visibility*属性设置为*hideName*——如果你记得的话，它在第22行基于所使用的浏览器类型而被设置回字符串*hide*或者*hidden*。显示一张幻灯片也是相同的道理，除了返回层的*visibility*属性在第23行被设为*showName*的值之外，也是基于浏览器来设置的。

hideSlide()和showSlide()要隐藏和显示的不仅是幻灯片，而且还有幻灯片菜单。函数的调用不是直接的;而是通过函数menuManager()来调用，如下所示:

JavaScript 技巧：eval()的功能

正如Netscape所说，eval()“不借助对特殊对象的引用，而对一个JavaScript代码的字符串求值。”可能这说得不完全对，但这个函数对所有对象都是可用的，这对我们这些程序员来说可是件好事。假设你要涉及某个对象，但是你不能确定它的索引号码（如果它是一个数组）或者你需要分离一个字符串以正确地访问对象。下面是这个应用程序的例子：

```
eval("document.all.styles." + name + ".visibility");
```

这里是另外一个例子：

```
eval("document.forms[0]." + elementName + ".value");
```

在许多情况下这都很容易处理，包括构造表单对象和图片翻转，以及执行数学计算，所有将字符串用作输入的事件。确保将eval()添加到你的库中。在http://developer.netscape.com/docs/manuals/communicator/jsref/glob8.htm地址下访问Netscape公司的DevEdge Online可以获得更多关于eval()的信息。

```
function menuManager() {
  if (isVis) { hideSlide('menu'); }
  else { showSlide('menu'); }
  isVis = !isVis;
  }
```

只要是幻灯片在显示的时候，变量 *isVis* 的值就为true；否则为false。因此，如果 *isVis* 为false，menuManager()就显示幻灯片，如果它为true就隐藏幻灯片，然后将 *isVis* 取反以进行下一次操作。

changeSlide()

既然我们能正确地得到幻灯片而不管用的是什么浏览器，而且我们用函数来隐藏和显示幻灯片（以及幻灯片菜单），那么我们的确需要一个函数将幻灯片转换到下一张。事实上，我们在这里可以使用两个函数：changeSlide()和setSlide()。

我希望我没有带着你忙于显示和隐藏幻灯片的工作。幻灯片的转换实际上包括三步：

1. 隐藏当前的幻灯片。

2. 确定下一步将显示哪一张幻灯片。

3. 显示幻灯片。

到此刻为止第一步和第二步似乎有点麻烦，但其实第二步比你所预想的更让人头痛。转换幻灯片有两种情形。第一种发生在你想一个接一个地进行转换，按一定的顺序向前或向后的时候。这种类型也就是当你用“<”和“>”箭头浏览幻灯片的时候。第二种情况发生在用自动操作来推动幻灯片放映的时候。函数 changeSlide()可以处置这两类情况。看看第 165～170 行：

```
function changeSlide(offset) {
  hideSlide('slide' + curSlide);
  curSlide = (curSlide + offset < 0 ? slideShow.length - 1 :
    (curSlide + offset == slideShow.length ? 0 : curSlide + offset));
  showSlide('slide' + curSlide);
  }
```

所发生的第一件事是调用 hideSlide()，附带表达式'slide' + curSlide的值。变量 *curSlide*在第13行初始化为 0。由于那是正在显示的幻灯片，函数 hideSlide()将隐藏 *slide0*，它是一张关于鸟类的幻灯片。这就足够了。现在哪一张幻灯片将会被显示？回忆一下，changeSlide()需要一个名为 *offset* 的参数。*offset* 是 1 或者 -1。为1时，slideShow数组中排在它后面的幻灯片将会被显示。由于 *curSlide* 是指示当前显示的幻灯片索引的整数，加 1 使它变成值 1，然后是2，然后是3，等等。如果值为 -1，则 *slideShow* 数组中它前边的一张幻灯片被显示。没有什么好奇怪的，如果 *curSlide* 是 3，加 -1 它将变成 2，然后是 1，然后是 0。

一切状况都似乎很好，但是如果你尝试隐藏一张名为 'slide' + -1 或 'slide'+ slideShow.length 的幻灯片时情况就不妙了。那些幻灯片是不存在的，并且你还会产生语法错误。因此你如何才能防止 *curSlide* 落到零以下或者大于 slideShow.length - 1？

第 167～168 行给出了解决方案：

```
curSlide = (curSlide + offset < 0 ? slideShow.length - 1 :
  (curSlide + offset == slideShow.length ? 0 : curSlide + offset));
```

curSlide 的值被一列嵌套的三元算子所确定。这里是一段对应的伪代码：

```
IF curSlide + offset小于0, THEN curSlide等于slideShow.length - 1
 ELSE
```

```
IF curSlide + offset等于slideShow.length THEN curSlide等于0
ELSE curSlide等于curSlide + offset
```

如果在 *curSlide* 上加 *offset* 会使 *curSlide* 过小，*curSlide* 就会被设为 slideShow.length -1。如果 *offset* 加上 *curSlide* 太大，只要将 *curSlide* 设为 0 就行了。否则，*curSlide* 就可以被设置为它本身加上 *offset* 的值而不出错。

一旦 *curSlide* 被确定，就可以放心地在第 169 行调用 showSlide()了。

setSlide()

changeSlide()是用来转换幻灯片的两个函数之一。但是 changeSlide()通过将当前显示的幻灯片加/减 1 来完成转换。而函数 setSlide()隐藏当前幻灯片，因此无论显示哪张幻灯片都和它接收到的作为参数的索引数有关。这是第172～177行代码：

```
function setSlide(ref) {
  if (tourOn) { return; }
  hideSlide('slide' + curSlide);
  curSlide = ref;
  showSlide('slide' + curSlide);
  }
```

第一行代码通过检测 *tourOn* 来确定自动操作模式是否运行，如果是，就立即返回。既然有自动放映，就没有必要转换幻灯片了；它已经自动为你实现了。

就像 changeSlide()一样，setSlide()隐藏当前幻灯片。但和 changeSlide()不同的是，setSlide()不考虑 *curSlide* 的当前的值是多少。setSlide()将参数的值赋给 *curSlide*，然后显示与那个数字有关的幻灯片。

autoPilot()

和你想像的一样，autoPilot()控制幻灯片放映的自动导航特性。autoPilot()从放映屏幕的相同链接来启动或者关闭。看一下第 186～198 行：

```
function autoPilot() {
  if (tourOn) {
    clearInterval(auto);
    imageSwap(slideShow[curSlide].name, curSlide, false);
    }
```

```
else {
  auto = setInterval('automate()', showSpeed);
  imageSwap(slideShow[curSlide].name, curSlide, true);
  showSlide('menu');
  visible = true;
  }
tourOn = !tourOn;
}
```

autoPilot()利用 *tourOn* 的值可以“知道”自动导航功能是开是关。如果 *tourOn* 为 false，则自动导航没有运行。因此，函数使用 *window* 对象的 setInterval() 方法每 *showSpeed* 毫秒调用一次函数 autoPilot()(一会儿再讨论)。

如果在放映过程中能看见幻灯片菜单，那可是件好事，在菜单图像中用高亮度表示当前的幻灯片。既然用户要单击“自动化”图像才能链接调用 autoPilot()，autoPilot()就负责显示幻灯片菜单(如果它没被显示)并且在特定的菜单图像中进行高亮度显示。函数 automate()负责其余工作，因此它仅需要被执行一次。

但是，如果autopilot现在是运行的(也就是 *tourOn* 为 false)，则 autoPilot()利用 clearInterval()函数 —— 也是 *window* 对象中的，来删除与变量 *auto* 有关的 setInterval()调用。为了保持内容简洁，对 imageSwap()的最后一次调用用来翻转高亮度的幻灯片菜单图片，使它得到和其他图片相同的亮度。

autoPilot()所做的最后一个工作是将 *tourOn* 的当前值取反。显然，如果你单击把它打开的话，下次你再单击时就会把它关掉，如此反复。

automate()

automate()是个小函数，它通过下面的操作来运行幻灯片放映:

1. 模拟 *onMouseOut* 事件，使当前幻灯片菜单中的高亮度图像转变成一般图像。这通过对 imageSwap()的调用来实现。
2. 调用 changeSlide()转换到下一张幻灯片。
3. 模拟 *onMouseOver* 事件，使当前幻灯片菜单中的一般图像转变成高亮度图像。这通过对 imageSwap()的调用来实现。

这里是第 200～204 行的代码，是关于这个问题的全部内容。

JavaScript 技巧：setInterval()和 clearInterval()的介绍

窗口的 setInterval()和 clearInterval()方法是它们的 JavaScript 1.0 同胞 setTimeout()和 clearTimeout()的升级。然而 setTimeout()只用它的第一个参数运行一次代码，而 setInterval()的代码运行是不定的。为了获得相同的结果，你必须调用 setTimeout()和递归地包含它的一个函数，就像这样：

```
y = 50;

function overAndOver() {
  // 此处省略其他代码
  y = Math.log(y);

  // 再次调用
  setTimeout("overAndOver()", 250);
  }
```

overAndOver()函数可以被这样调用：

```
<BODY onLoad="overAndOver()";>
```

setInterval()负责暗中控制递归，并且可以在某一次调用中负责这一工作。

```
y = 50;

function overAndOver() {
  // 此处省略其他代码
  y = Math.log(y);
  }
```

*onLoad*事件操作也能产生这些代码。只要确保你“关上”了 clearInterval()执行的操作。

```
function automate() {
  imageSwap(slideShow[curSlide].name, curSlide, false);
  changeSlide(1);
  imageSwap(slideShow[curSlide].name, curSlide, true);
  }
```

最后要注意一个问题。两个对 imageSwap()的调用都传递 *curSlide* 的值，就造成了一个错觉，仿佛是同样的幻灯片菜单图像在翻来翻去。但要记住，对 changeSlide()

的调用改变了 *curSlide* 的值。因此，第二个对 imageSwap()的调用使翻转发生在当前的幻灯片菜单图片上。

应用程序扩展

由于几乎所有的应用程序都使用DHTML,你能把几十种东西加到幻灯片放映中，以使它变得非常精彩。我只能尽量少介绍一些。

在自动导航中随机转换幻灯片

为什么不把事情综合一下呢？生成一个位于 0和 slideShow.length-1随机的整数。然后调用 setSlide()。函数可能会变成这样：

```
function randomSlide() {
  var randIdx = Math.floor(Math.rand() * slideShow.length);
  setSlide(randIdx);
  }
```

在函数 automate()中调用 randomSlide()代替对 changeSlide()的调用：

```
function automate() {
  imageSwap(slideShow[curSlide].name, curSlide, false);
  randomSlide();
  imageSwap(slideShow[curSlide].name, curSlide, true);
  }
```

幻灯片中生动的 GIF 或图片翻转

这些是明显可以改进的，但它们的确非常有帮助。用户确实非常喜欢添加交互性;网页上任何有很酷的动感和色彩的东西（除了可怜的 <BLINK>标签以外）都能使人耳目一新。

使幻灯片动起来

这个应用程序中所创建的每一张幻灯片都位于同一个地方。有一些幻灯片你看见了，却又没有看见。但是在整个放映期间，幻灯片都呆在同一个位置。为什么不让幻灯片从左边进来从右边出去呢？或者从上进从下出？

我正在向你打开通向一个完整的新程序的大门，因此我不会引入代码，但是我会告诉你在哪儿能得到 JavaScript 工具包，用来载入层效果。Netscape 有一个库文件正等着你的下载。

你能在http://developer.netscape.com/docs/technote/dynhtml/csspapi/xbdhtml.txt.地址下找到它。

注意到它有 *.txt* 扩展名。你把它存为本地文档时，将扩展名改为 *.js*。

浅尝辄止

这本书不是专门讲述 DHTML 的。如果你想探究 slideshow 的扩展，会有许多资源可用。我常用的是：

Netscape 交流中的 NetscapeDHTML:

http://developer.netscape.com/docs/manuals/communicator/dynhtml/index.htm

Microsoft 的 DHTML 参考书目：

http://msdn.microsoft.com/developer/sdk/inetsdk/help/dhtml/references/dhtmlrefs.htm

万维网联盟的 HTML 4.0 规范：

http://www.w3.org/TR/REC-html40/

Macromedia 公司的 DHTMLZone:

http://www.dhtmlzone.com/

Dynamic Drive 站点：

http://dynamicdrive.com/

第四章

多搜索引擎界面

应用程序要点
- 基于框架的多搜索引擎
- 单点击的搜索
- 简单的搜索引擎管理

JavaScript技巧
- 代码的重复使用
- 别去想面向对象了
- 数学计算与内存
- escape()的使用

网上到处都充斥着用JavaScript写的多搜索引擎应用程序。这种应用程序是在JavaScript的发展中最酷也是最简单的内容之一。为什么不是呢？你可以利用OPD（other people's data，别人的数据）来给站点安置一个通向网络世界的入口——这是我说的。当然，除此之外还有其他许多精彩的应用程序，但是这个应用非常容易地就给你带来了巨大的利益。图4-1给出了在浏览器中打开*ch04/index.html*时的界面。

这个应用程序的使用并不复杂。用户在左下角输入搜索文字，然后可以用箭头浏览一个关于可用的搜索引擎的基于层的菜单。用户所要做的只是点击所要使用的搜索引擎按钮，然后，结果就会显示在页面中部。我在Image Surfer图片数据库中搜索“andromeda”条目，得到如图4-2所示的结果。

所有能找到的东西都在这里了。注意到结果框架有一个黑色的边框。那是我在网页设计上玩的小花样。不过这只是个性化的选择，当然也可以很容易地把它变成一个更基本的两个框架（上部和下部）的设计。

顺便说一句，如果你是按部就班，一章一章地来看这本书的话，你会很快注意到这里与前面不同的是没有给出完整的新代码。实际上，我会告诉你怎样对第三章所涉

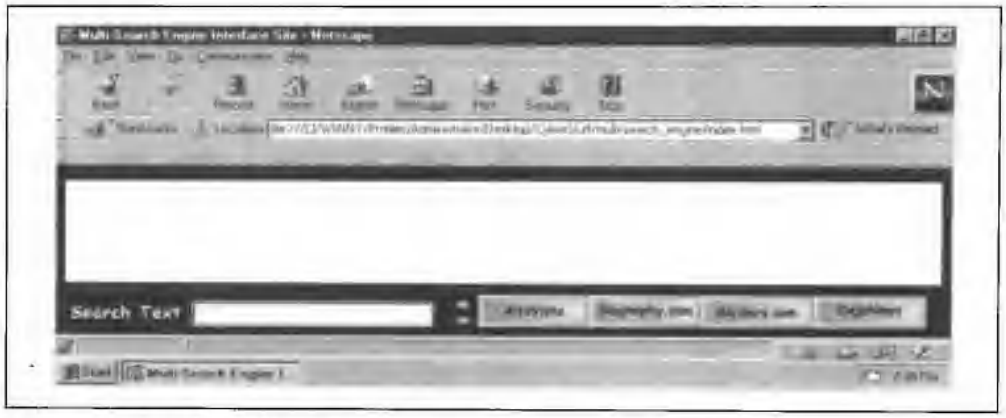


图 4-1 多搜索引擎的界面

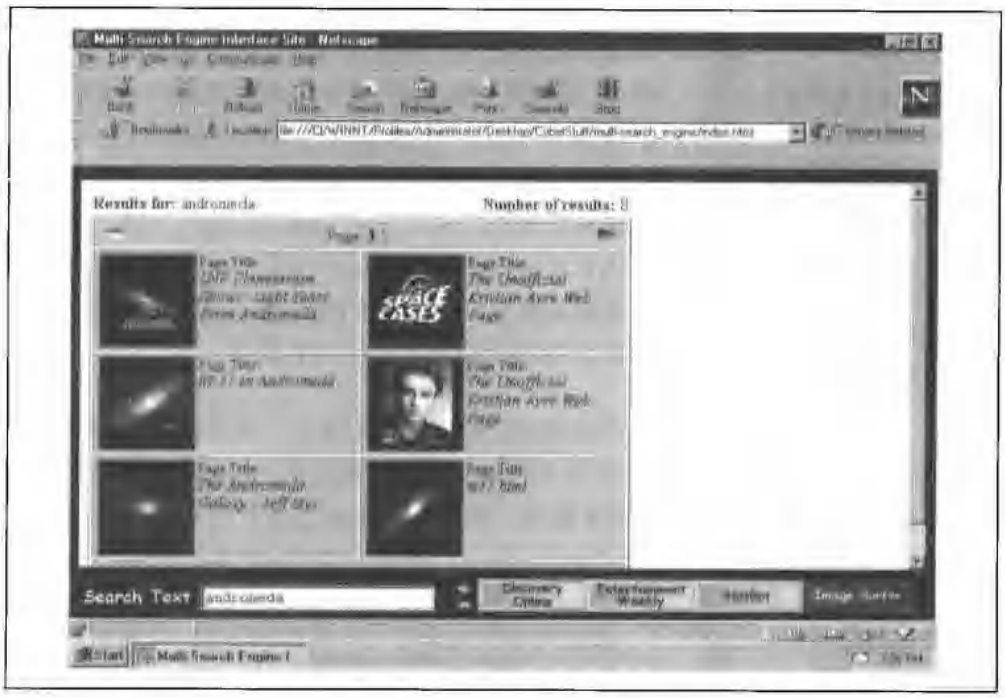


图 4-2 Image Surfer 图片数据库返回库中和条目“andromeda”有关的图片

及到的代码举一反三，应用自如。如果要了解如何重新运用以前的代码从而省时间，这将是一个很重要的途径。

执行条件

应用程序使用了DHTML，因此你要用Navigator或者IE 4.x才行。我在这里囊括了20个搜索引擎。但是你所能使用的搜索引擎数可以达到数百个。不过这个数字可能已经超过了用户所需数量的平均值。还要记住，这个应用程序虽然能在本地机器上运行得相当好，但对幻灯片放映来说，Internet的用户会为大量的图表上载花费很多时间。

语法细则

这个应用程序包括两个文件：*index.html*和*multi.html*。*index.html*在例4-1中给出，它利用框架嵌套实现环绕边的效果。

例4-1：index.html

```
 1 <HTML>
 2 <HEAD>
 3 <TITLE>Multi-Search Engine Interface Site</TITLE>
 4 <SCRIPT LANGUAGE="JavaScript1.2">
 5 <!--
 6 var black = '<BODY BGCOLOR=BLACK></BODY>';
 7 var white = '<BODY BGCOLOR=WHITE></BODY>';
 8 file://-->
 9 </SCRIPT>
10 </HEAD>
11 <FRAMESET ROWS="15,*,50" FRAMEBORDER=0 BORDER=0>
12 <FRAME SRC="javascript: parent.black;" SCROLLING=NO>
13 <FRAMESET COLS="15,*,15"  FRAMEBORDER=0 BORDER=0>
14   <FRAME SRC="javascript: parent.black;"SCROLLING=NO>
15   <FRAME SRC="javascript: parent.white;">
16   <FRAME SRC="javascript: parent.black;"
17     SCROLLING=NO>
18   </FRAMESET>
19 <FRAME SRC="multi.html" SCROLLING=NO>
20 </FRAMESET>
21 </HTML>
```

第6和第7行的两个JavaScript变量*black*和*white*，将作为第12行和第14～16行中框架SRC属性的HTML串值。我们回顾一下第二章中“JavaScript技巧：获取SRC属性”部分与此有关的内容。如果你是按顺序来阅读的，将对此很快有所理解和提高。可以看到，实际上动作的框架只有frames[2]（它显示搜索结果）和frame[4]

(包含搜索引擎界面)。其他框架纯粹是用来显示的。让我们来看看例4-2中给出的*multi.html*代码。

例4-2：multi.html

```
<HTML>
<HEAD>
<TITLE>Multi-Engine Menu</TITLE>
<SCRIPT LANGUAGE="JavaScript1.2">
<!--

parent.frames[2].location.href  = 'javascript: parent.white';

var NN      = (document.layers ? true : false);
var curSlide    = 0;
var hideName    = (NN ? 'hide' : 'hidden');
var showName    = (NN ? 'show' : 'visible');
var perLyr    = 4;
var engWdh    = 100;
var engHgt    = 20;
var left      = 375;
var top      = 10;
var zIdx      = -1;
var imgPath    = 'images/';
var arrayHandles = new Array('out', 'over');

for (var i = 0; i < arrayHandles.length; i++) {
  eval('var ' + arrayHandles[i] + ' = new Array()');
  }

var engines = new Array(
  new Array('HotBot',
    'http://www.hotbot.com/?MT=',
    'http://www.hotbot.com/'),
  new Array('InfoSeek',
    'http://www.infoseek.com/Titles?col=WW&sv=IS&lk=noframes&qt=',
    'http://www.infoseek.com/'),
  new Array('Yahoo',
    'http://search.yahoo.com/bin/search?p=',
    'http://www.yahoo.com/'),
  new Array('AltaVista',
    'http://www.altavista.com/cgi-bin/query?pg=q&kl=XX&q=',
    'http://www.altavista.digital.com/'),
  new Array('Lycos',
    'http://www.lycos.com/cgi-bin/pursuit?matchmode=and&cat=lycos' +
      '&query=',
    'http://www.lycos.com/'),
  new Array('Money.com',
    'http://jcgi.pathfinder.com/money/plus/news/searchResults.oft?' +
      'vcssortby=DATE&search=',
    'http://www.money.com/'),
  new Array('DejaNews',
```

```
48      'http://www.dejanews.com/dnquery.xp?QRY=',
49      'http://www.dejanews.com/'),
50    new Array('Insight',
51      'http://www.insight.com/cgi-bin/bp/870762397/web/result.html?' +
52        'a=s&f=p&t=A&d=',
53      'http://www.insight.com/'),
54    new Array('Scientific American',
55      'http://www.sciam.com/cgi-bin/search.cgi?' +
56      'searchby=strict&groupby=confidence&docs=100&query=',
57      'http://www.sciam.com/cgi-bin/search.cgi'),
58    new Array('Image Surfer',
59      'http://isurf.interpix.com/cgi-bin/isurf/keyword_search.cgi?q=',
60      'http://www.interpix.com/'),
61    new Array('MovieFinder.com',
62      'http://www.moviefinder.com/search/results/1,10,,00.html?' +
63        'simple=true&type=movie&mpos=begin&spat=',
64      'http://www.moviefinder.com/'),
65    new Array('Monster Board',
66      'http://www.monsterboard.com/pf/search/USresult.htm? ' +
67        'loc=&EmploymentType=F&KEYWORDS=',
68      'http://www.monsterboard.com/'),
69    new Array('MusicSearch.com',
70      'http://www.musicsearch.com/global/search/search.cgi?QUERY=',
71      'http://www.musicsearch.com/'),
72    new Array('ZD Net',
73      'http://xlink.zdnet.com/cgi-bin/texis/xlink/xlink/search.html?' +
74        'Utext=',
75      'http://www.zdnet.com/'),
76    new Array('Biography.com',
77      'http://www.biography.com/cgi-bin/biomain.cgi?search=FIND&field=',
78      'http://www.biography.com/'),
79    new Array('Entertainment Weekly',
80      'http://cgi.pathfinder.com/cgi-bin/ew/cg/pshell?venue=pathfinder&q=',
81      'http://www.entertainmentweekly.com/'),
82    new Array('SavvySearch',
83      'http://numan.cs.colostate.edu:1969/nph-search?' +
84        'classic=on&Boolean=OR&Hits=10&Mode=MakePlan&df=normal' +
85        '&AutoStep=on&KW=',
86      'http://www.savvysearch.com/'),
87    new Array('Discovery Online',
88      'http://www.discovery.com/cgi-bin/searcher/-?' +
89        'output=title&exclude=/search&search=',
90      'http://www.discovery.com/'),
91    new Array('Borders.com',
92      'http://www.borders.com:8080/fcgi-bin/db2www/search/' +
93        'search.d2w/QResults?doingQuickSearch=1&srchPage=QResults' +
94        '&mediaType=Book&keyword=',
95        'http://www.borders.com/'),
96    new Array('Life Magazine',
97      'http://cgi.pathfinder.com/cgi-bin/life/cg/pshell?' +
98        'venue=life&pg=q&date=all&x=15&y=16&q=',
99      'http://www.life.com/')
100       );
```

```
101
102 engines = engines.sort();
103
104 function imagePreLoad(imgName, idx) {
105   for(var j = 0; j < arrayHandles.length; j++) {
106     eval(arrayHandles[j] + "[" + idx + "] = new Image()");
107     eval(arrayHandles[j] + "[" + idx + "].src = '" + imgPath +
108     imgName + arrayHandles[j] + ".jpg'");
109     }
110   }
111
112 function engineLinks() {
113     genLayer('sliderule', left - 20, top + 2, 25, engHgt, true,
114       '<A HREF="javascript: changeSlide(1);" ' +
115       'onMouseOver="hideStatus(); return true;">' +
116       '<IMG SRC="' + imgPath + 'ahead.gif" BORDER=0></A><BR>' +
117       '<A HREF="javascript: changeSlide(-1);" ' +
118       'onMouseOver="hideStatus(); return true;">' +
119       '<IMG SRC="' + imgPath + 'back.gif" BORDER=0></A>');
120     lyrCount = Math.ceil
121     (engines.length / perLyr);
122     for (var i = 0; i < lyrCount; i++) {
123     var engLinkStr = '<TABLE BORDER=0 CELLPADDING=0 CELLSPACING=0><TR>';
124     for (var j = 0; j < perLyr; j++) {
125       var imgIdx   = (i * perLyr) + j;
126       if (imgIdx == engines.length) { break; }
127       var imgName  = nameFormat(engines[imgIdx][0]);
128       imagePreLoad(imgName, imgIdx);
129       engLinkStr += '<TD><A HREF="javascript: ' +
130         'callSearch(document.forms[0].elements[0].value, ' +
131         imgIdx + ');" onMouseOver="hideStatus(); imageSwap(\'' +
132         imgName + '\', ' + imgIdx + ', 1); return true" ' +
133         'onMouseOut="imageSwap(\'' + imgName + '\', ' + imgIdx +
134         ', 0);"><IMG NAME="' + imgName + '" SRC="' + imgPath + imgName +
135         'out.jpg' + '" BORDER=0></A></TD>';
136       }
137     engLinkStr += '</TR></TABLE>';
138     genLayer('slide' + i, left, top, engWdh, engHgt, false, engLinkStr);
139     }
140   }
142 function genLayer(sName, sLeft, sTop, sWdh, sHgt, sVis, copy) {
143   if (NN) {
144     document.writeln('<LAYER NAME="' + sName + '" LEFT=' + sLeft +
145       ' TOP=' + sTop + ' WIDTH=' + sWdh + ' HEIGHT=' + sHgt +
146       ' VISIBILITY=' + sVis + ' z-Index=' + (++zIdx) + '>' +
147       copy + '</LAYER>');
148     }
149   else {
150     document.writeln('<DIV ID="' + sName +
151       '" STYLE="position:absolute; overflow:none;left: ' +
152       sLeft + 'px; top:' + sTop + 'px; width:' + sWdh + 'px; height:' +
153       sHgt + 'px; visibility:' + sVis + ' z-Index=' + (++zIdx) +
154       '">' + copy + '</DIV>');
```

```
155      }
156    }
157
158 function nameFormat(str) {
159   var tempArray = str.split(' ');
160   return tempArray.join('').toLowerCase();
161   }
162
163 function hideSlide(name) { refSlide(name).visibility = hideName; }
164
165 function showSlide(name) { refSlide(name).visibility = showName; }
166
167 function refSlide(name) {
168   if (NN) { return document.layers[name]; }
169   else { return eval('document.all.' + name + '.style');
170   }
171
172 function changeSlide(offset) {
173   hideSlide('slide' + curSlide);
174   curSlide = (curSlide + offset < 0 ? slideShow.length - 1 :
175     (curSlide + offset == slideShow.length ? 0 : curSlide + offset));
176   showSlide('slide' + curSlide);
177   }
178
179 function imageSwap(imagePrefix, imageIndex, arrayIdx) {
180   document[imagePrefix].src = eval(arrayHandles[arrayIdx] +
181     "[" + imageIndex + "].src");
182   }
183
184 function callSearch(searchTxt, idx) {
185   if (searchTxt == "") {
186     parent.frames[2].location.href = engines[idx][2] +
187       escape(searchTxt);
188     }
189   else {
190     parent.frames[2].location.href = engines[idx][1] +
191       escape(searchTxt);
192     }
193   }
194
195 function hideStatus() { window.status = ''; }
196
197 //-->
198 </SCRIPT>
199
200 </HEAD>
201 <BODY BGCOLOR="BLACK" onLoad="showSlide('slide0');">
202 <SCRIPT LANGUAGE="JavaScript1.2">
203 <!--
204 engineLinks();
205 //-->
206 </SCRIPT>
207 <FORM onSubmit="return false;">
```

```
208 <TABLE CELLPADDING=0>
209   <TR>
210     <TD>
211     <FONT FACE=Arial>
212     <IMG SRC="images/searchtext.jpg">
213     </TD>
214     <TD>
215     <INPUT TYPE=TEXT SIZE=25>
216     </TD>
217   </TR>
218 </TABLE>
219 </FORM>
220 </BODY>
221 </HTML>
```

有长达200行以上的代码呢，但是其中的大部分你已经见过了。不过这并不表示它不是好东西哦。我们从第7行开始吧。

```
parent.frames[2].location.href ='javascript: parent.white';
```

如果你掂量一下*index.html*中的框架，你会发现frame[2]是用来显示搜索结果的。如果你决定重新载入应用，可以在这个框架中通过设置location.href属性来进行优化。这将自动地把结果文档内容设为一些“本地”HTML，这样，你就不必为重载入任何以前的查询内容而等上半天了。

顺便说一句，即使你在frames[2]中有了一个整洁的搜索引擎结果显示，但一旦选择了一个搜索结果链接，那就要完全依仗搜索引擎设计者的同情心了。有的会让你在同一个框架中得到链接结果。但不幸的是，比如InfoSeek，它会让文档跑到浏览器窗口的顶部去显示。

浏览“内存条”（Memory Lane）

让我们来从头浏览一下“内存条”（RAM，在这里可能让你觉得奇怪）。你查看下面这些变量的时候，会看到其中有一些新成员，不过许多变量都和第三章中用过的有惊人的相似之处。看，有*NN*和*curSlide*！后面还有*hideName*和*showName*。更别说*imagePath*和*zIdx*了：

```
var NN          = (document.layers ? true : false);
var curSlide    = 0;
var hideName    = (NN ? 'hide' : 'hidden');
var showName    = (NN ? 'show' : 'visible');
var perLyr      = 4;
```

```
var engWdh       = 100;
var engHgt       = 20;
var left         = 375;
var top          = 10;
var zIdx         = -1;
var imgPath      = 'images/';
var arrayHandles = new Array('out', 'over');
```

这些变量的功能和在第三章中的相同。它们只不过是从停止的地方又重新开始而已。在新变量中，*perLyr*定义了每页要显示的搜索引擎个数。变量*engwdh*和*engHgt*分别定义每一个层的宽和高的默认值。变量*left*和*top*保存表示层位置的值。变量*arrayHandles*则包含一个用于预载图片的数组。先有这样的一个概念，我们很快就要作相关的讨论。

如果说到“全家团圆”，变量可不是唯一熟悉的代码。看看老早以前的函数吧。

第142～156行：

```
function genLayer(sName, sLeft, sTop, sWdh, sHgt, sVis, copy) {
  if (NN) {
    document.writeln('<LAYER NAME="' + sName + '" LEFT=' + sLeft +
      ' TOP=' + sTop + ' WIDTH=' + sWdh + ' HEIGHT=' + sHgt +
      ' VISIBILITY=' + sVis + ' z-Index=' + (++zIdx) + '>' +
      copy + '</LAYER>');
    }
  else {
    document.writeln('<DIV ID="' + sName +
      '" STYLE="position:absolute; overflow:none;left: ' +
      sLeft + 'px; top:' + sTop + 'px; width:' + sWdh + 'px; height:' +
      sHgt + 'px; visibility:' + sVis + ' z-Index=' + (++zIdx) +
      '">' + copy + '</DIV>');
    }
  }
```

第163～177行：

```
function hideSlide(name) { refSlide(name).visibility = hideName; }

function showSlide(name) { refSlide(name).visibility = showName; }

function refSlide(name) {
  if (NN) { return document.layers[name]; }
  else { return eval('document.all.' + name + '.style'); }
  }

function changeSlide(offset) {
  hideSlide('slide' + curSlide);
```

```
curSlide = (curSlide + offset < 0 ? slideShow.length - 1 :
  (curSlide + offset == slideShow.length ? 0 : curSlide + offset));
showSlide('slide' + curSlide);
}
```

五个函数：genSlide()，refSlide()，hideSlide()，showSlide()和changeSlide()。它们做的工作和在第三章中的相同；如果你不清楚它们中的某一个是怎样运作的话，只要跳回上一章看一下就行了。实际上另外还有两个函数，imagePreLoad()和imageSwap()，它们也执行同样的操作，但是被修改了很多，值得我们重新作一点讨论。

预载图片

一个很大的Web应用范例会按照老规矩执行一些静态操作。如果你可以在运行中更好地控制它，那为什么不做得简单一点呢？这就是下面的代码对图片预载所做的贡献。在想要预载入图片，使其执行滚动效果的时候，典型做法是什么呢？可能会像这样：

```
var myImage1On = new Image();
myImage1On.src = 'images/myImgOn1.gif';
var myImage1Off = new Image();
myImage1Off.src = 'images/myImgOff1.gif';
```

非常简单。但要用四行代码来完成一对翻转图片。如果你有5对或10对图片呢？那就是20或40行代码。如果你想做一点改变，它马上就会变得乱七八糟。多种搜索引擎的界面介绍了一个同样实现预载的方法，不管你有多少对图片（理论上）。我们需要三个东西：

1. 为你所需的每套图片准备的一个*Image*对象数组。应用程序用一个数组来保存当鼠标箭头从链接上移过时所用的图片，一个数组保存鼠标箭头离开链接时的恢复图片。
2. 一个简单的图片命名协定。*myImg1On.gif/myImgOff1.gif*比较好用。参见第三章“JavaScript技巧：使用构造合理的命名协定”部分可以获得更多信息。命名协定必须在第一步中合并数组名。
3. eval()方法。

在第一步中，应用程序使用两个数组。一个被命名为*out*，包含那些鼠标箭头在链接

图片外面时翻转图片的*Image*对象。另一个被命名为*over*，包含的*Image*对象是那些当鼠标箭头移过链接图像时所用的翻转图片。现在，这些变量将用*arrayHandles*字符串数组来表示，第20行代码：

```
var arrayHandles = new Array('out', 'over')
```

在第二步，我们使用一个非常简单的命名协定。所有的图片对都有相同的前缀，后面跟着*out.jpg*或者*over.jpg*（根据图片而定）。例如，和InfoSeek相关的翻转图片被命名为*infoseekout.jpg*和*infoseekover.jpg*。

在第三步，我们首先迭代*arrayHandles*的每个元素，利用eval()来快速创建层，以保存*Image*对象。来看看第22－24行：

```
for (var i = 0; i < arrayHandles.length; i++) {
  eval('var ' + arrayHandles[i] + ' = new Array()');
  }
```

上边的*for*循环的执行功能和下面所示的代码是等价的：

```
var out    = new Array();
var over   = new Array();
```

为了完成预载，我们在函数preLoadImages()中再次用eval()来创建*Image*对象，并且将每一个的SRC属性赋值。位于第104～110行的函数体如下所示：

```
function imagePreLoad(imgName, idx) {
  for(var j = 0; j < arrayHandles.length; j++) {
    eval(arrayHandles[j] + "[" + idx + "] = new Image()");
    eval(arrayHandles[j] + "[" + idx + "].src = '" + imgPath +
      imgName + arrayHandles[j] + ".jpg'");
    }
  }
```

imagePreLoad()接收两个参数，一个名称前缀（比如*Infoseek*）和一个用来把相应数组元素赋值为一个新的*Image*对象的整数。再次用一个*for*循环迭代*arrayHandles*，利用每一个字符串元素来访问一个刚被创建的数组，并把唯一的一个索引值赋给它。例如，调用imagePreLoad('infoseek',0)等价于敲入下面的代码：

```
out[0]      = new Image();
out[0].src  = 'images/infoseekout.jpg';
over[0]     = new Image();
over[0].src = 'images/infoseekover.jpg';
```

但这是四行代码，正是我再三要注意避免的。每一次我需要一对翻转图片时，我可以调用preLoadImages()。而且这是个好办法，并不难。

JavaScript技巧：代码的重复使用

严格说来，这本身并不能算是JavaScript技巧。在任何一种语言中你都可以使用这种方法。如果以较高的标准来编程，而只在创建对象和函数的时候有例外情况，那么你会发现在不同的情况下可以使用相同的代码。考虑函数genSlide()，refSlide()，hideSlide()和showSlide()。它们执行非常基本但必须要有的操作。下面几条是有关说明。

- 用genSlide()创建跨浏览器的DHTML层。
- 用refSlide()引用跨浏览器的DHTML层。
- 用hideSlide()来隐藏跨浏览器的DHTML层。
- 用showSlide()显示跨浏览器的DHTML层。

思考一下我们从最后一章里的那些函数中所得到的东西。我们在这本书的后面还将看到它们。如果还没有的话，为你的可复用代码创建一个JavaScript源文件库。第六章将给出有关的所有内容。如果你设计了一个非常不错的函数或者对象，而且确定会再次用到，就把它放进一个.js文件以便以后访问。

启动你的引擎

第26～100行的*engines*变量表示一个数组，它的每个元素又包含由特殊搜索引擎信息作为元素构成的另外一个数组。变量*engines*有将近20个元素那么长，因此，让我们来看看第一个吧，如第27～29行所示：

```
new Array('HotBot',
          'http://www.hotbot.com/?MT=',
          'http://www.hotbot.com/'),
```

0元素确定搜索引擎的名字，HotBot。元素1利用搜索字符串确定URL，如果它包含搜索文字，就调用搜索引擎并返回结果页面。元素2表示搜索引擎主页的URL。如果用户提供一个空串（用空串来搜索），它将代替元素1发挥作用。

JavaScript技巧：别去想面向对象了

认真看了*engines*数组之后，你可能会想，我们为什么不用一个*searchEngine*构造器呢？难道这不适合用一个*searchEngine*构造器吗？你知道，如果函数就像这样：

```
function searchEngine(name, searchURL, homePage) {
  this.name = name;
  this.searchURL = searchURL;
  this.homePage = homePage;
  return this;
  }
```

那么引擎就会是这个样子：

```
var engines = new Array(
    new searchEngine('HotBot',
      'http://www.hotbot.com/?MT=',
      'http://www.hotbot.com/')
// etc., etc,
```

除了第102行的那个小技术外，这是我要采取的方法：

```
engines = engines.sort();
```

事实上，我想要按照字母顺序排列搜索引擎。用户应该希望自己能很快找到常用的搜索引擎。如果你采取面向对象方法，sort()方法将不会改变元素的顺序。但是第26～100行数组中的数组将基于每个数组的第一个元素被分类。这就是为什么搜索引擎的名称是每个新数组的第一个元素的原因。因为它可以马上生效，所以JavaScript将根据首元素来分类。对象本身其实没有首元素。这在第一章客户端搜索引擎里也是一样的。搜索结果按照字母顺序显示，所以所有的记录都用同样的方法编码。不要误会我的意思，我仍然是偏重面向对象的。这只是一个没有用到面向对象的应用程序而已。

engineLinks()

函数engineLinks()和第三章中的函数genScreen()类似，因为它负责层的创建。虽然如此，它们两者之间还是有差别的。看看第112～140行的代码。

层的管理

这个函数所做的第一件事是产生包含导航链接的层。

```
genLayer('sliderule', left - 20, top + 2, 25, engHgt, true,
  '<A HREF="javascript: changeSlide(1);" '+
  'onMouseOver="hideStatus(); return true;"><IMG SRC="' +
  imgPath + 'ahead.gif" BORDER=0></A><BR><A HREF="javascript: ' +
  'changeSlide(-1);" onMouseOver="hideStatus(); return true;">' +
  '<IMG SRC="' + imgPath + 'back.gif" BORDER=0></A>');
```

通过对genLayer()的调用来实现这个目的。这对你来说应该是稀松平常的。层中包含两个链接图片：一个向前箭头和一个向后箭头。注意，传递进来的左边缘和上边缘的像素值和所创建的引擎链接层左边和上边的位置（left-20和top+2）有关。

接下来，变量*lyrCount*确定所要创建的搜索引擎按钮层的数目，以每层所需的按钮数和分配给*engines*数组的引擎数为基础。这其实相当简单。将搜索引擎数（engines.length）除以每层（*perLyr*）要显示的引擎数。如果余数非0，则你还需要再加一个层。

我们来使用应用程序的值。Engines.length是20，*perLyr*为4。因此，变量*lyrcount*是5。如果我用了21个引擎，21/4=5.25。余数0.25表示还需要一个层，因此*lyrCount*将被设置为6。这里又一次给出了代码：

```
lyrCount = Math.ceil(engines.length / perLyr);
  Math.ceil(engines.length / perLyr);
```

带条件操作的执行和上面所述一致。如果余数为0，将*lyrCount*设为engines.length/perLyr；否则设为Math.ceil(engines.length/perLyr)。确定*lyrCount*是很重要的。一旦确定，第122～136行中engineLinks()将创建*lyrCount*个层：

```
for (var i = 0; i < lyrCount; i++) {
  var engLinkStr = '<TABLE BORDER=0 CELLPADDING=0 CELLSPACING=0><TR>';
  for (var j = 0; j < perLyr; j++) {
    var imgIdx  = (i * perLyr) + j;
    if (imgIdx >= engines.length) { break; }
    var imgName  = nameFormat(engines[imgIdx][0]);
    imagePreLoad(imgName, imgIdx);
    engLinkStr += '<TD><A HREF="javascript: ' +
      callSearch(document.forms[0].elements[0].value, ' + imgIdx +
      ');" onMouseOver="hideStatus(); imageSwap(\'' + imgName + '\', ' +
      imgIdx + ', 1); return true" onMouseOut="imageSwap(\'' + imgName +
```

```
        '\', ' + imgIdx + ', 0);"><IMG NAME="' + imgName + '" SRC="' +
        imgPath + imgName + 'out.jpg' + '" BORDER=0></A></TD>';
    }
```

在每一个层中，engineLinks()定义局部变量*engLinkStr*，它包含关于每张幻灯片的代码。如同你在123行所看到的一样，创建*engLinkStr*来定义用于装载图片的表，接下来，一个嵌套的*for*循环将用*perLyr*来控制循环变量，创建包含图片的表格单元。

对于每一个*perLyr*的循环体，局部变量*imgIdx*的值设为(i * perLyr) + j。这个表达式表示一个从0开始，在每个循环开始处加1的整数。*imgIdx*被用来确定图片前缀（它是*engines*数组中每个元素的0元素中的搜索引擎名称），然后像七文说的那样预载图片。表4-1在*perLyr*为4时给出了一个便捷的乘法计算。

表4-1 计算显示层（perLayer等于4）

i值	j的值域	(i * perLyr) + j（保持增量为1）
0	0, 1, 2, 3	0, 1, 2, 3
1	0, 1, 2, 3	4, 5, 6, 7
2	0, 1, 2, 3	8, 9, 10, 11
3	0, 1, 2, 3	12, 13, 14, 15
4	0, 1, 2, 3	16, 17, 18, 19

共有20个整数，0～19。

既然我们知道了*imgIdx*的值，我们应该确保自己没有为此大费周章。第126行代码对此作出处理：

```
if (imgIdx == engines.length) { break; }
```

由于*imgIdx*在每一次循环中都无条件地增加，一旦到达engines.length，就再也没有其他搜索引擎要显示了，因此函数会退出*for*循环。

JavaScript 技巧：数学计算与内存

为了替代一些不必要的表达式如(i * perLyr) + j，为什么不把一个变量(已命名的，例如 *count*) 设为 0，在每次循环中增加，就像 ++count 一样呢？当然，其实你的确可以这样做。但是何必要为大量的变量定义而分配更多的内存呢？即使它是局部的。

JavaScript 在 *i*、*perLyr*、*j* 中已经有了所需的变量去执行相应的计算。看起来这里就像不够完美一样，但是它可以为大的应用程序节约珍贵的内存空间。

预载图片

现在可以为每一个搜索引擎预载图片对了。在此之前，我们必须知道图片前缀。很简单，前缀就是搜索引擎名称的小写字母形式。也就是说，“InfoSeek”图片前缀是 *infoseek*；HotBot 图片前缀是 *hotbot*，如此类推。利用变量 *imgIdx* 在第 127 行确定正确的图片前缀：

```
var imgName = nameFormat(engines[imgIdx][0]);
```

engines 中每一个数组的 0 元素包含搜索引擎名称。变量 *imgIdx* 确定 *engines* 中正确的元素索引，返回相应搜索引擎的名称。所有剩下的事是将所有字母转变成小写。函数 nameFormat() 在第 158～161 行处理这一事件：

```
function nameFormat(str) {
  var tempArray = str.split(' ');
  return tempArray.join('').toLowerCase();
  }
```

按空格将字符串传递到数组元素中，就可以把字符串分隔开，并且删除了空格，然后再把字符串连接好。现在，*imgName* 有一个由小写字母组成的可以带空格的图片前缀。在第 128 行，它和 *imgIdx* 一起准备传递到 imagPreload() 中去。

创建链接

现在可以为每一个搜索引擎创建带有适当翻转代码的链接图片了。我们来进入第 129～135 行代码：

```
engLinkStr += '<TD><A HREF="javascript: ' +
  'callSearch(document.forms[0].elements[0].value, ' + imgIdx + ');" ' +
  'onMouseOver="hideStatus(); imageSwap(\'' + imgName + '\', ' + imgIdx +
  ', 1); return true" onMouseOut="imageSwap(\'' + imgName + '\', ' +
  imgIdx + ', 0);"><IMG NAME="' + imgName + '" SRC="' + imgPath +
  imgName + 'out.jpg' + '" BORDER=0></A></TD>';
```

我们考虑一下这个问题。每个搜索引擎链接有同样的四个要求：

1. 用户点击图片时用于调用相应搜索引擎的代码
2. 处理图片翻转的 *onMouseOver* 事件操作代码
3. 处理图片恢复的 *onMouseOut* 事件操作代码
4. 一个被设置为相应图片路径的有唯一 NAME 和 SRC 属性的 IMG 标签

分析设为 *engLinkStr* 的字符串，就可以弄明白每一个条件是怎样得到满足的。

第一个条件由下面的代码所满足：

```
HREF="javascript: callSearch(document.forms[0].elements[0].value, ' +
  imgIdx + ');"
```

你可以看到，创建的可点击链接将调用函数 callSearch()，document.forms[0].elements[0].value 连同 *imgIdx* 变量的相应值将传递到函数中去。我们很快还要具体地提到 callSearch()。现在，我们可以说第一个条件已经不成问题了。

第二个条件通过下面的代码得到满足：

```
'onMouseOver="hideStatus(); imageSwap(\'' + imgName + '\', ' + imgIdx +
  ', 1); return true" ' +
```

以上的代码负责创建对 hideStatus() 的调用，以清除状态栏中讨厌的 URL，然后，对 imageSway() 的调用传递三个必需的参数 *imgName*，*imgIdx* 和一个 *arrayHandles* 中元素的相应整数（1）。

第三个条件是这样来满足的：

```
'onMouseOut="imageSwap(\'' + imgName + '\', ' + imgIdx + ', 0);">' +
```

并没有多少变化。唯一值得一提的差别是，传递的整数是 0 而不是 1。

现在来看第四个条件的情况：

```
'<IMG NAME="' + imgName + '" SRC="' + imgPath + imgName + 'out.jpg' +
  '" BORDER=0></A></TD>';
```

每张图片的名称被设为*imgName*的值。这也就是函数imageSwap()引用它的方法。SRC属性被设置为*imgPath*，*imgName*和*out.jpg*的串联。由于图片会在鼠标箭头在*out*位置时开始出现，SRC标签就设置为和*out.jpg*子串所对应的图片。例如，打开的HotBot图片被定位于*images/hotbotout.jpg*。

第137～138行的代码作为结束。

```
engLinkStr += '</TR></TABLE>';
genLayer('slide' + i, left, top, engWdh, engHgt, false, engLinkStr);
```

也就是说，*engLinkStr*接收HTML来结束表格，剩下的事情是用genLayer()创建层。注意，这里所有对genLayer()的调用都传递false。还记得吗？传递false值将把显示的层隐藏。在页面被载入之前，所有的层都是隐藏的。在第201行的事件处理方法*onLoad*中，*slide0*被显示。

imageSwap()

你在第三章看到过它，但在这里有些不同。请看一下第179～182行：

```
function imageSwap(imagePrefix, imageIndex, arrayIdx) {
  document[imagePrefix].src = eval(arrayHandles[arrayIdx] + "[" +
  imageIndex + "].src");
  }
```

这个函数通常执行图片翻转。参数*imagePrefix*确定转换图片的来源。参数*imageIndex*和*arrayIdy*是用来正确访问*arrayHandles*中的图片对象的两个整数。

callSearch()

HTML和层就位后，用户只需要输入搜索文字，点击搜索引擎选项就可以了。当用户点击一幅搜索引擎图片的时候，函数callSearch()得到调用。下面是第184～193行的内容：

```
function callSearch(searchTxt, idx) {
```

JavaScript 技巧：escape()和 unescape()的应用

Escape()是一个 JavaScript 的内置函数，它将字符串中非文本和数字的符号转变为对应的十六进制。这就确保了保留的和允许的字符不会扰乱字符串的处理。例如，&记号用来分隔表单字段中的名－值对。因此，每个用户输入的&记号被%26所代替。Escape()一般用来对你要作为URL搜索字符串的一部分提交的字符串做格式化处理。无论什么时候你提交了一个表单，你的浏览器会为你处理这些东西。由于应用程序不用表单的提交，那么一定要添加符号转换功能。

unescape()在写 cookies 的时候非常方便适用。加号（+）和等号（=）用来给 cookies 属性赋值，比如 *name*，*domain* 和 *expires*。和你想像的一样，unescape()函数将十六进制表示法转变回它们原来的 ASCII 码。

```
if (searchTxt == "") {
  parent.frames[2].location.href = engines[idx][2] +
    escape(searchTxt); 0
  }
else {
  parent.frames[2].location.href = engines[idx][1] +
    escape(searchTxt);
  }
}
```

callSearch()需要两个参数。*searchTxt* 包含在文本域中输入的文字，*idx* 包含与 engines 数组中的搜索引擎信息相对应的整数。应用程序在 frames[2]中载入一个或两个文档。如果用户没有输入搜索文字，frames[2]就载入搜索引擎的默认主页。URL 包含在engines中每一个数组的元素2中。但是，如果用户输入了搜索文字，应用程序将载入带有 URL 和搜索引擎查询字符串的 frames[2]，还包括用户输入的文本。

你可能在猜测，*engines*的每一个数组中的元素1里那么长的搜索字符串我是从哪里得到的。我可能在哪儿得到这些值呢？

实际上，我查看每个搜索页面的源代码，基于用来提交搜索文字的HTML表单而构成查询字符串。我们从一个简单的例子开始。MusicSearch.com 有一个单一的文本域用于搜索。表单的 ACTION 属性是 http://www.musicsearch.com/global/search/search.cgi。域名为 *QUERY*。因此，带搜索字符串的 URL 应该是这样的：

```
http://www.musicsearch.com/global/search/search.cgi?QUERY= +
  escape(searchTxt);
```

这真是太简单了。只有一个名称－值对。但是搜索引擎可以拥有足够的选项。考虑元搜索引擎（一个搜索其他机构的数据库的搜索引擎）SavvySearch。使用SavvySearch，输入搜索文字，然后就可以用复选框来选择所要搜索的媒体，比如搜索引擎，新闻组，等等。也可以利用逻辑搜索规则，将结果数设置为从每个数据库的返回值，然后选择要显示的每个搜索结果相关信息的数量。

SavvySearch表单的ACTION属性为http://numan.cs.colostate.edu:1969/nph-search。这里是一个所需表单元素的列表：

- 逻辑搜索的选项列表名称为*Boolean*
- 结果数目选项列表的名称为*Hits*
- 结果数量的单选钮名称为*df*
- 文本域名为*KW*

我把*Boolean*选项列表的值定为OR，*Hit*选项列表值为10，结果显示按钮*df*的值为normal，还有搜索文本域*KW*的值设为escape(searchTxt)。值OR，10，还有normal可不是我创造的哦。它们要么是选择列表中的选项值，要么是单选钮的值，这些值都在HTML源代码中。

表单还包含两个隐藏域，一个名为*Mode*，另一个名为*AutoStep*。*Mode* HIDDEN域的值为MakePlan。HIDDEN域*AutoStep*的值为on。我不能确定它们究竟是做什么的，但是那并不重要。你所要做的只是把它们加到搜索字符串中。提交一个搜索给SavvySearch，然后，需要下面的URL：

```
http://numan.cs.colostate.edu:1969/nph-search? ' +
  classic=on&Boolean=OR&Hits=10&Mode=MakePlan&df=normal&AutoStep=on&KW= +
  escape(searchTxt)
```

关于搜索字符串的“编译”还有一个好消息，那就是名称-值对的顺序是无关紧要的。只要它们在搜索字符串里，就一切正常。

应用程序扩展：增加用户控制

就像在前边所提到的，这个应用程序总是让用户受默认值的约束。也就是说，用户完全没有或者只有一点点权力来定制自己的搜索。需要用户输入的只有搜索字符串。你可以对此作适当的编码，使用户也能对一些功能产生影响，比如每页返回的结果数目，和每一个结果一起显示的信息数量，或者是否适用诸如AND，OR，LINK和NOT LINK之类的逻辑搜索规则。在这个部分，我用HotBot来举例。

扩展功能性的最简单方法可能是增加每页所显示的结果数目。你需要与搜索引擎每页结果数相对应的名称－值对。表4-2列出了几个搜索引擎的名称和可能取值。

表4-2 搜索引擎和用于确定结果数目的变量

搜索引擎	域名	可能取值	举例
HotBot	DC	10,25,50,100	DC=10
InfoSeek advanced Search	Numberresults	10,20,25,50	Numberresults=10
Scientific American	Docs	10,25,50,100	Docs=10
Yahoo!	N	10,20,25,50	n=10

我从每一个搜索页的源代码中得到这些域名。有一些域来自搜索引擎的高级搜索页面，因此*engines*数组中的URL可能不起作用。还要记住，结果数可能不能调整。搜索引擎程序员可以将它设置为一个恰当的数字。如果你在允许改变结果数目的搜索页面中没有看见选项列表，你可以和有关单位联系，询问结果（或者其他问题）是否可以修改。否则，你必须添加一些默认类型，它们对特定的搜索引擎将不传递到结果域中。

也要注意，可能的取值会在搜索引擎范围内变动。你必须用一定的代码来作控制，但这也并不是难事。利用下面的步骤为你的应用程序添加一个搜索结果选项列表。然后把作为添加其他控制的基本方针。

1. 将选项列表加到包含文本域的框架。
2. 在包含engine数组相应搜索引擎的结果域名的每一个新数组中加一个特定元素。

3. 将包含相应搜索引擎的可能取值的一个特定new Array()加在*engines*数组中的每一个新数组上。

4. 从搜索字符串中删除预先确定的名称－值对(如果名称－值对存在于搜索字符串中的话)。

5. 调整函数callSearch()中的代码，以正确连接每个搜索引擎的搜索字符串。

让我们来看看HotBot的例子。

第1步

添加选择列表并不成问题。在应用程序中为所有搜索引擎选择共同的值可能是一件明智的事。这里是使用了10，25，50和100的代码：

```
<SELECT NAME="docs">
<OPTION VALUE="10">10
<OPTION VALUE="25">25
<OPTION VALUE="50">50
<OPTION VALUE="100">100
</SELECT>
```

第2步

这样，*engines*数组中new Array()的每个实例用三个元素来定义一个搜索引擎：元素0是搜索引擎名称；元素1为搜索引擎的搜索字符串；元素2为搜索引擎主页。这里再次给出HotBot记录：

```
new Array('HotBot',
  'http://www.hotbot.com/?MT=',
  'http://www.hotbot.com/')
```

将元素3设置为HotBot的相关域的名称。回想一下先前的表格，域名为*DC*。现在，HotBot记录是这样的：

```
new Array('HotBot',
  'http://www.hotbot.com/?MT=',
  'http://www.hotbot.com/',
  'DC')
```

如果你的一个或多个搜索引擎没有你可以设置的结果数（一个名称－值对），就将元素3设为null。

第3步

既然你已经分别确定了它们的名称，就把另一个包含所有合法值的数组加进来。可以通过在元素4中定义新的数组来实现。再次引用前面的表格，HotBot记录大概就像这样：

```
new Array('HotBot',
  'http://www.hotbot.com/?MT=',
  'http://www.hotbot.com/',
  'DC',
  new Array(10, 25, 50, 100))
```

第4步

这一步只在元素2中的默认搜索字符串包含用于结果设置的名－值对时适用。下面是元素2中HotBot的搜索字符串：

```
http://www.hotbot.com/?MT=
```

因为没有*DC*，我们可以跳过第4步。但仅仅作为一个例子，对于*Scientific American*的搜索引擎不包含名称－值对（docs=100）。我们来看一看：

```
http://www.sciam.com/cgi-bin/search.cgi?' +
  'searchby=strict&groupby=confidence&docs=100&query=
```

You would need to take that out so it looks like this:

```
http://www.sciam.com/cgi-bin/search.cgi?' +
  'searchby=strict&groupby=confidence&query=
```

如果一个或多个搜索引擎没有可以设置的结果（一个名称－值对），就不要给元素4创建值。

第5步

要做的最后一项工作，是编制用来构造搜索字符串的代码，它是在给传递给搜索引擎之前完成的。你可以在函数callSearch()中做这些事，下面就是它的初始代码：

```
function callSearch(searchTxt, idx) {
  if (searchTxt == "") {
    parent.frames[2].location.href = engines[idx][2] +
      escape(searchTxt);
    }
  else {
    parent.frames[2].location.href = engines[idx][1] +
      escape(searchTxt);
    }
  }
```

如果用户在表单的文本输入域中什么也不输入，应用程序仍然会进入搜索引擎主页。因此*if*条件部分可以保持不变。我们需要关注的是*else*部分。

```
else {
  if(engines[idx][3] != null) {
    for (var i = 0; i < engines[idx][4].length; i++) {
```

```
        var selRef = parent.frames[4].document.forms[0].docs;
        if (selRef.options[selRef.selectedIndex].value =
          engines[idx][4][i].toString()) {
          parent.frames[2].location.href = engines[idx][1] +
            escape(searchTxt) + '&' + engines[idx][3] + '=' +
            engines[idx][4][i];
          return;
          }
        }
      parent.frames[2].location.href = engines[idx][1] +
        escape(searchTxt);
      }
```

下面的代码将恰当的名称－值对加到搜索字符串上：

```
parent.frames[2].location.href = engines[idx][1] +
  escape(searchTxt) + '&' + engines[idx][3] + '=' +
  engines[idx][4][i];
```

在这里，你所拥有的一切就是搜索引擎的URL，加上*searchTxt*，加上名称（engines[idx][3]），加上用户所选的值。但是，这仅只发生在满足两个条件之后。如果不能满足，搜索被设置为engines[idx][1]元素中所定义的默认值。首先，搜索引擎必须有一个你能修改的结果特征。如果是这样的话，输入域被定义为engines[idx][3]中的字符串。否则，元素设为null。这发生在第三步。下面的*if*语句用来检验engines[idx][3]不为null：

```
if(engines[idx][3] != null) {
```

如果值为null，则第一个条件不成立。因此，将使用默认的搜索字符串。如果engines[idx][3]不为null，则JavaScript迭代可接受值（在engines[idx][4]数组中定义）的数目。如果在selRef.options[selRef.selectedIndex]所表示的选择列表里选择的数字和数组中可接受值的某一个相匹配，JavaScript就将用户搜索引擎URL及搜索文字与名称－值对连接起来，然后将frame[2]同文档结果一起上载，执行结束。

如果循环迭代了所有可接受的值但一个匹配也没有得到，第二个条件就失败了，转而使用默认的搜索字符串。

第五章

ImageMachine 程序

应用程序要点
- 动态图片翻转代码发生器和浏览工具
- 适用于少数JavaScript浏览器的代码
- 灵活可扩展的HTML属性设置
- 支持MouseDown事件的图片翻转

JavaScript技巧
- JavaScript的开发维护
- 全局变量的作用
- JavaScript 1.1和1.2中的搜索和替换

不管你在看本章中的哪个部分，应用程序都要按照一个人的意志来设计，这个人就是用户。你知道，我们当然希望用户像旅鼠一样蜂拥而至，访问我们的站点，使我们的流量剧增，购买我们的货物，还下载我们的软件。这个应用程序却打破了这个模式。你所要面对的人将是：开发者，站点管理员，或者设计师。

尽管 DHTML 已经提高了鼠标箭头从一些框架、部件(Widget)、按钮和样式表上移过时的性能，但是图片翻转仍然是网络中最热门最广泛的应用技术之一。

用JavaScript来支持翻转并不需要多少艰深的技术，但它的确可以使你轻松地拥有这样的一个应用程序，它有着高效的格式化图片翻转代码。这样，我们这些编程人员可以恰当地把它加在我们的页面中。来看看 ImageMachine（图像控制机）程序。图 5-1 显示了打开 *ch05/index.html* 时的界面。

这个应用程序使用起来相当简单。你只要对你的图片作出一系列的规格设置。如图 5-1 所示：

1. 选择你所需要的图片对数目。

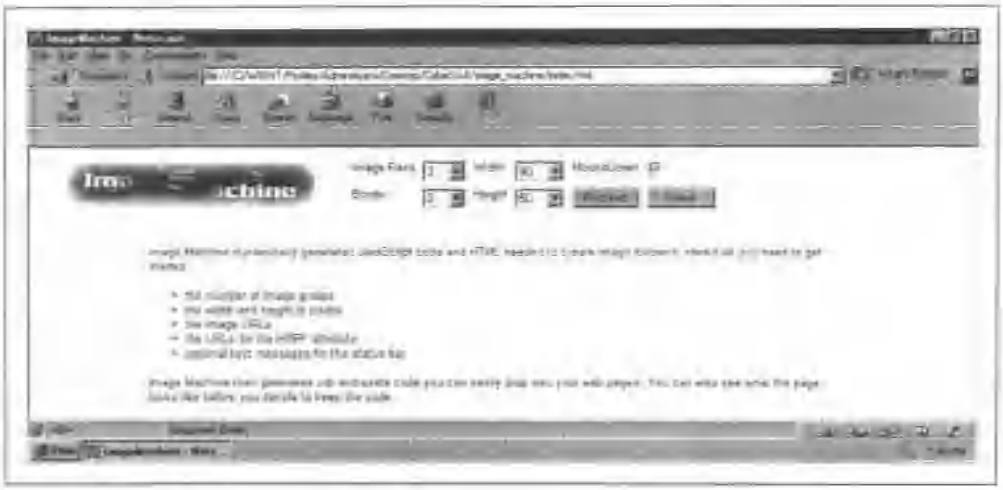


图 5-1 跃跃欲试的 ImageMachine

2. 设置所有图片的默认宽度，高度和边框值。(以后你可以作一些个人的调整。)
3. 如果你要对 *onMouseDown* 情况指定第三张图片，就点中“MouseDown”项。否则就让它空白。
4. 选择“Proceed（继续）”来进入下一步，或者选择“Reset（重置）”以重新设置。

一旦进入到这一步，ImageMachine 就会产生一个如图 5-2 所示的模板。

如果你没有选择“MouseDown”复选框，你将会获得为每一个图片组产生的两个 HTML 文本域：一个给基本图片，一个给翻转图片。如果你选择了“MouseDown”复选框，也会出现一个相应的文本域。每一张图片也从域中得来，进入 HREF 属性，这个属性用于每一个链接或者当用户把鼠标箭头从链接上移过时显示在状态栏中的状态文字。最后，三个小的文本域包含了每一个你用来作修改的图片组的默认宽度、高度和边框的值。执行最后几步后，就可以得到所需的代码：

1. 在文本域中输入每一个基本图片（鼠标箭头在任何其他地方时所显示的图片）的路径。它们是文件域而不是文本域，这是因为你的图片可能在你的本地机上。一旦你觉得代码的完成让你满意，就可以改变 URL 信息。如果你有翻转图片和 mousedown 图片的话，对它们作同样的处理。

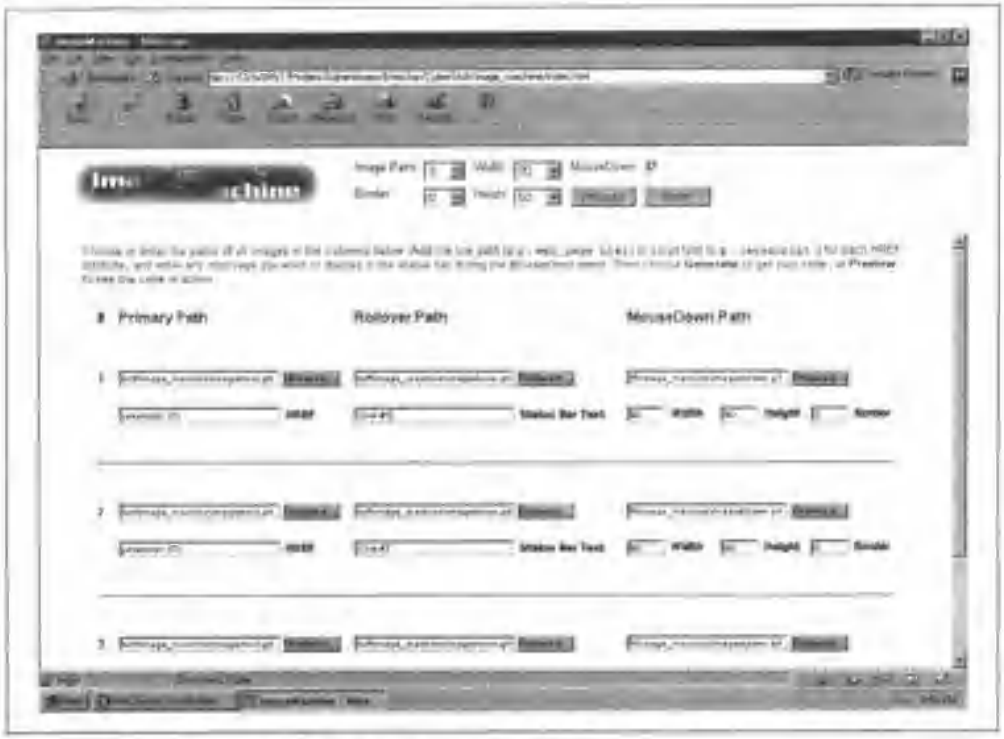


图 5-2 ImageMachine 根据你的选择产生一个定制模板

2. 在文本域中输入与每个图片组的 HREF 相关的绝对或相对 URL。
3. 在“Status Bar（状态条）”文本域中，输入任何你想在用户的鼠标箭头从链接上移过时显示在状态栏中的文字。
4. 在宽度，高度和边框属性各自的文本域中作任何的个人调整。
5. 选择“Generate（产生）”来查看代码，或选择“Preview（预览）”，以便检查代码在你的浏览器中运行时的版本。

图 5-3 显示了选择“Generate”后 imageMachine 的情况。查看 JavaScript 和 HTML，里面有详细的注解。注意，对于你所指定的每一个图片组，ImageMachine 产生用来预载图片和将它们设为翻转图片的代码。紧接着的 HTML 包括相应的 HREF 和 IMG 标签，将事件处理方法与图片属性完整载入。

图 5-3 查看你刚才所创建的代码

你会注意到在屏幕底部还有两个按钮。“Preview（预览）”可以使你看到运行的代码。“Change Info（修改信息）”使你可以回到模板作一些修改。图 5-4 给出了用来显示图片和已经预载入的翻转图片对的代码。

这些产生的代码最大的作用之一是依照所用浏览器中的JavaScript性能简化了相关的功能。换句话说，支持JavaScript 1.2及其以上版本的浏览器将执行*onMouseOver*和*onMouseOut*，以及*onMouseDown*的相关翻转。只支持 JavaScript 1.1 的浏览器将仅执行*onMouseOut*和*onMouseOver*相关翻转。只支持 JavaScript 1.0 的浏览器仅只执行用来设置状态栏文本的代码。

那么对较好的带宽来说又怎么样呢？只有确实能被用着的图片能被下载。如果浏览器没有使用它，就不会把它下载下来。JavaScript 1.1 浏览器不会预载任何与*onMouseDown*事件相关的图片。而 JavaScript 1.0 浏览器则不会预载任何图片！

图 5-4　选择“Preview”或“Change Info”获得更多选项

执行条件

虽然产生的代码能在所有支持 JavaScript 的浏览器中运行，但是，你作为一个开发者，必须使用支持 JavaScript 1.2 的浏览器。某些字符串置换和其他一些代码要求 1.2版本。在可伸缩性范围之内，可以创建和你的系统资源所能容纳的数目一样多的图片翻转。最大一般设为 50 组，这已经超过了我想在我的一个页面上拥有的数目。

顺便说一句，界面浏览至少要求 1024 × 768 的模式才能有较好的质量。图片模板和代码都要求页面有足够的宽度。

语法细则

在考虑任何代码之前，看一看程序流程可能会对你很有帮助。图 5-5 从头到尾给出

了基本流程。你基本上从创建图片表单和设置每个翻转规格开始。然后，你可以在预览，修改，还有产生代码中间前后跳转。当你看到你的结果是什么样子的时候，就可以拷贝和粘贴所产生的代码。

ImageMachine有三个文件：一个框架页和两个内容页面。框架页名为*index.html*，内容页名为*nav.html*和*base.html*。*index.html*没有JavaScript，也毫无惊人之处。你自己提提神再看吧，这里是所有的9行代码，如例5-1所示。

例5-1：index.html

```
<HTML>
<HEAD>
<TITLE>ImageMachine</TITLE>
</HEAD>
<FRAMESET ROWS="105, *" FRAMEBORDER="0" BORDER="0">
  <FRAME SRC="nav.html" NAME="nav" SCROLLING=NO>
  <FRAME SRC="base.html" NAME="base">
  </FRAMESET>
</HTML>
```

如果你看看*base.html*，就会看到许多静态HTML。在进入如例5-2所示的*nav.html*之前，要知道它对于理解你将要看到的代码是非常重要的。它很长（有400多行）并且不是那么容易弄明白，但也不是都很复杂。

例5-2：nav.html

```
<HTML>
<HEAD>
<TITLE>ImageMachine</TITLE>
<SCRIPT LANGUAGE="JavaScript1.2">

var platform = navigator.platform;
var lb = (platform.indexOf("Win" != -1) ? "\n\r" :
  (platform.indexOf("Mac" != -1) ? "\r" : "\n"));
var fontOpen = '<FONT COLOR=BLUE>';
var fontClose = '</FONT>';

function genSelect(name, count, start, select) {
  var optStr = "";
  for (var h = start; h <= count; h++) {
    optStr += "<OPTION VALUE=" + h +
      (h == select ? " SELECTED" : "") + ">" + h;
    }
  document.write("<SELECT NAME=" + name + ">" + optStr + "</SELECT>");
  }

function captureDefaultProfile(formObj) {
  setArrays();
```

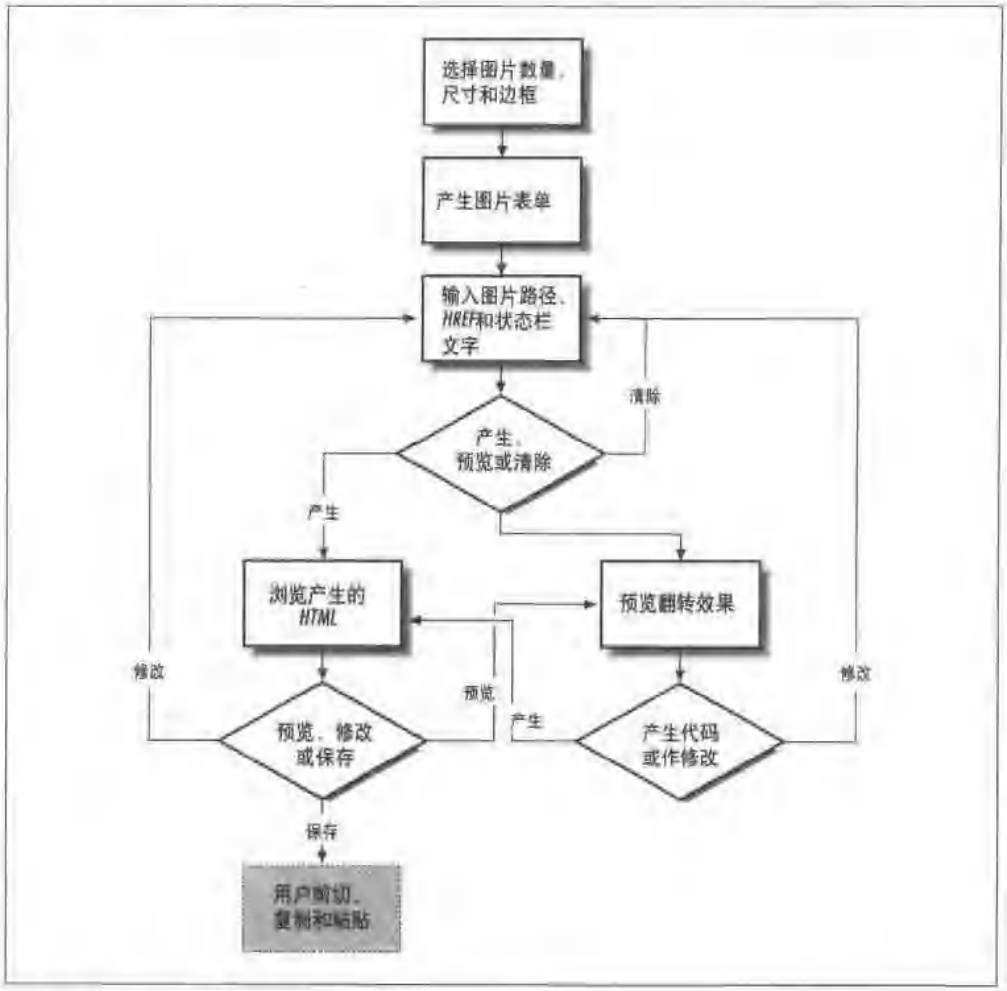


图 5-5 ImageMachine 流程图

```
23   imgDefaults    = formObj;
24   var imgQty      = (imgDefaults.imgnumber.selectedIndex + 1);
25   var imgHeight   = (imgDefaults.pxlheight.selectedIndex);
26   var imgWidth    = (imgDefaults.pxlwidth.selectedIndex);
27   var imgBorder   = (imgDefaults.defbdr.selectedIndex);
28   for (var i = 0; i < imgQty; i++) {
29     imgPrim[i] = "";
30     imgRoll[i] = "";
31     imgDown[i] = "";
32     imgLink[i] = "";
33     imgText[i] = "";
34     imgWdh[i]  = imgWidth;
```

```
35      imgHgt[i]   = imgHeight;
36      imgBdr[i]   = imgBorder;
37      }
38    generateEntryForm();
39    }
40
41  function setArrays() {
42    imgPrim = new Array();
43    imgRoll = new Array();
44    imgDown = new Array();
45    imgLink = new Array();
46    imgTxt  = new Array();
47    imgWdh  = new Array();
48    imgHgt  = new Array();
49    imgBdr  = new Array();
50    }
51
52
53  function generateEntryForm() {
54    with(parent.frames[1].document) {
55    open();
56      writeln("<HTML><BODY BGCOLOR=FFFFEE><FONT FACE=Arial SIZE=2>" +
57        "<BLOCKQUOTE>Choose or enter the paths of all images in the " +
58        "columns below. Add the link path (e.g., <FONT FACE=Courier>" +
59        "web_page.html</FONT>) or script text (e.g., <FONT FACE=Courier>" +
60        "javascript:</FONT>) for each HREF attribute, and  enter any " +
61        "message you want to display in the status bar during the " +
62        "<FONT FACE=Courier>MouseOver</FONT> event. Then choose " +
63        "<B>Generate</B> to get your code, or <B>Preview</B> to see the " +
64        "code in action.</BLOCKQUOTE><FORM NAME='imgProfile' " +
65        "onSubmit='return false;'><CENTER><TABLE BORDER=0 ALIGN=CENTER " +
66        "CELLSPACING=5 CELLPADDING=5><TH ALIGN=LEFT><FONT FACE=Arial>#" +
67        "<TH ALIGN=LEFT><FONT FACE=Arial>Primary Path" +
68        "<TH ALIGN=LEFT><FONT FACE=Arial>Rollover Path" +
69        (imgDefaults.mousedown.checked ? "<TH ALIGN=LEFT>" +
70        "<FONT FACE=Arial>MouseDown Path" : "") +
71        "<TR><TD><BR></TD></TR>");
72      }
73
74    for (i = 0; i < imgPrim.length; i++) {
75      with(parent.frames[1].document) {
76        writeln("<TR>" +
77        "<TD><FONT FACE=Arial SIZE=2><CENTER><B>" + (i + 1) +
78        "</B></CENTER><TD VALIGN=BOTTOM><FONT FACE=Arial SIZE=2>" +
79        "<INPUT TYPE=FILE NAME='prim" + i + "' VALUE='" + imgPrim[i] +
80        "'><TD VALIGN=BOTTOM><FONT FACE=Arial SIZE=2><INPUT TYPE=FILE " +
81        "NAME='seci" + i + "' VALUE='" + imgRoll[i] + "'>" +
82        (imgDefaults.mousedown.checked ? "<TD VALIGN=BOTTOM><FONT " +
83        "FACE=Arial SIZE=2><INPUT TYPE=FILE NAME='down" + i + "' VALUE='" +
84        imgDown[i] + "'>" : "") + "<TR><TD VALIGN=BOTTOM><FONT " +
85        "FACE=Arial SIZE=2>   </TD>" +
86        "<TD VALIGN=BOTTOM><FONT FACE=Arial SIZE=2><INPUT TYPE=TEXT " +
87        "NAME='href" + i + "' VALUE='" + imgLink[i] + "'> " +
```

```
88        "<IMG SRC='images/href.jpg'><TD VALIGN=BOTTOM><FONT FACE=Arial " +
89        "SIZE=2><INPUT TYPE=TEXT NAME='stat" + i + "' VALUE='" +
90        imgText[i] + "'><IMG SRC='images/statusbar.jpg'> " +
91        (!imgDefaults.mousedown.checked ?"<TR>" : "") +
92        "<TD VALIGN=BOTTOM><FONT FACE=Arial SIZE=2>" +
93        (!imgDefaults.mousedown.checked ?
94        "</TD><TD VALIGN=BOTTOM><FONT FACE=Arial SIZE=2>" : "") +
95        "<INPUT TYPE=TEXT NAME='wdh" + i + "' VALUE='" +
96        imgWdh[i] + "' SIZE=3> <IMG SRC='images/wdh.jpg'> " +
97        "   <INPUT TYPE=TEXT NAME='hgt" + i + "' VALUE='" +
98        imgHgt[i] + "' SIZE=3><IMG SRC='images/hgt.jpg'>  " +
99        (!imgDefaults.mousedown.checked ?
100       "<TD VALIGN=BOTTOM><FONT FACE=Arial SIZE=2>" : "") +
101       "<INPUT TYPE=TEXT NAME='bdr" + i + "' VALUE='" + imgBdr[i] +
102       "' SIZE=3> <IMG SRC='images/bdr.jpg'>" +
103       "<TR><TD VALIGN=BOTTOM COLSPAN=" +
104       (!imgDefaults.mousedown.checked ? "3" : "4") +
105       "><BR><HR NOSHADE><BR></TD></TR>");
106       }
107     }
108
109   with(parent.frames[1].document) {
110     writeln("</TABLE><CENTER><INPUT TYPE=BUTTON " +
111       "onClick='parent.frames[0].imgValid8(this.form, true);' " +
112       "VALUE='Generate'><INPUT TYPE=BUTTON " +
113       "onClick='parent.frames[0].imgValid8(this.form, false);' " +
114       "VALUE='Preview'> <INPUT TYPE=RESET VALUE='   Clear   '>" +
115       "</FORM></BODY></HTML>");
116     close();
117     }
118   }
119
120 function imgValid8(imgTemplate, mimeType) {
121   for (var  i = 0; i < imgPrim.length; i++) {
122     if (imgTemplate['prim' + i].value == "" ||
123       imgTemplate['seci' + i].value == "" ||
124       imgTemplate['href' + i].value == "") {
125       alert("All images and HREF attributes must have URLs.");
126       return;
127       }
128     if (imgDefaults.mousedown.checked) {
129       if(imgTemplate['down' + i].value == "") {
130         alert("All images and HREF attributes must have URLs.");
131         return;
132         }
133       }
134     }
135   genJavaScript(imgTemplate, mimeType);
136   }
137
138 function genJavaScript(imgTemplate, mimeType) {
139   imageLinks = '';
140
```

```
141   if (mimeType) {
142     lt       = "&lt;";
143     gt       = "&gt;";
144     br       = "<BR>";
145     HTML     = true;
146     nbsp     = "     ";
147     }
148   else {
149     lt       = "<";
150     gt       = ">";
151     br       = lb;
152     HTML     = false;
153     nbsp     = "     ";
154     }
155
156   if(imgTemplate != null) {
157     setArrays();
158     for (var i = 0; i < (imgDefaults.imgnumber.selectedIndex + 1); i++) {
159       imgPrim[i] = purify(imgTemplate['prim' + i].value);
160       imgRoll[i] = purify(imgTemplate['seci' + i].value);
161       if (imgDefaults.mousedown.checked) {
162         imgDown[i] = purify(imgTemplate['down' + i].value);
163         }
164       imgLink[i] = purify(imgTemplate['href' + i].value);
165       imgText[i] = purify(imgTemplate['stat' + i].value);
166       imgWdh[i]  = purify(imgTemplate['wdh' + i].value);
167       imgHgt[i]  = purify(imgTemplate['hgt' + i].value);
168       imgBdr[i]  = purify(imgTemplate['bdr' + i].value);
169       }
170     }
171
172   if (HTML) {
173     primJavaScript = "<HTML><HEAD><TITLE>Image Machine Code</TITLE>" +
174       "</HEAD><BODY BGCOLOR=FFFFEE><FONT FACE=Arial>" +
175       "<I>Cut and paste the code below into an HTML file. The blue " +
176       "code represents information you provided.</I>" +
177       "<BR><BR></FONT><FONT SIZE=2 FACE=Arial>" +
178       lt + "HTML" + gt + "<BR>" + lt + "HEAD" + gt + "<BR>" +
179       lt + "TITLE" + gt + "Image Machine Code" + lt + "/TITLE" + gt;
180     }
181   else {
182     primJavaScript = "<HTML><HEAD><TITLE>Image Machine Code</TITLE>";
183     }
184
185   primJavaScript += br + br + lt + "SCRIPT LANGUAGE=\"JavaScript\"" +
186     gt + br + br + "// Define global variables in JavaScript 1.0" + br +
187     "var canRollOver = false;" + br + "var canClickDown = false;" + br +
188     br + lt + "/SCR" + "IPT" + gt + br + br + lt +
189     "SCRIPT LANGUAGE =\"JavaScript1.1\"" + gt + br + br +
190     "// Change canRollOver to true in JavaScript 1.1" + br +
191     "canRollOver = true;" + br + br;
192
193   secJavaScript = lt + "SCRIPT LANGUAGE=\"JavaScript1.2\"" + gt + br +
```

```
194     br + "// Change canClickDown to true in JavaScript 1.2" + br +
195     "canClickDown = true;" + br + br;
196
197   for (var j = 0; j < imgPrim.length; j++) {
198     primJavaScript += "// Primary and rollover image sources #" +
199       (j + 1) + br +"switch" + (j + 1) + "out = new Image(" +
200       (HTML ? fontOpen : "") + imgWdh[j] +
201       (HTML ? "</FONT>," : ", ") +
202       (HTML ? fontOpen : "") + imgHgt[j]  +
203       (HTML ? fontClose : "") + "); " + br + "switch" + (j + 1) +
204       "out.src = '" +
205       (HTML ? fontOpen : "") +
206       (imgPrim[j].indexOf(":\\") != -1 ? pathPrep(imgPrim[j]) :
207         imgPrim[j]) +
208       (HTML ? fontClose : "") + "';" + br + "switch" + (j + 1) +
209       "over = new Image(" +
210       (HTML ? fontOpen : "") + imgWdh[j]  +
211       (HTML ? "</FONT>," : ", ") +
212       (HTML ? fontOpen : "") + imgHgt[j] +
213       (HTML ? fontClose : "") + "); " + br + "switch" + (j + 1) +
214       "over.src = '" +
215       (HTML ? fontOpen : "") +
216       (imgRoll[j].indexOf(":\\") != -1 ? pathPrep(imgRoll[j]) :
217         imgRoll[j]) +
218       (HTML ? fontClose : "") + "';" + br + br;
219
220     if (imgDefaults.mousedown.checked) {
221       secJavaScript += "// MouseDown image source #" + (j + 1) + br +
222         "switch" + (j + 1) + "down = new Image(" +
223         (HTML ? fontOpen : "") + imgWdh[j] +
224         (HTML ? "</FONT>," : ", ") +
225         (HTML ? fontOpen : "") + imgHgt[j]  +
226         (HTML ? fontClose : "") + "); " + br + "switch" +
227         (j + 1) + "down.src = '" +
228         (HTML ? fontOpen : "") +
229         (imgPrim[j].indexOf(":\\") != -1 ? pathPrep(imgDown[j]) :
230           imgDown[j]) +
231         (HTML ? fontClose : "") + "';" + br + br;
232       }
233
234     imageLinks += lt + "!-- <I> Image Link #" + (j + 1) +
235       " </I>//--" + gt + br + lt + "A HREF=\"" +
236       (HTML ? fontOpen : "") + imgLink[j] +
237       (HTML ? fontClose : "") + "\" " + br + nbsp +
238       "onMouseOver=\"imageSwap('switch" + (j + 1) +
239       "', 'over', false); display('" +
240       (HTML ? fontOpen : "") + imgText[j] +
241       (HTML ? fontClose : "") + "'); return true;\"" + br +
242       nbsp + "onMouseOut=\"imageSwap('switch" +
243       (j + 1) + "', 'out', false); display('');\"" +
244       (imgDefaults.mousedown.checked ?
245         br + nbsp + "onMouseDown=\"isDown=!isDown; imageSwap('switch" +
246         (j + 1) + "', 'down', true);\"" : "") +
247       gt + br + lt + "IMG SRC=\"" +
```

```
248       (HTML ? fontOpen : "") + pathPrep(imgPrim[j]) +
249       (HTML ? fontClose : "") + "\"" + br + nbsp +
250       "NAME=switch" + (j + 1) + br + nbsp + "WIDTH=" +
251       (HTML ? fontOpen : "") + imgWdh[j] +
252       (HTML ? fontClose : "") + br + nbsp + "HEIGHT=" +
253       (HTML ? fontOpen : "") + imgHgt[j] +
254       (HTML ? fontClose : "") + br + nbsp + "BORDER=" +
255       (HTML ? fontOpen : "") + imgBdr[j] +
256       (HTML ? fontClose : "") +
257       gt + "" + lt + "/A" + gt + br + br + br;
258     }
259
260   scriptClose = br + lt + "/SCR" + "IPT" + gt + br + br;
262   swapCode = br + lt + "/SCR" + "IPT" + gt + br + br +
263     lt + "SCRIPT LANGUAGE =\"JavaScript\"" + gt + br + br +
264     (imgDefaults.mousedown.checked ?
265       "var isDown = false;" + br + br : "") +
266     "// Conditionally perform the rollovers in JavaScript 1.0" + br +
267     "function imageSwap(imageName, imageSuffix) {" + br +
268     nbsp + "if (!canRollOver) { return; }" + br + nbsp +
269     (imgDefaults.mousedown.checked ?
270       "if (!isDown) { " + br + nbsp + nbsp : "") +
271     "document[imageName].src = " +
272     "eval(imageName + imageSuffix + \".src\");" + br + nbsp +
273     (imgDefaults.mousedown.checked ? nbsp + "}" + br + nbsp +
274     "else if (canClickDown) {" + br +
275     nbsp + nbsp + "document[imageName].src = " +
276     eval(imageName + imageSuffix + \".src\");" + br +
277     nbsp + nbsp + "}" + br + nbsp : "") + "}" + br + br +
278     "function display(stuff) { window.status = stuff; }" +
279         br + br + lt + "/SCR" + "IPT" + gt + br;
280
281   primHTML = br + lt + "/HEAD" + gt + br +
282     lt + "BODY BGCOLOR=FFFFEE" +
283     gt + br + br + (HTML ? "<FONT COLOR=RED>" : "") + lt +
284     "!-- <I> Image Code Begins </I> //--" + gt + br +
285     (HTML ? fontClose : "") + br + br;
286
287   secHTML = (HTML ? "<FONT COLOR=RED>" : "") +
288     lt + "!-- <I> Image Code Ends</I> //--" + gt +
289     (HTML ? fontClose : "") + br + br +
290     (HTML ? lt + "/BODY" + gt + br + lt + "/HTML" + gt : "") +
291     br + br + "<CENTER><FORM>" + br +
292     "<INPUT TYPE=BUTTON onClick='parent.frames[0].genJavaScript(null, " +
293     (HTML ? "false" : "true") + ");' VALUE='" +
294     (HTML ? 'Preview' : 'Generate') + "'>   " +
295     "<INPUT TYPE=BUTTON " +
296     "onClick='parent.frames[0].generateEntryForm();' " +
297     "VALUE='Change Info'>" + br + "</FORM></CENTER>" + br + br +
298     "</BODY></HTML>";
299
300   agregate = primJavaScript +
301     (imgDefaults.mousedown.checked ? scriptClose + secJavaScript : "") +
```

```
302     swapCode + primHTML + imageLinks + secHTML;
303
304   parent.frames[1].location.href =
305     "javascript: parent.frames[0].agregate";
306   }
307
308 function purify(txt) { return txt.replace(/\'|\"/g, ""); }
309
310 function pathPrep(path) {
311   if (path.indexOf(":\\") != -1) {
312     path = path.replace(/\\/g, "/");
313     path = path.replace(/:\//, "|/");
314     return "file:///" + path;
315     }
316   else { return path; }
317   }
318
319 </SCRIPT>
320 </HEAD>
321 <BODY BGCOLOR=FFFFEE>
322 <FORM>
323 <TABLE BORDER="0">
324   <TR>
325     <TD VALIGN=MIDDLE>
326     <IMG SRC="images/image_machine.gif" WIDTH=275 HEIGHT=56 HSPACE=25>
327     </TD>
328     <TD>
329     <!-- Create a the default template //-->
330     <TABLE BORDER="0" ALIGN="CENTER">
331       <TR>
332         <TD VALIGN="TOP">
333         <FONT FACE="Arial" SIZE=2>
334         Image Pairs
335         </TD>
336         <TD VALIGN="TOP">
337         <FONT FACE="Arial" SIZE=2>
338         <SCRIPT LANGUAGE="JavaScript1.2">
339         <!--
340         genSelect("imgnumber", 50, 1, 1);
341         //-->
342         </SCRIPT>
343         </TD>
344         <TD VALIGN="TOP">
345         <FONT FACE="Arial" SIZE=2>
346         Width
347         </TD>
348         <TD VALIGN="TOP">
349         <FONT FACE="Arial" SIZE=2>
350         <SCRIPT LANGUAGE="JavaScript1.2">
351         <!--
352         genSelect("pxlwidth", 250, 0, 90);
353         //-->
354         </SCRIPT>
```

```
355          </TD>
356          <TD VALIGN="TOP">
357          <FONT FACE="Arial" SIZE=2>
358          MouseDown
359          </TD>
360          <TD VALIGN="TOP">
361          <FONT FACE="Arial" SIZE=2>
362          <INPUT TYPE=CHECKBOX NAME="mousedown">
363          </TD>
364      </TR>
365      <TR>
366          <TD VALIGN="TOP">
367          <FONT FACE="Arial" SIZE=2>
368          Border
369          </TD>
370          <TD VALIGN="TOP">
371          <FONT FACE="Arial" SIZE=2>
372          <SCRIPT LANGUAGE="JavaScript1.2">
373          <!--
374          genSelect("defbdr", 10, 0, 0);
375          //-->
376          </SCRIPT>
377          </TD>
378          <TD VALIGN="TOP">
379          <FONT FACE="Arial" SIZE=2>
380          Height
381          </TD>
382          <TD VALIGN="TOP">
383          <FONT FACE="Arial" SIZE=2>
384          <SCRIPT LANGUAGE="JavaScript1.2">
385          <!--
386          genSelect("pxlheight", 250, 0, 50);
387          //-->
388          </SCRIPT>
389          </TD>
390          <TD VALIGN="TOP">
391          <FONT FACE="Arial" SIZE=2>
392          <INPUT TYPE=BUTTON VALUE="Proceed"
393            onClick="captureDefaultProfile(this.form);">
394          </TD>
395          <TD VALIGN="TOP">
396          <FONT FACE="Arial" SIZE=2>
397          <INPUT TYPE=RESET VALUE="  Reset  ">
398          </TD>
399        </TR>
400      </TABLE>
401      </TD>
402    </TR>
403 </TABLE>
404 </CENTER>
405 </FORM>
406 </BODY>
407 </HTML>
```

这是迄今为止我们所见过的应用程序中最常用的代码。它看上去有些混乱，但是并没有我们想像的那么糟糕。为了透彻地理解ImageMachine，让我们从典型的用户角度看一看应用程序。请看下列五个步骤：

1. 加载页面。
2. 用户输入图片对和默认值，然后选择“Proceed”。
3. 用户填写图片路径，HREF属性等内容，然后选“Generate（生成）”查看代码。
4. 用户选“Preview（预览）”查看运行代码。
5. 用户选“Change Info（修改信息）”进行修改。

第1步：加载页面

所有的东西看起来都相当普通。一个名为*index.html*的框架页，两个分别名为*nav.html*和*base.html*的框架内容页面。但是，JavaScript在用户能够做出动作之前要准备做一些工作。我要提醒你注意第323～403行的代码。在这里，你将会看到一个表格的代码，表格数据单元中有好几个JavaScript的函数调用：

```
<TD VALIGN="TOP">
<FONT FACE="Arial" SIZE=2>
Image Pairs
</TD>
<TD VALIGN="TOP">
<FONT FACE="Arial" SIZE=2>
<SCRIPT LANGUAGE="JavaScript1.2">
<!--
genSelect("imgnumber", 50, 1, 1);
//-->
</SCRIPT>
</TD>
<TD VALIGN="TOP">
<FONT FACE="Arial" SIZE=2>
Width
</TD>
<TD VALIGN="TOP">
<FONT FACE="Arial" SIZE=2>
<SCRIPT LANGUAGE="JavaScript1.2">
<!--
genSelect("pxlwidth", 250, 0, 90);
//-->
</SCRIPT>
</TD>
```

对genSelect()的调用使用JavaScript来创建选项列表。每一个选项列表可用于对图片属性设置默认值。在这里，选项列表比文本域更适用，因为你不必担心要作那么多的表单确认。用户不可能对边框或图片宽度输入一个不正确的值（比如一个非数字值），因为他必须从列表所给出的选项中作出选择。但是有谁愿意苦苦地敲入有250或者300个不同选项的列表，每一项一个数字？假设你必须修改选项数量，JavaScript可以通过对每一个我们要编辑的选项列表调用genSelect()来帮你的忙。看一下第12～19行代码：

```
function genSelect(name, count, start, select) {
  var optStr = "";
  for (var h = start; h <= count; h++) {
    optStr += "<OPTION VALUE=" + h +
      (h == select ? " SELECTED" : "") + ">" + h;
    }
  document.write("<SELECT NAME=" + name + ">" + optStr + "</SELECT>");
  }
```

函数genSelect()需要四个参数——包含选项列表名称的一个字符串，最大选项数目，显示的开始数字（然后就以1为增量递增），还有用来指定所选项的数字。GenSelect()从头开始迭代和计算，创建一个<OPTION>标签的字符串。这些事完成之后，JavaScript将这个字符串写在<SELECT>标签中间，然后放到文档中去。现在页面已经载入，并且准备就绪。让我们来看看用户输入默认值后将会发生什么情况。

第2步：输入图片对和默认值

在图5-1中我们注意到用户设置包括四个选择列表和一个复选框的默认值。最重要的设置是图片对的数目。ImageMachine可用于选择1～50个图片对。我怀疑你根本就不会需要50个，但是容量大一点对谁都不会有害处的。

对于所有的图片对，用户都可以选择宽度和高度的像素默认值范围。每个选择列表都有从1～250的像素选择范围。也许你后面会对它作修改，但现在要按照它来运行。选择的默认值宽度是90，高度是50，这是最为理想的矩形按钮尺寸。

最后一个选择列表是用来设置边框的，其范围是从0～10的像素值。大多数人总是愿意把它设置为0，但我还是见到过有边框的图片翻转。

选择复选框允许用户对事件处理方法*onMouseDown*添加一个图片翻转效果，这在

JavaScirpt 1.2 和 Navigator 的 DOM 以上版本，以及 IE 中都可以得到支持。用户要做的所有工作是选择“Proceed”，按照刚才提供的信息产生一个图片模板。

第 3 步：填写图片路径，HREF 属性和其他一些信息

用户一旦选择“Proceed”，ImageMachine 就会产生一个定制的图片模板，如图 5-2 所示。ImageMachine 利用三个函数来创建定制模板：captureDefaultProfile()，setArrays()和generateEntryForm()。

captureDefaultProfile()

函数 captureDefaultProfile()在用户选择“Proceed”时首先得到调用。下面是第 21～39 行的代码：

```
function captureDefaultProfile(formObj) {
  setArrays();
  imgDefaults    = formObj;
  var imgQty      = (imgDefaults.imgnumber.selectedIndex + 1);
  var imgHeight   = (imgDefaults.pxlheight.selectedIndex);
  var imgWidth    = (imgDefaults.pxlwidth.selectedIndex);
  var imgBorder   = (imgDefaults.defbdr.selectedIndex);
  for (var i = 0; i < imgQty; i++) {
    imgPrim[i] = "";
    imgRoll[i] = "";
    imgDown[i] = "";
    imgLink[i] = "";
    imgText[i] = "";
    imgWdh[i]  = imgWidth;
    imgHgt[i]  = imgHeight;
    imgBdr[i]  = imgBorder;
    }
  generateEntryForm();
  }
```

captureDefaultProfile()函数所做的第一件事是调用函数 setArrays()。如第 41～50 行代码所示，setArrays()定义并初始化八个数组。每一个数组负责保存每个图片组的特定属性值。例如，*imgPrim*包含所有基本图片翻转的图片路径。*imgRoll* 包含所有图片翻转（事件处理方法 *onMouseOver*）的图片路径。以此类推。如果数组还没有被定义，setArrays()将负责这个工作。如果数组已经得到定义（用户在前边产生了图片代码），这个函数将重置数组，将它的元素都设为 0。

```
function setArrays() {
  imgPrim = new Array();
  imgRoll = new Array();
  imgDown = new Array();
  imgLink = new Array();
  imgText = new Array();
  imgWdh  = new Array();
  imgHgt  = new Array();
  imgBdr  = new Array();
  }
```

函数 setArrays()返回以后，captureDefaultProfile()接着把表单对象*formObj*复制给变量*imgDefaults*。这是很重要的：*imgDefaults*是一个全局变量。它不像这个应用程序中其他许多局部变量一样“死亡”，因此，如果用户想要在预览、产生代码或修改图片属性之间转换的时候，它保存了一个用户默认值的复本。在应用程序中，这是*imgDefaults*得到设置的唯一地方。这就意味着如果用户要改变他们的默认值，唯一方法就是重新选择“Proceed”。

ImageMachine一旦有了一个用户默认值的复本，captureDefaultProfile()就定义四个局部变量。如下所示：

```
var imgQty      = (imgDefaults.imgnumber.selectedIndex + 1);
var imgHeight   = (imgDefaults.pxlheight.selectedIndex);
var imgWidth    = (imgDefaults.pxlwidth.selectedIndex);
var imgBorder   = (imgDefaults.defbdr.selectedIndex);
```

*imgQty*表示用户所要的翻转数目。*imgHeight*，*imgWidth*和*imgBorder*表示默认的宽度、高度和边框设置。这些变量使函数能够给 setArrays()中所定义的数组赋值。这里是第28～37行的代码：

```
for (var i = 0; i < imgQty; i++) {
  imgPrim[i] = "";
  imgRoll[i] = "";
  imgDown[i] = "";
  imgLink[i] = "";
  imgText[i] = "";
  imgWdh[i]  = imgWidth;
  imgHgt[i]  = imgHeight;
  imgBdr[i]  = imgBorder;
  }
```

*for*循环的循环次数和用户所需的翻转个数相同，它对每一个所定义的数组属性赋值。*imgPrim*保存*MouseOut*事件的路径。*imgRoll*保存*MouseOver*事件的值，而*imgDown*保存*MouseDown*事件的路径。*imgLink*和*imgText*分别保存HREF属性和

状态栏文字的值。由于用户会在图片模板中分别设置前五个元素的值，这五个数组中的每一个元素就会得到一个空串作为初始值。

剩下三个数组中的每一个元素都将得到同样的设置。默认宽度值对所有的图片都是一样的。默认高度和边框也是如此。用户在后面可以对此作修改，但现在它们的设置是相同的。

generateEntryForm()

函数中最后一件要做的事是调用generateEntryForm()。真正有趣的部分这才开始。这是用来在HTML中创建定制图片模板以供用户输入有关每一个图片组的特定数据的唯一函数。看一下第53～118行的内容。

有66行的代码都是属于一个函数的。这或许违反了每一个程序员对函数的长度限制。但是，generateEntryForm()仍然只执行一个任务：创建图片模板。你可以很容易地把函数分为HTML的三个部分：表格头（TH），表单文本域，以及按钮。整个函数其实是一系列对document.writeln()的调用。的确是这样。这里是第54～72行写表头的代码：

```
with(parent.frames[1].document) {
  open();
    writeln("<HTML><BODY BGCOLOR=FFFFEE><FONT FACE=Arial SIZE=2>" +
      "<BLOCKQUOTE>Choose or enter the paths of all images in the " +
      "columns below. Add the link path (e.g., <FONT FACE=Courier>" +
      "web_page.html</FONT>) or script text (e.g., <FONT FACE=Courier>" +
      "javascript:</FONT>) for each HREF attribute, and enter any " +
      "message you want to display in the status bar during the " +
      "<FONT FACE=Courier>MouseOver</FONT> event. Then choose " +
      "<B>Generate</B> to get your code, or <B>Preview</B> to see the " +
      "code in action.</BLOCKQUOTE><FORM NAME='imgProfile' " +
      "onSubmit='return false;'><CENTER><TABLE BORDER=0 ALIGN=CENTER " +
      "CELLSPACING=5 CELLPADDING=5><TH ALIGN=LEFT><FONT FACE=Arial>#" +
      "<TH ALIGN=LEFT><FONT FACE=Arial>Primary Path" +
      "<TH ALIGN=LEFT><FONT FACE=Arial>Rollover Path" +
      (imgDefaults.mousedown.checked ? "<TH ALIGN=LEFT>" +
      "<FONT FACE=Arial>MouseDown Path" : "") +
      "<TR><TD><BR></TD></TR>");
  }
```

注意到图片模板表单嵌入到一个表格中。除了第69～70行的代码之外，这部分的东西都是静态的。如果用户选择了“MouseDown”的复选框，JavaScript就会利用三元算子添加一个特殊的head部分，而不再加别的什么东西。我们来具体看一看：

```
(imgDefaults.mousedown.checked ? "<TH ALIGN=LEFT><FONT FACE=Arial>" +
  "MouseDown Path" : "")
```

这里才是真正的高潮：如果你理解了它，你就可以很快弄懂函数余下的内容，因为generateEntryForm()基于用户是否选择“MouseDown”复选框来编制所有代码。看一下位于第74～107行的文本域部分。

由于有*imgPrim*（也就是翻转的数目）中那样多的元素，ImageMachine添加一个新的TR，它包括两个（或者三个）FILE域的TD，还有一个用于HREF属性、状态栏文字、宽度、高度和边框像素值设置的文本域。如果你仔细查看，会发现setArrays()中定义的数组中索引*i*所指示的每个元素被设置为相关文本域中的一个值。

还记得*imgPrim*，*imgRoll*，*imgDown*，*imgLink*和*imgText*的初始值为空串吗？直到此刻，用户都没有机会给他们输入任何值。所以，把这些文本域的值设为空串是有意义的。但是，宽度、高度和边框的默认值是设置好了的。因此，*imgWidth*，*imgHeight*和*imgBorder*数组中索引*i*的相应元素分别设置为它们的文本域值也自有它们的用处。

注意循环代码：

```
(imgDefaults.mousedown.checked ?
```

每一种情况归结到表单中的一点，如果用户选择“MouseDown”复选框，就要添加用来提供附加图片的代码。

标头和表单文本域已经得到了处理。关于图片模板，唯一剩下的工作是创建“Generate”，“Preview”和“Clear”按钮。

```
with(parent.frames[1].document) {
  writeln("</TABLE><CENTER><INPUT TYPE=BUTTON " +
    "onClick='parent.frames[0].imgValid8(this.form, true);' " +
    "VALUE='Generate'><INPUT TYPE=BUTTON " +
    "onClick='parent.frames[0].imgValid8(this.form, false);' " +
    "VALUE='Preview'> <INPUT TYPE=RESET VALUE='   Clear   '>" +
    "</FORM></BODY></HTML>");
  close();
  }
```

“Clear”按钮是你的基本RESET按钮，因此我们把重点放在其他两个按钮上。注意到两个按钮在被点击时都调用函数imgValid8()。它们两个都传递一个表单的副本

给函数，只是一个传递 true，一个传递 false。这种差别可以决定 ImageMachine 是创建显示的代码还是注释代码。我们马上就会谈到。

顺便说一句，你可能会仔细查看 generateEntryForm() 的每一行代码，想弄清楚表单的 HTML 是怎样来的。你会看到一个非常长的字符串如何创建了表单，然后它将用户带入下一个步骤。generateEntryForm() 函数让用户填写好表单，然后，用户按照本章开头所述的四步程序选择 "Generate"。如上面的代码所示，调用 imgValid8()。

```
function imgValid8(imgTemplate, mimeType) {
  for (var  i = 0; i < imgPrim.length; i++) {
    if (imgTemplate['prim' + i].value == "" ||
      imgTemplate['seci' + i].value == "" ||
      imgTemplate['href' + i].value == "") {
      alert("All images and HREF attributes must have URLs.");
      return;
      }
    if (imgDefaults.mousedown.checked) {
      if(imgTemplate['down' + i].value == "") {
        alert("All images and HREF attributes must have URLs.");
        return;
        }
      }
    }
  genJavaScript(imgTemplate, mimeType);
  }
```

这个函数确认用户已经输入每一个图片路径的值。我们记得 "Generate" 按钮传递了一个有所有图片信息的表单副本。这个副本设置为 *imgTemplate*。ImageMachine 又一次用到 *imgPrim* 的长度来迭代包含图片路径在内的文本域。包含基本图片路径的域被命名为 prim+i，i 是从 0 到 imgPrim.length - 1 的某个数字。包含翻转图片路径的域用相同的方法命名，只是用 *seci* 代替 *prim*。如果用户把图片包括在 *MouseDown* 事件中，这些文本域会被依次命名。

genJavaScript()

如果检查到哪一个域是空的，就会警告用户告，并且函数返回。如果每一个域至少有一些文字，ImageMachine 调用函数 genJavaScript()，并传递 imgTemplate 和 mimeType 中还没有检查的布尔值。你可能已经猜到了，genJavaScript() 负责创建显示页面的 JavaScript 代码。这个函数非常长，但是仅只执行有关 generateEntryForm() 的单一工作。参见第 138～306 行。

JavaScript 技巧：JavaScript 开发维护

在产生代码中有 SCRIPT 标签，并且出现好几种不同的 LANGUAGE 并没有什么差错。你会在第 185 行 <SCRIPT LANGUAGE="JavaScript"> </SCRIPT> 标签中看到一些代码。第 189 行的 <SCRIPT LANGUAGE="JavaScript 1.1"> </SCRIPT> 标签中有另外一些代码，而在最后第 193 行代码的 <SCRIPT LANGUAGE="JavaScript1.2"> </SCRIPT> 中则有更多的代码。

这将给浏览器与对多样化 JavaScript 的兼容性、避免执行它不支持的代码而导致应用程序错误。例如，在 JavaScript 1.0 中不支持 Image() 对象。因此，除非 Image() 对象被嵌入包括“JavaScript 1.1”（或更高）LANGUAGE 属性的 <SCRIPT> 标签中，否则你将看不到任何它的代码。

你可以这样通过控制变量来维护编码，按照其所在的 <SCRIPT> 标签来设置它们的值。等到运行一个带有不确定的支持代码的函数时，只在控制变量的条件相符时才运行这些代码。你可以在 Image 对象代码的变量 *canRollOver* 中，以及事件处理方法 *onMouseDown* 代码的变量 *canClickDown* 中看到这种情况。

还要注意，这样处理 scirpt 可以减少下载时间。例如，不支持 JavaScript 1.2 的浏览器不会执行 <SCRIPT LANGUAGE="JavaScript1.2"> </SCRIPT> 标签中的任何代码。这可是个好消息，因为浏览器不一定非要用来做 *onMouseDown* 事件相关翻转的图片了。

你可能会认为，generateEntryForm() 真是太长了！但对它仍然作类似的处理。GenJavaScript() 执行一个单纯的任务：产生翻转代码，其中大部分是 JavaScript。

genJavaScript() 函数所做的第一件事是重置全局变量 *imageLinks*。一会儿我们还要讨论关于这些控制全局变量的代码的更多内容。接下来，genJavaScript() 按照 *mimeType* 的值设置了一些全局“规格”变量。下面是第 141～154 行的代码：

```
if (mimeType) {
  lt       = "&lt;";
  gt       = "&gt;";
  br       = "<BR>";
  HTML     = true;
  nbsp     = "     ";
  }
else {
```

```
        lt      = "<";
        gt      = ">";
        br      = lb;
        HTML    = false;
        nbsp    = "     ";
    }
```

如果*mimeType*的值为true，全局变量将会被设置为一个字符串值，用来完成代码的显示。变量*lt*和*gt*分别设置为<和>。然后，变量*br*被设置为
的串值。*HTML*是一个布尔变量，指出用户希望代码是注释性的（不显示在屏幕上）。等产生代码的时候，它就会开始运作。变量*nbsp*设置为一个没有空格的HTML串。*nbsp*在HTML中和Tab键作用相仿。

如果*mimeType*的值为false，就设置全局变量来使代码作为注释内容。变量*lt*和*gt*分别设置为<和>。然后变量*br*被设置为*lb*的字符串值。而变量*lb*的值将在第6～8行中随便用什么办法设为原值。下面是它的代码：

```
var platform = navigator.platform;
var lb = (platform.indexOf("Win" != -1) ? "\n\r" :
  (platform.indexOf("Mac" != -1) ? "\r" : "\n"));
```

和你所看到的一样，变量*platform*和*lb*在一起协同产生作用。*platform*包含做浏览器编译的操作系统值的字符串。而*lb*按照*platform*的值进行设定。在Windows(DOS)中，用\n\r来表示换行，这和按Enter或Return键作用相同。这种换行的方法可以避免所产生的代码行有两三行那么长。这对应用程序来说不是非要不可的，但是当ImageMachine打印HTML和JavaScript时，它会使查看源代码的时候变得简单一点。

JavaScript 技巧：全局变量的作用

全局变量对这个应用程序产生了相当重要的作用。ImageMachine产生打印代码或者执行代码。一个用来看，一个用来运行。但是，两种类型的代码几乎是同样的，除了执行代码使用HTML中的括号“<”和“>”，而打印代码则用了HTML括号的符号实体，<和>。全局变量*lt*和*gt*被设置为HTML括号或者符号实体，这依赖于你是否选择了“Generate”按钮或是“Preview”按钮。

变量*br*和*nbsp*得到类似的赋值。这就是全局变量的作用：仅通过改变变量的值，来产生执行显然不同但有相同用途的串。

接着往下看，被设置为false的*HTML*是一个布尔变量，表示用户希望代码是注释性的。这在产生代码的时候自会发挥作用。变量*nbsp*被设置为空格串。

现在imageMachine从模板的文本域中获得了信息，以及用户是否选择了打印代码或执行代码。确切说来，JavaScript的产生是从第185行开始的，并且一直到函数结束。

你查看代码的时候，会看见好几个对函数pathPrep()的调用。如果路径在Windows环境（参见第三章有关路径的内容）中以绝对路径的形式出现，这个函数将重新格式化图片路径串。没必要总是大惊小怪的。应该记住Windows用反斜杠符号（\）来分隔目录。浏览器使用斜杠（/）。Unix的机器也是这样做的。因此，将反斜杠转变为斜杠非常必要。实际上，很多浏览器都要在运行中作此转变。

问题是JavaScript将反斜杠作为无效符号。因此，JavaScript会把*C:\My_Directory\My.File*当作*C:My_DirectoryMy.File*。不过函数pathPrep()会对此作出处理。下面是第310～317行的代码：

```
function pathPrep(path) {
  if (path.indexOf(":\\") != -1) {
    path = path.replace(/\\/g, "/");
    path = path.replace(/:\//, "|/");
    return "file:///" + path;
    }
  else { return path; }
  }
```

浏览器还按照文件协议打开文档，这意味着我们需要把*file:///*加在URL前面，用一个管道符（|）来代替冒号（:），以符合规范。

决策时刻

现在到了按照已列出或已解释的规格来产生代码的时候。但是在ImageMachine建造代码之前，还需要知道是按照图片模板中的新数据呢，还是使用已经存储在数组中的信息。按照我们在前面说过的用户动作步骤，用户刚好在图片模板中完成了信息的输入。另一种情形是用户已经产生了代码，并且在“Generate”和“Preview”之间切换。我们马上要对此作讨论。

如果从图片模板中获取信息(在本例中)，genJavaScript()将通过调用setArrays()来重置*img*数组，这样，从图片模板中来的新信息将得到分派。ImageMachine通过

检测*imgTemplate*来决定是否调用setArray()和重赋值。函数genJavaScript()有三种调用的方法，从“Generate”按钮，还有“Preview”按钮以及从函数imgValid8()调用。从“Generate”和“Preview”来调用genJavaScript()将传递null值给*imgTemplate*。因此，如果*imgTemplate*不等于null，则genJavaScript()知道应该清空数组，并设法赋予新的信息。否则，*img*数组中的元素将不会被修改。仔细研究第156～170行的代码，看看这一过程：

```
if(imgTemplate != null) {
  setArrays();
  for (var i = 0; i < (imgDefaults.imgnumber.selectedIndex + 1); i++) {
    imgPrim[i] = purify(imgTemplate['prim' + i].value);
    imgRoll[i] = purify(imgTemplate['seci' + i].value);
    if (imgDefaults.mousedown.checked) {
      imgDown[i] = purify(imgTemplate['down' + i].value);
      }
    imgLink[i] = purify(imgTemplate['href' + i].value);
    imgText[i] = purify(imgTemplate['stat' + i].value);
    imgWdh[i]  = purify(imgTemplate['wdh' + i].value);
    imgHgt[i]  = purify(imgTemplate['hgt' + i].value);
    imgBdr[i]  = purify(imgTemplate['bdr' + i].value);
    }
  }
```

如果数组元素被修改，赋给它们的串值将经历由第308行的函数purify()来执行的一个快速而危险的字符删除过程。你将看到：

```
function purify(txt) {return txt.replace(/\'|\"/g,"");}
```

这将把你的值中的单引号和双引号都删除掉。它们的确并没有错，但是JavaScript必须用单引号和双引号来产生代码。除非它们像反斜杠符号那样不起作用，否则对你的代码产生将会产生阻碍。Purify()从传递的字符串中删除这些符号，然后返回新串。

产生代码

这些工作一旦完成，就该产生等了半天的代码了。这出现在第185～305行，将所有产生的代码赋值为好几个“代码保存“变量。这些变量如下所示：

PrimJavaScript

保存诸如HTML，HEAD，以及TITLE之类的HTML标签。也包含最初的JavaScript与*MouseOver*和*MouseOut*事件的相关翻转代码。

SecJavaScript

保存 JavaScript 1.2 中 *MouseDown* 事件的相关翻转代码。

imageLinks

保存用来显示链接的 HTML 代码。

scriptClose

保存一个 SCRIPT 的结束标签。

swapCode

保存将执行图片翻转的 JavaScript 函数。

primHTML

保存 BODY 标签和一些 HTML 注解。

secHTML

保存 HTML 的结束标签以及在代码产生后（“Generate”和“Change Info”，或者是“Preview”和“Change Info”）显示的“Form”按钮。

aggregate

这个变量是刚才所说的所有变量的合并。

第 197 行的 *for* 循环再次循环 imgPrim.length 次。在每一次循环中，变量 *primJavaScript*，*secJavaScript*（如果用户选择 MouseDown 选项）和 *imageLinks* 加入循环中下一组图片的相关代码。

变量 *scriptClose*，*swapCode*，*primHTML* 和 *secHTML* 不在 *for* 循环中。它们的内容可以通过变量 *HTML* 和 *imgDefaults.mousedown.checked* 和三元算子的共同作用而得到一次设置。

一旦结束 *for* 循环，其他变量也得到相应设置后，剩下的最后一件事就是将内容加到页面中去。第 300～305 行包括了它的全部过程：

```
agregate = primJavaScript +
  (imgDefaults.mousedown.checked ? scriptClose + secJavaScript : "") +
  swapCode + primHTML + imageLinks + secHTML;

parent.frames[1].location.href =
  "javascript: parent.frames[0].agregate";
```

第 4 步：选择“Preview”查看运行代码

如果到现在你已经非常疲惫了，我也不能怪你。幸运的是，最后的两个步骤非常快捷轻松。假设用户已经浏览了产生的代码，现在想看看运行效果。点击一下“Preview”就行。记住，点击“Preview”会调用genJavaScript()，但*mimeType*将是false而不是true，这是和“Generate”按钮的唯一分别。第141～154行的“规范”变量仅被设置为注释代码的映像而不像以前是打印代码。其他所有的东西都和“Generate”按钮一样。

第 5 步：选择“Change Info”来做修改

现在你已经看到了产生的代码和它的运行效果。假设用户想作点什么修改。选择“Change Info”，图片模板和信息就会重新出现。宽度，高度，状态文字……除了图片URL之外，一切你输入的东西都在。等一分钟……为什么没有URL呢？

出现这种情况是因为图片路径URL被存储在一个*FileUpload*对象中（例如，<INPUT TYPE=FILE>）。出于安全性原因，*FileUpload*对象是只读的。换句话说，你必须用键盘输入一个值，或者用鼠标在对话框中选择一个文件来手动生成一个域。幸好这是比较容易改变的。只要在generateEntryForm()中将TYPE=FILE换成TYPE=TEXT。在那里你会发现有三个要改的地方。唯一的问题是你将不能用鼠标和对话框寻找本地文件。依次输入是最便利的方法了。一旦完成了修改，就可以选择“Generate”或“Preview”来检验新代码。

应用程序扩展：添加模板属性

大的应用程序经常会被弄得很庞大。这一部分的内容是深入向你展示如何在模板中添加属性，使你能强有力地控制产生的代码。为了使事情简单一点，我会讲解如何在IMG属性中添加HSPACE和VSPACE。这个过程包括这样六个步骤：

1. 在默认模板中添加新的域。
2. 在setArrays()中创建一个数组来保存它的值。
3. 获取新的默认值。

4. 在generateEntryForm()中添加图片模板文本域。

5. 引用和设置genJavaScript()中新的属性值。

6. 产生用来显示genJavaScript()属性的HTML。

JavaScript 技巧：JavaScript 1.1 和 1.2 中的字符串置换

JavaScript 1.2 带来许多非常酷的新特点。其中之一是使用正则表达式完成字符串匹配和置换的功能。函数pathPrep()和purify()用 JavaScript 1.2 执行一个单一但非常有效的置换方法。这对支持 JavaScript 1.2 的浏览器来说真是太好了。但是Netscape 3.x仍然被广泛地使用着。这里是一个JavaScript 1.2中使用*Array*对象的方法来执行字符串置换的一个函数：

```
function replacev11(str, oldSubStr, newSubStr) {
  var newStr = str.split(oldSubStr).join(newSubStr);
  return newStr;
  }
```

这个函数得到一个字符串，按照你要删除的子串(*oldSubStr*)创建元素split()的一个数组，然后返回一个由join()制造的字符串，用新的子串(*newSubStr*)来获取数组。不是很干净利落，但是有用。

第 1 步：添加域

```
<TD VALIGN="TOP">
<FONT FACE="Arial" SIZE=2>
HSpace
</TD>
<TD VALIGN="TOP">
<FONT FACE="Arial" SIZE=2>
<SCRIPT LANGUAGE="JavaScript1.2">
<!--
genSelect("hspace", 25, 0, 0);
//-->
</SCRIPT>
</TD>
<TD VALIGN="TOP">
<FONT FACE="Arial" SIZE=2>
VSpace
</TD>
```

```
<TD VALIGN="TOP">
<FONT FACE="Arial" SIZE=2>
<SCRIPT LANGUAGE="JavaScript1.2">
<!--
genSelect("vspace", 25, 0, 0);
//-->
</SCRIPT>
</TD>
```

第 2 步：在 setArrays()中创建数组

```
function setArrays() {
  imgPrim = new Array();
  imgRoll = new Array();
  imgDown = new Array();
  imgLink = new Array();
  imgText = new Array();
  imgWdh  = new Array();
  imgHgt  = new Array();
  imgBdr  = new Array();
  imgHSpace  = new Array();  // For the HSPACE
  imgVSpace  = new Array();  // For the VSPACE
  }
```

这产生了存储新的默认值的空间。并将我们带入下一步——填写这些新的数组。

第 3 步：获取新的默认值

在 captureDefaultProfile()，中，我们将添加两个名为 *imgHspace* 和 *imgVspace* 的局部变量，然后将默认模板中所选的值赋给它们。现在看看如下所示的 captureDefaultProfile()：

```
function captureDefaultProfile(formObj) {
  setArrays();
  imgDefaults    = formObj;
  var imgQty      = (imgDefaults.imgnumber.selectedIndex + 1);
  var imgHeight   = (imgDefaults.pxlheight.selectedIndex);
  var imgWidth    = (imgDefaults.pxlwidth.selectedIndex);
  var imgBorder   = (imgDefaults.defbdr.selectedIndex);
  var imgHspace   = (imgDefaults.hspace.selectedIndex);
  var imgVspace   = (imgDefaults.vspace.selectedIndex);
  for (var i = 0; i < imgQty; i++) {
    imgPrim[i] = "";
    imgRoll[i] = "";
    imgDown[i] = "";
    imgLink[i] = "";
    imgText[i] = "";
```

```
    imgWdh[i]   = imgWidth;
    imgHgt[i]   = imgHeight;
    imgBdr[i]   = imgBorder;
    imgHSpace[i] = imgHspace;  // For HSPACE
    imgVSpace[i] = imgVspace;  // For VSPACE
    }
  generateEntryForm();
  }
```

ImageMachine现在能够包括图片模板中HSPACE和VSPACE的默认值了。

第4步：在generateEntryForm()中添加文本域

现在你可以在generateEntryForm()中添加HTML字符串来提供两个新的文本域。我们把它们放在其本身所属的TR中所有其他项目之后。等会儿你可以作适当的调整，使它看起来漂亮一点。第103～106行现在是这个样子的：

```
"<TR><TD VALIGN=BOTTOM COLSPAN=" +
(!imgDefaults.mousedown.checked ? "3" : "4") +
"><BR><HR NOSHADE><BR></TD></TR>");
```

在最后添加两个文本域，如下所示：

```
"<TR><TD VALIGN=BOTTOM><INPUT TYPE=TEXT NAME='hsp " + i +
"' VALUE='" + imgHspace[i] + "' SIZE=3> HSPACE </TD>" +
"<TR><TD VALIGN=BOTTOM><INPUT TYPE=TEXT NAME='vsp" + i +
"' VALUE='" + imgVspace[i] + "' SIZE=3> VSPACE </TD></TR>" +
"<TR><TD VALIGN=BOTTOM COLSPAN=" +
(!imgDefaults.mousedown.checked ? "3" : "4") + ">" +
"<BR><HR NOSHADE><BR></TD></TR>");
```

这段代码为每一个图片组添加了两个文本域，其中还显示了它们各自的默认值。用户能在以后对它们作修改，就像对其他的属性一样。

第5步：引用和设置genJavaScript()中的新属性值

一旦用户选择产生代码，ImageMachine需要从图片模板的文本域中获取信息。只要在第158～169行添加代码就可以了，如下所示：

```
for (var i = 0; i < (imgDefaults.imgnumber.selectedIndex + 1); i++) {
  imgPrim[i] = purify(imgTemplate['prim' + i].value);
  imgRoll[i] = purify(imgTemplate['seci' + i].value);
  if (imgDefaults.mousedown.checked) {
```

```
    imgDown[i] = purify(imgTemplate['down' + i].value);
    }
  imgLink[i] = purify(imgTemplate['href' + i].value);
  imgText[i] = purify(imgTemplate['stat' + i].value);
  imgWdh[i]  = purify(imgTemplate['wdh' + i].value);
  imgHgt[i]  = purify(imgTemplate['hgt' + i].value);
  imgBdr[i]  = purify(imgTemplate['bdr' + i].value);
  imgHSpace[i]  = purify(imgTemplate['hsp' + i].value);
  imgVSpace[i]  = purify(imgTemplate['vsp' + i].value);
  }
```

这一部分最后两行展示了ImageMachine如何把图片模板中的表单值赋给*imgHSpace*和*imgVSpace*的元素。我们几乎已经完成了这一步。剩下的唯一任务是确认新的属性已经被包括在代码的产生，打印和执行中了。

第6步：在genJavaScript()中产生附加的HTML

新代码会被加到变量*imageLinks*。有关字符串的最后几行代码如下所示：

```
(HTML ? fontClose : "") + br + nbsp + "HEIGHT=" +
(HTML ? fontOpen : "") + imgHgt[j] +
(HTML ? fontClose : "") + br + nbsp + "BORDER=" +
(HTML ? fontOpen : "") + imgBdr[j] +
(HTML ? fontClose : "") +
gt + "" + lt + "/A" + gt + br + br + br;
```

你所需要做的只是拷贝几行代码，然后将HEIGHT改成HSPACE，imgHgt改成imgHSpace，BORDER变为VSPACE，还有将imgBdr变为imgVSpace。下面是新的代码版本：

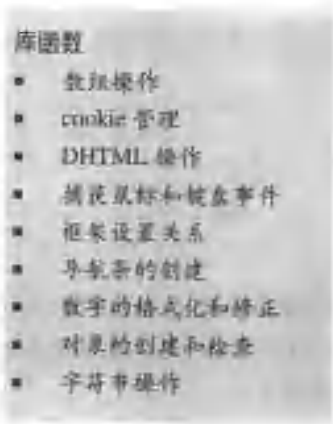
```
(HTML ? fontClose : "") + br + nbsp + "HEIGHT=" +
(HTML ? fontOpen : "") + imgHgt[j] +
(HTML ? fontClose : "") + br + nbsp + "BORDER=" +
(HTML ? fontOpen : "") + imgBdr[j] +
(HTML ? fontClose : "") + br + nbsp + "HSPACE=" +
(HTML ? fontOpen : "") + imgHSpace[j] +
(HTML ? fontClose : "") + br + nbsp + "VSPACE=" +
(HTML ? fontOpen : "") + imgVSpace[j] +
(HTML ? fontClose : "") +
gt + "" + lt + "/A" + gt + br + br + br;
```

这将为你的图片添加两个新的属性。你也许还想给图片内容添加一个ALT标签。没有必要给IMG标签加上修改类型的限制。在<A>标签中有足够的发挥余地；你可以将它定制为图片地图，等等诸如此类的东西。

第六章

JavaScript 源文件

如果从一开始就按顺序阅读本书，那么你一定已经细读代码，并且想要弄清楚函数和变量究竟如何共同作用，从而使那些应用程序最终能完成某种任务。我想如果打断你一下，去看看一个使所有的应用程序都变得简单明了的东西，或许是一个很好的建议。



本章不包含任何应用程序。而且它要证明包含在JavaScript源文件里的好几十个函数。虽然你可能不会觉得所有内容都有用，但其中或许有一些你马上就可以用上，另外，你还可以照搬一些东西，在其他什么地方派上用场。

我不想使你觉得我只交给你一捆函数，然后说："好了，程序员，这就是你所需要的。"我不想这样来讲完这些文件，这真是太可笑啦。这一章原本是想鼓励你用可以再次使用的代码构建你自己的JavaScript库。于是你不必在编制一个新的应用程序时重新去创造一些东西。下面的列表按字母顺序给出了*.js*文件及它们每一个的用途描述。

arrays.js

它包含数组处理函数。一些函数可用于在版本较低的浏览器中执行和JavaScript 1.2等价的操作。

cookies.js

这是个让人生畏的库，绝大部分来自于JavaScript老手Bill Dortch，使cookie起杠杆作用。

dhtml.js

你在第三章交互式的幻灯片放映，以及第四章多种类型搜索引擎的界面中已经看到了所有这些函数。这个包可以很容易地创建、显示，以及隐藏跨浏览器DHTML层。

events.js

这个源文件包含的代码能在Navigator和IE中捕获或忽略*mousemove*和*keypress*事件。

frames.js

这些函数可以帮助你把页面放在其他框架设置之内或者之外(不管你指定的是哪一个)。

images.js

它包含你可能在前边的章节中已经看到过的翻转代码，不过在这里显得非常井然有序。

navbar.js

它所包含的代码基于当前载入的文档产生一个动态的导航条，给人印象较深。

numbers.js

所包含的代码用来纠正JavaScript的舍入错误，并负责数字的格式化。

objects.js

所包含的代码用于一般对象的创建和对象检查。

strings.js

这个文件包含一些字符串处理函数。

除了*navbar.js*以外，每一个*.js*文件都有一个相应命名的HTML文档（例如，*array.html*相对于*arrays.js*)。在这里，函数没有和其在应用程序中一样多的细节说明。尽管也有一些例外情况，但其实没有什么必要。你阅读这一章的时候，思考一下每一个函数是怎样解决我们的一个普通问题的。如果它不像我们所需要的那样起作用，你可以考虑一下如何修改它才能使它为你所用。

每个描述一个 *.js* 文件的部分都是从文件名开始的，然后是实际应用，必需的 JavaScript 版本，还有一个包含在文件中的函数列表。

arrays.js

实际应用

数组处理

版本要求

JavaScript 1.2

函数

`avg(),high(),low(),jsGrep(),truncate(),shrink(),integrate(), reorganize()`

这些函数对你的数组进行处理，然后返回其他一些有用的信息，包括其他的数组。图 6-1 给出了 *arrays.html*。虽然没有什么惊人之处，但是你可以看到每一个函数的示范。

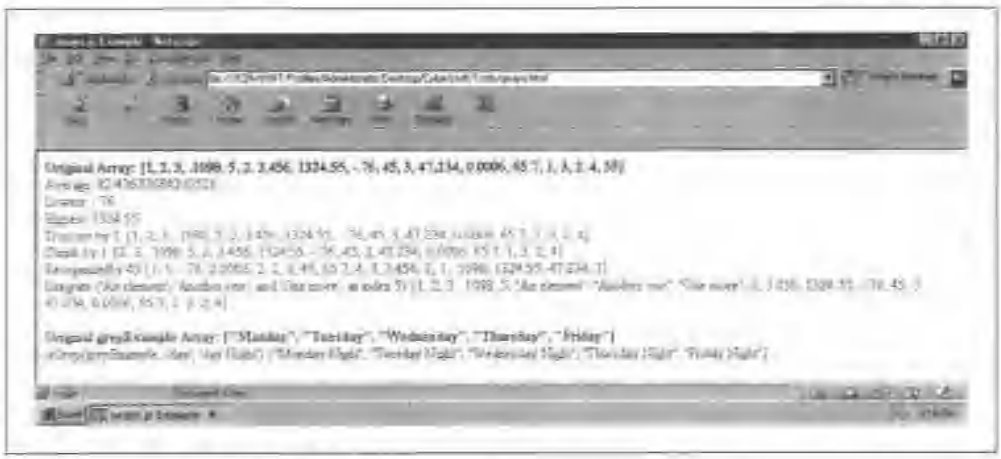


图 6-1 arrays.js 的功能演示

下面是 *arrays.js* 中函数及其用途的列表：

`avg()`

返回数组中所有数字的平均值

`high()`

返回数组中的最大数

`low()`

返回数组中的最小数

`jsGrep()`

在所有数组元素上执行字符串匹配和置换

`truncate()`

返回一个去除了最后一个元素的数组拷贝

`shrink()`

返回一个去除了第一个元素的数组拷贝

`integrate()`

从你定义的索引数开始将两个数组中的元素组合在一起

`reorganize()`

按照你选择的倍数对数组元素重排序

现在来看看*arrays.html*的代码，如例6-1所示。这里并没有多少内容，只是一个对`document.write()`的调用。显示出来的串包含所有函数调用的结果，以及示例数组`someArray()`和`grepExample()`。

例 6-1：arrays.html

```
<HTML>
<HEAD>
<TITLE>arrays.js Example</TITLE>
<SCRIPT LANGUAGE="JavaScript1.2" SRC="arrays.js"></SCRIPT>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript1.2">
<!--

var someArray = new Array(1,2,3,.1098,5,2,3.456,1324.55,-0.76,45,3,47.234,.
  00060,65.7,1,3,2,4,55);
var grepExample = new Array('Monday', 'Tuesday', 'Wednesday',
  'Thursday', 'Friday');
document.write("<B>Original Array: " + someArray + "</B><BR>" +
  "Average: " + avg(someArray) + "<BR>" +
```

```
  "Lowest: " + low(someArray) + "<BR>" +
  "Highest: " + high(someArray) + "<BR>" +
  "Truncate by 1: " + truncate(someArray) + "<BR>" +
  "Shrink by 1: " + shrink(someArray) + "<BR>" +
  "Reorganize(by 4): " + reorganize(someArray, 4) + "<BR>" +
  "Integrate ('An element', 'Another one', and 'One more', " +
  "at index 5): integrate(someArray, new Array('An element', " +
  "'Another one', 'One more'), 5) + "<BR><BR><B>Original grepExample " +
  "Array: " + grepExample + "</B><BR> " +
  "jsGrep(grepExample, /day/, \'day Night\'): " +
  jsGrep(grepExample, /day/, 'day Night') + "<BR>");

//-->
</SCRIPT>
</BODY>
</HTML>
```

你也许已经注意到了，两个SCRIPT标签都要求JavaScript1.2。唯一原因在于函数jsGrep()，它使用了JavaScript1.2的字符串匹配和置换功能。先把jsGrep()放在一边。你可以通过删除（或修改）jsGrep()而使你的应用在所有的JavaScript1.1浏览器中有效。既然你已经看到了我们是怎样调用函数的，现在就来看看函数本身吧，*arrays.js*，如例6-2所示。

例6-2：arrays.js

```
  var sum = 0;
  for (var i = 0; i < arrObj.length; i++) {
    sum += arrObj[i];
    }
  return (sum / i);
  }

function high(arrObj) {
  var highest = arrObj[0];
  for (var i = 1; i < arrObj.length; i++) {
    highest = (arrObj[i] > highest ? arrObj[i] : highest);
    }
  return (highest);
  }

function low(arrObj) {
  var lowest = arrObj[0];
  for (var i = 1; i < arrObj.length; i++) {
    lowest = (arrObj[i] < lowest ? arrObj[i] : lowest);
    }
  return (lowest);
  }

function jsGrep(arrObj, regexp, subStr) {
  for (var i = 0; i < arrObj.length; i++) {
    arrObj[i] = arrObj[i].replace(regexp, subStr);
```

```
27      }
28    return arrObj;
29    }
30
31  function truncate(arrObj) {
32    arrObj.length = arrObj.length - 1;
33    return arrObj;
34    }
35
36
37  function shrink(arrObj) {
38    var tempArray = new Array();
39    for(var p = 1; p < arrObj.length; p++) {
40      tempArray[p - 1] = arrObj[p];
41      }
42    return tempArray;
43    }
44
45
46  function integrate(arrObj, elemArray, startIndex) {
47    startIndex = (parseInt(Math.abs(startIndex)) < arrObj.length ?
48      parseInt(Math.abs(startIndex)) : arrObj.length);
49    var tempArray = new Array();
50    for( var p = 0; p < startIndex; p++) {
51      tempArray[p] = arrObj[p];
52      }
53    for( var q = startIndex; q < startIndex + elemArray.length; q++) {
54      tempArray[q] = elemArray[q - startIndex];
55      }
56    for( var r = startIndex + elemArray.length; r < (arrObj.length +
57      elemArray.length); r++) {
58      tempArray[r] = arrObj[r - elemArray.length];
59      }
60    return tempArray;
61    }
62
63  function reorganize(formObj, stepUp) {
64    stepUp = (Math.abs(parseInt(stepUp)) > 0 ?
65      Math.abs(parseInt(stepUp)) : 1);
66    var nextRound = 1;
67    var idx = 0;
68    var tempArray = new Array();
69    for (var i = 0; i < formObj.length; i++) {
70      tempArray[i] = formObj[idx];
71      if (idx + stepUp >= formObj.length) {
72        idx = nextRound;
73        nextRound++;
74        }
75      else {
76        idx += stepUp;
77        }
78      }
79    return tempArray;
80    }
```

函数 avg()，high()，以及 low()看起来都没有什么了不起的。avg()将所有的值加起来，然后将总数除以 arrObj.length，返回商的值。其他两个函数迭代数组，将每一个元素同其他的作比较，确定元素的最大和最小值。

函数 jsGrep()迭代数组元素，执行字符串匹配或者置换。每一个熟悉 Perl 的人可能已经用过 grep()子程序无数次了。Perl 的 grep()非常有用，但一般都按同样的方式工作。

用在 JavaScirpt 1.1 中的函数 truncate()和 shrink()其实只是 JavaScript 1.2 中数组函数 pop()和 shift()的简单等价物。实际上 pop()和 shift()是按照 Perl 中有相似命名的执行子程序来命名的。

函数 integrate()也是 JavaScript 1.1 中和 JavaScript 1.2 数组函数 slice()等价的函数。

slice()的命名也和 Perl 子程序相一致。这个函数相当简单。虽然有三个 *for* 循环，但循环次数始终是 arrObj.length+elemArray.length。

函数 reorganize()根据你所选的一个倍数对数组元素进行重排序。换句话说，如果你用 3 “改组”了一个有 10 个元素的数组（顺序为 0，1，2，3，4，5，6，7，8，9），那么新的顺序就会变成 0，3，6，9，1，4，7，2，5，8。

cookies.js

实际应用

计数器，表单重填，用户参数选择设置

版本要求

JavaScript 1.1

函数

getCookieVal()，GetCookie()，DeleteCookie()，SetCookie()

你需要客户状态管理吗？一个很酷的网站怎样来问候再次来访的使用者？需要建立一个语言转换界面或者用户参数设置吗？下面的代码将非常容易地建立和获取 cookies 信息。图 6-2、图 6-3 和图 6-4 显示了 *cookies.html* 的运行。注意图 6-2 中，

页面首次被载入时，提示用户输入一个名字。图 6-3 显示了第一次登录的用户所看到的欢迎信息。图 6-4 表示出再次来访的用户将得到友好的欢迎，并给出他的个人登录次数。

图 6-2　新用户填写名字……

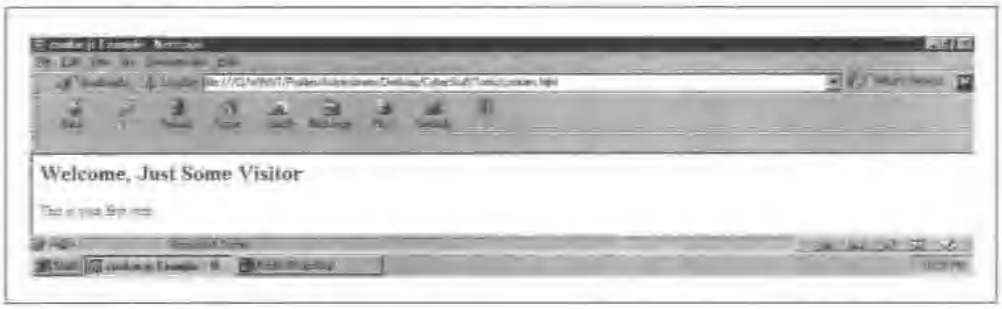


图 6-3　对新用户的欢迎界面

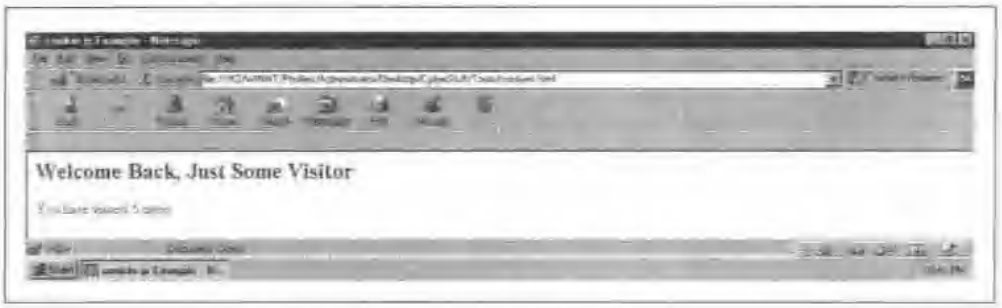


图 6-4　然后成为集体的一员

这是关于cookie功能的一个明确简单的例子。在第七章，基于cookie的用户参数选择中，将给出同样的代码来“回忆”用户参数选择。顺便说一句，如果你没有充分理解cookie概念，可以参见非正式化的cookie常见问题解答，它的网址是http://www.cookiecentral.com/unofficial_cookie_ faq.htm。因为它是非正式的，所以，对于你的所有问题，都只能得到并不太高明的答案。但我们会在第七章讨论有关的更多细节。

文件*cookies.html*是这样工作的：当用户载入页面，它就查找一个叫*user_id*的cookie。如果这个名字不存在(等于null)，它就提示用户输入名字。然后它将cookie的*user_id*设为用户名字的值，cookie的*hit_count*设为2（用户下次登录的次数）。

如果*user_id*存在，就获取它的值和*hit_count*的值。如果cookie的*user_id*存在，说明用户以前曾经访问过这个网站。那么可以很有把握地认为，cookie的*hit_count*已经被设置过了。它给出用户的名字和来过网站的次数，然后把*hit_count*的值重新设为表示hit_count+1的值的字符串。参见如例6-3所示的*cookies.js*文件，看一下所有这些设置及其有关情况。

例6-3：cookies.js

```
var today = new Date();
var expiry = new Date(today.getTime() + 365 * 24 * 60 * 60 * 1000);

function getCookieVal (offset) {
  var endstr = document.cookie.indexOf (";", offset);
  if (endstr == -1) { endstr = document.cookie.length; }
  return unescape(document.cookie.substring(offset, endstr));
  }

function GetCookie (name) {
  var arg = name + "=";
  var alen = arg.length;
  var clen = document.cookie.length;
  var i = 0;
  while (i < clen) {
    var j = i + alen;
    if (document.cookie.substring(i, j) == arg) {
      return getCookieVal (j);
      }
    i = document.cookie.indexOf(" ", i) + 1;
    if (i == 0) break;
    }
  return null;
  }

```

```
26 function DeleteCookie (name,path,domain) {
27   if (GetCookie(name)) {
28     document.cookie = name + "=" +
29     ((path) ? "; path=" + path : "") +
30     ((domain) ? "; domain=" + domain : "") +
31     "; expires=Thu, 01-Jan-70 00:00:01 GMT";
32     }
33   }
34
35 function SetCookie (name,value,expires,path,domain,secure) {
36   document.cookie = name + "=" + escape (value) +
37     ((expires) ? "; expires=" + expires.toGMTString() : "") +
38     ((path) ? "; path=" + path : "") +
39     ((domain) ? "; domain=" + domain : "") +
40     ((secure) ? "; secure" : "");
41   }
```

这里有四个函数，但是你只需要调用其中的三个：SetCookie()，GetCookie()和DeleteCookie()。getCookieVal()是一个内置函数。你根本不用直接调用它。

用GetCookie()创建cookie非常简单。你只需要传递一个cookie名称（以后用GetCookie()来访问它），要存储的信息（比如一个用户名字或者登录次数），还有按顺序的访问终止日期。你必须提供前两个参量。终止日期是用*today*和*page*两个变量来表示的。变量*expiry*按照用户登录的日期被设置为某一年的某个日子。这是通过将变量*today*置为一个新的*Date*对象并使用getTime()来实现的。下面是它的工作过程。

变量*today*是一个*Date*对象。因此，today.getTime()以毫秒的形式返回当前时间（从格林威治标准时间的1970年00:00:00算起）。它给了我们当前时间的毫秒数，但是我们要的是从现在开始的某一年的终止日期。一年有365天，一天24个小时，一小时有60分钟，一分钟有60秒，最后，一秒有1000毫秒。只要把它们相乘，然后把乘积（就是3.1536e10毫秒）加到getTime()的返回值上就行了。

GetCookie()和DeleteCookie()的语法也很简单。你所要做的所有工作就是传递cookie的相关名称。GetCookie()将返回cookie的值（如果找不到就为null），而DeleteCookie()将删除与传递进来的名字相关的cookie。删除工作仅仅意味着将cookie置为一个过去的终止日期。

dhtml.js

实际应用

DHTML 层的创建，隐藏和显示

版本要求

JavaScript 1.2

函数

`genLayer()`, `hideSlide()`, `showSlide()`, `refSlide()`

如果你按顺序阅读本书，你应该看到了前边所讲的两个应用程序（幻灯片放映和多样搜索引擎的界面）。图 6-5 和图 6-6 给出的代码用来创建层，并且可用于随意地隐藏或者显示它们。

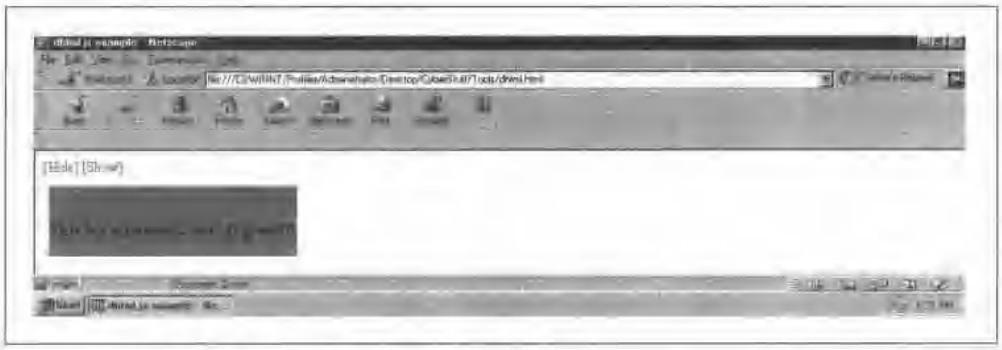


图 6-5 “引人注目”的 DHTML：现在你看见它了

图 6-6 现在又看不见了

例6-4给出了*dhtml.js*的内容。我没有作任何的改变。查看第二章和第五章的代码具体内容。

例 6-4：dhtml.js

```
var NN       = (document.layers ? true : false);
var hideName = (NN ? 'hide' : 'hidden');
var showName = (NN ? 'show' : 'visible');
var zIdx     = -1;
function genLayer(sName, sLeft, sTop, sWdh, sHgt, sVis, copy) {
  if (NN) {
    document.writeln('<LAYER NAME="' + sName + '" LEFT=' + sLeft +
      ' TOP=' + sTop +
      ' WIDTH=' + sWdh + ' HEIGHT=' + sHgt + ' VISIBILITY="' + sVis +
      '" z-Index=' + zIdx + '>' + copy + '</LAYER>');
    }
  else {
    document.writeln('<DIV ID="' + sName +
      '" STYLE="position:absolute; overflow:none;left:' + sLeft +
      'px; top:' + sTop + 'px; width:' + sWdh + 'px; height:' + sHgt +
      'px; visibility:' + sVis + ' z-Index=' + (++zIdx) + '">' + copy +
      '</DIV>');
    }
  }

function hideSlide(name) {
  refSlide(name).visibility = hideName;
  }

function showSlide(name) {
  refSlide(name).visibility = showName;
  }

function refSlide(name) {
  if (NN) { return document.layers[name]; }
  else { return eval('document.all.' + name + '.style'); }
  }
```

events.js

实际应用

跨浏览器事件处理方法设置，鼠标动作跟踪

版本要求

JavaScript 1.2

函数

enableEffects(), showXY(), keepKeys(), showKeys()

如果你没有接触过跨浏览器事件处理方法的脚本，这可以是你的入门读物。这个例子利用了三个事件处理方法：*onclick*，*onmousemove* 和 *onkeypress*。你第一次在文档空间的任何一处点击鼠标，JavaScript 就会获取浏览窗口中鼠标箭头的初始（x，y）坐标。然后，用户移动鼠标箭头时，状态栏中会相应显示每一时刻的（x，y）坐标。再次点击将“关闭”鼠标跟踪，并计算用户首次点击位置到当前位置的像素距离。你可以在图 6-7 和图 6-8 中看到相关内容。

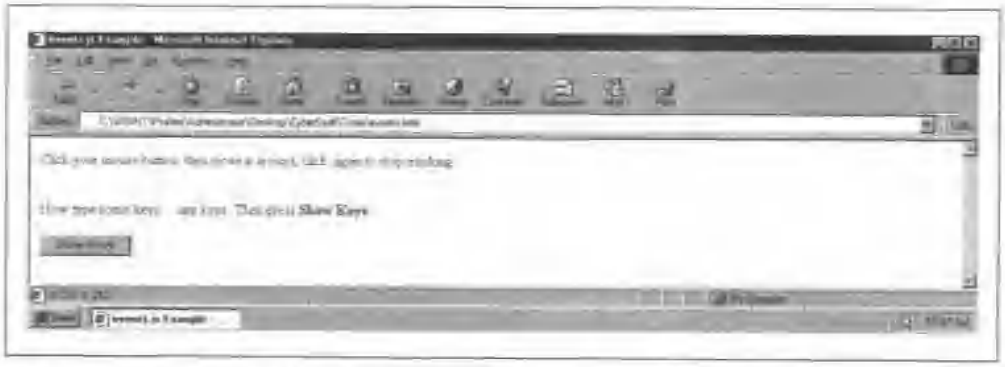


图 6-7　状态栏中鼠标的（x，y）坐标

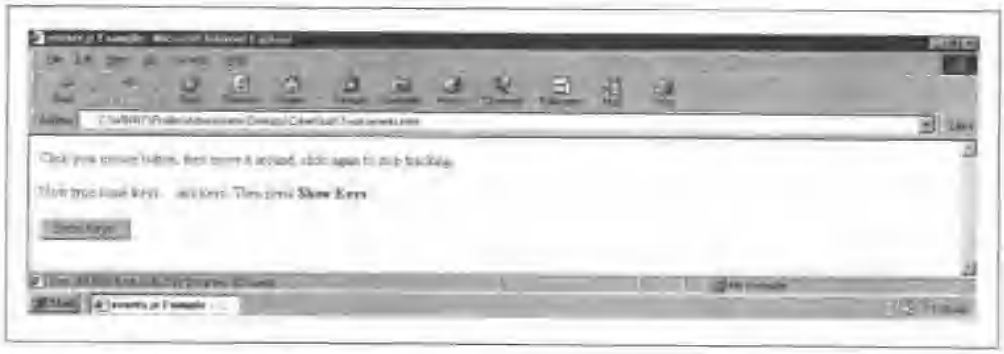


图 6-8　两个点之间的像素距离

如果要不受鼠标移动的影响，你也可以在键盘上按任何顺序的键。状态栏就会显示每一个你所敲的单键。完成的时候，选择“Show Keys”按钮，你就会得到一个 JavaScript 警告对话框，上面依次显示出你在那个点累积所敲的键。如图 6-9 所示。选择“OK”，又可以重新开始试验。

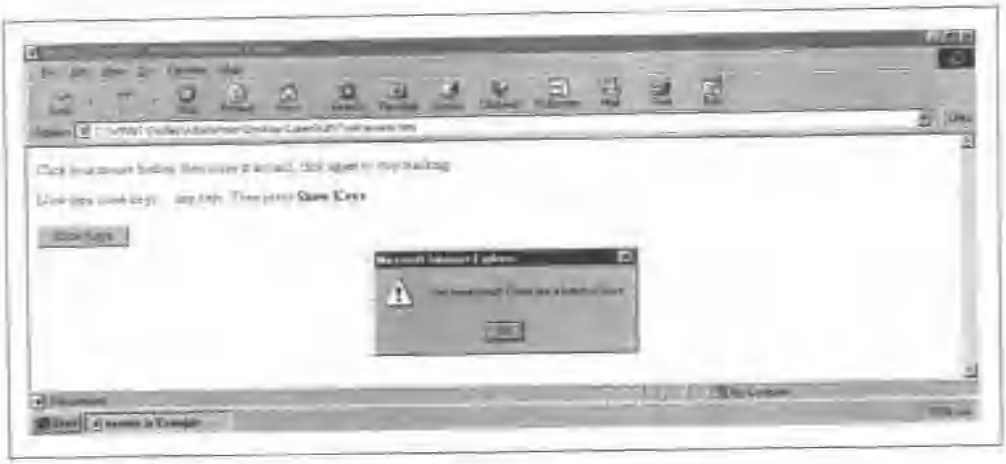


图 6-9　用户敲击的按键

现在，你已经熟悉了编制跨浏览器样式表的复杂性。你知道：一种浏览器中使用LAYER标签，而另外一种浏览器中使用DIV标签（注 1）。涉及到事件处理方法的时候，情况也好不了多少。如果你在 *events.html* 中查看源代码，会发现下面两行JavaScript：

```
document.onclick = enableEffects;
document.onkeypress = keepKeys;
```

Onclick 事件处理方法和函数 enableEffects() 有关，*onkeypress* 事件处理方法和函数 keepKeys() 相关。两个函数都在下面给出。注意，两个函数在语法上都不带圆括号。也就是说，代码不会像这样：

```
document.onclick = enableEffects();
document.onkeypress = keepKeys();
```

使用圆括号的话，就会在编译每行代码的时候调用每一个函数。你不会希望这样吧：事件处理方法之间变成由于对函数的引用而扯上关系。看一下例 6-5 中的代码：

注 1：　实际上，可以在Navigator 4.x 中使用DIV标签用于定位，只要你在STYLE属性中引入一个定位值。但是，在Netscape使用文档对象模型（DOM）之前，使用LAYER标签可以访问 *Layer* 对象的所有属性。

例 6-5：events.js

```
var keys = '';
var change = true;
var x1, x2, y1, y2;

function enableEffects(ev) {
  if(change) {
    if(document.layers) {
      x1 = ev.screenX;
      y1 = ev.screenY;
      document.captureEvents(Event.MOUSEMOVE);
      }
    else {
      x1 = event.screenX;
      y1 = event.screenY;
      }
    document.onmousemove = showXY;
    }
  else {
    if (document.layers) {
      x2 = ev.screenX;
      y2 = ev.screenY;
      document.releaseEvents(Event.MOUSEMOVE);
      }
    else {
      x2 = event.screenX;
      y2 = event.screenY;
      document.onmousemove = null;
      }
    window.status = 'Start: (' + x1 + ',' + y1 +
      ')  End: (' + x2 + ',' + y2 + ')  Distance: ' +
      (Math.round(Math.sqrt(Math.pow((x2 - x1), 2)+ Math.pow((y2 - y1), 2))))
      + ' pixels';
    }
  change = !change;
  }

function showKeys() {
  if (keys != '') {
    alert('You have typed: ' + keys);
    window.status = keys = '';
    }
  else { alert('You have to type some keys first.'); }
  }

function showXY(ev) {
  if (document.all) { ev = event; }
  window.status = 'X: ' + ev.screenX + ' Y: ' + ev.screenY;
  }

function keepKeys(ev) {
  if (document.layers) {
```

```
51      keys += String.fromCharCode(ev.which);
52      window.status = 'Key pressed: ' + String.fromCharCode(ev.which);
53      }
54    else {
55      keys += String.fromCharCode(event.keyCode);
56      window.status = 'Key pressed: ' + String.fromCharCode(event.keyCode);
57      }
58    }
```

函数enableEffects()是*click*和*mouseover*事件的核心。注意一下第6，18和33行：

```
if (change) { ....

else { ....

change = !change;
```

变量*change*以true开始，在每一个调用期间改变为它的取反值（例如，变为false，又变回true，如此类推）。由于点击调用了enableEffects()，而且*change*开始为true值。就得到如下面第7～15行所示的结果：

```
if(document.layers) {
  x1 = ev.screenX;
  y1 = ev.screenY;
  document.captureEvents(Event.MOUSEMOVE);
  }
else {
  x1 = event.screenX;
  y1 = event.screenY;
  }
```

这些代码获取了（x，y）坐标，用于*onmousemove*事件处理方法。如果document.layers存在，用户使用的就是Navigator浏览器。运行中所创建的event对象得到传递进来的参数，在本例中名为*ev*。全局变量*x1*和*y1*分别设为用户首次点击得到的(x,y)坐标值（包含在*screenX*和*screenY*中）。然后对文档函数captureEvents()的调用将引起*mousemove*事件处理方法的中断。

如果document.layers不存在，脚本就假设用户使用的是Internet Explorer，采取适当的行动完成上面同样的工作。但是，Microsoft的event模式把event对象定义为一个事件。属性*screenX*和*screenY*将会在此获得。在IE中对于事件获取不要求附加的函数调用，这把我们带到了第16行：

```
document.onmousemove = showXY;
```

然后，*onmousemove*事件处理方法在两种浏览器中都通过使用函数showXY()来完成。我们来浏览一下showXY()函数：

```
function showXY(ev) {
  if (document.all) { ev = event; }
  window.status = 'X: ' + ev.screenX + ' Y: ' + ev.screenY;
  }
```

每一次鼠标移动对showXY()的调用都显示鼠标箭头的（x，y）坐标。以前在同样的跨浏览器方式下也涉及到（x，y）坐标。用户将鼠标移来移去的时候，showXY()三番五次被调用。这要一直继续到用户决定来一次点击，才会导致另外的对enableEffects()的调用。但是，变量*change*在这期间为false值，因此，第19～31行将产生调用：

```
if (document.layers) {
  x2 = ev.screenX;
  y2 = ev.screenY;
  document.releaseEvents(Event.MOUSEMOVE);
  }
else {
  x2 = event.screenX;
  y2 = event.screenY;
  document.onmousemove = null;
  }
window.status = 'Start: (' + x1 + ',' + y1 +
  ')  End: (' + x2 + ',' + y2 + ')  Distance: ' +
  (Math.round(Math.sqrt((x1 - x2)*(x1 - x2)+(y1 - y2)*(y1 - y2))));
```

变量*x1*和*y1*保存首次点击位置的坐标值。现在，变量*x2*和*y2*设置为点击结束位置的坐标值。不再有必要保留*onmousemove*事件处理进程了。因此，在Navigator中，releaseEvents()方法被调用来终止*mousemove*事件。在IE中将document.onmousemove设为null可以有相同的结果。

剩余的所有工作是显示开始和结束位置之间的距离。你还记得距离公式吗？你可能在九年级的几何课上用过它。这里的第29～31行给出了同样的公式。

这负责*onclick*和*onmouseover*事件处理方法，只留下*onkeypress*不管。我们还记得，document.onkeypress被设置来在载入*events.html*期间调用函数keepKeys()。下面的第49～58行就是keepKeys()函数：

```
function keepKeys(ev) {
  if (document.layers) {
    keys += String.fromCharCode(ev.which);
```

```
    window.status = 'Key pressed: ' + String.fromCharCode(ev.which);
    }
  else {
    keys += String.fromCharCode(event.keyCode);
    window.status = 'Key pressed: ' + String.fromCharCode(event.keyCode);
    }
  }
```

关于事件模式

谢天谢地，Navigator 4 和 IE 4 事件模式有一些相同的东西。但是，在写到这里时，仍然有值得研究的重大区别。也许最大的区别在于，当 Navigator 事件从上往下进行的时候（例如，从窗口到框架到文档到表单到文本域），IE 事件却往上走（例如，从文本域到表单到文档到框架到窗口）。你可以在下面的URL中发现更多有关两种模式的更多信息。如果你想要做什么复杂的，跨浏览器的事件处理方法，这些信息是非常重要的。

对于 Navigator，可以访问：

http://developer.netscape.com/docs/manuals/communicator/jsguide4/evnt.htm

对于 IE，请参见：

http://msdn.microsoft.com/developer/sdk/inetsdk/help/dhtml/doc_object/event_model.htm#dom_event

使用相同的浏览器探测技术，空串变量keys被设置为它自身加上所按键的相应字符串。这利用 `String.fromCharCode()` 实现，和所使用的浏览器无关。但是 JavaScript 1.2 将按键表示为 ISO（国际标准化组织）Latin-1 字符。JScript 使用 Unicode 表示法。JavaScript 中的数字存储在 event 对象的 *which* 属性中。JScript 的数字放在 *event.keyCode* 属性中。因此，用户按了一些键，然后选择“Show Keys”按钮。这个函数给出按键的值，然后将它设置为空串。

frames.js

实际应用

强制框架载入

版本要求

JavaScript 1.1

函数

`keepIn()`, `keepOut()`

源文件仅包含两个函数。一个在一个特定的框架设置中保存你的文档。另一个保存其他文档。本例中*frames.js*要求多个HTML页面。例如，试着在你的浏览器中载入*ch06\frameset.html*。这个文件是一个有两个框架的框架设置。一个框架中有作为源文件的*frames.html*。*frames.html*利用*frames.js*来保证*frames.html*始终载入到窗口上部的框架中。这就是载入*frameset.html*会给你如图6-10和图6-11（浏览器载入*frames.html*）所示的结果的原因。

相反，那些想确保只在特定的框架设置载入自己的文件的人，可以使用*frames.js*。参见图6-10，它显示了你试图载入*ch06\frames2.html*时发生的情况。你会得到一个表示违反了frameset策略的警告，然后浏览器载入包含*frames2.html*的相应frameset。你可以在图6-11看到这一情况。

图6-10 frameset策略禁止

提供这个功能的代码非常简单明了。函数`keepOut()`将窗口上部的文档URL同当前框架的URL作比较。如果`location.href`属性不匹配，`keepOut()`就以一个警告对话框表示拒绝执行，并把文档载入到它自己的窗口上部。函数`keepIn()`执行相反的比较，如果比较失败，就将所传参数中包含的URL载入。例6-6给出了*frames.js*的代码。

图 6-11 允许这种情况

例 6-6：frames.js

```
function keepOut() {
  if (top.location.href != self.location.href) {
    alert('This document bows to no frameset.');
    top.location.href = self.location.href;
    }
  }

function keepIn(parentHREF) {
  if (top.location.href == self.location.href) {
    alert('[Wheez]. . .  [Gasp]. . . Must. . . load. . . ' +
      'original. . . frameset.');
    top.location.href = parentHREF;
    }
}
```

images.js

实际应用

图片翻转

版本要求

JavaScript 1.1

函数

`imagePreLoad()`, `imageSwap()`, `display()`

和 *dhtml.js* 中的函数一样，*images.js* 中的代码已经在前边的章节中给出了。第三、四和五章有例 6-7 中所列代码的多种版本。你可以预加载图片，用它们来做图片翻转。

例 6-7：images.js

```
var imgPath    = 'images/';
var arrayHandles = new Array('out', 'over');

for (var i = 0; i < arrayHandles.length; i++) {
  eval('var ' + arrayHandles[i] + ' = new Array()');
  }

for (var i = 0; i < imgNames.length; i++) {
  imagePreLoad(imgNames[i], i);
  }
function imagePreLoad(imgName, idx) {
  for(var j = 0; j < arrayHandles.length; j++) {
    eval(arrayHandles[j] + "[" + idx + "] = new Image()");
    eval(arrayHandles[j] + "[" + idx + "].src = '" + imgPath + imgName +
      arrayHandles[j] + ".gif'");
    }
  }

function imageSwap(imagePrefix, imageIndex, arrayIdx) {
  document[imagePrefix].src = eval(arrayHandles[arrayIdx] + "[" +
    imageIndex + "].src");
  }
functi
```

因为你了解图片翻转的过程，在这里我就不拿任何图表来解释不同之处了。

navbar.js

实际应用

动态页面导航

版本要求

JavaScript 1.1

函数

navbar()

这个源文件只包含一个函数，但这个函数非常有用哦。假设你在网站上有好几个页面内容，每一个都有一个通向其他所有页面的链接导航条。如果JavaScript能够建立一个迷人的导航条，它包括站点中除了当前载入的页面外的其他所有页面，那难道不是很美妙的一件事吗？图6-12显示了*ch06\astronomy.html*。导航条包含站点中对其他页面的链接：*Other Sciences*，*Sports*，*Musicians' Corner* 和 *Cool People*。

图 6-13 显示了 people 链接载入的文档，请注意导航条包括：*Astronomy*，*Other Sciences*，*Sports* 和 *Musicians' Corner*。没有对 *People* 的链接，因为它刚刚已经载入了。你可以在你的页面试做这个功能，你喜欢有多少页面都可以，如果文档要改变，你只需在 *navbar.js* 中作修改，省了不少时间。

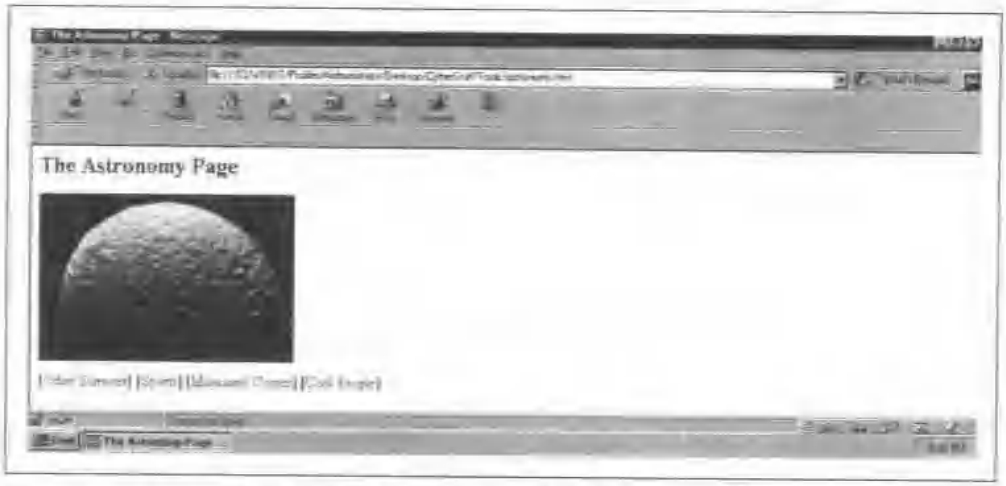


图 6-12 astronomy 页面不带有对 astronomy 的链接

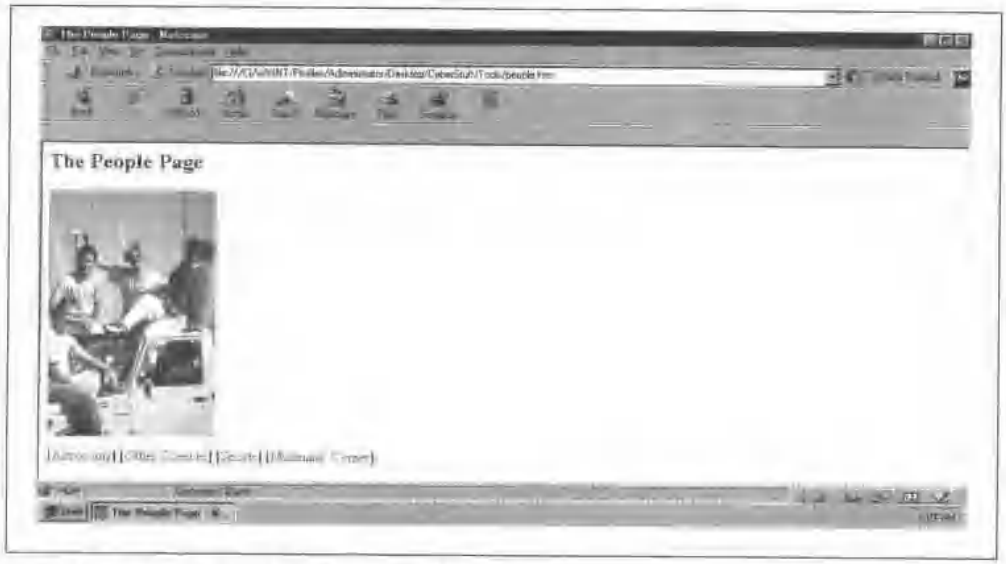


图 6-13 people 页面没有对 people 的链接

有关于这一点的代码是非常基础性的。仅只是用你的网页的文件名来生成*navURLs*数组，以及用你想在链接中显示的文本来构成数组*linkText*。函数navbar()迭代所有的文件名，就那些不在当前文档的*location.href*中出现的页面的相应文字产生一个链接。这很简单。看看例6-8中的代码。

例6-8：navbar.js

```
var navURLs  = new Array('astronomy.html', 'science.html','sports.html',
  'music.htm', 'people.htm');
var linkText = new Array('Astronomy', 'Other Sciences', 'Sports',
  'Musicians\' Corner', 'Cool People');

function navbar() {
  var navStr= '';
  for (var i = 0; i < navURLs.length; i++) {
    if (location.href.indexOf(navURLs[i]) == -1) {
      navStr += ' <B>[</B><A HREF="' + navURLs[i] + '">' + linkText[i] +
        '</A><B>]</B> ';
      }
    }
  document.writeln('<BR><BR>' + navStr);

}
```

你可以很好地扩展这个功能。在文字链接上是不能使用图片（带翻转的）的。如果你有许多的链接但不想把它们罗列在页面上，为什么不把它们嵌入到一个选择列表中呢？那样，你可以真的实现很多连接，而且节省了宝贵的空间。

numbers.js

实际应用

为购物车程序纠正舍入错误，并格式化数字

版本要求

JavaScript 1.1

函数

`twoPlaces()`, `round()`, `totals()`

JavaScript执行的浮点运算和我们所想像的有一点差别；结果是我们的许多计算也我们所期望的有一点差别。DevEdge新闻组FAQ使用例如0.119 * 100 = 11.899999

的形式，它们的网址是 http://developer1.netscape.com:80/support/faqs/champions/javascript.html#2-2。按照元和分显示数字也是很普通的。*numbers.js* 中的函数用来支持这两种情况。所有的这三个函数建立在由 Martin Webb 维护的 http://www.irt.org/script/number.htm 站点基础上。图 6-14 显示了载入的 *ch06\numbers.html*。

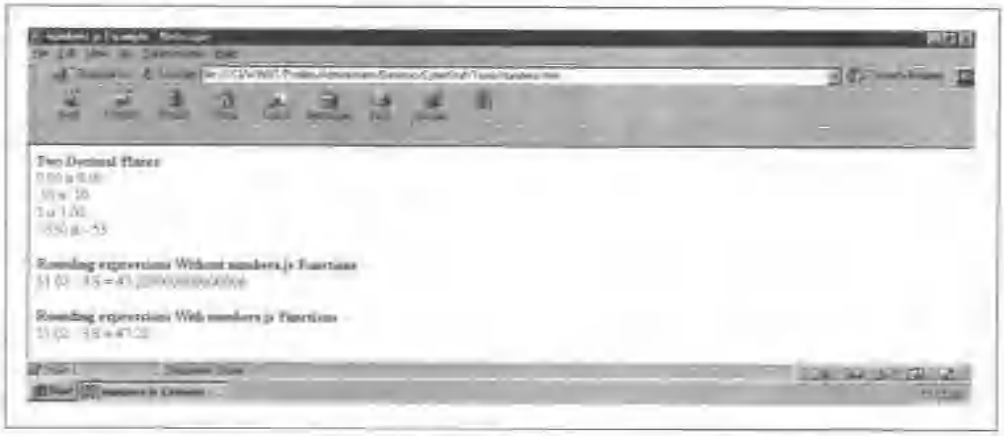


图 6-14　依靠 JavaScript，数字变得更好看了

显示在“Two Decimal Places（两位小数）”标题下的数字表示出函数 twoPlaces() 怎样格式化数字，将它们放在元和分的格式中。其他两个标题显示了表达式 51.02-3.8 在有或没有函数 round() 和 totals() 两种情况下的差别，其中一个起作用。例 6-9 给出了 *numbers.js*。

例 6-9：number.js

```
function twoPlaces(amount) {
  return (amount == Math.floor(amount)) ? amount + '.00' :
         ((amount*10 == Math.floor(amount*10)) ? amount + '0' : amount);
       }

    function round(number,X) {
      X = (!X ? 2 : X);
      return Math.round(number * Math.pow(10,X)) / Math.pow(10,X);
      }

```

函数 twoPlaces() 返回所接收数字的字符串值，它附加 .0 或者 .00，或者在数字已经正确格式化时什么都不附加。这个很长的条件表达式可以翻译成这样的语言：

- 如果数字等于不大于它的最大整数（Math.floor(amount)），返回值是它附加.00后的字符串值。
- 否则，如果数字的10倍等于不大于它的最大整数的10倍（Math.floor(amount) * 10），就返回将它附加.0的串值。
- 否则，就将数字作为字符串返回，因为这时候它的原样已经合格。

至于列在“Two Decimal Places”列表下的舍入错误，函数round()将所接收的数字舍入为整数x的小数位数精度的数字返回。x的默认值为2。因此，如果你没有输入一个x值的话，默认情况是舍入到两位。

objects.js

实际应用

一般对象的创建，对象检查

版本要求

JavaScript 1.1

函数

makeObj(), parseObj(), objProfile()

现在来讨论JavaScript对象。对此有很多要做的工作，但只有很少的时间来完成所有的东西。*objects.js*提供两种应用。一个是一般对象构造器，另一个是基本对象检查。在浏览器中打开*ch06\objects.html*。你可以看到的东西如图6-15所示。

对象检查函数parseObj()和objProfile()给出了两个对象的属性：一个用变量*someObject*来表示；另一个是窗口的位置对象。看看例6-10中的*objects.html*，了解这是怎样开始进行的。

例6-10：objects.html

```
1 <HTML>
2 <HEAD>
3 <TITLE>objects.js Example</TITLE>
4 <STYLE type="text/css">
5 <!--
6 td { font-family: courier new; font-size: 14}
```

图 6-15 objects.html 的结果

```
 7 -->
 8 </STYLE>
 9 <SCRIPT LANGUAGE="JavaScript1.1" SRC="objects.js"></SCRIPT>
10 </HEAD>
11 <BODY>
12 <SCRIPT LANGUAGE="JavaScript1.1">
13 <!--
14
15 function plainOldObject() {
16   this.name = 'some name';
17   this.numba = 1000;
18   this.objInherit = new makeObj('propertyOne', 'thisProperty',
19     'propertyTwo', 'thatProperty', 'propertyThree', 'theOtherProperty');
20   return this;
```

```
21   }
22
23 var someObject = new plainOldObject();
24
25 document.write(objProfile('someObject', 'self.location'));
26 //-->
27 </SCRIPT>
28
29 </BODY>
30 </HTML>
```

注意第23行设变量*someObject*等于一个新的plainOldObject()。plainOldObject()构造器有好几个属性，包括*name*，*numba*以及*objIngerit*。*objIngerit*表示一个对象，这个对象从*objects.js*中的类对象构造器makeObj()中建造得来。看看例6-11中的源文件。

例6-11：objects.js

```
 1 function makeObj() {
 2   if (arguments.length % 2 != 0) {
 3     arguments[arguments.length] = "";
 4     }
 5   for ( var i = 0; i < arguments.length; i += 2 ) {
 6     this[arguments[i]] = arguments[i + 1] ;
 7      }
 8   return this;
 9   }
10
11 function parseObj(obj) {
12   var objStr = '';
13     for (prop in obj) {
14       objStr += '<TR><TD>Property: </TD><TD><B>' + prop +
15         '</B></TD><TD>Type: </TD><TD><B>' + typeof(obj[prop]) +
16         '</B></TD><TD>Value: </TD><TD><B>'  + obj[prop] +
17         '</B></TD></TR>';
18       if (typeof(obj[prop]) == "object") {
19         objStr += parseObj(obj[prop]);
20         }
21       }
22   return objStr;
23   }
24
25 function objProfile() {
26   var objTable = '<TABLE BORDER=2 CELLSPACING=0><TR><TD><H1>' +
27     'Object Profile</H1></TD></TR>';
28   for (var i = 0; i < arguments.length; i++) {
29     objTable += '<TR><TD><BR><BR><H2><TT>' + (i + 1) + ') ' +
30       arguments[i] +  '</H2></TD></TR>';
31     objTable += '<TR><TD><TT><TABLE CELLPADDING=5>' +
32       parseObj(eval(arguments[i])) + '</TABLE></TD></TR>';
33     }
```

```
34    objTable += '</TABLE><BR><BR><BR>';
35    return objTable;
36    }
```

让我们先来看看 makeObj()；这里是源文件的一些代码：

```
function makeObj() {
  if (arguments.length % 2 != 0) {
    arguments[arguments.length] = "";
    }
  for ( var i = 0; i < arguments.length; i += 2 ) {
    this[arguments[i]] = arguments[i + 1] ;
     }
  return this;
  }
```

这个构造器通过给传递进来的成对的参数赋值来创建属性。如果传递进来的参数中有一个不成对的数字（意味着有一个参数没有配对），makeObj()就将一个空串作为附加元素赋给参数数组。现在，每一个参数元素有了一个伙伴。然后 makeObj() 两个两个地迭代参数元素，将对中的第一个元素赋值为对象属性的名字，第二个元素作为先前命名的属性的值。也就是说，调用 makeObj('name', 'Madonna', 'occupation', 'singer/songwriter')将会返回一个有如下属性的对象：

```
this.name = 'Madonna';
this.occupation = 'singer/songwriter';
```

因此，变量 *objIngerit* 现在指向一个对象，有下面的属性：

```
objInherit.propertyOne    = 'thisProperty';
objInherit.propertyTwo    = 'thatProperty';
objInherit.propertyThree  = 'theOtherProperty';
```

注意每一个属性都有作为值的串。你完全可以传递数字、对象，以及诸如此类的量。函数 makeObj()对创建多种多样的对象非常在行，每一个都有不同的属性，而不必对每一个都定义一个构造器。

另外一个受检查的对象是位置对象。它非常直截了当，但是检查工作是怎样进行的呢？函数 objProfile()和 parseObj()一起运作，递归地向下层递进入对象属性，并创建一个结果表格。表的每一行确定对象属性的名称，属性对象的类型，还有相关的值。让我们从 objProfile()开始：

```
function objProfile() {
  var objTable = '<TABLE BORDER=2 CELLSPACING=0><TR><TD><H1>' +
```

```
    'Object Profile</H1></TD></TR>';
  for (var i = 0; i < arguments.length; i++) {
    objTable += '<TR><TD><BR><BR><H2><TT>' + (i + 1) + ') ' +
      arguments[i] + '</H2></TD></TR>';
    objTable += '<TR><TD><TT><TABLE CELLPADDING=5>' +
      parseObj(eval(arguments[i])) + '</TABLE></TD></TR>';
    }
  objTable += '</TABLE><BR><BR><BR>';
  return objTable;
  }
```

objProfile()是你所调用和传递参数的函数。看一下*objects.html*中的第25行代码：

```
document.write(objProfile('someObject', 'self.location'));
```

传递进来的参数根本不是对象。它们是一些字符串。马上就要用它们来表示对象了。传递等价的字符串使JavaScript可以把对象名称显示在页面上。一旦完成必需的表格TR和TD的创建，这些传递到objProfile()中的字符串参数被第12行的eval()函数循环间接引用（dereference），传递到parseObj()中去。现在来看看parseObj()的过程：

```
function parseObj(obj) {
  var objStr = '';
    for (prop in obj) {
      objStr += '<TR><TD>Property: </TD><TD><B>' + prop +
        '</B></TD><TD>Type: </TD><TD><B>' + typeof(obj[prop]) +
        '</B></TD><TD>Value: </TD><TD><B>'  + obj[prop] +
        '</B></TD></TR>';
      if (typeof(obj[prop]) == "object") {
        objStr += parseObj(obj[prop]);
        }
      }
  return objStr;
  }
```

每一个间接引用的字符串以称作*obj*的对象传入。使用*for ... if*循环，parseObj()迭代*obj*中的所有属性，将它的属性、类型和值聚积成字符串，并加以适当的表格标签。parseObj()用typeof()操作访问对象类型。一旦属性、类型和值被加到字符串上，parseObj()就检查特定的属性类型是否为对象。如果是，parseObj()就调用自身，并且把属性（是一个*obj*对象）作为参数传递进去。这个递归允许层层深入的作用，直到最底层，并且显示一个最高层对象的内在层次。

当parseObj()解析完所有的对象时，包含属性、类型、值以及表格标签的整个的字符串表示为变量*objStr*，最终返回到函数objProfile()。然后objProfile()将这个字符串连接到它所创建的其他表格行和单元。这个保存在变量*objTable*中的字符串，最后在*objects.html*的第25行被写到页面上。

注意：这些对象检查函数用来对付相关的小对象，比如用户创建的对象。如果有太多的递归，Navigator和IE都将停滞脚本的运行。例如，我们来试试改变第25行的*objects.html*的下面内容：

```
document.write(objProfile('someObject', 'self.location'));
```

改变为：

```
document.write(objProfile('document'));
```

现在，将文档载入到IE中。你肯定将得到一个堆栈溢出消息。在Navigator中拿第25行来试试：

```
document.write(objProfile('window'));
```

你将得到："JavaScript Error: too much recursion (JavaScript错误：递归过多)"。

strings.js

实际应用

字符串处理，按字母顺序排列，计数器

版本要求

JavaScript 1.2

函数

camelCaps(), prepStr(), wordCount(), reorder()

对于你要对字符串做的处理，这些函数会给你很大的帮助，大约绝大部分的字符串都是由用户输入的。在浏览器中打开*ch06\string.html*，如图6-16所示。

图 6-16 用来接受字符串数据的三个表单，从字数开始

在这里，你有三个表单，使用三个函数。第一个表单包含一个用来输入文字的TEXTAREA。待用户输入了文字，按“Count”按钮之后，函数wordCount()产生一个带表格的新页面。表格列出了每一个TEXTAREA中输入的单词，以及每个单词出现的次数。你可以在图6-17中看见结果。

第二个表单也包含一个用来输入文字的TEXTAREA。然后用户可以选择“Upper”或者“Lower”来将每个单词的首字母转变为大写或者小写。函数camelCaps()负责这一工作。多半在用户必须输入名字和地址的时候，你可能会发现这个函数对于表单的确认非常有利。参见图6-18中的转变示例。

情况没有什么改变，最后一个表单仍然有一个TEXTAREA，等用户输入完文字之后，它能将单词情况分类。这是一个非常好的用于按字母排序的脚本。参见图6-19所示的结果。选择“Sort”若干次，使文字按照递增或递减的顺序排列。你已经看完了上面的步骤；现在该来看看是谁在处理字符串了。

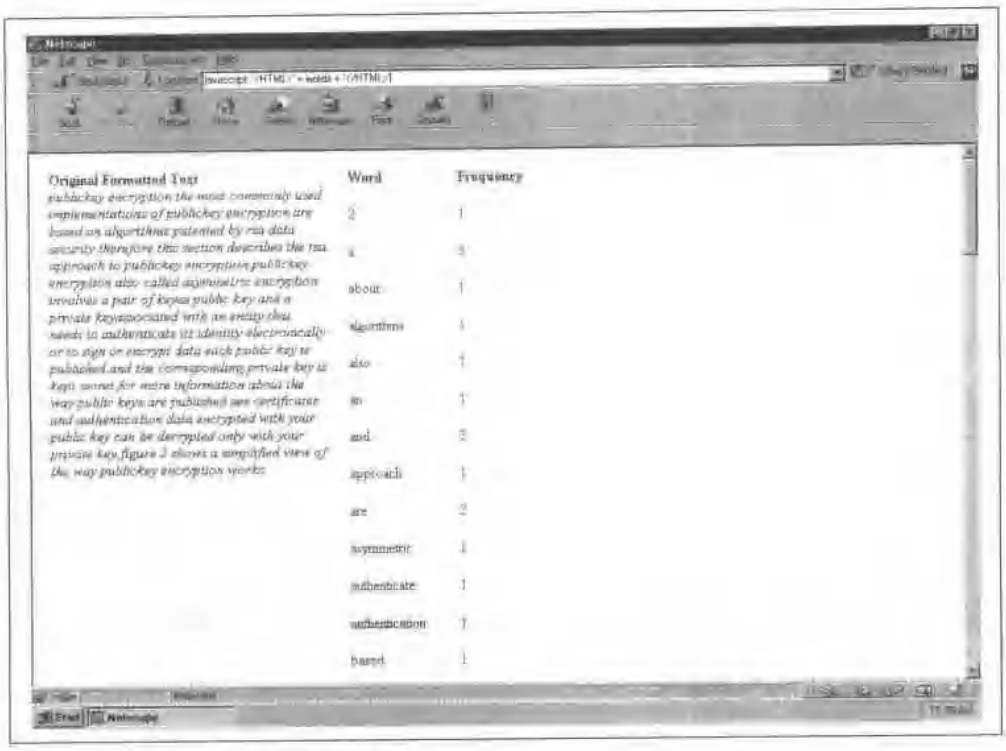


图 6-17　单词和分别出现次数的表格

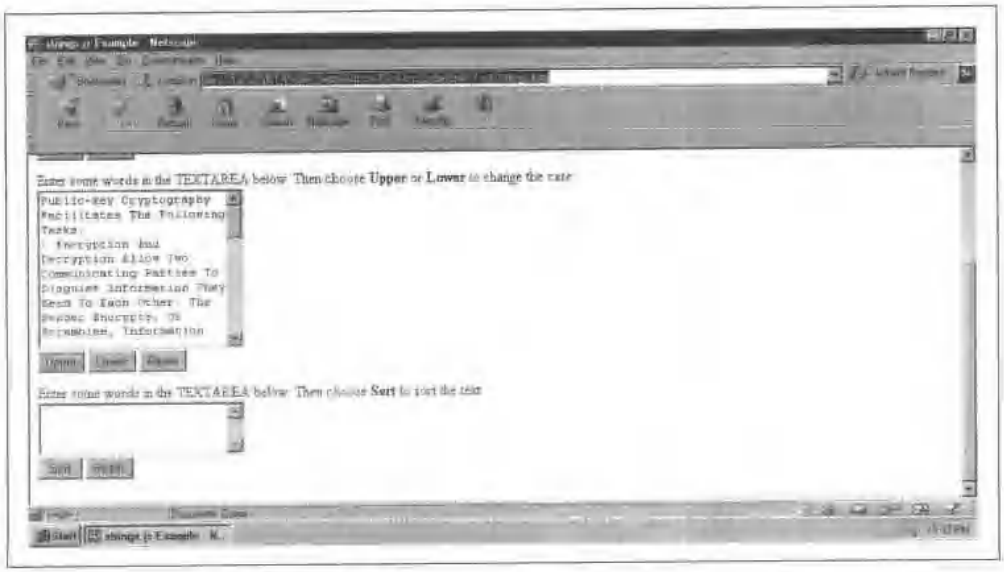


图 6-18　将每个单词的首字符改为大写

图 6-19 按字母顺序排列单词

例 6-12 给出了 *strings.js* 的代码。

例 6-12：strings.js

```
function wordCount(str, output) {
  var wordArray = new Array();
  str = prepStr(str);
  var tempArray = str.split(' ').sort();
  var count = 1;
  for (var i = 0; i < tempArray.length; i++) {
    if (wordArray[tempArray[i]]) {
      wordArray[tempArray[i]]++;
      }
    else { wordArray[tempArray[i]] = 1; }
    }
  if (output) { return wordArray; }
  else {
    var arrStr = '';
    for (word in wordArray) {
      if (word != "") {
        arrStr += '<TR><TD>' + word + '</TD><TD>' + wordArray[word] +
          '</TD></TR>';
        count++;
        }
      }
```

```
22      return '<TABLE BORDER=0><TR><TD WIDTH=300 VALIGN=TOP ROWSPAN=' +
23        count + '><B>Original Formatted Text</B><BR><I>' + str +
24        '</I><TD><B>Word</B><TD><B>Freqency</B></TR>' + arrStr +
25        '</TABLE>';
26      }
27    }
28
29 function prepStr(str) {
30    str = str.toLowerCase();
31    str = str.replace(/['"-]/g, "");
32    str = str.replace(/\W/g, " ");
33    str = str.replace(/\s+/g, " ");
34    return str;
35    }
36
37 function camelCaps(str, theCase) {
38    var tempArray = str.split(' ');
39    for (var i = 0; i < tempArray.length; i++) {
40      if (theCase) {
41        tempArray[i] = tempArray[i].charAt(0).toUpperCase() +
42          tempArray[i].substring(1);
43        }
44      else {
45        tempArray[i] = tempArray[i].charAt(0).toLowerCase() +
46          tempArray[i].substring(1);
47        }
48      }
49    return tempArray.join(' ');
50    }
52 var order = true;
53
54 function reorder(str) {
55    str = prepStr(str);
56    str = str.replace(/\d/g, "");
57    order = !order;
58    if(!order) { str = str.split(' ').sort().join(' '); }
59    else { str = str.split(' ').sort().reverse().join(' '); }
60    return str.replace(/^\s+/, "");
61 }
```

为了利用第一个表单产生一个单词记数，wordCount()完成下面的步骤：

1. 删除所有非字母，数字（或者下划线）或单个空格的符号
2. 创建一个文本中所有单词的数组
3. 计算每一个单词出现的次数
4. 在表格中显示结果

步骤1依靠另一个函数prepStr()来完成。这里是第29～35行的代码：

```
function prepStr(str) {
  str = str.toLowerCase();
  str = str.replace(/['"-]/g, "");
  str = str.replace(/\W/g, " ");
  str = str.replace(/\s+/g, " ");
  return str;
  }
```

字符串首先被全部转变为小写字母（没有必要把“Boat”和“boat”作为两个独立的单词来计数）。然后，函数执行一系列的字符串置换。引号和破折号在第31行被删除。任何不是字母、数字或下划线的符号在第32行被转化为单个空格。最后，一个或多个连续的空格被转变为单个的空格。这在很大程度上整理了文本，使连续的代码不至于会将？或者“string”（包含引号）算为一个单词。

我们回到wordCount()。prepStr()返回一个由空格分隔的单词字符串。这就使split()更加容易处理上面的第二步。第三步发生在第6～11行：

```
for (var i = 0; i < tempArray.length; i++) {
  if (wordArray[tempArray[i]]) {
    wordArray[tempArray[i]]++;
    }
  else { wordArray[tempArray[i]] = 1; }
  }
```

在第二个表单中，用户输入的文字作为一个字符串传递到函数camelCaps()中。CamelCaps()也接收到了第二个参数，用一个布尔值来表示转换规则是大写还是小写。看看下面的代码：

```
function camelCaps(str, theCase) {
  var tempArray = str.split(' ');
  for (var i = 0; i < tempArray.length; i++) {
    if (theCase) {
      tempArray[i] = tempArray[i].charAt(0).toUpperCase() +
        tempArray[i].substring(1);
      }
    else {
      tempArray[i] = tempArray[i].charAt(0).toLowerCase() +
        tempArray[i].substring(1);
      }
    }
  return tempArray.join(' ');
  }
```

局部变量 *tempArray* 被设置为一个包含文本中所有单词的数组。在本例中，一个单词表示空格之间的任何文字。现在的事情是要迭代所有的单词，将它们每一个的首字母转换成等价的大写或者小写字母。所有单词的首字母都转变完以后，函数返回一个由空格分隔的新单词的字符串。函数 camalCaps()实际上是重现在 split()中被删除的空格。

对于最后一个表单，函数 reorder()执行 sort()或者颠倒了的 sort()。还是自己来看看吧：

```
var order = true;
function reorder(str) {
  str = prepStr(str);
  str = str.replace(/\d/g, "");
  order = !order;
  if(!order) { str = str.split(' ').sort().join(' '); }
  else { str = str.split(' ').sort().reverse().join(' '); }
  return str.replace(/^\s+/, "");
  }
```

正如在 wordCount()中一样，prepStr()对传递进来的字符串作格式化处理。但是对于这个函数，我也通过对str.replace(/\d/g,"")的调用来删除数字。这可以保证把焦点放在单词上而不是数字上。变量order被取反，它用来确定种类顺序，并且作好下次点击“Sort”的准备工作，现在，考虑一下如果单词按照传统方法或者颠倒的方法分类，会发生什么情况？对于传统的分类方法：

1. 文本被分离成数组。
2. 那些数组元素被分类。
3. 数组被组合成一个字符串。

在颠倒的分类方法中，有几个步骤：

1. 文本被分离成数组。
2. 那些数组元素被分类。
3. 元素被颠倒。
4. 数组被组合成一个字符串。

Strings.js 的第 58～59 行利用顺序的值来确定采取哪一种方法。然后，返回字符串（减去任何可能是由 join() 创建的开头的空格）。

应用程序扩展

在这里应用确实有很多限制。当然，你可以在这些文件中添加出色的函数，或者使现有的函数锦上添花。但是，说句老实话，我怀疑你会想把函数放在同一个源文件中。也许你在设计一个方便的站点，而且你想用网站的名称来命名 *.js* 文件。这也不错。插入你所需要的函数，你就得到了一个工具箱。（将它命名为 *toolkit.js* 怎么样？）有一件重要的事应该记住，你需要使用一个工作良好的系统。不要让 *.js* 文件反而制约你。也可以采用其他方法。我倒真想知道你究竟会如何处理。

第七章

基于 cookie 的用户参数选择

应用程序要点

- 可更改的链接、背景图片和字体设置
- 灵活的用户优先选择应用程序
- 几十种设计组合
- 语言参数选择，颜色、图片和 DHTML 的简单定制

JavaScript 技巧

- 再次使用命名协议
- DHTML

本章包含一个不值一提的应用程序，但是先不要跳到下一章。我忘了说，这个应用程序的代码可以帮助你在网站上添加一些酷毙了的功能。我指的是设置用户参数选择。想想吧，每一个用户在网上冲浪的时候，他们的脑子里想的都是哪一个字？

“我”。

对，用户通常都是“自私的”家伙，他们总是考虑自己的利益和兴趣，才不管你是怎么想的呢。人们不管做什么，一般都倾向于寻求能激起个人意识的东西。这就是为什么 DHTML Zone 站点（http://www.dhtmlzone.com/）要挂出古里古怪的 DHTML，为什么购物者要跑到 Shopping.com（http://www.shopping.com/ibuy/），为什么天文爱好者（可能就有我在内，看见了吗？我刚才还去过呢）要访问天空与望远镜站点（http://www.skypub.com/）的原因。这年头，市场的商人、广告客户及售货员都要使用 Web。Web 也没有什么不同。使用一个虚拟的应用程序，本章将告诉你怎样在网站上添加个性化的功能，即使它只是和保存一个用户名一样简单。使用 JavaScript 的 cookie，访问者就可以定制自己在网站上的经历。

假如你已经建了一个站点，访问者是有一些额外现金的 Internet 投资者。用户在这个叫做 Take-A-Dive 经纪业务的虚拟的华尔街网络中心免费获得成员资格。他们其

实不能在这里交易，但他们可以得到一个定制的个人主页，并附带选择对其他经济网站和新闻信息的链接。图7-1显示了用户首次访问快速定制网页(*\ch07\dive.html*)时的情况。由于这是他第一次登录，所以要进入到用户参数选择页面。

图 7-1 首次进入用户参数选择界面

图 7-2 给出了用户参数选择页面 (*\ch07\dive.html*)。这是一个很长的表单，允许用户输入名字、年龄、职业，以及投资策略类型。一系列的复选框（checkbox）允许用户选择将哪一个（如果有）经济领域的链接放到定制的主页中去。

表单由一系列的可选列表构成，用户可以选择一幅背景图片、字体大小以及从所提供的小图片中选择字库。所有的东西都选择完了以后，点击“Save”按钮将选择结果写到所用的浏览器的cookie文件。然后用户得到是否要再看看更改内容的提示，如图7-3所示。选择“OK”将用户引到*dive.html*，这个页面给出用户所选的每一样东西 —— 个人信息、背景、甚至字体素材。

真是非常酷哦，但如果用户想要修改又怎么办呢？只要选择“Set Preferences”链接，就会回到*prefs.html*。注意用户的选择结果在这里仍然保留。所有的个人信息完整存在。最近选上的复选框仍然被选中，就连背景图片和字体细节都完整无缺。现在，用户可以用背景图片和字体规格来试验哪一种看起来更好一点。图7-4显示了一种可能的结合方式。

执行条件

这个应用程序全部是用JavaScript 1.2来写的。这是样式表和串置换的要求。用户会需要至少是Navigator 4.x，或者Internet Explorer 4.x。这个应用程序可以进行

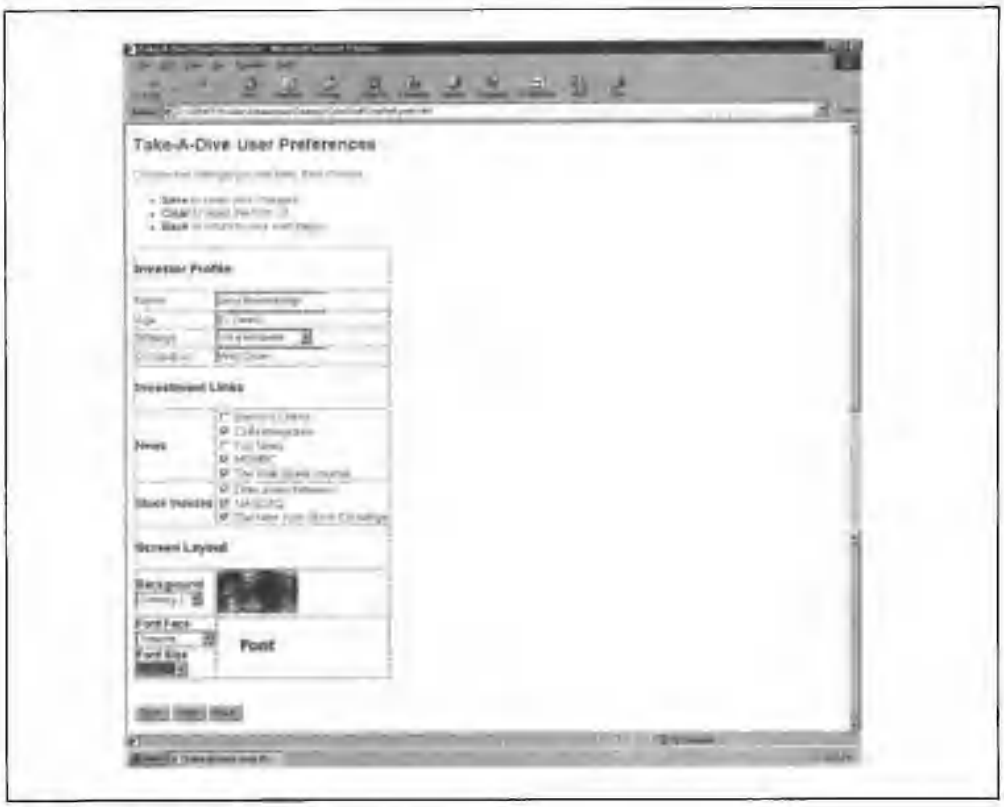


图7-2 设置用户参数选择

广阔的扩展，它使用了简单的样式表。唯一的局限性在于你的DHTML运用能力（换句话说，并不太多）。

这是一个基于cookie的应用程序，因此你要受到每一个浏览器cookie的局限。要参见其他内容，可以用下面所列的URL。

Netscape Navigator:

http://developer1.netscape.com:80/docs/manuals/communicator/jsguide4/cookies.htm

IE:

http://msdn.microsoft.com/msdn-online/workshop/author/dhtml/reference/properties/cookie.asp

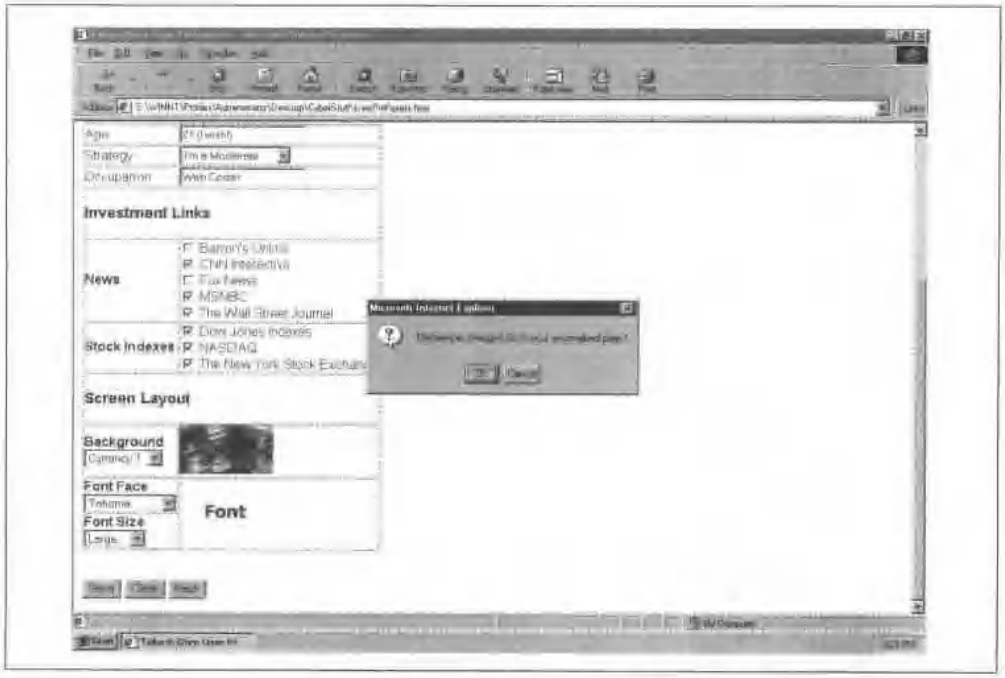


图 7-3 点击 Save 之后，用户得到是否显示个人化页面的询问

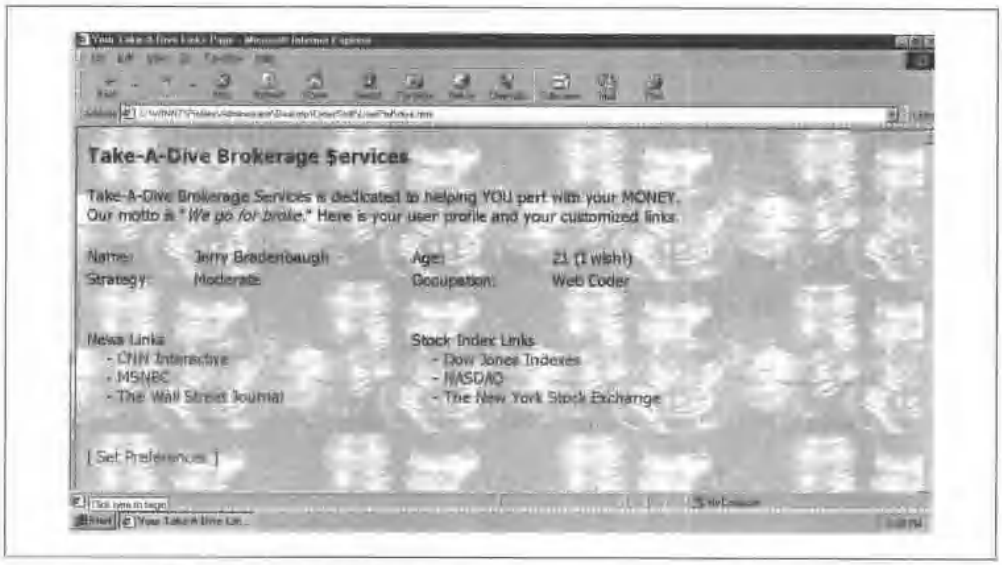


图 7-4 包括字体和背景图片的定制页面

但是不用担心。这个应用程序在Navigator和IE 4.x浏览器cookie的限制下仍然可以很好地运行。

语法细则

这个应用程序由两个HTML页面（*prefs.html*和*dive.html*）和一个JavaScript源文件*cookies.js*构成。下面的列表描述了每一个文件：

prefs.html

这个页面用来为*dive.html*设置参数选择。这个页面也同样受cookie信息的影响，因为cookie信息用来确定用户已经选择了哪一个设置，然后从中生成表单。

dive.html

这个页面从cookie中获取信息，是一个定制设计。

cookies.js

这个文件包含的函数用来将用户参数设置写到cookie，并且从cookie中获取参数。*Cookie.js*嵌入到两个HTML文件中。`GetCookie()`和`SetCookie()`函数你已经在这里的其他两个文件中看到过了。

文件*prefs.html*和*dive.html*都是新的，但*cookies.js*是包括在第六章中的一个源文件。你可以在那里看到关于它的功能的讨论。

prefs.html

虽然屏幕显示顺序引导你按正确的事件发展顺序前进（也就是用户开始的时候并没有参数选择设置），我们来假设用户以前已经设置了用户参数选择，现在正要回到*prefs.html*做一些修改。我想，你会觉得这很容易继续往下进行。例7-1给出了*prefs.html*。

例7-1：prefs.html

```
1 <HTML>
2 <HEAD>
3 <TITLE>Take-A-Dive User Preferences</TITLE>
4 <STYLE type="text/css">
5   BODY, TD { font-family: Arial; }
```

```
 6 </STYLE>
 7 <SCRIPT LANGAUGE="JavaScript1.2" SRC="cookies.js"></SCRIPT>
 8 <SCRIPT LANGUAGE="JavaScript1.2">
 9
10 var imagePath = 'images/';
11 var newsNames = new Array(
12   new Array('The Wall Street Journal','http://www.wsj.com/'),
13   new Array('Barron\'s Online','http://www.barrons.com/'),
14   new Array('CNN Interactive','http://www.cnn.com/'),
15   new Array('MSNBC','http://www.msnbc.com/'),
16   new Array('Fox News','http://www.foxnews.com/')
17   );
18
19 var indexNames = new Array(
20   new Array('The New York Stock Exchange','http://www.nyse.com/'),
21   new Array('NASDAQ','http://www.nasdaq.com/'),
22   new Array('Dow Jones Indexes','http://www.dowjones.com/')
23   );
24
25 var strategy = new Array(
26   new Array('Cheap', 'I\'m Really Cheap'),
27   new Array('Stingy', 'I\'m Pretty Stingy'),
28   new Array('Conservative', 'I\'m Conservative'),
29   new Array('Moderate', 'I\'m a Moderate'),
30   new Array('Agressive', 'I\'m Aggressive'),
31   new Array('Willing to sell mother', 'I\'d Sell My Mother!')
32   );
33
34 var background = new Array(
35   new Array(imagePath + 'goldthumb.gif', 'Gold Bars'),
36   new Array(imagePath + 'billsthumb.gif', 'Dollar Bills'),
37   new Array(imagePath + 'fistthumb.gif', 'Fist of Cash'),
38   new Array(imagePath + 'currency1thumb.gif', 'Currency 1'),
39   new Array(imagePath + 'currency2thumb.gif', 'Currency 2')
40   );
41
42 var face = new Array(
43   new Array('times', 'Times Roman'),
44   new Array('arial', 'Arial'),
45   new Array('courier', 'Courier New'),
46   new Array('tahoma', 'Tahoma')
47   );
48
49 var size = new Array(
50   new Array('10', 'Small'),
51   new Array('12', 'Medium'),
52   new Array('14', 'Large'),
53   new Array('16', 'X-Large')
54   );
55
56 indexNames = indexNames.sort();
57 newsNames = newsNames.sort();
58
```

```
var allImages = new Array();

var imageNames = new Array(
  'courier10', 'courier12', 'courier14', 'courier16',
  'arial10', 'arial12', 'arial14', 'arial16',
  'times10', 'times12', 'times14', 'times16',
  'tahoma10', 'tahoma12', 'tahoma14', 'tahoma16',
  'goldthumb','billsthumb','fistthumb','currency1thumb','currency2thumb',
  'blank'
  );

for (var i = 0; i < imageNames.length; i++) {
  allImages[i] = new Image();
  allImages[i].src = imagePath + imageNames[i] + '.gif';
  }

function makePath(formObj) {
  var fontName = imagePath +
    formObj.face.options[formObj.face.selectedIndex].value +
    formObj.size.options[formObj.size.selectedIndex].value + '.gif';
  swapImage("fontImage", fontName);
  }

function swapImage(imageName, imageBase) {
  document[imageName].src = imageBase;
  }

function genSelect(name, select, onChangeStr) {
  var optStr = "";
  var arrObj = eval(name);
  for (var i = 0; i < arrObj.length; i++) {
    optStr += '<OPTION VALUE="' + arrObj[i][0] +
     (i == select ? '" SELECTED' : '"') + '>' + arrObj[i][1];
    }
  return '<SELECT NAME="' + name + '"' + (onChangeStr ? ' onChange="' +
    onChangeStr + ';"' : '') + '>' + optStr + '</SELECT>';
  }
function genBoxes(handle, arrObj) {
  var boxStr = '';
  for (var i = 0; i < arrObj.length; i++) {
    boxStr += '<INPUT TYPE=CHECKBOX NAME="' + handle + i + '" VALUE="' +
      arrObj[i][0] + ',' + arrObj[i][1] + '"> ' + arrObj[i][0] + '<BR>';
    }
  return boxStr;
  }

function getPrefs(formObj) {
  var prefStr = GetCookie('userPrefs');
  if (prefStr == null) { return false; }
  var prefArray = prefStr.split('-->');
  for (var i = 0; i < prefArray.length; i++) {
    var currPref = prefArray[i].split('::');
    if (currPref[1] == "select") {
```

```
113       formObj[currPref[0]].selectedIndex = currPref[2];
114       }
115     else if (currPref[1] == "text") {
116       formObj[currPref[0]].value = currPref[2];
117       }
118     else if (currPref[1] == "checkbox") {
119       formObj[currPref[0]].checked = true;
120       }
121     }
122   return true;
123   }
124
125 function setPrefs(formObj) {
126   var prefStr = '';
127   var htmlStr = '';
128   for (var i = 0; i < formObj.length; i++) {
129     if (formObj[i].type == "select-one") {
130       prefStr += formObj[i].name + '::select::' +
131       formObj[i].selectedIndex + '-->';
132       htmlStr += formObj[i].name + '=' +
133         formObj[i].options[formObj[i].selectedIndex].value + '-->';
134       }
135     else if (formObj[i].type == "text") {
136       if (formObj[i].value == '') { formObj[i].value = "Not Provided"; }
137         prefStr += formObj[i].name + '::text::' +
138         safeChars(formObj[i].value) + '-->';
139         htmlStr += formObj[i].name + '=' + formObj[i].value + '-->';
140       }
141     else if (formObj[i].type == "checkbox" && formObj[i].checked) {
142       prefStr += formObj[i].name + '::checkbox::' + '-->';
143       htmlStr += formObj[i].name + '=' + formObj[i].value + '-->';
144       }
145     }
146   SetCookie('userPrefs', prefStr, expiry);
147   SetCookie('htmlPrefs', htmlStr, expiry);
148   if (confirm('Preferences changed. Go to your personalized page?')) {
149     self.location.href = "dive.html";
150     }
151   }
152
153 function safeChars(str) {
154   return str.replace(/::|=|-->/g, ':;');
155   }
157 function populateForm(formObj) {
158   if (getPrefs(formObj)) {
159     makePath(formObj);
160     swapImage('bkgImage',
161    formObj.background.options[formObj.background.selectedIndex].value);
162     }
163   else { resetImage(document.forms[0]); }
164   }
165
166 function resetImage(formObj) {
```

```
167   swapImage('bkgImage', formObj.background.options[0].value);
168   swapImage('fontImage', imagePath + formObj.face.options[0].value +
169     formObj.size.options[0].value + '.gif');
170   }
171
172 </SCRIPT>
173 </HEAD>
174
175 <BODY BGCOLOR=FFFFFF onLoad="populateForm(document.forms[0]);">
176 <DIV ID="setting">
177 <H2>Take-A-Dive User Preferences</H2>
178 Choose the settings you like best, then choose<BR>
179 <UL>
180   <LI><B>Save</B> to keep your changes,
181   <LI><B>Clear</B> to reset the form, or
182   <LI> <B>Back</B> to return to your links page.
183 </UL>
184
185 <FORM>
186 <TABLE BORDER=1 CELLBORDER=0 CELLPADDING=0 CELLSPACING=1>
187   <TR>
188     <TD COLSPAN=2>
189     <BR>
190     <H3>Investor Profile</H3>
191     </TD>
192   </TR>
193   <TR>
194     <TD>Name</TD>
195     <TD><INPUT TYPE=TEXT NAME="investor"></TD>
196   </TR>
197   <TR>
198     <TD>Age</TD>
199     <TD><INPUT TYPE=TEXT NAME="age"></TD>
200   </TR>
201   <TR>
202     <TD>Strategy</TD>
203     <TD>
204       <SCRIPT LANGUAGE="JavaScript1.2">
205       document.write(genSelect('strategy', 3));
206       </SCRIPT>
207     </TD>
208   </TR>
209   <TR>
210     <TD>Occupation</TD>
211     <TD>
212       <INPUT TYPE=TEXT NAME="occupation">
213     </TD>
214       <TR>
215         <TD COLSPAN=2>
216           <BR>
217           <H3>Investment Links</H3>
218         </TD>
219       </TR>
```

```
220       <TR>
221         <TD><B>News<B></TD>
222         <TD>
223           <SCRIPT LANUAGE="JavaScript1.2">
224           document.write(genBoxes('newsNames'));
225           </SCRIPT>
226         </TD>
227       </TR>
228       <TR>
229         <TD><B>Stock Indexes</B></TD>
230         <TD>
231           <SCRIPT LANUAGE="JavaScript1.2">
232           document.write(genBoxes('indexNames'));
233           </SCRIPT>
234         </TD>
235       </TR>
236       <TR>
237        <TD COLSPAN=2>
238        <BR>
239        <H3>Screen Layout</H3>
240        </TD>
241       </TR>
242       <TR>
243         <TD>
244          <B>Background</B>
245          <BR>
246          <SCRIPT LANGUAGE="JavaScript1.2">
247           document.write(genSelect('background', 0,
248             "swapImage('bkgImage',
249             this.options[this.selectedIndex].value)"));
250           </SCRIPT>
251           </TD>
252           <TD>
253           <IMG SRC="images/blank.gif"
254             NAME="bkgImage" WIDTH=112 HEIGHT=60>
255           </TD>
256     </TR>
257      <TR>
258         <TD>
259           <B>Font Face</B>
260           <BR>
261           <SCRIPT LANGUAGE="JavaScript1.2">
262           document.write(genSelect('face', 0, "makePath(this.form)"));
263           </SCRIPT>
264           </TD>
265           <TD ROWSPAN=2>
266             <IMG SRC="images/blank.gif" NAME="fontImage"
267               WIDTH=112 HEIGHT=60>
268           </TD>
270     </TR>
271     <TR>
272        <TD>
273          <B>Font Size</B>
```

```
274            <BR>
275            <SCRIPT LANGUAGE="JavaScript1.2">
276            document.write(genSelect('size', 0, "makePath(this.form)"));
277            </SCRIPT>
278          </TD>
279       </TR>
280 </TABLE>
281 <BR><BR>
282 <INPUT TYPE=BUTTON VALUE="Save" onClick="setPrefs(this.form);">
283 <INPUT TYPE=RESET VALUE="Clear" onClick="resetImage(this.form);">
284 <INPUT TYPE=BUTTON VALUE="Back" onClick="location.href='dive.html';">
285 <! -
286 <INPUT TYPE=BUTTON VALUE="Show"
287   onClick="alert(GetCookie('userPrefs'));alert(GetCookie('htmlPrefs'));">
288 <INPUT TYPE=BUTTON VALUE="Erase"
289   onClick="DeleteCookie('userPrefs'); DeleteCookie('htmlPrefs');">
290 //-->
291 </FORM>
292 </DIV>
293 </BODY>
294 </HTML>
```

第10~68行的内容相当理论化。除了确定图片的一个路径位置（第10行）外，这里的变量都用于*dive.html*的设计。每一个变量都被设置为一个“多维”数组。多维数组从本质上来说是一个包含许多数组的数组。例如，这里有一个一维的数组：

```
var oneDimension = new Array("This", "That", "The Other");
```

利用一对括弧和相应索引，你可以很容易地访问到每一个元素，比如oneDimension[0]。在一个“多维”数组中，每一个元素是包括许多元素的数组，就像这样：

```
var twoDimension =
  new Array(
    new Array(1,2,3),
    new Array(4,5,6),
    new Array(7,8,9)
    );
```

现在，twodimension[0]引用new Array(1,2,3)来代替诸如*This*的字符串。为了能访问到值1，2，或者3，你必须使用第二对括号，就像这样：

```
twoDimension[0][0]    // 访问1
twoDimension[0][1]    // 访问2
twoDimension[0][2]    // 访问3
twoDimension[1][0]    // 访问4
twoDimension[2][1]    // 访问5
twoDimension[3][2]    // 访问6
```

我之所以在“多维”上使用引号，是因为JavaScript在技术上根本不能创建多维数组。它只能模仿。我们回到script中，第11～68行中所定义的每一个变量被设置为一个信息的数组，如表7-1所示。

表7-1 表示用户页面相关信息的变量

数组名	内容
newsNames	一些在线经济新闻资源的名称和URL
indexNames	一些在线股票指标的名称和URL
strategy	可以区别用户阶层的投资策略的名称和类型
background	可用背景图片的名称和URL
face	图片句柄（马上就要讨论到）和可用字库的名称
size	图片句柄和可用字体尺寸的名称
allImages	一个当前为空，用来存储预载图片以便存取数组
imageNames	包含图片句柄的一个数组，这些串用来协助图片的预载

顾名思义，数组的每一个元素本身是一个包括两个元素的数组。基本上，一个元素包含一个用来显示的字符串；另一个包含一个用来做幕后工作的字符串。例如，`size[0][0]`等于10。这个值将会用来设置字体尺寸的像素值。而`size[0][1]`等于*Small*，是显示在页面上的描述内容。用户使用“small”文字比使用10个字母的词要简单快捷一点。对所有其他的数组基本上也一样。

第56～57行调用了数组对象的`sort()`函数来更有序地安排新闻和股市行情的链接。这不是非要不可的，但只是两行程序，可以算是有益无害。*Prefs.html*只用为数不多的一些图形，因此将它们预载入是一个很好的主意。第70～73行如下所示：

```
for (var i = 0; i < imageNames.length; i++) {
     allImages[i] = new Image();
     allImages[i].src = imagePath + imageNames[i] + '.gif';
     }
```

*imageNames*中的每一个元素都是一个字符串。等在变量*imagePath*和*string.gif*之间作连接的时候，这个图片句柄使迭代*imageNames*和preload中所有必要的元素成为可能。顺便说一句，要记住，图片的句柄（比如*courier10*）可不能任意命名。命名协议待会儿就会用到。

如果浏览SCRIPT标签中剩余的代码，你会看见其他所有的东西都用函数来定义。因此，所执行的任何代码都可以在其他地方被调用。这种情况发生了两次：

- HTML的内容被载入时
- 在BODY标签的*onLoad*事件处理方法

接下来，让我们来看看紧接JavaScript的HTML，在第174～294行。看一下例7-1的代码。

参数选择表单

界面是一个表单，它包含一个文本域，复选框以及用户用来指定参数选择列表元素。创建文本域时，按需要硬性编码每一项和设置名称可能都不是那么容易（至少在本例中）。对名字、年龄和职业的输入发生在第195、199和212行。

下一件要决定的事情是用户使用的投资策略。类别范围从非常保守（"吝啬"）到非常富有挑战性。用户不像以前那样填写文本域，而是从一个选项列表中选取。我们可以利用一个JavaScript函数在运行中写列表，代替编写每一个OPTION标签。函数genSelect()在第205、247～249、262和276行轻盈地出现在我们面前。第205行负责投资策略列表，而其他的调用为背景、字体形式和字体尺寸提供列表。看一下第86～95行的genSelect()：

```
function genSelect(name, select, onChangeStr) {
  var optStr = "";
  var arrObj = eval(name);
  for (var i = 0; i < arrObj.length; i++) {
    optStr += '<OPTION VALUE="' + arrObj[i][0] +
    (i == select ? '" SELECTED' : '"') + '>' + arrObj[i][1];
    }
  return '<SELECT NAME="' + name + '"' + (onChangeStr ? ' onChange="' +
    onChangeStr + ';"' : '') + '>' + optStr + '</SELECT>';
  }
```

你可能在第五章中见到过一个和genSelect()有几分相似的函数，它就是*ImageMachine*。那个函数严格地基于将整数作为串值使用来产生选择列表。我们在这里所讨论的函数不基于数字来产生OPTION标签，但是却基于一个包含许多元素的数组。我们在第205行调用了genSelect()：

```
document.write(genSelect('strategy', 3));
```

这个函数接收到两个参数：字符串 *strategy* 和数字3。你也许正在想。“等等...有一个叫做 *strategy* 的数组。为什么传递一个有相同名字的字符串呢？”

的确，有一个数组的名字叫 *strategy*。传递一个 *strategy* 数组的拷贝是有意义的，除了在设置用户参数选择的时候每个选择列表需要一个名称来作索引的情况。为了使事情简单一点，每一个选择列表假定为相关数组的名称。这就是为什么会传递等价字符串的原因。

然后，变量 *arrObj* 被设置为去除引号的串值。也就是说，eval (name) 等于 eval-('strategy')，等于 *strategy* 数组的一个索引。现在，genSelect()既有一个为OPTION项目作参量的数组(*arrObj*)，也有一个用来为选择列表赋值的名称(*name*)。

传递进来的第二个参数是一个整数，用来指定 *select*。这个值确定用来作为默认值的选项。如果 *i* 的值等于 *select*，那么和 *select* 相关的OPTION标签将在它的标签中有SELECTED属性。如果 *select* 等于0，那么第一个选项是默认的；如果 *select* 等于1，那么第二个选项就作为默认项，以此类推。你可以参见第91行的条件表达式：

```
(i == select ? '" SELECTED' : '"')
```

迭代完 *strategy* 中的所有元素，并且建立了所有OPTION标签的 *optStr* 之后，在第93～94行，一个简单的串联将所有的OPTION标签和其外部的SELECT标签放在一起。

现在，genSelect()中要定义的第三个参数是什么呢？它叫 *onChangeStr*，不在这个特定的调用中使用。但是，你可以在别处看见它。例如，看看第247～249行的代码：

```
document.write(genSelect('background', 0,
  "swapImage('bkgImage',
  this.options[this.selectedIndex].value)"));
```

在这个调用中，参数 *name* 被设置为字符串 *background*，参数 *select* 设为值0，而参数 *onchangeStr* 被设为值 swapImage('bkgImage',this.options[this.selectedIndex].value)。当genSelect()接收到一个给 *onChangeStr* 的参数，字符串被加到快速选择列表的 *onChange* 事件处理方法中。否则，没有 *onChange* 事件处理方法被添加。第247～249、262和276行的列表利用 *onChange* 事件处理方法来

翻转当前所选项的相应图片。你在字体尺寸类型中从“Small”移到“Medium”到“Large”再到“X-Large”的时候，就会翻转越来越大的图片，以表示更大的字体尺寸。

选择列表表单不是用JavaScript来创建的唯一物件。函数genBoxes()创建了两组复选框：一组用于新闻链接，另一组用于股票指数的链接——在第224和232行进行调用。

这里是位于第97～104行的genBoxes()：

```
function genBoxes(name) {
  var boxStr = '';
  var arrObj = eval(name);
  for (var i = 0; i < arrObj.length; i++) {
    boxStr += '<INPUT TYPE=CHECKBOX NAME="' + handle + i + '" VALUE="' +
    arrObj[i][0] + ',' + arrObj[i][1] + '"> ' + arrObj[i][0] + '<BR>'
    }
  return boxStr;
  }
```

在这里，事情和在genSelect()中发生的情况类似。和所需数组等价的字符串被传递进来，去除引号，产生所要的数组。迭代所有的元素，产生checkbox的一个字符串，最后作为返回值写到文档中。

装载存储过的参数选择

所有的HTML被载入以后，空的表单可以被用户早先的设置（如果有的话）所填充。BODY标签中的*onLoad*事件处理方法引出函数populateForm()，并且将空表单的一个拷贝传递进去以继续运行。下面是第157～164行的populateForm()函数：

```
function populateForm(formObj) {
  if (getPrefs(formObj)) {
    makePath(formObj);
    swapImage('bkgImage', formObj.background.options
    [formObj.background.selectedIndex].value);
          }
     else { resetImage(document.forms[0]); }
     }
```

populateForm()正是一个函数管理员，它调用其他函数来工作。它是这样操作的：如果以前已经作了参数选择设置，就把从cookie中收集来的信息填到适当的表单域

中。然后在屏幕设计部分将背景和字体图片设置来与所选的OPTION标签相匹配。如果还没有做过参数选择设置，就什么也不用做了。PopulateForm()会用函数getPrefs()在cookie中检查用户的参数选择。它在第106～123行：

```
function getPrefs(formObj) {
  var prefStr = GetCookie('userPrefs');
  if (prefStr == null) { return false; }
  var prefArray = prefStr.split('-->');
  for (var i = 0; i < prefArray.length; i++) {
    var currPref = prefArray[i].split('::');
    if (currPref[1] == "select") {
      formObj[currPref[0]].selectedIndex = currPref[2];
      }
    else if (currPref[1] == "text") {
      formObj[currPref[0]].value = currPref[2];
      }
    else if (currPref[1] == "checkbox") {
      formObj[currPref[0]].checked = true;
      }
    }
  return true;
  }
```

getPrefs()也要做一个决定。它这样运作：获取和*userPrefs*名称相关的cookie信息。如果它是一个null值，就返回false。这表示document.cookie的属性*userPrefs*还没有被设置。但是，如果*userPrefs*不等于null，就表示先前已经有用户设置了。在我们的例子中，*userPrefs*等于null，但现在最好看看当*userPrefs*包含可用信息时发生的情况。

如果*userPrefs*包含所需的东西，getPrefs()就按照分隔符来分隔*prefStr*的值，为它们创建一个数组，这些分隔符在setPrefs()中是用来连接每一个设置的。字符串是-->。现在，*prefsArray*中的元素包含用来表示表单元素类型和相关值的字符串，以::分隔。将相关值赋给表单元素需要迭代*prefsArray*中的元素，并依照表单元素的类型来对每一个元素赋值。第110～121行的代码可能会有助于你的理解：

```
for (var i = 0; i < prefArray.length; i++) {
  var currPref = prefArray[i].split('::');
  if (currPref[1] == "select") {
    formObj[currPref[0]].selectedIndex = currPref[2];
    }
  else if (currPref[1] == "text") {
    formObj[currPref[0]].value = currPref[2];
    }
  else if (currPref[1] == "checkbox") {
```

```
            formObj[currPref[0]].checked = true;
            }
        }
```

记住，用户通过三个途径来设置参数选择：

- 从选择列表中选取一项
- 在文本域中输入文本
- 点中 checkbox

因此，*prefsArray* 中的每一个值包含一个表单元素类型标识符（*text*，*checkbox*，或者 *select-one*）和另外一个表示表单元素值的串，这两者都用“::”分隔符分开。这一点你一会儿就更清楚了。下面列出了 *prefsArray* 元素的一些例子。

`strategy::select::0`

这表示叫 *strategy* 的表单元素是一个选择列表，并且选中的是 OPTION 0。

`newsNames0::checkbox::Barron's Online,http://www.barrons.com/`

这表示叫 newsNames0 的表单元素是一个 checkbox，有 Barron 在线的值，http://www.barrons.com/。

`investor::text::Jerry Bradenbaugh`

这表示叫 *investor* 的表单元素是一个文本域，其值为 Jerry Bradenbaugh。

在第 110 行的 *for* 循环迭代 *prefsArray* 中的元素的过程中，通过按照每一个::符号的位置来分离 prefsArray[i]，而将局部变量 *currPref* 设置为一个数组。这意味着 *currPref* 会有三个元素（两个是 checkbox 元素）。由于 currPref[1] 包含表单元素类型标识符，检查它可以确定 getPrefs() 会对 curPref[0] 和 currPref[2] 做什么操作。

如果 currPref[1] 等于 *select*，getPrefs() 利用第 113 行的代码将 currPref[0] 中的名称选择列表赋值为和 currPref[2]（其实就是 parseInt(currPref[2])）中 selectedIndex 相关的选项，但是 JavaScript 会把字符串转变为一个数字。

如果 currPref[1] 等于 *text*，getPrefs() 利用第 116 行的代码将把 currPref[2] 中的值赋给有 currPref[0] 中名称的文本域。

最后，如果 currPref[1]等于 *checkbox*，getPrefs()利用第 119 行的代码将 currPref[0]中名称所指的 checkbox 的 checked 属性设置为 true。在这里用不到 currPref[2]。如果 checkbox 在 cookie 信息中出现，就是选中它的唯一标志。

对 *prefsArray* 中的每一个元素都有相同的处理。一旦完成以后，用户就有了一个表单，它包括最后设置的所有信息。因此 getPrefs()做完了它的工作，给 populateForm()返回一个 true 值，表示一切运行正常。

图片布局

现在要处理的只剩下背景图像和字体图片以及表单中所选的项目了。注意到 HTML 中两个图的 SRC 属性都被设置为 *images/blank.gif*。在图片按照 cookie 中的信息发生变化之前，这显然仅是一个占位符。函数 populateForm()在第 159～161 行中。我们来看一看：

```
makePath(formObj);
swapImage('bkgImage',
  formObj.background.options[formObj.background.selectedIndex].value);
```

把它看成一个函数管理员，populateForm()调用 makePath()和 swapImage()来完成图片翻转。实际上，swapImage()是作任何翻转的唯一函数；makePath()只是处理一对字符串，建立一个合适的路径送给 swapImage()。我们来查看一下它们中比较简单的一个，也就是对 swapImage()的调用。等你看到 swapImage()要求两个参数，它们在第 160～161 行传递以后，来看一看第 82～84 行的代码：

```
function swapImage(imageName, imageBase) {
     document[imageName].src = imageBase;
     }
```

参数 *imageName* 表示图片对象的名称，这个图片对象的来源将会被翻转过来。参数 *imageBase* 是我们要翻转的图片的 URL。看一下传递进来的参数：

```
formObj.background.options[formObj.background.selectedIndex].value
```

这可真是一个很大的参数，但它除了是被选中的 OPTION 标签的值外，什么也不是。由于 getPrefs()刚刚完成把所有的选择列表赋值为先前被用户选中项目的工作，图片当然是匹配的。参见表 7-2，你会得到一个更好的主意。它包含 OPTION 标签的值，OPTION 文字（用户所看见的文字），以及传递给 swapImage()的参数。

表 7-2 背景的 OPTION

OPTION 值	OPTION 文字	给 swapImage()的参数
images/goldthumb.gif	Gold Bars	*images/goldthumb.gif*
images/billsthumb.gif	Dollar Bills	*images/billsthumb.gif*
images/fistthumb.gif	Fist of Cash	*images/fistthumb.gif*
images/currency1thumb.gif	Currency 1	*images/currency1thumb.gif*
images/currency2thumb.gif	Currency 2	*images/currency2thumb.gif*

这看起来真是再简单不过了。swapImage()接收到当前选定的OPTION值。它已经是一个URL了，因此马上发生翻转。附带的，用户改变背景选择列表中的选项时，用的是同样的代码。也发生同样的翻转。这里是你载入页面时所产生的HTML，我们来看看：两个传递到swapImage()的参数和我们刚才在第161行所看到的有惊人的相同之处：

```
<SELECT NAME="background" onChange="swapImage('bkgImage',
this.options[this.selectedIndex].value);">
```

背景已经得到了设置。字体图片又怎么样呢？那就要麻烦一点啦。对于背景图片，有一个图片使用一个选择列表来翻转。对于字体图片，也仍然只有一个用来翻转的图片，但是这个翻转图片是基于两个选择列表中的被选中OPTION的。你可以自己看看图片的URL是怎样制造出来的。表7-3显示了OPTION值和两个选择列表的相应文字。

表 7-3 字库和尺寸的 OPTION

字体 OPTION 值	尺寸 OPTION 值	字体 OPTION 值	尺寸 OPTION 值
Timesroman	10	Times Roman	Small
Arial	12	Arial	Medium
courier	14	Courier	Large
Tahoma	16	Tahoma	X-Large

现在，在把从每一个中得到的OPTION值结合起来时，我们来作一下匹配。看看所有可能的字体外观和尺寸的组合。

timesroman10	*arial10*	*courier10*	*tahoma10*
timesroman12	*arial12*	*courier12*	*tahoma12*
timesroman14	*arial14*	*courier14*	*tahoma14*
timesroman16	*arial16*	*courier16*	*tahoma16*

现在，这难道不像第61行*imageNames*数组中用来预载入图片的元素吗？它们确实很像。毕竟，要产生的是一个很酷的应用程序。现在，我们知道必须制造一个合成的URL。那将是一个从好几个字符串中制造出来的URL。我们仍然需要调用swapImage()，但在这样做之前，我们先要用前面表格中的结合物制造一个URL。那也就是应该调用makePath()的地方。populateForm()在第159行对它作调用。这里是第75～80行的代码：

```
function makePath(formObj) {
     var fontName = imagePath +
       formObj.face.options[formObj.face.selectedIndex].value +
       formObj.size.options[formObj.size.selectedIndex].value + '.gif';
     swapImage("fontImage", fontName);
     }
```

MakePath()将表单对象的一个拷贝作为参数接受。我们所用的两个被选OPTION标签的值和add *.gif*都是从*fromObj*中来的。现在，局部变量*fontName*是指向一个合法图片的字符串。第79行对swapImage()的调用完成了这个任务。当然，所有这些都是在假设用户先前已经设置了参数选择的基础上。如果getPrefs()返回false，populateForm()在第163行调用函数resetImage()来翻转背景和字体图片，变成与背景和字体选择列表中的OPTION标签0相对应的图片。参见我们马上要讨论到的“重置表单”，可以获得更多的具体内容。

让我们花点时间回顾一下：

- 表单元素已经写到页面上了，其中的一部分是通过调用函数genSelect()和genBoxes()实现的。
- 已经调用getPrefs()来用收集到的cookie信息给每一个表单元素赋值。
- 背景和字体图片已经通过调用函数swapImage()和makePath()而和所选中的OPTION标签保持一致了。

修改

用户现在看到一个包含最后设置内容的用户参数选择页面。让我们来看看，当用户想要作修改的时候会发生什么情况。

对用户来说，进行修改和在文本域输入新的文字，从选择列表中选择其他选项，以及选中或不选中一个新的checkbox组合一样简单。然后他可以选择“Save”，就做完了修改。用户的事一做完，你的工作就开始了。我们看看第282行“Save”按钮的代码：

```
<INPUT TYPE=BUTTON VALUE="Save" onClick="setPrefs(this.form);">
```

看起来非常典型。它调用了函数setPrefs()并且将表单的复本作为参数传递。函数位于第125～151行：

```
function setPrefs(formObj) {
  var prefStr = '';
  var htmlStr = '';
  for (var i = 0; i < formObj.length; i++) {
    if (formObj[i].type == "select-one") {
      prefStr += formObj[i].name + '::select::' + formObj[i].
      selectedIndex + '-->';
      htmlStr += formObj[i].name + '=' + formObj[i].options[formObj[i].
      selectedIndex].value + '-->';
    }
    else if (formObj[i].type == "text") {
      if (formObj[i].value == '') { formObj[i].value = "Not Provided"; }
      prefStr += formObj[i].name + '::text::' +
      safeChars(formObj[i].value) + '-->';
      htmlStr += formObj[i].name + '=' + formObj[i].value + '-->';
    }
    else if (formObj[i].type == "checkbox" && formObj[i].checked) {
      prefStr += formObj[i].name + '::checkbox::' + '-->';
      htmlStr += formObj[i].name + '=' + formObj[i].value + '-->';
    }
  }
  SetCookie('userPrefs', prefStr, expiry);
  SetCookie('htmlPrefs', htmlStr, expiry);
    if (confirm('Preferences changed. Go to your personalized page?')) {
      self.location.href = "dive.html";
    }
}
```

setPrefs()产生两个字符串——一个被设置为局部变量*prefStr*，另一个设置为局部变量*htmlStr*。我们确实需要两个cookie——一个为这个页面生成参数选择表单，另一个用来产生*dive.html*和用户定制链接页面的设计和链接。所存储的信息几乎是

相同的，除了它们各自有各自表示数据的方式以外。我们一会儿再来看。下面是setPrefs()的基本方式，我们按顺序来看：

1. 迭代*formObj*，建立两个以FORM元素的值为基础的cookie字符串。
2. 将两个cookie写到用户cookie文件。
3. 给用户提供选项，以便马上可以转到*dive.html*去看看新的变化。

第1步：迭代formObj

这倒是不成问题。从本书的一开始，就有许多零零散散的迭代*formObj*之类的内容。在这里也是相同的情况，除了setPrefs()需要知道寻找的内容之外。

再来看看参数选择表单。注意到每一个元素（除了底部的按钮）都是文本域，选择表单或者checkbox。因此，setPrefs()只需要知道当formObj[i]是那些元素类型之一时自己要采取哪种类型的行动就行了。第129～144行的代码设定了具体方法。

```
if (formObj[i].type == "select-one") {
     prefStr += formObj[i].name + '::select::' +
      formObj[i].selectedIndex + '-->';
     htmlStr += formObj[i].name + '=' +
      formObj[i].options[formObj[i].selectedIndex].value + '-->';
     }
else if (formObj[i].type == "text") {
     if (formObj[i].value == '') { formObj[i].value = "Not Provided"; }
     prefStr += formObj[i].name + '::text::' +
      safeChars(formObj[i].value) + '-->';
     htmlStr += formObj[i].name + '=' + formObj[i].value + '-->';
     }
else if (formObj[i].type == "checkbox" && formObj[i].checked) {
     prefStr += formObj[i].name + '::checkbox::' + '-->';
     htmlStr += formObj[i].name + '=' + formObj[i].value + '-->';
     }
```

有一个很酷的可用表单元素属性是*type*属性，它包含一个用来确定表单元素种类的字符串。SetPrefs()只需要管单项选择列表、文本和复选框就行了。前面复合的*if*语句代码对每一个类型采取了不尽相同的动作。而setPrefs()不管类型是什么，都采取一种或其他方法做下面的工作：

- 连接表单元素名称，表单元素类型的等价字符串，可能还有表单元素值或者选中的索引，它们之间用定界符隔开。

- 将字符串和*prefStr*及*htmlStr*的现有值连接。

如果元素类型是一个选择列表，那么选中项的索引就被加到字符串上去。如果元素是一个checkbox，那么checkbox名称就被加到字符串上。如果元素是一个文本域，就将它的名称和值加到字符串上。注意，函数safeChars()对所有文本域的值进行操作。发生这种情况是因为选择列表和复选框的相应值是预先确定了的。由于用户可以输入任何东西，就很可能会输入一个保留的定界符字符串（在本例中是::,-->和=）。这将会在下一次应用程序试图获得和解析cookie信息的时候引起很多乱七八糟的结果。这里是第153～155行的函数：

```
function safeChars(str) {
    return str.replace(/::|=|-->/g, ':;');
    }
```

函数safeChars()只是从用户输入的任何东西中删除保留字符串，然后返回处理完后的字符串。每一个名称/类型或者值，或者当选项的索引串都用-->分隔。变量*prefStr*中字符串的每一个部分用::隔开。而*htmlStr*使用的是=。这两者都不一定是定界符，但它们都非常简单。下面这个例子是关于用相同的表单建立字符串时它们可能的形象：

*prefStr*看起来可能会是这样：

```
investor::text::Not Provided-->age::text::Not Provided-->strategy::
select::3-->occupation::text::Not Provided-->newsNames0::checkbox::-->
newsNames1::checkbox::-->newsNames2::checkbox::-->newsNames4::checkbox::
-->indexNames0::checkbox::-->indexNames2::checkbox::--
>background::select
::2-->face::select::3-->size::select::2-->
and htmlStr might look like this:
```

而*htmlStr*可能是这样的：

```
investor=Not Provided-->age=Not
Provided-->strategy=Moderate-->occupation
=Not Provided -->newsNames0=Barron's Online,http://www.barrons.com/-->
newsNames1=CNN Interactive,http://www.cnn.com/-->newsNames2=Fox News,
http://www.foxnews.com/-->newsNames4=The Wall Street Journal,
http://www.wsj.com/-->indexNames0=Dow Jones Indexes,
http://www.dowjones.com/-->indexNames2=The New York Stock Exchange,
http://www.nyse.com/-->background=images/fishthumb.gif-->face=tahoma-->
size=14-->
```

记住，-->在*prefStr*和*htmlStr*中将表单元素和其他东西分开，而::和=在两个变量中分别把单个的表单元素分隔。你知道，这里通过第286～289行代码来获取这个信息。那些按钮可用于使用“Show”来显示变量的值，用“Erase”来删除cookie信息。你的用户并不需要它们，但是它们却可以帮助你更方便地调试。

如果以上的代码看起来像一个丑陋的变量，先不要着急。等我们处理*dive.html*的代码的时候就会分解*htmlStr*。幸运的是，我们已经完成了用函数`getPrefs()`来“解码”*prefStr*的值。回顾一下这部分内容就更好了。比较一下`setPrefs()`怎样将cookie信息放到一起，`getPrefs()`又怎样分析那些cookie数据可以帮助你更好地理解这些工作总体上是怎样进行的。

第2步：将信息写到cookie文件

一旦*prefStr*和*htmlStr*随着用户参数选择信息一起被载入，第146～147行对`SetCookie()`的调用将把信息存储到一个cookie文件中。

Netscape Navigator将有关cookie信息保存到一个叫*cookies.txt*的文件中。下面是我的这个文件的一部分：

```
.hotwired.com    TRUE    /    FALSE    2145917529    p_uniqid
   2sfurM4NNMfDKAqQ8A
.hotbot.com    TRUE    /    FALSE    946739221    p_uniqid
   3MarneJsXGwNqxWbFA
www.allaire.com    FALSE    /    FALSE    2137622729    CFTOKEN    97611446
```

另一方面，IE 4.x/15.x将cookie信息保存为分离的文件。cookie文件的命名所依照的范围是cookie的来源和设置cookie时用户登录所用的名字。这里是我的WinNT上IEcookie文件的一部分内容。我是作为管理员登录的。

```
Cookie:administrator@altavista.com
Cookie:administrator@amazon.com
Cookie:administrator@builder.com
Cookie:administrator@cnn.com
Cookie:administrator@dejanews.com
Cookie:administrator@hotbot.com
Cookie:administrator@infoseek.com
```

第2步：给提供用户对新选择的浏览

最后一个要对setPrefs()做的工作是让用户进入*dive.html*页，看看修改对布局的影响。这里是第148～150行的代码：

```
if (confirm('Preferences changed. Go to your personalized page?')) {
  self.location.href = "dive.html";
  }
```

这包括了*prefs.html*的功能。不管怎样，总有一个项目在大多数情况下可以浏览——表单的重置。

重置表单

一个简单的<INPUT TYPE=RESET>按钮难道不能处理这件事吗？是的，一个reset按钮将文本域的值都设为空串，使所有的复选框都不被选中，而且使每一个选择列表的选中项都变为OPTION 0。这很不错，但是那些背景和字体图片却不能被删除。它们都应该翻转成和OPTION 0相对应的图片。这就是要写第283行代码的原因：

```
<INPUT TYPE=RESET VALUE="Clear" onClick="resetImage(this.form);">
```

不仅要选择"Clear"来重置表单，而且要调用函数resetImage()。这里是第166～170行的代码。

```
function resetImage(formObj) {
  swapImage('bkgImage', formObj.background.options[0].value);
  swapImage('fontImage', imagePath + formObj.face.options[0].value +
    formObj.size.options[0].value + '.gif');
  }
```

这是另一个调用函数的函数，它调用了swapImage()两次。第一次对swapImage()的调用将背景图片翻转来和OPTION 0对应，也就是formObj.background.options[0].value。第二次调用做相同的工作，但是创建和OPTION 0相对应的图片路径。和makePath()中的逻辑相似，resetImage()用imagePath来建立URL，两个与字体相关的选择列表的OPTION值（虽然这一次用0来代替selectedIndex），后面跟着字符串*.gif*。这两个调用在需要的地方设置了图片。

这使我们到了*prefs.html*功能的最后。让我们来认真看看*dive.html*。

dive.html

用户参数选择已经修改完了。现在应该来看看怎样把这些作了记录的修改转变成为直观的视觉效果。这个过程并不长，但是它的细节可不简单。到现在为止，已经非常明显可以获得存储在cookie中的信息了。得到的信息可以用在三个方面。

- 作为一个背景图片的URL
- 作为链接的URL和显示文字
- 作为字库和尺寸的样式表声明的一部分

我们顺着往下走的时候会碰到每一个应用。现在，我们来深入看一下例7-2给出的*dive.html*。

例7-2：dive.html

```
<HTML>
<HEAD>
<TITLE>
Your Take-A-Dive Links Page
</TITLE>
<SCRIPT LANGAUGE="JavaScript1.2" SRC="cookies.js"></SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.2">
<!--

var newsNames = new Array();
var indexNames = new Array();
function getAttributes() {
  var htmlStr = GetCookie('htmlPrefs');
  if (htmlStr == null) {
    alert('Welcome. You must first set your user preferences.' +
    'Please choose OK to load the User Settings page.');
    self.location.href = 'prefs.html';
  }
  var htmlArray = htmlStr.split('-->');
  for (var i = 0; i < htmlArray.length; i++) {
    var tagInfo = htmlArray[i].split('=');
    if (tagInfo[0] != "") {
      if (tagInfo[0].indexOf('newsNames') == 0) {
        newsNames[newsNames.length] = tagInfo[1];
        }
      else if (tagInfo[0].indexOf('indexNames') == 0) {
        indexNames[indexNames.length] = tagInfo[1];
        }
      else { eval(tagInfo[0] + ' = "' + tagInfo[1] + '"'); }
      }
    }
```

```
33    }
34
35 getAttributes();
37 function genLinks(linkArr) {
38    var linkStr = '';
39    for (var i = 0; i < linkArr.length; i++) {
40      var linkParts = linkArr[i].split(',')
41      linkStr += '    - <A HREF="' + linkParts[1] + '"> ' +
42        linkParts[0] + '</A><BR>'
43      }
44    return linkStr;
46
45    }
47 //-->
48 </SCRIPT>
49 <SCRIPT LANGUAGE="JavaScript1.2">
50    document.write('<STYLE type="text/css"> TD ' +
51      { font-family: ' + face + '; font-size: ' + size + 'pt; } </STYLE>');
52 </SCRIPT>
53 </HEAD>
54 <SCRIPT LANGUAGE="JavaScript">
55    document.write('<BODY BACKGROUND="' +
56      background.replace(/thumb/, "") + '">');
57 </SCRIPT>
58 <TABLE BORDER=0>
59    <TR>
60      <TD VALIGN=TOP COLSPAN=4>
61      <H2>Take-A-Dive Brokerage $ervices</H2>
62      </TD>
63    </TR>
64    <TR>
65      <TD VALIGN=TOP COLSPAN=4>
66      Take-A-Dive Brokerage Services is dedicated to helping YOU part with
67      your MONEY. <BR> Our motto is "<I>We go for broke.</I>"
68      Here is your user profile
69      and your customized links.
70      <BR><BR>
71      </TD>
72    </TR>
73    <TR>
74      <TD VALIGN=TOP>
75      Name:</TD>
76      <TD VALIGN=TOP>
77        <SCRIPT LANGUAGE="JavaScript1.2">document.write(investor);</SCRIPT>
78            <TD VALIGN=TOP>
79            </TD>
80            Age:
81            <TD VALIGN=TOP>
82            <SCRIPT LANGUAGE="JavaScript1.2">document.write(age);</SCRIPT>
83            </TD>
84       </TR>
85       <TR>
86            <TD VALIGN=TOP>
```

```
87            Strategy:
88            </TD>
89            <TD VALIGN=TOP>
90            <SCRIPT LANGUAGE="JavaScript1.2">
91              document.write(strategy);
92            </SCRIPT>
93            </TD>
94            <TD VALIGN=TOP>
95            Occupation:
96            </TD>
97            <TD VALIGN=TOP>
98            <SCRIPT LANGUAGE="JavaScript1.2">
99              document.write(occupation);
100           </SCRIPT>
101           </TD>
102       </TR>
103       <TR>
104          <TD VALIGN=TOP COLSPAN=2>
105          <BR><BR>
106          News Links<BR>
107          <SCRIPT LANGUAGE="JavaScript1.2">
108            document.writeln(genLinks(news));
109          </SCRIPT>
110          <TD VALIGN=TOP COLSPAN=2>
111          <BR><BR>
112          Stock Index Links <BR>
113          <SCRIPT LANGUAGE="JavaScript1.2">
114            document.writeln(genLinks(indexes));
115          </SCRIPT>
116          </TD>
117       </TR>
118       <TR>
119          <TD VALIGN=TOP COLSPAN=2>
120          <BR><BR>
121          [ <A HREF="prefs.html">Set Preferences</A> ]
122          </TD>
123       </TR>
124 </TABLE>
125 </BODY>
126 </HTML>
```

解析 cookie

看看所有那些SCRIPT标签。光凭看看你就可以发现这个页面实际上是一个格式，将会在运行中按计划定制。第一步是要解析cookie。函数getAttributes()将得到调用。为了有效地创建一个设计方案，我们很快就需要这些信息(在载入页面之前)。这就是为什么在定义getAttributes()后一两行就调用它的原因。这里是第13～33行的代码：

```
function getAttributes() {
  var htmlStr = GetCookie('htmlPrefs');
  if (htmlStr == null) {
    alert('Welcome. You must first set your user preferences.' +
    'Please choose OK to load the User Settings page.');
    self.location.href = 'prefs.html';
  }
  var htmlArray = htmlStr.split('-->');
  for (var i = 0; i < htmlArray.length; i++) {
    var tagInfo = htmlArray[i].split('=');
    if (tagInfo[0] != "") {
      if (tagInfo[0].indexOf('newsNames') == 0) {
        newsNames[newsNames.length] = tagInfo[1];
      }
      else if (tagInfo[0].indexOf('indexNames') == 0) {
        indexNames[indexNames.length] = tagInfo[1];
      }
    else { eval(tagInfo[0] + ' = "' + tagInfo[1] + '"'); }
    }
  }
}
```

局部变量*htmlStr*被设置为函数`GetCookie()`的返回值。在*prefs.html*中，名为*prefStr*的cookie有必要的表单信息，但是*dive.html*需要的是名为*htmlStr*的cookie。如果发现*htmlStr*等于null，就意味着用户先前并没有设置过参数选择，因此应该给他作出提示，然后快速地转到*prefs.html*来做一些修改。

否则，*htmlStr*将会被`split()`通过每一个`-->`分隔开，返回一个设为局部变量*htmlArray*的数组。一个*for*循环用来迭代每一个元素，紧接着给每个元素赋值。顺便说一句，这和*prefs.html*中`getPrefs()`函数的逻辑是相同的。看看第110～120行*prefs.html*列表中的代码。在第20～32行上对它们做一下比较。

```
var htmlArray = htmlStr.split('-->');
for (var i = 0; i < htmlArray.length; i++) {
  var tagInfo = htmlArray[i].split('=');
  if (tagInfo[0] != "") {
    if (tagInfo[0].indexOf('news') == 0) {
      newsNames[newsNames.length] = tagInfo[1];
    }
    else if (tagInfo[0].indexOf('indexes') == 0) {
      indexes[indexNames.length] = tagInfo[1];
    }
  else { eval(tagInfo[0] + ' = "' + tagInfo[1] + '"'); }
  }
}
```

处理未知事物

非常有趣。*Prefs.html* 中的 getPrefs() 知道自己将要处理 *formObj* 所指出的表单，并且因此而为值，控制和选项赋值。但是 getAttributes() 没有这么繁杂。至少必须有一些关于要从 cookie 中获得什么东西的概念。例如，我们知道会有一些关于在线新闻资源的链接信息。在线股票指数也是如此。谁知道每一个会有多少呢：0，10，也许是50？因为这是未知的，我们可以将两者的链接信息放到单独的数组中。这里是第 10～11 行代码：

```
var newsNames = new Array();
var indexNames = new Array();
```

变量 *newsNames* 将保存新闻资源的链接信息，而 *indexNames* 保存股票指数的链接信息。看一看你在最后部分所看到的简单的 cookie。注意粗体字部分：

```
investor=Not Provided-->age=Not Provided-->strategy=Moderate-->
occupation=Not Provided-->newsNames0=Barron's Online,
http://www.barrons.com/-->newsNames1=CNN Interactive,
http://www.cnn.com/-->newsNames2=Fox News,
http://www.foxnews.com/-->newsNames4=The Wall Street Journal,
http://www.wsj.com/-->indexNames0=Dow Jones Indexes,
http://www.dowjones.com/-->indexNames2=The New York Stock Exchange,
http://www.nyse.com/-->background=images/fistthumb.gif-->
face=tahoma-->size=14-->
```

这些粗体名字是前边所定义的新数组变量链接信息标头的标记。现在，我们还知道将会有一些变量，但是我们怎么给它们命名呢？它们会有多少？*dive.html* 中的代码根本没有给出任何答案。但事实是，这并没有关系。只要你知道变量名是什么，*dive.html* 中的逻辑就不必包括外在的，编写麻烦的定义。这就是我的意思。再看看 GetCookies('htmlPrefs') 的简单的值。同样要注意一下粗体字：

```
investor=Not Provided-->age=Not Provided-->strategy=Moderate-->
occupation=Not Provided-->newsNames0=Barron's Online,
http://www.barrons.com/-->newsNames1=CNN Interactive,
http://www.cnn.com/-->newsNames2=Fox News,http://www.foxnews.com/-->
newsNames4=The Wall Street Journal,http://www.wsj.com/-->
indexNames0=Dow Jones Indexes,http://www.dowjones.com/-->
indexNames2=The New York Stock Exchange,http://www.nyse.com/-->
background=images/fistthumb.gif-->face=tahoma-->size=14-->
```

这个例子中的粗体字表示马上就要定义的变量的名称。GetAttributes() 中的 *for* 循环给数组元素提供分配的值，并“未知”变量的定义。它在第 22～31 行中：

```
var tagInfo = htmlArray[i].split('=');
  if (tagInfo[0] != "") {
    if (tagInfo[0].indexOf('newsNames') == 0) {
      newsNames[newsNames.length] = tagInfo[1];
    }
    else if (tagInfo[0].indexOf('indexNames') == 0) {
      indexNames[indexNames.length] = tagInfo[1];
    }
  else { eval(tagInfo[0] + ' = "' + tagInfo[1] + '"'); }
}
```

*HtmlArray*中的每一个元素包含一个“=”符号，它把标识符和我们真正想要的值分开。在*for*循环的每一次循环中，htmlArray[i]被“=”符号分开，这个两元素的子数组被设为局部变量*tagInfo*。如果tagInfo[0]不等于一个空串，我们就有了一个有效的标识符值对。执行空串的检查是由JavaScript从split()函数返回数组的方法决定的。

每一个有效的标识符值对都属于两个类别中的某一个：一个数组元素或者一个正规变量。如果值对是一个数组变量，它又是两种类型之一，一种包含新闻链接，另一种包含股票指数链接。下面的if-else语句确定在所有情况下所采取的动作：

```
if (tagInfo[0].indexOf('newsNames') == 0) {
  newsNames[newsNames.length] = tagInfo[1];
}
else if (tagInfo[0].indexOf('indexNames') == 0) {
  indexNames[indexNames.length] = tagInfo[1];
}
```

因由于*prefs.html*中所使用的命名协议的关系，如果tagInfo[0]包含字符串*newNames*，它必须和新闻链接相关。因此，tagInfo[1]的值被赋给newsNames中下一个可用元素。如果tagInfo[0]包含字符串*indexNames*，它必须和股票指数相关。因此，tagInfo[1]就到indexNames中的下一个元素中去。如果tagInfo[0]什么串也不包含，那么tagInfo[0]和tagInfo[1]必须包含一个要定义的变量的名称和一个要赋给它的值。第30行的代码有关于这一点的处理：

```
eval(tagInfo[0] + ' = "' + tagInfo[1] + '"');
```

当这个循环完成时，会生成如下所示的等价代码：

```
newsNames[0] = 'Barron's Online,http://www.barrons.com/';
newsNames[1] = 'CNN Interactive,http://www.cnn.com/';
newsNames[2] = 'Fox News,http://www.foxnews.com/';
newsNames[3] = 'The Wall Street Journal,http://www.wsj.com/';
```

JavaScript 技巧：再度使用命名协定

我在前面已经吹捧了半天命名协定的作用。我不想再来布道，但只要考虑一下这些新闻和股票指数链接元素和设计变量是怎样形成的。它在编译任何 *dive.html* 的代码之前发生。它还在 *prefs.html* 中设置任何 cookie 之前。甚至，它在用户在 *prefs.html* 的表单中作第一次修改之前发生。它是从表单域的命名开始的。

每一个新闻和股票指数链接变量都有一个标识符（例如 *newsNames()* 或者 *indexNames3*），标识符包含 *prefs.html* 中一个选择列表的名称。每一个设计变量保存一个表单元素的名称，比如背景或者尺寸。这个名称包括在 cookie 字符串中，以便获取和定义。计划周密的命名协议不仅使事情变得更加简单，还增加了某些情况的可能性。要看好时机在你的代码中使用它们。

记住，文件中没有所列出的任何代码。你可以迭代 *newsNames* 和 *indexNames* 的元素来访问它们的值，但是你必须知道其他变量名称以访问它们。

这些是新闻链接：

```
indexNames[0] = 'Dow Jones Indexes,http://www.dowjones.com/';
indexNames[1] = 'The New York Stock Exchange,http://www.nyse.com/';
```

这些是股票指数链接：

```
var investor = 'Not Provided';
var age = 'Not Provided';
var strategy = 'Moderate';
var occupation = 'Not Provided';
var background = 'images/fistthumb.gif';
var face = 'tahoma';
var size = '14';
```

而这些都是设计变量。

我们有了所需的所有信息来按照用户要求建立页面。一旦我们将信息写到页面上，它也就完成了。我们要使用 document.write() 来在页面上添加我们所需要的一切东西。一共有八个对 document.write() 的调用。表 7-4 给出了每一个调用及其描述。

表7-4 使用document.write()来创建HTML

行	代码	描述
50～51	document.write('<STYLE type="text/css"> TD { font-family: ' + face + '; font-size: ' + size + 'pt; } </STYLE>');	在运行中创建一个样式表
55～56	document.write('<BODY BACKGROUND="' + background.replace(/thumb/, "") + '">');	为背景图片定义URL
77	document.write(investor);	添加访问者名字
82	document.write(age);	添加访问者年龄
91	document.write(strategy);	添加访问者策略
99	document.write(occupation);	添加访问者职业
108	document.write(genLinks(news));	添加新闻链接
114	document.write(genLinks(indexes));	添加股票指数

虽然从第三个到第六个调用都非常简单易懂，但前两个和后两个调用就有点儿麻烦了。让我们从第50～51行的代码开始：

```
document.write('<STYLE type="text/css"> TD { font-family: ' + face +
  '; font-size: ' + size + 'pt; } </STYLE>');
```

这个调用将一个样式表写到页面上，但是我们将要插入设计变量face和size来规定字体类型和尺寸。

随着所创建的样式表的到位，我们转移到关于引入正确的背景图片的讨论。这里是第55～56行的代码：

```
document.write('<BODY BACKGROUND="' +
  background.replace(/thumb/, "") + '">');
```

记得变量*background*包含有值*images/fistthumb.gif*。这太好了，只可惜这只是有关背景图片的一点点文字而已。我们需要真实的东西。没有问题，每一个小图片都确切地作为成熟版本来命名，除了添加*thumb*字符串以外。我们可以通过从*background*（在本例中是*images/fist.gif*）中删除“thumb”来得到背景图片。replace()函数可以快速地完成修改。

JavaScript 技巧: DHTML

DHTML，我是指JavaScript在运行中写DHTML。考虑你可以使用document.write()来显示HTML和更多的JavaScript。如果你将这个功能和样式表的优点结合起来的话，你就可以在很少的代码中拥有许多设计功能。这里是它在应用程序中的用法。

```
document.write('<STYLE TYPE="text/css"> TD { font-family: ' + face +
  '; font-size: ' + size + 'pt; } </STYLE>');
```

在这里，我所完成的所有工作是变量*face*和*size*。现在，字体的类型和尺寸可以通过修改两个变量的值来确定。只有一行的样式表可也不坏。想一下这个可能性，如果你有一个很长的确定的样式表来控制更多元素的样式，比如标题，表单元素及其他更多东西。样式表使你能高质量地控制文档元素。用JavaScript产生样式表可以实现同样的动态控制。

最后两个对document.write()的调用只用了一个其他的函数，它是在页面genLinks()中定义的。这个函数同*prefs.html*中的genBoxes()和genSelect()很相像，循环迭代一个数组中的元素，创建一个定制的HTML串。唯一的区别在于，这个函数返回的是一个链接字符串，而不是checkbox或者OPTION标签。这个“戏法”发生在第37～45行：

```
function genLinks(linkArr) {
  var linkStr = '';
  for (var i = 0; i < linkArr.length; i++) {
    var linkParts = linkArr[i].split(',')
    linkStr += '    - <A HREF="' + linkParts[1] + '"> ' +
      linkParts[0] + '</A><BR>'
    }
  return linkStr;
  }
```

genLinks()准备接收一个经过划界的字符串数组来作为它唯一的一个参数。每一个数组的第一个元素是一个用来作为链接文字显示的字符串。第二个元素是HREF属性的URL。其中的每一个都用逗号隔开，因此按照逗号使用split()函数，然后将数组结果赋值给局部变量*linkParts*，我们就能获得各个分开的部分。*for*循环和通常一样进行迭代，创建一个链接字符串，等结束时返回。

应用程序扩展

甚至一点小小的创造都能改变你的这个应用程序。这里是一些能想到的可能性：

- 通过添加域，来操作表格单元背景和字体颜色，以及其他设计外观的元素。
- 允许用户选择整个页面主题。
- 添加一对较大文本域以使用户能够添加他自己常用的站点链接（使用名称）。
- 按照用户的cookie参数选择粘贴广告条幅。

对设计外观的更多选择

用户喜欢有很多的选择。你能配给用户页面的所有东西都应该可以被操作。这包括层中的，其他框架中的，以及外部窗口中的内容和图片。

添加主题

这个主意滋生于Win95的桌面主题。以其让用户分别选取项目，诸如字体外观、尺寸和颜色，为何不提供一对设计主题使用户可以一次做完选择呢？假设你有一个有关音乐的网站。考虑下面的选择列表：

```
<SELECT NAME="themes"onChange="swapImage('theImage',
  this.options[this.selectedIndex].value);">
<OPTION VALUE="none">None
<OPTION VALUE="bigband">Big Band
<OPTION VALUE="rocknroll">Rock 'n Roll
<OPTION VALUE="rap">Rap
<OPTION VALUE="country">Country
<OPTION VALUE="reggae">Reggae
<OPTION VALUE="grunge">Grunge
<OPTION VALUE="jazz">Jazz
<OPTION VALUE="club">Club Music
</SELECT>
```

每一个OPTION值可以和扩展主题的一个小图片关联上。你甚至可以使用*prefs.html*中的genSelect()和swapImage()来创建列表，执行翻转。但是要记住，通过选择一个主题，你会莫名其妙地丧失对单个设计特征的控制能力，比如背景图片和字体格式。注意，第一个OPTION显示“None”。你可能会想要在这里包括一个OPTION标签，就可以使用户能按照自己的愿望作单个设计特征的选择。

让用户创建他们自己的链接

*Take-A-Dive*参数选择表单让用户从预先确定的链接中作出选择。你也可以添加一对文本域给用户，使他们能够输入自己喜欢的链接。下面的表将会给你一个美好的开始：

```
<TABLE>
  <TR>
    <TD><B>Extra Links</B></TD>
    <TD>
      <INPUT TYPE=BUTTON VALUE="  Add  " onClick="addOpt(this.form);">
      <INPUT TYPE=BUTTON VALUE="Delete" onClick="deleteOpt(this.form);">
    </TD>
  </TR>
  <TR>
    <TD>Link Name</TD>
    <TD><INPUT TYPE=TEXT NAME="linkname" SIZE=20></TD>
  </TR>
  <TR>
    <TD>Link URL</TD>
    <TD><INPUT TYPE=TEXT NAME="linkURL" SIZE=20></TD>
  </TR>
</TABLE>
```

用户可以通过输入链接名称和URL，然后选择“Add”或“Delete”来添加或者删除链接。然后你可以在一个数组里存储这些变量，用诸如在前边代码中提到的addOpt()和delOpt()之类函数来添加或者删除元素。如果你有足够的精力和兴趣，你可以创建一个链接项目的选择列表，以在添加或删除链接时使用。

指挥广告标幅营销

为什么不让你的广告活动迎合用户的兴趣呢？在这个虚拟投资的网站示例中，我编写了有关的代码，因此，那些自认为是保守投资者的用户将会收到提供稳定然而效率低的投资的广告标幅，比如债券。从另一方面说，那些胆大包天的访问者将收到那些提供不稳定但利润高的债券以及海外投资方面的广告标幅。

第八章

购物袋：JavaScript 购物车

应用程序要点

- 通用的客户端购物车
- 友好的用户界面使购物轻松愉快
- 不要求服务器端处理（直到结帐）
- 程序监控所有选择并保存不断变化的总数
- 数据库支持精确的产品搜索

JavaScript 技巧

- 管理多种窗口和文档内容
- 用对象保存客户状态
- 添加对象属性
- 重新使用JavaScript数据库
- 数字舍入和字符串转换

如果这本书里有一个最为丰富多变的应用程序的话，那就是我们将要说到的这个。利用购物袋，你只需要图形和产品细节就可以快速而且简单地在站点上添加一个在线购物车。你不必创建巨大的文件来显示产品。购物袋会在运行中管理这个问题。同样不需要一个服务器来计算税金和总额。购物袋也会在运行中精确地计算它们。在用户的购物袋中添加和删除产品只需要作一两下点击就行了。它不用让用户等待，这和基于服务器的购物袋大不相同。

购物袋概述

一些东西对于在线消费者来说非常简单和直观，但对程序员就通常意味着很麻烦的工作。你可别失望，这就是这里的例情况。但是，你在成果中得到的乐趣和强大的应用功能会使你的努力没有白费。购物袋功能性的描述和代码以下面的例子为基础。

这里是四个步骤的过程：

1. 载入应用程序。
2. 购物者通过类别和产品搜索来浏览产品，然后作出选择。
3. 购物者浏览当前所选项，在数量和内容上作修改，以满足要求。
4. 最终满意之后，用户决定结帐付帐。

这个应用程序也有几个简单的规则，用户在使用购物袋的时候必须遵循。你可以在每一个部分中适当的地方看到这些规则。让我们来帮助一个用户（就把她叫做Daisy Deep Pocket好了）把她辛辛苦苦赚来的钱花掉一点。

第 1 步:载入应用程序

打开 *ch08\index.html* 将得到如图 8-1 所示的屏幕。开始界面就是这样，是一种飞溅屏幕（没有斑点）。当 Daisy 点击 "begin" 链接，她就会看见如图 8-2 所示的屏幕。

图 8-1 友好的购物袋欢迎界面

在这里，你将看到的是打开的（还有帮助）载入屏幕，它底部的一个框架中有导航链接。在离开之前先认真地看一看页面的内容。

为什么会有两个浏览窗口呢？你反正不能仅仅靠一个主窗口来工作，但是这个方法给了你在购物时所需固定页面的最大数目。此外，用户将不会被诸如 "Bookmarks（书签）" 和 "Search（搜索）" 之类的浏览器按钮弄得头晕脑胀。这意味你可能会将更多的注意力放在产品上。使用的时候，你可以将主窗口作为一个登陆页面来分辨成员和客人购物者。

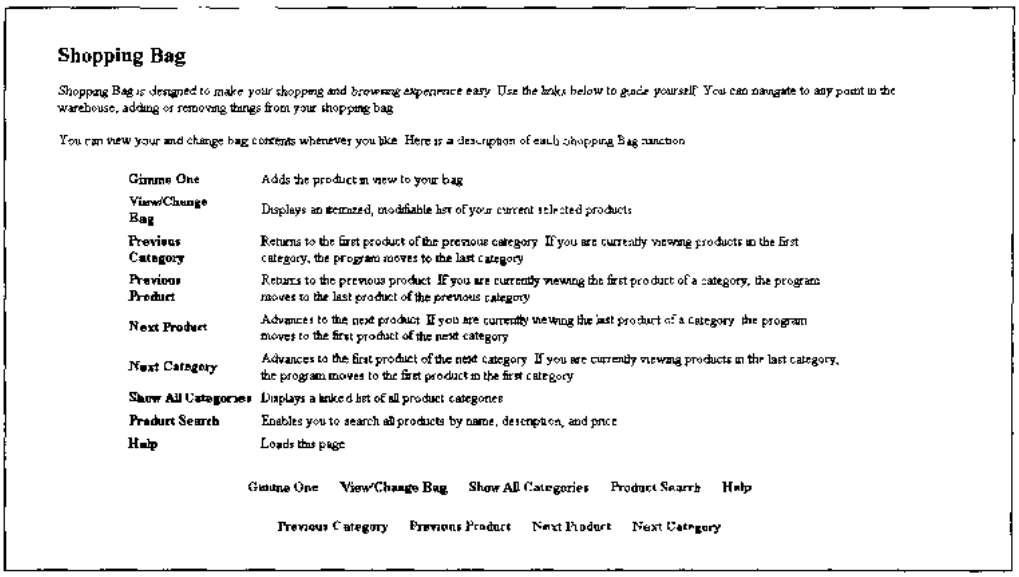


图 8-2　购物袋界面

第 2 步：产品的浏览和选择

好的，Daisy 现在进来了。首先应该到处看看。她选择“Show All Categories（显示所有种类）”来产生一个产品种类的链接列表。

注意：规则 1：在购物袋最初载入的时候，用户必须选择“Show All Categories”或者“Product Search（产品搜索）”，并且从得到的页面中选择一个链接来浏览产品。然后，在导航条中选择“Category（种类）”或者“Product（产品）”按钮就可以得到需要的结果。

它显示一个琳琅满目的产品种类的数组。我们来看看，有建筑，食品，硬件——谁不愿意在网上购买这些东西呢？等到从这些让人眼花缭乱的选择中恢复过来，Ms. Pockets 决定不要搞太多的建筑模型收藏，并决定看看购物袋会提供些什么东西。

跟随“Buildings（建筑）”链接，她再一次被诸如谷仓、城堡和塔等项目的热卖价格给吓坏了。她觉得很沮丧，只对一个圆顶建筑感到满意，如图 8-3 所示。选择“Gimme One”，圆顶建筑就被加到了她的购物袋中，如警告对话框所示。

图 8-3 将圆顶建筑放到她的袋子里

注意：规则 2：选择“Gimme One”只添加一个产品到购物袋中。用户可以通过选择“View/Change Bag（查看/改变购物袋）”来修改数量或删除所选的商品。

仍然想找便宜货，我们这个无情的购物者决定使用搜索功能来看看她能不能到一个更热门的项目中去。她选择了“Product Search”，然后进入如图 8-4 所示的简单的搜索界面。为了寻找便宜一点的商品，她进入“1.15”来查找这个价格的产品。真是幸运，她的目的达到了。有五个价位在 1.15 上的产品，如图 8-5 所示。吹风机、领带，还有几种价钱都是 1.15 美元的油炸食品。她要了油炸食品，随便看了看具体情况，然后很快地把它们加到自己的袋子里。

图 8-4 从这里开始搜索产品

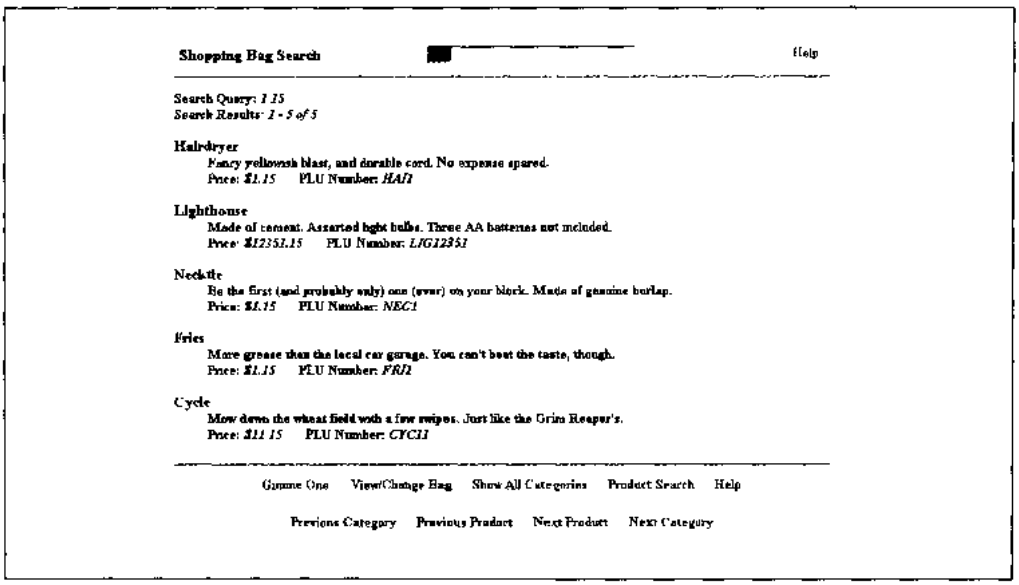


图 8-5 一个搜索，得到多种便宜产品

Daisy 又一次选择“Show All Categories”回到种类列表。这一次，服装种类激发起了她的兴趣。跟随这个链接，她看到了领带 —— 列表中一个价位在 $1.15 上的商品，真让人难以置信。她早先忽视了它，但现在看来还真是不错哦。把它加到袋子里好了。

第 3 步：浏览和修改定单

Daisy 想，她今天买的东西已经够多了，就选择了“View/Change Bag”。购物袋应用程序产生了如图 8-6 所示的她的购物袋中的内容。注意，购物袋程序为每一个她所加到袋子里的东西都保存制表符，接着是税金、运费以及总计。

仍然还是很喜欢圆顶建筑，Daisy就把她的圆顶建筑的定单数量修改到6个。她生活在一个温暖的地带，因此，这些用雪做的东西是放不久的。最好起码要很多个。那些油炸食品在图中看来非常不错，因此她把数量增加到 2。不幸的是，她口袋里的钱没有自己想像的那么多，所以她只好忘记领带了。反正下一次也还有啦。

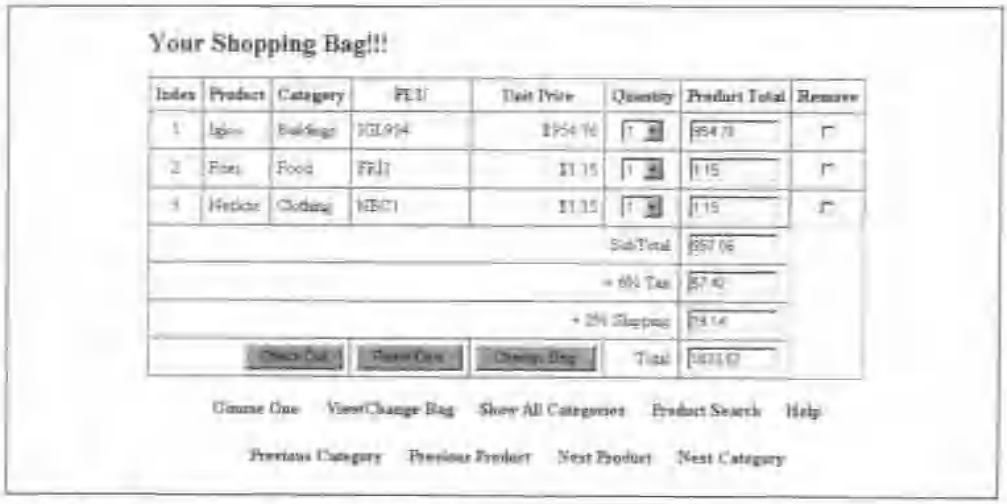


图 8-6 Daisy 的购物袋内容及总价格

注意：规则 3：购物者必须选择“Change Bag（修改购物袋）”来记录他们对购物袋所做的修改。换句话说，仅仅通过选择任何“Quantity（数量）”选项列表或者选中“Remove（删除）”复选框，然后就转换到其他页面的话，修改是不起作用的。

看看 Daisy 怎么才能修改她的定单，其实是很简单的。她利用“Quantity”选择列表来修改数量，用“Remove”复选框来删除一些商品。每一个产品条目都有一个选择列表以供修改数量，还用一个复选框来从定单中删除产品。修改定单之后，Daisy 选择“Change Bag”，就显示她的新定单，反映出她修改数量和删除产品的所有情况。参见图 8-7。现在，她的袋子中有 6 个圆顶小屋，两个油炸食品定购，还有一个领带。再加上税金和运费，使总额达到了 6190.57 美元。

注意：规则 4：提交购物袋定购表单将会清空购物袋。如果你现在还想要更多的东西，你只能从头开始填充一个新的空袋子了。

第 4 步：结帐

Daisy 对结束后的结果感到很满意，她选择“Check Out（结帐）”，打开如图 8-8 所示的表单。Daisy 可以填写她的订购信息，提交，然后等待她的邮件出货通知。

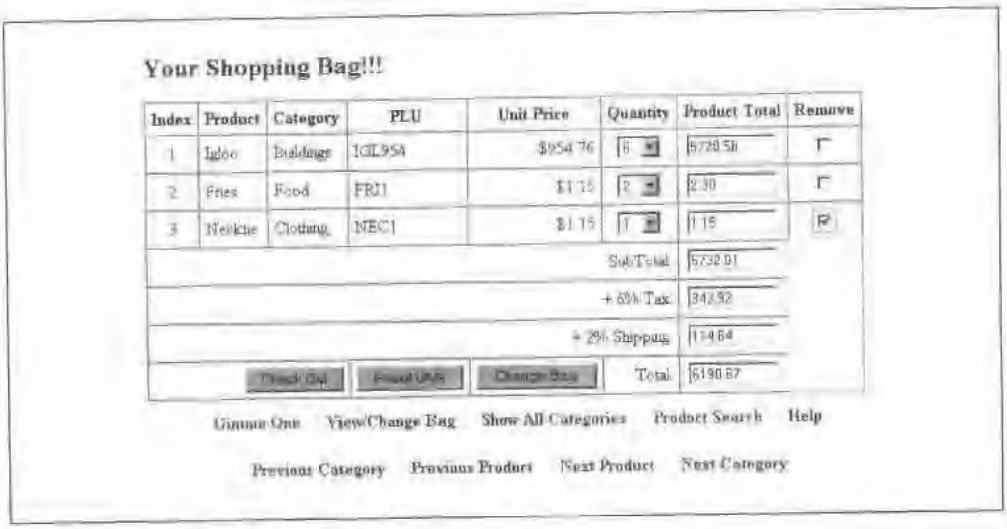


图 8-7 Daisy 精确的购物袋

过程如此重复进行……又是一个满意的消费者。我们到下一节去看看，购物袋程序是怎样按照这个路线来满足消费者的。

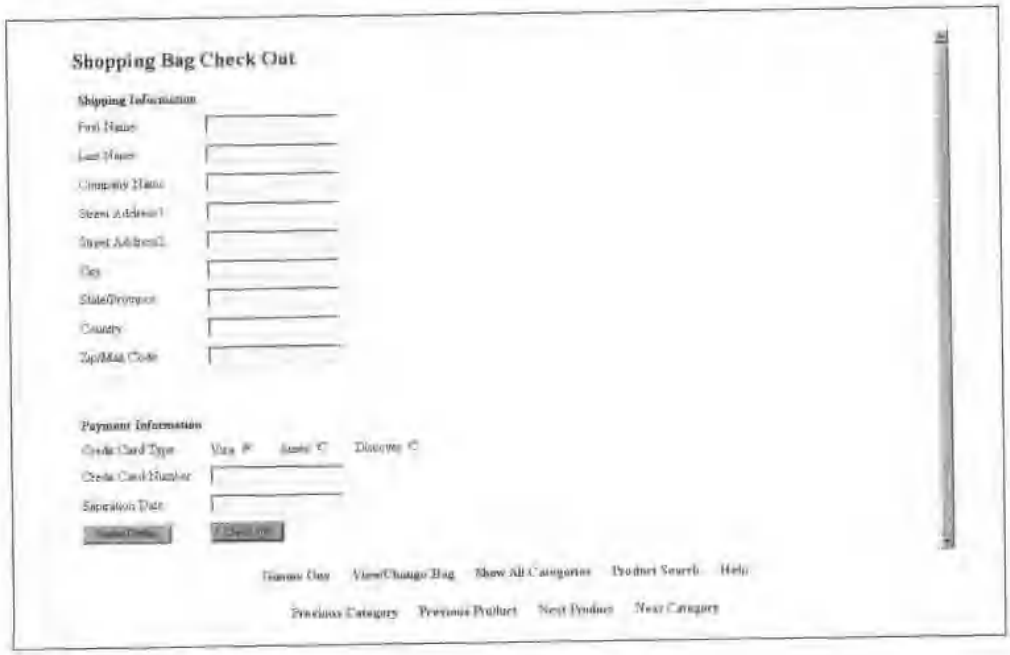


图 8-8 订购表单

执行条件

购物袋应用程序利用JavaScript 1.2和一些CSS，因此3.x代的浏览器版本不适用。但是，要记住购物用户的重要数量，而不要总是使用最新的浏览器。你可以很容易地删除CSS来使购物袋应用程序和IE及Navigator 3.x兼容。

在容纳范围之内，你应该可以适当地在购物袋仓库中填满至少500个项目。毕竟，添加一种产品只花一行代码而已。在我的128M RAM的120MHz的PC上，我到700个项目时就停止了实验。因此，除非你要同沃尔马特竞争，否则的话购物袋都可以良好地运行。

语法细则

让我们来看看购物袋应用程序的流程图。图8-9显示用户怎样开始购物，浏览和选择产品，在结帐之前做完最后的修改，以及怎样填好付款和购物信息，最后提交整个定单给服务器。

这个应用程序由八个文件构成。下面的列表描述了每一个文件以及它们的用途：

index.html

这个购物袋应用的打开页面负责窗口管理。

shopset.html

这是全屏窗口的框架。包含*intro.html*和*manager.html*。

intro.html

最大一个框架的默认页面，是列出每一个导航条特征函数的帮助文档。

manager.html

这是购物袋应用程序的核心。所有的核心功能都来源与这个文件中的代码，这将是本章的焦点所在。

inventory.js

包含函数，构造器，以及用来建立购物袋清单的数组。它的大部分工作在载入时进行。

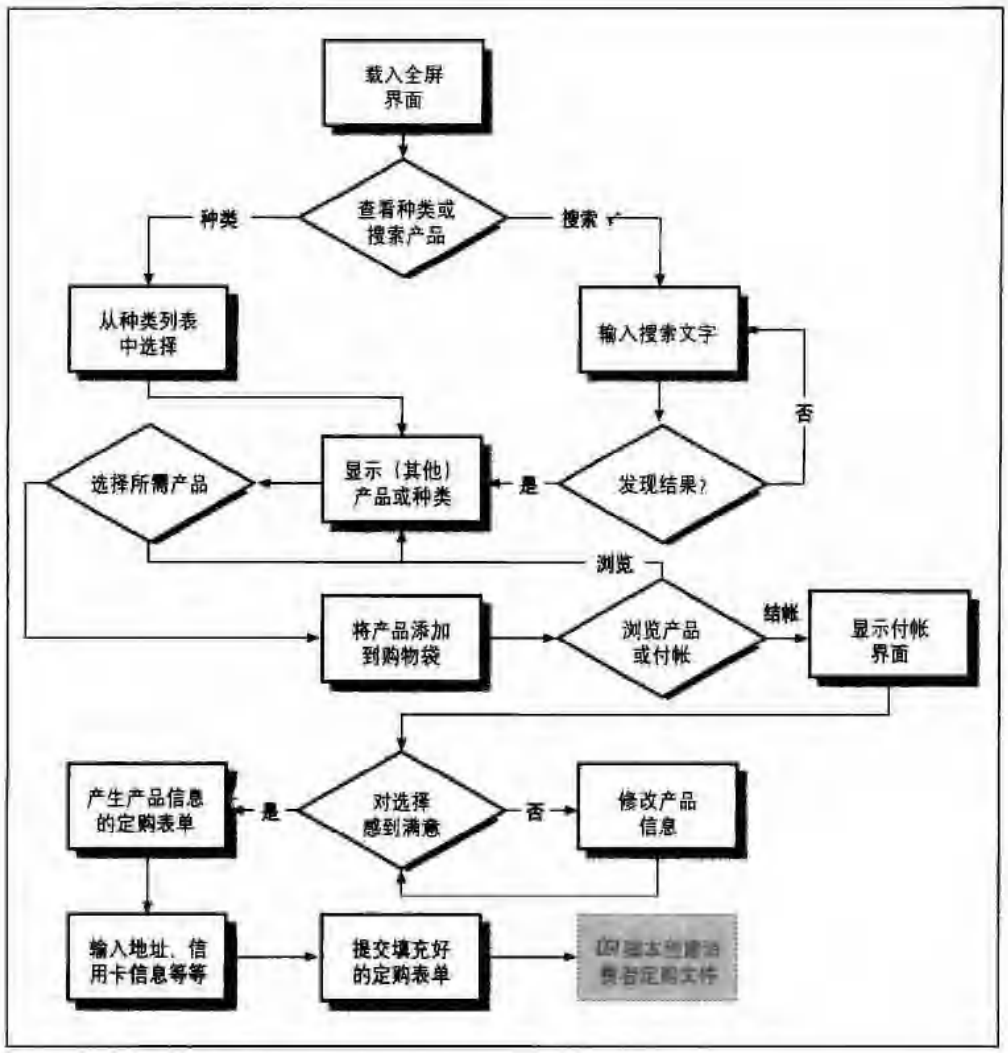


图 8-9　购物袋概况

search/index.html

载入搜索应用程序的框架设置。如果你看过第一章，你会把它认为是那里所讨论的搜索引擎的一个修改版本。

search/main.html

搜索引擎的帮助页面。还包括许多例子，非常完善。

search/nav.html

搜索引擎应用程序的核心

由于应用程序非常长，而且你能在其他章节中（第三、四、六、九、十和十一章）看到用来管理层的一些JavaScript代码的原因，我会把重点更多地放在购物袋的特别之处。

为了避免从头到尾一一讲解这些文件，我们按照它的五个基本操作项目来囊括这个应用程序：

1. 载入购物袋：建立清单并载入显示
2. 显示产品：从一个种类到另一个种类，从一个产品到另一个产品
3. 将产品添加到购物袋：保存每样东西在袋子中的记录
4. 搜索产品：运行有库存产品的文字搜索
5. 修改购物袋内容/结帐：作修改和付帐

如果你把这里的文件描述同前面的作比较，你可以不完全地推断（也就是猜）出哪一个文件伴随哪一个操作。它没有被切断也没有中途结束，但是操作1包括*index.html*，*shopset.html*，*inventory.js*的代码；操作2、3和5又包含*manager.html*中的代码；而操作4中的搜索文件目录是唯一的。

你可能还是想以文件到文件的顺序来看代码，但那样的话，我们会为了囊括JavaScript操作函数而弄得很麻烦。这五个部分的每一个都几乎是按照用户动作来描述的，比如搜索，修改产品数量，获得帮助，等等。我们也按照这个顺序来讨论相关的技巧。让我们从载入购物袋开始吧。

第1步：载入购物袋应用程序

JavaScript和常用浏览器负责这里的大部分工作，哪怕用户运行的只是其中的一个小部分。考虑载入的第一个页面——*index.html*。它的具体内容如例8-1所示。

例8-1：index.html

```
1 <HTML>
```

```
<HEAD>
<TITLE>Shopping Bag</TITLE>
<STYLE TYPE="text/css">
<!--
#welcome { text-align: center; margin-top: 150}
//-->
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
var shopWin = null;
var positionStr = '';
function whichBrowser() {
  if(navigator.appVersion < 4) {
    alert("You need MSIE 4.x or Netscape Navigator 4.x to use " +
      "Shopping Bag.")
    return false;
    }
  return true;
  }

function launch() {
  if(!whichBrowser()) { return; }
  if(navigator.appName == "Netscape")
    { positionStr = ",screenX=0,screenY=0"; }
  else { positionStr = ",fullscreen=yes"; }
  if(shopWin == null) {
    shopWin = open("shopset.html", "", "width=" + screen.width +
      ",height=" + screen.height + positionStr);
    }
  }
function closeUpShop() {
  if (shopWin != null) {
    if (typeof(shopWin) == "object") {
      shopWin.close();
      }
    }
  }
window.onunload = closeUpShop;
//-->
</SCRIPT>
</HEAD>
<BODY>
<DIV ID="welcome">
  <H1>Welcome to Shopping Bag!!!</H1>
  <A HREF="javascript: launch();">Begin</A>
</DIV>
</BODY>
</HTML>
```

这么多JavaScript看起来好像只能构成一个在屏幕上打印五个单词的页面；但是，附加的代码对应用程序有一定的好处。JavaScript 定义并利用一个最高阶层成员来负

责多种窗口的管理，并确定浏览器类型，以便在其他窗口打开的时候提供交叉平台代码。

最高阶层成员

第11行和第32～38行的变量和函数用来施行我们的规则：如果主窗口关闭，就关闭其他窗口，否则，比方说如果用户要重载子窗口的话，购物袋应用程序可能会导致一个很重大的JavaScript错误而死机。

这里是第11行代码：

```
var shopWin = null;
```

以及第32～38行：

```
function closeUpShop() {
  if (shopWin != null) {
    if (typeof(shopWin) == "object") {
      shopWin.close();
      }
    }
  }
window.onunload = closeUpShop;
```

变量*shopWin*的初始值为null，后来被设置为子窗口对象（我的叙述有跳跃，你可以看第27行）。这个窗口关闭的时候，函数`closeUpShop()`将会被调用。这个函数用来确定用户是否还有打开的购物袋子窗口，若有，就把它关闭。如果*shopWin*不等于null，而是一个类型对象，那么子窗口必须是打开的。`closeUpShop()`恰好在卸载前关闭其他窗口。

在这里，用户所关心的唯一事情是点击“Begin”链接来打开子窗口。这将打开*shopset.html*，它一个框架设置。你可以参见例8-2中的代码。

例8-2：shopset.html

```
1 <HTML>
2 <HEAD>
3 <TITLE>Shopping Bag Frameset</TITLE>
4 <SCRIPT LANGUAGE="JavaScript1.2">
5 <!--
6 function resetOpener() {
7   opener.shopWin = null;
```

```
 8   }
 9 //-->
10 </SCRIPT>
11 </HEAD>
12 <FRAMESET ROWS="80%,20%" FRAMEBORDER=0 BORDER=0 onLoad="self.focus();"
13   onUnLoad="resetOpener();">
14 <FRAME SRC="intro.html" NORESIZE>
15 <FRAME SRC="manager.html" NORESIZE>
16 </FRAMESET>
17 </HTML>
```

这是你的基本框架设置，它包括两个行。一个分配给*intro.html*，另一个给*manager.html*。这里没有太多的JavaScript，但是让我们来看看有什么：

```
function resetOpener() {
  opener.shopWin = null;
  }
```

第6～8行有一个叫resetOpener()的函数，只要是父窗口（在本例中是框架设置）中的文档被卸载的任何时候这个函数都被调用。通过将opener.shopWin设置为null，resetOpener()允许用户关闭购物袋子窗口并且再用同样的“Begin”链接重新打开它。

这可能看起来有点微不足道，甚至是多余的。但是要注意，在*index.html*（第27行）中，只有*shopWin*等于null的时候，特定的窗口才打开。关闭窗口不会把*shopWin*的值置为null，因此要用resetOpener()来帮忙。还要注意，FRAMESET标签中的*onLoad*事件处理方法被设置为self.focus()。这可以确保子窗口不会在主窗口之后打开和载入，只留下用户在那里迷惑究竟发生了什么事。

它基本上控制这个框架设置的载入。仍然有三个页面需要载入——*intro.html*、*manager.html*和*inventory.js*。*intro.html*是一个静态的帮助文件。在*manager.html*载入的时候，内含的源文件*inventory.js*一同载入。在这一部分的后面，*manager.html*值得我们对它作一点讨论，但是*inventory.js*中有我们现在就要查看的代码。它相当长，但如果按照结构来建立清单的话将会很有帮助。

inventory.js

*inventory.js*包含三个函数。前两个是构造器函数。一个定义构造器，另一个定义构造器类型。最后一个函数为这些构造器所创建的对象创建数组。自己来看看例8-3的内容。

JavaScript 技巧：多种窗口和文档管理

当你处理一个应用程序，只使用一个主浏览器窗口的时候，你不必为许多的窗口操心。但是，你一旦产生出另一个窗口，你就会需要一个JavaScript窗口管理。窗口应该总是在顶部或者底部吗？父窗口仍然打开吗？当父窗口或子窗口关闭的时候应用程序会发生什么情况？

你大约不想为每一个多窗口的应用程序处理所有这些问题，但是这个问题的确需要解决。通过将变量设置为一直表示窗口状态的值可以使你在这个游戏中处于领先位置。例如，变量*shopWin*在子窗口打开的时候就等于子窗口对象，在它关闭的时候等于null。购物袋应用基于*shopWin*的值来采取动作。你也可以对框架和框架设置采取相似的策略。

变量*gimmeControl*和*browseControl*执行一个相似的函数来管理文档内容。换句话说，基于文档当前的内容，你可以调整你的应用程序的动作。

例 8-3：inventory.js

```
function product(name, description, price, unit) {
  this.name = name;
  this.description = description;
  this.price = price;
  this.unit = unit;
  this.plu = name.substring(0, 3).toUpperCase() +
    parseInt(price).toString();
  this.icon = new Image();
  return this;
  }
function category(name, description) {
  this.name = name;
  this.description = description;
  this.prodLine = eval(name);
  var imgDir = "images/" + name.toLowerCase() + "/";
  for (var i = 0; i < this.prodLine.length; i++) {
    this.prodLine[i].icon.src = imgDir +
      this.prodLine[i].name.toLowerCase() + ".gif";
    }
  return this;
  }
function makeProducts() {
  Appliances = new Array(
    new product("Dryer",
      "Stylish pastel design, contemporary two-button engineering.",
      263.37  ,
      "each"),
```

```
    new product("Hairdryer",
      "Fancy yellowish blast, and durable cord. No expense spared.",
      1.15,
      "pair"),
    new product("Oven",
      "Made in the 1850's, this coal-powered unit quickly blackens any" +
        "favorite dish.",
      865.78,
      "each"),
    new product("Radio",
      "Revolutionary one-channel technology. White noise and static" +
        "included.",
        15.43,
        "each"),
    new product("Toaster",
      "BBQ-style toaster. Only a moderate shock hazard.",
      25.78,
      "each"),
    new product("Washer",
      "Does a great job on partially everything.",
      345.61,
      "each")
     );

  Buildings = new Array(
    new product("Barn",
      "Complete with rusty silo and rotting doors. Pig sty sold" +
      "separately.",
      6350.57,
      "each"),
    new product("Lighthouse",
      "Made of cement. Assorted light bulbs. Three AA batteries " +
      "not included.",
      12351.15,
      "each"),
    new product("Igloo",
      "Made from top grade snow blocks, and includes a chimney and " +
        "5-ton air conditioning unit.",
      954.76,
      "each"),
    new product("City",
      "Buildings, streets, lights, skyline. Excellent volume purchase.",
      334165.95,
      "each"),
    new product("Castle",
      "Sturdy medieval design, complete with alligators in moat, and " +
        "remote control drawbridge.",
      93245.59,
      "each"),
    new product("Tower",
      "Really tall. Ideal for winning friends and spotting forest " +
        "fires.",
        24345.87,
```

```
81          "pair")
82       );
83
84    Clothing = new Array(
85      new product("Bowtie",
86        "Swell red fabric. Doubles a bow for Christmas wreaths or " +
87          "birthday gifts.",
88        5.41,
89        "five"),
90      new product("Necktie",
91        "Be the first (and probably only) one (ever) on your block. " +
92          "Made of genuine burlap.",
93        1.15,
94        "each"),
95      new product("Purse",
96        "Attractive green material. Wards off most mammals.",
97        18.97,
98        "each"),
99      new product("Jacket",
100       "Plush fake fur with fiberglass lining. Washer safe.",
101       180.72,
102       "each"),
103     new product("Glove",
104       "Covers all four fingers and one thumb. Fancy latex design.",
105       6.59,
106       "three"),
107     new product("Dress",
108       "Found at a garage sale. Also doubles as a picnic table cover.",
109       7.99,
110       "each"),
111     new product("Watch",
112       "Geuine replica. Doesn't tell time. You have to look at it.",
113       6.19,
114       "each")
115     );
116
117   Electronics = new Array(
118     new product("Camcorder",
119       "Solar-powered. Free microphone. Custom-built for blackmailing " +
120         "close relatives.",
121       60.45,
122       "each"),
123     new product("Stereo",
124       "Quadraphonic, pre 8-track sound. Leisure suit and roach killer " +
125          "shoes are optional",
126       54.91,
127       "each"),
128     new product("Speaker",
129       "Extra piece of hi-fi junk. Works best if discarded.",
130       1.90,
131       "each"),
132     new product("Remote",
133       "Dozens of buttons. Controls everything- TV, VCR, stereo, " +
```

```
134         "pets, local government.",
135       465.51,
136       "each"),
137     new product("Cellphone",
138       "Product of tin can technology. 35-ft calling area. Dandy " +
139         "lavender plastic.",
140       64.33,
141       "each"),
142     new product("Camera",
143       "Takes brilliant one-color photos. Landfill safe.",
144       2.95,
145      "each"),
146     new product("Television",
147       "Two-channel UHF only model. Wow.",
148       22.57,
149       "each")
150     );
151
152   Food = new Array(
153     new product("Cheese",
154       "Wait 'til you get a wiff. Puts bleu cheese to shame.",
155       3.05,
156       "chunk"),
157     new product("Fries",
158       "More grease than the local car garage. You can't beat the " +
159         "taste, though.",
160       1.15,
161       "box"),
162     new product("Eggs",
163       "The standard breakfast staple.",
164       1.07,
165       "dozen"),
166     new product("Drumstick",
167       "This leg of pterodactyl is a sure crowd pleaser.",
168       100.00,
169       "half ton"),
170     new product("Chips",
171       "Opened-bag flavor. Guaranteed stale, or your money back.",
172       1.59,
173       "bag"),
174     new product("Shrimp",
175       "Great raw, served above room temperature.",
176       2.95,
177       "each")
178     );
179
180   Hardware = new Array(
181     new product("Chainsaw",
182       "Be your own eager beaver with this tree-cutting machine.",
183       226.41,
184       "each"),
185     new product("Cycle",
186       "Mow down the wheat field with a few swipes. Just like the " +
```

```
187         "Grim Reaper's.",
188       11.15,
189       "each"),
190     new product("Hammer",
191       "Tempered steel head, fiberglass handle. Perfect for hitting " +
192         "things.",
193       9 87,
194       "each"),
195     new product("Lawnmower",
196       "Self-propelled (you propel it yourself).",
197       165.95,
198       "each"),
199     new product("Pliers",
200       "Perfect for eye brows and nose hairs.",
201       6.59,
202       "each"),
203     new product("Stake",
204       "This 2-in-1 miracle secures tents or gets rid of vampires.",
205       3.95,
206       "pair")
207     );
208
209   Music = new Array(
210     new product("Bongos",
211       "Great little noise makers for even the most sophisticated " +
212         "occasions.",
213       35.50,
214       "bongo"),
215     new product("Piano",
216       "It ain't grand, but this baby will make you sound like tavern " +
217         "material in no time.",
218       1001.40,
219       "each"),
220     new product("Notes",
221       "Choose from A, B, C, D, E, F, or G. Can be reused in any song.",
222       2.97,
223       "note"),
224     new product("Guitar",
225       "Strum, strum. This one is your fast track to fame and fortune.",
226       241.11,
227       "each"),
228     new product("Trumpet",
229       "Solid copper body, and not many dents. Extra spit valve " +
230         "included.",
231        683.59,
232       "each")
233     );
234
235   categorySet = new Array(
236     new category("Appliances", "Kitchen machines to make life easier"),
237     new category("Buildings", "Architectural structures your can't " +
238       "resist"),
239     new category("Clothing", "Fashionably questionable apparel for " +
```

```
240       "the 21st century"),
241     new category("Electronics", "Nifty gizmos that drain your wallet"),
242     new category("Food", "The best product to order over the Net"),
243     new category("Hardware", "All kinds of general purpose " +
244       "construction tools"),
245     new category("Music", "The hottest new instruments from places " +
246       "you've never heard of")
247       );
248   }
```

产品属性

还记得我们在前边的章节中所用到的JavaScript对象吗？它们回来复仇了！每个产品都被作为一个带有数个属性的对象；也就是说，每一个产品有如下的属性：

name

产品名称

description

产品的基本描述

price

产品价格

unit

产品出售单位，就比如说是按照一打、一对或者单个来出售

plu

价格查找号码：用来做清单跟踪或定购处理的任意的产品号码

icon

每一个产品的图片

为了达到所要的结果，产品构造函数的定义如第1～10行代码所示：

```
function product(name, description, price, unit) {
  this.name = name;
  this.description = description;
  this.price = price;
  this.unit = unit;
  this.plu = name.substring(0, 3).toUpperCase() +
    parseInt(price).toString();
  this.icon = new Image();
  return this;
  }
```

注意这里创建了六个属性，但是只需要四个参数。属性的数目和所需参数的数目是不相关的，但是考虑一下每个属性如何得到它的值。前面的四个是显而易见的。属性名，描述，价格，还有出售单位都被赋为名称相匹配的参数的值。

但是*plu*的境遇就不同了。它实际上是名称和价格属性的合成。名称前边的三个字母的大写形式加上价格的整数值就构成了PLU数字。因此，一个价值5501.00美元的小艇的实体*plu*就是BOA5501。要记住这是任意的。所出售的产品大约都有它们自己的跟踪号码。这样做是为了让事情变得简单一点。最后一个属性是*icon*，它现在被设置为一个新的图片对象。其实对于参数来说没有必要这样做。

产品种类属性

我们知道每个产品是一个*product*对象。同样，每一个产品种类也是一个*category*对象。就像产品有属性一样，种类也是如此。下面来看一看种类对象的属性：

name

种类名称

description

种类基本描述

prodLine

种类中的所有产品

第11～21行的category构造器保存了日期：

```
function category(name, description) {
  this.name = name;
  this.description = description;
  this.prodLine = eval(name);
  var imgDir = "images/" + name.toLowerCase() + "/";
  for (var i = 0; i < this.prodLine.length; i++) {
    this.prodLine[i].icon.src = imgDir +
      this.prodLine[i].name.toLowerCase() + ".gif";
    }
  return this;
  }
```

每一个种类有三个属性——一个名为*name*的字符串，另一个叫*description*的字符串，以及一个名为*prodLine*的数组。属性*name*和*description*看起来很直观，但数组是从哪里来的，你又怎样对它施加eval()函数呢？两个问题的答案等一会儿

就会明了了，但这是一个基本的策略：不管你怎样命名种类，产品队列都将是有相同名称的一个数组。例如，如果你将某个种类命名为 *stereos*（立体声系统），那么包含所有立体声系统产品的数组就会被叫做 *stereos*。也就是说，*prodLine* 将会是变量 *stereo* 的一个拷贝，是一个不同立体声系统产品的数组。

我们记得每一个产品都有一个叫做 *icon* 的属性，它是一个还没有被设置来源的图片对象。让我们就种类名称作更多的相关讨论。不仅每一个种类将它的产品队列保存在有相同名称的数组中，而且这个种类中所有的产品图片也都存储在有相同名称的目录下。

音乐种类的所有产品都保存在 *music/* 目录下。硬件种类的图片目录为 *hardware/*，如此类推。这听起来是非常合理的。如果这种目录结构到位的话，那么我们就可以在构建种类的时候预载入这个种类所有的图片。处理有关工作的代码位于第16～19行：

```
var imgDir = "images/" + name.toLowerCase() + "/";
for (var i = 0; i < this.prodLine.length; i++) {
  this.prodLine[i].icon.src = imgDir +
    this.prodLine[i].name.toLowerCase() + ".gif";
  }
```

如果查看 *\ch08* 中的目录结构，可以看到：

```
images/
  appliances/
  buildings/
  clothing/
  electronics/
  food/
  hardware/
  music/
```

第17行代码将每一个 icon（一个图片）的 SRC 属性设置为 *images/*+产品名称的小写字母形式+*/*+ "*.gif*"。这就又回到了我已经在前面几章中涉及过的命名协议问题。每个产品都有一个有相同名称的图片，它被定位在和产品所属种类有相同名称的一个文件夹中。

```
每一个产品的URL = images/category/product_name.gif
```

如果浏览 *ch08\images*，你会注意到每一个图片名称都和一个购物袋产品相对应，这个产品定位于和购物袋种类相应的文件夹中。这使得事情很简单，让你可以非常容易地添加、删除和保存产品的相关记录。

注意：如果你有很多较大的图片，可以考虑免除图片的预载入。把图片放到客户端上让浏览过程没有延迟当然是好事，但是如果你有许多高质量，大尺寸的图片的话，用户大概就不愿意一直等待500k的图片载入。你自己根据情况来判断好了。

创建产品和种类

你已经看到了构造函数；现在让我们来启动它们。第一件事情，是要创建产品，然后是种类。函数makeProducts()可以完成这两个任务。这里是第22～248行的代码。由于它们绝大多数是对产品构造函数再三的相同调用，我们还是来看看精简了的版本好了：

```
function makeProducts() {
  Appliances = new Array(
    new product("Dryer",
      "Stylish pastel design, contemporary two-button engineering.",
      263.37  ,
      "each"),
    new product("Hairdryer",
      "Fancy yellowish blast, and durable cord. No expense spared.",
      1.15,
      "pair"),
    new product("Oven",
      "Made in the 1850's, this coal-powered unit quickly blackens any" +
        "favorite dish.",
      865.78,
      "each"),
    new product("Radio",
      "Revolutionary one-channel technology. White noise and static" +
        "included.",
        15.43,
        "each"),
    new product("Toaster",
      "BBQ-style toaster. Only a moderate shock hazard.",
      25.78,
      "each"),
    new product("Washer",
      "Does a great job on partially everything.",
      345.61,
      "each")
     );
...
...  等等...
...
  categorySet = new Array(
    new category("Appliances", "Kitchen machines to make life easier"),
    new category("Buildings", "Architectural structures your can't " +
      "resist"),
```

```
    new category("Clothing", "Fashionably questionable apparel for " +
      "the 21st century"),
    new category("Electronics", "Nifty gizmos that drain your wallet"),
    new category("Food", "The best product to order over the Net"),
    new category("Hardware", "All kinds of general purpose " +
      "construction tools"),
    new category("Music", "The hottest new instruments from places " +
      "you've never heard of")
      );
}
```

首先出现的是产品。变量*Appliances*设置为一个数组。数组中的每一个元素都是一个产品对象。每一次对 product 的调用都附带构建一个产品所需的参数——名称，描述，以及出售单位。对 Buildings，Clothing，Electronics，Food，Hardware 和 Music 都是如此。

剩下的所有工作是建造种类。实际上，种类的名称已经有了（用具，建筑，服装，等等）；我们只需要让购物袋应用程序知道这些情况。相关代码在第 235～248 行：

```
categorySet = new Array(
    new category("Appliances", "Kitchen machines to make life easier"),
    new category("Buildings", "Architectural structures your can't " +
      "resist"),
    new category("Clothing", "Fashionably questionable apparel for " +
      "the 21st century"),
    new category("Electronics", "Nifty gizmos that drain your wallet"),
    new category("Food", "The best product to order over the Net"),
    new category("Hardware", "All kinds of general purpose " +
      "construction tools"),
    new category("Music", "The hottest new instruments from places " +
      "you've never heard of")
      );
```

变量*categorySet*也是一个数组。数组中的每一个元素是一个种类对象，它带有两个参量，一个名称，一个描述。第一个参量赋值给 *name* 属性，而第二个赋值给 *description* 属性。再来看看第 14 行种类构造器中的代码：

```
this.prodLine = eval(name);
```

prodLine 被设置为 `eval(name)` 的值。因此，第 249 行对 `category()` 的调用意味着 *prodLine* 等于 `eval("Appliances")`，它的值为 *Appliances*。现在名为 *Appliances* 的种类知道了它所有的产品（在 *prodLine* 数组中）。categorySet 中的每一个元素表示另一个购物袋种类。这对它们的添加和删除是很方便的。

创建购物袋

所有的产品都已经创建好了。剩下的唯一工作是在载入期间创建……哦……创建一个购物袋。所有这些购物袋都需要一些用来处理付款的属性和一个存储用户所选的所有产品的数组。*manager.html*中的Bag()构造器定义一个且仅只一个购物袋。这里是*manager.html*中的第21～31行的代码，在本章后面的例8-4中还要给出：

```
function Bag() {
  this.taxRate   = .06;
  this.taxTotal  = 0;
  this.shipRate  = .02;
  this.shipTotal = 0;
  this.subTotal  = 0;
  this.bagTotal  = 0;
  this.things    = new Array();
  }
shoppingBag = new Bag();
```

有两个用来保存特定比率的变量，*taxRate*和*shipRate*。一个是用来计算正式营业税的倍数。另一个呢，也是一个倍数，它用来计算运费。你的付税结构可能会不尽相同，但是你至少可以在这里把用法弄明白。另外的三个变量，*taxTotal*，*subTotal*和*shipTotal*，分别用来表示税金总数，所有产品选择总数及它们的数量，以及用户应该付款的总数。最后一个变量是一个数组。*things*包含所有的产品，包括用户所选的数量。然后，变量*shoppingBag*被设置为new Bag()值。现在我们可以四处买东西了。

> **JavaScript技巧：用JavaScript对象保存客户端状态**
>
> 注意所有的用户产品选择和购买总量都存储在shoppingBag()的属性中。这是一个有效的方法，可以把在应用程序活动周期中增长和修改的信息组织起来。一般说来，用户可以在数量或者产品选择上作任何的修改，shooppingBag中的属性会作出相应的调整。要记住这个技巧，随时可能用得着。

第2步：显示产品

应用程序装载后，用户要做的第一件事是浏览清单。用户可以用“Previous Category”和“Next Category”链接从一个种类导航到另一个种类，也可以用

“Previous Product”和“Next Product”链接从一个产品导航到另一个产品。这里解释了它是如何工作的。回想一下 *inventory.js* 中的第 235～248 行：

```
categorySet = new Array(
    new category("Appliances", "Kitchen machines to make life easier"),
    new category("Buildings", "Architectural structures your can't " +
      "resist"),
    new category("Clothing", "Fashionably questionable apparel for " +
      "the 21st century"),
    new category("Electronics", "Nifty gizmos that drain your wallet"),
    new category("Food", "The best product to order over the Net"),
    new category("Hardware", "All kinds of general purpose " +
      "construction tools"),
    new category("Music", "The hottest new instruments from places " +
      "you've never heard of")
      );
```

categorySet 有七个种类对象。第一个在 categorySet[0]中，下一个是 categorySet[1]，依次类推。无论用户在浏览什么产品，购物袋知道该产品所属种类的号码（在本例中是0～6）。如果用户决定转移到前一个种类，购物袋就从当前种类号码中减掉1，并且显示这个种类的第一个产品。如果购物袋已经到了种类0，而用户还想转移到前一种类，购物袋就会又从最大的种类号码开始（在本例中为6）。

如果用户想浏览下一种类，购物袋在当前种类号码上加1。如果用户已经到了最后一个种类，购物袋就又从种类0开始。

这一过程对产品同样有效。每一个种类都有一定数量的产品。购物袋知道表示当前浏览产品的号码，因此，选择“Previous Product”或者“Next Product”，将会仅通过减1或者加1转到下一个或者前一个产品的显示，这就要看用户想怎么做了。

当用户到达一个种类中最后一个产品时（并还在向前走），购物袋会发现这一情况，并从下一个种类的产品0开始。当用户到达一个种类中的第一个产品时（并还在向后退），购物袋会发现这个情况，并从前一个种类的最后一个产品开始。

如果上面的解释让你感到头痛的话，下面的图表应该会有帮助。图8-10显示了购物袋怎样带领用户浏览各个种类。对于按照产品进行的导航也是同样的道理。当到达某个种类的最后一个产品时，你会转移到下一个种类的第一个产品。

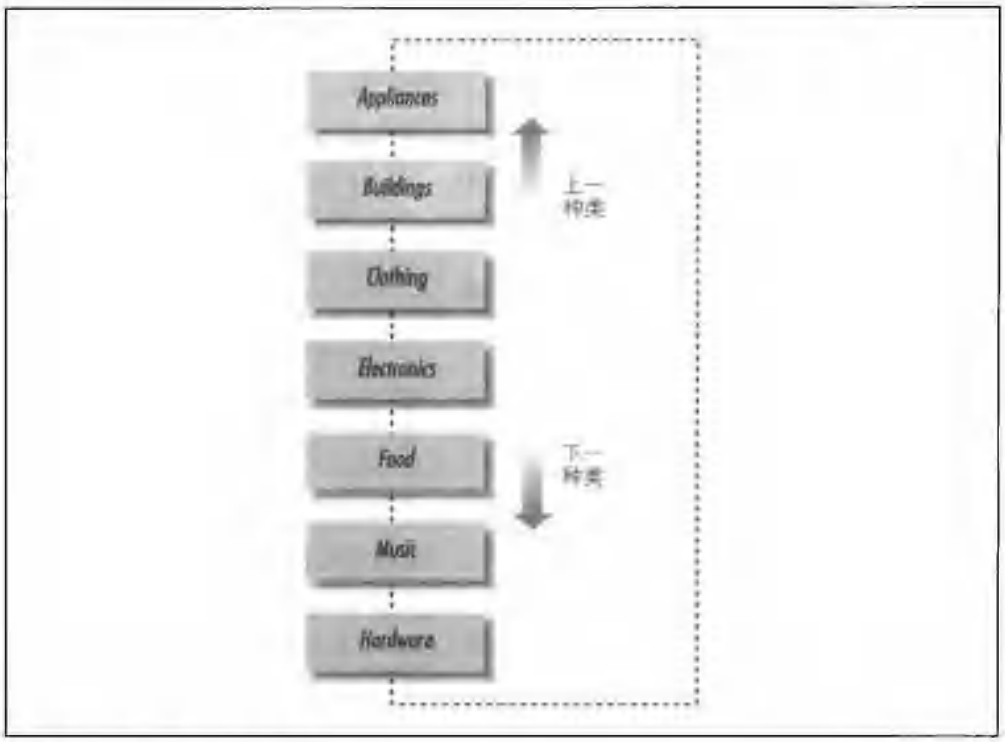


图 8-10 浏览各个种类

manager.html

所有的这些功能都是从 *manager.html* 文件中来的。例 8-4 给出了它的代码。

例 8-4：manager.html

```
<HTML>
<HEAD>
<TITLE>Shopping Bag Manager</TITLE>
<STYLE TYPE="text/css">
<!--
TD {font-weight: bold; margin-left: 20; margin-right: 20; padding: 10}
//-->
</STYLE>
</HEAD>
<BODY onLoad="freshStart(); makeProducts();"
  LINK=BLUE ALINK=BLUE VLINK=BLUE>
<SCRIPT LANGUAGE="JavaScript1.2" SRC="inventory.js"></SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.2">
<!--
var gimmeControl = false;
```

```
16 var browseControl = false;
17 var curCLoc = -1;
18 var curPLoc = -1;
19 var infoStr = '';
20 var shoppingBag;
21 function Bag() {
22   this.taxRate   = .06;
23   this.taxTotal   = 0;
24   this.shipRate   = .02;
25   this.shipTotal  = 0;
26   this.subTotal   = 0;
27   this.bagTotal   = 0;
28   this.things   = new Array();
29   }
30
31 shoppingBag = new Bag();
32
33 function showStore() {
34   gimmeControl = false;
35   var header = '<HTML><TITLE>Category</TITLE><BODY BGCOLOR=FFFFFF>';
36   var intro = '<H2>Shopping Bag Product Categories</H2><B>';
37   var footer = '</DL></BLOCKQUOTE></BODY></HTML>';
38   var storeStr = '<BLOCKQUOTE><DL>';
39   for (var i = 0; i < categorySet.length; i++) {
40     storeStr += '<DT><A HREF="javascript: parent.frames[1].reCall(' +
41       i + ', 0);">' + categorySet[i].name + '</A><DD>' +
42       categorySet[i].description + '<BR><BR>';
43     }
44   infoStr = header + intro + storeStr + footer;
45   parent.frames[0].location.replace('javascript:
46 parent.frames[1].infoStr');
47   }
48
49 function portal() {
50   gimmeControl = false;
51   parent.frames[0].location.href = "search/index.html";
52   }
53 function display(cOffset, pOffset) {
54   if(!browseControl) {
55     alert("Start shopping by selecting a product category from " +
56       "Show All Categories or searching products from Product Search.");
57     return;
58     }
59   gimmeControl = true;
60   if (curPLoc + pOffset < 0 || curPLoc + pOffset ==
61     categorySet[curCLoc].prodLine.length) {
62     if (curPLoc + pOffset < 0) {
63       if (curCLoc - 1 < 0) { curCLoc = categorySet.length - 1; }
64       else { curCLoc--; }
65       curPLoc = categorySet[curCLoc].prodLine.length - 1;
66       }
67     else if (curPLoc + pOffset == categorySet[curCLoc].prodLine.length) {
68       if (curCLoc + 1 == categorySet.length) { curCLoc = 0; }
69       else { curCLoc++; }
```

```
70        curPLoc = 0;
71        }
72      }
73    else {
74      if (curCLoc + cOffset < 0 || curCLoc + cOffset ==
75        categorySet.length) {
76        curCLoc = (curCLoc + cOffset < 0 ? categorySet.length - 1 : 0);
77        }
78      else { curCLoc += cOffset; }
79      if (cOffset == -1 || cOffset == 1) { curPLoc = 0; }
80      else if (pOffset == 0) {
81        curPLoc = (curPLoc >= categorySet[curCLoc].prodLine.length ? 0 :
82          curPLoc)
83        }
84      else { curPLoc = curPLoc + pOffset; }
85      }
86    infoStr = '<HTML><HEAD><TITLE>Product Name</TITLE></HEAD>' +
87      '<BODY><TABLE CELLPADDING=3><TR><TD VALIGN=TOP COLSPAN=2>' +
88      '<FONT FACE=Tahoma><H2>Shopping Bag: <I>' +
89      categorySet[curCLoc].name + '</I></H2><TR>' +
90      '<TD VALIGN=TOP><IMG SRC="' +
91      categorySet[curCLoc].prodLine[curPLoc].icon.src +
92      '"></TD><TD VALIGN=TOP><FONT FACE=Tahoma>' +
93      '<B>Name: </B>' +  categorySet[curCLoc].prodLine[curPLoc].name +
94      '<BR><B>Description: </B>' +
95      categorySet[curCLoc].prodLine[curPLoc].description + '<BR>' +
96      '<B>Price: </B> $' +
97      numberFormat(categorySet[curCLoc].prodLine[curPLoc].price) + '/' +
98      categorySet[curCLoc].prodLine[curPLoc].unit + '<BR>' +
99      '<B>PLU: </B>' + categorySet[curCLoc].prodLine[curPLoc].plu +
100     '</TD></TR></TABLE></BODY></HTML>';
101   parent.frames[0].location.href = 'javascript:
102 parent.frames[1].infoStr';
103   }
104
105 function reCall(cReset, pReset) {
106   browseControl = true;
107   curCLoc = cReset;
108   curPLoc = pReset;
109   display(0, 0);
110   }
111
112 function gimmeOne() {
113   if (!gimmeControl) {
114     alert("Nothing on this screen to give you.");
115     return;
116     }
117   for (var i = 0; i < shoppingBag.things.length; i++) {
118     if (categorySet[curCLoc].prodLine[curPLoc].plu ==
119       shoppingBag.things[i].plu) {
120       alert("That's already in your bag. You can change the quantity " +
121         "by choosing View/Change Bag.");
122       return;
123       }
```

```
124     }
125   shoppingBag.things[shoppingBag.things.length] =
126     categorySet[curCLoc].prodLine[curPLoc];
127   shoppingBag.things[shoppingBag.things.length - 1].itemQty = 1;
128   shoppingBag.things[shoppingBag.things.length - 1].category =
129     categorySet[curCLoc].name;
130   alert("OK. You put the " +
131      shoppingBag.things[shoppingBag.things.length - 1].name +
132     " in your bag.");
133   }
134
135 function showBag() {
136   if (shoppingBag.things.length == 0) {
137     alert("Your bag is currently empty. Put some stuff in.");
138     return;
139     }
140   gimmeControl = false;
141   var header = '<HTML><HEAD><TITLE>Your Shopping Bag</TITLE>' +
142     '</HEAD><BODY BGCOLOR=FFFFFF ' +
143     'onLoad="parent.frames[1].runningTab(document.forms[0]);">';
144   var intro = '<H2>Your Shopping Bag!!!</H2>' +
145     '<FORM onReset="' +
146     'setTimeout(\'parent.frames[1].runningTab(document.forms[0])\', ' +
147     '25);">';
148   var tableTop = '<TABLE BORDER=1 CELLSPACING=0 CELLPADDING=5>' +
149     '<TR><TH><B>Index' +
150     '<TH><B>Product<TH><B>Category' +
151     '<TH><B>PLU<TH><B>Unit Price' +
152     '<TH><B>Quantity<TH><B>Product Total' +
153     '<TH><B>Remove' +
154     '</TR>';
155   var itemStr = '';
156   for (var i = 0; i < shoppingBag.things.length; i++) {
157     itemStr += '<TR>' +
158       '<TD ALIGN=CENTER>' + (i + 1) + '</TD>' +
159       '<TD>' + shoppingBag.things[i].name + '</TD>' +
160       '<TD>' + shoppingBag.things[i].category + '</TD>' +
161       '<TD>' + shoppingBag.things[i].plu + '</TD>' +
162       '<TD ALIGN=RIGHT>$' +
163         parent.frames[1].numberFormat(shoppingBag.things[i].price) +
164       '</TD>' +
165       '<TD ALIGN=CENTER>' +
166       parent.frames[1].genSelect(shoppingBag.things[i].price,
167         shoppingBag.things[i].itemQty, i) + '</TD>' +
168       '<TD ALIGN=CENTER><INPUT TYPE=TEXT SIZE=10 VALUE="' +
169       parent.frames[1].numberFormat(shoppingBag.things[i].price *
170         shoppingBag.things[i].itemQty) +
171       '" onFocus="this.blur();"></TD>' +
172       '<TD ALIGN=CENTER><INPUT TYPE=CHECKBOX></TD>' +
173       '</TR>';
174     }
175   var tableBottom = '<TR>' +
176     '<TD ALIGN=RIGHT COLSPAN=6>SubTotal:</TD>' +
177     '<TD ALIGN=CENTER><INPUT TYPE=TEXT SIZE=10 NAME="subtotal" ' +
```

```
178     onFocus="this.blur();"></TD></TR>' +
179     '<TR><TD ALIGN=RIGHT COLSPAN=6> + 6% Tax:</TD>' +
180     '<TD ALIGN=CENTER><INPUT TYPE=TEXT SIZE=10 NAME="tax" ' +
181     'onFocus="this.blur();"></TD></TR><TR><TD ALIGN=RIGHT COLSPAN=6>' +
182     '2% Shipping:</TD><TD ALIGN=CENTER><INPUT TYPE-TEXT ' +
183     'SIZE=10 NAME="ship" onFocus="this.blur();"></TD></TR>' +
184     '<TR>' +
185     '<TD ALIGN=RIGHT COLSPAN=3><INPUT TYPE=BUTTON VALUE="Check Out" ' +
186     'onClick="parent.frames[1].checkOut(this.form);"></TD>' +
187     '<TD ALIGN=RIGHT><INPUT TYPE=RESET VALUE="Reset Qtys"></TD>' +
188     '<TD ALIGN=RIGHT><INPUT TYPE=BUTTON VALUE="Change Bag" ' +
189     'onClick="parent.frames[1].changeBag(this.form, true);"></TD>' +
190     '<TD ALIGN=RIGHT>Total:</TD><TD ALIGN=CENTER>' +
191     '<INPUT TYPE=TEXT NAME="total" SIZE=10 onFocus="this.blur();">' +
192     '</TD></TR>';
193
194   var footer = '</TABLE></FORM></BODY></HTML>';
195   infoStr = header + intro + tableTop + itemStr + tableBottom + footer;
196   parent.frames[0].location.replace('javascript:
197      parent.frames[1].infoStr');
198   }
199
200 function genSelect(priceAgr, qty, idx) {
201   var selStr = '<SELECT onChange="this.form.elements[' + (idx * 3 + 1) +
202       '].value = this.options[this.selectedIndex].value;
203   parent.frames[1].runningTab(this.form);">';
204   for (var i = 1; i <= 10; i++) {
205     selStr += '<OPTION VALUE="' + numberFormat(i * priceAgr) + '"' +
206       (i == qty ? ' SELECTED' : '') + '>' + i;
207     }
208   selStr += '</SELECT>';
209   return selStr;
210   }
211
212 function runningTab(formObj) {
213   var subTotal = 0;
214   for (var i = 0; i < shoppingBag.things.length; i++) {
215     subTotal += parseFloat(formObj.elements[(i * 3) + 1].value);
216     }
217   formObj.subtotal.value = numberFormat(subTotal);
218   formObj.tax.value = numberFormat(subTotal * shoppingBag.taxRate);
219   formObj.ship.value = numberFormat(subTotal * shoppingBag.shipRate);
220   formObj.total.value = numberFormat(subTotal +
221       round(subTotal * shoppingBag.taxRate) +
222       round(subTotal * shoppingBag.shipRate));
223   shoppingBag.subTotal = formObj.subtotal.value;
224   shoppingBag.taxTotal = formObj.tax.value;
225   shoppingBag.shipTotal = formObj.ship.value;
226   shoppingBag.bagTotal = formObj.total.value;
227   }
228
229 function numberFormat(amount) {
230   var rawNumStr = round(amount) + '';
231   rawNumStr = (rawNumStr.charAt(0) == '.' ? '0' + rawNumStr : rawNumStr);
```

```
232   if (rawNumStr.charAt(rawNumStr.length - 3) == '.') {
233     return rawNumStr
234     }
235   else if (rawNumStr.charAt(rawNumStr.length - 2) == '.') {
236     return rawNumStr + '0';
237     }
238   else { return rawNumStr + '.00'; }
239   }
240 function round(number,decPlace) {
241   decPlace = (!decPlace ? 2 : decPlace);
242   return Math.round(number * Math.pow(10,decPlace)) /
243     Math.pow(10,decPlace);
244   }
245
246 function changeBag(formObj, showAgain) {
247   var tempBagArray = new Array();
248   for (var i = 0; i < shoppingBag.things.length; i++) {
249     if (!formObj.elements[(i * 3) + 2].checked) {
250       tempBagArray[tempBagArray.length] = shoppingBag.things[i];
251       tempBagArray[tempBagArray.length - 1].itemQty =
252         formObj.elements[i * 3].selectedIndex + 1;
253       }
254     }
255   shoppingBag.things = tempBagArray;
256   if(shoppingBag.things.length == 0) {
257     alert("You've emptied your bag. Put some stuff in.");
258     parent.frames[1].showStore();
259     }
260   else { showBag(); }
261   }
262
263 function checkOut(formObj) {
264   gimmeControl = false;
265   if(!confirm("Do you have every product in the right quantity " +
266     "you need? Remember that you have to choose Change Bag to " +
267     "remove products or change quantities. If so, choose OK to check " +
268     "out.")) {
269     return;
270     }
271   if(shoppingBag.things.length == 0) {
272     showStore();
273     return;
274     }
275   var header = '<HTML><TITLE>Shopping Bag Check Out</TITLE>' +
277
276     '<BODY BGCOLOR=FFFFFF>';
278   var intro = '<H2>Shopping Bag Check Out</H2><FORM METHOD=POST ' +
279     'ACTION="http://your.webserver.com/cgi-bin/bag.cgi" ' +
280     'onSubmit="return parent.frames[1].cheapCheck(this);">';
281
282   var shipInfo = '<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=5>' +
283     '<TR><TD><B>Shipping Information</TD></TR>'+
284     '<TR><TD>First Name</TD><TD><INPUT TYPE=TEXT NAME="fname"></TD>' +
285     '</TR><TR><TD>Last Name</TD><TD>' +
```

```
286     '<INPUT TYPE=TEXT NAME="lname"></TD></TR><TR><TD>Company Name</TD>' +
287     '<TD><INPUT TYPE=TEXT NAME="cname"></TD></TR><TR>' +
288     '<TD>Street Address1</TD><TD><INPUT TYPE=TEXT NAME="saddress1">' +
289     '</TD></TR><TR><TD>Street Address2</TD>' +
290     '<TD><INPUT TYPE=TEXT NAME="saddress2"></TD></TR><TR>' +
291     '<TD>City</TD><TD><INPUT TYPE=TEXT NAME="city"></TD></TR>' +
292     '<TR><TD>State/Province</TD>' +
293     '<TD><INPUT TYPE=TEXT NAME="stpro"></TD></TR><TR>' +
294     '<TD>Country</TD><TD><INPUT TYPE=TEXT NAME="country"></TD></TR>' +
295     '<TR><TD>Zip/Mail Code</TD><TD><INPUT TYPE=TEXT NAME="zip"></TD>' +
296     '</TR><TR><TD><BR><BR></TD></TR></TABLE>';
297
298   var payInfo = '<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=5>' +
299     '<TR><TD><B>Payment Information</TD></TR>'+
300     '<TR><TD>Credit Card Type:       </TD>' +
301     '<TD>Visa <INPUT TYPE=RADIO NAME="ctype" VALUE="visa" CHECKED> ' +
302     '      ' +
303     'Amex <INPUT TYPE=RADIO NAME="ctype" VALUE="amex"> ' +
304     '      ' +
305     'Discover <INPUT TYPE=RADIO NAME="ctype" VALUE="disc"> ' +
306     '      </TD></TR>' +
307     '<TR><TD>Credit Card Number</TD>' +
308     '<TD><INPUT TYPE=TEXT NAME="cnumb"></TD></TR><TR>' +
309     '<TD>Expiration Date</TD><TD><INPUT TYPE=TEXT NAME="edate"></TD>' +
310     '</TR><TR><TD><INPUT TYPE=SUBMIT VALUE="Send Order"></TD>' +
311     '<TD><INPUT TYPE=RESET VALUE="Clear Info"></TD></TR>' +
312     '</TABLE>';
313
314   var itemInfo = '';
315   for (var i = 0; i < shoppingBag.things.length; i++) {
316     itemInfo += '<INPUT TYPE=HIDDEN NAME="prod' + i +
317       '" VALUE="' + shoppingBag.things[i].plu + '-' +
318       shoppingBag.things[i].itemQty + '">';
319     }
320   var totalInfo = '<INPUT TYPE=HIDDEN NAME="subtotal" VALUE="' +
321     shoppingBag.subTotal + '">' +
322     '<INPUT TYPE=HIDDEN NAME="taxtotal" VALUE="' +
323     shoppingBag.taxTotal + '">' +
324     '<INPUT TYPE=HIDDEN NAME="shiptotal" VALUE="' +
325     shoppingBag.shipTotal + '">' +
326     '<INPUT TYPE=HIDDEN NAME="bagtotal" VALUE="' +
327     shoppingBag.bagTotal + '">';
328
329   var footer = '</FORM></BODY></HTML>';
330
331   infoStr = header + intro + shipInfo + payInfo + itemInfo +
332     totalInfo + footer;
333     parent.frames[0].location.replace('javascript:
334        parent.frames[1].infoStr');
335   }
336
337 function cheapCheck(formObj) {
338   for (var i = 0; i < formObj.length; i++) {
339     if (formObj[i].type == "text" && formObj.elements[i].value == "") {
```

```
340       alert ("You must complete all fields.");
341       return false;
342       }
343     }
344   if(!confirm("If all your information is correct, " +
345     "choose OK to send your order, or choose Cancel to make changes.")) {
346     return false;
347     }
348   alert("Thank you. We'll be living off your hard-earned money soon.");
349   shoppingBag = new Bag();
350   showStore();
351   return true;
352   }
353
354 function help() {
355   gimmeControl = false;
356   parent.frames[0].location.href = "intro.html";
357   }
359 function freshStart() {
360   if(parent.frames[0].location.href != "intro.html") { help(); }
361   }
362
363 //-->
364 </SCRIPT>
365 <TABLE ALIGN=CENTER BORDER=0>
366   <TR>
367     <TD>
368     <A HREF="javascript: gimmeOne();">Gimme One<A>
369     </TD>
370     <TD>
371     <A HREF="javascript: showBag();">View/Change Bag<A>
372     </TD>
373     <TD>
374     <A HREF="javascript: showStore();">Show All Categories<A>
375     </TD>
376     <TD>
377     <A HREF="javascript: portal();">Product Search<A>
378     </TD>
379     <TD>
380     <A HREF="javascript: help();">Help<A>
381     </TD>
382   </TR>
383 </TABLE>
384 <TABLE ALIGN=CENTER BORDER=0>
385   <TR>
386     <TD>
387     <A HREF="javascript: display(-1,0);">Previous Category<A>
388     </TD>
389     <TD>
390     <A HREF="javascript: display(0,-1);">Previous Product<A>
391     </TD>
392     <TD>
393     <A HREF="javascript: display(0,1);">Next Product<A>
394     </TD>
```

```
395      <TD>
396      <A HREF="javascript: display(1,0);">Next Category<A>
397      </TD>
398    </TR>
399 </TABLE>
400 </BODY>
401 </HTML>
```

很快地浏览一遍下就行。你看到所有的JavaScript都嵌入到BODY标签后面了吗？因为在开始部分有很多的图片预载和对象创建，Netscape Navigator需要将可恶的灰色背景显示在窗口中（或者是框架），直到所有的工作结束。只有在那时它才会来解析后面的内容。随后，浏览器在着手于所有的工作之前将会解析BODY标签，然后才是BGCOLOR属性。

变量

接下来是完成产品显示的代码。第15～18行建立了四个变量，而第53～103行定义了函数display()。变量*gimmeControl*会向购物袋指示屏幕上是否有什么（一个产品）可以添加到购物袋中的东西。变量*browseControl*控制用户必须通过点击“Show All Categories”或者“Product Search”来开始浏览（见规则1）。你可以看到这两个变量贯穿整个应用程序，但是display()会首先来处理它们，因此，先来做个介绍：

```
var gimmeControl = false;
var browseControl = false;
var curCLoc = -1;
var curPLoc = -1;
```

变量*curCLoc*和*curPLoc*分别保存当前显示的种类和产品的索引号码。它们是我在前边的部分已经提到过的数字。虽然两者都被任意地设置为-1，但它们会在用户选择一个种类或者一个产品的时候发生改变。一会儿我们还要讨论有关于这一点的更多内容。现在，让我们来看看这整个的过程。这里是第53～103行的代码：

```
function display(cOffset, pOffset) {
  if(!browseControl) {
    alert("Start shopping by selecting a product category from Show " +
      "All Categories or searching products from Product Search.");
    return;
    }
  gimmeControl = true;
  if (curPLoc + pOffset < 0 || curPLoc + pOffset ==
    categorySet[curCLoc].prodLine.length) {
    if (curPLoc + pOffset < 0) {
```

```
        if (curCLoc - 1 < 0) { curCLoc = categorySet.length - 1; }
        else { curCLoc--; }
        curPLoc = categorySet[curCLoc].prodLine.length - 1;
        }
      else if (curPLoc + pOffset == categorySet[curCLoc].prodLine.length) {
        if (curCLoc + 1 == categorySet.length) { curCLoc = 0; }
        else { curCLoc++; }
        curPLoc = 0;
        }
      }
    else {
      if (curCLoc + cOffset < 0 || curCLoc + cOffset ==
        categorySet.length) {
        curCLoc = (curCLoc + cOffset < 0 ? categorySet.length - 1 : 0);
        }
      else { curCLoc += cOffset; }
      if (cOffset == -1 || cOffset == 1) { curPLoc = 0; }
      else if (pOffset == 0) {
        curPLoc = (curPLoc >= categorySet[curCLoc].prodLine.length ? 0 :
          curPLoc)
        }
      else { curPLoc = curPLoc + pOffset; }
      }
    infoStr = '<HTML><HEAD><TITLE>Product Name</TITLE></HEAD>' +
      '<BODY><TABLE CELLPADDING=3><TR><TD VALIGN=TOP COLSPAN=2>' +
      '<FONT FACE=Tahoma><H2>Shopping Bag: <I>' +
        categorySet[curCLoc].name + '</I></H2><TR>' +
      '<TD VALIGN=TOP><IMG SRC="' +
      categorySet[curCLoc].prodLine[curPLoc].icon.src +
      '"></TD><TD VALIGN=TOP><FONT FACE=Tahoma>' +
      '<B>Name: </B>' +  categorySet[curCLoc].prodLine[curPLoc].name +
      '<BR><B>Description: </B>' +
      categorySet[curCLoc].prodLine[curPLoc].description + '<BR>' +
      '<B>Price: </B> $' +
      numberFormat(categorySet[curCLoc].prodLine[curPLoc].price) + '/' +
      categorySet[curCLoc].prodLine[curPLoc].unit + '<BR>' +
      '<B>PLU: </B>' + categorySet[curCLoc].prodLine[curPLoc].plu +
      '</TD></TR></TABLE></BODY></HTML>';
    parent.frames[0].location.href =
'javascript:parent.frames[1].infoStr';
    }
```

display()

display()要完成三个工作:

1. 确定是否允许显示一个产品。
2. 确定哪个种类/产品是用户要浏览的。
3. 显示指定产品。

第一个任务倒是非常简单。如果*browseControl*为true，那么答案就是yes。*browseControl*的初始值为false。用户一旦从“Product Search”中选择一个产品或者从“Show All Categories”中选择某个种类，*browseControl*就会被设置为true。现在，display()可以执行任务2和3了。由于要显示产品，就把*gimmeControl*设置为true。

注意到display()需要两个参数，*cOffset*和*pOffset*。一个保存用来确定从当前种类号码移动多少距离的值。另一个也一样，不过处理的是产品号码。*cOffset*和*pOffset*可以是正数也可以是负数，还可以为0。为了让事情简单一点，我们来假设购物者Daisy Deep Pockets已经对第一步工作非常满意，现在可以用“Next”和“Previous”来浏览整个存货清单。看看第386～397行中每一个链接的代码：

```
<TD>
<A HREF="javascript: display(-1,0);">Previous Category<A>
</TD>
<TD>
<A HREF="javascript: display(0,-1);">Previous Product<A>
</TD>
<TD>
<A HREF="javascript: display(0,1);">Next Product<A>
</TD>
<TD>
<A HREF="javascript: display(1,0);">Next Category<A>
</TD>
```

每一个链接都调用display()并传递一对整数。表8-1说明每一个函数调用要做些什么工作。记住，*curCLoc*是种类号码，而*curPLoc*是产品号码。

表8-1 确定curCLoc和curPLoc的值

链接	传递的参数	说明
Previous Category（上一种类）	−1, 0	加−1到*curCLoc*; 加0到*curPLoc*
Previous Product（上一产品）	0, −1	加0到*curCLoc*; 加−1到*curPLoc*
Next Product（下一产品）	0, 1	加0到*curCLoc*; 加1到*curPLoc*
Next Category（下一种类）	1, 0	加1到*curCLoc*; 加0到*curPLoc*

异常处理

这是很有意义的。如果你想要退回上一个种类，可以从种类号码上减1。如果你想

看下一个种类，就在种类号码上加 1。但是存在三种异常情况，它们需要附加一些判断：

1. 没有哪个种类或者产品的号码会是 -1。如果种类或者产品号码是 0，而且用户选择了“Previous Category”或是“Previous Product”，就会导致购物袋应用程序错误。
2. 没有哪一个种类会有号码 categorySet[categorySet.length]。由于只有 categorySet.length 个种类，所以种类号码根本不可能比 categorySet.length -1 更大。如果种类号码是 categorySet.length -1，而用户选择“Next Product”或者“Next Category”，我们会得到一个 JavaScript 错误。对于产品的浏览也是同样的情况。
3. 从种类到种类的浏览总是显示一个种类中的第一个产品，而不管用户当前浏览的产品的号码是多少。

第 60～85 行的代码提供我们所要的功能，并且对这三中例外情况作调整。这可是对 if-else 语句的一个非常恰当有效的运用，因此你可能会希望再来看看它。

```
if (curPLoc + pOffset < 0 || curPLoc + pOffset ==
  categorySet[curCLoc].prodLine.length) {
  if (curPLoc + pOffset < 0) {
    if (curCLoc - 1 < 0) { curCLoc = categorySet.length - 1; }
    else { curCLoc--; }
    curPLoc = categorySet[curCLoc].prodLine.length - 1;
    }
  else if (curPLoc + pOffset == categorySet[curCLoc].prodLine.length) {
    if (curCLoc + 1 == categorySet.length) { curCLoc = 0; }
    else { curCLoc++; }
    curPLoc = 0;
    }
  }
else {
  if (curCLoc + cOffset < 0 || curCLoc + cOffset == categorySet.length) {
    curCLoc = (curCLoc + cOffset < 0 ? categorySet.length - 1 : 0);
    }
  else { curCLoc += cOffset; }
  if (cOffset == -1 || cOffset == 1) { curPLoc = 0; }
  else if (pOffset == 0) {
    curPLoc = (curPLoc >= categorySet[curCLoc].prodLine.length ? 0 :
      curPLoc)
    }
  else { curPLoc = curPLoc + pOffset; }
  }
```

下面的伪代码可以把这一部分内容表达得更清楚而易于理解。每一行后面跟着实际代码的行号：

```
 1 IF 产品号码过小或者过大 THEN (73)
 2     IF 产品号码过小 THEN (74)
 3         IF 种类号码过小 THEN 种类号码等于种类总数减1 (75)
 4         ELSE 种类号码等于它自身减1 (76)
 5         产品号码等于这个种类中的产品总数减1 (77)
 6 ELSE IF 产品号码过大 (79)
 7         IF 种类号码过大 THEN 种类号码等于0 (80)
 8         ELSE 种类号码等于它自身加1 (81)
 9         产品号码等于0 (82)
10 ELSE (85)
11     IF 种类号码多大或者过小 THEN (86)
12         IF 种类号码过小 THEN 种类号码等于种类数减1 (87)
13         ELSE 种类号码等于 0 (88)
14     ELSE 种类号码等于它自身加上种类偏移量 (89)
15     IF 种类偏移量等于-1或者1 THEN 产品号码等于0 (90)
16     ELSE IF 产品偏移量等于0 THEN (91)
17         IF 产品号码大于等于种类号码下的产品总数 THEN 产品号码等于 0 (92)
18     ELSE 产品号码等于它自身加上产品偏移量 (94)
```

如果产品号码属于前两种异常情况，最外面的*if*部分对变量做出处理。最外层的else部分处理种类号码属于前两种情况时的变量。为了解决第三种情况，如果种类偏移量按1来增减的话，第80行代码就把产品号码设为0。

建立显示页面

知道了种类和产品的号码，购物袋现在可以建立HTML来显示相应的产品了。display()中几乎全部剩余的代码都用来将产品显示到屏幕上。看一下第86～102行：

```
infoStr = '<HTML><HEAD><TITLE>Product Name</TITLE></HEAD>' +
  '<BODY><TABLE CELLPADDING=3><TR><TD VALIGN=TOP COLSPAN=2>' +
  '<FONT FACE=Tahoma><H2>Shopping Bag: <I>' + categorySet[curCLoc].name +
  '</I></H2><TR><TD VALIGN=TOP><IMG SRC="' +
  categorySet[curCLoc].prodLine[curPLoc].icon.src +
  '"></TD><TD VALIGN=TOP><FONT FACE=Tahoma><B>Name: </B>' +
  categorySet[curCLoc].prodLine[curPLoc].name + '<BR>' +
  '<B>Description: </B>' +
  categorySet[curCLoc].prodLine[curPLoc].description + '<BR>' +
  '<B>Price: </B> $' +
  numberFormat(categorySet[curCLoc].prodLine[curPLoc].price) + '/' +
  categorySet[curCLoc].prodLine[curPLoc].unit + '<BR>' +
  '<B>PLU: </B>' + categorySet[curCLoc].prodLine[curPLoc].plu +
  '</TD></TR></TABLE></BODY></HTML>';
parent.frames[0].location.href = 'javascript: parent.frames[1].infoStr';
```

正如你所看到的，所有的东西都以一个巨大的HTML合并计算为基础，赋值给一个初始为空串的变量*infoStr*。注意到*curPLoc*和*curCLoc*的值对于所有正确的产品信息是起着关键作用的。categorySet[curCLoc]指向正确的种类，而categorySet[curCLoc].prodLine[curPLoc]指向正确的产品。一旦确定了*curCLoc*和*curPLoc*的值，就可以按照任何你喜欢的方式来显示产品了。

等到*infoStr*有了所有要用来显示产品的HTML的时候，顶部框架的*href*属性就按照JavaScript:协议被设置为它的值。记住，因为协议的范围限制，你必须给它提供一个绝对的引用索引（也就是parent.frames[1].infoStr，而不是刚才的*infoStr*）。要获得更多的相关细节内容，可以参见第二章的JavaScript技巧部分。

第3步：显示所有的种类

选择“Show All Categories”可以用另外一种方法来导航产品。函数showStore()已经做好了准备来完成这个任务，如第33～47行所示：

```
function showStore() {
  gimmeControl = false;
  var header = '<HTML><TITLE>Category</TITLE><BODY BGCOLOR=FFFFFF>';
  var intro = '<H2>Shopping Bag Product Categories</H2><B>';
  var footer = '</DL></BLOCKQUOTE></BODY></HTML>';
  var storeStr = '<BLOCKQUOTE><DL>';
  for (var i = 0; i < categorySet.length; i++) {
    storeStr += '<DT><A HREF="javascript: parent.frames[1].reCall(' + i +
      ', 0);">' + categorySet[i].name + '</A><DD>' +
      categorySet[i].description + '<BR><BR>';
    }
  infoStr = header + intro + storeStr + footer;
  parent.frames[0].location.replace('javascript:
    parent.frames[1].infoStr');
  }
```

显示第一个产品

当然，规则1表明，用户第一次（只是第一次）显示一个产品必须从“Show All Categories”或“Product Search”开始。“Product Search”的问题一会儿再说。现在让我们先来看看“Show All Categories”。显示所有的种类其实相当简单。showStore()只是迭代*categorySet*中的所有元素，产生一个附带每个种类的名称和

描述的链接列表。完成最后一个种类之后，这个链接列表字符串（也叫做 *infoStr*）就被作为顶部框架的 HREF 属性了。注意，每一个 HREF 标签中提供的代码等于：

```
javascript: parent.frames[1].reCall(' + i + ', 0)
```

点击任何一个种类链接都将调用 *manager.html* 中的 `reCall()` 函数。这里是第 105～110 行代码：

```
function reCall(cReset, pReset) {
  browseControl = true;
  curCLoc = cReset;
  curPLoc = pReset;
  display(0, 0);
  }
```

`reCall()`需要两个参数，由第 42 行的 i 值表示的种类号码和数字 0。*i* 的值被设置为 *curCLoc*。当然，这确定了用户想要浏览的产品。数字 0 被赋给 *curPLoc*。还记得第三种异常情况吗？按种类浏览的方式通常让用户从本种类中的第一个产品开始，它也就是 `prodLine[0]`。

在此之后，`reCall()`调用函数 `display()`，传递两个 0 作为参数。当我们查看 *if* 语句的第 60～85 行部分，我们假定用户总是用一个大于或小于 *curCLoc*，或者等于 *curPLoc* 的值来浏览产品或者种类。具体情况是，函数 `reCall()`已经给这些变量赋了值，因此就没有必要“到任何地方去”。用户想看的是当前 *curCLoc* 和 *curPLoc* 的相应产品。这就是传递两个 0 的意思，第 60～85 行代码处理这部分内容。

DHTML 在哪儿？

注意到产品页面没有 DHTML。没有层。为什么会没有呢？这里的绝大多数浏览器支持 JavaScript 1.2。我甚至花了两章的内容来讲跨浏览器 DHTML。为什么现在又不要它了？当购物车最初载入的时候，难道你没有为每一个产品创建一个层，然后按要求显示和隐藏它们吗？对，你这样做了的，但是……

过多的图片预载会给你带来很大害处。就像前面曾经提到过的一样，如果你有很多图片，将它们全部预载简直就是在考验购物者的耐心。为每个产品创建一个层就载入了这些图片。不管有没有预载，使用简单的 HTML 就可以把这些工作做完。

现在，用户可以随心所欲地来按照产品和种类导航了。该来看看当用户发现什么东西值得买，并且决定拿一个来放到购物袋里的时候，将会出现什么情况了。

第 4 步：将产品添加到购物袋

把东西放到购物袋里是非常简单的。用户只需点击亲切的“Gimme One”链接。这将调用有相应名称的函数 gimmeOne()。第 112～133 行是相关细节内容：

```
function gimmeOne() {
  if (!gimmeControl) {
    alert("Nothing on this screen to give you.");
    return;
    }
  for (var i = 0; i < shoppingBag.things.length; i++) {
    if (categorySet[curCLoc].prodLine[curPLoc].plu ==
      shoppingBag.things[i].plu) {
      alert("That's already in your bag. You can change the quantity " +
        "by choosing View/Change Bag.");
      return;
      }
    }
  shoppingBag.things[shoppingBag.things.length] =
    categorySet[curCLoc].prodLine[curPLoc];
  shoppingBag.things[shoppingBag.things.length - 1].itemQty = 1;
  shoppingBag.things[shoppingBag.things.length - 1].category =
    categorySet[curCLoc].name;
  alert("OK. You put the " +
    shoppingBag.things[shoppingBag.things.length - 1].name +
    " in your bag.");
  }
```

gimmeOne()首先做的工作是确保屏幕上的确有可以放到购物袋里的东西。变量 *gimmeControl* 在显示产品之前的瞬间被设置为 true。不然，任何其他函数在屏幕上显示信息都将把 *gimmeControl* 置为 false。因此，如果 *gimmeControl* 是 false 的话，就表明屏幕上根本没有什么产品。用户接到警告信息，gimmeOne()返回。否则，gimmeOne()迭代 *things* 数组（它是用户的 *shoppingBag* 对象中的属性）的所有元素，来检查当前显示的产品是否已经在用户的袋子里了。

gimmeOne()不需要任何参数。它依赖的是 *curCLoc* 和 *curPLoc* 的当前值。假定每一个产品都有一个唯一的 PLU 号码，gimmeOne()就寻找和当前袋中的产品相匹配的任何PLU号码。如果发现了一个匹配，用户就得到这个产品已经在袋中的提示。

如果产品还没有被放到袋子里，gimmeOne()会将它放入。但是，这将造成一个很有意思的情况。来回答一下这个问题：袋子里的产品仍然是一个 *product* 对象——对还是错？如果回答“对”的话，那么你回答正确了。但是，用户购物袋中的任何

产品肯定是一个扩展的，更加复杂的*product*。袋子里的每个产品仍然有名称，描述，PLU，以及价格，但是它们每一个也同样需要一个属性来指示购买的数量，还要“知道”它是属于哪一个种类的。

因此，必须给*things*中的每一个元素添加有用的属性。第125～129行的代码显示了函数gimmeOne()是怎样给*things*添加这些特殊产品，以及怎样给每一个对象添加属性的：

```
shoppingBag.things[shoppingBag.things.length] =
  categorySet[curCLoc].prodLine[curPLoc];
shoppingBag.things[shoppingBag.things.length - 1].itemQty = 1;
shoppingBag.things[shoppingBag.things.length - 1].category =
  categorySet[curCLoc].name;
```

shoppingBag.things[shoppingBag.things.length]创建了一个对categorySet[curCLoc].prodLine[curPLoc]中当前*product*对象的引用索引。下面的两行分别添加属性*itemQty*（初始值为1），以及当前*product*对象所属种类的名称category。

gimmeOne()最后完成的工作是提示用户，产品已经成功地放到购物袋中去了。这个过程对用户放到购物袋中的每一个产品都会执行一次，直到最后结帐。

JavaScript技巧：添加对象属性

有两种方法可以给用户定义的对象添加属性。最简单的一种只考虑属性名称和值，然后将它添加到对象。things中的每一个元素都是一个产品，但是这些产品被赋了两个新的值，itemQty和category。有关代码如下所示：

```
shoppingBag.things[shoppingBag.things.length - 1].itemQty = 1;
shoppingBag.things[shoppingBag.things.length - 1].category =
  categorySet[curCLoc].name;
```

不管如何，这些对象已经被建立起来了，因此，每一次都必须添加属性。如果你想要给以后构造的所有对象添加属性的话，可以使用*prototype*属性。假设你想要给所建造的任何其他产品添加一个出售价格：

```
product.prototype.salePrice = 0.00;
```

那么构造的任何其他对象现在都将有一个*salePrice*属性，它的默认值是0.00。

搜索产品

到现在，你可能已经注意到了："产品搜索"功能简直就是第一章的移植。客户端搜索引擎被调整来适应购物袋用户的需要。每一样东西几乎都完全相同。但搜索性能被减少到仅用一个默认的OR逻辑操作搜索。换句话说，如果在产品信息里发现任何用户输入的文字，那个产品就被认为是一个匹配。并没有AND逻辑搜索，也没有利用URL的搜索。尽管如此，它的性能对于购物袋应用程序来说还是绰绰有余的。还有一个第一章所没有的重要特征。用户可以通过仅仅敲Enter键来达到输入空字符串的目的。这将执行一个空搜索，它返回的是数据库中所有的产品信息。

我们在这里不打算做和第一章一样细致深入的讨论，但是你应该读一遍接下来的几个段落，就可以看到扩展有组织的JavaScript应用是多么简单。毕竟，和其他任何搜索引擎一样，用户只是想输入文字，然后产生一个链接结果列表。为了让它能为购物袋应用服务而又不对应用程序的代码产生太多的改变，搜索引擎需要做下面的调整：

- 显示的链接结果要支持在第2步中所述的产品/种类导航系统
- 能够搜索现有的数据库
- 对数据库中所有的产品返回链接

幸运的是，这些改变都能在一个文件——*search/nav.html*中进行。因此你不必老盯着另外一堆两百来行的代码，我只在*search/nav.html*中给出相关的代码。

为产品和种类绘制"地图"

我们需要把所有的东西都作一点点的改变以使操作更加正常和稳定。这些变化包括一个有两个新变量和一个新函数的表单，如下所示：

```
var ref = top.frames[1];
var prodProfiles = new Array();
function genProfile() {
  for (var i = 0; i < ref.categorySet.length; i++) {
       for (var j = 0; j < ref.categorySet[i].prodLine.length; j++) {
        prodProfiles[prodProfiles.length] = new Array(i, j);
         }
    }
  }
```

变量 *ref* 用作 top.frames[1]的一个别名。由于这个搜索引擎应用中所涉及的绝大多数对象和变量都定位于 *manager.html* 中，因此，湮没在众多对象中的对象将得到很多的引用，带有非常多的圆点符号。使用 *ref* 来代替 top.frames[1]减少了一点书写的麻烦。*prodProfiles* 以空数组作为初始值，但是很快将被对函数 genProfile() 的调用所填充。

genProfile()有一个任务且只有一个任务——建立一个系统，通过种类号码和产品号码来指示任何种类中的任何产品。种类号码伴随一个产品号码。

例如，假定种类号码 *i* 为 1，而产品号码 *j* 为 2。如果查看 *inventory.js*，你将会发现 categorySet[i].prodLine[j]指向“Buildings”种类中的“Igloo”。这就像是一个地图中的测绘坐标一样。

genProfile()中嵌套的 *for* 循环迭代每一个 category(i)和 product(j)。如果你愿意，genProfile()将通过在它自己的一个数组里存储整数对 *i*, *j* 来为种类中的每一个产品位置作记录。所有这一切都做完之后，*prodProfiles* 中的每一个元素表示一个种类/产品号码对，指向某个种类中特定的一个产品。

你可能会问：这难道不是指示产品的方法吗？答案是 yes。反正，搜索引擎函数现在知道了所有可能的组合。每一个对作为一个元素存储在 *prodProfiles* 中。这使得指引（还有搜索和显示）数据库中关于任何产品的信息变得非常简单。

搜索现有数据库

最初的版本搜索一个网页的名称，描述和URL。购物袋有相似的搜索项目。问题在于，你必须按照现有的数据库来工作。幸运的是，你可以对函数 allowAny()作一些修改。这里是 *search/nav.html* 中的相关内容：

```
function allowAny(t) {
  var findings = new Array();
  for (var i = 0; i < prodProfiles.length; i++) {
    var compareElement  = ref.categorySet[prodProfiles[i][0]].
prodLine[prodProfiles[i][1]].name + ' ' +
      ref.categorySet[prodProfiles[i][0]].prodLine[prodProfiles[i][1]].
        description + ' ' +
      ref.categorySet[prodProfiles[i][0]].prodLine[prodProfiles[i][1]].
        price.toString() + ' ' +
      ref.categorySet[prodProfiles[i][0]].prodLine[prodProfiles[i][1]].
```

```
        plu;
      compareElement = compareElement.toUpperCase();
    for (var j = 0; j < t.length; j++) {
      var compareString = t[j].toUpperCase();
      if (compareElement.indexOf(compareString) != -1) {
        findings[findings.length] = new Array(prodProfiles[i][0],
          prodProfiles[i][1]);
        break;
        }
      }
    }
  verifyManage(findings);
  }
```

改变的其实并没有多少内容。唯一需要我们关心的是搜索。因此，用户可以搜索产品名称，描述，价格和PLU。`allowAny()`在数据库中将每一个产品的这四项内容连接在一起。这就制造了一个很长的字符串，用来和用户输入的每一个词作比较。如果出现一个匹配，下一个可用的*findings*元素就被设置为一个新的数组，`prodProfiles[i][0]`和`prodProfiles[i][1]`将作为它的元素。记住这两个元素是要马上要用来打印结果的整数。

支持产品/种类导航

假设你执行了一个搜索，返回一系列结果，现在要把它弄到页面上去。如果你决定编制代码从搜索引擎来显示产品，而忽略当前的产品/种类系统的话，恐怕你将会成为腕骨综合症的第一个候选人了——工作量太大了。事实上，不管要将什么产品链接显示到结果页面上，用户都必须能够像平时一样显示产品，并且用“Next”和“Prev”按钮自由地导航。我们只对函数`formatResults()`做一点修改：

```
function formatResults(results, reference, offset) {
  docObj.open();
  docObj.writeln('<HTML>\n<HEAD>\n<TITLE>Search Results</TITLE></HEAD>' +
    '<BODY BGCOLOR=WHITE TEXT=BLACK>' +
    '<TABLE WIDTH=780 BORDER=0 ALIGN=CENTER CELLPADDING=3><TR><TD>' +
    '<HR NOSHADE WIDTH=100%></TD></TR><TR><TD VALIGN=TOP><B>' +
    'Search Query: <I>' +
    parent.frames[0].document.forms[0].query.value + '</I><BR>\n' +
    'Search Results: <I>' + (reference + 1) + ' - ' +
    (reference + offset > results.length ? results.length :
      reference + offset) +
    ' of ' + results.length + '</I><BR><BR>' + '<B>' +
    '\n\n<!- Begin result set //-->\n\n\t<DL>');
  var currentRecord = (results.length < reference + offset ?
    results.length : reference + offset);
```

```
for (var i = reference; i < currentRecord; i++) {
  docObj.writeln('\n\n\t<DT>' + '<FONT SIZE=4>' +
    '<A HREF="javascript: top.frames[1].reCall(' + results[i][0] +
    ', ' + results[i][1] + ')">' +
    ref.categorySet[results[i][0]].prodLine[results[i][1]].name +
    '</A></FONT>\t<DD>' +
   ref.categorySet[results[i][0]].prodLine[results[i][1]].description +
    '\t<DD>' + 'Price: <I>$' +
   ref.numberFormat(ref.categorySet[results[i][0]].
     prodLine[results[i][1]].price) +
    '</I>       ' + 'PLU Number: <I>' +
    ref.categorySet[results[i][0]].prodLine[results[i][1]].plu +
    '</I><P>');
  }
docObj.writeln('\n\t</DL>\n\n<!- End result set //-->\n\n');
prevNextResults(results.length, reference, offset);
docObj.writeln('<HR NOSHADE WIDTH=100%>' +
  '</TD>\n</TR>\n</TABLE>\n</BODY>\n</HTML>');
docObj.close();
document.forms[0].query.select();
}
```

每一个结果显示产品的名称，描述，价格和PLU号码。这个函数迭代结果中的元素，并利用 results[i][0]和results[i][1]中的整数来访问它们各自的 *prodLine* 信息。也就是说，如果结果是这样的：

```
results = new Array(
  new Array(0, 1),    // 记住，0元素代表
  new Array(2, 2),    // 种类号，1元素
  new Array(4, 1)     // 代表产品号
  );
```

那么搜索结果就会包含吹风机（种类 0，产品 1），钱包（种类 2，产品 2），以及油炸食品（种类 4，产品 1）。使用这些号码对可以轻松地存储少量的信息，并在以后引用它们。

链接中的代码

已经显示了结果，但是怎样才能给出适当的编码，使链接可以利用我们所讲的导航系统呢？

formatResults()提供了如下所示的解决之道：

```
'<A HREF="javascript: top.frames[1].reCall(' + results[i][0]+ ', ' +
  results[i][1] + ')">'
```

JavaScript 技巧：重新使用 JavaScript 数据库

如果程序员能够使用以JavaScript对象和数组形式存储的数据，他一定非常高兴。如果能从一个不同的应用程序中访问信息，又不用从头开始恐怕会更高兴。这就是购物袋和产品搜索功能所产生的情况。由于相关设计很简单，在搜索引擎要进行搜索的时候数据库没有必要做修改。

搜索引擎的一些代码发生了变化，情况又像往常一样摆到了面前。如果你预见到了这样的情形，有不止一个应用程序要访问你的JavaScript数据库，那么应该让它足够简单，使所有的应用程序能够获得数据而不需要增加代码。

为了获得屏幕上的产品信息而需要做的所有事情就是使用号码对。formatResults()中的*for*循环通过在如下所示的变量中嵌入号码对来打印名称、描述、价格和PLU号码。

```
ref.categorySet[results[i][0]].prodLine[results[i][1]].name
ref.categorySet[results[i][0]].prodLine[results[i][1]].description
ref.numberFormat(ref.categorySet[results[i][0]].prodLine[results[i][1]]
  price)
ref.categorySet[results[i][0]].prodLine[results[i][1]].plu
```

每一个结果将连同上面这些包含在字符串中的值一起显示。一个简单的结果举例：

吹风机

奇特的淡黄色气流。经久耐用，物美价廉。
价格：$1.15
PLU 号码：HA11

每一个链接都使用一个javascript:协议来调用函数reCall()，它同样是用来从“Show All Categories”列表中浏览产品的函数。就如同你可能“回忆”起来的一样，reCall()需要两个参数——一个种类号码和一个产品号码。这是我们已经在搜索引擎中使用过的东西。我们所要做的所有工作是在调用中囊括号码对的每一个元素，然后就完工了。因此，关于吹风机的例子将会有如下的链接：

```
<A HREF="javascript: top.frames[1].reCall(0, 1)">Hairdryer</A>
```

来看一看reCall()启用0和1时会发生什么情况：

```
function reCall(cReset, pReset) {
  browseControl = true;
  curCLoc = cReset;
  curPLoc = pReset;
  display(0, 0);
  }
```

变量 *curCLoc* 被设置为 *cReset* 中种类号码的值；在 *curPLoc* 和 *pReset* 中也同样如此。搜索引擎存在于剩余的购物袋应用程序当中，并不需要多少调整。

第 5 步：修改定单 / 付帐

等用户的钱快花光了，或者找不到其他想买的东西，就该找大门出来啦。点击“View/Change Bag”链接打开如图 8-8 所示的界面。用户的购物袋除了显示他的选择外还有其他更多的事要做。考虑所有的要求：

- 显示每一个产品及其种类，PLU 号码和单位价格。
- 提供一个交互式的表单来修改产品数量，删除产品选择，以及重新计算产品价钱。
- 显示包括每个产品和数量的总量，分量，以及包括恰当的税金在内的总计。

你恐怕一点都不会吃惊，还有好几个函数正准备着提供这些购物袋应用需求。如下所示：

`showBag()`

显示购物袋的内容。

`genSelect()`

产生用来修改产品数量的动态选择列表。

`runningTab()`

管理任何价格和花费的计算和显示。

`numberFormat()`

确保 0.00 形式下计算和内容显示的正确性。

`round()`

确保计算正确。

changeBag()

在用户的购物袋中删除产品选择和修改产品数量。

在用户点击链接的时候将调用函数 showBag()。看一看第 135～198 行：

```
function showBag() {
  if (shoppingBag.things.length == 0) {
    alert("Your bag is currently empty. Put some stuff in.");
    return;
    }
  gimmeControl = false;

  var header = '<HTML><HEAD><TITLE>Your Shopping Bag</TITLE>' +
    '</HEAD><BODY BGCOLOR=FFFFFF ' +
      'onLoad="parent.frames[1].runningTab(document.forms[0]);">';

  var intro = '<H2>Your Shopping Bag!!!</H2><FORM onReset="' +
    'setTimeout(\'parent.frames[1].runningTab(document.forms[0])\', ' +
    '25);">';

  var tableTop = '<TABLE BORDER=1 CELLSPACING=0 CELLPADDING=5>' +
    '<TR><TH><B>Index' +
    '<TH><B>Product<TH><B>Category' +
    '<TH><B>PLU<TH><B>Unit Price' +
    '<TH><B>Quantity<TH><B>Product Total' +
    '<TH><B>Remove' +
    '</TR>';

  var itemStr = '';
  for (var i = 0; i < shoppingBag.things.length; i++) {
    itemStr += '<TR>' +
      '<TD ALIGN=CENTER>' + (i + 1) + '</TD>' +
      '<TD>' + shoppingBag.things[i].name + '</TD>' +
      '<TD>' + shoppingBag.things[i].category + '</TD>' +
      '<TD>' + shoppingBag.things[i].plu + '</TD>' +
      '<TD ALIGN=RIGHT>$' +
      parent.frames[1].round(shoppingBag.things[i].price) + '</TD>' +
      '<TD ALIGN=CENTER>' +
      parent.frames[1].genSelect(shoppingBag.things[i].price,
      shoppingBag.things[i].itemQty, i) + '</TD>' +
      '<TD ALIGN=CENTER><INPUT TYPE=TEXT SIZE=10 VALUE="' +
      parent.frames[1].numberFormat(shoppingBag.things[i].price *
      shoppingBag.things[i].itemQty) + '" onFocus="this.blur();"></TD>' +
      '<TD ALIGN=CENTER><INPUT TYPE=CHECKBOX></TD></TR>';
    }

  var tableBottom = '<TR>' +
    '<TD ALIGN=RIGHT COLSPAN=6>SubTotal:</TD>' +
    '<TD ALIGN-CENTER><INPUT TYPE=TEXT SIZE=10 NAME="subtotal" ' +
    'onFocus="this.blur();"></TD></TR><TR>' +
    '<TD ALIGN=RIGHT COLSPAN=6> + 6% Tax:</TD>' +
```

```
    '<TD ALIGN=CENTER><INPUT TYPE=TEXT SIZE=10 NAME="tax" ' +
    'onFocus="this.blur();"></TD></TR><TR>' +
    '<TD ALIGN=RIGHT COLSPAN=6> + 2% Shipping:</TD>' +
    '<TD ALIGN=CENTER><INPUT TYPE=TEXT SIZE=10 NAME="ship" ' +
    'onFocus="this.blur();"></TD></TR>' +
    '<TR><TD ALIGN=RIGHT COLSPAN=3>' +
    '<INPUT TYPE=BUTTON VALUE="Check Out" ' +
    'onClick="parent.frames[1].checkOut(this.form);"></TD>' +
    '<TD ALIGN=RIGHT><INPUT TYPE=RESET VALUE="Reset Qtys"></TD>' +
    '<TD ALIGN=RIGHT><INPUT TYPE=BUTTON VALUE="Change Bag" ' +
    'onClick="parent.frames[1].changeBag(this.form, true);"></TD>' +
    '<TD ALIGN=RIGHT>Total:</TD><TD ALIGN=CENTER>' +
    '<INPUT TYPE=TEXT NAME="total" SIZE=10 onFocus="this.blur();">' +
    '</TD></TR>';

  var footer = '</TABLE></FORM></BODY></HTML>';
  infoStr = header + intro + tableTop + itemStr + tableBottom + footer;
  parent.frames[0].location.replace('javascript:
     parent.frames[1].infoStr');
  }
```

你会看到showBag()除了产生如图8-8所示的表格和表单以外什么也没有做。但是showBag()首先必须检验购物袋中的确有产品可供显示：

```
if (shoppingBag.things.length == 0) {
  alert("Your bag is currently empty. Put some stuff in.");
  return;
  }
```

如果things.length等于0，那么用户并没有在袋子里放任何东西。就没有必要再继续下去了。如果用户袋子里至少有一样东西，过程就继续。第140～154行设置变量*header*，*intro*和*tableTop*，产生表格的表头和必要的列。你可以看到showBag()调用了其他好几个函数：

```
gimmeControl = false;
var header = '<HTML><HEAD><TITLE>Your Shopping Bag</TITLE>' +
  '</HEAD><BODY BGCOLOR=FFFFFF ' +
  'onLoad="parent.frames[1].runningTab(document.forms[0]);">';

var intro = '<H2>Your Shopping Bag!!!</H2><FORM onReset=' +
  '"setTimeout(\'parent.frames[1].runningTab(document.forms[0])\', ' +
  '25);">';

var tableTop = '<TABLE BORDER=1 CELLSPACING=0 CELLPADDING=5>' +
  '<TR><TH><B>Index' +
  '<TH><B>Product<TH><B>Category' +
  '<TH><B>PLU<TH><B>Unit Price' +
  '<TH><B>Quantity<TH><B>Product Total' +
  '<TH><B>Remove' +
```

```
    '</TR>';
```

产生的所有东西都是静态的，除了onLoad事件处理对parent.fames[1].running-Tab()的调用之外。我们将看到它们怎样在瞬间发生作用。当表格的表头已经确定，就该迭代用户购物袋中所有的产品了。你可能已经猜到了，showBag()利用things.length进行循环，为每一个产品建立一个表格填充相应的数据。这里再次给出第155～174行的代码：

```
var itemStr = '';
for (var i = 0; i < shoppingBag.things.length; i++) {
  itemStr += '<TR>' +
    '<TD ALIGN=CENTER>' + (i + 1) + '</TD>' +
    '<TD>' + shoppingBag.things[i].name + '</TD>' +
    '<TD>' + shoppingBag.things[i].category + '</TD>' +
    '<TD>' + shoppingBag.things[i].plu + '</TD>' +
    '<TD ALIGN=RIGHT>$' +
    parent.frames[1].round(shoppingBag.things[i].price) + '</TD>' +
    '<TD ALIGN=CENTER>' +
    parent.frames[1].genSelect(shoppingBag.things[i].price,
    shoppingBag.things[i].itemQty, i) + '</TD>' +
    '<TD ALIGN=CENTER><INPUT TYPE=TEXT SIZE=10 VALUE="' +
    parent.frames[1].numberFormat(shoppingBag.things[i].price *
    shoppingBag.things[i].itemQty) + '" onFocus="this.blur();"></TD>' +
    '<TD ALIGN=CENTER><INPUT TYPE=CHECKBOX></TD>' +
    '</TR>';
  }
```

建造选择列表

为了匹配刚才创建的表格标头，*for*循环创建了一个产品索引列（这样你可以一个接一个地清点产品），一个名称、种类、PLU、单位价格、数量选择列表、总价，还有一个用来删除选择的复选框。每项内容都包含在它自己的TD标签中。数量选择列表的创建比平时要麻烦一点，值得认真看一看。列表是由函数genSelect()创建的。在这本书里可能有你较为熟悉的版本。在这里我们再看看第200～210行的代码：

```
function genSelect(priceAgr, qty, idx) {
 var selStr = '<SELECT onChange="this.form.elements[' + (idx * 3 + 1) +
     '].value = this.options[this.selectedIndex].value; ' +
     'parent.frames[1].runningTab(this.form);">';
 for (var i = 1; i <= 10; i++) {
   selStr += '<OPTION VALUE="' + numberFormat(i * priceAgr) + '"' +
     (i == qty ? ' SELECTED' : '') + '>' + i;
   }
 selStr += '</SELECT>';
```

```
return selStr;
}
```

这个函数接收了三个参数——产品价格，当前数量和用来访问要显示在选择列表后面的文本域的数字（是showBag()中*for*循环的*i*当前值）。我事先已经把用户所能订购的最大数量设为了10。你可以根据自己的需要随便把它改成多少。为了创建列表，genSelect()从1循环到10，按照下面的语法创建一个OPTION标签，赋值给*selStr*。

```
selStr += '<OPTION VALUE="' + numberFormat(i * priceAgr) + '"' +
  (i == qty ? ' SELECTED' : '') + '>' + i;
```

所创建的每一个OPTION标签都非常简单。它的VALUE属性被设置为单位价格乘以i的计算结果，其中*i*是这个选项的相关产品数量。例如，单位价格为$1.00的一个产品将会产生下面的OPTION标签：

```
<OPTION VALUE="1.00" SELECTED>1
<OPTION VALUE="2.00">2
<OPTION VALUE="3.00">3
<OPTION VALUE="4.00">4
<OPTION VALUE="5.00">5
<OPTION VALUE="6.00">6
<OPTION VALUE="7.00">7
<OPTION VALUE="8.00">8
<OPTION VALUE="9.00">9
<OPTION VALUE="10.00">10
```

另外一个问题：如果*i*等于产品的当前数量（*qty*），有同样数量的OPTION标签就会被标记为SELECTED。由于最初放到购物袋中的每个项目的默认值都是1，那么文字为1的OPTION标签就是被选中的。这在用户想要修改数量的时候非常方便。我们马上就要对此作讨论。

我跳过了*selStr*的初始值。现在让我们倒回去看看：

```
var selStr = '<SELECT onChange="this.form.elements[" + (idx * 3 + 1) +
  '].value = this.options[this.selectedIndex].value; ' +
  'parent.frames[1].runningTab(this.form);">';
```

每一个选择列表都有和它相关的onChange事件处理方法，它将elements[(idx * 3)+ 1]的值改成当前选项的值。记住，每一个OPTION的值都是产品单价与1至10的某个整数的乘积。对此，我们举$1.00为例，如果用户从选择列表中选了4，

elements[(idx * 3) + 1]的值就会变成4.00。它可真是很机灵哦。这是哪一个表单元素呢？为了得到答案，我们重新回顾一下showBag()中第166～167行的代码：

```
parent.frames[1].genSelect(shoppingBag.things[i].price,
  shoppingBag.things[i].itemQty, i)
```

现在来看看第200行中genSelect()所要求的参数：

```
function genSelect(priceAgr, qty, idx) {
```

从它们两个中，你可以看到*idx*的值总是*i*的当前值，它经过了初始化，在第156行加1。如果用户的购物袋中有10个产品，*idx*将会在1～10的范围内变化。因此，showBag()要创建的选择标签将是如下所示的样子：

```
<!-- For the 1st Product //-->
<SELECT onChange='this.form.elements[1].value =
  this.options[this.selectedIndex].value;
  parent.frames[1].runningTab(this.form);'>
<!--For the 2nd Product //-->
<SELECT onChange='this.form.elements[4].value =
  this.options[this.selectedIndex].value;
  parent.frames[1].runningTab(this.form);'>
<!--For the 3rd Product //-->
<SELECT onChange='this.form.elements[10].value =
  this.options[this.selectedIndex].value;
  parent.frames[1].runningTab(this.form);'>
```

……依次类推。仔细考虑一下。form.elements[1]是紧跟在第一个选择列表后面的文本域。或者至少会是：在创建*onChange*事件处理方法的时候，这个域不存在。form.elements[4]表示下一行中紧跟在选择列表后面的文本域。指示文本域是一个计算在创建表单之后它将有哪一个元素索引的问题。这里给出的是我得到(idx * 3) + 1的方法。

选中的每一个产品都被显示在表格的一行中，每一个表格行有三个相同顺序的表单元素：

- 一个数量选择列表
- 一个用来显示产品总数的文本域
- 一个用来删除产品的复选框

这表示第一个文本域是elements[1]，下面一个是elements[4]。因此，文本域是三个元素中的第二个。genSelect()通过不断地乘3再加1来创建正确的代码。

保存帐单记录

onChange事件处理方法中剩余的代码又怎么样呢？这个事件处理方法不仅用适当的总数来分别生成产品总数域，还调用runningTab()来重新计算购物的总花销。这里是位于第212～227行的runningTab()：

```
function runningTab(formObj) {
  var subTotal = 0;
  for (var i = 0; i < shoppingBag.things.length; i++) {
    subTotal += parseFloat(formObj.elements[(i * 3) + 1].value);
    }
  formObj.subtotal.value = numberFormat(subTotal);
  formObj.tax.value = numberFormat(subTotal * shoppingBag.taxRate);
  formObj.ship.value = numberFormat(subTotal * shoppingBag.shipRate);
  formObj.total.value = numberFormat(subTotal +
  round(subTotal * shoppingBag.taxRate) + round(subTotal *
    shoppingBag.shipRate));
  shoppingBag.subTotal = formObj.subtotal.value;
  shoppingBag.taxTotal = formObj.tax.value;
  shoppingBag.shipTotal = formObj.ship.value;
  shoppingBag.bagTotal = formObj.total.value;
  }
```

这个函数相当简单。它执行三个基本操作：

1. 计算和显示所有产品总数的合计（第213～217行）。
2. 计算和显示营业税，运费，以及它们的总数（第218～222行）。
3. 在*shoppingBag*属性中存储这些总数（第223～226行）。

函数numberFormat()和round()确保所有的数学计算都是正确的，并且都按照0.00或者.00的形式来显示。它们在第229～239行：

```
function numberFormat(amount) {
  var rawNumStr = round(amount) + '';
  rawNumStr = (rawNumStr.charAt(0) == '.' ? '0' + rawNumStr :
rawNumStr);
  if (rawNumStr.charAt(rawNumStr.length - 3) == '.') {
    return rawNumStr
    }
```

```
  else if (rawNumStr.charAt(rawNumStr.length - 2) == '.') {
    return rawNumStr + '0';
    }
  else { return rawNumStr + '.00'; }
  }
```

numberFormat()仅仅返回一个完成舍入的0.00形式下的amount。这是通过调用round()并将amount作为参数传递来完成的。函数round()将数字舍入为默认的两位小数形式，如下所示：

```
function round(number,decPlace) {
  decPlace = (!decPlace ? 2 : decPlace);
  return Math.round(number *
    Math.pow(10,decPlace)) / Math.pow(10,decPlace);
  }
```

完成 showBag()：显示总数和其他信息

现在，每一个选中的产品都有了它自己的表格行，并带有用来计算数量和从购物袋中删除产品的小部件。那么就到了添加最后两个行的时候了。这些行包含用来显示总计、税金总额和总和的表单域。它们还包含用户动作按钮“Check Out（结帐）”，“Reset Qtys（重置数量）”和“Change Bag（修改购物袋）”。这里是第175～194行的代码：

```
var tableBottom = '<TR>' +
  '<TD ALIGN=RIGHT COLSPAN=6>SubTotal:</TD>' +
  '<TD ALIGN=CENTER><INPUT TYPE=TEXT SIZE=10 NAME="subtotal" ' +
  'onFocus="this.blur();"></TD></TR><TR>' +
  '<TD ALIGN=RIGHT COLSPAN=6> + 6% Tax:</TD><TD ALIGN=CENTER>' +
  '<INPUT TYPE=TEXT SIZE=10 NAME="tax" onFocus="this.blur();">' +
  '</TD></TR><TR><TD ALIGN=RIGHT COLSPAN=6> + 2% Shipping:</TD>' +
  '<TD ALIGN=CENTER><INPUT TYPE=TEXT SIZE=10 NAME="ship" ' +
  'onFocus="this.blur();"></TD></TR><TR>' +
  '<TD ALIGN=RIGHT COLSPAN=3><INPUT TYPE=BUTTON VALUE="Check Out" ' +
  'onClick="parent.frames[1].checkOut(this.form);"></TD>' +
  '<TD ALIGN=RIGHT><INPUT TYPE=RESET VALUE="Reset Qtys"></TD>' +
  '<TD ALIGN=RIGHT><INPUT TYPE=BUTTON VALUE="Change Bag" ' +
  'onClick="parent.frames[1].changeBag(this.form, true);"></TD>' +
  '<TD ALIGN=RIGHT>Total:</TD><TD ALIGN=CENTER>' +
  '<INPUT TYPE=TEXT NAME="total" SIZE=10 onFocus="this.blur();">' +
  '</TD></TR>';
var footer = '</TABLE></FORM></BODY></HTML>';
```

JavaScript 技巧：数字舍入和字符串转化

你可能会想，要求JavaScript用alert()来处理1.15 × 3的乘积，也就是油炸食品的单价乘上数字3，将不会是一个很复杂的计算。我们都知道它的结果会等于3.45，对吗？但是你来试一试的话，你得到的将是3.4499999999999997。这是从哪里来的？JavaScript用带符号的64位IEEE-754浮点值来表示数字。浮点值的精度位导致了这个结果。如果你想知道更多详情，请参见下面的地址：

http://help.netscape.com/kb/client/970930-1.html
http://www.psc.edu/general/software/packages/ieee/ieee.html

不管它导致的是什么，我们需要的是一个工作区。让JavaScript处理115 × 3的乘积又会怎么样呢？注意到这次的结果是345，是前边乘积的100倍。JavaScript给出的答案也同样是这个数字。对于整数运算，工作是正常的。这里的工作区执行的是整数的运算，然后把JavaScript中的*Number*转变成一个JavaScript的*String*，在正确的地方嵌入小数点。这正是函数numberFormat()和round()一起完成的工作。如果你需要执行更多的计算，你可以把字符串还原成数字，删除小数点，然后就可以继续运算了。

名为*amount*的参数是一个数字，如果它等于用Math.round()舍入后的结果，那么*amount*就是一个整数，因此它需要在后面添加.00来构成正确的格式。如果amount*10等于Math.round(amount*10)，就意味着*amount*是一个0.0形式的数，需要在后面加上一个0才正确。否则，*amount*是一个至少有百分位的值，就不再需要处理了。

所有必需的格式化工作完成之后，round()接收到一个*amount*的等价串。

重新浏览这些HTML，你可以看到用来显示总数的域最初是空的。在文档的*onLoad*事件处理方法中调用runningTab()，可以很快地用所需的值生成好这些域。还要注意，每一个域中有如下的代码：

```
onFocus='this.blur();'
```

因为没有必要让用户修改这些域，直接在文本域中点击鼠标可能会把域弄乱。这些代码能防止人们改变文字，对产品总数输入域也是如此。

有三个按钮："Check Out"，"Reset Qtys" 和 "Change Bag"。我们来看看每一个的内容。

"Check Out" 按钮

当用户有足够的东西之后，他只需要输入支付信息并且发送定单。点击这个按钮将调用函数 checkOut()。这个函数做下面的两个工作：

- 产生一个定制的表单以输入支付信息
- 产生一个附加的域用来表示所有选中的产品

这个函数太长了，所以我们只看其中的两个部分。这里是第 263～312 行代码：

```
function checkOut(formObj) {
  gimmeControl = false;
  if(!confirm("Do you have every product in the right quantity " +
    "you need? Remember that you have to choose Change Bag to remove " +
    "products or change quantities. If so, choose OK to check out.")) {
    return;
    }
  if(shoppingBag.things.length == 0) {
    showStore();
    return;
    }
  var header = '<HTML><TITLE>Shopping Bag Check Out</TITLE>' +
    '<BODY BGCOLOR=FFFFFF>';
  var intro = '<H2>Shopping Bag Check Out</H2><FORM METHOD=POST " +
    'ACTION="http://your_web_server/cgi-bin/bag.cgi" ' +
    'onSubmit="return parent.frames[1].cheapCheck(this);">';

  var shipInfo = '<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=5>' +
    '<TR><TD><B>Shipping Information</TD></TR><TR>' +
    '<TD>First Name</TD><TD><INPUT TYPE=TEXT NAME="fname"></TD></TR>' +
    '<TR><TD>Last Name</TD><TD><INPUT TYPE=TEXT NAME="lname">' +
    '</TD></TR><TR><TD>Company Name</TD>' +
    '<TD><INPUT TYPE=TEXT NAME="cname"></TD></TR><TR>' +
    '<TD>Street Address1</TD><TD><INPUT TYPE=TEXT NAME="saddress1">' +
    '</TD></TR><TR><TD>Street Address2</TD><TD>' +
    '<INPUT TYPE=TEXT NAME="saddress2"></TD></TR><TR>' +
    '<TD>City</TD><TD><INPUT TYPE=TEXT NAME="city"></TD></TR>' +
    '<TR><TD>State/Province</TD><TD><INPUT TYPE=TEXT NAME="stpro">' +
    '</TD></TR><TR><TD>Country</TD><TD>' +
    '<INPUT TYPE=TEXT NAME="country"></TD></TR><TR>' +
    '<TD>Zip/Mail Code</TD><TD><INPUT TYPE=TEXT NAME="zip"></TD>' +
    '</TR><TR><TD><BR><BR></TD></TR></TABLE>';
```

```
var payInfo = '<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=5><TR>' +
  '<TD><B>Payment Information</TD></TR><TR>' +
  '<TD>Credit Card Type:       </TD>' +
  '<TD>Visa <INPUT TYPE=RADIO NAME="ctype" VALUE="visa" CHECKED> ' +
  '      ' +
  'Amex <INPUT TYPE=RADIO NAME='ctype' VALUE="amex"> ' +
  '      ' +
  'Discover <INPUT TYPE=RADIO NAME="ctype" VALUE="disc"> ' +
  '      </TD></TR><TR>' +
  '<TD>Credit Card Number</TD><TD><INPUT TYPE=TEXT NAME="cnumb">' +
  '</TD></TR><TR><TD>Expiration Date</TD>' +
  '<TD><INPUT TYPE=TEXT NAME="edate"></TD></TR><TR>' +
  '<TD><INPUT TYPE=SUBMIT VALUE="Send Order"></TD>' +
  '<TD><INPUT TYPE=RESET VALUE="Clear Info"></TD></TR></TABLE>';
```

它的确很长，但是所有的代码都百分之百是静态的。这里的代码提供一个付帐表单，如图 8-8 所示。这个表单包含让用户输入基本支付信息的域。每一个域都有唯一的名称，这样，服务器端进程的脚本就能正确地确定每一个信息。

函数checkOut()的最后部分准备HIDDEN域来记录用户选择的所有产品的信息。这里是第 314～319 行的代码：

```
for (var i = 0; i < shoppingBag.things.length; i++) {
  itemInfo += '<INPUT TYPE=HIDDEN NAME="prod' + i +
    '" VALUE="' + shoppingBag.things[i].plu + '-' +
    shoppingBag.things[i].itemQty + '">';
```

这产生一个HIDDEN域，命名为*prod+i*的值。而值设置为*PLU-quantity*的形式。因此，如果用户要订购两个油炸食品，那么HIDDEN域的value属性就是VALUE="FRI1-2"。待到每一个当选产品及其数量的HIDDEN域创建好了之后，checkOut()将所有作为HIDDEN域的value的总数连接起来。

第 320～327 行给出了具体的方法：

```
var totalInfo = '<INPUT TYPE=HIDDEN NAME="subtotal" VALUE="' +
  shoppingBag.subTotal + '">' +
  '<INPUT TYPE=HIDDEN NAME="taxtotal" VALUE="' +
  shoppingBag.taxTotal + '">' +
  '<INPUT TYPE=HIDDEN NAME="shiptotal" VALUE="' +
  shoppingBag.shipTotal + '">' +
  '<INPUT TYPE=HIDDEN NAME="bagtotal" VALUE="' +
  shoppingBag.bagTotal + '">';
```

添加一个“Send Order”按钮来提交信息，一个“Clear Info”按钮来清空表单，这就完成了整个付帐表单。在我们开始其他内容之前，注意这个运行中生成的表单的

onSubmit 事件处理方法调用函数 cheapCheck()。这个函数只用来确保在提交表单的时候，那些用户必须输入信息的域不是空的。

下面是第 337～352 行的代码：

```
function cheapCheck(formObj) {
  for (var i = 0; i < formObj.length; i++) {
    if (formObj[i].type == "text" && formObj.elements[i].value == "") {
      alert ("You must complete all fields.");
      return false;
      }
    }
  if(!confirm("If all your information is correct, choose OK to send " +
    "your order, or choose Cancel to make changes.")) {
    return false;
    }
  alert("Thank you. We'll be living off your hard-earned money soon.");
  shoppingBag = new Bag();
  showStore();
  return true;
  }
```

如果有任何一个域是空白的，cheapCheck()就会给用户警告，并返回 false 防止表单的提交。你或许会想设计定制一个表单确认函数来满足你自己的需要，但这至少给了你一个起点。还要注意，如果用户正确地填完了表单，那么变量 *shoppingBag* 就会被设置为一个崭新的 Bag()，并且通过调用 showStore()将用户转到所有购物袋种类的显示页面上去。

完成显示

既然 showBag()已经将变量 *header*，*intro*，*tableTop*，*itemStr*，*totalInfo*，*tableBottom* 和 *footer* 设置为所有要用来创建一个连续显示的值，这个函数将所有这些信息放到屏幕上去。查看一下第 195～197 行的代码：

```
infoStr = header + intro + tableTop + itemStr + tableBottom + footer;
parent.frames[0].location.replace('javascript:parent.frames[1].infoStr');
```

服务器端的情况如何呢？

购买过程在客户端已经结束了。但用户究竟应该怎样付款和接货呢？你最终需要一些类型的服务器端处理来记录订购信息，也就是说，需要一个用来处理这些问题的

数据库。我已经包括了一个用Perl写的框架CGI脚本，它创建一个服务器上的特定ASCII文件，用于处理每一个写入此文件的将所有产品和支付信息。你可以在*\ch08\bag.cgi*中看到这个脚本。下面的程序将向你说明怎样使其为购物袋应用程序工作。假设你的Web服务器上已经装了Perl，并且*bag.pl*所在的目录已经写入并执行了授予的许可。

1. 将*bag.cgi*拷贝到能运行CGI脚本的文件夹（也就是*cgi-bin*）。
2. 在第279行将ACTION属性改为Web服务器上存放*bag.cgi*位置的URL。

当用户选择“Check Out”时，*bag.cgi*将会被调用来处理信息，并返回一个接收的基本确认。顺便说一句，处于安全性考虑，你也可以用SSL（Secure Socket Layer，安全套接字层）服务器或者某种加密技术，来交换服务器与客户之间信用卡和订购信息。

“Reset Qtys”

这是一个简单的“Reset”按钮，用来清除有任何改变的表单。如果你选择“Change Bag”，那么就可以改变袋子中的内容和数量。

“Change Bag”

假设用户已经在产品数量上做了一些修改，并且点中了另外两个产品的“Remove”复选框。选择“Change Bag”将记录这些修改，然后重新显示购物袋，给出新的内容和数量。这里是第246～261行的函数changeBag()：

```
function changeBag(formObj, showAgain) {
  var tempBagArray = new Array();
  for (var i = 0; i < shoppingBag.things.length; i++) {
    if (!formObj.elements[(i * 3) + 2].checked) {
      tempBagArray[tempBagArray.length] = shoppingBag.things[i];
      tempBagArray[tempBagArray.length - 1].itemQty =
        formObj.elements[i * 3].selectedIndex + 1;
      }
    }
  shoppingBag.things = tempBagArray;
  if(shoppingBag.things.length == 0) {
    alert("You've emptied your bag. Put some stuff in.");
    parent.frames[1].showStore();
    }
```

```
    else { showBag(); }
    }
```

这个技巧非常简单：

1. 创建一个称为 *tempBagArray* 的空数组。
2. 迭代 *things* 数组中的每一个元素（一个 *product*）。
3. 如果相关复选框被点中，将它作为 *tempBagArray* 的下一个元素添加进去。
4. 将 *tempBagArray* 最近添加的产品数量设置为用户在相关选择列表中所选的数量。
5. 将 *things* 设为与 *tempBagArray* 相等，并显示购物袋内容。

与函数 runningTab() 设置每一个产品总数值的方法相同，对各个复选框进行引用。下面每一个复选框用 index(i*3)+2 元素来访问。如果迭代之后发现 *tempBagArray* 中没有元素，那么用户已经从袋子中删除了所有的产品。将向提示用户这个信息，并且开始下一次选取。

被遗忘的函数

我们已经几乎讲完了所有东西，但还有三个函数尚未涉及。它们扮演的是小角色，但值得一提。它们是 portal()，help()，freshStart()。这里是第 49～52 行的 portal()：

```
function portal() {
  gimmeControl = false;
  parent.frames[0].location.href = "search/index.html";
  }
```

由于显示产品搜索引擎界面不会显示任何产品，变量 *gimmeControl* 将会在载入 *search/index.html* 之前被设置为 false。第 354～357 行的 help() 也是这样：

```
function help() {
  gimmeControl = false;
  parent.frames[0].location.href = "intro.html";
  }
```

唯一的区别是，parent.frames[0] 载入的是 *intro.html*。最后，还有 freshStart()：

```
function freshStart() {
  if(parent.frames[0].location.href != "intro.html") { help(); }
  }
```

函数freshStart()确保在每一次用户载入（准确的说是重载入）*shopset.html*的时候，parent.frames[0]总是从*intro.html*开始。你可以在第10行中的*onLoad*事件处理方法中看到它。

应用程序扩展

只要一点点创造力就可以让你改变这个应用程序。下面是一些能想到的可能性：

- 制造“更聪明”的产品。
- 添加精确的搜索性能。
- 为频繁的购买者添加cookie性能。

制造“更聪明”的产品

好像不是每一个产品都有它自己的智商。但是，假设你在*product*构造器中添加一个属性，一个数组，用来保存那些和当前所浏览的产品相关的名称和种类/产品号码对。那么*product*构造器可能会是这个样子：

```
function product(name, description, price, unit, related) {
  this.name = name;
  this.description = description;
  this.price = price;
  this.unit = unit;
  this.related = related;
  this.plu = name.substring(0, 3).toUpperCase() +
    parseInt(price).toString();
  this.icon = new Image();
  return this;
  }
```

参数*related*表示一个可以赋值给*related*属性的数组。不管用户什么时候浏览产品，你都能迭代*related*元素，为它们产生链接。这个方法可以交叉营销你的产品。

如果它听起来让你感到有点陌生的话，可以访问amazon.com。搜索一本讲你最喜欢

的主题的书。当你从results列表中点击这个链接时，描述这本书的页面也将提供对其他一些书的链接，买了你浏览的这本书的人还买了所链接的这些书（译注1）。

添加精确的搜索性能

这又回到了搜索引擎性能的扩展，但是你可以用好几种方法对它做修改以满足购物袋的需要。首先，考虑添加逻辑操作AND搜索性能。这并不会太难。只要从第一章的应用程序中拷贝函数requireAll()，然后对所有的owAny()进行如上所述的修改。你还需要调整validate()来表示要使用哪一个搜索函数。

用户可能只想搜索一个或多个种类，而不是去搜索整个的产品数据库。考虑在*nav.html*中添加一个多种类的选择列表：

```
<SELECT MULTIPLE SIZE=5>
<OPTION VALUE="Appliances">Appliances
<OPTION VALUE="Building">Building
<OPTION VALUE="Clothing">Clothing
<OPTION VALUE="Electronics">Electronics
<OPTION VALUE="Food">Food
<OPTION VALUE="Hardware">Hardware
<OPTION VALUE="Music">Music
</SELECT>
```

虽然编制这些代码很麻烦，但是你可以使用和genSelect()中相似的逻辑来获得一个动态的结果。搜索完成之后，你可以只在用户所选的种类中搜索这些产品，因而缩小搜索范围。

另外一个你可以添加的性能是，可以用价格范围来进行搜索。如果购物者在寻找低于$50，高于$100，或者可能在$50到$100之间的产品，你可以承认诸如>，<，>=和<=的标记。你将需要在validate()和formatResults()中进行适当的修改来适应这个功能。

添加cookie

假设你的站点中产品变化得很快。可能经常有购物者频繁地访问。如果他们不必每一次都填写支付信息的话，那将是非常方便的。为什么不按照第七章讲述的方法来

译注1：你可以在Amazon.com上随处看到这种应用。

添加cookie函数呢？那样的话，用户的信息被保存在客户浏览器中，可以被访问，并用来生成支付表单。这个强大的功能可能有一点挑战性，但等你完成的时候，你会觉得自己像是一个JavaScript英雄。虽然函数GetCookie()和SetCookie()能让你写cookie以及从cookie获取数据，但你会需要一个函数来集中表单域中的数据（可能是一个单纯的串），还需要一个函数来在数据被取出时生成表单域。

第九章

JavaScript 中的加密技术

应用程序要点

- 使用多种密码的消息加密技术
- 面向对象的设计使加密更容易
- 有趣的应用程序和对访问者的功能

JavaScript 技巧

- 为对象定义函数
- 更多的字符串匹配和替代
- 进入JavaScript对象继承
- 使用交替的语法

如果你刚看完了上一章的内容，那么这一章可就要让你费费脑筋了。本章是一个热点，基于简单有趣的JavaScript加密技术处理一个应用程序。应用程序会将文字消息打乱成一堆乌七八糟的东西，以对付那些妄图窃取情报的人。

图 9-1 所显示的界面的确很简单。随着 Caesar 密码的选取，将得到一段描述密码的文字，一个用来选取数字密钥的选择列表，以及一个用来输入文字以编码和解码的文本域。

在文本域中输入文字 "JavaScript is the scripting language of choice across the planet, don't you agree?" 在 "Shift" 列表中选择 6，然后选择 "Encipher（编码）" 按钮来产生如图 9-2 所示的乱七八糟的文字。也就是：

```
pglgyixovz oy znk yixovzotm rgtm0gmk ul inuoik jutz 4u0 gmxkk
```

选择 "Decipher（解码）" 将在初始表单中返回文字。注意文字全部以小写字母的形式返回。Vigenere 密码以同样的方式工作。从顶部的密码列表中选择 "Vigenere Cipher"，你将会看到如图 9-3 所示的结果。

图 9-1 加密界面

图 9-2 使用 Caesar 密码

使用这种密码，就没有用来选择数字密钥的列表。但是这一次，有一个用来输入单词或者短语作为密码的域。检验一下条目“code junky”怎样算出原始文字。我们在这里列出，图 9-4 中也是一样的。

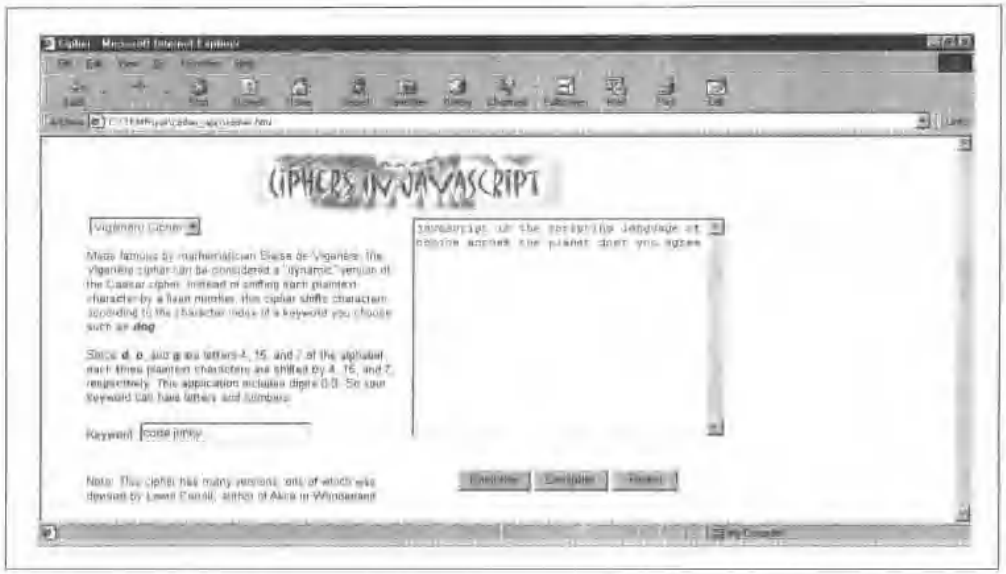


图 9-3 Vigenere Cipher 界面

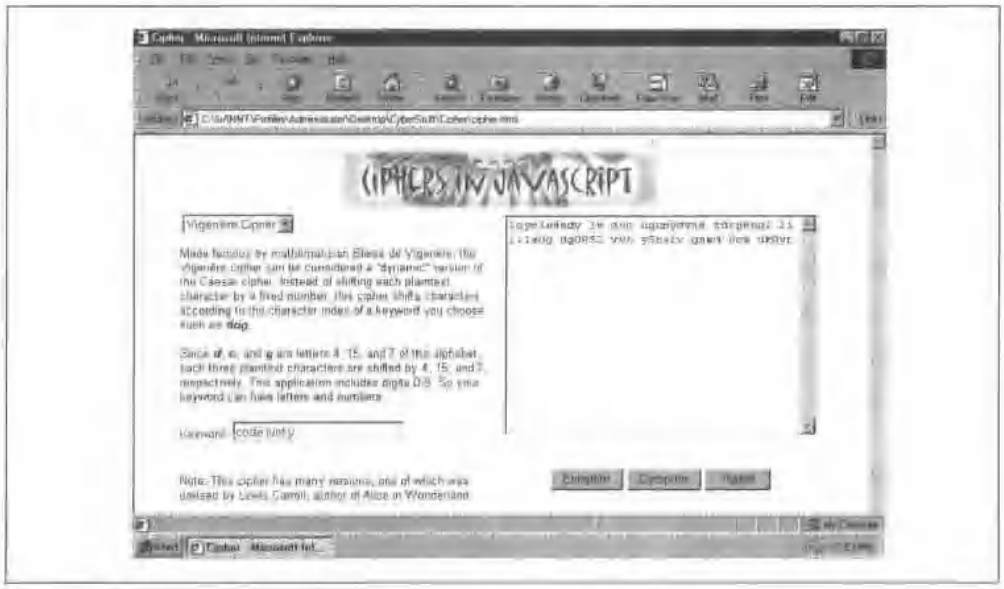


图 9-4 Vigenere 密码的运行

```
loye1w4sdv 1w duo uqumydvx4 zdrpenq2 2i 111s0g dg0852 vvh y5nx2v gswd 8cw
dk0yr
```

当然，选择“Decipher”会将“code junky”作为密钥而将文字返回给相应的表单。由于你可能是第一次接触密码概念，这里有一个关于此主题的快速教程。它将讲解密码的基础知识，并给出应用程序中所使用的两种密码——Caesar密码和Vigenere密码的具体内容。

密码如何工作

现在我们来看加密究竟是指什么？加密是一个算法或者算法集，它能系统地将对人有意义的文字转变成看来毫无意义的乱码，这些乱码只可以被经授权的收件人转变回发送人的原始文字。下面的条目和定义可以帮助你理解一般的译码和解码，以及它们后面的代码。

条目原文表示发送人的原始消息。原文中的含义是发送人要传达给接收人的信息。

加密文字是指已经被编码或被算术改变的原文。加密文字一旦被解码就变成了原文。

许多密码使用一个或多个密钥。密钥是文本或者位字符串，用来编码或解码数据。RSA 数据安全有限公司（http://www.rsa.com/），是一个技术领先的加密技术公司，规定密钥用来确定原文到加密文字之间的映射。密钥可以是任何东西，比如单词“cleveland”，短语“winners never quit, quitters never win”，二进制数字10011011，或者甚至是一些莫名其妙的串，比如%_-:;(<<*&^。

发送器和接收器都用同样的密钥来编码和解码消息的加密技术被称为*symmetric-key cryptosystem*（对称密钥密码系统）。而数据编码和解码使用一对密钥——一个自由公开给公众，另一个只被接收者掌握——被称为*public-key cryptosystem*（公共密钥密码系统）。这个应用程序中的加密术使用的是对称密钥密码系统。

有记载的密码多达数百种，有的要追溯到几千年以前，由历史上那些伟大领袖或者科学家所发明；其他的可能就是上个星期，由那些在古墓丽影游戏中取得优秀成绩的经验丰富的青少年发明的。不管是哪一种来源，密码技术终归属于三种常规类型：隐藏、转换和置换。

隐藏密码是指在密码中包含了原文。它让接收器知道要剔除密码中的哪些字母或符号以产生原文。这里是一个隐藏密码的例子：

```
i2l32i5321k34el245ch456oc12ol234at567e
```

删除所有的数字，你将会得到 *I like chocolate*。这个呢：

```
Larry even appears very excited. No one worries.
```

摘取每一个单词的首字母，就得到 *leave now*。实际这两个都很简单，但是有更多花哨的主意来隐藏情报。有一个更加极端的例子，在公元前五世纪时，有一个名叫Histiaeus的人，他把自己信任的一个奴隶剃光了头，然后把情报刺在他的光头上。等到这个奴隶的头发长复原了以后，Histiaeus把他派到串通好的收报人Aristagoros那里去。Aristagoros 又把这个奴隶的头发剃光，得到了通知他起义的消息。

变换密码也在密码中保留有原文的字符。密码是简单地通过改变现有原文字符的顺序得到的。我们来试一个例子：

```
uo yn os dn ep ed yx al ag eh tf oy te fa se ht
```

将这些字符串联起来，再颠倒它们的顺序。你就会得到消息“the safety of the galaxy depends on you”。

置换密码是用另外的字母或者符号来代替原文中的每一个字符。考虑这个：

```
9-15-14-12-25-20-8-9-14-11-9-14-14-21-13-2-5-18-19
```

如果你用字母表中的相应字母来代替每一个数字，你将会破译这句话“I only think in numbers”。（例如，“I”是字母表中的第 9 个字母，“o”是第 15 个，等等）置换密码几乎能利用任何字符序列来完成加密和解密技术。这个应用程序中的两种密码技术都是置换密码。

关于破解密码

第一眼看去，这个应用程序所产生的密码可能显得非常复杂。事实上，任何正规的密码分析都能够仅用铅笔和纸就把密码分成许多微小的部分。幸运的是，使用诸如 RSA，IDEA 和三倍 DES（数据加密标准）这样的算法可以更好地确保安全性。我不方便告诉你怎样破解，但是我会给你一个提示来说明简单的置换和变换密码是多么容易受安全性攻击。

攻击这些密码类型的主要武器是字母频率分布图表。也就是说，一些字母在日常英语会话中比其他字母出现的频率高。在英语中，最常用的字母从多到少依次是E-T-N-R-O-A-I-S。最不常用的则是J，K，Q，X和Z（注1）。

另外一种破解简单密码的方法是分析两字母词和三字母词。两字母词就是一个两个字符的串，比如*ab*或者是*cd*。三字母词则是一个包含三个字母的串，比如*abc*或者*bcd*。两字母词和三字母词在英语中也有出现的频率高低之分。美国军方认为下面的两字母词是最常用的：*en*，*er*，*re*，*nt*，*th*，*on*和*in*。最不常用的是*df*，*hu*，*ia*，*lt*和*mp*。对于三字母词，最常用的是*ent*、*ion*，*and*，*ing*，*ive*，*tho*和*for*。最不常用的又是*eri*，*hir*，*iet*，*der*和*dre*。

最常用的字母，双字母词和三字母词不仅可以暗示可能有多少个字母，而且可以预示它们以及它们周围的字母很可能不会是什么。考虑一下你在日常会话中用到多少双字母词和三字母词：*is*，*be*，*am*，*or*，*not*，*are*，*yes*，*the*。如此列下去。即使这个应用程序中所用的密码不是最棒的，但它们仍然很有趣，并且是一个抵抗入侵的重要方法。

Caesar密码

古罗马的凯撒用这种密码来和他的军队首领沟通消息，它是已知最早用来为保密情报服务的密码之一。这里的算法是将字母表中的字母移1到25位（从b到z），这样，移动3位就使原文的字母*a*变成了密码中的字母*d*，反之亦然。移到*z*以后的字母又从头开始。换句话说，移动三位使原文的字母*z*变成密码中的*c*。对于发送和接收的当事人来说，用来加密和解密的密钥都是数字。

注意，一旦选定了一个数字，那么每一个字符都总是有与之相对应的同样的原文或者密码字符。例如，移动3位意味着原文的*a*总是密码中的*d*。也就是说，只有一个密码字母表。Caesar密码被认为是单字母表密码。

Vigenere密码

这种密码是由16世纪的数学家Blaise de Vigenere提出的。它是一个多字母表密码，

注1： 参见美国《Army Field Manual》34-40-2。

因为它使用了多个密码字母表。也就是说，原文的 a 并不像 Caesar 密码移动 3 位那样总是等于密码中的 *d*。

这种密码使用一个关键字来代替数字。假设你想要编制密码 *meet at midnight*，并且你选用了关键字 *vinegar*。那么关键字中的字母就会和原文中的字母一起连续排列，就像这样：

vine ga rvinegar
meet at midnight

好了。*V* 是字母表中的第 22 个字母，*I* 是第 9 个。字母 *n*、*e*、*g*、*a* 和 *r* 分别是第 14，5，7，1 和第 18 个字母。因此，原文中的字母 *m* 就移动 22 位，第一个 *e* 移动 9 位，第二个 *e* 移动 14 位，如此类推。你将得到的结果是：

hmrx gt ddlammhk

如果你想一下，其实这种密码就相当于是运行中产生的 Caesar 密码。在每一个字符上执行一个新的 Caesar 加密。

注意：如果想得到有关于密码的更多内容，可以在 http://www.und.nodak.edu/org/crypto/crypto/army.field.manual/separate.chaps/ 地址下载以前是“机密的”美国军方的多份 PDF 格式文档。

这个备份存储在 Crypto Drop Box 网站。你可以参见主页 http://www.und.nodak.edu/org/crypto/crypto/。在那里，可以发现够你忙好几天的资料。

执行条件

这个应用程序使用 JavaScript 1.2 和 DHTML，因此，只有 4.x 及以上版本的浏览器才能够运行。有许多的字符串匹配和置换要用到 JavaScript 1.2。

语法细则

幸运的是，这个应用程序只有两个文件。但更妙的是，你只消看它们其中一个文件

的代码。这两个文件是*index.html*和*dhtml.js*（*dhtml.js*在第六章中已经有所涉及）。在我们浏览代码之前，先来考虑有关这个应用程序看起来如何的几个抽象概念。这个应用程序是从一个非常基本的面向对象观点来构建的。第八章中包含了另一个利用对象定位的应用，但是加密应用在这方面将有更深入的讨论。

这个应用程序包含两种密码。每一种密码都有和其他所有密码所共有的东西，而不管它们是什么类型的密码。记住，有三种基本的密码——隐藏、变换和置换。这个应用程序包含两种置换密码：Caesar密码和Vigenere密码。图9-5展示了我们所描述的基本层次结构。

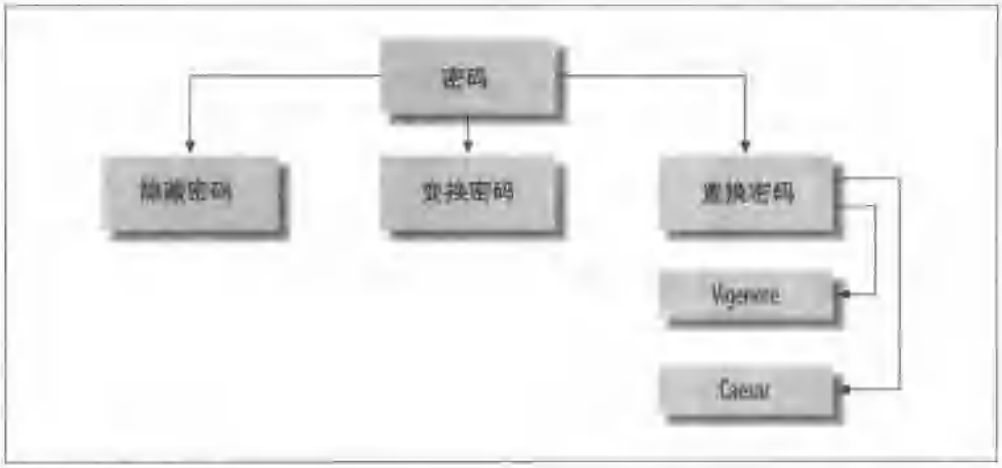


图9-5 密码结构

本图表示出*ConcealmentCipher*（隐藏密码），*TranspositionCipher*（变换密码）和*SubstitutionCipher*（置换密码）对象从*Cipher*对象继承所有的东西，有点儿像是一个子类。因此，Vigenére密码和Caesar密码是*SubstitutionCipher*对象的实例，并且包含它所有的属性和函数。

为了满足你的好奇心，让我们来看看怎样扩展这个模式。图9-6显示出将其他密码类型加到层次结构中而不需重新设计其实是多么简单。层次结构图中的粗体字表示用在这个应用程序中的部分。

正如你所看到的，许多密码类型和单独的密码都可以加到这个结构图中来，直到无穷，而不用改变当前已定位密码的任何现有代码。你还可以在子类上再生成子类。

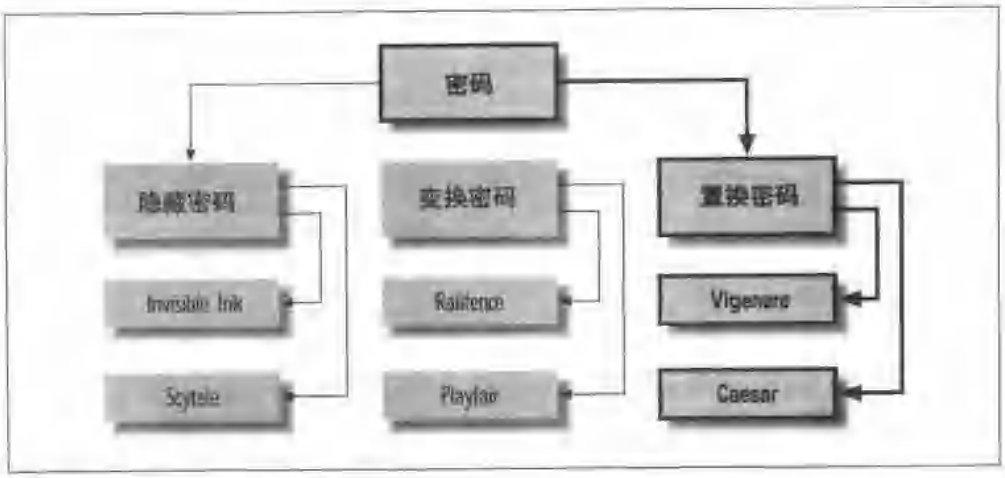


图 9-6 密码结构扩展

面向对象设计的优点再一次得到了体现。等我们浏览下面几页的支持代码时你可要记着这一点哦。你将会看到给应用程序添加更多的密码而不重组整个结构其实是多么简单。

让我们来看一看例 9-1 中的 *index.html*。

例 9-1：index.html

```
<HTML>
<HEAD>
  <TITLE>Cipher</TITLE>
<STYLE TYPE="text/css">
<!--
BODY { margin-left: 50 px; font-family: arial; }
I { font-weight: bold; }
//-->
</STYLE>
<SCRIPT LANGUAGE="JavaScript1.2" SRC="dhtml.js"></SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.2">
<!--

var caesar = '<FONT SIZE=2>Made famous by Julius Caesar, this cipher ' +
  'performs character shifting (substitution). Plaintext is ' +
  'enciphered by shifting forward each character of the alphabet a ' +
  'fixed number of characters.<BR><BR>For example, shifting by 1 ' +
  'changes plaintext <I>a</I> to <I>b</I>, <I>b</I> to <I>c</I>, ' +
  'and so on. Plaintext characters at the end of, say, the alphabet, ' +
  'are enciphered by starting at the beginning. In other words, ' +
  '<I>z</I> becomes <I>a</I>. This application also includes digits ' +
  '0-9. So a plaintext <I>z</I> becomes <I>0</I>, and a plaintext ' +
```

```
23    '<I>9</I> becomes <I>a</I>. The process is reversed for deciphering.' +
24    '<BR><FORM>Shift: ' +
25    genSelect('Shift', 35, 0, 0) +
26    '</FORM><BR>Note: Caesar was rumored to prefer a shift of 3.';
27
28  var vigenere = '<FONT SIZE=2>Made famous by mathematician Blaise de ' +
29    'Vigenere, the Vigenere cipher can be considered a "dynamic" ' +
30    'version of the Caesar cipher. Instead of shifting each plaintext ' +
31    'character by a fixed number, this cipher shifts characters ' +
32    'according to the character index of a keyword you choose such as ' +
33    '<I>dog</I>.<BR><BR>Since <I>d</I>, <I>o</I>, and <I>g</I> are ' +
34    'letters 4, 15, and 7 of the alphabet, each three plaintext ' +
35    'characters are shifted by 4, 15, and 7, respectively. This ' +
36    'application includes digits 0-9. So your keyword can have letters ' +
37    'and numbers.<BR><BR><FORM>Keyword: <INPUT TYPE=TEXT NAME="KeyWord" ' +
38    'SIZE=25></FORM><BR>Note: This cipher has many versions, one of ' +
39    'which was devised by Lewis Carroll, author of Alice in Wonderland.';
40
41  var curCipher = "caesar";
42
43  function Cipher() {
44    this.purify = purify;
45    this.chars = 'abcdefghijklmnopqrstuvwxyz0123456789';
46    }
47
48  function purify(rawText) {
49    if (!rawText) { return false; }
50    var cleanText = rawText.toLowerCase();
51    cleanText = cleanText.replace(/\s+/g,' ');
52    cleanText = cleanText.replace(/[^a-z0-9\s]/g,'');
53    if(cleanText.length == 0 || cleanText.match(/^\s+$/) != null) {
54      return false;
55      }
56    return cleanText
57    }
58
59  function SubstitutionCipher(name, description, algorithm) {
60    this.name = name;
61    this.description = description;
62    this.substitute = substitute;
63    this.algorithm = algorithm;
64    }
65  SubstitutionCipher.prototype = new Cipher;
66
67  function substitute(baseChar, shiftIdx, action) {
68    if (baseChar == ' ') { return baseChar; }
69    if(action) {
70      var shiftSum = shiftIdx + this.chars.indexOf(baseChar);
71      return (this.chars.charAt((shiftSum < this.chars.length) ?
72        shiftSum : (shiftSum % this.chars.length)));
73        }
74      else {
75        var shiftDiff = this.chars.indexOf(baseChar) - shiftIdx;
```

```
76        return (this.chars.charAt((shiftDiff < 0) ?
77          shiftDiff + this.chars.length : shiftDiff));
78        }
79      }
80
81  function caesarAlgorithm (data, action) {
82    data = this.purify(data);
83    if(!data) {
84      alert('No valid text to ' + (action ? 'cipher.' : 'decipher.'));
85      return false;
86      }
87    var shiftIdx =
88      (NN ? refSlide("caesar").document.forms[0].Shift.selectedIndex :
    document.forms[1].Shift.selectedIndex);
89    var cipherData = '';
90    for (var i = 0; i < data.length; i++) {
91      cipherData += this.substitute(data.charAt(i), shiftIdx, action);
92      }
93    return cipherData;
94    }
95
96  function vigenereAlgorithm (data, action) {
97    data = this.purify(data);
98    if(!data) {
99       alert('No valid text to ' + (action ? 'cipher.' : 'decipher.'));
100     return false;
101     }
102   var keyword = this.purify((NN ?
103     refSlide("vigenere").document.forms[0].KeyWord.value :
104     document.forms[2].KeyWord.value));
105   if(!keyword || keyword.match(/\^s+$/) != null) {
106     alert('No valid keyword for ' + (action ? 'ciphering.' :
107       'deciphering.'));
108     return false;
109     }
110   keyword = keyword.replace(/\s+/g, '');
111   var keywordIdx = 0;
112   var cipherData = '';
113   for (var i = 0; i < data.length; i++) {
114     shiftIdx = this.chars.indexOf(keyword.charAt(keywordIdx));
115     cipherData += this.substitute(data.charAt(i), shiftIdx, action);
116     keywordIdx = (keywordIdx == keyword.length - 1 ? 0 : keywordIdx + 1);
117     }
118   return cipherData;
119   }
120
121 var cipherArray = [
122   new SubstitutionCipher("caesar", caesar, caesarAlgorithm),
123   new SubstitutionCipher("vigenere", vigenere, vigenereAlgorithm)
124   ];
125
126 function showCipher(name) {
127   hideSlide(curCipher);
```

```
128   showSlide(name);
129   curCipher = name;
130   }
131
132 function routeCipher(cipherIdx, data, action) {
133   var response = cipherArray[cipherIdx].algorithm(data, action);
134   if(response) {
135     document.forms[0].Data.value = response;
136     }
137   }
138
139 //-->
140 </SCRIPT>
141 </HEAD>
142 <BODY BGCOLOR=#FFFFFF>
143
144 <DIV>
145   <TABLE BORDER=0>
146     <TR>
147       <TD ALIGN=CENTER COLSPAN=3>
148       <IMG SRC="images/cipher.jpg">
149       </TD>
150     </TR>
151     <TR>
152       <TD VALIGN=TOP WIDTH=350>
153       <FORM>
154       <SELECT NAME="Ciphers"
155         onChange="showCipher(this.options[this.selectedIndex].value);">
156       <OPTION VALUE="caesar">Caesar Cipher
157       <OPTION VALUE="vigenere">Vigenére Cipher
158       </SELECT>
159       </TD>
160       <TD ALIGN=CENTER>
161       <TEXTAREA NAME="Data" ROWS="15" COLS="40" WRAP="PHYSICAL"></TEXTAREA>
162       <BR><BR>
163       <INPUT TYPE=BUTTON VALUE="Encipher"
164         onClick="routeCipher(this.form.Ciphers.selectedIndex,
165           this.form.Data.value, true);">
166       <INPUT TYPE=BUTTON VALUE="Decipher"
167         onClick="routeCipher(this.form.Ciphers.selectedIndex,
168           this.form.Data.value, false);">
169       <INPUT TYPE=BUTTON VALUE="  Reset  "
170         onClick="this.form.Data.value='';">
171       </FORM>
172       </TD>
173     </TR>
174   </TABLE>
175 </DIV>
176
177 <SCRIPT LANGUAGE="JavaScript1.2">
178 <!--
179 document.forms[0].Ciphers.selectedIndex = 0;
```

```
180 genLayer("caesar", 50, 125, 350, 200, showName, caesar);
181 genLayer("vigenere", 50, 125, 350, 200, hideName, vigenere);
182 //-->
183 </SCRIPT>
184 </BODY>
185 </HTML>
```

JavaScript源文件 *dhtml.js* 是我们要讨论的第一部分代码。这个文件中的代码利用DHMTL来建立层，并在运行中产生选择列表。我们马上就会说到下一部分有趣的代码出现在第14～39行中。定义变量 *caesar* 和 *vigenere*，并将它们设置为HTML串值。你可能猜得到，它们每一个都定义一个密码界面层。除了在 *caesar* 的值中调用了函数 genSelect()之外，其他的东西都是静态的。如下所示：

```
genSelect('Shift', 35, 0, 0)
```

它创建了一个名为 *Shift* 的选择列表，从0开始，到35结束，其中选项0被选中。VALUE和TEXT属性被设置为用来计算的数字。这些代码直接从第五章中搬来。如果你像我在第六章中所建议的那样创建JavaScript库，你应该已经非常干脆地把这些代码包括进去了。你会发现在 *dhtml.js* 的最后部分定义了这个函数。

定义密码

接下来的几行代码定义所有已经有的和将要出现的密码。第43～46行中包含了Cipher()构造器：

```
function Cipher() {
  this.purify = purify;
  this.chars = 'abcdefghijklmnopqrstuvwxyz0123456789';
  }
```

这真是一个很小的构造器。如果你原本希望一些很大的，带有微分方程和立体几何的复杂定义来分解第四维度，那么对不起，我让你失望了。Cipher()在相当高的水平上定义密码。对这个应用程序中所有密码所做的唯一两个假设是：

- 它们都知道怎样格式化用户数据（使用它自己的方法），成为原文或者密码文字。
- 每一个密码都知道它将包括哪一个用来计算的字符（用它自己的属性）。

JavaScript技巧：给对象指派函数

Cipher()介绍了一个和它一样小的概念。也就是，我们在其他章节中所创建的对象只包含属性。Cipher()构造器有一个叫做*chars*的属性。同时还有一个名为purify()的函数。

属性很容易赋值。只消用语法this.variable_name将你所要的值赋给一个变量。但给函数赋值就有点儿麻烦了。你首先要定义一个函数，然后用同样的this.variable_name语法来引用函数。事实上，这就是Cipher()构造器中所发生的情况。脚本定义了函数purify()。Cipher()用一个名为*this.purify*的变量来引用purify函数。注意，这里是没有圆括号的。它确定一个引用。因为*this.purify*已经设置为purify()，那么purify()函数就已经被调用过了，而*this.purify*将被设置为函数返回的任何值。

在构造器内部引用一个函数，会将purify()赋值给任何设置为new Cipher()的变量。这就是cipherArray中的元素所发生的情况，你很快就会看到。

不管数据是否被加密或者解密，它都必须符合特定的规则。下面就是我们所说的规则：

- 每一个字符必须在a～z或者0～9之间。所有其他的字符都将被忽略。这种情况并不要紧。
- 不对空格进行加密或者解密。若干像邻的空格将被减少成单一的空格。
- 一个或多个换行符将被转换成单一的空格。

要注意这些规则，它们其实也很简单。我们所需的一切只不过是要有什么东西来执行它们。来看看第48～57行的函数purify()：

```
function purify(rawText) {
  if (!rawText) { return false; }
  var cleanText = rawText.toLowerCase();
  cleanText = cleanText.replace(/\s+/g,' ');
  cleanText = cleanText.replace(/[^a-z0-9\s]/g,'');
  if(cleanText.length == 0 || cleanText.match(/^\s+$/) != null) {
    return false;
    }
  return cleanText
  }
```

JavaScript 技巧：更多的字符串匹配和置换

你的确应该喜欢 JavaScript 1.2 中的正则表达式匹配。这个应用程序比前边的应用更多地用到它。让我们再来看看 52 行的正则表达式。

```
/[^a-z0-9\s]/g
```

尽管它并不长，但所用的语法却不简单。这个正则表达式被作为一个否定字符集。换句话说，任何不包含在这个定义中的东西都是匹配的。你可以在一个正则表达式中运用方括号来指定要包括（或者，在本例中是不包括的）的字符范围。考虑这个：

```
/[a-z]/g
```

这个表达式匹配字母表中的任何小写字母。g表示搜索在此范围内所有匹配的字符，在合格范围内你想要多少就可以包括多少，而不仅只是碰到的第一个。

```
/[a-z0-9\s]/g
```

这个表达式匹配字母表中的任何小写字母或者任何数字或是空格。但是，本例中的密码应用程序感兴趣的是不在这个范围内的任何东西。方括号中的抑扬符号（^）用来否定它后面的任何特定字符，产生了我们最初的语法。

```
/[^a-z0-9\s]/g
```

这是字符串匹配的顶尖表达。你可以用这个正则表达式来验证或者格式化社会安全数字，Email 地址，URL，电话号码，邮递区号，日期，时间，等等。如果你是首次接触正则表达式，在 http://developer1.netscape.com:80/docs/manuals/communicator/jsguide/regexp.htm 网址下“JavaScript 1.2 的新内容”中你可以获得足够多的正则表达式定义和特殊字符含义的参考信息。

这个函数返回这两个值中的一个：false，或者准备做加密的格式化文字。返回false将会取消任何加密动作，如果*rawText*包含任何要格式化的东西，purify()首先把所有的字母转换成小写形式。下面是它的具体做法：

```
cleanText = cleanText.replace(/\s+/g,' ');
```

使用正则表达式匹配，*String*对象的replace()函数搜索整个字符串中的所有空格，不管它们有多少个连在一起，都转换成单一的空格。然后，purify()用一个空字

符来代替其他所有不在a~z或0~9之间，也不是空格的字符。这将删除所有非法字符。下面再次给出责任重大的replace()函数的工作情况：

```
cleanText = cleanText.replace(/[^a-z0-9\s]/g,'');
```

格式化工作就结束了。现在应该来检查一下是否有什么有用的东西被留下来进行加密。只要格式化后的字符串至少包含一个a~z或0~9之间的字符，就万事大吉。但是，在这一条件不成立的时候有两种情况：

- 当所有的非法字符被删除以后，没有什么字符留下来。
- 当所有的非法字符被删除以后，只有空格留下来。

如果两种情况都不是，就该依次调用操作来产生更好的数据了。第53～55行的代码执行检查工作。如果什么都不产生，它就使purify()返回false：

```
if(cleanText.length == 0 || cleanText.match(/^\s+$/) != null) {
  return false;
  }
```

只要知道哪一个字符是合法的，Cipher就使用下面的字符串：

```
this.chars = 'abcdefghijklmnopqrstuvwxyz0123456789';
```

定义置换密码

现在，所有密码的母亲 —— Cipher() —— 已经定义好了，让我们来创建一个更明确的版本。这正是所有置换密码的规格：SubstituinCipher()。研究一下第59～65行的代码：

```
function SubstitutionCipher(name, description, algorithm) {
  this.name = name;
  this.description = description;
  this.substitute = substitute;
  this.algorithm = algorithm;
  }
SubstitutionCipher.prototype = new Cipher;
```

对每一个Cipher对象的假定是，它们都知道怎样来格式化任何用户数据。置换密码包含更多的假设，如下所示：

1. 每一个都有名称和描述。
2. 每一个在加密和解密中都用共同的方法来置换字符。
3. 对于共同的置换方法，每个对象都有一个特殊的执行。这就使一个置换密码不同于其他置换密码。
4. 每一个 *SubstitutionCipher* 对象也是一个 *Cipher* 对象。

给其中的每一个对象赋上一个名称和描述的值是很简单的。当你调用new SubstitutionCipher()时传递任何两个串都能很好地工作。顺带要说的是，前面所提到的变量*caesar*和*vigenere*和所有的HTML将作为每一个对象的描述。它们负责第一部分的假设。现在，定义一个普通的置换方法又如何呢？这个方法可以将一个字符置换为另外一个。就是这样。对此方法的每一个调用都返回一个字符，它是对另一个字符的替代品。

执行基本置换

每一个SubstitutionCipher都用相同的方法以另外一个字符来代替*chars*字符串中的一个字符。如下所示的函数substitute()被定义为每一个SubstitutionCipher示例的函数：

```
function substitute(baseChar, shiftIdx, action) {
  if (baseChar == ' ') { return baseChar; }
  if(action) {
    var shiftSum = shiftIdx + this.chars.indexOf(baseChar);
    return (this.chars.charAt((shiftSum < this.chars.length) ?
      shiftSum : (shiftSum % this.chars.length)));
    }
    else {
    var shiftDiff = this.chars.indexOf(baseChar) - shiftIdx;
    return (this.chars.charAt((shiftDiff < 0) ?
      shiftDiff + this.chars.length : shiftDiff));
    }
  }
```

这个函数需要三个参数。*baseChar* 是将被代替的字符。*shiftIdx* 是用来确定适当的位移量以发现正确置换的整数。*action* 是一个布尔值，它用来指定*baseChar*是否应该作为原文或者加密文字。为了保证空格不被修改，如果*baseChar*的确是一个空格的话，第一行代码就返回*baseChar*本身。否则，这个函数利用*action*来确定怎样计

算位移量。如果*action*是true，就使用加密算法。如果*action*为false，就使用解密算法。

记住，*chars*包含一个所有合法字符的串。加密算法只确定*chars*中*baseChar*的索引，然后将索引加上*shiftIdx*的值，按照这个结果来选择*chars*中的字符。

这里是一个例子。假设*baseChar*是d，*shiftIdx*为8，而chars.indexOf(`d`)为3。这样就得到第70行的结果：

```
var shiftSum = shiftIdx + this.chars.indexOf(baseChar);
```

变量*shiftSum*等于11 (8 + 3)。因此chars.charAt(11)就是字母l。这也就是本例中substitute()将返回的结果。这是很直观的。但如果假设*baseChar*是字母o，而*shiftIdx*为30。现在看到math. *shiftSum*等于45。问题是，*chars*只有36个字符(a～z和0～9)。因此，chars.charAt(45)根本不存在。

当算法到达chars中的最后一个字符的时候，必须重新回到0，然后从那里开始重新计数。你可以通过取模运算获得想要的结果。想一想这个问题：取模运算返回的是两个操作数的余数。这里是几个例子：

4 % 3 = 1。4除以3余数为1。
5 % 3 = 2。5除以3余数为2。
6 % 3 = 0。6除以3没有余数。

你所需做的是利用取模运算所返回的数。因此，你将使用的是shiftSum % chars.length，它等于9，而不用等于45的shiftSum。chars.charAt(9)是字母j。这是接下来用于加密运算的代码：

```
return (this.chars.charAt((shiftSum < this.chars.length) ? shiftSum :
  (shiftSum % this.chars.length)));
```

在本例中，substitute()返回chars.charAt(shiftSum)或者chars.charAt(shiftSum % this.chars.length)，这依赖于*sbiftSum*的大小和*chars*的长度来决定。关键字*this*又怎么样呢？你可能会捉摸不透它在这究竟干什么。要记住，substitute()并不是一个函数；它是一个方法，是实例化为SubstitutionCipher变量的方法。在此方法中使用*this*，可以引用已实例化变量的任何属性。由于SubstitutionCipher继承了Cipher的所有属性，实例化变量“拥有”一个叫*chars*的属性。

这个过程和解密运算并没有太多的不同。唯一的变化是它将减去*shiftIdx*来到达正确的*chars*中的字符。在本例中，变量*shiftDiff*的设置和*baseChar*和*shiftIdx*的索引不同，如下所示。

```
var shiftDiff = this.chars.indexOf(baseChar) - shiftIdx;
```

这仍然十分简单。如果*shiftDiff*小于0，你也要进入和*shiftSum*大于chars.length-1时一样的情况。解决的方法是在chars.length上加*shiftDiff*。这是对的……把它加上。*shiftDiff*是负值就意味着将两个数加在一起产生的数*shiftDiff*将小于chars.length，这也就是解密所需的索引。下面的代码指出substitute()是否使用*shiftDiff*或者*shiftDiff* + chars.length来作为解密的索引：

```
return (this.chars.charAt((shiftDiff < 0) ?
  shiftDiff + this.chars.length : shiftDiff));
```

对不同密码的不同置换

我们刚才查看了*SubstitutionCiphers*所拥有的共同东西——substitute()函数。现在，让我们来看一看究竟是什么把它们分开的。*SubstitutionCiphers*构造器要求一个名为*algorithm*的参数。这个参数并不是一个字符串，一个逻辑值，一个数字，或者是一个对象。这个参数指向一个函数，而这个函数将会以特殊的方法执行（调用）substitute()函数。

对Caesar密码来说，这个传递进来的参数指向函数caesarAlgorithm()。当然，Vigenere密码中所接收进来的将指向函数vigenereAlgorthm()。让我们来看看每一个函数。

Caesar算法

Caesar算法是较先出现的一个。第81～94行中包含的代码是：

```
function caesarAlgorithm (data, action) {
  data = this.purify(data);
  if(!data) {
    alert('No valid text to ' + (action ? 'cipher.' : 'decipher.'));
    return false;
    }
  var shiftIdx =
  (NN ? refSlide("caesar").document.forms[0].Shift.selectedIndex :
```

```
    document.forms[1].Shift.selectedIndex);
  var cipherData = '';
  for (var i = 0; i < data.length; i++) {
    cipherData += this.substitute(data.charAt(i), shiftIdx, action);
    }
  return cipherData;
  }
```

前边的几行负责将数据作格式化处理，然后查看是否有任何合法字符被留下来。参数 *data* 中的字符串通过调用 purify() 并传递参数 *data* 完成格式化处理。只要对 purify() 的调用返回的不是 false，密码处理就继续。看看 purify() 方法前边的部分，可以得到关于返回值的具体细节。

下面要做的事是确定用户要用来作位移的字符数字。这可真是很简单。它来自 *caesar* 层表单中的选择列表。我还没有提到过有关于这一点的任何情况，但如果你想看看用来创建两个层的函数调用，你可以跳到前边的第 180～181 行去。但是，Navigator DOM 和 IE DOM 在访问不同层的表单元素方面有所不同。选择列表的名称是 *Shift*。

在 Navigator 中，看起来是这样的：

```
document.layers['caesar'].document.forms[0].Shift.selectedIndex
```

但在 IE 中又是这个样子：

```
document.forms[1].Shift.selectedIndex
```

注意：正如你所看到的，访问层中的表单和表单元素需要不同的语法。Navigator 中的文档对象样式和 IE 里的不同。在这本书中我们可不是第一次说到这个。实际上，*dhtml.js* 中的大部分代码只在两种浏览器中创建和操作层的时候才存在。你自己随便吧。要确保你在必须适应这两者或者不必都适合的情况下知道这一点。在我们看到一个统一的 DOM 之前，记住下面的便利资源。

Microsoft 的 DHTML 对象：

http://www.microsoft.com/workshop/author/dhtml/reference/objects.asp

Netcape 的样式表参考和客户端 JavaScript 参考：

http://developer1.netscape.com:80/docs/manuals/communicator/dynhtml/jss34.htm 和 http://developer.netscape.com/docs/manuals/js/client/jsref/index.htm

---

变量*shiftIdx*利用*NN*变量来确定访问两个中的哪一个，以此来表现不同之处。第88行对refSlide()的调用是用来得到document.layers["caesar"]的简单方法。现在，*shiftIdx*已经得到了赋值，caesarAlgorithm()循环data.length次，每一次都调用substitute()，并且将返回值连接到开始为空的局部变量*cipherData*。每一次都传参数action，用来给substitute()正确指出是应该加密还是解密。最后一次循环完成以后，caesarAlgorithm()返回*cipherData*，它现在包含的是密码处理之后的字符串。

Vigenere 算法

这是我们所说的两中密码运算中比较简单的一种。我们来看看vigenereAlgorithm()。在这里，主要的区别是传递给caesarAlgorithm()中substitute()的参数*shiftIdx*，它保存的是常量。通过这个函数，*shiftIdx*可以（通常都可以）随着每一次对substitute()的调用而改变。另外一个不同之处在于，用户选择一个关键词来代替一个数字。这里是第96～119行的代码：

```
function vigenereAlgorithm (data, action) {
  data = this.purify(data);
  if(!data) {
    alert('No valid text to ' + (action ? 'cipher.' : 'decipher.'));
    return false;
    }
  var keyword =
    this.purify((NN ?
      refSlide("vigenere").document.forms[0].KeyWord.value :
      document.forms[2].KeyWord.value));
  if(!keyword || keyword.match(/\^s+$/) != null) {
    alert('No valid keyword for ' +
     (action ? 'ciphering.' : 'deciphering.'));
    return false;
    }
  keyword = keyword.replace(/\s+/g, '');
  var keywordIdx = 0;
  var cipherData = '';
  for (var i = 0; i < data.length; i++) {
    shiftIdx = this.chars.indexOf(keyword.charAt(keywordIdx));
    cipherData += this.substitute(data.charAt(i), shiftIdx, action);
    keywordIdx = (keywordIdx == keyword.length - 1 ? 0 : keywordIdx + 1);
    }
  return cipherData;
  }
```

开始的五行代码和caesarAlgorithm()中的相同。它们也同样进行格式化和确认。

下面几行对关键词执行相似的工作。关键词来自位于*vigenere*层的表单域。记住我们必须能同时适应Navigator和IE DOMs。

在Navigator中，它将是这样的：

```
document.layers['vigenere'].document.forms[0].KeyWord.value
```

但在IE中，它又是这样：

```
document.forms[2].KeyWord.value
```

然后变量*keyword*得到如下所示的赋值：

```
var keyword = this.purify((NN ?
  refSlide("vigenere").document.forms[0].KeyWord.value :
  document.forms[2].KeyWord.value));
```

注意，这里又一次用到了purify()函数。它是用来对原文和加密文字服务的，但是对关键词的要求很相似。由于substitute()函数只能置换*chars*中的字符，关键词也就必须包含chars中的字符。可用的关键词包括*people*，*machines*，*init2wnit*和*1or2or3*。但是，同样可以使用不在*chars*中的字符。记得purify()删除了所有不在a～z或者0～9中的字符，并且用单个的空格来取代换行符，回车符和多个连接空格。尽管用户可能会输入1@@#derft来作为一个关键词，但是purify()会将这个串格式化，返回*1derft*，只包含合法的字符。现在，考虑带空格的关键词，包括中间所有的空格。除了那些空格外，它还包含合法字符。第110行将它们删除。

```
keyword = keyword.replace(/\s+/g, '');
```

最后一行是：只要关键词中至少存在一个合法字符，那它就是将要用在vigenere-Algorithm()中的。

shiftIdx如何改变

原文(或者加密文字)和关键词都已经被格式化过了。因此，剩下要做的只是置换每一个字符了。用Vigenere的定义，每一个文本字符按照关键词中下一个字符的索引来加密或者解密。这样我们就进入了第111～118行：

```
var keywordIdx = 0;
var cipherData = '';
```

```
for (var i = 0; i < data.length; i++) {
  shiftIdx = this.chars.indexOf(keyword.charAt(keywordIdx));
  cipherData += this.substitute(data.charAt(i), shiftIdx, action);
  keywordIdx = (keywordIdx == keyword.length - 1 ? 0 : keywordIdx + 1);
  }
return cipherData;
```

使用初始值为0的*keywordIdx*，我们可以得到如下所示的每一个关键词字符的索引：

```
keyword.charAt(keywordIdx)
```

对于*data*（原文或加密文字）的每一个字符，*shiftIdx* 被设置为 keyword.charAt(keywordIdx) 中*chars*的索引。然后将变量*cipherData*设为它本身加上 substitute() 函数的返回值，substitute() 接收的是data.charA(i)的一个新的拷贝和*shiftIdx*，连同 action 一起。然后将 *keywordIdx* 增加 1 来开始下一轮循环。

每一个 SubstitutionCipher 也是一个密码

不管它们是什么类型的密码，所有的密码都必须有相同的基本特征，*SubstitutionCipher* 构造器必须继承 *Cipher* 的所有属性。这发生在一行代码中：

```
SubstitutionCipher.prototype = new Cipher;
```

现在，每一个示例 *SubstitutionCipher* 对象都有一个叫做 *chars* 的属性和一个叫做 purify() 的函数。然后每个 *SubstitutionCipher* 都是一个 *Cipher* 的特殊版本。

JavaScript 技巧：进入 JavaScript 对象继承

如同在最后一章中所给出的，JavaScript 使用基于原型的继承，而不是如 Java 之类的语言中常见的基于对象的继承。第八章中的“JavaScript 技巧：添加对象属性”向你展示了怎样添加诸如字符串或数字的属性到现有的对象中去。你也可以利用构造函数的原型属性来创建继承层次。这相应于第 65 行代码的内容。*SubstitutionCipher* 继承了 *Cipher* 的所有属性。这让你实现了面向对象设计的真正威力（在 JavaScript 的相关范围内）。你可以在 Netscape's DevEdge 在线中得到关于 JavaScript 继承的更多内容：

http://developer1.netscape.com:80/docs/manuals/communicator/jsobj/contents.htm#1030750

到目前为止，我们已经看到了这两个密码是如何运作的。现在，应该来探讨一下如何创建表示两个密码的对象，以及怎样构造它们的使用界面。创建对象只花了四行代码。它们在第121～124行：

```
var cipherArray = [
    new SubstitutionCipher("caesar", caesar, caesarAlgorithm),
    new SubstitutionCipher("vigenere", vigenere, vigenereAlgorithm)
    ];
```

变量*cipherArray*设置为一个数组。其中每一个元素都是*SubstitutionCipher*。为什么把它们放在一个数组里呢？原因在于应用程序知道按照页面中第一个选择列表的中选OPTION应该使用哪一个密码。我们马上就要对此作讨论。

JavaScript技巧：使用替换语法

在JavaScript 1.2中，你可以这样来替换代码：

```
var myArray = new Array(1,2,3);
```

附带一个如下所示的短小版本：

```
var myArray = [1,2,3];
```

你也可以像下面这样在运行中创建对象。替换这个：

```
function myObj() {
  this.name="A New Object";
  this.description = "Old School Object";
  }

var objOne = new myObj();
```

试着换为：

```
var myObj = {name: "A New Object", description: "New School Object"};
```

注意属性和方法名——值对是用逗号分隔开的。IE的4.x版本和Navigator都支持这个形式。你自己研究一下吧。

现在，注意每一个对SubstitutionCipher()的调用都传递适当的字符串，名称和描述，还有对函数的指示——它将被赋值为所创建的每个*SubstitutionCipher*对象的algorithm属性。这就创建了对象。我们来看一看界面。它位于BODY标签中：

```
<DIV>
  <TABLE BORDER=0>
    <TR>
      <TD ALIGN=CENTER COLSPAN=3>
      <IMG SRC="images/cipher.jpg">
      </TD>
    </TR>
    <TR>
      <TD VALIGN=TOP WIDTH=350>
      <FORM>
      <SELECT NAME="Ciphers"
        onChange="showCipher(this.options[this.selectedIndex].value);">
      <OPTION VALUE="caesar">Caesar Cipher
      <OPTION VALUE="vigenere">Vigenére Cipher
      </SELECT>
      </TD>
      <TD ALIGN=CENTER>
      <TEXTAREA NAME="Data" ROWS="15" COLS="40"
        WRAP="PHYSICAL"></TEXTAREA>
      <BR><BR>
      <INPUT TYPE=BUTTON VALUE="Encipher"
        onClick="routeCipher(this.form.Ciphers.selectedIndex,
        this.form.Data.value, true);">
      <INPUT TYPE=BUTTON VALUE="Decipher"
        onClick="routeCipher(this.form.Ciphers.selectedIndex,
        this.form.Data.value, false);">
      <INPUT TYPE=BUTTON VALUE="  Reset  "
        onClick="this.form.Data.value='';">
      </FORM>
      </TD>
    </TR>
  </TABLE>
</DIV>
```

这些代码创建了一个两行的表格。上边的一行在TD中包含图表，COLSPAN的值设为2。下面的一行包含两个数据单元。左边的一个包括一个单纯的选择列表，如下所示：

```
<SELECT NAME="Ciphers"
  onChange="showCipher(this.options[this.selectedIndex].value);">
<OPTION VALUE="caesar">Caesar Cipher
<OPTION VALUE="vigenere">Vigenére Cipher
</SELECT>
```

这个列表用来确定究竟要显示哪一个密码界面。由于只有两个，它要么是这个要么是另外一个。*onChange*事件处理方法调用showCipher()函数，传递当前所选项的值。这个函数非常短。你会在第126～130行看到它：

```
function showCipher(name) {
  hideSlide(curCipher);
  showSlide(name);
```

```
curCipher = name;
}
```

里边的代码看起来似乎有点眼熟。好像在前边的章节中有类似的东西，比如第三章或者是第六章。你会在*dhtml.js*中看到函数hideSlide()和showSlide()。查阅第三章可以获得具体的内容。

注意，数据单元宽度被设置为350个像素。和选择列表相比，数据单元真是太空了。幸运的是，两个层将会填在可用浏览器空间中。你可以看到第180～181行用来创建它们的调用。函数genLayer()创建密码层，它也在*dhtml.js*中。它同样是一个前边提到过现在又使用到的函数：

```
genLayer("caesar", 50, 125, 350, 200, showName, caesar);
genLayer("vigenere", 50, 125, 350, 200, hideName, vigenere);
```

这将创建为每个密码显示的文字，连同Caesar密码的选择列表和Vigenére密码的文本域。正如刚才所说，你可以在上部的选择列表中修改Caesar密码到Vigenére密码之间的选项，然后显示恰当的密码层。

至于下面一行中的另外一个数据单元，它包含一个文本域和三个按钮。这里再一次给出它们位于第161～170行的代码：

```
<TEXTAREA NAME="Data" ROWS="15" COLS="40" WRAP="PHYSICAL"></TEXTAREA>
<BR><BR>
<INPUT TYPE=BUTTON VALUE="Encipher"
  onClick="routeCipher(this.form.Ciphers.selectedIndex,
  this.form.Data.value, true);">
<INPUT TYPE=BUTTON VALUE="Decipher"
  onClick="routeCipher(this.form.Ciphers.selectedIndex,
  this.form.Data.value, false);">
<INPUT TYPE=BUTTON VALUE="  Reset  " onClick="this.form.Data.value='';">
```

textarea域保存原始文本（或者加密文字）。“Encipher”按钮使其中所包含的文字得到加密。它和“Decipher”按钮正好相反。它们都调用相同的函数routeCipher()。两个都传递textarea域的值。不同之处在于最后一个参数，一个是true，而另一个是false。

选择正确的密码

选择正确的密码其实很简单。正确的密码总是和表单上部的选择列表的指数和cipherArray的指数相对应。你可以在routeCipher()中看到，如下所示：

```
function routeCipher(cipherIdx, data, action) {
  var response = cipherArray[cipherIdx].algorithm(data, action);
  if(response) {
    document.forms[0].Data.value = response;
    }
  }
```

这个函数接收三个参数。我们已经讨论过后面两个了。*data*是文本域中的文字，而*action*是true或者false。第一个，*cipherIdx*，来自document.forms[0].Ciphers.selectedIndex。它必须是0或者是1。不管是哪一个，都会调用与cipherArray中*SubstitutionCipher*对象相对应的algorithm()函数。如果algorithm()返回一个非false的值，那么它必定是合格的加密（或解密）文本。

最后的问题

到现在为止，我们差不多已经理解了，但是第179行的代码：

```
document.forms[0].Ciphers.selectedIndex = 0;
```

仅仅重新将上部选择列表的选中OPTION置为第一个。这迫使当选OPTION和当前显示的密码层相匹配，即使用户重载页面。

应用程序扩展

虽然这个应用程序用起来已经很酷了，但它的下一个目标是用在Email上。你可以通过三个步骤实现这个目标。首先，复制下面的函数，然后粘贴到你的SCRIPT标签下：

```
function sendText(data) {
  paraWidth = 70;
  var iterate = parseInt(data.length / paraWidth);
  var border = '\n-------\n';
  var breakData = '';
  for (var i = 1; i <= iterate; i++) {
```

```
    breakData += data.substring((i - 1) * paraWidth, i * paraWidth) +
      '\r';
    }
  breakData += data.substring((i - 1) * paraWidth, data.length);
  document.CipherMail.Message.value = border + breakData + border;
  document.CipherMail.action =
    "mailto:someone@somewhere.com\?subject=The Secret Message";
  return true;
  }
```

它在发送Email之前执行瞬间的格式化工作。格式化将在每一个*paraWidth*字符后插入回车。这确保收信人得到的Email不会有40英里长的文本行。下面要做的是添加所需的第二个表单。在当前文档的FORM结束符之后插入这些代码：

```
FORM NAME="CipherMail" ACTION="" METHOD="POST" ENCTYPE="text/plain"
  onSubmit="return sendText(document.forms[0].Data.value);">
<INPUT TYPE=HIDDEN NAME="Message">
<INPUT TYPE=SUBMIT VALUE="   Send   ">
</FORM>
```

这个名为CipherMail的表单包含一个单独的HIDDEN域。最后，要在密码运算函数中修改表单的引用。

修改第87～89行：

```
var shiftIdx = (NN ?
  refSlide("caesar").document.forms[0].Shift.selectedIndex :
  document.forms[1].Shift.selectedIndex);
```

变成：

```
var shiftIdx = (NN ?
  refSlide("caesar").document.forms[0].Shift.selectedIndex :
  document.forms[2].Shift.selectedIndex);
```

然后把第102～104行从：

```
var keyword = this.purify((NN ?
  refSlide("vigenere").document.forms[0].KeyWord.value :
  document.forms[2].KeyWord.value));
```

变成：

```
var keyword = this.purify((NN ?
  refSlide("vigenere").document.forms[0].KeyWord.value :
document.forms[3].KeyWord.value));
```

你需要作这些修改，因为你在前边步骤的层次中添加了另一个表单。sendText()将隐藏域的值赋为文本域中所输入的任何值。然后 sendText()提交这个表单，它的ACTION属性设为mailto:your_e-mail@your_mail_server.com。图9-7显示出等它送到的时候信息将是什么模样的。这是从我的Hotmail来估计的。在收到的信息上，用户可以剪切和粘贴虚线之间的文字，然后用以前协商好的密码和密钥来作解密工作。现在你的访问者正在看解密后的邮件，你可真是幕后的天才啊！

图 9-7　解密后的邮件

附：这只当用户有正确设定的MM或IE的Email客户时才起作用，就像本例中的一样。

再附：*\ch09\cipher2.html* 中有附加的Email功能。

第十章

网络贺卡：拖放 Email

建立这个应用程序只是为了多一些趣味性。用户可以在你的站点上打发很多时光，他们可以把贺卡发送给好朋友和情人 —— 都是些狂野的背景和幼稚的文字，构成他们所要表达的信息。图10-1给出了打开的界面。

应用程序要点

- 发送精巧的自定义贺卡
- 用户可在背景选择上定位多个图标
- 友好的用户界面
- 细致的消息测试

JavaScript技巧

- 区分Web代码
- 定义结构通信
- 优化你的函数

左边是一个空的表单。用户在其中填写必要的成分，包括收信人地址，信息，以及贺卡。在这里用户也能选择背景，点击“Backgrounds - ->”，直到他们看到所满意的一个。图标也是这样。“Icons - ->”起相同的作用。

右半部分是用来显示的。在这里，用户能看见可用的背景和图标。然后用户可以“抓起”一个图标，将它拖到背景显示区域中。任何这样的图标都包括在如图10-2所示的贺卡中，图中给出的是一个很好的例子。

贺卡做完以后，用户可以通过选择“Test”来预览制作结果。这将打开一个子窗口，展示当收信人浏览这个贺卡的时候它会是什么样子。请看图 10-3。

如果用户感到满意了，就选择“Send”。它将把所有表单信息提交给正准备着的服务器端脚本，它创建贺卡，并且返回最终确认的包含“Send”按钮的页面，如图10-4所示。如果点中按钮，脚本就将Email发送给收信人，并提供贺卡的URL。

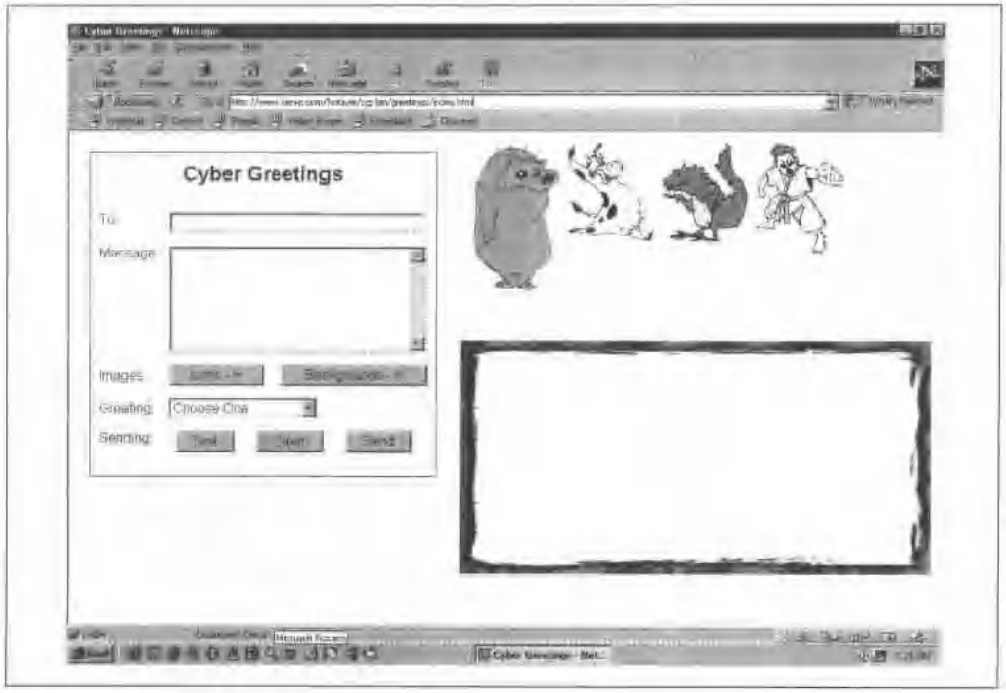


图 10-1　电子贺卡的默认画面

图 10-2　在这一组照片中你认识谁吗？

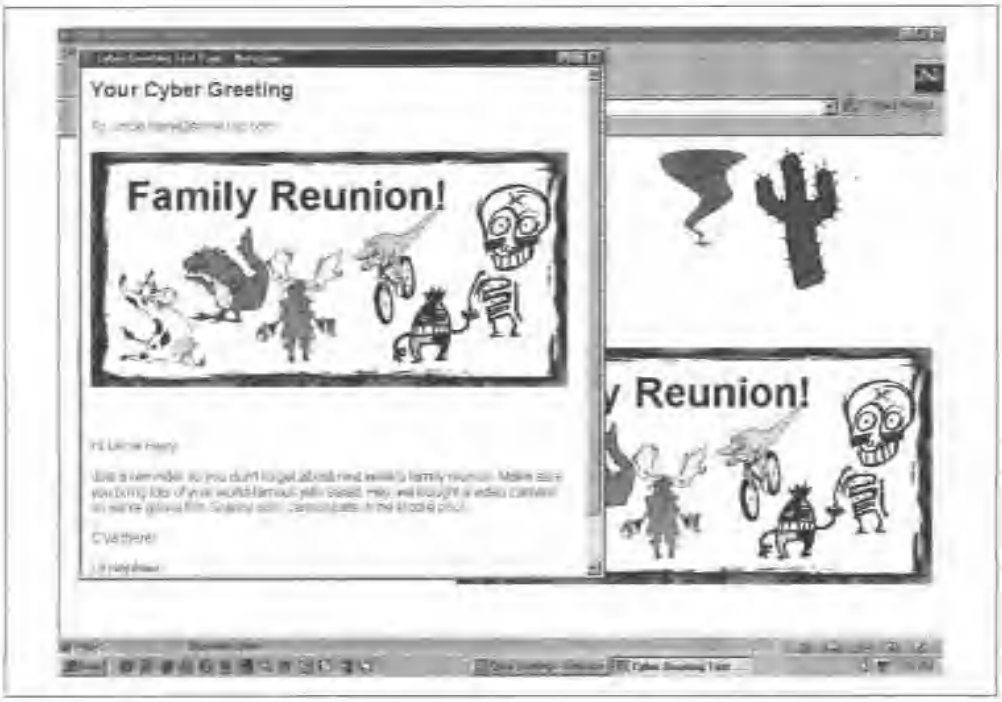


图 10-3 收信人将看到的结果

图 10-4 成功：已经提交表单，完成了消息传输

现在，用户该来读一读邮件了。(我没有什么Hairy叔叔，因此我将这一份贺卡送给我的Hotmail列表。) 图 10-5 显示出一个发布的贺卡是什么样子的。它没有多少东

西，只是一个简单的消息和链接。只要收件人点击 Email 消息中的链接，载入的文档就恰好是发件人所创建的。其流程如图 10-6 所示。

图 10-5　电子贺卡发布

图 10-6　电子贺卡流程

执行条件

需要使用 IE 或者 Navigator 4.x 及以上版本，因为有 DHTML 和庞大的样式表的存在。这个程序要求至少 1024 × 768 的分辨率，但是你可以做适当的修改来适应 800 × 600 的设置。我可不想再低了。

这个程序也需要一个Web服务器，带有服务器端脚本环境。如果你对服务器端脚本不熟悉，也不要害怕。我已经给出了一个脚本，它可以非常容易地装入地球上几乎所有的Web服务器中。它是用Perl写的。你只需将它拷贝到正确的目录下，并设立一些许可就行了。你可以在附录三，使用Perl脚本中获得有关细节内容。

语法细则

这是另外一个在我们看代码之前要先研究一下其流程图的应用程序。图10-7显示出用户如何输入收件人Email地址和消息，选择一个贺卡和背景，然后将所需的图标定位到贺卡卡片上面。接下来，用户预览工作结果，如果满意，就发送到服务器，等等。

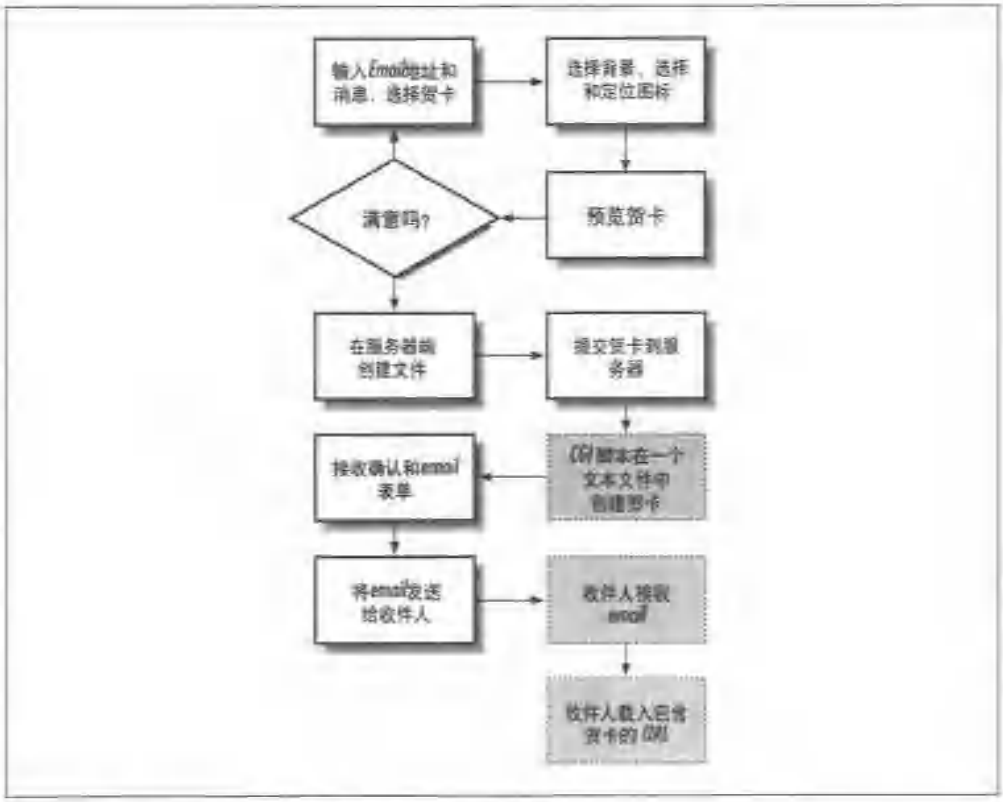


图 10-7 电子贺卡流程：收件人如何获得消息

这个应用程序在两个层次上运行——在客户浏览器上和在Web服务器上。很明显，用户是在浏览器中创建整个的贺卡——背景，图标，以及消息。当用户提交HTML表单，信息就发送回Web服务器，在那里创建一个匹配贺卡的文件。Web服务器返回一个HTML表单，这样，贺卡发送者就可以发送一个Email消息给收件人。这个消息只包括一个电子贺卡发布和一个收件人必须使用的链接。收件人点击这个链接，载入对应的电子贺卡。

让我们先以客户端来看一看这个应用程序，然后再研究服务器端的内容。共有四个文件。下面的列表给出了一个纲要：

index.html

顶层文件；包含框架设置。

back.html

包含用来选择贺卡，背景和图标的工作区。

front.html

包含创建和发送消息的界面

greet.pl

服务器端脚本，用来在一个文件中创建和存储贺卡，然后创建一个HTML表单来将Email发送给收件人

正如你在屏幕上所看到的，这个界面有两个部分。后一个部分（*back.html*）是贺卡显示，用户在此能看到他在选择贺卡和设定所有图片之后所拥有的东西。前一部分（*front.html*）包含登陆表单，并且负责Email地址和消息的输入，选择背景，以及产生可用图形（比如图标）的工作。两个文档都在*index.html*中引用。具体代码如例10-1所示。

例10-1：index.html

```
<HTML>
<HEAD>
   <TITLE>Cyber Greetings</TITLE>
<SCRIPT LANGUAGE="JavaScript1.2">
<!--

var greetings = [
   'Choose One', 'Family Reunion!',
   'Get Well Soon','Thinking Of You',
   'Big Party!', 'Psst... You\'re Invited.',
```

```
11     'Happy Birthday!', 'Congratulations!',
12     'We\'re Gonna Miss U', 'Just A Reminder',
13     'Don\'t Forget'
14     ];
15
16 var baseURL = ".";
17
18 //-->
19 </SCRIPT>
20 </HEAD>
21 <FRAMESET COLS="450,*" FRAMEBORDER="2" BORDER="0">
22    <FRAME SRC="front.html" NAME="Front" NORESIZE>
23    <FRAME SRC="back.html" NAME="Back" NORESIZE SCROLLING="NO">
24 </FRAMESET>
25 </HTML>
```

index.html 中第 7～14 行包含一个称为 *greetings* 的数组。用户可以在其中选择自己喜欢的某种贺卡。*baseURL* 包含这个应用程序在服务器上的基本目录。所有的东西都包括在里头：所有四个文件，所有图片，还有将包含每一个用户创建的贺卡的目录。*baseURL* 甚至包括在贺卡本身中。当你改变这个值的时候，你就对整个应用程序的相关内容作了改变——客户端和服务器。

这样的话，为什么要在这个文件中定义一个变量和一个数组呢？两个定义在 *index.html* 框架中的文件都需要这些贺卡，以在载入的时候创建所需的页面。如果贺卡被定义在其他两个文件的某一个中，在另外一个文件里的 JavaScript 代码试图访问它的时候，它就可能不会被载入。对于 *baseURL* 也是如此。这有助于避免载入时间错误。

另外两个文档

前一部分和后一部分的概念和传统的明信片类似。前边部分（我认为）上边有地址和消息，而后面部分有海滩风景的精美图片。在本例中，*back.html* 包含背景显示区域和供你拖曳的图标。这个文件在文档载入期间负责绝大部分的初始设置工作。*front.html* 掌管在文档载入后的事宜，比如输入消息，选择和发送一个贺卡。那么，先来讨论 *back.html* 是有意义的。幸运的是，你可能在前边的章节中已经见过它的大部分内容了。看一看例 10-2。

例 10-2：back.html

```
1 <HTML>
2 <HEAD>
3   <TITLE>Cyber Greetings</TITLE>
```

```
<STYLE TYPE="text/css">
<!--

.Greeting
  {
    font-family: Arial;
    font-size: 48px;
    font-weight: bold;
  }

//-->
</STYLE>
<SCRIPT LANGUAGE="JavaScript1.2">
<!--

var NN       = (document.layers ? true : false);
var hideName = (NN ? 'hide' : 'hidden');
var showName = (NN ? 'show' : 'visible');
var zIdx     = -1;

var iconNum  = 4;
var startWdh = 25;
var imgIdx   = 0;
var activate = false;
var activeLayer = null;

var backImgs = [];
var icons = [
    'bear', 'cowprod', 'dragon', 'judo',
    'robot', 'seniorexec', 'dude', 'juicemoose',
    'logo1', 'logo2', 'logo3','tree',
    'sun', 'gator', 'tornado', 'cactus'
    ];

function genLayout() {

  for (var i = 0; i <= 7; i++) {
    backImgs[i] = new Image();
    backImgs[i].src = top.baseURL + '/images/background' + i +
      '.jpg';
    }

  genLayer("Back", 10, 250, backImgs[1].width, backImgs[1].height,
    showName, '<IMG NAME="background" SRC="' + top.baseURL +
    '/images/background0.jpg">');

  for (var j = 0; j < parent.greetings.length; j++) {
    genLayer("greeting" + j, 50, 275, 500, 100, hideName,
      '<SPAN CLASS="Greeting">' + parent.greetings[j] + '</SPAN>');
    }

  for (var i = 0; i < icons.length; i++) {
    if (i % iconNum == 0) { startWdh = 25; }
```

```
57      else { startWdh += 110; }
58      genLayer(icons[i], startWdh, 15, 100, 100, (i < iconNum ? showName :
59        hideName), '<A HREF="javascript: changeAction(\'' + icons[i] +
60        '\',' + (i + 1) + ');"><IMG SRC="' + top.baseURL +
61        '/images/' + icons[i] + '.gif" BORDER="0"></A>');
62      }
63    startWdh = 25;
64    }
65
66  function genLayer(sName, sLeft, sTop, sWdh, sHgt, sVis, copy) {
67    if (NN) {
68      document.writeln('<LAYER NAME="' + sName + '" LEFT=' + sLeft +
69      ' TOP=' + sTop + ' WIDTH=' + sWdh + ' HEIGHT=' + sHgt +
70      ' VISIBILITY="' + sVis + '" z-Index=' + (++zIdx) + '>' + copy +
71      '</LAYER>');
72      }
73    else {
74      document.writeln('<DIV ID="' + sName +
75        '" STYLE="position:absolute; overflow:none; left:' +
76        sLeft + 'px; top:' + sTop + 'px; width:' + sWdh + 'px; height:' +
77        sHgt + 'px; visibility:' + sVis + '; z-Index=' + (++zIdx) + '">' +
78        copy + '</DIV>'
79        );
80      }
81    }
82
83  function hideSlide(name) {
84    refSlide(name).visibility = hideName;
85    }
86
87  function showSlide(name) {
88    refSlide(name).visibility = showName;
89    }
90
91  function refSlide(name) {
92    if (NN) { return document.layers[name]; }
93    else { return eval('document.all.' + name + '.style'); }
94    }
95
96  function motionListener() {
97    if (NN) {
98      window.captureEvents(Event.MOUSEMOVE);
99      window.onmousemove = grabXY;
100     }
101   else {
102     document.onmousemove = grabXY;
103     }
104   }
105
106 function grabXY(ev) {
107   if (activate) {
108     if(NN) {
109       var itemWdh = refSlide(activeLayer).document.images[0].width;
```

```
110       var itemHgt = refSlide(activeLayer).document.images[0].height;
111       refSlide(activeLayer).left = ev.pageX - parseInt(itemWdh / 2);
112       refSlide(activeLayer).top = ev.pageY - parseInt(itemHgt / 2);
113       }
114     else {
115       var itemWdh = document.images[imgIdx].width;
116       var itemHgt = document.images[imgIdx].height;
117       refSlide(activeLayer).left = event.x - parseInt(itemWdh / 2);
118       refSlide(activeLayer).top = event.y - parseInt(itemHgt / 2);
119       }
120     }
121   }
122
123 function changeAction(name, MSIERef) {
124   activate = !activate;
125   activeLayer = name;
126   imgIdx = MSIERef;
127   }
128
129 //-->
130 </SCRIPT>
131 </HEAD>
132 <BODY onLoad="motionListener();">
133
134 <SCRIPT LANGUAGE="JavaScript1.2">
135 <!--
136
137 genLayout();
138
139 //-->
140 </SCRIPT>
141
142 </BODY>
143 </HTML>
```

在发送者能够创建一个贺卡之前，要用几个函数来产生许多的层，并且确定和文档有关的鼠标箭头的位置。这里有和第三章和第八章中相似的功能。实际上，有好几个函数直接从这些章节中来。我会在讨论的时候提到它们。现在，我们来看看前边的第19～36行中所定义的变量：

```
var NN    = (document.layers ? true : false);
var hideName = (NN ? 'hide' : 'hidden');
var showName = (NN ? 'show' : 'visible');
var zIdx     = -1;

var iconNum  = 4;
var startWdh = 25;
var imgIdx   = 0;
var activate = false;
var activeLayer = null;
```

```
var backImgs = [];
var icons = [
    'bear', 'cowprod', 'dragon', 'judo',
    'robot', 'seniorexec', 'dude', 'juicemoose',
    'logo1', 'logo2', 'logo3','tree',
    'sun', 'gator', 'tornado', 'cactus'
    ];
```

前四个变量在前边的脚本中使用过。*NN* 用来确定使用的是哪一种浏览器。*showName*和*hideName*是分别用来显示和隐藏层的不同的串，它们还依赖于所使用的浏览器。而*zIdx*是用来为创建的每一个层设置*z*坐标的整数。变量*iconNum*用来确定一次在屏幕上显示多少个图标的整数。我们从4开始。*startWdh*用来在开始的时候定位所有的图标。你马上会在函数genLayout()中看到有关内容。

*imgIdx*被用来跟踪图片。*activate*是一个布尔变量，它确定一个层是应该被拖动还是被放置。*activeLayer*确定用户在点击后将鼠标箭头从哪一个层上拖过。

如果这还不够的话，那么还有两个数组变量。*backImags*的初始值为一个空数组。它很快就会被用*Image*对象来生成，每一个元素包含一个背景图片。背景图片被命名为*background0.jpg*，*background1.jpg*，*background2.jpg*，如此类推。

*icons*是一个字符串数组，它确定每一个图标的名称。这表示每一个图标将会被创建在和图标元素同名的一个层上。用作图标的图片也是同样的名称。例如，名为*bear*的层将包含图片*bear.gif*。顺便说一句，所有的图标图片都是透明的GIF格式。透明色为白色。由于背景图片基本上是白色的，你可以把图标放在另外一个图标上，但仍然能看到背景，而不会出现一个图标遮住另外一个的情况。

一些熟悉的内容

如果你已经看过这本书的其他一些章节，你会很高兴地知道，你的一些艰苦的工作并没有白费。这里所用的一些函数已经在很多地方使用过，因此你对付它们可以不费吹灰之力。这在另一章中也有类似情况，但这里犹为显著。

表10-1有助你确定哪些函数是你曾经见过的。

表 10-1 层操作函数

函数	用途	所出现的章
genLayer()	创建 Navigator 或 IE 中的层	3, 4, 6, 9, 11
hideSlide()	通过名称隐藏层	3, 4, 6, 9, 11
showSlide()	通过名称显示层	3, 4, 6, 9, 11
refSlide()	通过名称引用层	3, 4, 6, 9, 11
motionListener()	跟踪鼠标动作	11
grabXY()	获取元素的 x，y 坐标	11

表中的前边四个函数和其他章中的完全一样。如果你对这些函数不熟悉，可以看看第三章中的相关讨论。但 motionListener() 稍微有一些变化，使它有必要再说一说。函数 grabXY() 可以在第十一章中找到。它也经过了很大的变化。这里是函数余下的部分，执行一定的操作。

各就各位！

载入应用程序的时候，*back.html* 坚持不懈地预载所有的图片，创建并定位所有的层，然后根据需要显示或隐藏它们。函数 genLayout() 调整有关的每一点内容。在第 38～64 行我们可以看到各个东西是怎样就位的：

```
function genLayout() {
  for (var i = 0; i <= 7; i++) {
    backImgs[i] = new Image();
    backImgs[i].src = top.baseURL + '/images/background' + i +
      '.jpg';
    }

  genLayer("Back", 10, 250, backImgs[1].width, backImgs[1].height,
    showName, '<IMG NAME="background" SRC="' + top.baseURL +
    '/images/background0.jpg">');

  for (var j = 0; j < parent.greetings.length; j++) {
    genLayer("greeting" + j, 50, 275, 500, 100, hideName,
      '<SPAN CLASS="Greeting">' + parent.greetings[j] + '</SPAN>');
    }

  for (var i = 0; i < icons.length; i++) {
    if (i % iconNum == 0) { startWdh = 25; }
    else { startWdh += 110; }
    genLayer(icons[i], startWdh, 15, 100, 100, (i < iconNum ? showName :
```

```
        hideName), '<A HREF="javascript: changeAction(\'' + icons[i] +
        '\',' + (i + 1) + ');"><IMG SRC="' + top.baseURL +
        '/images/' + icons[i] + '.gif" BORDER="0"></A>');
      }
    startWdh - 25;
    }
```

genLayout()所处理的第一件事是预载背景图片。用户很可能希望在做选择之前看到所有的图片，因此，预载可是一个不错的注意哦。使用*backImgs*，函数为每一个元素创建一个*Image*对象，并用top.baseURL（在前边的*index.html*中定义过，还记得吗?），*background*字符串，*i*的值和字符串*.jpg*来为它的来源赋值。

```
for (var i = 0; i <= 7; i++) {
  backImgs[i] = new Image();
  backImgs[i].src = top.baseURL + '/images/background' + i + '.jpg';  }
```

所有的图片都载入以后，要着手的第一件事是默认背景。你可以选择任何一个，但为了简单，我选择的是*background0.jpg*，并将它放到一个名为*Back*的层中。将层的宽度和高度设置为背景图片的大小。这在后面定位图标的时候是很重要的：

```
genLayer("Back", 10, 250, backImgs[1].width, backImgs[1].height,
  showName, '<IMG NAME="background" SRC="' + top.baseURL +
  '/images/background0.jpg">');
```

现在，背景层和默认值都设置好了。下面要做的是创建贺卡。它们其实就是简单的层，附带巨大的文字，比如“Family Reunion”或者“Thinking Of You”。这些贺卡来自*index.html*的*greetings*数组。我们看看第50～53行的代码：

```
for (var j = 0; j < parent.greetings.length; j++) {
  genLayer("greeting" + j, 50, 275, 500, 100, hideName,
    '<SPAN CLASS="Greeting">' + parent.greetings[j] + '</SPAN>');
  }
```

这意味着总共有parent.greetings.length那么多的贺卡，它们的左边和上边坐标分别是50和275。用户不能移动它们，但它们已经被精确地安排在了背景显示区域的左上部。每一个贺卡都包含在它自己的层中。层包括一系列的SPAN标签，来使用样式表类名为*Greeting*的定义，定义位于文档的前边部分。

背景和贺卡都就位以后，剩下的所有任务就是图标的定位了。看一看第55～62行：

```
for (var i = 0; i < icons.length; i++) {
  if (i % iconNum == 0) { startWdh = 25; }
  else { startWdh += 110; }
  genLayer(icons[i], startWdh, 15, 100, 100, (i < iconNum ? showName :
```

```
    hideName), '<A HREF="javascript: changeAction(\'' + icons[i] +
    '\',' + (i + 1) + ');"><IMG SRC="' + top.baseURL +
    '/images/' + icons[i] + '.gif" BORDER="0"></A>');
}
```

icons 中的每一个元素表示一个图标层。变量 *iconNum* 规定一次显示四个图标。并且，每一个图片的宽度为 100 像素，高度不定。变量 *startWdh* 从 25 开始。它的值将确定创建的每一个层的左边缘像素值。我随便将四个图标的间距选为 10 个像素宽。换句话说，从左边空白的右边缘的 25 个像素开始，每 110 个像素（100 个像素是层的宽度，10 个像素是图标间距）放置一个图标。待 *iconNum* 个图标都被创建和定位以后，进程从同样的 25 像素的控制点重新开始。有两个设计特征来支持这个过程。一个是在用 genLayer() 创建每一个层之前执行的 if-else 语句；另外一个是模数运算（%）的使用。来看看这两个内容：

```
if (i % iconNum == 0) { startWdh = 25; }
else { startWdh += 110; }
```

在 *for* 循环的执行过程中，*i* 变得越来越大。每当 *i* 是 *iconNum*（本例中为 4）的倍数时，就该从新的一组图标开始了，将组中的第一个图标又定位在距左边缘 25 个像素的位置。将 *startWdh* 的值设置为 25。例如，当 *i* 到达 4，8，12，16 和 20 的时候就要重新开始。如果 *i* 是任何其他值，就表示下一个图标将会在距上一个图标左边缘 110 个像素的位置显示。这就是将 110 加到 *startWdh* 的当前值上的原因。模数运算返回的整数表示某个商的余数。如果余数是 0，*i* 就是 *iconNum* 的倍数。

知道将层放置在什么位置是一个比较难的部分。现在，函数 genLayout() 将通过数次调用 genLayer()，为每个图标创建一个层，完成这部分的工作：

```
genLayer(icons[i], startWdh, 15, 100, 100, (i < iconNum ? showName :
  hideName), '<A HREF="javascript: changeAction(\'' + icons[i] + '\',' +
  (i + 1) + ');"><IMG SRC="' + top.baseURL + '/images/' +
  icons[i] + '.gif" BORDER="0"></A>');
```

每一个图标层包含一个独立的 IMG 标签，它在一个锚标签中。注意，传递给 genLayer() 的第二和第三个参数是层的左边和上边属性。*startWdh* 总是表示左边；上边固定为 15 个像素。传递进来的第六个参数确定图标是否应该显示或者隐藏。默认值是仅显示创建的第一组图标。在本例中，也就是最开始的四个层。因此，第六个参数中的条件运算就是，如果 *i* 小于 *iconNum*（例如 0，1，2 或 3），层就是可见的。其他所有的层都被隐藏。如果图标可见，变量 *showName* 就被传递进来。否则，将调用 *hideName*。

在进入下一部分内容之前最后要考虑的是锚标签的自然状态。它究竟是用来做什么的？考虑这个。不管用户第一次把鼠标从图标上移过时点击的是什么，他都明显是想拾起这个图标，把它拖到什么地方去。为了完成这一动作，HREF 属性中的 javascript:协议调用了函数 changeAction()，我们马上就要说到这个函数。所有的HREF属性都同样调用这个函数，但是每一个图标链接必须将它自己的特殊信息传递给 changeAction()。

首先，changeAction()需要知道所要处理的图标的名称。这是很简单的。将icons[i]传递进来，它包含有正确的字符串。接下来，我们需要传递进一个整数，它表示IE文档对象模式中的图标。也就是说，为了使这个拖放功能可以顺利运行，IE需要知道它要的究竟是哪一个图片。记住，页面上创建的第一个图片是背景图片，因此它就是document.images[0]。第一个图标是document.images[1]。其他所有图标为document.images[i + 1]。这就是传递(i + 1)的原因。等我们看changeAction()和 grabXY()时，情况就显而易见了。

对于一个只有 27 行的函数，我们做的说明真够多的。现在，我们将 *startWdh* 设置为 25，然后继续以后的内容。

跟踪鼠标位置

motionListener()使 JavaScript 能利用 *onmousemove* 事件处理方法来捕获鼠标动作。它非常容易构建。两种浏览器之间唯一的区别在于Navigator需要调用captureEvents()函数，它从窗口执行 *mousemove* 事件。IE 从文档中做相同动作。这里是第 96～104 行的代码：

```
function motionListener() {
  if (NN) {
    window.captureEvents(Event.MOUSEMOVE);
    window.onmousemove = grabXY;
    }
  else {
    document.onmousemove = grabXY;
    }
  }
```

不管用户什么时候移动鼠标，都调用函数grabXY()。记住，调用函数不需加圆括号。*onmousemove* 引用 grabXY，而不是调用它。*onLoad* 事件在第 132 行调用 motionListener()函数。它只被调用一次，因此，鼠标跟踪只在应用程序持续时间内发生。

调用所有图标

用户点击一个图标的时候，对 changeAction()的调用本质上使图标被激活，使它可以被用来传输。第 123～127 行包括其中的细节内容：

```
function changeAction(name, IERef) {
  activate = !activate;
  activeLayer = name;
  imgIdx = IERef;
  }
```

还记得变量 *activate* 和 *activeLayer* 吗？它们很早以前是在文档的开头部分定义的。*activate* 的初始值是 false，这表示 —— 就如我们很快就会在 grabXY()中看到的一样 —— 并不按照鼠标的动作来重新定位任何层。第一次调用 changeAction()的时候，*activate* 变为 true，让 grabXY()开始运作。层会跟着鼠标箭头移动。只有再次点击才会结束这种状况。这次，全局变量 *activate* 变成了它的相反值，也就是 false。拖动就停止了。

还记得 changeaction()是怎样被设计来传递两个参数的吗？一个是要执行动作的层的名称，赋值为 *name*。另外一个是用来在 IE 下正确引用 document.images 数组中图片的图片索引。这个值被赋为 *IERef*。activeLayer 被设置为 *name* 的值，而把 *imgIdx* 设置为 *IERef* 的值。这就是我们拖动这些图标时所需要的，不管使用的是哪一种浏览器。

移动图标

每一次用户移动鼠标都调用 grabXY()。你可以在第 106～121 行看到发生的情况：

```
function grabXY(ev) {
  if (activate) {
    if(NN) {
      var itemWdh = refSlide(activeLayer).document.images[0].width;
      var itemHgt = refSlide(activeLayer).document.images[0].height;
      refSlide(activeLayer).left = ev.pageX - parseInt(itemWdh / 2);
      refSlide(activeLayer).top = ev.pageY - parseInt(itemHgt / 2);
      }
    else {
      var itemWdh = document.images[imgIdx].width;
      var itemHgt = document.images[imgIdx].height;
      refSlide(activeLayer).left = event.x - parseInt(itemWdh / 2);
      refSlide(activeLayer).top = event.y - parseInt(itemHgt / 2);
```

```
        }
      }
    }
```

grabXY()被调用，但是函数中任何代码运行的唯一条件是变量*activate*是否为true。用户首次点击一个图标链接时，*activate*就变成了true。嵌套的*if-else*语句运行。如果用户使用的是Navigator，就执行*if*语句部分。否则，执行*else*部分。它们两者都执行相同的函数，但是针对特殊的浏览器。

两个部分的代码都定义局部变量*itemWdh*和*itemHgt*。这将确定点中图标的左边和上边位置。因此，为什么不将它们设置为当前鼠标箭头位置的坐标呢？毕竟，这是我们本来用于跟踪鼠标运动的数据。

你可以这样做，但是存在一个问题。如果你这样做了，就意味着鼠标箭头在拖动期间将会在图标的左上角。这看起来有点儿古怪，但更糟的是，用户可能会把鼠标移动得太快，以至脱离了拖曳动作，点击在图标以外的其他地方。用户可能不得不点击两次来“释放“图标。

解决的方法是将图标定位，这样，鼠标箭头就总是在图标的中心位置上。不管使用哪一种浏览器，*itemWdh*和*itemHgt*都分别表示用户所点击的图标的宽度和高度。你必须依照浏览器而使这些值有所不同。注意一下它们的不同之处。Navigator会让它像这样：

```
var itemWdh = refSlide(activeLayer).document.images[0].width;
var itemHgt = refSlide(activeLayer).document.images[0].height;
```

为了得到图片，你必须引用对应的层，然后是文档，然后是images[0]（它是层中唯一的图片）。它和IE中是不同的：

```
var itemWdh = document.images[imgIdx].width;
var itemHgt = document.images[imgIdx].height;
```

IE没有层数组。你可以直接从*images*数组中访问到图片。但是你需要知道正确的图片，它是由*imgIdx*来索引的。记住，我们每一次都调用changeAction()来对此作设置。如果不太清楚，就回顾一下这个函数。

现在我们有一个正确图片的宽度和高度的句柄。我们需要的所有东西是一些快速计算，以便把鼠标箭头定位到图标的中心。

让我们通过例子来理解这一过程。假设用户点击了某个图标并且开始拖动。在点击的时刻，鼠标箭头在文档左边缘的右边 100 个像素和文档（不是屏幕）上边缘的下边 100 个像素的位置。让我们来假设图标的高和宽分别是 150 和 100 个像素。如果我们将图标的 left 和 top 属性设置为 100，100，就将图标的左上角直接放到了鼠标箭头下。但是注意，我们希望将鼠标箭头置于图标的中心。

你可以通过从左边位置中减去图片宽度的一半，从上边位置中减去图片高度的一般来到达图标的中心。我们知道，*itemWdh* 是 100，*itemHgt* 是 150。这里是新位置的分解：

```
Icon left = pointer arrow horizontal (x) location-(100/2)=100-(50)=50
Icon top = pointer arrow vertical (y) location-(150/2)=100-(75)=25
```

我们没有将 *left* 和 *top* 设置为 100，100，它们被设置成了 50，25。这将指示箭头放到了图标的中心。要确信，这两个分割给出了一个整数。我们将使用 parseInt() 来返回整数部分。在看一看 grabXY() 中的代码。这里是 Navigator 中这一过程的执行情况：

```
refSlide(activeLayer).left = ev.pageX - parseInt(itemWdh / 2);
refSlide(activeLayer).top = ev.pageY - parseInt(itemHgt / 2);
```

Navigator 中的事件模式使用一个在运行中创建的事件对象，在这里用局部变量 *ev* 来指示。这个事件对象的 *pageX* 和 *pageY* 属性包含活动层的 *x,y* 坐标值。另一方面，IE 有一个全局事件对象，从中可以访问坐标：

```
refSlide(activeLayer).left = event.x - parseInt(itemWdh / 2);
refSlide(activeLayer).top = event.y - parseInt(itemHgt / 2);
```

属性 *x* 和 *y* 包含同样意义的值。现在，图标可以被随心所欲地拖动和放置了。

文档载入之后

真正的动作开始于 *front.html*。例 10-3 给出了它的代码。大约前边的几十行包含样式表属性。下面的两百行定义 JavaScript 变量和函数，负责获取创建，测试和最后发送贺卡的所有信息。

例 10-3：front.html

```
1 <HTML>
2 <HEAD>
```

```
<TITLE></TITLE>
<STYLE TYPE="text/css">
<!--

TD
  {
    font-family: Arial;
  }

.Front
  {
    position: absolute;
    left: 25;
    top: 25;
    width: 325;
    border: 1px solid;
    background: #ffffee;
  }

//-->
</STYLE>
<SCRIPT LANGUAGE="JavaScript1.2">
<!--


var curGreet = iconIdx = 0;
var backgroundIdx = 0;
var bRef = parent.Back

function showGreeting(selIdx) {
  if (selIdx > 0) {
    bRef.hideSlide("greeting" + curGreet);
    bRef.showSlide("greeting" + selIdx);
    curGreet = selIdx;
    }
  }

function nextBackground() {
  backgroundIdx = (backgroundIdx == bRef.backImgs.length - 1 ?
    backgroundIdx = 0 : backgroundIdx + 1);
  if(document.all) {
    bRef.document.background.src = bRef.backImgs[backgroundIdx].src;
    }
  else {
    bRef.document.layers["Back"].document.images[0].src =
      bRef.backImgs[backgroundIdx].src;
    }
  }

function nextIcons() {
  for (var i = bRef.iconNum * iconIdx; i < (bRef.iconNum * iconIdx) +
    bRef.iconNum; i++) {
    if (i < bRef.icons.length && !onCard(i)) {
      bRef.hideSlide(bRef.icons[i]);
```

```
57        }
58      }
59    iconIdx = (iconIdx >= (bRef.icons.length / bRef.iconNum)  - 1 ? 0 :
60      iconIdx + 1);
61    for (var i = bRef.iconNum * iconIdx; i < (bRef.iconNum * iconIdx) +
62      bRef.iconNum; i++) {
63      if (i < bRef.icons.length) {
64        bRef.showSlide(bRef.icons[i]);
65        }
66      else { break; }
67      }
68    }
69
70  function resetForm() {
71    if (document.all) {
72      bRef.hideSlide("greeting" +
73        document.EntryForm.Greetings.selectedIndex);
74      document.EntryForm.reset();
75      }
76    else {
77      bRef.hideSlide("greeting" +
78      document.layers["SetupForm"].document.EntryForm.Greetings.selectedIndex);
79      document.layers["SetupForm"].document.EntryForm.reset();
80      }
81    }
82
83  function onCard(iconRef) {
84    var ref   = bRef.refSlide(bRef.icons[iconRef]);
85    var ref2 = bRef.refSlide("Back");
86    if(document.all) {
87      if((parseInt(ref.left) >= parseInt(ref2.left)) &&
88        (parseInt(ref.top) >= parseInt(ref2.top)) &&
89        (parseInt(ref.left) + parseInt(ref.width) <= parseInt(ref2.left) +
90          parseInt(ref2.width)) &&
91        (parseInt(ref.top) + parseInt(ref.height) <= parseInt(ref2.top) +
92        parseInt(ref2.height))) {
93        return true;
94        }
95      }
96    else {
97      if((ref.left >= ref2.left) &&
98        (ref.top >= ref2.top) &&
99        (ref.left + ref.document.images[0].width <=  ref2.left +
100       ref2.document.images[0].width) &&
101       (ref.top + ref.document.images[0].height <= ref2.top +
102       ref2.document.images[0].height)) {
103       return true;
104       }
105     }
106   ref.left = ((iconRef % bRef.iconNum) * 110) + bRef.startWdh;
107   ref.top = 15;
108   return false;
109   }
```

```
110
111 function shipGreeting(fObj) {
112 if (fObj.Recipient.value == "") {
113     alert('You need an email address in the To: field');
114     return false;
115     }
116   else if (fObj.Message.value == "") {
117     alert("You need to type a Message.");
118     return false;
119     }
120   else if (fObj.Greetings.selectedIndex == 0) {
121     alert('You need to choose a Greeting.');
122     return false;
123     }
124
125   fObj.EntireMessage.value = genGreeting(fObj);
126
127   fObj.BaseURL.value = top.baseURL;
128   return true;
129   }
130
131 function testGreeting(fObj) {
132   var msgStr = '<HTML><TITLE>Cyber Greeting Test Page</TITLE>' +
133     genGreeting(fObj) + '<TABLE ALIGN="CENTER"><TR><TD><FORM>' +
134     '<INPUT TYPE=BUTTON VALUE="     OK     " onClick="self.close();">' +
135     '</FORM></TD></TR></TABLE></HTML>';
136   newWin =  open('', '', 'width=' + (
137     bRef.backImgs[backgroundIdx].width + 50) +
138     ',height=600,scrollbars=yes');
139   with(newWin.document) {
140     open();
141     writeln(msgStr);
142     close();
143     }
144   newWin.focus();
145   }
146
147 function genGreeting(fObj) {
148   var greetingIdx = fObj.Greetings.selectedIndex;
149   var msg = fObj.Message.value;
150
151   msg = msg.replace(/\r+/g, "");
152   msg = msg.replace(/\n+/g, "<BR><BR>");
153
154   var msgStr = '<TABLE BORDER=0><TR><TD COLSPAN=2><FONT FACE=Arial>' +
155     '<H2>Your Cyber Greeting</H2>To: ' + fObj.Recipient.value +
156     '<BR><BR></TD></TR><TR><TD VALIGN=TOP><IMG SRC="' +
157     top.baseURL + '/images/background' + backgroundIdx + '.jpg">' +
158     '<DIV STYLE="position:relative;left:40;top:-255;font-family:Arial;' +
159     'font-size:48px;font-weight:bold;">' + parent.greetings[greetingIdx] +
160     '</DIV>';
161
162   var iconStr = '';
```

```
163   for (var i = 0; i < bRef.icons.length; i++) {
164     if(onCard[i]) {
165       iconStr += '<DIV STYLE="position:absolute;left:' +
166       bRef.refSlide(bRef.icons[i]).left + ';top:' +
167       (parseInt(bRef.refSlide(bRef.icons[i]).top) -
168       (document.all ? 140 : 150)) + ';"><IMG SRC="' +
169       top.baseURL + '/images/' +  bRef.icons[i] + '.gif"></DIV>';
170       }
171     }
172
173   msgStr += iconStr + '</TD></TR><TR><TD WIDTH=' +
174     bRef.backImgs[backgroundIdx].width + '><FONT FACE=Arial>' + msg +
175     '</TD></TR></TABLE>';
176   return msgStr;
177   }
178
179 //-->
180 </SCRIPT>
181
182 </HEAD>
183 <BODY onLoad="resetForm();">
184 <DIV ID="SetupForm" CLASS="Front">
185 <FORM NAME="EntryForm"
186   ACTION="http://www.your_domain.com/cgi-bin/greetings/greet.pl"
187   METHOD="POST" TARGET="_top" OnSubmit="return shipGreeting(this);">
188 <INPUT TYPE=HIDDEN NAME="EntireMessage">
189
190 <INPUT TYPE=HIDDEN NAME="BaseURL">
191   <TABLE CELLSPACING="0" CELLPADDING="5" WIDTH="375">
192     <TR>
193       <TD COLSPAN="3"><CENTER><H2>Cyber Greetings</H2></CENTER></TD>
194     </TR>
195     <TR>
196       <TD HEIGHT="40" VALIGN="TOP">
197       To:
198       </TD>
199       <TD COLSPAN="2" VALIGN="TOP">
200       <INPUT TYPE=TEXT NAME="Recipient" SIZE="25">
201       </TD>
202     </TR>
203     <TR>
204       <TD HEIGHT="80" VALIGN="TOP">Message: </TD>
205       <TD COLSPAN="2" VALIGN="TOP">
206       <TEXTAREA ROWS="7" COLS="25" NAME="Message" WRAP="PHYSICAL">
207         </TEXTAREA>
208       </TD>
209     </TR>
210     <TR>
211       <TD>Images:</TD>
212       <TD HEIGHT="40" COLSPAN="2">
213       <INPUT TYPE=BUTTON VALUE="  Icons - >  " onClick="nextIcons();">
214          
215       <INPUT TYPE=BUTTON VALUE="Backgrounds - >"
216         onClick="nextBackground();">
```

```
217       </TD>
218     </TR>
219     <TR>
220       <TD>Greeting:</TD>
221       <TD HEIGHT="40" COLSPAN="2">
222       <SCRIPT LANGUAGE="JavaScript1.2">
223       <!--
224
225       var sel = '<SELECT NAME="Greetings"
226         onChange="showGreeting(this.selectedIndex);">';
227       for (var i = 0; i < parent.greetings.length; i++) {
228         sel += '<OPTION>' + parent.greetings[i];
229         }
230       sel += '</SELECT>';
231       document.writeln(sel);
232
233       //-->
234       </SCRIPT>
235       </TD>
236     </TR>
237     <TR>
238       <TD VALIGN=TOP>Sending: </TD>
239       <TD HEIGHT="40" ALIGN="CENTER">
240       <INPUT TYPE=BUTTON VALUE="  Test  "
241         onClick="testGreeting(this.form);">
242           
243       <INPUT TYPE=BUTTON VALUE="  Clear  " onClick="resetForm();">
244           
245       <INPUT TYPE=SUBMIT VALUE="  Send  ">
246       </FORM>
247       </TD>
248     </TR>
249   </TABLE>
250 </FORM>
251 </DIV>
252 </BODY>
253 </HTML>
```

变量介绍

front.html 不包含和 *back.html* 中一样多的变量，这里是第 28～30 行给出的变量：

```
var curGreet = iconIdx = 0;
var backgroundIdx = 0;
var bRef = parent.Back;
```

变量 *curGeet* 表示贺卡选择列表的索引。它的初始值设为 0。*iconIdx* 是一个用来通过索引跟踪图标的变量。它的初始值也是 0。列表中最后一个变量是 *bRef*，它只是 *Back* 框架中脚本和窗口对象的一个引用。这可以使事情更简单一点。

JavaScript 技巧：区分 Web 代码

这一章中的应用程序和其他章中的大部分不同，实际上，它有大量麻烦的HTML编码。我建议你让不同的代码看起来有明显的区别。例如，在所有客户端设计中，我的HTML通常都是大写字母，而在JavaScript中就不大用大写字母。任何人随便看上一眼就可以发现这两种代码的不同，但这种做法使差别更明显了。

这听起来好像没什么。但我是从一个喜欢使用 ColdFusion Markup Language (CFML，一种流行的服务器端脚本语言）的程序员那里学到这个经验的。他的所有代码包括HTML，CFML，JavaScript和用来做数据库访问的SQL。同一个脚本中就有四种语言。要是你在使用ASP，就会用到VBScript，JavaScript，JScript和SQL。你还需要多少缩写词？

当然了，我很快找到了自己的策略。

显示贺卡

既然贺卡的选择列表已经都设置好了，用户可以通过加亮当前选项来显示贺卡选择了。选择列表中的 *onChange* 事件处理调用 showGreeting()，如下所示：

```
function showGreeting(selIdx) {
  if (selIdx > 0) {
    bRef.hideSlide("greeting" + curGreet);
    bRef.showSlide("greeting" + selIdx);
    curGreet = selIdx;
    }
  }
```

下面一个值得回顾的部分出现在第225～231行中。在浏览器解析HTML的时候，这里执行的JavaScript用 *index.html* 中所定义的 *greetings* 数组创建一个选择列表。我们来仔细看一看：

```
var sel = '<SELECT NAME="Greetings" ' +
  'onChange="showGreeting(this.selectedIndex);">';
for (var i = 0; i < parent.greetings.length; i++) {
  sel += '<OPTION>' + parent.greetings[i];
 }
sel += '</SELECT>';
document.writeln(sel);
```

JavaScript 技巧：跨框架通信

你可能记得在第一章中，我用一个叫 *docObj* 的变量，就可以很容易地引用 (parent.frames[1]) 中的文档对象。这里也是一样，但我们引用的是 *Back* 窗口。同样在 *front.html* 中使用 *back.html* 中定义的变量。使用一个变量来引用 parent.Back 看起来比较容易编制代码（只用 *bRef* 来代替 parent.Back），并且可以更容易地从其他框架中引用数据。通过创建这样的一个变量，并将它用在代码中，可以很大程度地方便你自己的工作。考虑下面的函数 onCard()，它是在 *front.html* 中定义的。这个函数不仅使用 *bRef*，还创建了另外两个“别名”变量 *ref* 和 *ref2* 来引用特定的层。函数如下所示：

```
function onCard(iconRef) {
  var ref = bRef.refSlide(bRef.icons[iconRef]);
  var ref2 = bRef.refSlide("Back");
  if(document.all) {
    var ref = bRef.refSlide(bRef.icons[iconRef]);
    var ref2 = bRef.refSlide("Back");
    if((parseInt(ref.left) >= parseInt(ref2.left)) &&
      (parseInt(ref.top) >= parseInt(ref2.top)) &&
      (parseInt(ref.left) + parseInt(ref.width) <= parseInt(ref2.left) +
      parseInt(ref2.width)) &&
      (parseInt(ref.top) + parseInt(ref.height) <= parseInt(ref2.top) +
      parseInt(ref2.height))) {
      return true;
      }
    }
  else {
    if((ref.left >= ref2.left) &&
      (ref.top >= ref2.top) &&
      (ref.left + ref.document.images[0].width <=  ref2.left +
      ref2.document.images[0].width) &&
      (ref.top + ref.document.images[0].height <= ref2.top +
      ref2.document.images[0].height)) {
      return true;
      }
    }
  ref.left = ((iconRef % bRef.iconNum) * 110) + bRef.startWdh;
  ref.top = 15;
  return false;
  }
```

这当然不是我写的最长的函数，但考虑一下去掉 *bRef*，*ref* 和 *ref2* 的同样的函数。它会变得非常长，并且很复杂。

```
function onCard(iconRef) {
  if(document.all)
```

```
    if((parseInt(parent.Back.refSlide(parent.Back.icons[iconRef]).left) >=
      parseInt(parent.Back.refSlide("Back").left)) &&
      (parseInt(parent.Back.refSlide(parent.Back.icons[iconRef]).top) >=
      parseInt(parent.Back.refSlide("Back").top)) &&
      (parseInt(parent.Back.refSlide(parent.Back.icons[iconRef]).left) +
      parseInt(parent.Back.refSlide(parent.Back.icons[iconRef]).width) <=
      parseInt(parent.Back.refSlide("Back").left) +
      parseInt(parent.Back.refSlide("Back").width)) &&
      -parseInt(parent.Back.refSlide(parent.Back.icons[iconRef]).top) +
      parseInt(parent.Back.refSlide(parent.Back.icons[iconRef]).height) <=
      parseInt(parent.Back.refSlide("Back").top) +
      parseInt(parent.Back.refSlide("Back").height))) {
      return true;
        }
      }
    else {
      if((parent.Back.refSlide(parent.Back.icons[iconRef]).left >=
        parent.Back.refSlide("Back").left) &&
        (parent.Back.refSlide(parent.Back.icons[iconRef]).top >=
        parent.Back.refSlide("Back").top) &&
        (parent.Back.refSlide(parent.Back.icons[iconRef]).left +
        parent.Back.refSlide(parent.Back.icons[iconRef]).document.
images[0].width <=
        parent.Back.refSlide("Back").left +
        parent.Back.refSlide("Back").document.images[0].width) &&
        (parent.Back.refSlide(parent.Back.icons[iconRef]).top +
        parent.Back.refSlide(parent.Back.icons[iconRef]).document.
images[0].height <= parent.Back.refSlide("Back").top +
        parent.Back.refSlide("Back").document.images[0].height)) {
        return true;
        }
      }
    parent.Back.refSlide(parent.Back.icons[iconRef]).left =
      ((iconRef % parent.Back.iconNum) * 110) + parent.Back.startWdh;
    parent.Back.refSlide(parent.Back.icons[iconRef]).top = 15;
    return false;
    }
```

showgreeting()需要一个变量——贺卡选择列表的*selectedIndex*。只要*selIdx*不为0（贺卡不在“Choose One”项上），showGreeting()就隐藏当前可见贺卡层，并显示和选中项相关的贺卡层。然后*selIdx*的值变成可见层的当前值，为下一次作好准备。

浏览所有图片

想要从头至尾浏览可用的背景图片，用户只需点击“Backgrounds-->”按钮，直到

他发现最合适的一张。点击这个按钮将调用函数nextBackground()，它出现在第40～50行：

```
function nextBackground() {
  backgroundIdx = (backgroundIdx == bRef.backImgs.length - 1 ?
    backgroundIdx = 0 : backgroundIdx + 1);
  if(document.all) {
    bRef.document.background.src = bRef.backImgs[backgroundIdx].src;
    }
  else {
    bRef.document.layers["Back"].document.images[0].src =
      bRef.backImgs[backgroundIdx].src;
    }
  }
```

背景图片在*back.html*文件的第40～44行中已经预载了。由于每一个都按照*background0.jpg*，*background1.jpg*，*background2.jpg*等的形式来命名，你可以用一个整数*backgroundIdx*，接在字符串后面，就能迭代所有的图片。文档载入的时候，*backgroundIdx*的值为0。每一次用户点击“Background -->”按钮，它的值就加1，直到再没有其他图片可供浏览。换句话说，当*backgroundIdx*到达top.Back.backImgs.length -1时，就被重新设置为0，再从头开始。

我们可以利用这个新近确定的值来修改对应*Image*对象的*src*属性。由于背景图片被定位在还需精确调整的层上，我们必须采取不同的方法来访问全然不同的Navigator和IE的DOM。

对于IE，图片被当作是文档对象的一个属性，就像这样：

```
top.Back.document.background.src
```

对Navigator，你必须在层内访问文档。由于层被命名为Back，到达所需的图片就会像这样：

```
top.Back.document.layers["Back"].document.images[0].src
```

一旦确定了语法，你就只要用*backgroundIdx*将路径设置给*backImgs*图片的*src*属性。顺便说一句，这已经不是我第一次用这种循环方法了。你可以在第三章和第八章中发现相似的例子。

现在，用户可以循环浏览背景了。对于图标我们需要某些熟悉的东西。所以nextIcons()来到了我们面前，看第52～68行的代码：

```
function nextIcons() {
  for (var i = bRef.iconNum * iconIdx; i < (bRef.iconNum * iconIdx) +
    bRef.iconNum; i++) {
    if (i < bRef.icons.length && !onCard(i)) {
      bRef.hideSlide(bRef.icons[i]);
      }
    }
  iconIdx = (iconIdx >= (bRef.icons.length / bRef.iconNum)  - 1 ? 0 :
    iconIdx + 1);
  for (var i = bRef.iconNum * iconIdx; i < (bRef.iconNum * iconIdx) +
    bRef.iconNum; i++) {
    if (i < bRef.icons.length) {
      bRef.showSlide(bRef.icons[i]);
      }
    else { break; }
    }
  }
```

用户就像浏览背景一样要依次浏览图标，但是除了改变单个图片的*src*属性之外还有别的事情要做。实际上，每一个图标都是它自己的层中的一个图片。因此，点击“Icons -->”按钮就有点儿麻烦了。我们不仅需要隐藏当前显示的所有层，还必须确定究竟要显示哪一个层，这个工作需要批量进行。

对用户来说，点击20次来看20个图标是没有多大意义的。这会弄得非常单调，也是浏览器资源的浪费。你可以在前边的图表中看到，我已经选择按照四个一组的数量来显示图标。不管你选择的是什么数，它都由*back.html*中第24行的变量*iconNum*来表示。由于这是*front.html*，指向它的脚本就应该是top.Back.iconNum。具体方案是用户每点击一次“Icons -->”按钮就显示*iconNum*个图标。当然，我们也想能随心所欲地添加或者删减图标。如果你删除某个图标，用户将会看到这个图标组由四个变成了三个。其间，你不必在nextIcons()中作任何修改。

这非常简单。从头四个开始，然后将它们隐藏，并显示下面四个，如此进行，直到你显示完了所有的图标。然后又重新开始。因此，翻译成自然语言就是：隐藏四个旧的，显示四个新的。让我们来仔细看看JavaScript版本。为了确定每一组图标，我们需要使用变量*iconIdx*，它的初始值为0。第一组图标对应的*iconIdx*值为0，第二组为1，如此类推。

只要用户选择了“Icons -->”按钮，我们必须隐藏任何与*iconIdx*相对应的图标组：

```
for (var i = bRef.iconNum * iconIdx; i < (bRef.iconNum * iconIdx) +
  bRef.iconNum; i++) {
```

```
    if (i < bRef.icons.length && !onCard(i)) {
      bRef.hideSlide(bRef.icons[i]);
      }
    }
```

将变量*i*设置为iconNum * iconIdx。只要*i*小于(iconNum * iconIdx) + iconNum，它就会按1来做增量运算。如果这显得有点儿混乱的话，你可以想一下当文档载入结束的时候发生的情况。*iconNum*为4，而*iconIdx*为0。这意味着这个函数第一次被调用的时候，*i*将会是0，1，2和3。下一次被调用时，*iconIdx*变成了1（发生在函数尾部），因此*i*就会是4，5，6和7。如此类推。

变量*i*保存的整数将会被用来访问*icons*数组中的某个元素。为什么呢？每一个图标都是它自己的层，不是吗？*back.html*中的代码按照*icons*数组中的元素来给每一个层命名。例如，icons[0]指向*bear*层。

我们所需做的所有工作是隐藏和0，1，2，3相关的层，除非用户已经将其中的某个图标拖到了背景显示区域。我们马上要说到，这个工作是由函数onCard()来完成的。现在，让我们假设没有哪个图标被拖动，这样，我们就可以比较容易地来讨论这个函数。按照这个假设，我们只需从*back.html*中调用*hideSlide*，并将用*i*来访问的正确层名传递进去，正是这样：

```
bRef.hideSlide(bRef.icons[i]);
```

旧的图标被隐藏了，我们还需要把下一组显示出来。但在此之前，我们必须确定我们还没有处理完最后一组图标。如果已经处理完，我们就要将*iconIdx*重新设为0。否则，就要将*iconIdx*加1。相关代码在第59～60行：

```
iconIdx = (iconIdx >= (bRef.icons.length / bRef.iconNum)  - 1 ? 0 :
  iconIdx + 1);
```

在下一个循环中，新的图标组将变成可见的。第61～67行包含的*for*循环完成这个步骤：

```
for (var i = bRef.iconNum * iconIdx; i < (bRef.iconNum * iconIdx) +
  bRef.iconNum; i++) {
  if (i < bRef.icons.length) {
    bRef.showSlide(bRef.icons[i]);
    }
  else { break; }
  }
```

计划是循环*iconNum*次，将下一组图标显示出来。先前我们在第59～60行将*iconIdx*增量或者重置。因此，剩下的工作是使用一个和隐藏上一组时所用*for*循环几乎相同的循环。但这一次，我们要用的是showSlide()。不过这有点儿问题。还记得计划是进行*iconNum*次循环，但如果这是重新开始之前的最后一组图标，还剩有*iconNum*个图标吗？如果你有20个图标，你想一次显示4个，那就会有5个图标组。但如果你有19个图标也要每次显示4个的话，仍然会有5个图标组，只是最后一组将只有3个图标了。这就是我们之所以需要*if-else*语句来测试它是否小于图标总数的原因。如果小于，nextIcons()就显示图标。否则，这个组中就已经没有图标了，就要用*break*来终止循环。

定位所拖动的图标

正如你刚才所看到的，浏览所有的图标需要处理旧图标的隐藏和新图标的显示。除了用户已经拖动一个图标到背景显示区域的情况以外，它的运行都是良好的。不管怎样，我们想把它留在所在的地方。函数onCard()为确定它所检查的每一个图标是否应该被单独留下，或者被隐藏并回到它原来的位置付出了艰苦的劳动。这里是第83～109行的代码。onCard()不负责任何隐藏和显示任务。它返回true或者false，这样，其他函数可以做出相应的动作：

```
function onCard(iconRef) {
  var ref = bRef.refSlide(bRef.icons[iconRef]);
  var ref2 = bRef.refSlide("Back");
  if(document.all) {
    if((parseInt(ref.left) >= parseInt(ref2.left)) &&
      (parseInt(ref.top) >= parseInt(ref2.top)) &&
      (parseInt(ref.left) + parseInt(ref.width) <= parseInt(ref2.left) +
      parseInt(ref2.width)) &&
      (parseInt(ref.top) + parseInt(ref.height) <= parseInt(ref2.top) +
      parseInt(ref2.height))) {
      return true;
      }
    }
  else {
    if((ref.left >= ref2.left) &&
      (ref.top >= ref2.top) &&
      (ref.left + ref.document.images[0].width <=  ref2.left +
      ref2.document.images[0].width) &&
      (ref.top + ref.document.images[0].height <= ref2.top +
      ref2.document.images[0].height)) {
      return true;
      }
    }
```

```
ref.left = ((iconRef % bRef.iconNum) * 110) + bRef.startWdh;
ref.top = 15;
return false;
}
```

在认真阅读onCard()之前，我们需要知道用什么来证明一个图标到了背景显示区域中。最简单的例子，图标所有的边界（甚至是透明边界）都必须在背景图片的所有边缘之内或者重合。图10-8给出了什么要留下，什么要抛开的示例。

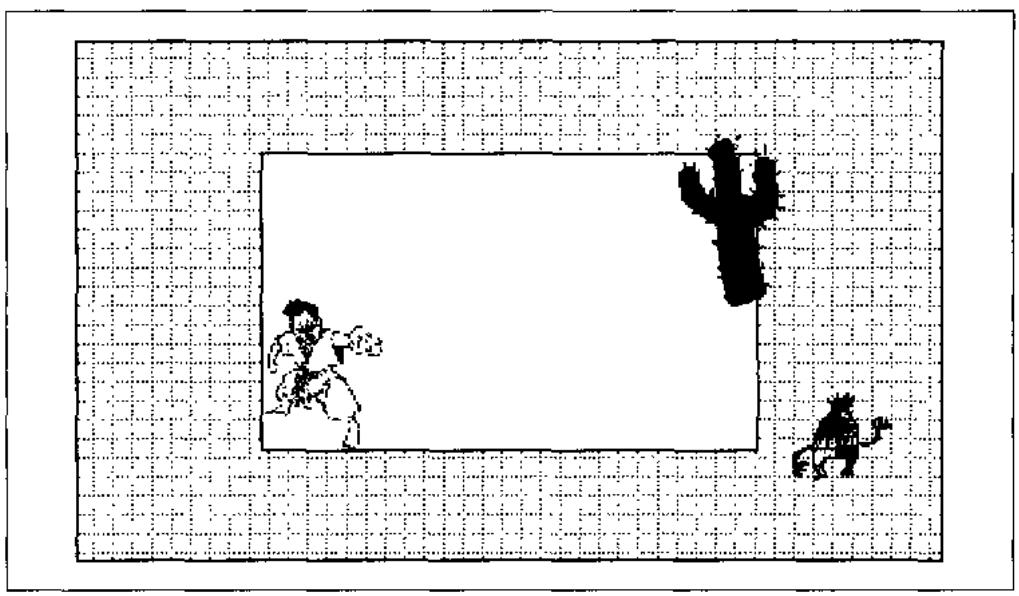


图10-8 超出范围：只有“柔道小子（judo kid）”可以留下

假设空白处是背景显示区域，那么右边的小家伙就算出局了。仙人掌很近，但它有两个边超出了背景图片的范围。仙人掌也必须返回。只有柔道小子可以继续显示。下面是它的工作过程。

所有东西都按照像素来作处理。由于背景包含在一个层内，我们可以用DHTML来确定它的左边和上边与文档的左边和上边的相对位置。因为层中只包含一个图片，我们可以用*Image*对象的*width*和*height*属性来精确地判定层的高和宽。对于图标也是如此。我们可以使用层的*left*和*top*属性，以及*Image*的*width*和*height*属性。这个函数有两个嵌套的*if*语句，但最外层的if和else子句基本上执行相同的动作——一个针对IE，一个针对Navigator。这里是onCard()的前面部分，这个部分为IE服务：

```
if(document.all) {
  if((parseInt(ref.left) >= parseInt(ref2.left)) &&
    (parseInt(ref.top) >= parseInt(ref2.top)) &&
    (parseInt(ref.left) + parseInt(ref.width) <= parseInt(ref2.left) +
    parseInt(ref2.width)) &&
    (parseInt(ref.top) + parseInt(ref.height) <= parseInt(ref2.top) +
    parseInt(ref2.height))) {
    return true;
    }
  }
```

背景图片有四个边缘。图标也是一样。因此，我们需要四个测试来确定没有一个图标边缘是否超出了背景的范围。这里是第87～94行*if*语句的自然语言版本：

IF 图标的左边缘触到了背景的左边缘或在它右边，
AND 图标的上边缘与背景上边缘重合或在它下面，
AND 图标的右边缘触到了背景的右边缘或在它左边，
AND 图标的下边缘与背景下边缘重合或在它上面，则
RETURN TRUE.

确定每一个层的左边缘和上边缘位置倒是不难。只要用每个层的*left*和*top*属性就行了。确定每层的右边和下边也难不了多少。右边缘是左边缘位置加上层的宽度。而下边缘只要上边缘位置加高度就行了。

到现在，你可能已经注意到了两件事情。首先，变量*ref*和*ref2*已经被分别设置成了图标和背景层。我这样做只是为了提高代码的易读性。第二，函数parseInt()都处都是。IE将left和top属性以250px这样的字符串返回，而不是250。parseInt()将字符串值转换为数字，这样我们才能做数学运算。

最外层的else子句对Navigator做同样的工作。你真的不需要调用parseInt()，因为Navigator返回的top和left属性是数字形式。

```
else {
  if((ref.left >= ref2.left) &&
    (ref.top >= ref2.top) &&
    (ref.left + ref.document.images[0].width <=  ref2.left +
    ref2.document.images[0].width) &&
    (ref.top + ref.document.images[0].height <= ref2.top +
    ref2.document.images[0].height)) {
    return true;
    }
  }
```

因此，如果当前处理的图标通过了所有四个检测，两种浏览器都会返回true。如果没有，看看发生了什么：

```
ref.left = ((iconRef % bRef.iconNum) * 110) + bRef.startWdh;
ref.top = 15;
return false;
```

onCard()发现图标不在背景显示区域内，就将它们打回原来的位置。所有的图标都有相同的上边缘位置，15像素。但每一个的左边缘要看它在组中的序号了。没关系。用变量*iconRef*和*iconNum*做的一个快速计算可以确定图标最初的位置在哪里。按照规定，每个图标的宽度为100像素，在每两个之间有10像素的水平距离，因此，图标的位置就很容易确定啦。最后，函数返回false。

操作测试

如果发送者可以预览收件人所能看到的结果，那一定非常不错。选择“Test”将打开一个新的浏览器窗口，达到这个目的。第131～145行给出了具体过程：

```
function testGreeting(fObj) {
  var msgStr = '<HTML><TITLE>Cyber Greeting Test Page</TITLE>' +
    genGreeting(fObj) + '<TABLE ALIGN="CENTER"><TR><TD><FORM>' +
    '<INPUT TYPE=BUTTON VALUE="     OK     " onClick="self.close();">' +
    '</FORM></TD></TR></TABLE></HTML>';
  newWin =  open('', '', 'width=' + (bRef.backImgs[backgroundIdx].width +
    50) + ',height=600,scrollbars=yes');
  with(newWin.document) {
    open();
    writeln(msgStr);
    close();
    }
  newWin.focus();
  }
```

testGreeting()只有两个责任：打开一个足够宽的窗口来显示消息，然后将传递进来的不管什么内容写到新窗口的文档流。内容存储在局部变量*msgStr*中。它有少量的静态HTML和动态的贺卡内容组成，贺卡内容来自函数genGreeting()，依次准备齐全。*msgStr*在末尾也包含一个表单，它带有一个用来关闭子窗口的按钮。轻轻一点就可以了。一旦*msgStr*中载入了内容，testGreeting()就打开一个窗口，它的宽度比背景图片大50像素，高度可以随便定为600像素。函数将内容写到文档流中，并申请将焦点放到新打开的窗口，这个动作也在历史记录中。

创建实际贺卡

testGreeting()为预览贺卡提供了一个窗口；但是，genGreeting()创建其中的动作。

这里是第 147～177 行，共 31 行代码：

```
function genGreeting(fObj) {
  var greetingIdx = fObj.Greetings.selectedIndex;
  var msg = fObj.Message.value;

  msg = msg.replace(/\r+/g, "");
  msg = msg.replace(/\n+/g, "<BR><BR>");

  var msgStr = '<TABLE BORDER=0><TR><TD COLSPAN=2><FONT FACE=Arial>' +
    '<H2>Your Cyber Greeting</H2>To: ' + fObj.Recipient.value +
    '<BR><BR></TD></TR><TR><TD VALIGN=TOP><IMG SRC="' + top.baseURL +
    '/images/background' + backgroundIdx + '.jpg">' +
    '<DIV STYLE="position:relative;left:40;top:-255;font-family:Arial;' +
    'font-size:48px;font-weight:bold;">' +
    parent.greetings[greetingIdx] + '</DIV>';

  var iconStr = '';
  for (var i = 0; i < bRef.icons.length; i++) {
    if(onCard(i)) {
      iconStr += '<DIV STYLE="position:absolute;left:' +
        bRef.refSlide(bRef.icons[i]).left + ';top:' +
        (parseInt(bRef.refSlide(bRef.icons[i]).top) -
        (document.all ? 140 : 150)) + ';"><IMG SRC="' + top.baseURL +
        '/images/' +  bRef.icons[i] + '.gif"></DIV>';
      }
    }

  msgStr += iconStr + '</TD></TR><TR><TD WIDTH=' +
    bRef.backImgs[backgroundIdx].width + '><FONT FACE=Arial>' + msg +
   '</TD></TR></TABLE>';
  return msgStr;
  }
```

这个函数有点儿麻烦，但我们按照必须提供的项目顺序来依次阅览，就会简单得多。考虑一下。genGreeting()只需要返回表示下面内容的 HTML：

- 显示收件人 Email 的文字
- 背景图片
- 正确定位的贺卡

- 正确定位的图标
- 发送者的文字消息

这并没有多少东西，但在创建内容之前，我们必须做点儿内部管理。看一看第148～152行的代码：

```
var greetingIdx = fObj.Greetings.selectedIndex;
var msg = fObj.Message.value;
msg = msg.replace(/\r+/g, "");
msg = msg.replace(/\n+/g, "<BR><BR>");
```

我们定义了两个局部变量，*greetingIdx*和*msg*。*greetingIdx*是“贺卡”选择列表的*selectedIndex*。*msg*表示用户在消息域中所输入的消息。由于贺卡将被显示为HTML，换行符不会被编译。这意味着任何中断都将用
标签来代替。现在，我们可以开始创建贺卡了。让我们从头开始，依次往下进行。第154～160行给出了这个过程：

```
var msgStr = '<TABLE BORDER=0><TR><TD COLSPAN=2><FONT FACE=Arial>' +
  '<H2>Your Cyber Greeting</H2>To: ' + fObj.Recipient.value +
  '<BR><BR></TD></TR><TR><TD VALIGN=TOP><IMG SRC="' + top.baseURL +
  '/images/background' + backgroundIdx + '.jpg">' +
  '<DIV STYLE="position:relative;left:40;top:-255;font-family:Arial;' +
  'font-size:48px;font-weight:bold;">' + parent.greetings[greetingIdx] +
  '</DIV>';
```

将所有的东西都包含在一个表格中显得比较简单。因此，将局部变量*msgStr*设置为表格的标头。第一行包含一个标题，它后面跟着所需四个项目的头一项——收件人的Email地址。它在*EntryForm*表单的收件人域值中。背景图片在下面一行。用变量*baseURL*和*backgroundIdx*，创建一个表示到适当图片的路径的字符串就不是什么复杂的事。记住，如果*Greetings*选择列表的*selectedIndex*值为4，那么*background4.jpg*就是当前所显示的。

注意，在页面上定位背景图片的时候并没有用到DHTML。由于标题和贺卡大约只各占一行，就确实不需要DHTML的出现了。我们知道，图片会显示在接近页面上部靠左的某个地方。在这些代码的最后几行中，发送者所选的贺卡会被放在和已经定位的背景图片相关的位置上。左边缘被设为40像素。上边是-255。这些都是定制的设置，它们是*back.html*中所示的背景图片位置和贺卡中背景图片的确切位置的作用结果。当添加背景时，你可能会不得不在*back.html*和预览文档之间左顾右盼，调整左边和上边的位置，直到非常匹配。然后，你就不会为它感到麻烦了，直到作一些重大的修改之后，比如调整图片的尺寸。

让我们开始另外三个必需的项目，先不说后两个。下一个“项目”是用户拖到显示区域的图标。我们给出了第 162～171 行的代码：

```
var iconStr = '';
for (var i = 0; i < bRef.icons.length; i++) {
  if(onCard(i)) {
    iconStr += '<DIV STYLE="position:absolute;left:' +
    bRef.refSlide(bRef.icons[i]).left +
    ';top:' + (parseInt(bRef.refSlide(bRef.icons[i]).top) -
    (document.all ? 140 : 150)) + ';"><IMG SRC="' +
    top.baseURL + '/images/' +  bRef.icons[i] + '.gif"></DIV>';
    }
  }
```

局部变量的初始值为空串，它将保存背景显示区域中所有图标的URL和位置。这个过程很简单：迭代所有的图标。对每一个在显示区域内的，创建HTML来复制这个图标（图片）到另外一个页面上，保持和背景图片的同样的相对位置（左边和上边的位置）。记住，函数 onCard()确定图标的位置，并以此判断它是否合格。

为每一个图标所产生的代码是DIV标签内的 IMG标签。为DIV标签设置一个 STYLE 属性，并为当前图标的左边缘和上边缘设置一个绝对的位置，是 140或-150，就要取决于所用的浏览器是 Internet Explorer还是 Navigator。两个问题，对吗？

1. 当图标的上侧位置偏移了大约100个像素时我为什么还能使用左侧位置？
2. 为什么像素偏移量在每一种浏览器中有所不同？

问得好。

首先想一想贺卡界面中背景图片的所在之处。它大约在页面的中下部分，这依靠你的分辨率来决定。现在考虑测试贺卡的背景图片。它靠近页面的上部，就在标题和收件人 Email地址之下。像素偏移量用来弥补这个区别。让背景图片出现在界面和测试贺卡（它也和真的贺卡一样精确）的相同位置，这个位置并不是必需的。至于两种浏览器间像素偏移量的区别，每一个差别都让层的定位稍有不同。用10个像素的差异来作纠正。

为每个定位了的图标创建一个层之后，将 *iconStr* 和 *msgStr* 连接起来，伴随一些HTML的结束标签和 *msg*，用来创建一个嵌入表格中的完善的电子贺卡：

```
msgStr += iconStr + '</TD></TR><TR><TD WIDTH=' +
```

```
    bRef.backImgs[backgroundIdx].width + '><FONT FACE=Arial>' +  msg +
    '</TD></TR></TABLE>';
  return msgStr;
```

msg 被插入到一个数据单元中，这个单元的宽度和背景图片相同，以达到更好的格式化效果。然后返回这整个的串。

发送

用户已经制作了一份精彩的贺卡，并且测试了几次，最后感到很满意。最后要做的就是选择“Send”。这样做要调用函数 shipGreeting()。它位于第 111～129 行，如下所示：

```
function shipGreeting(fObj) {
if (fObj.Recipient.value == "") {
    alert('You need an email address in the From: field');
    return false;
    }
  else if (fObj.Message.value == "") {
    alert("You need to type a Message.");
    return false;
    }
  else if (fObj.Greetings.selectedIndex == 0) {
    alert('You need to choose a Greeting.');
    return false;
    }
  fObj.EntireMessage.value = genGreeting(fObj);
  fObj.BaseURL.value = top.baseURL;
  return true;
  }
```

这个函数着实很短，通过第 187 行的 *onSubmit* 事件处理方法得到调用。shipGreeting()不仅准备表单以提交到 Web 服务器，它还执行快速的确认。只要发送者是按照一些简单的指示来做的，事情就会进行的得很顺利。我所给的唯一规则是发送者必须输入一个 Email 地址和一个消息的相关内容，并从选择列表中选择一个贺卡。你可能会觉得受到拘束，但它们现在就有用了。

如果输入的信息通过了所有三个“严格”的检测，shipGreeting()将会修改第 188～190 行的三个隐藏域的值。初始值为空串的 *EntireMessage* 和 *BaseURL* 域将会设置为服务器端脚本所需的信息。为了避免服务器端脚本不得不创建贺卡的HTML，就将 EntireMessage.value 设置为 genGreeting()的输出。

不管这个数字是什么，它将会被保存在用来存储收件人的贺卡的文件名中。如果UniqueID.value被设置为1324，用来存储它的文件将被命名为*greet1324.html*。不会象证明Format的最后一项定理那么复杂，但这种随机数字命名文件的协定可以让你创建成千上万唯一的文件名。在我们的想像里，创建两个同名文件的机会是非常微小的。另一端脚本所需信息的其他部分是基本URL，它有助于指向所有的贺卡，背景图片和图标。因此，将BaseURL.value设置为top.baseURL的值。此后，把表单提交给ACTION属性*greet.pl*中所指出的URL。你可以在第186行看到它。

JavaScript 技巧：优化你的函数

你怎么知道在函数中应该放多少代码同时不让它变得过长呢？什么时候你会对自己说："OK。在这里打住，并且把剩余的部分放到另外一个函数里"？我没法为此设计一套指导方针，但我总是试图以最少量的代码发挥最大的效用。让我们看一看。函数genGreeting()。它协助产生贺卡预览和实际提交两部分的代码。onCard()又如何呢？当用户点击任何可用的图标时，它完全可以用来确定图标的位置。它还在创建预览和提交代码时担任一角。在大多数情况下，你可以通过把函数变小而获得这个有利条件。这并不永远是可行的或者可生效的，但它不失为一个很好的方法。

注意

在我们讲述服务器端脚本之前，还要最后强调一点。位于第70～81行的resetForm()只是在每次FORM首次载入或刷新的时候清空登录表单。用BODY标签中的onLoad事件处理方法来调用它。在这个前提下，让我们来看看当应用程序需要Web服务器时会发生什么情况。

服务器端

和第八章中的购物车应用程序相似，这个应用程序也需要某些类型的服务器端装置。用户创建的贺卡在客户端利用JavaScript来产生。然后，这个信息被送回Web服务器的一个应用程序——服务环境的脚本中，比如ASP、服务器端JavaScript，或者ColdFusion。

这个脚本环境将解读用户提交的信息，然后创建一个唯一的文件，将贺卡代码写在其中。这个唯一的文件名和Email消息中发送给收件人的那个文件名是一样的。当收件人点击Email中的链接时，文件将会被准备妥当，翘首以待。

之后，脚本将会打印一个响应信息给贺卡发送者，表示一切正常，更重要的是准备一个现成的HTML表单，让创建者提交，来发送给收件人。

现在，我想起你买这本书是冲着JavaSript来的，但我所给出的脚本却是用Perl写的，它短小而且精干。此外，Perl是一种相对简单并且功能强大的语言，正在成为WinNT平台所选用的脚本语言。参见附录三，可以获得更多有关安装Perl运行脚本的信息，并且可以获得一个有关这本书中应用程序的两个Perl脚本如何运作的说明。

应用程序扩展

让我们来看几个扩展这个应用程序的途径。

添加返回电子贺卡的链接

为什么不添加一个返回电子贺卡网站的链接呢？你可以通过在*front.html*的genGreeting()函数中添加如下的代码来达到这个目的：

```
+ ' <A HREF="' + top.baseURL + '/index.html">Go To Cyber Greeting</A>';
```

这假设*index.html*、*front.html*和*back.html*都定位在*baseURL*所指的目录下。如果不是，用你想要的URL来代替上面语句中的top.baseURL。

添加图片主题

这个应用程序就是为图片主题而产生的。圣诞节，情人节，圣帕特里克节，万圣节前夜，反正包括你所知道的一切。没有必要把注意力放在假日上。你可以使用季节性的背景和图标，加上身体健康或生日快乐的主题，或者关于任何你希望加入的题目。

广告标幅策划

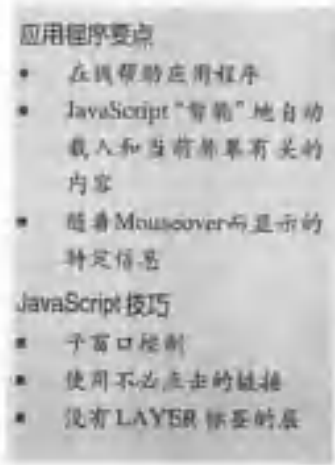
如果你想让人们免费使用这个服务，为什么不获取一点收入呢？在shipGreeting()函数中，可以用一些逻辑来在贺卡底部选择一个标幅，插入IMG标签代码。如果你使用前面所说的主题，就可以按照选择的主题来设置标幅。

使贺卡有更多交互式内容

这个应用程序利用JavaScript事件来创建定制贺卡。是的，是定制，但没有多少交互。你可以试试在发送的贺卡中附带图片翻转或者点击物件，或者动来动去的东西。如果你有一个或两个很酷的Java程序，你也可以把它们作为贺卡。人们喜欢玩没有什么用途的东西；成千上万的站点都是如此。

第十一章

能感知当前环境的帮助



不管你认为自己的程序使用起来是多么简单，但有些用户还是不可避免地要碰到麻烦。一些用户问题的答案显而易见。还有一些用户会对你吹毛求疵，没办法，谁叫这东西是你写的呢！根据应用程序质量的好坏，在线文档可以让用户按路线很快返回，也可以把他们带得更远。帮助文件使用起来也必须简单。这就是本应用程序要解决的问题。

不要仅只为了方便用户导航，还要考虑方便你自己的建设和维护。而这个程序本身似乎没什么用。相比起来，第七章中的应用程序更讨人喜欢，它里面的代码可以放在你的某个应用程序中，为你服务。

我将这个应用程序命名为"选择列表JavaScript"，它展示了一些不能让你产生较深印象但的确很有用的东西（用JavaScript和选择列表完成的）。图11-1给出了打开的界面。

这个应用程序展现了如何用选择列表来改变背景颜色，载入文档和生成其他选择列表，所有这一切都只需少量的JavaScript。它的功能和理论上的预想差不多，选择标有"Help"的链接按钮，可以用改变文档背景色的相应文件打开一个子窗口。如图11-2所示。

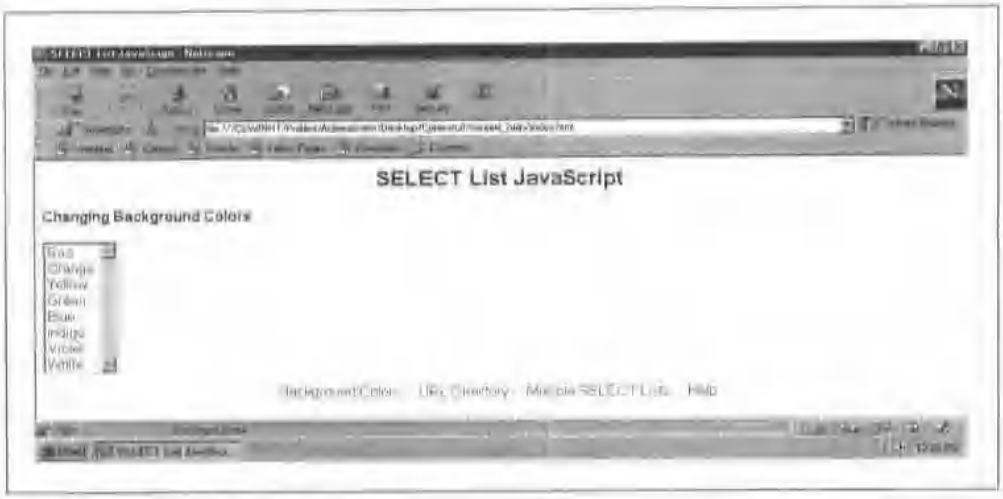


图 11-1 选择列表 JavaScript

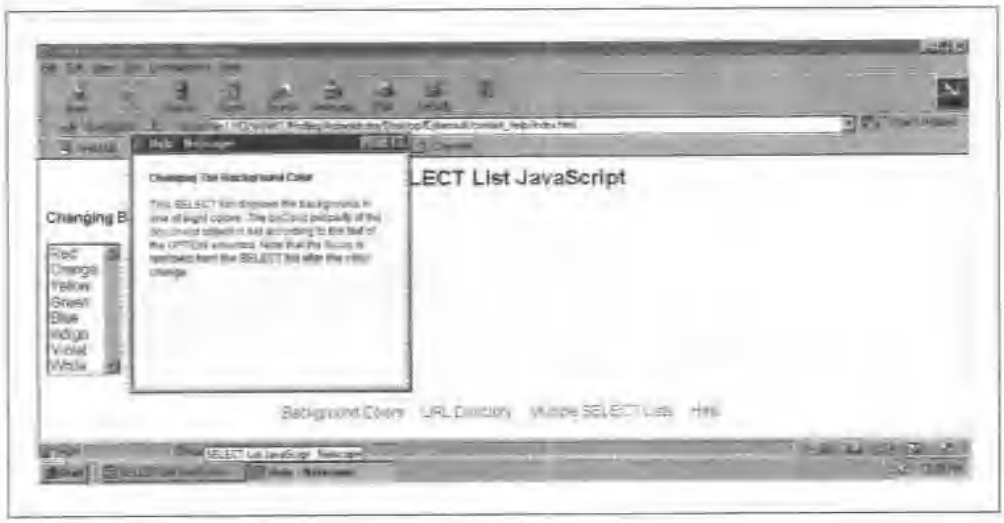


图 11-2 改变背景色的过程

这没有什么稀罕的，我承认。但是，点击“URL Directory”链接（载入URL目录页面），然后点击“Help”将会得到关URL载入过程的文件。如图11-3所示。这真实太酷了。现在，用户不必搜索准确的帮助文件了。他们可能碰巧会有一个关于当前显示内容的问题。帮助文件现在能够感知当前环境（context-sensitive）。

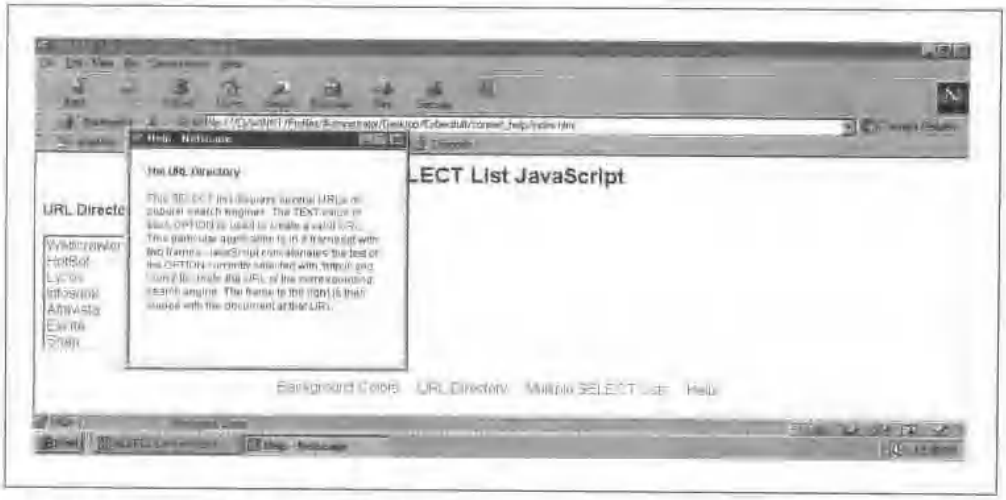


图 11-3 同样的链接，不同的文件

好像应用程序有特殊的智力似的。它总是“知道”用户“在”哪里。作为附加的联系，一些文件文字被超链接到其他信息上，以把内容继续下去。但是，用户不必点击链接来载入另外的文档。只要将鼠标箭头移到任何链接上，就会在高亮度的层中显示相关信息。如图 11-4 所示。

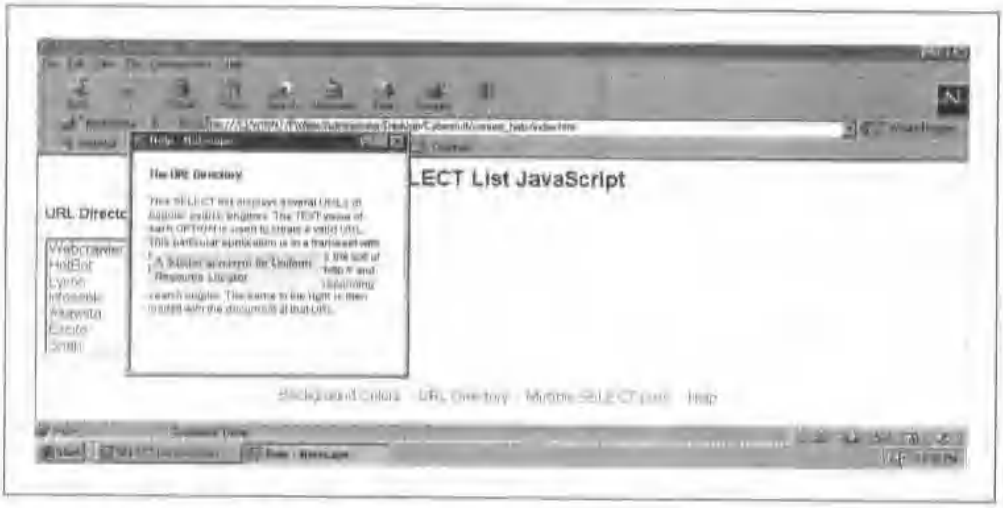


图 11-4 不必等待的特定信息

现在，用户在使用链接之后不必回到初始文档。将鼠标箭头从链接上移走就会重新把层隐藏起来。这个功能可以用在几乎所有用户需要帮助的应用程序中。

执行条件

你需要 IE 或 Navigator 4.x 或更高版本，因为使用了 DHTML 和新的事件模式。要确保你有一个至少 1024 × 768 的分辨率。这并不是强制的，但不这样的话帮助窗口就会覆盖大部分主窗口的内容。

语法细则

这个应用程序包含在一个有许多文件的框架设置中。下面是一个简单的总结：

index.html

顶层文件，含有框架设置和顶层变量

top.html

显示选择列表 JavaScript 标题

nav.html

显示页面链接

background.html

改变背景色

multiselect.html

基于其他两个的选择而组装一个选择列表

urldirectory.html

基于选择列表选项载入搜索引擎

help/background.html

和 *background.html* 相对应的帮助文档

help/multiselect.html

和 *multiselect.html* 相对应的帮助文档

help/urldirectory.html

和 *urldirectory.html* 相对应的帮助文档

help/help.js

JavaScript 源文件。

你不太可能想钻到 *background.html*，*multiselect.html* 和 *urldirectory.html* 中那些选择列表功能的后面去看看它们的机理。它其实非常容易了解，但这不是我们这一章要讲的内容。不过至少要看一看 *multiselect.html* 中选择列表的重新生成。它很方便易用。但是我们现在要通过两个步骤来开展我们的工作：

1. 上下文有关的帮助：在帮助窗口（在 *nav.html* 中）中载入正确的文档。
2. 显示和隐藏特定信息：使用 mouseover（在 *help/help.js* 中）。

能感知当前环境的帮助

这个内容十分简单。它完全包括在 *nav.html* 中。例 11-1 给出了代码。

例 11-1：nav.html

```
<HTML>
<HEAD>
  <TITLE>top.html</TITLE>
</HEAD>
<STYLE TYPE="text/css">
<!--

A
  {
    text-decoration: none;
  }

BODY
  {
    font-family: Arial;
    text-align: center;
  }

//-->
</STYLE>
<SCRIPT>
<!--
var helpWin;

```

```
25 function inContext(currFile) {
26   var start = currFile.lastIndexOf('/') + 1;
27   var stop  = currFile.lastIndexOf('.');
28   var helpName = currFile.substring(start, stop);
29   if(helpWin == null || helpWin.closed) {
30     helpWin = open('help/' + helpName + '.html', 'helpFile',
31       'width=' + top.wdh + ',height=' + top.hgt +
32       ',left=100,top=100,scrollbars=no');
33     }
34   else {
35     helpWin.location.href = 'help/' + helpName + '.html';
36     }
37   helpWin.focus();
38   }
39
40 //-->
41 </SCRIPT>
42 <BODY>
43
44 <A HREF="background.html" TARGET="WorkArea">Background Colors</A>
45    
46 <A HREF="urldirectory.html" TARGET="WorkArea">URL Directory</A>
47    
48 <A HREF="multiselect.html" TARGET="WorkArea">Multiple SELECT Lists</A>
49    
50 <A HREF="javascript: inContext(parent.WorkArea.location.href);">Help</
A>
51
52 </BODY>
53 </HTML>
```

函数inContext()的工作前提是：对于你想要显示帮助的每一个文档，在一个与其同名的，扩展名为*.html*的文件中创建帮助文件。也就是说，*background.htl*这个用来改变背景色的文件，在*help/*目录下有一个相应的文件，名为*background.html*。第25～38行给出了细节内容：

```
function inContext(currFile) {
  var start = currFile.lastIndexOf('/') + 1;
  var stop  = currFile.lastIndexOf('.');
  var helpName = currFile.substring(start, stop);
  if(helpWin == null || helpWin.closed) {
    helpWin = open('help/' + helpName + '.html', 'helpFile', 'width=' +
      top.wdh + ',height=' + top.hgt +
      ',left=100,top=100,scrollbars=no');
    }
  else {
    helpWin.location.href = 'help/' + helpName + '.html';
    }
```

```
helpWin.focus();
}
```

这个函数要求一个URL作为它的参数。*currFile*可以是一个绝对的URL，比如http://some.place.com/some/document.html，也可以是一个相对的附带一个查询的URL，比如document.cgi?search=all。从URL中，我们只需要文件名，而不需要主机和域，或者在它前面的上级目录，也不需要文件扩展名或跟在后面的查询语句。换句话说，我们需要的就是最后一个斜杠（/）后面的所有东西，直至但不包括URL中的最后一个句点（假设每一个文件名都有扩展名，且总是只有一个）。

因此，变量*start*给了我们最后一个斜杠+1的索引。假设在URL中没有斜杠，那也不要紧。lastIndexOf()如果没有碰到斜杠，就返回-1。在上面加1，你得到的是0。这就是我们要开始的地方。变量*stop*被设置为URL中最后一个节段的索引。现在第28行的substring()函数逐渐从URL中采集了所要的子串，并将它赋值给helpName。我们来仔细看一下：

```
var helpName = currFile.substring(start, stop);
```

接下来的几行代码用*helpName*打开了一个窗口，它和帮助文件的文档内容一致。第30～32行的open()函数的第一个参数指向正确的帮助文件：

```
helpWin = open('help/' + helpName + '.html', 'helpFile', 'width=' +
  top.wdh + ',height=' + top.hgt + ',left=100,top=100,scrollbars=no');
```

注意，子窗口的宽度和高度是在运行中利用*top.wdh*和*top.hgt*变量来确定的，两个都设置为300。这两个变量位于*index.html*中。由于应用程序在应用的其他部分引用这些变量，我就把它们放在那里，以便使用。待会儿，你将会看到我们使用这些变量来确定窗口尺寸的原因。现在你唯一需要的是一个用来调用函数的良好链接。这里是第50行代码：

```
<A HREF="javascript: inContext(parent.WorkArea.location.href);">Help</A>
```

点击这个链接将会调用inContext()，并传递定位在*WorkArea*框架中的文档的URL。只要你在*help/*目录下有一个命名相似的文档，新的帮助弹出系统就可以通过伸缩来适应几乎所有的程序。

JavaScript 技巧：远程窗口控制

用户一次需要打开多少个帮助窗口呢？最好的猜测是一个。如何实现这一目的呢？你是否注意到，全局变量*helpWin*在定义但未初始化之后被设置成了子窗口对象。换句话说，*helpWin*得到了定义但什么设置也没有，它的值是null。然后，通过设置返回值，打开帮助窗口。

用户第一次点击帮助的时候，下面的代码将“决定”是打开一个新的窗口呢，还是定位现有的已经打开的窗口：

```
if(helpWin == null || helpWin.closed) {
  helpWin = open('help/' + helpName + '.html', 'helpFile',
    'width=' + top.wdh + ',height=' + top.hgt +
    ',left=100,top=100,scrollbars=no');
  }
else {
  helpWin.location.href = 'help/' + helpName + '.html';
  }
```

如果*helpWin*为null，那么它就还没有被open()函数的返回值初始化过。然后inContext()打开一个新的子窗口。如果helpWin已经是一个对象，那么作为一个窗口对象，*helpWin*包含closed属性，如果窗口被关闭它是true，否则为false。因此，如果*helpWin.closed*是true，那么用户已经打开又关闭了子窗口。这表示我们将需要另外一个窗口。

如果*helpWin.closed*为false，则子窗口仍然是开着的，因此第35行仅仅载入了适当的文档而没有调用open()。这样的话怎样才会打开很多窗口呢？如果用户在关闭他所打开的窗口之前再次点击“Help”，就会弹出另外一个窗口。再点击一次，将会有第三个。谁需要这么多呢？查看null值和*closed*属性就可以避免这种情况出现。如果用户让窗口开着也没有关系。

注意： 这个经由 / 和 . 来产生文件名的方法并不十分可靠。例如，指向默认文件名的URL将破坏这个规则，比如http://web.net.com/或../。你想用它来引用什么类型的文件名称呢？如果你打算使用它们的话，要确保修改你的代码来适应这些URL。

显示和隐藏特定信息

刚才看到的弹出技术可以载入我们所需要的帮助信息。利用链接和鼠标移动来显示特定的帮助信息需要DHMTL技术——我们已经用到过一些，还有一些是尚未接触的。幸运的是，它的绝大部分内容都在JavaScript源文件*help/help.js*中。例11-2给出了它的代码。

例11-2：help/help.js

```
var NN    = (document.layers ? true : false);
var hideName = (NN ? 'hide' : 'hidden');
var showName = (NN ? 'show' : 'visible');
var zIdx     = -1;
var helpWdh  = 200;
var helpHgt  = 200;
var x, y, totalWidth, totalHeight;

function genLayer(sName, sLeft, sTop, sWdh, sHgt, sVis, copy) {
  if (NN) {
    document.writeln('<LAYER NAME="' + sName + '" LEFT=' + sLeft +
    ' TOP=' + sTop + ' WIDTH=' + sWdh + ' HEIGHT=' + sHgt +
    ' VISIBILITY="' + sVis + '" z-Index=' + (++zIdx) + '>' + copy +
    '</LAYER>');
    }
  else {
    document.writeln('<DIV ID="' + sName +
      '" STYLE="position:absolute; overflow:none; left:' +
      sLeft + 'px; top:' + sTop + 'px; width:' + sWdh + 'px; height:' +
      sHgt + 'px; visibility:' + sVis + '; z-Index=' + (++zIdx) + '">' +
      copy + '</DIV>'
      );
    }
  }

function hideSlide(name) {
  refSlide(name).visibility = hideName;
  }

function showSlide(name) {
  refSlide(name).visibility = showName;
  }

function refSlide(name) {
  if (NN) { return document.layers[name]; }
  else { return eval('document.all.' + name + '.style'); }
  }

function motionListener() {
  if (NN) {
```

```
41      window.captureEvents(Event.MOUSEMOVE);
42      window.onmousemove = grabXY;
43      }
44    else {
45      document.onmousemove = grabXY;
46      }
47    }
48
49  function grabXY(ev) {
50    if(NN) {
51      x = ev.pageX;
52      y = ev.pageY;
53      }
54    else {
55      x = event.x;
56      y = event.y;
57      }
58    }
59
60  function helpDisplay(name, action) {
61    if(action) {
62      totalWidth = x + helpWdh;
63      totalHeight = y + helpHgt;
64      x = (totalWidth > opener.top.wdh ? x -
65        (totalWidth - opener.top.wdh + 75) : x);
66      y = (totalHeight > opener.top.hgt ? y -
67        (totalHeight - opener.top.hgt) : y);
68      refSlide(name).left = x - 10;
69      refSlide(name).top = y + 8;
70      showSlide(name);
71      }
72    else { hideSlide(name); }
73    }
74
75  motionListener();
```

让我们按两个步骤来查看这个功能。首先我们将要讨论关于创建包含特定信息的层的问题。然后，我们去看看有关显示和隐藏这些层的内容。

创建层

如果你看过了第二，四，六，七或第九章中的任何一章，那么这里的前面二十几行代码会使你感到特别亲切。如果你没有看过，可以参见第三章中关于函数genLayer()，hideSlide()，showSlide()和refSlide()的具体内容。我们将会像在其他一些章节中所做的那样创建一些层。但现在，我们需要添加一个额外的步骤。变量*helpWdh*和*helpHgt*都被设置为200像素。它们用来确定每个层的默认宽

度和高度。这是很重要的，因为我们马上要用它们以及*top.wdh*和*top.hgt*来完成定位工作。

这些函数是我们创建层的工具。我们所需做的所有工作是调用genLayer()，把内容和其他变量一起传递进去。你会发现每一个帮助文件中都有对这个函数的调用。在其他的帮助文档中情况是那么相似，让我们来看看*help/background.html*中的代码：

```
var helpOne = '<SPAN CLASS="helpSet">This property is a string that ' +
  'reflects the current background color of the document.</SPAN>';
var helpTwo = '<SPAN CLASS="helpSet">This property of the ' +
  '<TT>window</TT> object contains the object hierachy of the current ' +
  'Web page.</SPAN>';

genLayer("bgColor", 0, 0, helpWdh, helpHgt, hideName, helpOne);
genLayer("document", 0, 0, helpWdh, helpHgt, hideName, helpTwo);
```

变量*helpOne*包含用来显示第一个特定链接（"bgColor"）的字符串，而*helpTwo*对文档链接执行相同的任务。尽管这不完全是原文。两个字符串包含两个SPAN标签，赋值为层叠样式表类定义*.helpSet*。*.helpSet*类在STYLE标签中定义。在这里只给出大概内容。这并不是你见过的最精确的样式表类定义，但它却很有助于定义层。

```
.helpSet
  {
    background-color: #CCFFCC;
    padding: 5px;
    border: 2px;
    width: 200px;
    font: normal 10pt Arial;
    text-align: left;
  }
```

这一段脚本包含两次对genLayer()的调用。注意，取代传递数字给每个层的宽度和高度的是，变量*helpWdh*和*helpHgt*被传递进来了。稍后，这将为我们完成更好的定位。每一个层最开始都借助于变量*hideName*来设置为隐藏的。

层现在已经创建好了。所有用户需做的事只是根据需要有效地显示它们。这个功能来自函数motionListener()，grabXY()和helpDisplay()。第一个函数motionListener()位于第39～47行中：

```
function motionListener() {
  if (NN) {
    window.captureEvents(Event.MOUSEMOVE);
    window.onmousemove = grabXY;
```

```
    }
  else {
    document.onmousemove = grabXY;
    }
  }
```

我们应该在页面上指向某个层的链接的位置显示层。为了达到这个效果，我们必须跟踪屏幕上鼠标的位置，在它移到链接上的时候。函数 motionListener() 将 *onMouseMove* 事件处理方法设置为在每次用户移动鼠标的时候调用函数grabXY()。Navigator 和 IE 两种浏览器都支持 *onMouseMove*，但 Navigator 在窗口对象中执行，而 IE 针对的是文档对象。Navigator 也需要调用 captureEvents() 函数来指定 *mousemove* 事件。

每次用户移动鼠标时，函数 grabXY() 将变量 *x* 和 *y* 设置为鼠标箭头的水平和垂直坐标像素值。这里是第 49～58 行的代码：

```
function grabXY(ev) {
  if(NN) {
    x = ev.pageX;
    y = ev.pageY;
    }
  else {
    x = event.screenX;
    y = event.screenY;
    }
  }
```

这些坐标在 IE 和 Navigator 中的执行是不同的。Navigator 4.x 在运行中为每一个对事件处理方法的调用创建一个事件对象。这个对象得自参数 *ev*。另一方面，IE 有一个内置的事件对象。每次用户移动鼠标时对 grabXY() 的调用使变量 *x* 和 *y* 不断变化。当用户移过一个链接，*x* 和 *y* 将建立一个用来定位特定信息层的良好的参考点。

信息显示

当用户把鼠标从一个链接上移过时，将会调用函数 helpDisplay()。下面首先给出调用这个函数的 HTML（来自 *background.html*），然后是函数本身：

```
<A HREF="javascript: void(0);"
  onMouseOver="helpDisplay('bgColor', true);"
  onMouseOut="helpDisplay('bgColor', false);"> bgColor
  </A>
```

*onMouseOver*事件处理方法负责层的显示；*onMouseOut*负责将层重新隐藏起来。它们两个都用helpDislay()来处理。看第60～73行的代码：

```
function helpDisplay(name, action) {
  if(action) {
    totalWidth = x + helpWdh;
    totalHeight = y + helpHgt;
    x = (totalWidth > opener.top.wdh ? x -
      (totalWidth - opener.top.wdh + 75) : x);
    y = (totalHeight > opener.top.hgt ? y -
      (totalHeight - opener.top.hgt) : y);
    refSlide(name).left = x - 10;
    refSlide(name).top = y + 8;
    showSlide(name);
    }
  else { hideSlide(name); }
  }
```

helpDisplay()需要两个参数。一个是要隐藏或显示的层的名称，另外一个是用来确定究竟是显示还是隐藏的布尔值。这是判断隐藏或是显示的第一个判定。如果*action*为false，程序就简单了。只要调用hideSlide()，就可以用它来完成任务。如果*action*为true，就意味着应该显示层。只要调用showSliede()，对吗？不全对。它可有点儿复杂。

JavaScript技巧：不必点击，使用链接

有时候，你想链接到某个东西，只需将鼠标从链接上移过或移开，而不必点击。有两个方法防止因点击引发不想出现的结果：

在HREF属性中使用javascript: void(0)

void运算符将忽略任何动作的返回值，包括点击事件。你不一定要用0，但它是一个简单的操作数。后面你将会看到一个例子。

使用onClick="return false;"

返回false将取消HREF属性所指定的任何文档载入。你可以试试这样：

```
<A HREF="" onMouseOver="doSomething();" onClick="return false;">Do
Something</A>
```

不管HREF属性中是什么，文档都不会被载入。

管理链接位置

调用 showSlide()将完成任务，但是你可能会得到比计划要多的结果。还记得 *nav.html* 中的第 30～32 行代码吗？

```
helpWin = open('help/' + helpName + '.html', 'helpFile', 'width=' +
  top.wdh + ',height=' + top.hgt + ',left=100,top=100,scrollbars=no');
```

窗口的宽度是 300 像素。高度也是如此。现在回头查看一下 *help.js* 中的第 5 和第 6 行：

```
var helpWdh  = 200;
var helpHgt  = 200;
```

每一个层的宽和高都被设置为 200 像素。实际上，层的高度将按照其中内容的数量而增长，就像表格里的数据单元一样。不管怎么说，我们仍然需要一个参考值。很容易就可以判定链接在页面上的位置是否已经向右超了 100 个像素（实际上应该小于这个数，因为在文档显示的地方，300 个像素是用来表示窗口外部的宽度，而不是内部宽度），那么至少层有一部分将会超出显示区域。为了避免这种情况，我们在定位和显示层之前要进行一些数学计算。

它是这样运作的：如果鼠标箭头的水平坐标加上显示的层的宽度大于子窗口的可显示宽度，就通过一个偏移像素值将层往左边挪。考虑第 62 行的代码：

```
totalWidth = x + helpWdh;
```

变量 *totalWidth* 等于水平坐标加上层的宽度。现在，你可以看到我们在用变量来设置尺寸的时候要用变量 *helpWdh* 和 *helpHgt* 而不是诸如 200 的数字了。现在考虑第 64～65 行：

```
x = (totalWidth > opener.top.wdh ? x -
  (totalWidth - opener.top.wdh + 75) : x);
```

如果 *totalWidth* 大于子窗口的宽度（少于边框），水平坐标就需要调整。我们只将它减去 totalWidth 和子窗口宽度的差，来往左边挪动。这确保所有层的显示都是水平的。高度也是如此。你可以在第 63 行和第 66～67 行看到。如果你设置的 *helpHgt* 值相对偏低，然后创建带有大量文字的层，上面的功能可能就不起作用了。

JavaScript 技巧：没有 LAYER 标签的层

一直以来，你都看见DHTML层在IE中的形式是DIV标签，在Navigator中的形式是LAYER标签。实际上，LAYER标签很好用，但它并不是标准。所有来自万维网联盟的迹象表明，标准文档对象模型和IE所支持的很相似。考虑下面的代码：

```
<HTML>
<HEAD>
  <TITLE>DHTML Layer</TITLE>
<SCRIPT LANGUAGE="JavaScript1.2">
<!--
var action = true;
function display(name) {
  if (document.all) {
    var layerObj = eval("document.all." + name + ".style");
    var hide = "hidden";
    var show = "visible";
    }
  else {
    var layerObj = eval("document." + name);
    var hide = "hide";
    var show = "show";
    }
  layerObj.visibility = (action ?  hide : show);
  action = !action;
  }

//-->
</SCRIPT>
</HEAD>
<BODY>
<DIV ID="dhtml"
  STYLE="position:relative;background-color:#FFACEE;width:200;">
  This is a DHTML layer.
</DIV>
<BR>
<A HREF="javascript:display('dhtml');">Show/Hide</A>
</BODY>
</HTML>
```

这是文件*\ch11\layer.html*。和你所看到的一样，响应这段代码（它按照点击来隐藏和显示层）的Navigator和IE中都没有出现LAYER标签。尽管Navigator一般不允许访问文档对象模式的大部分元素，但它的确允许访问定位属性。

那么究竟哪一种最好？我为什么没有在其他章节使用这个方法呢？两者都可以很好地工作，但我个人更倾向于在Navigator中创建LAYER标签，在IE中创建DIV标签的genLayer()函数。重要的是你可以有其他的选择。两个都试试吧。看看哪一个为你干得更好。

应用程序扩展

这里所介绍的帮助应用程序可以用在大多数小型到中型的应用程序中。在应用程序水平以上，仍然可以实现更多的帮助性能。考虑下面的建议以改进你的在线帮助。

目录

有时候，用户要寻找和屏幕当前的显示内容无关的文件。让用户浏览的最简单的方法之一，是提供一个目录页面，列出到所有帮助文档的链接。可以用一个静态的HTML页面来完成，或者使用JavaScript靠一个数组来产生列表。

```
function showContents() {
  var helpDocs = ['background', 'multiselect', 'urldirectory'];
  var helpLinks = '<UL>';
  for (var i = 0; i < helpDocs.length; i++) {
    helpLinks += '<LI><A HREF="' + helpDocs[i] + '.html">' +
      helpDocs[i] + '</A>';
    }
  helpLinks = '</UL>';
  document.writeln(helpLinks);
  }
```

可搜索的帮助文件

如果你需要一些帮助文档，何不使用第一章中的搜索应用程序呢？这通常是一个满足用户交互需要的好方法。

提出问题

有时候用户恰好找不要他们要的东西。如果你手下有员工，可以考虑添加一个基于表单的Email页面，这样，用户就可以提出一些问题，并把它们发送给有资格的员工。

电话簿

如果你确实需要客户服务，可以提供一个电话号码和Email地址的列表，这样用户就能联系到某个相关人员。与基于表单的Email选项相同，这需要较多的资源。在

你把电话号码发布出来之前，要确保你有人员来接这些电话。人们会打来的。我已经试着让用户访问我的站点之后打电话给我，结果我的电话都拨不通了。

后记

我想自己写书还有一个原因。当我和你一样翻完了其他某一本 Web 书籍的最后一章，最后一页（它们的页数只有《圣经》、《战争与和平》可以媲美）时，你究竟得到了什么呢？一些附录和一些索引而已。那的确不错，但就像登上高峰之巅以后，向下俯瞰，四下里却空空如也。结束的意义何在呢？

如果你读到了这一页，那么你已经经历了一个漫长的路程，这是不能仅仅用你翻阅的页数来衡量的。回想一下你在书店第一次翻看这本书并决定买下来的时候，你是一个什么水平的 JavaScript 编程者。然后再想一想现在你所知道的东西。回味一下自己的发展进步，一步步接近绝顶，感觉真是不错。

当然，我们可以为自己的成功而沾沾自喜，但是我们不能由此而马放南山。技术是按照竞争的速度在不断变化的。只要这本书被束之高阁，开发者就可能已经着手于另外一批 JavaScript 扩展和技术了。我也打算尽快接触新的东西。如果你有了什么好的，请跟我打个招呼。网上见。

— Jerry Bradenbaugh
hotsyte@mail.serve.com

原书空白页

附录一

JavaScript 参考

本附录包含如下范围的 JavaScript 语法参考：

- 对象，方法和属性
- 顶层函数和属性
- 事件处理方法

一般说来，JavaScript是一个三元结构：核心，客户端，以及服务器端。核心JavaScript指的是那些在客户端和服务器端两方面都可以运用的功能。客户端和服务器端JavaScript除核心JavsScript之外还有特定环境的扩展。例如，客户端JavaScript包含 *window* 和 *document* 对象，还有一些服务器用不着的东西。同样，服务器端JavaScript 包含 File 对象。

这里的资源目前是客户端 JavaScript 1.3 和核心 JavaScript 1.4。以下页面的所有内容都是直接从 Netscape 站点搬过来的：

http://developer1.netscape.com:80/docs/manuals/js/core/jsref/index.htm
http://developer1.netscape.com:80/docs/manuals/js/client/jsref/index.htm

你可以根据下面的网址找到 Microsoft Jscript 资源：

http://msdn.microsoft.com/scripting/default.htm?/scripting/jscript/default.htm

你可以把这些作为一个简单快捷的参考，它总是指向Netscape和Microsoft资源中的最新内容。

浏览器兼容性

表A-1说明不同版本的Navigator和IE支持哪种版本的JavaScript。

表A-1 JavaScript兼容性

JavaScript版本	Navigator版本	IE版本
JavaScript 1.0	Navigator 2.0	IE 3.x
JavaScript 1.1	Navigator 3.0	
JavaScript 1.2	Navigator 4.0-4.05	IE 4.x - 5.0
JavaScript 1.3	Navigator 4.06-4.5	
JavaScript 1.4		

对象、方法和属性

这一部分将按描述、兼容性、属性和方法几个方面来详细说明JavaScript对象。

Anchor

文档中超文本链接的目标所在。使用HTML的A标签或者调用`String.anchor`方法，JavaScript运行时引擎创建一个*Anchor*对象，它和提供NAME属性的文档中每一个A标签相对应。它把这些对象放到*document.anchors*属性中的一个数组里。你可以通过索引这个数组来访问一个*Anchor*对象。

兼容性

JavaScript 1.0 / 客户端

方法概要

这个对象从`Object`中继承`watch`和`unwatch`方法。

Applet

HTML 的 <APPLET> 标签。JavaScript 运行时引擎创建一个 *Applet* 对象，和文档中的每一个 *applet* 相对应。它将这些对象放在 *document.applets* 属性中的一个数组里。你可以通过索引这个数组来访问一个 *Applet* 对象。

兼容性

JavaScript 1.1 / 客户端

属性概要

Applet 对象继承 Java 中 applet 的所有公共属性。属性如表 A-2 所列。

方法概要

Applet 对象继承 Java 的 applet 中所有的公共方法。表 A-3 列出了具体的方法。

表 A-2 Applet 属性

属性	描述	版本
constructor	指定创建一个对象原型的函数	1.1
index	对每一个用正则表达式匹配来创建的数组，从 0 开始建立索引	1.2
input	对每一个用正则表达式匹配来创建的数组，表示与正则表达式所匹配的字符串相反的原始字符串	1.2
length	表示一个数组中的元素个数	1.1
prototype	允许属性对所有对象的动作	1.1

表 A-3 Applet 方法

方法	描述	版本
concat	指定创建一个对象原型的函数	1.2
join	将数组中的所有元素连接成一个字符串	1.1
pop	从一个数组中删除最后一个元素，并将这个元素作为返回值	1.2

表 A-3 Applet 方法（续）

方法	描述	版本
push	将一个或多个元素加到一个数组的末尾，并返回数组的新长度	1.2
reverse	颠倒一个数组中元素的顺序：第一个元素变成最后一个，而最后一个变为第一个	1.2
shift	从一个数组中删除第一个元素，并返回这个元素	1.2
slice	在一个数组中摘取某个部分，并返回得到的新数组	1.2
sort	将一个数组中的元素分类	1.1
splice	在数组中添加和（或）删除元素	1.2
toSource	返回一个字面上表示所指定数组的数组；你可以用这个值来创建一个新的数组。不包括 Object.toSource 方法	1.3
toString	返回一个表示数组及其元素的字符串。不包括 Object.toString 方法	1.1
unshift	在数组前面添加一个或多个元素，并返回数组的新长度	1.2
valueOf	返回数组的原始值。不包括 Object.valueOf 方法	1.1

Area

定义一个图像区域作为图像映射。当用户点击这个区域的时候，区域的超文本链接内容就载入到它自己的目标窗口。*Area* 对象是一种 *Link* 对象。

兼容性

JavaScript 1.1 / 客户端

Array

用于处理数组。

兼容性

JavaScript 1.1 / 核心

Boolean

Boolean 对象是一个布尔值的对象包装。

兼容性

JavaScript 1.1 / 核心

属性概要

Boolean 属性如表 A-4 所列。

方法概要

Boolean 方法如表 A-5 所列。

表 A-4 Boolean 属性

属性	描述	版本
constructor	指定创建一个对象原型的函数	1.1
prototype	定义一个由所有 *Boolean* 对象共享的属性	1.1

表 A-5 Boolean 方法

方法	描述	版本
toSource	返回一个字面上表示特定 *Boolean* 对象的数组；你可以使用这个值来创建一个新的数组。不包括 Object.toSource 方法	1.3
toString	返回一个表示特定对象的字符串。不包括 Object.toString 方法	1.1
valueOf	返回一个布尔对象的原始值。不包括 Object.valueOf 方法	1.1

Button

HTML 表单上的按钮。

兼容性

JavaScript 1.0 / 客户端

事件处理方法

onBlur, onClick, onFocus, onMouseDown, onMouseUp

属性概要

属性如表 A-6 所列。

方法概要

表 A-7 列出了方法。另外，这个对象从 Object 中继承了 watch 和 unwatch 方法。

表 A-6 Button 属性

属性	描述	版本
form	指定包含 *Button* 对象的表单	1.0
name	表示 NAME 属性	1.0
type	表示 TYPE 属性	1.1
value	表示 VALUE 属性	1.0

表 A-7 Button 方法

方法	描述	版本
blur	将焦点从按钮上移开	1.0
click	在按钮上模仿一个鼠标点击	1.0
focus	将焦点放到按钮上	1.0
handleEvent	对指定的事件调用处理方法	1.2

Checkbox

HTML表单上的一个复选框。复选框是一个让用户只能将某个值设为开或关的锁定开关。

兼容性

JavaScript 1.0 / 客户端

事件处理方法

onBlur, onClick, onFocus

属性概要

属性如表 A-8 所列。

方法概要

表 A-9 列出了方法。另外，这个对象从 Object 中继承了 watch 和 unwatch 方法。

表 A-8 Checkbox 属性

属性	描述	版本
checked	表示复选框当前状态的布尔属性	
defaultChecked	表示 CHECKED 属性的布尔属性	
form	指定包含 checkbox 对象的表单	1.0
name	表示 NAME 属性	1.0
type	表示 TYPE 属性	1.1
value	表示 VALUE 属性	1.0

表 A-9 Checkbox 方法

方法	描述	版本
blur	将焦点从复选框上移开	1.0
click	在复选框上模仿一个鼠标点击	1.0
focus	将焦点放到复选框上	1.0
handleEvent	对指定的事件调用处理方法	1.2

Date

让我们来看看日期和时间。

兼容性

JavaScript 1.0 / 核心

属性概要

属性如表 A-10 所列。

方法概要

方法如表 A-11 所列。

表 A-10 Date 属性

属性	描述	版本
constructor	指定创建一个对象原型的函数	1.1
prototype	允许将属性加到一个 *Date* 对象上	1.1

表 A-11 Date 方法

方法	描述	版本
getDate	按指定的日期依照当地时间返回月份中的天数	1.0
getDay	按指定的日期依照当地时间返回星期中的天数	1.0
getFullYear	按指定的日期依照当地时间返回年份	1.3
getHours	依照当地时间返回指定日期中的小时数	1.0
getMilliseconds	依照当地时间返回指定日期中的毫秒数	1.3
getMinutes	依照当地时间返回指定日期中的分钟数	1.0
getMonth	依照当地时间返回指定日期中的月份	1.0
getSeconds	依照当地时间返回指定日期中的秒数	1.0
getTime	依照当地时间返回和指定日期相对应的时间的数值	1.0
getTimezoneOffset	对当前场所返回时区的分钟偏移量	1.0
getUTCDate	依照格林威治时间返回指定时间月份中的天数	1.3

表 A-11 Date 方法（续）

方法	描述	版本
getUTCDay	依照格林威治时间返回指定时间星期中的天数	1.3
getUTCFullYear	依照格林威治时间返回指定时间的年份	1.3
getUTCHours	依照格林威治时间返回指定时间的小时数	1.3
getUTCMilliseconds	依照格林威治时间返回指定时间的毫秒数	1.3
getUTCMinutes	依照格林威治时间返回指定时间的分钟数	1.3
getUTCMonth	依照格林威治时间返回指定时间的月份	1.3
getUTCSeconds	依照格林威治时间返回指定时间的秒数	1.3
getYear	依照当地时间返回指定时间的年份	1.0
parse	以日期字符串的形式返回从当地时间的 1970 年 1 月 1 日 00:00:00 以来的毫秒数	1.0
setDate	依照当地时间按指定日期设置月份中的天数	1.0
setFullYear	依照当地时间按指定日期设置完整的年份	1.3
setHours	依照当地时间按指定日期设置小时数	1.0
setMilliseconds	依照当地时间按指定日期设置毫秒数	1.3
setMinutes	依照当地时间按指定日期设置分钟数	1.0
setMonth	依照当地时间按指定日期设置月份	1.0
setSeconds	依照当地时间按指定日期设置秒数	1.0
setTime	依照当地时间设置 *Data* 对象的值	1.0
setUTCDate	依照格林威治时间但指定时间设置月份中的天数	1.3
setUTCFullYear	依照格林威治时间但指定时间设置完整的年份	1.3
setUTCHours	依照格林威治时间但指定时间设置小时数	1.3
setUTCMilliseconds	依照格林威治时间但指定时间设置毫秒数	1.3

表 A-11 Date 方法（续）

方法	描述	版本
setUTCMinutes	依照格林威治时间但指定时间设置分钟数	1.3
setUTCMonth	依照格林威治时间但指定时间设置月份	1.3
setUTCSeconds	依照格林威治时间但指定时间设置秒数	1.3
setYear	依照当地时间按指定时间设置年份	1.0
toGMTString	使用因特网 GMT（格林威治标准时间）协议将日期转换成字符串	1.0
toLocaleString	使用当前场所的协议将日期转换为字符串	1.0
toSource	返回一个字面上表示指定 *Date* 对象的对象；你可以用这个值来创建一个新的对象。不包括 Object.toSource 方法	1.3
toString	返回一个表示指定 *Date* 对象的字符串。不包括 Object.toString 方法	1.1
toUTCString	用格林威治时间协议将一个日期转换成字符串	1.3
UTC	返回从格林威治时间 1970 年 1 月 1 日 00:00:00 以来某个 *Date* 对象中的毫秒数	1.0
valueOf	返回一个 *Date* 对象的原始值。不包括 Object.valueOf 方法	1.1

Document

包含有关当前文档的信息，并提供方法，用来为用户显示 HTML 输出。

兼容性

JavaScript 1.0 / 客户端

事件处理方法

onClick, onDblClick, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseUp

属性概要

表 A-12 列出了属性。

方法概要

方法如表 A-13 所列。另外，这个对象从 Object 中继承 watch 和 unwatch 方法

表 A-12 Document 属性

属性	描述	版本
alinkColor	指定 ALINK 属性的字符串	1.0
anchors	包含一个文档中所有 anchor 条目的一个数组	1.0
applets	包含一个文档中所有 applet 条目的一个数组	1.1
bgColor	指定 BGCOLOR 属性的字符串	1.0
cookie	定义一个 cookie	1.0
domain	指定为某个文档服务的服务器的域名	1.1
embeds	包含文档中所有 plug-in（插件程序）条目的一个数组	1.1
fgColor	定义 TEXT 属性的一个字符串	1.0
formName	文档中每一个已命名表单的单独属性	1.1
forms	包含一个文档中所有表单条目的一个数组	1.1
images	包含一个文档中所有图片条目的一个数组	1.1
lastModified	指定文档最新修改日期的字符串	1.0
layers	包含一个文档内所有层条目的一个数组	1.2
linkColor	指定 LINK 属性的一个字符串	1.0
links	包含一个文档中所有链接的条目的一个数组	1.0
plugins	包含一个文档中所有 plug-in 的条目的一个数组	1.0
referrer	指定调用文档的 URL 的字符串	1.1
title	指定 TITLE 标签内容的一个字符串	1.0

表 A-12 Document属性（续）

属性	描述	版本
URL	指定文档完整 URL 的一个字符串	1.0
vlinkColor	指定 VLINK 属性的一个字符串	1.0

表 A-13 Document方法

方法	描述	版本
captureEvents	设置获取指定类型的所有事件的文档	1.2
close	关闭输出流并显示数据	1.0
getSelection	返回一个包含当前选择的文本的字符串	1.2
handleEvent	调用指定事件的处理方法	1.2
open	打开一个流来收集 write 和 writeln 方法的输出	1.0
releaseEvents	让窗口或者文档释放指定类型的捕获事件，将事件送到事件层较远的对象中	1.2
routeEvent	沿普通事件层次传递捕获的事件	1.2
write	在指定的窗口中将一个或多个 HTML 表达式写到文档中	1.0
writeln	在指定的窗口中将一个或多个 HTML 表达式写到文档中，并在后面加上一个换行符	1.0

Event

事件对象包含用来描述一个 JavaScript 事件的属性，并在事件发生时作为一个参数传递到一个事件处理方法中。

兼容性

JavaScript 1.2 / 客户端

属性概要

属性如表 A-14 所列。不是所有这些属性都和每一个事件类型有关。

表 A-14 Event 属性

属性	描述	版本
data	返回一个包含所在对象的 URL 的字符串数组。*DragDrop* 事件随同传递	1.2
height	表示窗口或框架的高度	1.2
layerX	这个数字用来指定传递调整大小事件时的对象宽度，或者事件发生层中的指针水平坐标像素值。注意，layerX 和 x 同义	1.2
layerY	这个数字用来指定传递调整大小事件时的对象高度，或者事件发生层中的指针垂直坐标像素值。注意，layerY 和 y 同义	1.2
modifiers	这个字符串用来指定一个鼠标或键盘事件的相应 modifier 关键词。Modifier 关键词的值为：ALT_MASK，CONTROL_MASK，SHIFT_MASK 和 META_MASK	1.2
pageX	这个数字用来指定与页面相关的指针水平位置像素值	1.2
pageY	这个数字用来指定与页面相关的指针垂直位置像素值	1.2
screenX	用来指定与屏幕相关的指针水平位置像素值的数字	1.2
screenY	用来指定与屏幕相关的指针竖直位置像素值的数字	1.2
target	用来表示对象最初所送到的事件（所有事件）的字符串	1.2
type	表示事件类型的字符串（所有事件）	1.2
which	这个数字用来描述点下的鼠标按钮，或者一个按键的 ASCII 值。对于鼠标，1 代表坐键，2 为中键，3 为右键	1.2
width	表示一个窗口或框架的宽度	1.2

表 A-14 Event 属性（续）

属性	描述	版本
x	和 layerX 同义	1.2
y	和 layerY 同义	1.2

FileUpload

兼容性

JavaScript 1.0 / 客户端

指HTML表单上的一个文件上载元素。文件上载元素允许用户提供一个文件作为输入。

事件处理方法

onBlur, onChange, onFocus

属性概要

属性如表 A-15 所列。

方法概要

方法如表 A-16 所列。另外，这个对象从 Object 中继承了 watch 和 unwatch 方法

表 A-15 FileUpload 属性

属性	描述	版本
form	指定包含 *FileUpload* 对象的表单	1.0
name	表示 NAME 属性	1.0
type	表示 TYPE 属性	1.1
value	表示文件上载元素域的当前值；这和上载的文件名有关	1.0

表 A-16 FileUpload 方法

方法	描述	版本
blur	从对象上移开焦点	1.0
focus	将焦点设置给对象	1.0
handleEvent	对指定事件调用处理方法	1.2
select	选择文件上载域的输入区域	1.0

Form

允许用户输入文字，并从诸如复选框，单选钮和选择列表的Form元素中做出选择。你也可以用一个表单来将数据送到服务器。

兼容性

JavaScript 1.0 / 客户端

事件处理方法

onReset, onSubmit

属性概要

属性如表 A-17 所列。

方法概要

方法如表 A-18 所列。

表 A-17 Form 属性

属性	描述	版本
action	表示 ACTION 属性	1.0
elements	表示所有表单元素的一个数组	1.0
encoding	表示 ENCTYPE 属性	1.0
length	表示一个表单中的元素数目	1.0
method	表示 METHOD 属性	1.0
name	表示 NAME 属性	1.0
target	表示 TARGET 属性	1.0

表 A-18 Form 方法

方法	描述	版本
handleEvent	对指定事件调用事件处理方法	1.2
reset	在调用表单的 reset 按钮上模拟鼠标点击	1.1
submit	提交表单	1.0

Frame

窗口可以在单个的屏幕上显示多样的、能滚动的独立框架，每一个框架有它自己的特定的URL。这些框架是用 <FRAMESET> 标签中的FRAME标签创建的。框架可以指向不同的URL，或作为其他URL的目标，所有的一切都在相同的屏幕中。一系列框架构成一个页面。在考虑构建这些框架的对象时，*Frame* 对象是一个便利的工具。无论如何，JavaScript 实际上用一个 *window* 对象来表示一个框架。每一个 *Frame* 对象都是一个 *window* 对象，拥有一个 *window* 对象应有的所有属性和方法。框架的窗口和一个顶层窗口之间，有一点点不同。你可以看看 window 以获得有关于框架的全部信息。

兼容性

JavaScript 1.0 / 客户端

Function

指定 JavaScript 代码的一个字符串，将它作为一个函数来编译

兼容性

JavaScript 1.1 / 核心

属性概要

属性如表 A-19 所列。

方法概要

方法如表 A-20 所列。

表 A-19 Function 属性

属性	描述	版本
arguments	传递给函数的参数的相应数组	1.1
arguments.callee	指定当前执行函数的函数体	1.2
arguments.challer	指定调用当前所执行函数的函数名称	1.1
arguments.length	指定传递给函数的参数个数	1.1
arity	指定函数所期望的参数个数	1.2
constructor	指定创建对象原型的函数	1.1
length	指定函数所期望的参数个数	1.1
prototype	允许将一个属性添加到 *Function* 对象中	1.1

表 A-20 Function 方法

方法	描述	版本
apply	可用于在一个对象（产生调用的对象）的上下文中申请使用另外一个对象的方法	1.3
call	可用于在一个对象（产生调用的对象）的上下文中调用（执行）另外一个对象的方法	1.3
toSource	返回一个表示函数源代码的字符串。不包括 Object.toSource 方法	1.3
toString	返回一个表示函数源代码的字符串。不包括 Object.toSrring 方法	1.1
valueOf	返回一个表示函数源代码的字符串。不包括 Object.Value 方法	1.1

Hidden

在显示 HTML 表单的时候不显示出来的 Text 对象。一个 Hidden 对象被用来在提交表单的时候传递一个名 / 值对。

兼容性

JavaScript 1.0 / 客户端

属性概要

属性如表 A-21 所列。

方法概要

这个对象继承了 Object 中的 watch 和 unwatch 方法。

表 A-21 Hidden 属性

属性	描述	版本
form	指定包含 *Hidden* 对象的表单	1.0
name	表示 NAME 属性	1.0
type	表示 TYPE 属性	1.1
value	表示 *Hidden* 对象的当前值	1.0

History

包含客户在一个窗口中访问过的URL信息数组。这个信息被存储在一个历史记录列表中，可以通过浏览器的 Go 菜单访问到。

兼容性

JavaScript 1.0 / 客户端

属性概要

属性如表 A-22 所示。

方法概要

方法如表 A-23 所列。这个对象从 Object 中继承了 watch 和 unwatch 方法。

表 A-22 History 属性

属性	描述	版本
current	指定当前历史记录条目的 URL	1.1
length	表示历史记录列表中的条目数量	1.0
next	指定下一个历史记录条目的 URL	1.1
previous	指定上一个历史记录条目的 URL	1.1

表 A-23 History 方法

方法	描述	版本
back	载入历史记录列表中上一个 URL	1.0
forward	载入历史记录列表中下一个 URL	1.0
go	从历史记录列表中载入 URL	1.0

Image

在 HTML 表单中指向一个图片。

兼容性

JavaScript 1.1 / 客户端

事件处理方法

onAbort, onError, onKeyDown, onKeyPress, onKeyUp, onLoad

属性概要

属性如表 A-24 所列。

方法概要

方法如表 A-25 所示。这个对象从 Object 中继承了 watch 和 unwatch 方法。

表 A-24 Image 属性

属性	描述	版本
border	表示 BORDER 属性	1.1
complete	指示 Web 浏览器是否已经完成图片载入的逻辑值	1.1
height	表示 HEIGHT 属性	1.1
hspace	表示 HSPACE 属性	1.1
lowsrc	表示 LOWSRC 属性	1.1
name	表示 NAME 属性	1.1
src	表示 SRC 属性	1.1

表 A-24 Image 属性（续）

属性	描述	版本
vspace	表示 VSPACE 属性	1.1
width	表示 WIDTH 属性	1.1

表 A-25 Image 方法

方法	描述	版本
handleEvent	对指定事件调用事件处理方法	1.2

Java

用于访问*java.**包中任何 Java 类的顶层对象。*java* 对象是 *Packages.java* 属性的方便工具。

兼容性

JavaScript 1.1

JavaArray

从 JavaScript 代码中来访问的包装好的 Java 数组，它是 JavaArray 类型的成员。

兼容性

JavaScript 1.1 / 核心

属性概要

属性如表 A-26 所列。

方法概要

方法如表 A-27 所列。

表 A-26 JavaArray 属性

属性	描述	版本
length	用 JavaArray 表示的 Java 数组的元素个数	1.1

表 A-27 JavaArray 方法

方法	描述	版本
toString	在 JavaScript 1.4 中，这个方法被继承的 java.lang.Object.toString 方法所覆盖。在 JavaScript 1.3 及更早的版本中，这个方法返回一个字符串，将此对象作为一个 *JavaArray*	1.1

JavaClass

一个 Java 类的 JavaScript 指示。一个 *JavaClass* 对象指示 Java 包中的一个类，比如 netscape.javascript.JSObject。一个 *JavaPackage* 对象指示一个 Java 包，比如 netscape.javascript。在 JavaScript 中，JavaPackage 和 JavaClass 层表示 Java 包和类的层。

兼容性

JavaScript 1.1 / 核心

属性概要

JavaClass 对象的属性是 Java 类中的静态字段。

方法概要

JavaClass 对象的方法是 Java 类中的静态方法。

JavaObject

是预先包装的 Java 对象的类型，从 JavaScript 代码内进行访问。*JavaObject* 对象是在 JavaScript 内创建的或传递给 JavaScript 的 Java 类的一个实例。JavaObject 是实例的包装；所有对类实例的指示都通过 JavaObject 来建立。任何带到 JavaScript 中的数据都转换成 JavaScript 数据类型。当 *JavaObject* 传回 Java 的时候，它被卸装，然后可以由 Java 代码来使用。

兼容性

JavaScript 1.1 / 核心

属性概要

JavaPackage 属性是它所包含的 *JavaClass* 对象和任何其他 *JavaPackage* 对象。

JavaPackage

在Java中，包是Java类或其他Java包的集合。例如，netscape包就包含了netscape.javascript包；netscape.javascript包又包含JSObject和JSException类。

在JavaScript中，一个JavaPackage是对Java包的一个引用。例如，对netscape的引用是一个JavaPackage。netscape.javascript是一个JavaPackage，同时也是netscape JavaPackage的一个属性。一个*JavaClass*对象是对包中某个类的引用，比如netscape.javascript.JSObject。JavaPackage和JavaClass层表示Java包和Java类的层。虽然JavaPackage中所包含的这些包和类是它的属性，但你不能像列举其他对象的属性那样使用for...in语句来列举它们。

兼容性

JavaScript 1.1 / 核心

属性概要

JavaPackage属性是它所包含的*JavaClass*对象和任何其他*JavaPackage*对象。

Layer

和HTML页面中的一个层相对应，并提供操作那个层的方法。

兼容性

JavaScript 1.2 / 客户端

属性概要

属性如表A-28所列。

方法概要

方法如表A-29所列。这个对象继承了Object中的watch和unwatch方法。

表A-28 Layer属性

属性	描述	版本
above	文档里所有层中按z轴顺序在这个层上面的一层，如果这个层已经是顶层，就是装入窗口对象	1.2

表 A-28 Layer 属性（续）

属性	描述	版本
background	用来作层背景的图片	1.2
bgColor	用来作为层背景色的固定颜色	1.2
below	文档里所有层中按 z 轴顺序在这个层下面的一层，如果这个层已经是最低层，就是装入窗口对象	1.2
clip.bottom	裁剪矩形的下边缘（层的一个可见部分）	1.2
clip.height	裁剪矩形的高度（层的一个可见部分）	1.2
clip.left	裁剪矩形的左边缘（层的一个可见部分）	1.2
clip.right	裁剪矩形的右边缘（层的一个可见部分）	1.2
clip.top	裁剪矩形的上边缘（层的一个可见部分）	1.2
clip.width	裁剪矩形的宽度（层的一个可见部分）	1.2
document	层的相关文档	1.2
left	层左边缘的水平位置像素值，和其父层的原点有关	1.2
name	一个指定层名称的字符串，它是通过 LAYER 标签的 ID 属性来赋值的	1.2
pageX	层的水平位置像素值，和页面有关	1.2
pageY	层的垂直位置像素值，和页面有关	1.2
parentLayer	包含这个层的层对象，或者如果这个层没有嵌套在其他层中的话，则为装入的窗口对象	1.2
siblingAbove	文档里所有有相同父层的层中按 z 轴顺序在这个层上面的一层，如果这个层没有兄弟就为 null	1.2
siblingBelow	文档里所有有相同父层的层中按 z 轴顺序在这个层下面的一层，如果这个层已经是最底层就为 null	1.2
src	指定层的内容的 URL 的字符串	1.2

表 A-28 Layer 属性（续）

属性	描述	版本
top	层的上边缘的纵向位置的像素值，和父层的原点有关	1.2
visibility	指定层是否可见	1.2
zIndex	同时考虑层的兄弟层而确定的层的 z 轴顺序	1.2

表 A-29 Layer 方法

方法	描述	版本
captureEvents	使窗口或文档捕获指定类型的所有事件	1.2
handleEvent	对指定事件调用事件处理方法	1.2
load	将层的来源改为指定文件的内容，同时修改层的 HTML 内容所要用的宽度	1.2
moveAbove	将此层堆叠在参数所指定的层之上，不改变层的水平和垂直坐标	1.2
moveBelow	将此层堆叠在参数所指定的层之下，不改变层的水平和垂直坐标	1.2
moveBy	通过应用指定的像素值来改变层的位置	1.2
moveTo	将窗口的左上角移动到指定的屏幕坐标处	1.2
moveToAbsolute	将层的位置改为页面内（而不是所包含的层）指定的像素坐标	1.2
releaseEvents	使层释放所捕获的指定类型的事件，将事件送到事件层中更远的对象	1.2
resizeBy	用指定的高度和宽度值（像素值）调整层的大小	1.2
resizeTo	将层调整为指定的高度和宽度值（像素值）	1.2
routeEvent	沿普通事件层传递一个捕获的事件	1.2

Link

通过使用HTML的A或AREA标签，或者通过对String.link方法的调用来实现。JavaScript运行时间引擎创建一个*Link*对象，它与提供HREF属性的文档中每个A和AREA标签相对应。它将这些对象作为一个数组放在*document.links*属性中。你可以通过索引这个数组而访问一个*Link*对象。

兼容性

JavaScript 1.0 / 客户端

事件处理方法

Area对象有如下的事件处理方法：onDblClick, onMouseOut, onMouseOver

Link对象有如下的事件处理方法：onClick, onDblClick, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseOut, onMouseUp, onMouseOver

属性概要

属性如表A-30所列。

方法概要

方法如表A-31所列。这个对象从Object中继承了watch和unwatch方法。

表A-30 Link属性

属性	描述	版本
hash	指定URL中一个锚的名称	1.0
host	指定一个网络主机的主机名和域名，或IP地址	1.0
hostname	指定URL的host:port部分	1.0
href	指定完整的URL	1.0
pathname	指定URL的URL路径部分	1.0
port	指定服务器所用的通信部分	1.0
protocol	指定URL的开始部分，包括冒号在内	1.0
search	指定一个查询字符串	1.0
target	表示TARGET属性	1.0
text	一个包含A标签的相应内容的字符串	1.0

表 A-31 Link 方法

方法	描述	版本
handleEvent	调用指定事件的处理	1.2

Location

包含当前 URL 的信息。

兼容性

JavaScript 1.0 / 客户端

属性概要

属性如表 A-32 所列。

方法概要

方法如表 A-33 所列。这个对象从 Object 中继承了 watch 和 unwatch 方法。

表 A-32 Location 属性

属性	描述	版本
hash	指定 URL 中一个锚的名称	1.0
host	指定一个网络主机的主机名和域名，或 IP 地址	1.0
hostname	指定 URL 的 host:port 部分	1.0
href	指定完整的 URL	1.0
pathname	指定 URL 的 URL 路径部分	1.0
port	指定服务器所用的通信部分	1.0
protocol	指定 URL 的开始部分，包括冒号在内	1.0
search	指定一个查询字符串	1.0

表 A-33 Location 方法

方法	描述	版本
reload	在当前历史记录条目上载入指定的 URL	1.1
replace	强令执行窗口中当前文档的重载入	1.1

Math

一个嵌入式的对象，它的属性和方法包含有精确的常量和方法。例如，*Math* 对象的 *PI* 属性是 *pi* 的值。

兼容性

JavaScript 1.0 / 核心

属性概要

属性如表 A-34 所列。

方法概要

方法如表 A-35 所列。

表 A-34 Math 属性

属性	描述	版本
E	欧拉常数，自然对数的底数，大约为 2.718	1.0
LN10	10 的自然对数，大约是 2.302	1.0
LN2	2 的自然对数，大约是 0.693	1.0
LOG10E	底数为 10 的 E 的对数，大约是 0.434	1.0
LOG2E	底数为 2 的 E 的对数，大约是 1.442	1.0
PI	圆周率，大约是 3.14159	1.0
SQRT1_2	1/2 的平方根；即 1 除以 2 的平方根，大约是 0.707	1.0
SQRT2	2 的平方根，大约为 1.414	1.0

表 A-35 Math 方法

方法	描述	版本
abs	返回一个数的绝对值	1.0
acos	返回一个数的弧度余弦（弧度）	1.0
asin	返回一个数的弧度正弦（弧度）	1.0
atan	返回一个数的弧度正切（弧度）	1.0

表 A-35 Math 方法（续）

方法	描述	版本
atan2	返回参数之商的弧度正切	1.0
ceil	返回大于或等于一个数的最小整数	1.0
cos	返回一个数的余弦	1.0
exp	返回 E 作为数字的幂，数字为参数，E 是欧拉常量，自然对数的底数	1.0
floor	返回小于或等于一个数的最大整数	1.0
log	返回一个数的自然对数（E 为底数）	1.0
max	返回两个数中的较大值	1.0
min	返回两个数中的较小值	1.0
pow	为指数幂返回基数，也就是指数幂的基数	1.0
random	返回 0 到 1 之间的一个模拟随机数	1.0
round	返回一个数最相近的整数舍入值	1.0
sin	返回一个数的正弦	1.0
sqrt	返回一个数的平方根	1.0
tan	返回一个数的正切	1.0

MimeType

一个客户端支持的 MIME 类型（Multipurpose Internet Mail Extension，多用途因特网邮件扩展）。

兼容性

JavaScript 1.1 / 客户端

属性概要

属性如表 A-36 所列。

方法概要

这个对象从 Object 中继承了 watch 和 unwatch 方法。

表 A-36 MimeType 属性

属性	描述	版本
description	一个 MIME 类型的描述	1.0
enabledPlugin	引用为 MIME 类型而设定的 *Plugin* 对象	1.0
suffixes	一个列出 MIME 类型可能的文件扩展名的字符串，例如："mpeg，mpg，mpe，mpv，vbs，mpegv"	1.0
type	MIME 类型的名称，例如，"video/mpeg"或"audio/x-wav"	1.0

Navigator

用于处理数字的值。*Number* 对象是原始数值的对象包装。

兼容性

JavaScript 1.0 / 客户端

属性概要

属性如表 A-37 所列。

方法概要

方法如表 A-38 所列。这个对象从 Object 中继承了 watch 和 unwatch 方法。

表 A-37 Navigator 属性

属性	描述	版本
appCodeName	指定浏览器名称的代码	1.0
appName	指定浏览器名称	1.0
appVersion	指定 Navigator 的版本信息	1.0
language	表示所用 Navigator 的语言	1.2
mimeTypes	一个包含所有客户端所支持的 MIME 类型的数组	1.1
platform	表示编译 Navigator 的机器类型	1.2

表 A-37 Navigator 属性（续）

属性	描述	版本
plugins	一个包含客户机中当前所装的所有 plug-in 的数组	1.1
userAgent	表示用户代理的标题	1.1

表 A-38 Navigator 方法

方法	描述	版本
javaEnabled	测试是否支持 Java	1.1
plugins.refresh	使最近所装的 plug-in 可用，并任意重载包含 plug-in 的打开文档	1.1
preference	允许一个带符号 script 获得或设置一定的 Navigator 参数选择	1.2
taintEnabled	指定数据是否可用	1.1

Netscape

一个用来访问 netscape.* 包中任何 Java 类的顶层对象。*netscape* 对象是顶层的，预定义的 JavaScript 对象。你可以自动地访问它，而不需要使用构造器或方法。

兼容性

JavaScript 1.1 / 核心

Number

用于处理数字值。*Number* 对象是原始数值的对象包装。

兼容性

JavaScript 1.1 / 核心

属性概要

属性如表 A-39 所列。

方法概要

方法如表 A-40 所列。

表 A-39 Number 属性

属性	描述	版本
constructor	指定创建对象原型的函数	1.1
MAX_VALUE	最大可表示数	1.1
MIN_VALUE	最小可表示数	1.1
NaN	表示“非数字”值	1.1
NEGATIVE_INFINITY	表示负无穷；返回堆栈溢出	1.1
POSITIVE_INFINITY	表示无穷大；返回堆栈溢出	1.1
prototype	允许将属性加到 *Number* 对象中	1.1

表 A-40 Number 方法

方法	描述	版本
toSource	返回一个表示指定 *Number* 对象的对象；你可以用这个值来创建一个新的对象。不包括 Object.toSource 方法	1.3
toString	返回表示指定对象的一个字符串。不包括 Object.toString 方法	1.1
valueOf	返回指定对象的原始值。不包括 Object.valueOf 方法	1.1

Object

Object 是 JavaScript 的原始对象类型。所有的 *JavaScript* 对象都继承 Object。也就是说，所有的 JavaScript 对象拥有为 Object 所定义的方法。

兼容性

JavaScript 1.0 / 核心

属性概要

属性如表 A-41 所列。

方法概要

方法如表 A-42 所列。

表 A-41 Object 属性

属性	描述	版本
constructor	指定创建对象原型的方法	1.1
prototype	允许将属性加到所有的对象中	1.1

表 A-42 Object 方法

方法	描述	版本
eval	不是很重要。对指定对象内容中 JavaScript 代码的一个字符串求值	1.1
toSource	返回一个表示指定对象的对象直接量；可以用这个值来创建一个新的对象	1.3
toString	返回表示指定对象的一个字符串	1.0
unwatch	从对象的属性中删除一个视点（watchpoint）	1.2
valueOf	返回指定对象的原始值	1.1
watch	在对象属性中添加一个视点	1.2

Option

对应于 SELECT 列表中的一个选项。

兼容性

JavaScript 1.1 / 客户端

属性概要

属性如表 A-43 所列。

方法概要

方法如表 A-44 所列。这个对象从 Object 中继承了 watch 和 unwatch 方法。

表 A-43 Option 属性

属性	描述	版本
defaultSelected	指定选项的初始选择状态	1.1
selected	指定选项的当前选择状态	1.1
text	指定选项的文字	1.1
value	指定在选中了选项且提交表单的时候返回给服务器的值	1.1

表 A-44 Option 方法

方法	描述	版本
reload	在当前历史记录条目上载入指定的 URL	1.1
replace	强令执行窗口中当前文档的重载入	1.1

Packages

一个用来从 JavaScript 代码内访问 Java 类的顶层对象。

兼容性

JavaScript 1.1 / 核心

属性概要

属性如表 A-45 所示。

表 A-45 Packages 属性

属性	描述	版本
ClassName	除 netscape，java 或 sun 以外的支持 JavaScript 的包中 Java 类的完整合格名称	1.1
java	Java 包 java.* 中所有的类	1.1
netscape	Java 包 netscape.* 中所有的类	1.1
sun	Java 包 sun.* 中所有的类	1.1

Password

HTML表单中的一个文本域，它通过显示星号（*）来隐藏本身的值。但用户在域中输入文字时，星号（*）隐藏它们的显示。

兼容性

JavaScript 1.0 / 客户端

事件处理方法

onBlur, onFocus

属性概要

属性如表 A-46 所列。

方法概要

方法如表 A-47 所列。这个对象从 Object 中继承 watch 和 unwatch 方法。

表 A-46 Password 属性

属性	描述	版本
defaultValue	表示 VALUE 属性	1.0
form	指定包含 *Password* 对象的表单	1.0
name	表示 NAME 属性	1.0
type	表示 TYPE 属性	1.1
value	表示 *Password* 对象域的当前值	1.0

表 A-47 Password 方法

方法	描述	版本
blur	从对象上移开焦点	1.0
focus	将焦点置给对象	1.0
handleEvent	调用指定事件的处理	1.2
select	选择对象的输入域	1.0

Plugin

一个装入客户端的插件（plug-in）模块。*Plugin*对象是预先确定的*JavaScript*对象，你可以通过*navigator.plugins*数组来访问它。一个*Plugin*对象是装入客户端的插件。插件是一种软件模块，浏览器可以调用它来显示嵌入浏览器内的数据的特定类型。

兼容性

JavaScript 1.1 / 客户端

属性概要

属性如表 A-48 所列。

表 A-48 Plugin 属性

属性	描述	版本
description	一个插件的描述	1.1
filename	磁盘中插件文件的名称	1.1
length	*MimeType* 对象的插件的数组元素个数	1.1
name	插件的名称	1.1

这个对象从 Object 中继承了 watch 和 unwatch 方法。

Radio

HTML表单中一系列单选按钮中的一个单选按钮。用户可以使用一系列的单选按钮来从一个列表中选择一个项目。

兼容性

JavaScript 1.0 / 客户端

属性概要

属性如表 A-49 所列。

方法概要

方法如表 A-50 所列。这个对象从 Object 中继承了 watch 和 unwatch 方法。

表 A-49 Radio 属性

属性	描述	版本
checked	选择一个单选按钮	1.0
defaultChecked	表示 CHECKED 属性	1.0
form	指定包含 Radio 对象的表单	1.0
name	表示 NAME 属性	1.0
type	表示 TYPE 属性	1.1
value	表示 VALUE 属性	1.0

表 A-50 Radio 方法

方法	描述	版本
blur	从对象上移开焦点	1.1
click	在单选按钮上模拟鼠标点击	1.0
focus	将焦点设置给对象	1.1
handleEvent	调用指定事件的处理	1.2

RegExp

一个包含正则表达式模式的正则表达式对象。它有属性和方法，用正则表达式来查找和替换字符串中的匹配。除了可以用 RegExp 构造函数来创建的单个正则表达式对象外，预先确定的*RegExp*对象拥有静态属性，它在任何正则表达式被使用时得到设置。

兼容性

JavaScript 1.2 / 核心

属性概要

属性如表 A-51 所列。

方法概要

方法如表 A-52 所列。

表 A-51 RegExp 属性

属性	描述	版本
$1, ..., $9	将匹配子串加上括号	1.2
$_	同 input	1.2
$*	同 multiline	1.2
$&	同 lastMatch	1.2
$+	同 lastParen	1.2
$`	同 leftContext	1.2
$'	同 rightContext	1.2
constructor	指定创建对象原型的函数	1.2
global	是测试正则表达式是否有字符串中所有可能的匹配，还是只测试第一个	1.2
ignoreCase	在字符串中查找匹配的时候是否忽略情况	1.2
input	一个正则表达式所匹配的字符串	1.2
lastIndex	下一个匹配从哪个项目开始的索引	1.2
lastMatch	最后一个匹配的字符	1.2
lastParen	最后一个加括号的子串匹配	1.2
leftContext	先于最近匹配的子串	1.2
multiline	是否越行在串中搜索字符串	1.2
prototype	允许将属性加到所有的对象中	1.1
rightContext	跟在最近匹配后面的子串	1.2
source	模式的文本	1.2

表 A-52 RegExp 方法

方法	描述	版本
compile	编译一个正则表达式对象	1.2
exec	在它的字符串参数中执行一个匹配搜索	1.2

表 A-52 RegExp 方法（续）

方法	描述	版本
test	在它的字符串参数中测试一个匹配	1.2
toSource	返回一个表示指定对象的对象；你可以用这个值来创建一个新的对象。不包括 Object.toSource 方法	1.3
toString	返回一个表示指定对象的字符串。不包括 Object.toString 方法	1.1
valueOf	返回指定对象的参数值。不包括 Object.valueOf 方法	1.1

Reset

HTML 表单中的一个 reset 按钮。一个 reset 按钮将表单中的所有元素返回到它们的初始值。

兼容性

JavaScript 1.0 / 客户端

事件处理方法

onBlur, onClick, onFocus

属性概要

属性如表 A-53 所列。

方法概要

方法如表 A-54 所列。这个对象从 Object 中继承了 watch 和 unwatch 方法。

表 A-53 Reset 属性

属性	描述	版本
form	指定包含 *Reset* 对象的表单	1.0
name	表示 NAME 属性	1.0
type	表示 TYPE 属性	1.1
value	表示 VALUE 属性	1.0

表 A-54 Reset 方法

方法	描述	版本
blur	从对象上移开焦点	1.0
click	在 reset 按钮上模拟鼠标点击	1.0
focus	将焦点设置给对象	1.0
handleEvent	调用指定事件的处理	1.2

Screen

兼容性

JavaScript 1.0/ 客户端

属性概要

属性如表 A-55 所列。

表 A-55 Screen 属性

属性	描述	版本
availHeight	指定屏幕的高度像素值，减去操作系统显示的固定和半固定的用户界面部件，比如 Microsoft Windows 上的任务栏	1.2
availWidth	指定屏幕的宽度像素值，减去操作系统显示的固定和半固定的用户界面部件，比如 Microsoft Windows 上的任务栏	1.2
colorDepth	调色板的位深，如果使用了某一个；否则，这个值来源于 screen.pixelDepth	1.2
height	显示屏幕高度	1.2
pixelDepth	显示屏幕颜色设置（每个像素的位值）	1.2
width	显示屏幕宽度	1.2

Select

HTML表单中的一个选择列表。用户可以按照列表的创建形式从选择列表中选一项或多项。

兼容性

JavaScript 1.0 / 客户端

事件处理方法

onBlur, onChange, onFocus

属性列表

属性如表 A-56 所列。

方法概要

方法如表 A-57 所列。这个对象从 Objecet 中继承了 watch 和 unwatch 方法。

表 A-56 Select 属性

属性	描述	版本
form	指定包含选择列表的表单	1.0
length	表示选择列表中选项的数目	1.0
name	表示 NAME 属性	1.0
options	表示 OPTION 标签	1.0
selectedIndex	表示所选项的索引（如果选了多项，则为第一个选项）	1.0
type	指示对象表示一个选择列表，以及它是否可以有一个或多个被选项	1.1

表 A-57 Select 方法

方法	描述	版本
blur	从选择列表上移开焦点	1.0
focus	将焦点设置给对象	1.0
handleEvent	调用指定事件的处理	1.2

String

在一个字符串中表示一系列字符的对象。

兼容性

JavaScript 1.0 / 核心

属性概要

属性如表 A-58 所列。

方法概要

方法如表 A-59 所列。这个对象从 Object 中继承了 watch 和 unwatch 方法。

表 A-58 String 属性

属性	描述	版本
constructor	指定创建对象原型的函数	1.1
length	表示字符串的长度	1.0
prototype	允许将属性添加到 *String* 对象	1.1

表 A-59 String 方法

方法	描述	版本
anchor	创建一个作为超文本目标的锚	1.0
big	使字符串以大字体显示，如同在 <BIG> 标签中	1.0
blink	使字符串闪烁显示，如同在 <BLINK> 标签中	1.0
bold	使字符串以粗体显示，如同在 <B> 标签中	1.0
charAt	返回指定索引的字符	1.0
charCodeAt	按所给出的索引返回一个表示字符的双字节编码的数	1.2
concat	合并两个字符串的文本，返回一个新的字符串	1.2
fixed	使字符串以固定斜度显示，如同在 <TT> 标签中	1.0
fontcolor	使字符串按指定颜色显示，如同在 <FONT COLOR=color> 标签中	1.0

表 A-59 String 方法（续）

方法	描述	版本
fontsize	使字符串按指定大小显示，如同在 <FONT SIZE=size> 标签中	1.0
fromCharCode	返回一个用指定双字节编码顺序值创建的字符串	1.2
indexof	返回指定值下首先发生的对 *String* 对象的调用中的索引，如果没有发现，就返回 -1	1.0
italics	使字符串显示为斜体，如同在 <I> 标签中	1.0
lastindexOf	返回指定值下最后发生的对 *String* 对象的调用中的索引，如果没有发现，就返回 -1	1.0
link	创建一个指向另外一个 URL 的 HTML 超文本链接	1.0
match	用来比较一个正则表达式和一个字符串	1.2
replace	用来在一个正则表达式和一个字符串之间查找匹配，并用一个新的子串替代匹配的子串	1.2
search	在一个正则表达式和一个指定字符串之间执行匹配搜索	1.2
slice	从字符串中取出一部分，返回一个新的字符串	1.0
small	使字符串以小字体显示，如同在 <SMALL> 标签中	1.0
split	通过将字符串分离成一些子串而将 *String* 对象分割成一个数组	1.0
strike	使字符串以拉长字体显示，如同在 <STRIKE> 标签中	1.0
sub	使字符串显示为下标，如同在 <SUB> 标签中	1.0
substr	返回一个字符串中从指定位置开始指定长度的字符	1.0
substring	将一个字符串中两指定位置之间的字符返回为一个字符串	1.0

表 A-59 String 方法（续）

方法	描述	版本
sup	使字符串显示为上标，如同在 <SUP> 标签中	1.0
toLowerCase	返回转换为小写的字符串调用值	1.0
toSource	返回一个表示指定对象的字符串；你可以用这个值创建一个新的对象，不包括 Object.toString 方法	1.3
toString	返回一个表示指定对象的对象；不包括 Object.valueOf 方法	1.1
toUpperCase	返回转换成大写的字符串调用值	1.0
valueOf	返回指定对象的原始值。不包括 Object.valueOf 方法	1.1

Submit

和 HTML 表单中的“submit”按钮相对应。表单通过 submit 按钮送到服务器。

兼容性

JavaScript 1.0 / 客户端

事件处理方法

onBlur, onClick, onFocus

属性概要

属性如表 A-60 所列。

方法概要

方法如表 A-61 所示。这个对象从 Object 继承了 watch 和 unwatch 方法。

表 A-60 Submit 属性

属性	描述	版本
form	指定包含 Submit 对象的表单	1.0
name	表示 NAME 属性	1.0

表 A-60 Submit 属性（续）

属性	描述	版本
type	表示 TYPE 属性	1.1
value	表示 VALUE 属性	1.0

表 A-61 Submit 方法

方法	描述	版本
blur	从对象上移开焦点	1.0
click	在 submit 按钮上模拟鼠标点击	1.0
focus	将焦点设置给对象	1.0
handleEvent	调用指定事件的处理	1.2

sun

用来访问 sun.* 包中任何 Java 类的顶层对象。*sun* 对象是顶层的，预先确定的 *JavaScript* 对象。你可以自动访问到它而不必用构造函数或者函数调用。sun对象和 Packages.sun 等价。

兼容性

JavaScript 1.1 / 核心

Text

HTML 表单中一个文本输入域。用户可以在文本域中输入一个单词、短语或一系列数字。

兼容性

JavaScript 1.0 / 客户端

事件处理方法

onBlur, onChange, onFocus, onSelect

属性概要

属性如表 A-62 所列。

方法概要

方法如表 A-63 所列。这个对象从 Object 中继承了 watch 和 unwatch 方法。

表 A-62 Text 属性

属性	描述	版本
defaultValue	表示 VALUE 属性	1.0
form	指定包含 Text 对象的表单	1.0
name	表示 NAME 属性	1.0
type	表示 TYPE 属性	1.1
value	表示 Text 对象域的当前值	1.0

表 A-63 Text 方法

方法	描述	版本
blur	从对象上移开焦点	1.0
focus	将焦点设置给对象	1.0
handleEvent	调用指定事件的处理	1.2
select	选择对象的输入域	1.0

Textarea

HTML 表单中的多中输入域。用户可以用一个文本区域来输入单词、短语或数字。

兼容性

JavaScript 1.1 / 客户端

事件处理方法

onBlur, onChange, onFocus, onKeyDown, onKeyPress, onKeyUp, onSelect

属性概要

属性如表 A-64 所列。

方法概要

方法如表 A-65 所列。这个对象从 Object 中继承了 watch 和 unwatch 方法。

表 A-64 Textarea 属性

属性	描述	版本
defaultValue	表示 VALUE 属性	1.0
form	指定包含 *Textarea* 对象的表单	1.0
name	表示 NAME 属性	1.0
type	表示 TYPE 属性	1.1
value	表示 *Textarea* 对象域的当前值	1.0

表 A-65 Textarea 方法

方法	描述	版本
blur	从对象上移开焦点	1.0
focus	将焦点设置给对象	1.0
handleEvent	调用指定事件的处理	1.2
select	选择对象的输入域	1.0

Window

表示一个浏览器窗口或框架。这是每一个文档、位置和历史记录对象群体的顶层对象。

兼容性

JavaScript 1.0 / 客户端

事件处理方法

onBlur, onDragDrop, onError, onFocus, onLoad, onMove, onResize, onUnload

属性概要

属性如表 A-66 所列。

方法概要

方法如表 A-67 所列。这个对象从 Object 中继承了 watch 和 unwatch 方法。

表 A-66 Window 属性

属性	描述	版本
closed	指示一个窗口是否已经关闭	1.1
defaultStatus	表示显示在窗口状态条中的默认消息	1.0
document	包含当前文档的信息，并提供向用户显示 HTML 输出的方法	1.0
frames	一个表示窗口中所有框架的数组	1.0
history	包含用户在一个窗口中已经访问过的 URL 信息	1.1
innerHeight	表示窗口内容区域的纵向尺度像素值	1.2
innerWidth	表示窗口内容区域的横向尺度像素值	1.2
length	窗口中的框架数目	1.0
location	包含当前 URL 的信息	1.0
locationbar	表示浏览器窗口的位置条	1.2
menubar	表示浏览器窗口的菜单条	1.2
name	用来指定这个窗口的唯一的名称	1.0
opener	在窗口被 open 方法打开时表示文档调用的窗口名	1.1
outerHeight	表示窗口外边界的纵向像素高度	1.2
outerWidth	表示窗口外边界的横向像素宽度	1.2
pageXOffset	提供窗口浏览页面的 x 位置的当前像素值	1.2
pageYOffset	提供窗口浏览页面的 y 位置的当前像素值	1.2
parent	包含当前框架的窗口或框架	1.0
personalbar	表示浏览器窗口的“个人条（personal bar）”（也叫管理条）	1.2
scrollbars	表示浏览器窗口的滚动条	1.2
self	指当前的窗口	1.0

表 A-66 Window 属性（续）

属性	描述	版本
status	说明窗口状态条中的优先或临时消息	1.0
statusbar	表示浏览器窗口的状态条	1.2
toolbar	表示浏览器窗口的工具条	1.2
top	顶层浏览器窗口	1.0
window	指当前窗口	1.0

表 A-67 Window 方法

方法	描述	版本
alert	显示一个带有消息和一个“OK”按钮的 Alert 对话框	1.0
back	在顶层窗口的任何框架中撤消历史记录的最新一步	1.2
blur	从指定对象上删除焦点	1.0
captureEvents	设置窗口或文档来捕获所有指定类型的事件	1.2
clearInterval	取消随 setInterval 方法一起设置的一个时间中断	1.2
clearTimeout	取消随 setTimeout 方法一起设置的一个时间中断	1.0
close	关闭指定窗口	1.0
confirm	显示一个带有特定消息和“OK”和“Cancel”按钮的确定对话框	1.0
disableExternalcap true	取消通过 enableExternalCapture 方法设定的外部事件捕获设置	1.2
enableExternalCapt ure	允许带框架窗口捕获从不同位置（服务器）载入的页面中的事件	1.2
find	在指定窗口的内容中查找指定文本串	1.2

表 A-67 Window 方法（续）

方法	描述	版本
focus	将焦点设置给指定对象	1.1
forward	载入历史记录列表中下一个 URL	1.2
handleEvent	对指定事件调用处理方法	1.2
home	将浏览器指向在参数选择中指定为用户主页的 URL	1.2
moveBy	按指定的数量移动窗口	1.2
moveTo	将窗口的左上角移动到指定的屏幕坐标处	1.2
open	打开一个新的 Web 浏览器窗口	1.0
print	打印窗口或框架的内容	1.2
prompt	显示带有消息和输入域的提示对话框	1.0
releaseEvents	设置窗口释放所捕获的指定类型事件，将事件对象沿事件层送到较远的对象	1.2
resizeBy	通过移动窗口的右下角指定的数量调整窗口的大小	1.2
resizeTo	将窗口大小调整到指定的外边缘高度和宽度	1.2
routeEvent	沿标准事件层传递一个捕获的事件	1.2
scroll	滚动窗口到一个指定的坐标	1.1
scrollBy	按指定的数量来滚动窗口的显示区域	1.2
scrollTo	滚动窗口显示区域到指定的坐标，这样，指定的点就落在左上角	1.2
setInterval	在每一个指定的毫秒数间隔内对一个表达式求值或调用一个方法	1.2
setTimeout	每次到指定的毫秒数时间之后，对一个表达式求值，或调用一个方法	1.0
stop	终止当前的下载	1.2

顶层属性和函数

顶层属性和函数和任何对象无关。表 A-68 列出了顶层属性的具体内容。表 A-69 列出顶层函数。

表 A-68 Top-Level 属性

属性	描述	版本
infinity	一个表示无穷大的数值	1.3
NaN	一个表示非数的值	1.3
undefined	未定义的值	1.3

表 A-69 Top-Level 函数

函数	描述	版本
escape	返回 ISO Latin-1 字符设置中一个参数的十六进制编码，用它来创建添加到 URL 的字符串	1.0
eval	对一个并不指示特定对象的 JavaScript 代码字符串求值	1.0
isFinite	对一个参数求值，确定它是否是一个数字	1.3
isNaN	返回一个表示指定对象的字符串。不包括 Object.toString 方法	1.0
Number	将一个对象转换为一个数字	1.2
parseFloat	解析一个字符串参数，并返回一个浮点数	1.0
parseInt	解析一个字符串参数，并返回一个整数	1.0
string	将一个对象转换为一个字符串	1.2
unescape	对指定的十六进制编码值返回一个 ASCII 字符串	1.0

事件处理方法

这一部分包含 JavaScript 的 23 个事件处理方法的语法。

onAbort

图片的事件处理方法。在一个异常中断事件发生的时候执行JavaScript代码；也就是用户中断图片的载入（比如，点击了一个链接或点击“Stop”按钮）。

兼容性

JavaScript 1.1

事件属性

onAbort有如下的属性：

Type

表示事件类型

Target

表示事件最初传送到的对象

onBlur

对Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, Textarea和Window的事件处理方法。当一个模糊事件发生的时候执行JavaScript代码；也就是当一个表单元素失去焦点，或者一个窗口或框架失去焦点的时候。

兼容性

JavaScript 1.0

事件属性

onBlur有如下所示的事件属性：

Type

表示事件类型

Target

表示事件最初传送到的对象

onChange

当一个修改事件发生时执行JavaScript代码；也就是当一个Select，Text或Textarea域失去焦点并且其值被修改的时候。它是FileUpload，Select，Text和Textarea的事件处理方法。

兼容性

JavaScript 1.0

事件属性

onChange有如下所示的事件属性：

Type

表示事件类型

Target

表示事件最初传送到的对象

onClick

Button，Document，Checkbox，Link，Radio，Reset和Submit的事件处理方法。在一个点击事件发生时执行JavaScript代码；也就是表单中的某个对象被点击的时候。（一个点击事件由*MouseDown*和*MouseUp*事件组成。）

兼容性

JavaScript 1.0

事件属性

onClick有如下所示的事件属性：

Type

表示事件类型

Target

表示事件最初传送到的对象

LayerX，*layerY*，*pageX*，*pageY*，*screenX*，*screenY*

表示onClick事件发生时的指针位置

Which

用 1 表示点击鼠标左键，3 表示右键

Modifiers

包含一个事件发生时的修改键（modifier key）列表

onDblClick

文档和链接的事件处理方法。在 *DblClick* 事件发生时执行 JavaScript 代码；也就是在用户双击一个表单元素或链接的时候。

兼容性

JavaScript 1.2

事件属性

onDbClick 有如下所示的事件属性：

Type

表示事件类型

Target

表示 onDblClick 事件最初传送到的对象

LayerX，*layerY*，*pageX*，*pageY*，*screenX*，*screenY*

表示事件发生时的指针位置

Which

用 1 表示点击鼠标左键，3 表示右键

Modifiers

包含一个事件发生时的修改键列表

onDragDrop

文档和链接的事件处理方法。在 *DragDrop* 事件发生的时候执行 JavaScript 代码；也就是在用户将某个对象放在浏览窗口上的时候，比如放置一个文件。

兼容性

JavaScript 1.2

事件属性

onDragDrop 包含如下所示的事件属性:

Type

表示事件类型

Target

表示事件最初传送到的对象

lvt

返回包含放置对象 URL 的 *Strings* 数组

Modifiers

包含一个事件发生时的缩减修改键列表

ScreenX, screenY

表示事件发生时的指针位置

onError

在一个error事件发生的时候执行JavaScript代码，也就是当载入文档或图片引起错误的时候。

兼容性

JavaScript 1.1

事件属性

onError 有如下所示的事件属性：

Type

表示事件类型

Target

表示事件最初传送到的对象

onFocus

Button, Checkbox, FileUpload, Frame, Layer, Password, Radio, Reset, Select, Submit, Text, Textarea和Window的事件处理方法。在一个focus事件

发生的时候执行 JavaScript 代码；也就是在一个窗口、框架或一个框架设置得到焦点或一个表单元素得到输入焦点的时候。

兼容性

JavaScript 1.0

事件属性

onFocus 有如下所示的事件属性：

Type

表示事件类型

Target

表示事件最初传送到的对象

onKeyDown

Document，Image，Link 和 Textarea 的事件处理方法。在一个 *KeyDown* 事件发生的时候执行 JavaScript 代码；也就是在用户按下某个键的时候。

兼容性

JavaScript 1.2

事件属性

onKeyDown 有如下所示的事件属性：

Type

表示事件类型

Target

表示事件最初传送到的对象

LayerX, layerY, pageX, pageY, screenX, screenY

对于窗口中的一个事件，它们表示在事件发生时的指针位置。对于表单中的一个事件，它们表示表单元素的位置

Which

表示所按键的 ASCII 值。为了得到所按键的真实字母、数字或符号，要使用 `String.fromCharCode` 方法。为了在 ASCII 值未知的时候设置这个属性，要使用 `String.charCodeAt` 方法

Modifiers

包含事件发生时的一个修改键列表

onKeyPress

Document，Image，Link和Textarea的事件处理方法。在一个*KeyPress*事件发生的时候执行JavaScript代码；也就是在当用户按下或按着某个键的时候。

兼容性

JavaScript 1.2

事件属性

onKeyPress有如下所示的事件属性：

Type

表示事件类型

Target

表示事件最初传送到的对象

LayerX, layerY, pageX, pageY, screenX, screenY

对于窗口中的一个事件，它们表示在事件发生时的指针位置。对于表单中的一个事件，它们表示表单元素的位置

Which

表示所按键的ASCII值。为了得到所按键的真实字母、数字或符号，要使用`String.fromCharCode`方法。为了在ASCII值未知的时候设置这个属性，要使用`String.charCodeAt`方法

Modifiers

包含事件发生时的一个修改键列表

onKeyUp

Document，Image，Link和Textarea的事件处理方法。当一个*KeyUp*事件发生的时候执行JavaScript代码；也就是在用户释放一个键的时候。

兼容性

JavaScript 1.2

事件属性

onKeyUp 有如下所示的事件属性：

Type

表示事件类型

Target

表示事件最初传送到的对象

LayerX, layerY, pageX, pageY, screenX, screenY

对于窗口中的一个事件，它们表示在事件发生时的指针位置。对于表单中的一个事件，它们表示表单元素的位置

Which

表示所按键的ASCII值。为了得到所按键的真实字母、数字或符号，要使用 String.fromCharCode 方法。为了在 ASCII 值未知的时候设置这个属性，要使用 String.charCodeAt 方法

Modifiers

包含事件发生时的一个修改键列表

onLoad

Image, Layer 和Window 的事件处理方法。在一个载入事件发生的时候执行JavaScript代码；也就是在浏览器完成窗口或<FRAMESET>标签中所有框架的载入之时。

兼容性

JavaScript 1.0

事件属性

onLoad 有如下所示的事件属性：

Type

表示事件类型

Target

表示事件最初传送到的对象

Width, height

对一个窗口中而不是层中的事件，它们表示窗口的宽度和高度

onMouseDown

Button, Document和Link的事件处理方法。在一个MouseDown事件发生的时候执行JavaScript代码；也就是在用户按下一个鼠标键的时候。

兼容性

JavaScript 1.2

事件属性

onMouseDown有如下所示的事件属性：

Type

表示事件类型

Target

表示事件最初传送到的对象

LayerX, layerY, pageX, pageY, screenX, screenY

表示onMouseDown事件发生时的指针位置

Which

用1表示点击鼠标左键，3表示右键

Modifiers

包含一个事件发生时的修改键列表

onMouseMove

由于鼠标移动事件发生频繁，就把onMouseMove事件作为每一个对象的默认事件。你必须将它设置为和一个特定对象相关。当一个*MouseMove*事件发生的时候执行JavaScript代码；也就是当用户移动指针的时候。

兼容性

JavaScript 1.2

事件属性

onMouseMove 有如下所示的事件属性：

Type

表示事件类型

Target

表示事件最初传送到的对象

LayerX, layerY, pageX, pageY, screenX, screenY

表示 MouseMove 事件发生时的指针位置

onMouseOut

Layer 和 Link 的事件处理方法。当一个 *MouseOut* 事件发生的时候执行 JavaScript 代码；也就是每次鼠标箭头从内部离开某个区域（客户端图片地图）或链接的时候。

兼容性

JavaScript 1.2

事件属性

onMouseOut 有如下所示的事件属性：

Type

表示事件类型

Target

表示事件最初传送到的对象

LayerX, layerY, pageX, pageY, screenX, screenY

表示 MouseOut 事件发生时的鼠标位置

onMouseOver

Layer 和 Link 的事件处理方法。在一个 *MouseOver* 事件发生的时候执行 JavaScript 代码；也就是每次鼠标箭头从外面开始移过一个对象或区域的时候。

兼容性

JavaScript 1.2

事件属性

onMouseOver 有如下所示的事件属性：

Type

表示事件类型

Target

表示事件最初传送到的对象

LayerX, layerY, pageX, pageY, screenX, screenY

表示 MouseOver 事件发生时的指针位置

onMouseUp

Button, Document 和 Link 的事件处理方法。在一个 MouseUp 事件发生的时候执行 JavaScript 代码；也就是用户松开一个鼠标键的时候。

兼容性

JavaScript 1.2

事件属性

onMouseUp 有如下所示的事件属性：

Type

表示事件类型

Target

表示事件最初传送到的对象

LayerX, layerY, pageX, pageY, screenX, screenY

表示 MouseUp 事件发生时的指针位置

Which

用 1 表示点击鼠标左键，3 表示右键

Modifiers

包含一个事件发生时的修改键列表

onMove

Window的事件处理方法。在一个移动事件发生的时候执行JavaScript代码；也就是用户或script移动一个窗口或框架的时候。

兼容性

JavaScript 1.2

事件属性

onMove有如下所示的事件属性：

Type

表示事件类型

Target

表示事件最初传送到的对象

ScreenX, screenY

表示窗口或框架的左上角位置

onReset

Form事件处理方法。在一个reset事件发生的时候执行JavaScript代码；也就是用户重置一个表单的时候（点击“Reset”按钮）。

兼容性

JavaScript 1.1

事件属性

onReset有如下所示的事件属性：

Type

表示事件类型

Target

表示事件最初传送到的对象

onResize

Window事件处理方法。在一个调整大小事件发生的时候执行JavaScript代码；也就是当用户或script调整一个窗口或框架的大小的时候。

兼容性

JavaScript 1.2

事件属性

onResize有如下所示的事件属性：

Type

表示事件类型

Target

表示事件最初传送到的对象

Width, height

表示窗口或框架的高和宽

onSelect

Text和Textarea的事件处理方法。在一个选择事件发生的时候执行JavaScript代码；也就是用户在一个text或textarea域中选择一些文字的时候。

兼容性

JavaScript 1.0

事件属性

onSelect有如下所示的事件属性：

Type

表示事件类型

Target

表示事件最初传送到的对象

onSubmit

Form事件处理方法。在一个提交事件发生的时候执行JavaScript代码，也就是用户提交一个表单的时候。

兼容性

JavaScript 1.0

事件属性

onSubmit有如下所示的事件属性：

Type

表示事件类型

Target

表示事件最初传送到的对象

onUnload

Window事件处理方法。在一个提交事件发生的时候执行JavaScript代码；也就是用户提交一个表单的时候。

兼容性

JavaScript 1.0

事件属性

onUnload有如下所示的事件属性：

Type

表示事件类型

Target

表示事件最初传送到的对象

附录二

Web 资源

本附录提供 JavaScript 和 Web 的相关资源。包括 JavaScript，Perl 和 CGI 的参考和图形资源部分。最后一个部分，“类似的应用程序”指向那些和本书所给应用程序类似程序的网站。看一看如何用其他代码解决相似的问题或许对你会有帮助。

JavaScript 酷站

下面列出了几个使用 JavaScript 来实现效果的网站。大部分的代码是纯粹的 JavaScript 和 DHTML。有一个站点要求 Macromedia Flash 3 插件，但任何 IE 和 Navigator 的 4.x 或 5.x 的新近版本都有这个功能。

GaboCorp:

http://www.gabocorp.com/

Doc Ozone:

http://www.ozones.com/blueprint.html

Vivatrix 的 Scrutinizer:

http://vivatrix.com/demos/en/scrutinizer

本田汽车:

http://www.honda1999.com/

Haznet的 Fallout Shelter:

http://www.hudziak.com/haznet/javascript.html

JavaScript 参考

这些URL伴我度过了几个月的快乐时光。它们同样也会把快乐带给你。你会发现有关JavaSript的参考素材、指南、脚本、文章，反正有足够多的东西。

核心 JavaScript 参考:

http://developer1.netscape.com/docs/manuals/js/core/jsref/contents.htm

Doc JavaScript:

http://www.webreference.com/js/

HotSyte —— JavaScript 资源:

http://www.serve.com/hotsyte/

Microsoft 脚本技术 (JScript):

http://msdn.microsoft.com/scripting/default.htm?/scripting/jscript/default.htm

Microsoft 网络开发 (JScript):

http://msdn.microsoft.com/developer/default.htm

JavaScript 资源:

http://javascript.internet.com/

Cut-n-Paste JavaScript:

http://www.infohiway.com/javascript/indexf.htm

JavaScript World:

http://www.jsworld.com/

webmonkey /JavaScript:

http://www.hotwired.com/webmonkey/javascript/?tw=javascript

CNET 的 Builder.com:

http://builder.cnet.com/Programming/

JavaScripts.com:

http://www.javascripts.com/

eScriptZone.com:

http://www.escriptzone.com/

JavaScript FAQ

如果你有问题，他们会给你答案。这些常见问题（FAQ），尤其是IRT.org上的，所解答的问题从基础的到非常复杂的都应有尽有。

IRT.org:

http://www.irt.org/script/faq.htm

JS Beginner:

http://www.geol.uni-erlangen.de/geojs/JS_tutorial/JS_beginners.html

DevEdge 新闻组 FAQ:

http://developer.netscape.com/support/faqs/champions/javascript.html

JavaScript Beginner FAQ:

http://www.it97.de/JavaScript/JS_tutorial/3rdless.html

JavaScript 迷你 FAQ:

http://www.dannyg.com/javascript/jsminifaq.html

DHTML 参考

这些都是很好的东西，可以让你总能获得最新的DHTML信息。这些站点中许多文章包含跨浏览器主题，或是IE或Navigator特定主题的详细讨论。

DevHead dHTML:

http://www.zdnet.com/devhead/filters/dhtml/

DHTML Zone:

http://www.dhtmlzone.com/index.html

Dynamic Drive:

http://www.dynamicdrive.com/

Inside DHTML:

http://www.insidedhtml.com/

Dynamic Duo:

http://www.dansteinman.com/dynduo/

DHTML Guru Resource:

http://www.htmlguru.com/

webmonkey/dynamic_html:

http://www.hotwired.com/webmonkey/dynamic_html/

DHTML 常见问题:

http://www.microsoft.com/workshop/author/dhtml/dhtmlqa.asp

文档对象模型参考

这些资源提供有关万维网联盟制定的文档对象模式标准的信息。

文档对象模型:

http://www.w3.org/DOM/

文档对象模型常见问题:

http://www.w3.org/DOM/faq.html

Perl/CGI 参考

如果你想要深入探究CGI和Perl的话，到这里来就对了。Perl资源和这门语言一样博大精深。在此之前应确保已经下载并安装了功能强大的模块。

Perl.com:

http://www.perl.com/pace/pub

Perl 参考页面:

http://reference.perl.com/query.cgi?section=tutorials

CGI 入门:

http://www.utoronto.ca/webdocs/CGI/cgi1.html

图形资源

由于图形和JavaScript关系密切，因此这里的URL将会给你一些文章资源和丰富的图形。

Cooltype.com:

http://cooltype.webpedia.com/

AndyArt:

http://www.andyart.com/

Site Buider Workshop 中的图库:

http://www.microsoft.com/workshop/c-frame.htm#/gallery/images/default.asp

类似的应用程序

这个部分提供一些站点链接，这些站点有和本书中所讨论的应用程序相类似的应用(或者至少领域类似)。

客户端搜索引擎

我在网站中发现的大部分JavaScript搜索引擎都是将搜索询问送到多搜索引擎中，比如Yahoo!，Infoseek和AltaVista。这里是几个有客户端JavaScript搜索应用的站点。它们看起来都很不错，但没有第一章中应用程序的功能。不管哪一个都可以用。如果你想学习源代码的话可以从中得到一些指示。

Computer Crap 搜索引擎:

http://www.geocities.com/SiliconValley/Horizon/2188/search.html

我看见过更多有创造性的主题，但作者 Nathan Wiegand 在这里所讨论的问题上使用相似的代码。

JavaScript 搜索函数：

http://www.serve.com/hotsyte/wildman/web_search/site_search.html

你可以在 hotsyte.com 发现 Tim Hobbs 所写的应用程序。

在线测试

这些内容看起来有点困难，但你还是可以找到 JavaScript 测试。这里是我所发现的一些。

D2 测试：

http://inetpubl.com/psy/psy.htm

Inet Publishing的这个集中测试使用JavaScript让你的精神和技巧与时间较量。

硬件基本应用测试：

http://www.cit.ac.nz/smac/hf100/test1s.htm

这个测试非常明确。它是一个关于磁盘格式化、驱动器分区、存储容量和其他一些东西的测试。它之所以酷，是因为这个应用程序让你一回答完就可以知道自己是对还是错。它实际上是一连串的测试练习。你可以试试从 *test2.htm* 到 *test9.htm* 的类似测试（*test6.htm* 中空无一物）。你可以在 http://www.cit.ac.nz/smac/cprogram/c_054s.htm 地址下发现 C 编程主题的相同内容。这些测试是由新西兰科技中心研究院提供的。

Norm 的多项选择测试：

http://lisa.unomaha.edu/2.0/test.html

你可能不会为黄色背景感到心烦，这个测试也是用 JavaScript 完成的。这个测试在表单域中插入问题和答案，而没有每次重写页面。它同样跟踪你的累积成绩。

Test 2000：

http://www.jchelp.com/test2000/drvframe.htm

这个多项选择测试用来帮助驾驶员准备加州驾驶测试的笔试部分。它可用于在最后浏览全部答案，而不是一个一个地看。你也可以浏览正确答案和说明。

幻灯片放映

这里有一些非常有趣的幻灯片放映应用程序。

管理项目 —— 幻灯片放映:

http://www.geocities.com/~mohan_iyer/slideshow.htm

这个DHTML应用程序有一些动画幻灯。要花一定的时间来下载图片，但你的等待会是值得的。你可以在很难找到的 *sildeshow.js* 文件中发现JavaScript。

Web blazonry:

http://blazonry.com/javascript/slideshow/

这个很酷的幻灯片放映显示了 *mkaz* 网站的发展变化，大约从它的最初成立开始。是非常纯粹的图片翻转。

Apartment Home Animated Virtual Tour:

http://www.mark-taylor.com/virtualtour/index.htm

这个动画幻灯片带领用户进行一次复杂住宅的虚拟漫游。要下载的图片相当长，但开发者清楚这个情况，想要留住你，因此你不能点击其他地方。

多搜索引擎界面

网上的JavaScript搜索引擎应有尽有。这里是我所碰到的几十种搜索引擎。

搜索引擎终极万用界面 (UUISE):

http://www.cris.com/~anathema/UUISE/index.html

这个灵活的程序利用cookie来让用户定制自己的多引擎搜索。用户可以从几十个选项中选出10个搜索引擎。程序剩余的部分显示在一个简单的子窗口中，包含一个文本域和指向10个选择引擎的图片。

WebSight:

http://rampages.onramp.net/~jnardo/websight/website1.htm

这个应用程序和我的相似，但可用于执行特殊的逻辑过滤。以一个帮助文件结束。

Virtual Meta Search 2:

http://WebcastLinks.com/vmsearch/vmscarch.html

这个应用程序让你从31个搜索引擎中选出6个，然后在不同的框架中返回每一个引擎的搜索结果。

Computer ESP Bargain Agent:

http://www.shopper.com/shop/

这个应用程序搜索在线计算机存储数据库，因此你可以发现需要的传动装置、这比向搜索引擎的数据库提交查询要好。

翻转发生器

网上有很多翻转应用程序。为什么不访问这些站点去看看它们的内容呢？

Mighty Mouseover Machine:

http://builder.cnet.com/Programming/Kahn/012898/index.html

Charity Khan的站点可能比Builder.com有更多的访问量。她的应用程序利用一个子窗口来创建一个图片模板，最终创建代码。

OnMouseOver Whipper:

http://wsabstract.com/mousewhipper/index.htm

这个Web抽象应用程序一次只允许一个图片对。但是编程人员添加了用来访问图片高和宽的自动检测功能，这使它显得非常酷。应用程序会自动帮你得到它们。创建一个*Image()*对象和从中获取属性似乎有点问题，只能祝你工作顺利啦。

JavaScript 翻转发生器:

http://webreview.com/wr/pub/98/03/13/coder/rollover.html

就像Whipper一样，一次也只能用一个图片对。但你可以控制很多的设置。你也能在保存代码之前浏览翻转效果。

库

在JavaScript库方面肯定不能搜罗无遗。这里所列出的包含一些制作DHTML的重量级代码。等你有空的时候，可以下载这些库，查看其中的代码。你可能会因此而丧失一些脑细胞，但这种努力会对你的编程起很大的作用。

DHTMLLib版本2:

http://www.insidedhtml.com/dhtmllib/page1.asp

这个采用InsideDHTML.com的库是跨浏览器应用程序创建的网上联播大本营。基于IE的文档对象模式，这个库使你写的代码能得到两种对象模式(Navigator和IE)的支持。

FreeDOM:

http://www.builder.com/Programming/FreeDOM/

这个可用在C|Net上的库让你在DOM的上下文之外更好地创建和操作JavaScript对象。

JavaScript菜单组件:

http://developer.netscape.com/viewsource/smith_menu/smith_menu.html

Netscape的库文件(*menu.js*)让你轻松地为子菜单添加浮动菜单。想了解跨浏览器吗？现在我们来看一看。

JavaScript DHTML可折叠列表:

http://developer.netscape.com/docs/technote/dynhtml/collapse/index.html#xbrowser

这里所提供的代码让你将Windows的类似Explorer重新组织其结构，链接上你的内容。你还可以发现一个到代码产生器的链接，它帮助你建设DHTML。

ScriptBuilder.com:

http://www.scriptbuilder.com/netobjects/library.nsf/By+Language

NetObjects脚本知识库有足够的剪贴代码。不是所有的代码都在in.*js*文件中，但你和你的文本编辑器可以对全部的内容作改动。

cookie

接下来的两个链接指向很多cookie应用。它们拥有从简单到高级的内容，足够你忙一阵子了。

Cookie 演示:

http://www.cookiecentral.com/demomain.htm

这个页面包含一个从coolie核心到cookie成分的链接列表。应用程序功能设置了用户参数选择，单个访问者登陆次数和浏览器探测。

使用 cookie 的购物车:

http://www.ozemail.com.au/~dcrombie/cartdemo/index.html

在这里，你可以看到如何在购物车应用中使用cookie。你可以重新做一遍，并把它加到第八章中的购物车应用程序里。

购物车

JavaScript 购物车对网络而言可不是什么新鲜事儿。参见如下所示的条目。除前两个之外都是免费的。

Shop@ssitant:

http://www.floyd.co.uk/

这个商业应用程序对 Web 购物者有许多功能和优点。它的发展靠的是一个 JavaScript 顶尖团队，因此不要因此而自惭形秽。

Shopmaster:

http://www.shopmaster.net/shopmaster/shop.htm

这个站点的应用程序显示了一个非常棒的购物车，并因为有成组显示的小图片，随 mouseover 事件来显示的产品信息和其他一些东西而显得很完善。

一个简单的购物车程序:

http://lymdenlodge.hypermart.net/ShoppingCart.htm

它的确非常简单，但如果你要的只是一些知名产品，它就已经足够了。产品选择、价钱计算和付款信息都在页面上。

JShop JavaScript 购物车:

http://javaboutique.internet.com/JShop/

这个来自JavaBoutique的应用是一个很不错的基础购物车。它可以扩展，但你可能需要添加很多代码才能使其为你所用。

Wildman 的购物车:

http://www.serve.com/hotsyte/wildman/shopping_cart/shop_cart._intro.html

作者Timothy Hobbs将这个应用程序放在HotSyte上。它用cookie来存储用户的选择。

使用 cookie 的购物车:

http://www.ozemail.com.au/~dcrombie/cartdemo/index.html

这可是一个看cookie如何被用在购物车应用中的好地方。然后你可以重新做一遍，把它加到第八章的购物车程序里。

密码

你可以发现一些零零碎碎的站点，其中有重要的密码信息。但是那些用 JavaScript 来做示范的应用非常稀少。如果你发现了其他的，请跟我说一声。

Gordon McComb 的密码:

http://gmccomb.com/commerce/frame.html

这个站点来自一个最初一批JavaScript作者之手。可以得到JavaScript密码和JavaScript密码保护示例。

RSA 算法 JavaScript 页面:

http://www.orst.edu/dept/honors/makmur/

关于RSA加密技术的多页面介绍，向你展示如何用JavaScript执行一个非常基本的RSA表单。如果你想要查看代码，应该具备一定的获得素数的知识。

JavaScript 中的密码:

http//www.serve.com/hotsyte/ciphers/

当然总要有我自己的站点呀。这个应用程序囊括了置换和变换密码。

拖放式概念

这个链接指向一些权衡浏览器事件模型和 JavaScript 的资源。

DHTMLLib Demos 的拖放程序:

http://www.insidedhtml.com/dhtmllib/demos/dragdrop.asp

这个演示是前边所说的专业站点 DHTMLLib 库的一个部分。它展示了一个跨浏览器 DHTML，支持平滑的拖放功能。

Dynamic Duo 的拖放概念:

http://www.dansteinman.com/dynduo/dragconcepts/dragconcepts.html

Dan Steinman 给了你一个关于鼠标事件的一个全面的 JavaScript/DHTML 指南。有足够的代码和示例。

Netscape 的 Visual DHTML:

http://developer.netscape.com/docs/examples/dynhtml/visual/index.htm

我对浏览器专有技术并不是很花精力，但这个唯一的 Netscape 拖放式 DHTML 编辑器却真是不错。这个应用程序用 JavaScript 1.2 在最大限度上给用户一个基于 Web 的 GUI。即使你不打算使用这个应用程序，但可以运行演示来看看整个的工作情况。

拖放:

http://www.dpunkt.de/javascript/bsp/script2/dragdrop/index.html

这个快速示例页面（我相信是用德语写的）给你另外一个拖放的视角。

Coolnerds Dynamic HTML 例子:

http://www.milliscrip.com/webauth/dhtml/dragdrop.htm

我已经访问过这个 Potato Head 先生的网站。你可以通过拖放来对头发、眼睛、嘴、鼻子和其他一些选择，设计你自己的形象。这实际上是一个 VBScript 例子，但总之你可以看到 DOM 究竟是如何工作的。

能感知当前环境的帮助

我还是需要一个如第十一章中所述的某个网上在线帮助应用程序。但你会发现下面

链接中的代码比你想像的更值一读。其中的每一个都用相似的代码来扮酷，其效果也没有多大的差别。

Microsoft 主页:

http://www.microsoft.com/ms.htm

在主页右边，Microsoft 的开发者已经在导航条中嵌入了可扩展的列表。每一个列表都提供对其他页面的链接，或者是对其他列表的链接。

JavaScript 菜单组件:

http://developer.netscape.com/viewsource/smith_menu/smith_menu.html

这篇文章让你一步步地查看创建跨浏览器DHTML菜单的全过程，和Misrosoft主页中的类似。作者 Gary Smith 采用了一个非常有价值的面向对象方法。

菜单工具箱:

http://www.insidedhtml.com/constsets/menus/menubar.asp

这个演示来自多个DHTML工具箱的其中之一。请认真理解其中这些代码，吸取其精华。

附录三

使用 Perl 脚本

本附录包含下面四个部分：

1. Perl/CGI 概述
2. 获得 Perl
3. 购物袋脚本 —— *bag.pl*
4. 电子贺卡脚本 —— *greet.pl*

第一个部分包含一个很短的Perl背景介绍，并提及它的一些优点。下一个部分告诉你如何下载Perl，给你几种基于操作系统的选择。最后两个部分说明第八章，以及第十章中的Perl脚本是如何工作的。

Perl/CGI 概述

缩写 Perl 是指实用摘取和报告语言（Practical Extraction and Report Language）。它最初是用来做文本和文件操作的，但对系统管理作业和创建动态Web内容也很有效。Perl 来源于 C，sed，awk 和 sh 等语言。

Perl有什么优点呢?

Perl的流行有很多原因。就语言的现状来说，Perl非常简单易学，同时它又有着强大的功能。它几乎能用在每一种可以想像的到的编程情况下。这里列出了无数Perl用途中的一部分：

- 动态网页内容
- CGI脚本中数不清的Web应用程序
- 访问数据库
- 建立搜索引擎和Web自动设置
- 密码保护和其他加密技术
- 系统管理，站点导航日志和预定任务
- 联网和其他脚本编程
- 聊天服务器和留言板

Perl很快发现了自己走向其他舞台的途径。你可以使用Perl:

- 扩展Java，C，Visual Basic，Delphi及其他代码
- 在XML应用程序中
- 作为PerlScript，一个ActiveX脚本引擎

Perl是可以自由获得的。你可以在CPAN(Comprehensive Perl Archive Network，综合Perl文档网络)站点http://www.perl.com/CPAN/得到。Windows用户也可以在ActiveState站点http://www.activestate.com/中获得。

Perl有一个非常庞大且忠心耿耿的拥护者队伍。编程人员正在不断增加数百个——如果不是数千的话——模块和应用程序，它们已经被写出来，你可以在自己的网站上轻松地执行。这些魅力不可阻挡的大部分素材资料都是自由的。Perl的流行意味着你也可以在全球发现大量的文档、支持和经验丰富的编程人员。

Perl可在许多平台上运行，包括Unix，VMS，MS-DOS，Window NT/98/95，OS/2等等。你写的大部分代码都可以轻便地从一种操作系统转到另一种操作系统。

Perl 有什么缺点呢?

开发者对Perl感到最苦恼的事是它的性能。比起C这样的编译语言，Perl的执行速度相对较慢。在Web环境中，用Perl（和其他语言）所写的CGI脚本（马上就会谈到）必须从硬盘驱动器上读出，并且每次调用都要作为一个新的进程来载入。诸如ASP和Java Servlet这样的技术可以在与Web服务器相同的存储空间运行，这很大程度上加快了执行。用于ISAPI的Perl和更多PerlEx等进展显著地提高了执行速度。

另外一个方面是Perl并不被认为是一种精美的（有模有样的）语言。Perl为了使用方便而牺牲了优雅性。它可以工作得很好，但可能看起来很丑。

Perl 和 CGI

如果你用浏览器从Web服务器请求一个带*.html*扩展名的文件，所得到的是一个静态的文档。这意味着该文件存储在计算机的某个目录下。你可以坐在包含这个文件的计算机面前，在文本编辑器中打开它，浏览和正送往你的浏览器相同的数据。

如果你请求的是一个*.cgi*文件（或者是*.pl*，*.plx*，或其他文件扩展名），你不会得到写在文件中的代码。Web服务器将用任何一种文件所配置使用的引擎来执行文件，并将输出返回到浏览器。你在浏览器中所看到的并不存在于Web服务器上。这些内容是你发出请求时创建的。

这整个的请求和接收输出的过程都是通过CGI发生的。CGI是HTTP服务器接口的标准。顺便说一句，服务器端语言不一定要是Perl。存在着用C和C++，Python，Fortran，AppleScript和其他语言所写的大量CGI脚本。要获得更多的细节，可以在http://www.cpan.org/doc/FAQs/cgi/perl-cgi-faq.html地址下查找Perl CGI编程常见问题。

为何使用 CGI?

虽然普通的CGI脚本可以用ColdFusion，ASP和其他一些技术来代替执行，但CGI标准仍然在网上有普遍的使用。其中仍有相当多的Web服务器运行Perl。而且它是自由的。由于这些素材几乎到处都是，每一个人都可以用他们非常容易就可以熟悉的东西来写JavaScript应用程序的服务器端部分，这是很有意义的。我想你会发现这些脚本也同样非常容易理解。

获得 Perl

你需要将Perl安装到你的服务器上。绝大部分的服务器主机都是如此。如果你需要自己安装Perl，你可以在ftp://ftp.rge.com/pub/languages/perl/ports/index.html地址下得到最新Perl的自由发行版。

只要点击和你的操作系统相对应的链接就行了。如果你运行的是WinNT/98/95，你可以在http://www.activestate.com/pw32/下得到最新的Perl二进制版本和信息。如果你需要更多的功能和效果，可以参见http://www.activestate.com/ActivePerl/下的ActivePerl，它是Perl的Win32发行版中一个重要的进展。

这两个网站都提供安装和配置Perl的文件。如果你有Windows，那么事情就很简单了。不管你安装的是正规Win32二进制版本还是ActivePerl包，安装都只需要寥寥几个步骤，并将配置你的Web服务器（比如Microsoft的互连网信息服务器，Peer网络服务，或者个人网络服务器）以便执行Perl脚本。为Unix和其他操作系统安装和配置Perl通常要麻烦一点。一定要查看所包括的文件是否齐全。

Perl一旦安装完毕，则需要配置Web服务器来执行Perl脚本。如果你的Web服务器主机正在运行Perl,就表示这个工作很可能是已经完成了的。可以和你的网站管理员联系以询问具体情况。如果全部工作都是由你自己来完成，你就可能需要配置Web服务器。诸如Microsoft的互连网信息服务器，Netscape的企业服务器，O'Reilly的 网站Pro，Apache Group的Apache，以及其他流行的Web服务器对此都只采用简单的操作就可以完成。可以在网上查看你的Web服务器软件的介绍文件。

购物袋脚本 —— bag.pl

例C-1是来自第八章的脚本。提供了大部分的脚本可以方便地理解购物袋从头到尾是如何工作的。脚本运行良好，但它有点儿粗糙。如果你希望得到一个有无数服务器端选项的功能强大的购物程序，我建议你采取一个更有效的解决方案。

警告： 还记得购物者在定货单中提供了敏感的信用卡信息。如果这个信息在提交之前没有得到恰当的保护，就很容易导致错误和非法的使用。为了避免这个情况发生，你的Web服务器主机应该支持一些保护类型，比如Secure Sockets Layer (SSL)，它是一个加密协议。大部分的服务器主机都支持它。你应该和你的网站管理员取得联系，在http://webopedia.internet.com/TERM/S/SSL.html地址下获得有关SSL的更多信息。

bag.pl 有三个任务：

1. 获得所有产品和消费者信息
2. 在 Web 服务器上一个指定名称的文件中保存这些信息
3. 打印一个确认页面返回浏览器

注意： 一些Web服务器主机要求你以一个*.cgi*扩展名来运行CGI脚本，而不是 *.pl*扩展名。并不麻烦，只要将文件重命名为*bag.cgi*就行了。很快就好。

我们开看看*bag.pl*如何完成这三个工作。在你看代码的时候，记住你不必理解语法。这本书不是写 Perl 的。只要一步步试着理解代码的作用就行了。

例 C-1：bag.pl

```
#!/usr/bin/perl

require "cgi-lib.pl";

print "Content-type: text/html\n\n";

&ReadParse(*in);

srand($$ ^ time);
$filename = $in{'lname'} . int(rand(999));
if (-e "$filename.txt") {
  print "The order for $filename has already been placed.";
  exit 0;
  }

open(FILE,">$filename.txt");
select(FILE);
printInfo();
close(FILE);
select(STDOUT);
printInfo();

exit 0;

sub printInfo() {
$clock = localtime();

print <<CUSTOMER_INFO;
<PRE><FONT FACE=Tahoma SIZE=3>
<H2>Shopping Bag Order Confirmation Receipt</H2>

```

```
33 <B>$clock</B>
34 <B>Reference Code: $filename</B>
35
36
37 --------------------
38 Customer Information
39
40 Customer First Name: $in{'fname'}
41 Customer Last Name   $in{'lname'}
42 Company Name:        $in{'cname'}
43 Street Address 1:    $in{'saddress1'}
44 Street Address 2:    $in{'saddress2'}
45 City:                $in{'city'}
46 State/Province:      $in{'stpro'}
47 Country:             $in{'country'}
48 Zip/Mail Code:       $in{'zip'}
49
50 CUSTOMER_INFO
51
52 print <<PAYMENT_INFO;
53
54 Payment Information
55
56 Credit Card Type:    $in{'ctype'}
57 Credit Card Number:  $in{'cnumb'}
58 Expiration Date:     $in{'edate'}
59
60 PAYMENT_INFO
61
62 print "Product Information\n\n";
63
64 $idx = 0;
65
66 while ($in{'prod' . $idx}) {
67   @getProdInfo = split("-", $in{'prod' . $idx});
68   print "Product PLU:\t\t$getProdInfo[0]\n";
69   print "\tQuantity:\t$getProdInfo[1]\n\n";
70   $idx++;
71   }
72
73 print <<TOTAL_INFO;
74
75 Total Information (\$US)
76
77 Subtotal:        $in{'subtotal'}
78 Tax Total:       $in{'taxtotal'}
79 Ship Total:      $in{'shiptotal'}
80 Bag Total:       $in{'bagtotal'}
81
82 --------------------
83 </FONT></PRE>
84 TOTAL_INFO
85 }
```

获得产品信息

第一个步骤确实相当麻烦。但其他编码者的辛苦劳动使它变成了小事一桩。让我们从第 1～7 行开始：

```
#!/usr/bin/perl

require "cgi-lib.pl";

print "Content-type: text/html\n\n";

&ReadParse(*in);
```

第一行是所有CGI脚本的共同代码。也叫“工作”行，它告诉服务器应该在哪里发现Perl。不同机器的路径是不相同的，因此你可能需要询问你的网站管理员。如果你在运行一个Windows的机器，就不必在意这一行。下面的一行通知Perl要包含来自叫*cgi-lib.pl*（大部分Perl安装中的标准名称）的库文件中的代码。它包含的代码将解读从HTML表单提交来的信息。我们只需要调用恰当的Perl子程序就行了，脚本将会读所有从表单提交上来的信息，并存储在我们触手可及的变量中。

下面的一行打印一个 HTTP 首部。这个特定的首部将确定给浏览器的 MIME (Multipart Internet Mail Extension) 类型，它规定了所要求的信息类型。它被设置为 *text/html*。其他 MIME 类型包括 *image/gif* 和 *text/plain*。

JavaScript使用函数和方法；Perl使用函数、方法和子程序。来自*cgi-lib.pl*的子程序ReadParse()从标准输入中读出HTML表单数据，并将它们放在称为*%in*的对应数组中。我们很快就可以得到表单数据，但现在就当我们已经的得到它好了。我们继续下面的内容。

将信息保存到 Web 服务器中的一个文件

现在我们已经有了表单数据。让我们创建一个文件来存储它。第9～15行创建了一个唯一的文件名：

```
srand($$ ^ time);
$filename = $in{'lname'} . int(rand(999));
if (-e "$filename.txt") {
  print "The order for $filename has already been placed.";
  exit 0;
  }
```

代码srand($$ ^ time)；利用了Perl的随机数产生器。接下来，Perl创建了一个叫做*$filename*的变量，它将包含唯一的文件名。文件名的创建是通过将购物者的姓和一个0-999之间的随机整数连接起来实现的：

```
$filename = $in{'lname'} . int(rand(999));
```

购物者的姓来自表单数据，对吗？由于数据被存储在相应的*%in*数组中，我们需要做的所有工作就是访问包含姓的元素。这里是它的语法：

```
$in{'lname'}.
```

回忆一下，lname是提交的HTML表单中的一个表单域。Perl（JavaScript也一样）中的相应数组是通过名称来索引的，因此，$in{'lname'}就指向购物者输入的姓，不管究竟是什么（例如，“Jones”）。

Perl的rand()函数产生一个从0到所传递的数字（也就是999）之间的一个随机浮点数。为了让事情更清楚些，Perl的int()函数返回rand()所选随机数的整数形式。因此，*$filename*可能有诸如Jones23，Jones997，Jones102等等值。

一旦*filename*的值得到确定，Perl就检查是否已经有同名的文件存在。这是一个简单的确认技术，用来避免文件中已经存在的定单被覆盖。例如，Jones是一个很普通的姓。如果Ed Jones和Jimmy Jones在同一天去购物，并且由于某些巧合而产生了相同的随机整数，那他们的文件名将是相同的。后一个购物者将覆盖前一个的定购信息：

```
if (-e "$filename.txt") {
  print "The order for $filename has already been placed.";
  exit 0;
  }
```

如果文件已经存在，Perl将把消息打印给购物者，告知此数字的定单已经存在，并退出脚本。购物者只需要重新载入脚本来产生另外一个随机文件名，再次运行。如果文件不存在，Perl产生如第17～20行所示的动作：

```
open(FILE,">$filename.txt");
select(FILE);
printInfo();
close(FILE);
```

这些代码用变量*$filename*打开（创建）一个文件，并添加一个*.txt*扩展名。代码

select(FILE)通知 Perl 打印输出到新创建的文件，直到有其他的通知。唯一要做的事就是积累内容。这由子程序 printInfo()来完成。你可以参见第 26～85 行的代码：

```
sub printInfo() {
$clock = localtime();

print <<CUSTOMER_INFO;
<PRE><FONT FACE=Tahoma SIZE=3>
<H2>Shopping Bag Order Confirmation Receipt</H2>

<B>$clock</B>
<B>Reference Code: $filename</B>


--------------------
Customer Information

Customer First Name:   $input{'fname'}
Customer Last Name:    $input{'lname'}
Company Name:          $input{'cname'}
Street Address 1:      $input{'saddress1'}
Street Address 2:      $input{'saddress2'}
City:                  $input{'city'}
State/Province:        $input{'stpro'}
Country:               $input{'country'}
Zip/Mail Code:         $input{'zip'}

CUSTOMER_INFO

print <<PAYMENT_INFO;
Payment Information

Credit Card Type:      $input{'ctype'}
Credit Card Number:    $input{'cnumb'}
Expiration Date:       $input{'edate'}

PAYMENT_INFO

print "Product Information\n\n";

$idx = 0;

while ($input{'prod' . $idx}) {
  @getProdInfo = split("-", $input{'prod' . $idx});
  print "Product PLU:\t\t$getProdInfo[0]\n";
  print "\tQuantity:\t$getProdInfo[1]\n\n";
  $idx++;
  }
```

```
print <<TOTAL_INFO;

Total Information (\$US)

Subtotal:          $input{'subtotal'}
Tax Total:         $input{'taxtotal'}
Ship Total:        $input{'shiptotal'}
Bag Total:         $input{'bagtotal'}

--------------------
</FONT></PRE>
TOTAL_INFO
}
```

顾名思义，printInfo()打印表单数据。它所做的第一件事是通过将变量*$colck*设置为函数localtime()的输出而创建一个时间-日期标志。这马上就要用到。打印工作是从第29行开始的。代码<<CUSTOMER_INFO识别叫做*here string*的东西，它基本上是定位在两个标识符之间的多行字符串。CUSTOMER_INFO是标识符。在这里，字符串的唾手可得是因为你不必为回车或者单引号双引号操心。你用起来就像是在输入一个字符。这样，Perl就指向CUSTOMER_INFO和CUSTOMER_INFO之间的任何东西，它也就是：

```
<PRE><FONT FACE=Tahoma SIZE=3>
<H2>Shopping Bag Order Confirmation Receipt</H2>

<B>$clock</B>
<B>Reference Code: $filename</B>


--------------------
Customer Information
Customer First Name:   $input{'fname'}
Customer Last Name:    $input{'lname'}
Company Name:          $input{'cname'}
Street Address 1:      $input{'saddress1'}
Street Address 2:      $input{'saddress2'}
City:                  $input{'city'}
State/Province:        $input{'stpro'}
Country:               $input{'country'}
Zip/Mail Code:         $input{'zip'}
```

注意到你不必跨越数行来连接一个字符串。你只不过是输入，再接着输入。同时注意到变量已经被编译了。换句话说，$input{'lname'}不会打印$input{'lname'}；它打印的是如“Jones”的东西。

如果你认真地参看代码，就会看到第一个打印的信息是购物者的通信信息。紧接的

是支付信息。所有这些信息来自已知的表单域，比如名为*fname*，*lname* 和 *city* 的域。这些是购物者要填写的域，因此我们知道它们的位置。

可产品信息又如何呢？任何一个产品数量都可能会是。Perl 如何知晓究竟有多少产品呢？它其实根本不知道。第 62～71 行会说明这并不要紧：

```
print "Product Information\n\n";

$idx = 0;

while ($in{'prod' . $idx}) {
  @getProdInfo = split("-", $in{'prod' . $idx});
  print "Product PLU:\t\t$getProdInfo[0]\n";
  print "\tQuantity:\t$getProdInfo[1]\n\n";
  $idx++;
  }
```

记得购物袋曾为购物者定购的每一个产品都产生一个隐藏表单域。这些隐藏表单域中的每一个都按照“prod”＋整数命名惯例来命名（例如，*prod0*，*prod1*，*prod2* 等等）。Perl 只定义一个变量，将它设置为 0。借助 66 行一个熟悉的 *while* 循环，Perl 用 $in{'prod'.$idx} 来查看 *prod0* 是否存在。如果存在，变量一定包含产品信息。

然后，Perl 用它的 split() 函数来创建一个称为 *@getProdInfo* 的数组，它包括两个元素。第一个元素包含产品的 PLU 号码；第二个元素则包含定购数量。打印这个信息后，*$idx* 得到增量。Perl 依次处理 *prod1*，*prod2* 等等。直到 $in{'prod' . $idx} 不存在。

这个过程一旦结束，Perl 在第 73～84 行打印小计、使用税金和总数。注意美元符号要加反斜杠来代替 (\$)。由于我们确实需要打印字符串 $US，我们就必须告知 Perl，没有名为 $US 的变量：

```
print <<TOTAL_INFO;

Total Information (\$US)

Subtotal:          $input{'subtotal'}
Tax Total:         $input{'taxtotal'}
Ship Total:        $input{'shiptotal'}
Bag Total:         $input{'bagtotal'}

---------------------
</FONT></PRE>
TOTAL_INFO
```

对购物者返回一个确认页面

这让我们来到了子程序的尾声，因此Perl继续执行printInfo()的调用下面的第20～24行的代码：

```
close(FILE);
select(STDOUT);
printInfo();

exit 0;
```

由于文件有了全部的信息，我们可以圆满地关闭它了。close(FILE)执行这个任务。购物者怎么样了呢？让他知道定单已经被接受了当然是最好不过的。为什么不像我们在文件中所做的那样给购物者打印同样的返回信息呢？对select(STDOUT)的调用告知Perl再次打印标准输出，这表示返回到浏览器（记得我们在第17行将输出目标转变为*$filename.txt*）。现在，我们只需要再次调用printInfo()，这个处理就大功告成了。购物者得到了一个打印输出。

设立脚本

随着脚本的设立，它将在脚本所定位的相同目录下为产品定单创建全部的文本文件。换句话说，不管你将*bag.pl*放在何处，你的产品定单文件也就在那里。没有你需要建立或遵循的目录结构。任何有执行权限（这样Web服务器就可以调用脚本）和写权限的目录都可以。

如果你是刚接触权限概念，记住它们控制用户所属的对文件和目录的访问类型。例如为了重新得到基本HTML文件，包含HTML文件的目录需要读出分配给它的权限。为了执行CGI和其他脚本，目录必须要有分配给它的执行权限。为了创建和修改目录中的文件，这个目录就需要写权限。包含*bag.pl*的目录需要写权限和执行权限。这听起来倒是挺简单的，但其实还需要费些手脚。

给予一个目录写和执行权限至少要冒两个风险。如果你有一个Web服务器主机，那里的人们知道这个情况，就有可能要求你把脚本放在只有执行权限的目录下，比如*cgi-bin/*或*Script/*，同时把你的文件写到另外一个只有写权限的目录下。如果是这种情况，你需要在脚本中作一个简单的修改来表明产品定单文件的新目录。只要在设置变量*$filename*的值时包括目录名就行了。假设你打算在一个名为*orders/*的目录

下写你的文件，这个目录定位于你的脚本所在目录的上一层目录中。只要修改第10行的如下内容：

```
$filename = $in{'lname'} . int(rand(999));
```

修改为：

```
$filename = "../orders/" . $in{'lname'} . int(rand(999));
```

如果*orders/*脱离了根目录，这一行就会像这样：

```
$filename = "/orders/" . $in{'lname'} . int(rand(999));
```

现在你的脚本在一个目录下，而你的产品定单文件在另外一个目录下。

你准备到达那个目录。要确保*\ch08\manager.html*第276行的ACTION属性包含正确的*bag.pl*的URL。

电子贺卡脚本 —— greet.pl

例C-2是从第十章来的脚本。这个脚本将会解读用户提交的信息，然后创建一个唯一的文件，在其中写贺卡代码。然后，脚本会返回给发送者一个确认页面，其中包含一个HTML表单。通过提交表单，发送者就将Email发送给收件人了。这个Email消息包含一个对刚才所创建的*greet.pl*文件的链接。收件人在他的Email消息中点击链接时，就准备使用这个文件。

设立脚本

和第八章中的脚本不同，你需要遵循某个目录结构，以便使用*greet.pl*。在你放置*greet.pl*的任何目录中，目录必须同时有分配给它的执行权限和写权限。你也必须在其中有一个叫*greetings/*的目录，分配有写权限。

正如我在本章前边所讲过的一样，如果你对权限的概念不是很熟悉的话，就记住它们控制用户对文件和目录所能有的访问类型。例如为了重新获得基本HTML文件，包含HTML文件的目录就需要分配有读权限。为了执行CGI和其他脚本，目录必须分配有执行权限。为了在某个目录下创建和修改文件，这个目录需要有写权限。

实际上，现在可能是考虑你的目录结构形式的好时机。让我们来假设你正打算将这个应用程序中所有的文件放到Web服务器的*cgi-bin/*目录下。那么你的目录结构看起来应该会像这样：

```
cgi-bin/
  greet.pl
  index.html
  back.html
  front.html
  greetings/
  images/
```

*cgi-bin/*包含客户端应用程序的*greet.pl*和三个HTML文件。*images/*目录包含用户选来定制贺卡的所有图标和背景图片。但要记住，这就是从客户端产生的代码所要指向的地方，目的是访问这些图形并为每一个贺卡下载它们。

*greetings/*目录最初是空的。在这里创建每一个命名唯一的贺卡文件，并存储起来以备收件人下载。由于贺卡是在这里创建的，这个目录就必须分配有写权限。随着这个结构的完成和所有文件的就位，你就作好了运行的准备。要确保*\ch10\front.html*的第186行中ACTION属性包含*greet.pl*正确的URL。

你可能不喜欢上述的目录结构。你可能想要这些HTML文件在*cgi-bin*之外的某个地方。没有问题。只要确保你的表单指向正确的脚本路径就行了。

注意：一些Web服务器主机要求你运行的CGI脚本有*.cgi*扩展名，而不是*.pl*扩展名。这并不麻烦，只要将文件重命名为*greet.cgi*，很快就解决了。

让我们按照如下所示的三个任务条目来看脚本：

1. 从提交表单中获得定制贺卡
2. 将信息保存在Web服务器上命名唯一的某个文本文件中
3. 打印一个确认页面发送给浏览器，以Email表单结束

在你看代码的时候，要记住你不必理解语法。这本书不是为Perl而写的。只要一步一步地试图理解代码的作用就行了。

例 C-2：greet.pl

```
 1 #!/usr/bin/perl
 2
 3 require 'cgi-lib.pl';
 4
 5 &ReadParse(*in);
 6 $msg     = $in{'EntireMessage'};
 7 $fileID  = $in{'UniqueID'};
 8 $recip   = $in{'Recipient'};
 9 $baseURL = $in{'BaseURL'};
10
11 open(FILE, ">greetings/greet$fileID.html") || die "No can do: $!";
12 select(FILE);
13 print <<GREETING;
14 <HTML>
15 <HEAD>
16   <TITLE>Your Personal Cyber Greeting</TITLE>
17 </HEAD>
18 <BODY>
19   $msg
20 </BODY>
21 </HTML>
22 GREETING
23 close(FILE);
24 select(STDOUT);
25
26 print "Content-type: text/html\n\n";
27
28 print <<RESPONSE;
29
30 <HTML>
31 <HEAD>
32   <TITLE>Cyber Greeting Response</TITLE>
33 </HEAD>
34 <BODY>
35 <TABLE WIDTH="500">
36   <TR>
37     <TD>
38     <H2>Congratulations!</H2>
39     You have successsfully created a Cyber Greeting for
40            <B>$recip</B>. All you have to do is send him or her an
41            e-mail to announce the greeting. Just push the button below,
42            and the e-mail will be on the way.
43     <CENTER>
44     <FORM NAME="SendEmail" ENCTYPE-"text/plain"
45              ACTION="mailto:$recip?Subject=You Have A Cyber Greeting!">
46     <INPUT TYPE=HIDDEN NAME="Message"
47              VALUE="You have a Cyber Greeting. You can pick it up at
48              $baseURLgreet$fileID.html">
49     <INPUT TYPE=SUBMIT VALUE="Send CyberGreeting">
50     </FORM>
51     </CENTER>
52     <BR>
```

```
53      You might experience a delay while your e-mail software
54      contacts your mail server.
55      <BR><BR>
56      <A HREF="index.html">Return To Cyber Greeting</A>
57      </TD>
58    </TR>
59 </TABLE>
60 </BODY>
61 </HTML>
62
63 RESPONSE
64
65 exit 0;
```

获得贺卡信息

我们所需做的第一件事就是获得用户提交的所有数据，把它放到手边的表单中。让我们从第1～5行开始，它和先前的PERL脚本基本一致：

```
#!/usr/bin/perl

require 'cgi-lib.pl';

&ReadParse(*in);
```

如“购物袋”部分所注，第一行是“工作”行，告知服务器应该在哪里发现Perl。第3行通知Perl包含来自名为*cgi-lib.pl*的库文件中的代码。它包含Perl子程序ReadParse()，用来从HTML表单中读提交的信息。

ReadParse()从标准输入中读出HTML表单数据，并将其放在称为*%in*的相应数组中，然后创建一个数组元素，它的名称是提交来的表单元素。每个数组元素被赋予一个值，这个值和提交的表单元素的值相对应。依照*\ch10\front.html*中HTML表单里的所有元素，*%in*有如下所示的元素：

`$in{'EntireMessage'}`-

这是用户输入的格式化消息。

`$in{'UniqueID'}`-

它包含用来创建唯一文件的一个随机数。

`$in{'BaseURL'}`-

它包含基础目录路径。

`$in{'Recipient'}`-

它是收件人的 Email 地址。

`$in{'Message'}`-

这是用户输入的未被格式化的初始消息。我们不需要它，因为我们已经有了`$in{'EntireMessage'}`中的格式化版本。

`$in{'Greetings'}`-

它包含发送者从*\ch10\front.html*的选择列表中所选的贺卡名称。

为了更简单地引用这些元素，脚本将它们的值赋给更短的变量名。但是我们只需要其中的四个。它们在第 6～9 行代码中：

```
$msg     = $in{'EntireMessage'};
$fileID  = $in{'UniqueID'};
$recip   = $in{'Recipient'};
$baseURL = $in{'BaseURL'};
```

将贺卡保存在命名唯一的文件中

我们已经有了将贺卡打印到文件的一切必须条件。我们只需要创建一个唯一的文本文件，将对应的 HTML 输出到其中。参见第 11～22 行的代码：

```
open(FILE, ">greet$fileID.html") || die "No can do: $!";
select(FILE);
print <<GREETING;
<HTML>
<HEAD>
  <TITLE>Your Personal Cyber Greeting</TITLE>
</HEAD>
<BODY>
  $msg
</BODY>
</HTML>
GREETING
```

这几行代码创建了一个新文件，将 JavaScript 在浏览器中创建的随机数用作文件名的一部分。如果这个数是 25000，那么文件就被命名为*greet25000.html*，并且被定位于*greetings/*目录下。创建好唯一的文件之后，Perl 给它写一些 HTML，包括来自*front.html*的*EntireMessage*表单域，并且现在存储在*$msg*变量中的 DHTML。

代码<<GREETING识别称为*here string*的东西，本质上是定位在两个标识符之间的多行字符串。GREETING是标识符。在这里，字符串唾手可得的原因在于你不必为回车符或单引号、双引号操心。你用起它们来简单得就好像在输入一个字母。这样，Perl打印GREETING和GREETING之间的任何东西。你只需要输入，再输入。也要注意，变量已经得到了编译。换句话说，*$msg*不会打印出“$msg”；它将打印格式化贺卡。

这就是所有的相关内容。脚本在第23行关闭了文件，它准备着供收件人浏览。

输出确认页面

剩下的唯一任务是发送一个确认页面给发送者。这个页面将会包含一个HTML表单，在表单的提交按钮上，要有贺卡的发送公告和接收URL。第26～63行代码负责这个工作。

在第26行，print"Content-type: text/html\n\n"代码打印一个HTTP标题。这个特定的标题确定给浏览器的MIME类型，它用来规定所需的信息类型。

在可以在第28和63行看到，RESPONSE确定另外一个*here string*。注意粗体显示的变量：

```
print "Content-type: text/html\n\n";

print <<RESPONSE;

<HTML>
<HEAD>
  <TITLE>Cyber Greeting Response</TITLE>
</HEAD>
<BODY>
<TABLE WIDTH="500">
  <TR>
    <TD>
    <H2>Congratulations!</H2>
    You have successsfully created a Cyber Greeting for <B>$recip</B>
    All you have to do is send him or her an e-mail to announce the
    greeting. Just push the button below, and the e-mail will be on the
    way.
    <CENTER>
    <FORM NAME="SendEmail" ENCTYPE="text/plain"
      ACTION="mailto:$recip?Subject=You Have A Cyber Greeting!">
    <INPUT TYPE=HIDDEN NAME="Message"
```

```
      VALUE="You have a Cyber Greeting. You can pick it up at
      $baseURL/greet$fileID.html">
    <INPUT TYPE=SUBMIT VALUE="Send CyberGreeting">
    </FORM>
    </CENTER>
    <BR>
    You might experience a delay while your e-mail software
    contacts your mail server.
    <BR><BR>
    <A HREF="index.html">Return To Cyber Greeting</A>
    </TD>
  </TR>
</TABLE>
</BODY>
</HTML>

RESPONSE
```

大部分的代码都是HTML，它给发送者一个贺卡建立成功的确认，还提供发送Email的用法说明。认真看一看第44～50行的代码。这是负责发送Email给收件人的表单：

```
<FORM NAME="SendEmail" ENCTYPE="text/plain"
  ACTION="mailto:$recip?Subject=You Have A Cyber Greeting!">
<INPUT TYPE=HIDDEN NAME="Message"
  VALUE="You have a Cyber Greeting. You can pick it up at
  $baseURL/greet$fileID.html">
<INPUT TYPE=SUBMIT VALUE="Send CyberGreeting">
</FORM>
```

脚本在ACTION属性中使用一个mailto:协议来输出一个表单，并将ENCTYPE设置为text/plain。这个表单有一个名叫Message的隐藏域，它包含贺卡通知和收件人用来访问贺卡的URL。URL是用*$baseURL*变量和*$fileID*变量来创建的。

这个表单还有一个提交按钮，以此来发送Email。只要发送者的浏览器Email客户设立正确（虽然他要得到承认提示），邮件消息就可以上路了。

原书空白页

词汇表

class
　　类
collection
　　收集
concatenate
　　合并
constructor
　　构造函数
container
　　容器
context
　　环境
detect
　　检测
DHTML（Dynamic Hypertext Markup Language）
　　动态超文本标记语言
DOM（Document Object Model）
　　文档对象模型
entity
　　实体
event handler
　　事件处理方法
field
　　（文本）域
form
　　表单
frameset
　　框架集合
framework
　　框架
frame
　　框架
garbage
　　无用存储单元
hardcode
　　直接编码
identifier
　　标识符
instance
　　实例

interpreter
　　解释器
iterate
　　迭代
method
　　方法
package
　　包
plugin
　　插件
positioning
　　定位
property
　　属性
public
　　公有的
scope
　　作用域
script
　　脚本
subclass
　　子类
tag
　　标签
widget
　　（图形界面）部件
window
　　窗口
wrapper
　　包装