TURING 图灵程序设计丛书

Apress

Foundation HTML5 Canvas

For Games and Entertainment

HTML5 Canvas 基础教程

[英] Rob Hawkes 著

周广新 曾少宁 盛海艳 等译

人民邮电出版社

POSTS & TELECOM PRESS

“大爱！如果你想真正理解HTML5 Canvas，学习使用Canvas开发游戏，这本书就是为你写的。虽然书中讲解了不少基础概念，但深入阅读就会发现，即使是有经验的开发人员也能够从中得到启发。不要犹豫，坚决拿下！”

——亚马逊读者

Foundation HTML5 Canvas For Games and Entertainment

HTML5 Canvas基础教程

Canvas是在桌面、平板和手机上开发跨平台动画和游戏的标准解决方案，还具备对图像和视频进行像素级操作的能力，为将来的Web图形和视频处理创造了广阔的空间。学习HTML5，必学Canvas。本书可以为Canvas开发打下坚实的基础。

这是一本真正面向初学者的书。从HTML5的历史和JavaScript（以及jQuery）的基本概念讲起，让HTML5和JavaScript新手轻松入门。然后，作者条理清晰、循序渐进地介绍了Canvas的各个方面，包括渲染上下文、坐标系统、画布状态、变形合成，以及高级的图像和视频处理。在此基础上，本书总结了与创建动画相关的数学和物理知识。最后，通过详细分析“太空保龄球”和“躲避小行星”两款经典游戏的构思、设计和编码，把读者引入了HTML5游戏开发的殿堂。

亚马逊读者对本书的评价颇高，称其为学习Canvas的必读之作。

本书的主要内容：

- HTML5的新特性及其应用
- JavaScript的基本编程知识和捕获用户输入
- Canvas的概念与特性
- 如何使用Canvas的基本和高级特性
- 如何使用Canvas和一点点物理知识创建出逼真的动画
- 如何使用Canvas、JavaScript及其他HTML5特性创建交互游戏

Apress®

图灵社区：www.ituring.com.cn
反馈／投稿／推荐信箱：contact@turingbook.com
热线：(010)51095186转604

分类建议 计算机／网络开发／程序设计

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-27101-3

9 787115 271013 >

ISBN 978-7-115-27101-3

定价：49.00元

TURING 图灵程序设计丛书

Foundation HTML5 Canvas

For Games and Entertainment

HTML5 Canvas 基础教程

[英] Rob Hawkes 著

周广新 曾少宁 盛海艳 等译

人民邮电出版社

北京

图书在版编目（CIP）数据

HTML5 Canvas基础教程 /（英）霍克斯（Hawkes, R.）著；周广新，曾少宁，盛海艳 等译. -- 北京：人民邮电出版社，2012.1
（图灵程序设计丛书）
书名原文：Foundation HTML5 Canvas : For Games and Entertainment
ISBN 978-7-115-27101-3

Ⅰ. ①H… Ⅱ. ①霍… ②周… ③曾… ④盛… Ⅲ. ①超文本标记语言，HTML 5－程序设计－教材② JAVA语言－程序设计－教材 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2011)第258498号

内 容 提 要

本书从 HTML5 和 JavaScript（以及 jQuery）的基础知识讲起，全面介绍了 HTML5 Canvas 的各种特性，包括渲染上下文、坐标系统、绘制图形、保存和恢复画布状态，以及变形、合成、处理图像和视频等，让读者对 Canvas 建立起完整的认识。随后讨论了动画循环、记忆形状、模拟运动、碰撞检测等基本而又重要的概念，带领读者温习了必要的数学和物理知识。通过带领读者动手开发“太空保龄球”和“躲避小行星”这两款小游戏，让读者掌握开发游戏的基本流程，学会响应用户操作、创造虚拟环境、循环利用对象、设计计分系统等游戏开发必备的知识。

本书适合各层次 Web 设计及开发人员阅读。

图灵程序设计丛书

HTML5 Canvas基础教程

◆ 著 [英] Rob Hawkes
译 周广新 曾少宁 盛海艳 等
责任编辑 李松峰
执行编辑 杨 爽

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京鑫正大印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：16
字数：378千字 2012年1月第1版
印数：1－4 000册 2012年1月北京第1次印刷

著作权合同登记号 图字：01-2011-3261号

ISBN 978-7-115-27101-3

定价：49.00元

读者服务热线：(010)51095186转604 印装质量热线：(010)67129223
反盗版热线：(010)67171154

译 者 序

HTML5指的是万维网（WWW）核心语言HTML的第5个版本。目前，万维网中主要采用的HTML正式版本为1999年发布的4.01版，这个版本在万维网发展过程中发挥了巨大作用。然而，随着万维网从面向文档（Document-Based）的方法向应用（Application）和移动（Mobile）等方向发展，HTML 4.01已经不能满足现代万维网在应用开发和移动性等方面的新需求了，HTML需要在语法、语言和API等方面进行革新，因此，HTML5应运而生。目前，HTML5仍处于草案阶段，但是已经引起了浏览器厂商及Web开发人员的广泛关注，基于这一点，我们看到了它巨大的发展潜力和创新意义。

在HTML5的诸多新特性中，HTML5 Canvas是其中最吸引人的特性之一。它由JavaScript脚本进行控制，可以动态地绘制出各种2D图形，甚至可以对图像像素进行任意处理。目前，HTML5Canvas已经得到了广泛的浏览器支持，包括Mozilla Firefox、Google Chrome、IE、Safari和Opera等在内。而且，有许多的网页游戏厂商和开发者也已经开始使用它来开发网页游戏。我们可以大胆地想象，在不久的将来，采用HTML5开发的无插件网页游戏将广泛流行。

本书是面向游戏和娱乐应用开发的HTML5Canvas图书，它将帮助你掌握先进的下一代网页游戏开发技术。本书共有10章内容，根据讨论的主题可以分成如下5部分。

- 第1～2章：背景知识。分别讲述HTML5背景知识和JavaScript基础知识，帮助读者了解HTML5的一些基本概念，为读者掌握本书后续内容提供所需要的JavaScript基础知识。
- 第3～5章：画布。由浅及深地介绍画布的各种概念和使用方法，除了详细的API讲解，书中还穿插了多个实例，帮助读者理解画布的具体开发流程。
- 第6～7章：动画。分为基础和高级两部分，循序渐进地讲解在逼真的游戏开发中必不可少的动画制作方法。
- 第8～9章：实例。通过前面学习的Canvas技术，向读者介绍实际开发一个游戏的方法和过程。
- 第10章：展望。关于一些扩展学习的建议。

正如作者所介绍的，本书既适合初学者学习，也适合高级人员作为参考手册使用，实例和代码齐全。因此，译者在此向广大喜爱和对HTML5技术感兴趣的读者诚意推荐本书。

最后，衷心感谢人民邮电出版社图灵公司各位编辑在翻译工作中给予的帮助和宝贵意见。由于译者水平有限，错误在所难免，恳请读者批评指正。

2011年9月

献给Lizzy，她的关爱与支持使我不致因写书而疯狂。献给我的家人，他们从小培养了我对技术和Web的兴趣，尽管他们并不十分理解这些（恐怕现在也一样）。

——Rob Hawkes

致　谢

首先，我要感谢我的女友莉莎，她在过去半年无比耐心并默默支持着我。我从未想到撰写本书会遇到如此大的困难，特别是当我将自己封闭起来艰难地处理书中最细小的细节时。我非常希望能和你在一起，就像你对我一样。

没有我的父母，就没有我（的确如此）。我要感谢他们在我孩提时就支持我沉迷于技术和计算机，而且还在我的成长过程中继续给予我支持。我绝不会忘记在父亲的网吧度过的那段日子，特别是头戴着逼真的眼动系统玩Doom时的情景。它那不可思议的虚拟现实从来没有像它承诺的那样席卷世界。那时，你们可能还无法完全理解所有这些技术的工作原理，也不理解我为什么对它感兴趣，但是我很高兴您（父亲）仍然允许我玩这个游戏，允许我学习它。非常感谢您！

我的妹妹劳拉一直是我的灵感源泉。我们在一起生活了很长的时间，看到她生活美满、工作顺利，我感到很高兴。我们可能会争吵，但是你永远是我最好的妹妹。

在技术方面，我要感谢Redweb的戴维·伯顿（David Burton）和其他同事，因为你们，我可以在过去两个夏天在创新部门进行这些疯狂的尝试。是大卫给予了我时间和勇气，让我真正去接触HTML5 Canvas和学习如何使用它。在此，我要感谢你，没有这些日子的可靠实验，我很可能无法完成本书的撰写！

约翰和汉娜是我的两位挚友，一直支持我并且一直勇于向我的观点和工作提出疑问。很少有人能够做到这一点，对此我非常感激，谢谢你们多次阻止我犯下愚蠢的错误。我还要再次感谢你们两位，以及两位在ExplicitWeb播客中精辟诙谐的争论；我从未像在我们录制节目时那样开心地笑过。

最后，我想感谢Apress和Friends of ED出版社的整个团队。特别感谢编辑本（Ben）和科尔宾（Corbin）无比耐心地帮助我完成了整本书的写作。谢谢你们曾帮助我化解压力。同时，如果没有本以及他对我的信任，本书是不可能完成的，我欠你一个人情。另外，我不会忘记道格，本书的技术审稿人，对我的代码和技术内容的仔细检查，保证了它们的准确性。你向我展示的准确性和经验给了我很多启发。感谢你。

我知道还有许多人在我撰写本书的过程中给予了帮助，例如推特（Twitter）上的网友对本书表现出了极大的热情，还帮助我测试了所有的游戏。我无法在此一一列出所有的名字。但是，我要感谢曾经支持和鼓励过我的每一个人。你们是最棒的！

前　言

如果一年前你问我是否会想象自己将来写一本书，我可能会说不会。但是，现在我成为少数撰写HTML5Canvas图书的作者了。我的目标是写一本两年前我刚开始学习Canvas的时候希望看到的书。在我开始学习Canvas时，几乎没有什么关于它的资料。幸好，现在情况比以前好多了。

在过去两年里，我投入了大量时间对Canvas和其他一些刚刚崭露头脚的Web新兴技术进行实验。这些实验包括在2010年9月重新创建了交互式Google圆球标志，使用HTML5 Canvas和Web Socket制作了一个成熟的多人游戏。正是这些实验使我具备了撰写本书和向其他人传授所学技术的知识和经验。

我痴迷于使用Canvas和JavaScript等技术进行动画和游戏开发。我真诚地希望本书能够表达我的这种热情，而且我希望它能够帮助你掌握HTML5 Canvas的使用方法。

读者对象

本书的行文通俗易懂，既贴近初学者，也同样适合专业人员。它主要面向初学HTML5和JavaScript的Web设计人员，内容涉及使用HTML5 Canvas元素进行交互式游戏和应用程序开发的最基础知识。经验丰富的Web设计人员和程序员也能够学习到Canvas的所有特性，以及在项目中使用这些特性的方法。那些希望将知识面拓宽到Web和移动设备上的Flash和Silverlight开发人员也能从中受益。

组织结构

本书从简介开始，引导你理解HTML5和强大的Web新功能。第1章的目标是介绍必要的HTML5背景知识和相关技术。

在了解HTML5基础知识之后，我们将开始学习JavaScript知识。第2章的目标是介绍使用HTML5 Canvas和使用它开发奇妙动画和游戏所需要的各种技术。

第3章是对画布元素的初步认识，你将学习如何使用它来绘制一些基本图形和文本。在这一章中，我希望你开始喜欢上画布及其简单性。

第4章将介绍画布的高级功能。你将学习到如何执行变形及如何绘制复杂图形。你还将学习到如何把在画布上绘制的图形保存为图像。

第5章继续前一章介绍的知识，并在此基础上介绍在画布中处理图像和视频的内容。这是真正开始有趣的部分，我希望你从中真正了解画布的实际运用。

第6章和第7章将更上一层楼，向你介绍如何使用JavaScript和画布实现动画。在这两章中，你将学习动画的基础知识，以及如何运用物理知识使动画更加逼真。

第8章和第9章是本书的高潮部分。这两章将分别介绍HTML5游戏的创建过程，从使用画布开发游戏的核心方面，到用户输入，再到使用HTML5音频添加声音。你在这两章学习的全部知识都是实用的HTML5游戏开发技术。

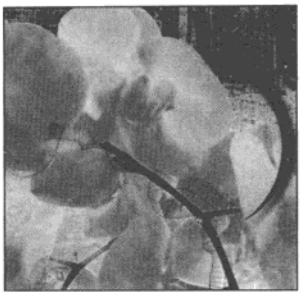
本书最后将展望画布元素的未来，以及向你介绍如何进行更深入的学习。我希望最后这一章能启发你利用所学的全部知识，并且真正将你的技术水平提高一个层次。

我尽力使本书结构更有条理，使你既能够循序渐进地学习，也可以直接阅读某个具体主题。本书的目标是成为一本利用JavaScript操作和开发动画的教程和参考手册。

本书的所有代码都可以从出版社网站（http://www.apress.com）的本书页面中下载[①]。

① 或者可以到图灵社区本书页面（turingbook. com/book/776）下载。——编者注

目　录



第 1 章

HTML5简介

我们将从零开始学习HTML5。在本章中，我们将讲一下HTML的历史，了解它的起源和过去。接着，将探讨HTML5带来的新特性，详细介绍何时以及如何使用它们。然后，将剖析一个使用HTML5开发的网页示例，以便深入理解它在实际环境中的应用。最后，我们将澄清一些将HTML5混同其他新的Web技术的错误认识。最终，通过本章的学习，你将更好地理解HTML5，同时掌握深入学习本书的基础知识。

1.1 HTML 简史

1997年12月，万维网联盟（World Wide Web Consortium，W3C）正式宣布将HTML 4.0作为W3C的推荐标准[①]。HTML 4.0与其前面的版本差别很大，而且带来了一些强大的新功能，如CSS和客户端脚本。这些特性极大地改变了人们开发网站的方法，使人们摆脱了对表格式呈现技术的依赖，转而采用诸如JavaScript这样的脚本语言来开发动态网站。在这之前，HTML主要是一种静态且受限的环境，基本上只是由文字和图片构成的，而缺少现代网页所具有的大多数普通特性。

HTML 4.01于1999年12月发布，但它并没有引入任何重要的新功能[②]。它的主要目的是修正规范中的一些错误，并做了一些小的修改。毕竟，当时HTML 4发布只有两年时间，所以还不需要对它进行大的改动。在接下来的5年时间里，它的应用情况良好。在这期间，CSS进行了一些更新，XHTML 1.0发布了，而声名狼藉的浏览器IE 6也发布了。在2005年，由于WHATWG（Web Hypertext Application Techonlogy Working Group，Web超文本应用技术工作组）制定的Web Applications 1.0的草案规范发布，情况便开始发生巨大的变化。

HTML 4.0和XHTML 1.0（以及正在起草中的XHTML 2.0）诞生时，还没有当今的博客、网店和论坛等概念。这些版本的问题在于它们作为标记语言只适用于处理静态文档，而博客和网店并非静态文档，它们实质上是应用程序。Web Application 1.0正是为解决这个问题而开发的——通过

① http://www.w3.org/html/wg/wiki/History。

② http://www.w3.org/TR/html4/。

增加新的元素（用于标记内容的标签）和功能来扩展HTML，以适应这些类型的网站需求。到2008年，Web Applications 1.0广为流行，被W3C采纳，并被转变为HTML5规范的第一个草案。它引入了大量强大的新特性，本书将详细讨论。

在简单了解了HTML的历史之后，我们回到现在。HTML5目前正处于工作草案形式，就在我们谈论它的时候，它正在发生变化。它在技术上仍然未完成，并且还需要一定的时间才能完成；工作草案是指草案之后的一个阶段，但它只是全部6个阶段中的第3个[①]。然而，好消息是HTML5的很多部分已经在各种浏览器中实现，因此我们现在已经可以使用它的一些令人兴奋的新特性了。任何出色的Web设计人员或程序员确实都没有理由忽视HTML5。简而言之，它是Web的未来。当每个人都在尽享它的乐趣时，你真的甘心落后吗？

1.2　为什么需要HTML5

正如我们之前所讨论的，HTML5是为了解决广泛存在的问题而创建的。但是这些问题到底是什么？而HTML5又是如何解决它们的呢？现在让我们一起来研究一下。

1.2.1　问题

在HTML 4.01及更早的版本中，我们知道有一种所谓面向文档的方法。Web最初是用来显示和共享科学文档的。虽然文档中包含的不仅仅是科学数据，还有其他更普通的信息，但这个观念一直存在于Web出现的早期。然而，在HTML 4.01出现多年之后，动态网站和在线应用程序开始出现，它们主要是通过使用Adobe Flash和其他第三方插件实现的，这些插件允许你创建功能丰富的交互式应用程序。CMS（Content Management System，内容管理系统）和诸如WordPress这样的服务开始兴起，这样，任何人都能够使用模板来创建博客或管理大量内容。Flickr支持照片共享，而YouTube则支持视频共享。而且，在.com泡沫破灭之后，随着人们对因特网的信心的恢复，网店和拍卖网站也开始遍地开花，销售范围包括书籍及其他各种商品。Web中的静态内容越来越少，越来越多的内容都是由用户动态生成的。

本质上，Web已经不再是早期孤立的静态文档了，现在的Web更侧重于使用模板显示大量的动态数据。当然，完全使用这个定义来描述Web也是不对的，但是它强调了Web正在经历一次彻底的转变。问题是HTML从来就不是出于这个目的而开发的，它是为了处理非常严格的基于文档的内容（段落、图像、标题等）而开发的。任何其他的内容，如媒体或CMS内容，都需要使用外部插件（如Adobe Flash）和非标准代码来支持。另外，有时还需要使用其他技术来增强HTML，使之适合于媒体，支持更好、更符合语义的代码。

1.2.2　解决问题

HTML5对旧的面向文档的Web进行了大量的改进。它创建了新的元素来标记动态的模板化内

① http://wiki.whatwg.org/wiki/FAQ#When_will_HTML5_be_finished.3F。

容。另外增加了其他一些元素来替代显示视听内容的外部插件。这些元素本身已经能够解决许多问题，但是W3C（归功于WHATWG）还不满足于此。例如，表单验证（以前只能够通过JavaScript实现）也成为了HTML5的一部分。特别是对于Web上用户生成的内容，相当一部分是使用某些输入表单创建的，所以浏览器的验证比以往任何时候都重要。

与HTML5同时应运而生的其他一些技术一样也都希望解决相关的问题。其中一个例子就是现代的Web比10年前更具移动特性。据估计，95%的手机安装了某种因特网浏览器[①]。越来越多的人希望访问与他们所处位置相关的内容，那么在浏览器中实现地理定位还会没有意义吗？这些技术与HTML5都是为彻底解决上述问题而开发的。

HTML5引入了一整套新功能，W3C和WHATWG网站上的规范对这些功能进行了详细的描述。本书将介绍理解这一规范的基础内容，详细介绍HTML5的重要新特性（尽量通俗易懂）。

> **注意** 精确定义什么是（以及什么不是）HTML5并不容易。这个规范目前的状态仍不稳定，特别是在W3C和WHATWG之间还存在分歧。此外，HTML5最开始定义的许多特性已经发展并转移到单独的规范中，如地理定位和SVG（Scalable Vector Graphics，可缩放矢量图形）。在本书中，HTML5的定义仅参照WHATWG的规范[②]。
>
> 正如我们在本章末尾将要讨论的，许多人把很多实际上不属于这个标准的其他技术（如CSS3）归入HTML5。但是，我们将介绍的canvas元素则确定无疑地属于HTML5规范，所以不需要担心。

1.3 HTML5 的新特性

至此，我们已经知道HTML5增加了许多新元素，其中包括内容结构和媒体。HTML5还引入了一系列新的、更好的特性，例如为表单新增的特性。但是，这些特性具体是哪些，它们的作用是什么呢？HTML5新特性很多，完全可以用一整本书来阐述，而下面几节将主要阐述其中一些最有趣且最重要的特性。

> **注意** 详细解释HTML5的每个细节超出了本书的范围。如果你想要了解更多关于新特性的介绍，可以通读HTML5规范。规范可能会令你眼花缭乱，但如果你想完全理解所有方面，那么里面的内容是非常值得仔细阅读的。

1.3.1 结构和内容元素

可以说，每个网站都会使用某种形式的结构和内容元素。从段落（<p>）到分区（ ），

① TomiAhonen: Mobile Industry Numbers 2010。

② http://www.whatwg.org/specs/web-apps/current-work/multipage/。

这些类型的元素都是Web的基本元素。然而，HTML的问题是它不支持文档概念之外的内容格式。幸好，HTML5引入了许多新元素，它们是专门用来解决这个问题的，它为内容赋予了更多的语义。

1. HTML5结构元素

HTML5中新的结构元素提供了许多用于描述各种网页组成部分的方法。通常，你只能使用大量的div元素和span元素来设计文档结构。现在不再是这样了！现在能够使用更多的元素，如section、header、hgroup、footer、nav、article和aside。每一种新元素都有独特的用途，能够帮助我们把现代网页上最常见的区域一一区分开。

- section元素

可以认为HTML5中的div是section元素。HTML5规范将它描述为网站的一个普通的分块元素。具体来说，它的作用是按主题对内容进行分组，即将内容分组到一些具有独特主题或关注点的区域中。例如，文章正文内容的章节，或者网站上相同页面的区域，如介绍、产品展示和联系表单。

> **注意**　section元素有一条重要规则就是不要将它看做div元素的替代。section是用来定义网站中特定的可区别的区域的。div元素只有在不得已的情况下才使用，通常是在不适合使用其他元素时才使用。

- header元素

无论是显示公司的Logo还是名称，大多数网站都会有某种形式的页眉（header）。以前，页眉与网站的大多数区域一样，很可能都是使用div元素或其他一些并非专门支持这个用途的普通元素创建的。新的header元素给了我们很大的方便，它允许定义一个特定的网站区域，其中可以包含标题、Logo、导航及其他与页眉相关的内容。还可以在网站上添加多个header，这就像在内容中添加多个标题一样。

- hgroup元素

hgroup元素是将标题进行分组的元素，它可用于包含多个标题元素（h1~h6）。hgroup最常见的用法是显示带有标题和小标题的内容。在HTML5之前，包含一组标题的唯一方法是使用普通的HTML 4元素（如div），这种方法的问题是它不具备任何语义。在一个标题元素中包含一个hgroup元素也是很常见的。

- footer元素

版权声明和网站制作者信息通常位于网页的底部。footer元素就是专门用来显示这些内容的，因为大多数网站都会使用某种形式的页脚。footer还可以包含一些相关内容的链接，这意味着它们很适合在section和article中使用。与header元素类似，可以在一个页面中使用多个footer。

- nav元素

你现在可能已经猜到，所有新的HTML5元素出现的原因都是因为人们一直尝试把一些HTML4元素用于该元素设计之外的用途。导航也一样。nav元素的唯一用途是显示指向页面某一

部分或网站其他页面的导航链接。nav元素的最常见用途是包含网站的主导航菜单。它通常位于一个header元素中，旁边是Logo或header元素中的其他常见内容。

● article元素

任何独立成文且可以以其他格式重用（例如，通过RSS分发）的内容都应该置于一个article元素中。这里最典型的例子就是博客文章。你可以删除一篇博客文章周围的其他内容，而它仍然能够保持其原有的意义。博客文章通常也会通过其他格式（如RSS）进行重用和分发。其他类似的例子还有评论、论坛帖子和新闻文章。

> 注意 人们可能不太清楚应该何时使用article元素。Bruce Lawson撰写了一篇很好的博客，对这个问题进行了清晰的阐述[①]。关于article的使用，有一条很好的经验法则，那就是判断其内容在RSS阅读器中是否能作为一个整体独立存在。

● aside元素

最后一个结构元素是aside元素。其用途是包含内容周围的相关内容。它的典型应用是引文和旁注。

2. HTML5内容元素

正如我们所见，HTML5引入了许多能够帮助我们创建网站结构的新元素。此外还有更好的消息，我们能使用的新元素还远不止这些。接下来，我们将关注能够帮助我们对主体内容进行分组和标记的一些新元素，如figure、figcaption、mark和time。

● figure元素

figure元素的一个典型用途是包含图像、代码和其他内容对主内容给出某方面的说明。figure元素中的内容应该可以从主内容中删除而不会破坏主内容。换言之，即使删除figure元素，读者应该仍然能够理解原文内容。

```
<figure>
        <img src="example.jpg" alt="An example image">
</figure>
```

● figcaption元素

有一些注解内容需要使用一个简短的标题，通常这些内容是显示在原始内容的上下文之外的。要在figure元素中加入标题，需要使用figcaption元素。这很简单！

```
<figure>
        <img src="example.jpg" alt="An example image">
        <figcaption>This example image will help you understand.</figcaption>
</figure>
```

● mark元素

那些突出显示以表示引用的内容应该包含在mark元素中。一个例子是在一段引文中，对原

① http://www.brucelawson.co.uk/2010/html5-articles-and-sections-whats-the-difference/。

作者没有加以强调的一句话（可能原作者认为这句话并不重要）进行突出显示。mark元素的另一个用途就是突出显示与用户当前活动相关的内容。例如，如果一个用户来自一个搜索引擎，那么可以用mark元素来突出显示用户在搜索中使用的关键词。

```
<p>This is a great example of <mark>HTML5 canvas</mark>.</p>
```

> **注意**　不要将mark元素与em或strong元素混淆在一起。后者表示原作者认为重要或需要强调的内容。mark元素表示某人在一个不同的上下文中引入原作者的话时认为重要的内容，或者是用户活动的结果。

- time元素

当需要在内容中显示时间或日期时，建议使用time元素。一定要记住，所使用的任何时间都必须采用24小时格式，而日期则必须采用预定义的公历日期，因为这是西方地区大部分人使用的日历。time元素还可以包含两个属性：datetime，表示在元素中指定的确切日期和时间；pubdate，表示文章（或整个文档）发布时time元素所指定的日期和时间。这两个属性都使用YYYY-MM-DD格式的日期，如2010-12-25。当time元素中提到的日期是一些模棱两可的表述时（如，不带年份的24 October），datetime属性就是非常有用的。

```
<time datetime="2011-09-06">My birthday this year</time>
```

1.3.2　表单

如果你曾经有幸处理过表单，那么一定对表单验证和安全检查的复杂性深有感触。你也一定遇到过创建表单时控件严重不够用的情况。例如，你希望使用一个只接收电子邮件的特殊输入框，但却只有一个普通文本输入框。或者，你希望使用一个能弹出日历的输入框，这样用户就能够选择日期，但却需要花上半辈子的时间用JavaScript来实现它。当然，实际情况可能并没有那么糟糕，但是毫无疑问，验证表单和实现智能功能的唯一方法就是使用JavaScript。然而，从Web Forms 2.0开始，HTML5彻底改变了这一切。

1. 浏览器端验证

在HTML5表单特性中，我最喜欢的是浏览器内置的验证机制。这意味着我们不再需要JavaScript，这是Web设计人员和开发人员乐于见到的。目前采用JavaScript实现的问题是无法保证所有人都在浏览器上启用JavaScript。这就是为什么必须同时使用服务器端验证（如PHP）和客户端验证（如JavaScript）。使用浏览器内置的验证不仅能够解决客户端的问题，而且还能够节省很多编写JavaScript验证代码的时间。

默认情况下，如果浏览器支持HTML5，那么表单验证都是启用的，不过可以在form元素中添加novalidate属性来关闭验证。在撰写本书时，完全支持HTML5表单的浏览器只有Opera（9.5及以上版本），如果验证失败，它会显示一条友好的警告信息。其他浏览器，如Safari和Chrome，目前只有不太完善的实现，实际上并没有什么用处——对验证错误都没有可视化的反馈。幸好，浏览器支持正迅速完善起来，所以很可能当你阅读本书时，所有浏览器都已经支持表单验证了。

2. 输入类型

以前编写过表单的人都应该知道输入类型。input元素（及其他元素，如select、textarea等）通常用来获取用户在表单中的输入。通过设置input元素的type属性，就能够使这个表单输入元素实现一些有趣的功能。例如，当输入密码（type = "password"）时，向用户显示可供选择的复选框（type = "checkbox"）时，或者将它变成一个提交按钮（type = "submit"）时，将显示不同的外观。input实际上是HTML表单中最强大的控件。

利用HTML5中新的输入类型，可以在按钮、复选框和文本框之外获得更多的控件。我们现在能够定义电子邮件地址、电话号码和URL的输入框。另外还有一些新的输入类型，如datetime，让我们可以使用类似于日历的界面来选择日期。此外，range控件则允许我们将用户输入的数字限制在一对最小值和最大值之间。此外，color类型允许通过与图形应用程序常用的颜色拾取器类似的界面来选择一个颜色值。HTML5规范所选择实现的控件有些不可思议，但大多数都绝对有用！

3. 输入属性

除了输入类型，input元素还可以使用许多其他的属性。这其中包括在输入框中预设提示信息的placeholder，以及切换是否基于以前填写过的数据为用户输入自动填充的autocomplete属性。

表单还有很多其他的输入类型、属性和特性。然而，虽然表单确实很有趣，但我们最好还是把更多的篇幅用来介绍与本书有关的其他HTML5特性。没错，我指的是更强大的媒体元素！

1.3.3 媒体元素

HTML5出现之前，在浏览器中播放媒体需要借助外部插件和应用程序——这是唯一的方法。浏览器本身不支持图形处理，你只能嵌入编辑好的图像。HTML5的一个核心目标是改进媒体支持，而它也正是这样做的。新的元素（如audio和video）已经引入，浏览器不需要使用外部插件（如Adobe Flash和Microsoft Silverlight）就能够直接播放媒体。新增加的canvas元素使浏览器能够直接创建和处理图像和图形。

所有这些新增加的媒体元素的一个最重要特点是它们都具有开放的JavaScript API，它们能够简化我们对媒体内容的控制。想要用一个自定义按钮重播视频吗？没问题。能够随时给一些canvas图形添加动画效果吗？很简单。HTML5的媒体功能是非常强大易用的，而且不需要依赖专用技术就能够创建这些媒体内容。在浏览器中实现这种控制这还是有史以来头一次。

1. audio元素

当你开发网站时，会经常需要使用音频播放器吗？大多数人对这个问题的回答都是从不或很少。毫无疑问，音频播放器是Web开发中很少见的需求，至少目前采用Flash格式的音频是很少的。这个现象源于20世纪90年代时大量网站使用自动播放的背景音乐而造成的负面影响。那么，为什么HTML5中有一个专门的audio元素呢？这是因为目前的方法要求你使用第三方的插件，仅此而已！audio元素最强大的地方是它不需要在页面上呈现就能够使用。这种做法的优点是它能够

补充网站的其他特性，而又不会影响视觉效果。例如，可以使用一个没有用户界面的audio元素来实现游戏的音效——我们将在本书的后面演示这个方法。

目前，主流浏览器对audio元素的支持各不相同。这些浏览器总共采用了5种音频编解码器，但是没有一种编解码器是所有浏览器全都支持的。其中最常用的格式是MP3和OGG（一个开源的免费编解码器）。现在实现跨浏览器HTML5音频支持的最佳方法是针对两种以上的编解码器提供不同的音频源。这不是最理想的方法，但是这种方法非常简单——只需要将audio元素的src属性替换成一个或多个source元素即可。

```
<audio controls>
       <source src="http://yourwebsite.com/sound.ogg">
       <source src="http://yourwebsite.com/sound.mp3">
       <!- 在此插入后备音频内容，比如Flash 播放器->
</audio>
```

希望将来有一种编解码器会得到所有主流浏览器的支持。但是，现在我们只能采用这种方法进行处理。

新媒体元素最方便的地方是很容易为不支持新特性的浏览器添加后备内容，如Flash播放器。而后备内容是通过在媒体元素中添加普通的HTML元素实现的。

除了已经介绍过的src，还可以使用其他一些属性对audio元素进行细粒度的控制。首先是controls属性，它可用来指示浏览器提供默认的音频内容播放控制界面（参见图1-1）。

```
<audio src="http://yourwebsite.com/sound.ogg" controls>
        <!-在此插入后备音频内容，比如Flash 播放器->
</audio>
```

图1-1　Google Chrome浏览器的HTML5音频控件

如果希望音频循环播放一次，可以使用loop属性。但是，需要注意的是，不同的浏览器实现循环播放的方式是不同的。

```
<audio src="http://yourwebsite.com/sound.ogg" controls loop>
        <!-在此插入后备音频内容，比如Flash 播放器->
</audio>
```

所有属性都很简单。preload可以指示浏览器如何预加载音频。

```
<audio src="http://yourwebsite.com/sound.ogg" controls preload="auto">
        <!-在此插入后备音频内容，比如Flash 播放器->
</audio>
```

最后一个是autoplay属性。它允许你设置在浏览器加载完成时自动播放音频内容（请在使用这个功能时考虑一下用户的感受）。

```
<audio src="http://yourwebsite.com/sound.ogg" controls autoplay>
        <!-在此插入后备音频内容，比如Flash 播放器->
</audio>
```

> **注意** Mozilla开发了一个Audio Data API，它支持对音频数据进行直接处理。可以使用它在代码中直接读写原始音频内容。它的使用示例包括音频合成器和可视化，与在iTunes中看到的功能类似。目前，只有Firefox支持这个API，但是它是表明浏览器现在所支持的媒体功能的一个很好的信号。[①]

2. video元素

音频功能很不错，但是在音频之上再增加移动画面（视频）就更好了。与音频不同，视频内容在Web上应用极为广泛，在浏览网站时很难不看见视频。然而，与音频类似，目前实现视频内容的方法都需要使用外部插件，如Flash。同样，与音频类似，目前的方法过于臃肿和混乱，所以为了扭转这一局面在HTML5中引入一个专门的video元素就不足为奇了。

认为在HTML5中实现视频会比实现音频难也是情有可原的。但实际上，这两种媒体元素源于相同的设计思想，这意味着它们使用相似的属性并且是采用相似的方法实现的。一个简单的video元素如下所示（参见图1-2）。

```
<video src="http://yourwebsite.com/video.ogv" controls>
        <!--在此插入后备视频内容,比如 Flash 播放器-->
</video>
```

是不是与音频的例子很像？HTML5的一致性绝对是第一流的！

图1-2 Google Chrome中的HTML5视频和控件

有许多种视频编解码器可供使用，但是，与音频类似，仍然没有任何一种编解码器是所有浏览器都支持的。要解决这个问题，可以采用与audio元素一样的方式，即使用source元素。

① https://wiki.mozilla.org/Audio_Data_API。

```
<video controls>
        <source src="http://yourwebsite.com/video.ogv">
        <source src="http://yourwebsite.com/video.mp4">
        <!-在此插入后备视频内容,比如 Flash 播放器->
</video>
```

与对音频编解码器的期望类似，但愿将来有一种编解码器能够被所有浏览器支持。这样播放视频就更简单了。

HTML5的媒体功能很强大，video和audio元素都采用了相同的属性，如src、loop、preload和controls。video元素还具有一些只适用于视频内容的特殊属性。

要在视频无法播放时显示图像，可以使用poster属性。

```
<video src="http://yourwebsite.com/video.ogv" controls
poster="http://yourwebsite.com/video'poster.jpg">
        <!-在此插入后备视频内容,比如 Flash 播放器->
</video>
```

要使视频默认为静音播放，可以使用audio属性。

```
<video src="http://yourwebsite.com/video.ogv" controls audio="muted">
        <!-在此插入后备视频内容,比如 Flash 播放器->
</video>
```

如果要为视频定义特定的宽度和高度，可以使用width和height属性。

```
<video src="http://yourwebsite.com/video.ogv" controls width="1280" height="720">
        <!-在此插入后备视频内容,比如 Flash 播放器->
</video>
```

由于较早的浏览器对编解码器缺乏支持，HTML5视频还尚未成为主流，但是这并不意味着不能使用这个功能。有一些大型的视频提供商，如YouTube和Vimeo，已经开始全面实验HTML5视频，以便为HTML5的成熟和未来采用做好准备。无论如何，如果一定要在网站上播放视频，还是应该使用传统的Flash视频作为后备。

> **注意**　HTML媒体如此重要的原因在于它支持一种完全开放的音频和视频实现方法。以前只能使用封闭的系统，如Flash，虽然它的功能很全面，但是很难在插件以外对媒体进行控制。HTML5并没有这样的限制，它允许你根据需要在任何时候采用任何方式对媒体进行控制和操作。请参考Silvia Pfeiffer的*Definitive Guide to HTML5 Video*（Apress）了解HTML5视频处理的详细介绍。

3. canvas元素

毫无疑问，我最喜欢的HTML5特性是canvas元素，这也是撰写本书的原因，在一定程度上，我也希望这是你阅读本书的原因。这个元素与其他元素是非常不同的，它的主要用途是处理或者从头创建2D图形，而不是像嵌入audio和video元素那样直接将现有媒体嵌入到网页中。我想你可能会将它想象成构建在浏览器中的Microsoft画图程序。但是不要担心，我可以向你保证它比你

想象的要强大得多，而且实际上，它一点儿也不像Microsoft画图程序。我认为应该将它看做一个2D图形环境，而不是简单的嵌入媒体的容器。

> **注意** 你知道吗？苹果公司最初发明canvas元素的目的是为了开发Dashboard 微件，但是它很快吸引了其他浏览器开发商的注意，并且最终将它直接加入到HTML5规范中。

如果我不详细介绍Canvas的深层发展潜力，那么对它来说是不公平的。是的，它可能只是一个2D平台（就目前而言），但这并不意味着它不能做出一些令人惊叹的东西。例如，通过使用JavaScript API来操作canvas元素，可以创建响应用户交互的动态图形和动画（如游戏——有一些Google玩家甚至将第一人称3D射击游戏Quake II移植到Canvas中实现）。可以使用它来基于HTML表格中的数据创建动态更新的数据可视化图形。还可以使用它来构建一个Web应用程序的用户界面，尽管我可能更愿意推荐你采用传统的HTML和CSS。这里我想说明的是canvas元素是很简单易用的，但是通过JavaScript API和一些创新运用，Canvas本身可以成为一个能够创建动态图形和交互体验的强大工具。简而言之，Canvas确实令人兴奋！

> **注意** 你可能已经注意到了，我使用了canvas元素和Canvas两个概念。这两个概念之间是有一些细微差别的。当直接讨论HTML5元素及其特性但不涉及JavaScript API时，我会使用canvas元素。当从整体角度讨论整个与Canvas相关的特性时（包括canvas元素、JavaScript API以及你在使用它的过程中得到的非凡体验等），我会使用Canvas（无强调语气）。

前面就是所有你需要知道的关于canvas元素的信息。现在你就可以开始创建这些炫丽的图形和游戏了。但是，就只有这些了吗？我猜你会问这个问题。如果是这样，你就问对了。确实不止这些。关于Canvas及其功能还有许多内容要讲，而且即使讲了这些，我们甚至都还没有触及它的表面。后面的章节将详细介绍这个元素的强大功能和使用方法。你很快就会精通Canvas的使用。

1.4 剖析 HTML5 页面的结构

了解HTML5的新元素当然很好，但脱离上下文你是很难想象它们的使用方法的。而这正是我们接下来要做的事情。首先展示一个HTML5示例页面，然后逐行分析，了解新元素在实际环境中是如何使用的。这个示例页面是一个虚构的博客首页，其中包括一个页眉、一些博客文章、一个附注和一个页脚。我们不会为页面设置样式，所以暂时不涉及CSS及其他有关表现的属性。我们将只关注如何设计一个HTML5网站的结构。

下面是这个HTML博客首页的完整代码。如果觉得代码有点多，请先不要担心。

```
<!DOCTYPE html>

<html>
        <head>
                <title>A basic HTML5 blog homepage</title>
                <meta charset="utf-8">
                <!-- CSS 及 JavaScript 代码放在这里-->
        </head>

        <body>
                <header>
                        <!--网站名称及导航-->
                        <h1>My amazing blog</h1>
                <nav>
                        <ul>
                             <li><a href="/">Home</a></li>
                             <li><a href="/archive/">Archive</a></li>
                             <li><a href="/about/">About</a></li>
                             <li><a href="/contact/">Contact</a></li>
                        </ul>
                </nav>
          </header>

          <section>
                <!--博客文章-根据需要可重复出现多次-->
                <article>
                        <header>
                               <hgroup>
                                      <h1><a href="/blog/first-post-
link/">Main heading of the first blog post</a></h1>
                                     <h2>Sub-heading of the first blog
post</h2>
                               </hgroup>
                               <p>Posted on the <time pubdate datetime=➥
"2010-10-30T13:08">30 October 2010 at 1:08 PM</time></p>
                               </header>

                               <p>Summary of the first blog post.</p>
                        </article>

                        <article>
                               <header>
                                      <hgroup>
                                             <h1><a href="/blog/second-post-
link/">Main heading of the second blog post</a></h1>
                                             <h2>Sub-heading of the second blog
post</h2>
                                      </hgroup>
                                      <p>Posted on the <time pubdate datetime=➥
"2010-10-26T09:36">26 October 2010 at 9:36 AM</time></p>
                               </header>

                               <p>Summary of the second blog post.</p>
```

```
                    </article>

                    <article>
                        <header>
                            <hgroup>
                                <h1><a href="/blog/third-post-
1link/">Main heading of the third blog post</a></h1>
                                <h2>Sub-heading of the third blog
post</h2> </hgroup>
                            <p>Posted on the <time pubdate datetime=➥
"2010-10-21T17:13">21 October 2010 at 5:13 PM</time></p>
                        </header>

                        <p>Summary of the third blog post.</p>
                    </article>

                    <!-- Blog sidebar -->
                    <aside>
                        <h2>Subscribe to the RSS feed</h2>
                        <p>Make sure you don't miss a blog post by➥
<a href="/rss">subscribing to the RSS feed</a>.</p>
                    </aside>
                section>

                <footer>
                    <!--版权信息及其他内容-->
                    <p>My amazing blog &copy; 2010</p>
                </footer>
            </body>
</html>
```

逐行分析

首先，我们从第一行代码开始分析。

```
<!DOCTYPE html>
```

这是新的HTML5文档类型声明，虽然看起来有点像元素，但它实际上不是元素。文档类型的目的是告诉浏览器我们使用哪个版本的HTML（或XHTML）——如何呈现以及使用哪种类型的验证。在HTML 4.01中，文档类型声明是非常不同的，其中包括你是否使用一种严格的、过渡的或带框架集的文档类型。例如：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

太长了，不是吗？幸好，HTML5文档类型简单得多，它只包含最简单的文本“html”。新的文档类型的另一个优点是，当遇到不支持HTML5的浏览器时，它不会影响显示，它会转换为所谓的标准模式——以尽可能接近W3C标准的方式呈现页面。所以，你完全可以马上使用HTML5文档类型声明。

```
<html>
```

html元素是页面的根，其他元素都放置在这个元素之中。每个页面都只能包含一个html元素。

```
<head>
```

我们页面的所有meta元素都位于head元素之中。每个页面都只能包含一个该元素。

```
<title>A basic HTML5 blog homepage</title>
```

通过title元素给页面设置名称，它会显示在浏览器窗口的标题栏中。每个页面都只能包含一个该元素。

```
<meta charset="utf-8">
```

meta元素一般用于表示页面的其他元数据。charset属性声明了我们希望使用的字符编码方法。除非你有合理的原因使用其他编码方式，否则一定要使用UTF-8字符编码。

```
<!-- CSS及JavaScript代码放在这里-->
```

这是一个HTML注释。应该将它替换成对CSS和JavaScript文件的链接。

```
</head>
```

页面的元数据设置完成，结束head元素。

```
<body>
```

页面的主体内容位于body元素中。每个页面也只能包含一个该元素。

```
<header>
```

页面中所有与页眉相关的内容，包括网站名称和导航，都必须位于HTML header元素之中。

```
<h1>My amazing blog</h1>
```

这个h1元素会在页面上显示网站名称。

```
<nav>
```

网站导航放在HTML5 nav元素之中。

```
<ul>
```

使用一个无序列表来组织导航菜单。

```
<li><a href="/">Home</a></li>
<li><a href="/archive/">Archive</a></li>
<li><a href="/about/">About</a></li>
<li><a href="/contact/">Contact</a></li>
```

每一条导航都是列表项（li元素）中的一个链接（href元素）。如果不添加样式，便会从上到下逐一列出导航项。

```
</ul>
```

所有的导航项均已完成，结束这个无序列表。

```
</nav>
```

网站导航已经完成，结束导航元素。

```
</header>
```

同样，结束header。

```
<section>
```

我们使用一个HTML5 section元素来显示博客文章。然而并非一定要这样做，如果我们知道网页上只有博客内容，那么完全可以删除这个section元素。

```
<article>
```

为了区分每一篇博客文章，可以将它们置于一个HTML5 article元素之中。记住，我们使用article元素来显示能够在RSS阅读器上独立阅读的内容。

```
<header>
```

现在添加博客文章的标题。

```
<hgroup>
```

由于我们在博客文章中使用了多个标题，所以将它们包含在一个HTML5 hgroup元素中。

```
<h1><a href="/blog/first-post-link/">Main heading of the first blog post</a></h1>
<h2>Sub-heading of the first blog post</h2>
```

这里声明了博客文章的主标题，包括链接和小标题。

```
</hgroup>
```

标题完成了，结束hgroup元素。

```
<p>Posted on the <time pubdate datetime="2010-10-30T13:08">30 October 2010 at 1:08
PM</time></p>
```

还是在页眉中，使用HTML5 time元素输出博客文章发布的日期和时间。注意我们设置pubdate属性的方式，该该属性在此表明这里的时间是这部分内容发布时的日期。此外，要注意使用datetime属性时需要指定格式正确的日期和时间版本。

```
</header>
```

博客文章的标题完成了，结束header元素。

```
<p>Summary of the first blog post.</p>
```

用一个简单的段落来显示博客文章的摘要。

```
</article>
```

我们完成了第一篇博客文章，可以结束article元素了。其他博客文章的处理方式是完全相同的，这里不再一一复述。

```
<aside>
```

要创建相关的内容，可以使用HTML5 aside元素。与section元素类似，使用这个元素是因为附注是与博客文章的内容直接关联的。

```
<h2>Subscribe to the RSS feed</h2>
```

这是附注的标题。请注意这里使用的是h2元素，因为这个标题在这个页面上不是最重要的。

```
<p>Make sure you don't miss a blog post by <a href="/rss">subscribing to the RSS
feed</a>.</p>
```

这是一个简单的段落，其中包含虚构的RSS源的链接。

```
</aside>
```

现在完成了附注，结束aside元素。

```
</section>
```

所有与博客文章相关的内容都已经完成，结束section元素。

```
<footer>
```

我们将所有的页脚内容，如版权声明，置于一个HTML5 footer元素之中。

```
<p>My amazing blog &copy; 2010</p>
```

用一个段落来显示页面的版权声明。

```
</footer>
```

这就是页脚的全部内容，结束footer元素。

```
</body>
```

页面的所有内容均已完成，结束body元素。

```
</html>
```

至此，我们完成了页面的所有内容，最后结束html元素。

不算复杂，对吧？我希望你现在觉得使用HTML5并不是一件很难的事。主要问题是理解所有新元素的作用是什么，什么时候使用它们，以及为什么要使用这种元素而不是另一种元素，最后一点最重要。掌握HTML5的最佳方法是使用新元素构建一个网站。在自己的网站上尝试HTML5，甚至不需要将它部署上线，只将它转换成HTML5就足以理解它了。

1.5　对HTML5的误解

对于所有新技术而言，肤浅的理解往往会导致对它的目的及作用产生误会。HTML 5也不例外，关于它的特性和其他技术，很多人都会有理解不当和混淆的时候。这种问题的普遍流行，使人们很难把握什么是HTML5（以及什么不是）。甚至有时候我自己都觉得很困难。所以，我认为有必要对HTML5的最常见的误解进行澄清。我希望掌握了这些信息之后，你能够更进一步，不仅增进自己的理解，还能帮助整个Web社区。

1.5.1　CSS3误解

支持HTML的CSS是从1996年开始出现的，它是一项老技术。最新版本CSS3是从2005年开始开发的，但它距离W3C的正式推荐还差得很远。事实上，它还没有完成开发，也还未被推荐作为

一个大的规范。它分成许多个独立的模块，在这些模块还在开发时，就有浏览器在同时实现它们了。因此，尽管与HTML5一样，大多数浏览器都已经支持CSS3的部分特性，但是很难说CSS3会在什么时候100%完成开发。

对CSS需要注意的重要方面是，它在开发和使用方面都是与HTML独立的。CSS3不属于HTML5规范（从来就不是，以后也不是）。它们是两种完全不同的技术，HTML5是用来组织结构和布局的，而CSS是用来呈现的。由于它们的使用场景很接近，因此经常被混为一谈。到底什么意思呢？不要将CSS3说成是HTML5。

CSS3属于HTML5吗

不——它是应用样式的技术，与内容或结构无关。可以在这个网站查找更多关于CSS3的信息：www.w3.org/Style/CSS/current-work。

1.5.2 Web Fonts 误解

Web设计的一个不足是很难使用自定义字体。直到最近，我们能使用的唯一方法就是创建静态图片来描述所希望使用的单词，或者使用复杂的系统，如sIFR或cufon。Web Fonts将`@font-face`规则引入CSS，这使我们通过简短的代码就能够使用自定义字体。非常简单！

Web Fonts属于HTML5吗

不——它属于CSS3，并且技术上已经不再称为Web Fonts。可以在这个网站上查找更多关于CSS Fonts的信息：http://dev.w3.org/csswg/css3-fonts/。

1.5.3 Geolocation 误解

想象一下，不管你在哪里，都可以在Web上自动获取与你当前位置有关的信息。Geolocation API就支持这个功能，它通过一些简单的JavaScript向Web开发人员提供一种了解用户位置的方法。用户必须允许共享他的位置信息，但是这样做会带来很多可能，如相关广告、有用的搜索结果和相关网站内容。所有这些都在不需要人为输入或选择位置的情况下完成。

Geolocation API属于HTML5吗

不——它是一种JavaScript API，是由想支持它的浏览器实现的。可以在这个网站上查找更多关于Geolocation API的信息：http://dev.w3.org/geo/api/spec-source.html。

1.5.4 SVG 误解

SVG是一种允许你使用XML创建2D矢量图形的语言。它在功能和用途上与Canvas非常类似，但是在其他方面有很大区别（将在第10章讨论）。

SVG属于HTML5吗

不——它是一种使用XML来描述图形的完全独立的技术。可以在这个网站查找更多关于SVG的信息：www.w3.org/TR/SVG/。

1.5.5　Web Storage误解

Cookies[①]，是大家都喜欢的零食，同时也是在用户计算机上存储少量信息的方法。直到现在，它仍然是唯一一个在客户端（用户计算机）上存储数据的有效方法，但是它有一些小缺陷可能会导致许多问题。Web Storage是一组JavaScript API，它的目的是实现一组强大的新存储方法，解决使用cookie的内在缺陷。表面上看它们没什么新奇之处，但是它们提供了一些很好的功能，如离线查看基于浏览器的电子邮件收件箱。这是很棒的！

Web Storage属于HTML5吗

不——它是一个JavaScript API，是由想支持它的浏览器实现的。可以在这个网站上查找更多关于Web Storage的信息：http://dev.w3.org/html5/webstorage/。

1.5.6　Web Workers误解

Web Workers实际上是一个不知疲倦的JavaScript“工作狂”，随时准备执行你的命令。它们的唯一用途是在后台执行一些繁重的计算及其他高强度的任务，从而不会导致网页显示减慢或影响用户体验。Web Workers的使用场景并不多，但是了解它们的用处是很有帮助的。

Web Workers属于HTML5吗

不——它是一个JavaScript API，是由想支持它的浏览器实现的。可以在这个网站上查找更多关于Web Workers的信息：www.whatwg.org/specs/web-workers/current-work/。

1.5.7　WebSocket误解

一般的Web通信是通过HTTP实现的，HTTP是一种一次只允许在一个方向进行通信的方法，每当你需要获取新数据时，都需要请求一个网页。为了解决这个问题，Web开发人员使用许多方法来回避请求新页面。诸如Ajax和Comet等技术正是为此目的而诞生的。然而，它们仍然不是真正的双向通信方法——每次信息都是在一个方向传输的。

WebSocket则不同，它使用支持在客户端（你的计算机）和服务器之间实现真正的双向通信的TCP协议。这意味着你不需要重新发送新的数据请求到服务器，因为当新数据出现时，信息是实时传输到你的计算机的。这是一个复杂的概念，但是一旦掌握了这种技术，它就是一种非常强大的工具。

WebSocket API属于HTML5吗

不——它是一个JavaScript API，是由想支持它的浏览器实现的。可以在这个网站查找更多关于WebSocket API的信息：http://dev.w3.org/html5/websockets/。

虽然以上这些技术都不属于HTML5规范，但是它们都有各自的独特用途，且我们应该欢迎这些功能并尽量与HTML5结合使用。本节主要是强调这些技术为什么不属于HTML5，而不是劝说你不要去使用它们。例如，通过组合使用WebSocket和HTML5 Canvas，可以创建出色的实时多人游戏。这是两种不同技术的完美组合。

① “饼干”。——编者注

1.6 小结

本章介绍了HTML的发展历史。其时间跨度比较大，从1999年HTML 4.01推荐标准，直到HTML5的开发，HIML5现在仍在开发中。解释了需要HTML5的原因，以及它是如何满足这些需求的。介绍了HTML5的全部主要的新元素和新特性，包括它们的用途和使用方法。还讲述了如何使用这些新特性来构建HTML5网页。最后，划清了HTML5与其他Web新技术的界限。

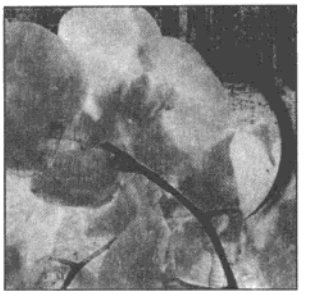
本章文字内容较多（祝贺你坚持读完了），我向你保证我们已经讲完基础知识了，后续几章将会讲更多动手实践的内容。毕竟，你阅读本书是为了学习新技术，而不是听我啰嗦HTML5历史。

下一章，我们将介绍JavaScript基础。

更多的 HTML5 资源

如果你想了解更多关于HTML5历史的信息，或者想了解更多我们还未深入讨论的元素和特性，下面是一些可供你参考的资源。

- *Introducing HTML5*，作者Bruce Lawson和Remy Sharp [http://introducinghtml5.com/]。
- HTML5 Doctor [http://html5doctor.com/]。
- *Dive Into HTML5*，作者Mark Pilgrim [http://diveintohtml5.org/]。
- WHATWG HTML5规范 [www.whatwg.org/specs/web-apps/currentwork/multipage/]。



第2章

JavaScript基础

在本章中，我们将深入探讨一下JavaScript。首先讲述它的历史和不同实现。接下来,介绍jQuery代码库，以及它所能带来的开发效率。本章剩下的内容将逐一介绍JavaScript的基本特性，包括在使用HTML5 Canvas进行游戏开发时所需要的全部特性。本章的内容很多，我的出发点是尽可能详细地介绍这些特性，以便帮助你顺利掌握JavaScript。

2.1 JavaScript概述

在第1章中，我们提到了JavaScript。我们知道它是一门脚本语言，但是它到底是什么？能够做什么？接下来，我将逐一解释这些问题。在本章中，我们将结合相关背景讲述JavaScript的演变过程。

通过与Sun Microsystems公司合作，在1995年，浏览器开发公司Netscape（还记得这家公司吗？）的Brendan Eich创造了JavaScript。刚开始时，它的名称实际上是Mocha，后来被重命名为LiveScript，最后在1995年12月改为JavaScript。最初创造JavaScript是为了简化Web开发人员的工作，因为当时唯一能够实现动画或其他比较酷效果的工具是Java：一门相当复杂的编程语言。问题是，在浏览器中使用Java的唯一方法是将代码编译（并打包）为一个applet。这种方法很好，但是这意味着applet内部的所有代码都与外界无关，它位于一个类似于被围墙封闭的环境中。

JavaScript与Java最大的区别是代码不需要编译，它是嵌入到浏览器中的，并由浏览器解释。这意味着不需要创建大量代码就能够实现所需要的功能。只需要一小段代码，就可以开始执行它。简而言之，JavaScript的设计比严格意义上的编程语言（如Java）更简单和宽松。它特别适合那些可能从未使用过传统编程语言的Web开发人员，以及那些只希望快速地给网站增加一些特效的人。重要的一点是，JavaScript与Java没有任何关系，它们只是名称上有些相似。

> **注意** JavaScript代码是由浏览器解释的，这意味着所有代码在运行时都由浏览器进行分析。相比之下，编译后的代码一般是转换成一种可以直接运行的应用程序，而不需要对代码进行解释。

如果技术公司之间不爆发法律纠纷，互联网的故事就不完整。JavaScript的故事也是这样。在Netscape发布JavaScript之后，微软很快就在它的IE浏览器上实现了这种流行的脚本语言。但是，微软并没有得到拥有JavaScript商标的Sun Microsystems的授权，所以它被迫将其命名为JScript。这给一些开发人员造成困惑，他们一开始认为JavaScript和JScript是两种完全不同的语言。实际上，这两种语言几乎完全相同，虽然它们各自都具有一些对方所不具备的功能。需要注意的是，除了IE之外的所有主流浏览器（Firefox、Chrome等）都使用JavaScript，而不使用JScript。在本书中，当我提到JavaScript时，它同时代表Netscape的JavaScript和微软的JScript。

> **注意** 你在Web开发过程中可能还听说过ECMAScript。它的名称与JavaScript如此相似，是因为ECMAScript是一种标准化的脚本语言，它来源于Netscape的JavaScript。支持ECMAScript规范是JavaScript和JScript的共同目标。

2.2 jQuery

如果你没听说过jQuery，那么你现在可能有些困惑。这是什么？不要紧，归根到底它是一种能够帮助你提高开发效率的工具。现在请让我细细道来。

2.2.1 jQuery是什么

jQuery是一个JavaScript库。它实际上是对JavaScript中最复杂和最耗时的任务的一个简单封装，如遍历DOM（Document Object Model，文档对象模型）、事件处理和动画。先不要担心这些概念，后面将会对它们进行详细介绍。jQuery的官方宣传语说得好："jQuery是改变JavaScript编写方式的工具。"这句话的意义深远，不是吗？换言之：jQuery使你能够用更少的代码做更多的事。对我而言，这是非常棒的！

2.2.2 为什么要使用它

不借助类似于jQuery的库，使用纯JavaScript语法编写代码不是一件简单的工作。大部分核心特性在所有主流浏览器中都是显而易见的，这很好。但不好的是许多浏览器在实现其他一些特性时采用的方法却有些不同。例如，检测HTML文档什么时候完成加载，这是一个非常重要的任务（将在后面详细介绍）。但是，没有任何一种方法是在所有主流浏览器上都能用的，这就很难保证JavaScript代码对所有用户都产生相同效果。jQuery提供了执行这些任务的功能，能够在所有主流浏览器中都表现一致。例如，通过一行jQuery代码，就能够检查一个HTML文档是否加载完成，而如果使用纯JavaScript代码，支持所有的浏览器可能需要几十行。jQuery在这里并没有做什么神秘的操作（它实际上也是使用一些纯JavaScript代码），它只是封装了所有代码，帮你完成所有复杂的浏览器检查。它能够帮助你实现所需要的功能，同时可靠地支持所有主流浏览器。

> **注意**　应该选择jQuery还是其他的JavaScript库（如Prototype和YUI）并没有统一的标准。我个人认为jQuery使用起来非常简单，而更重要的是学习起来也很简单。我没有理由向你讲述一个我还没有完全精通的JavaScript库。如果你已经使用过其他的库，那么可以不用jQuery，将它的代码替换成你所选库的等价代码即可。

2.2.3　这是在误导你吗

有些人会认为通过jQuery等库来学习JavaScript并不是一种好方法，特别是那些已经学会了编写纯JavaScript代码的人。他们的观点是，如果你只学习jQuery的方法，那么就不能真正理解JavaScript的工作方式，例如，出现错误应该如何解决。虽然我在一定程度上认同这个观点（不理解JavaScript的工作方式是错误的），但是我不认为同时学习类似于jQuery的库有问题。可以这样说：事实上，Web开发人员的时间很宝贵，他们更关心实现想要的功能，而不关心它是否会在其他浏览器上出错。因此，大多数使用JavaScript的人都会使用一种库来简化这部分工作。那么，为什么我要教给你一种故意使问题复杂化而且与人们实际工作方法不同的JavaScript编程方法呢？

2.2.4　是否不需要理解纯 JavaScript

不是。在本书中，我只有在需要时才使用jQuery。我将向你讲述纯JavaScript语言的基础内容，并使用jQuery来简化你对知识的理解。每当使用一个新的jQuery特性时，我都会解释它的作用，并介绍使用纯JavaScript实现的等价方法。我这样做不仅仅是为了比较两种方法，还是为了帮助你理解JavaScript的实现方式，并且证明jQuery之所以能够简化编码的原因。最终，jQuery就是JavaScript，但是它并不是只提供制作JavaScript这个蛋糕的材料，如面粉、鸡蛋、奶油和糖，而是直接给你一个制作好的蛋糕。因此，通过使用jQuery，你实际上也同时使用和学习了纯JavaScript技术。

2.2.5　如何使用 jQuery

在项目中使用jQuery是非常简单的，只需要一行代码就足够了。只需要将jQuery文件链接到HTML文档中。如下所示：

```
<script type="text/javascript" src="jquery.js"></script>
```

现在，你需要转到jQuery网站，下载程序库，然后将它保存到HTML页面所在的同一个文件夹中。这都是非常简单的操作。或者，你可以将收集工作交由Google完成，使用托管在Google服务器上的jQuery库文件。第二个方法看起来有些奇怪，但是它实际上是很有用的。如果Web开发人员通过Google Libraries API使用Google托管的库文件，那么这些网站的用户就可以使用缓存在他们自己计算机上的jQuery版本。换句话说，如果他们曾经访问过其他也使用Google托管的jQuery

的网站，他们就不需要重新下载jQuery库文件。简言之，它能够加快用户的访问速度，这是非常好的。然而，需要注意一个问题：如果你使用Google托管的文件，那么在测试JavaScript时，你需要连接因特网。如果你希望以离线方式进行测试，那么需要下载jQuery，并使用前一个例子所描述的方法。

要使用Google托管的版本，只需要修改上一个例子的src属性：

```
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1/
jquery.min.js"></script>
```

你可能注意到了，jQuery文件是以.min结尾的。这意味着这个文件是经过压缩的，即它已经压缩到最小了。当你完成了网站开发之后，使用压缩版的jQuery通常是很好的做法，因为下载它的速度更快，而且它的大小不到未压缩的原始版本的1/7。

> **注意** 请访问jQuery网站了解更多关于这个库的信息。如果你希望了解更多高级特性，网站上的文档非常有帮助：http://jquery.com。

2.3 在 HTML 页面上添加 JavaScript

你现在应该知道了什么是jQuery及为什么需要使用它了，那么让我们开始动手创建一个包含JavaScript的基本的HTML页面。

```
<!DOCTYPE html>

<html>
        <head>
                <title>Adding JavaScript to a HTML page</title>
                <meta charset="utf-8">

                <!-- CSS 代码放在这里-->

                <script type="text/javascript" src="http://ajax.googleapis.com
/ajax/libs/jquery/1/jquery.min.js"></script>

                <script type="text/javascript">
                        $(document).ready(function() {
                                alert("Hello, World!");
                        });
                </script>
        </head>

        <body>

        </body>
</html>
```

首先，你会注意到我们正在使用与前面例子相同的HTML5代码。唯一的区别是我们删除了所有内容，并添加了两个script元素到head元素中。这就是放置JavaScript代码的地方。

script元素允许我们在HTML页面上添加JavaScript及其他脚本代码。我们使用第一个script元素从Google引入一个外部文件：

```
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1/
jquery.min.js"></script>
```

要执行这个操作，我们首先需要将type属性设置为“text/javascript”，定义我们所使用的脚本类型。这能够使浏览器知道我们将提供的脚本是JavaScript，而不是其他脚本。然后，我们将src属性设置为Google URL，从而引用jQuery库文件。src属性会告诉浏览器从一个外部位置获取脚本文件——外部指的是当前的HTML页面文件之外。当然，也可以使用src属性从本地文件系统获取脚本。

第二个script元素并没有src属性，因为我们将在元素中直接编写JavaScript代码。这可能听上去很简单，但一定要分清它们的区别。如果你在HTML页面中直接编写JavaScript，那么就不需要使用src属性——有时候这称为内部JavaScript。

> **注意**　一般而言，好的做法是将JavaScript代码保存在一个独立的文件中，并使用src属性将它引入到HTML页面中。这样做有利于保持页面代码的整洁，而且这意味着可以在多个页面中重用相同的JavaScript代码，而不需要重新编写代码。

在第二个script元素中，我们编写了第一段真正的JavaScript代码：

```
$(document).ready(function() {
        alert("Hello, World!");
});
```

很简单，不是吗？在此先忽略第一行和最后一行代码，因为下一节将更详细地介绍它们。现在，只需要知道它们是某个jQuery函数的一部分，这个函数能够保证JavaScript在HTML文档加载完成之后才会执行。

我现在关心的是第二行代码，即alert（警告）什么的奇怪代码。这句JavaScript代码是让浏览器有所反应的最简单的方式之一。alert函数（将在后面详细介绍）可用于在浏览器中打开一个对话框，它要求用户必须阅读完对话框内容，然后才能够继续浏览网页内容。它看起来确实有些不友好，但却是一个很好的例子。通过给alert函数传入一个字符串（一些文本），我们就能够修改对话框中显示的消息。当我们在页面上传入“Hello, World!”字符串时，就能够看到如图2-1所示的结果。

它起作用了！你现在可能看到对话框了——可以用对话框来做很多事情，如欢迎用户访问网站，或者当用户提交错误表单时发出警告。幸运的是，我们不再需要使用它们了，因为它们会打断用户的操作，并且通常会让人反感。而我们演示的目的是说明alert函数是JavaScript的一种快捷输出消息的方法。当然，这仅仅是出于测试的目的。

图2-1 “Hello, World！”示例输出

这样，你就有了一个基本的HTML页面，其中还有一些JavaScript代码。这并不复杂，不是吗？

> **注意** 如果你运行这个例子，但是没有出现任何效果，那么必须确保你能上网，因为这里使用了jQuery的外部Google版本。如果你希望以离线方式开发，那么要从官方网站下载jQuery，并使用下载的文件，具体见2.2节的介绍。如果仍然有问题，最好检查浏览器是否启用了JavaScript。

2.4 在页面加载之后运行 JavaScript

前面已经介绍了要在HTML加载完成之后再运行JavaScript。但是，为什么这样做很重要呢？为什么不能在任意时间执行JavaScript呢？答案有二。第一，如果让JavaScript在被添加到HTML页面中的地方运行，它会马上开始加载，并在完成加载之前暂停HTML其余元素的显示。通常，好的做法是先加载整个页面内容，然后再使用JavaScript进行后续处理。通过这种方式，网站用户可以开始查看网站内容，同时JavaScript在后台执行。第二，如果JavaScript在内容完成加载之前执行，那么它就无法访问未加载的内容。如果你尝试使用JavaScript处理HTML元素，如本书后文所述，那么这就是一个严重的问题。所以，我们需要用一种方法暂停JavaScript的运行，以等待HTML文档（我们的页面）完成加载。这里有三个选择：一个是错误的、一个是冗长的、一个简单的。下面我们将介绍这三种方法，因为理解它们的区别是很重要的。

2.4.1　错误的方法（window.onload事件）

最初，开发人员使用window.onload事件来解决上述问题。其工作方式是先完成网页全部内容的加载，然后再执行window.onload。修改alert消息框的例子，代码如下所示

```
window.onload = function() {
        alert("Hello, World!");
};
```

看起来很干净简单。然而，为什么又说它是错误的呢？问题就在于window.onload等待的时间太长了。只有在网页的所有内容加载完毕之后，它才会执行。但是，有问题吗，我们不就是想等所有内容全部加载完吗？不完全是。我们希望等到内容加载完成，但是我们并不希望待所有的内容都加载完才显示出来。例如，如果内容中有一个很大的图片，可能是几兆字节大小。（但是为了用户考虑，希望你在实际中不要用这么大的图片！）在加载网页时，这个图片需要相当长的时间才能加载完成，甚至可以看着图片一行行加载。如果在网页上使用window.onload，那么要等到所有图片都完成下载之后它才会执行，这个过程可能需要几分钟时间。

我们需要一种更好的方法，这种方法要在浏览器获得内容之后但在内容真正显示在屏幕之前就开始执行。幸运的是，我们可以利用DOM。

2.4.2　冗长的方法（DOM）

DOM是一种表示和访问文档元素（对于网页，那就是HTML元素）的方法。这种方法允许我们直接通过JavaScript获取HTML页面上的所有元素的信息，并处理所有元素及其属性。即使现在觉得DOM理解起来有些困难也不要担心，后面将对它进行详细的介绍。现在一定要知道的是DOM表示内容的原始结构，这意味着它必须在内容显示在屏幕上之前先创建好。如果能够确定DOM何时完成加载，就知道了内容何时可以访问，无论它是否在屏幕上可见。

可是，检查DOM何时加载完成是很难的。没有一种方法是在所有主流浏览器上都能用的（很意外吧）。即使有，实现这种方法肯定也不容易。好消息是有一些爱好者（如自由程序员Dean Edwards）已经帮我们完成了这部分艰难的工作，他们找出了一种在所有主流浏览器上都能工作的方法。这是他们辛勤工作的成果。但是你很快会明白为什么称之为“冗长的方法”：

```
// Dean Edwards/Matthias Miller/John Resig
function init() {
        //如果此函数已经被调用则退出
        if (arguments.callee.done) return;

        //标记此函数，同一件事不要做两次
        arguments.callee.done = true;

        //清除定时器
        if (_timer) clearInterval(_timer);

        //执行代码
};
```

```
/*针对Mozilla或Opera9
if (document.addEventListener) {
        document.addEventListener("DOMContentLoaded", init, false);
}

/*针对IE*/
/*@cc_on @*/
/*@if (@_win32)
        document.write("<script id=__ie_onload defer
src=javascript:void(0)><\/script>");
        var script = document.getElementById("__ie_onload");
        script.onreadystatechange = function() {
                if (this.readyState == "complete") {
                        init(); //调用加载处理器
                }
        };
/*@end @*/

/*针对Safari */
if (/WebKit/i.test(navigator.userAgent)) { //嗅探
        var _timer = setInterval(function() {
                if (/loaded|complete/.test(document.readyState)) {
                        init(); //调用加载处理器
                }
        }, 10);
}

/*针对其他浏览器*/
window.onload = init;
```

即使你是一位JavaScript老手，这也一定不是你希望在每一个项目中都动手实现的工作。肯定还有一种更简单的方法能够检测DOM的加载时间。这就是jQuery。

> **注意** 本书中不会逐行介绍Dean Edwards的脚本。如果要了解它的更多内容，以及创建它时Dean Edwardsr所遇到的问题，请访问他的博客: http://dean.edwards.name/weblog/2006/06/again/。

2.4.3 简单的方法（jQuery 方法）

还记得上一节所列举的这两行代码吗？现在回忆一下……或者，我也可以再给出一遍以节约咱们的时间：

```
$(document).ready(function() {
        //将想在页面加载完之后运行的Java Script代码放在这里
});
```

这是jQuery做得最棒的地方，它使我们能够以一种优雅且简单的方式完成复杂的任务。当分析它的核心组件时，我们可以看到jQuery真正的强大之处。

第一部分是$(document)，它是一个jQuery选择器。它允许你从所处理的DOM中选择一个元素，在我们的例子中是文档对象——包含了所有HTML元素的根节点。2.12节中将会更详细地介绍jQuery的选择器。

第二部分是.ready()，它是整个功能中的重要部分。它的唯一用途是让你知道DOM在什么时候完成加载。它只有一行代码，而且只有一个单词，但它的作用与前面的复杂方法中的数十行代码是一样的。然而，我要清楚地告诉你jQuery的.ready()方法并没有什么神奇的地方。它实际上基于Dean Edwards的方法，并且把所有代码封装到了一个极为易用的包中。换句话说，它的实际实现方法与我们前面看见的复杂方法的代码完全相同。

最后，我们需要添加在DOM加载之后立即执行的代码。.ready()方法通过在圆括号中使用一个回调函数实现这个功能；回调函数中包含一些在特定事件发生后执行的JavaScript代码。在我们的例子中，使用了一个空函数，可在这个函数中添加在DOM加载后立即执行的代码。这是最难的地方了！

现在，我希望你已经认识到与原始的纯JavaScript相比，使用jQuery的好处了。我们并没有自欺欺人，只是使用jQuery来处理重复且复杂的工作而已。

你现在应该已经了解了jQuery的工作方式。让我们继续前行，学习JavaScript编程的基础知识。

2.5　变量与数据类型

变量和数据类型是最基础的知识。它们是我们进行JavaScript开发的最基本元素。

2.5.1　变量

你在编程过程中一定使用过变量。变量的唯一作用就是保存一个值（一块数据），以备后面使用。这种方式可能与你在学校中学习代数时所见到的方式非常相似。还记得那些将数字赋值给一个字母的等式吗（例如，x=3）？这些字母就是变量，它们保存了一个值。

JavaScript的变量也一样。先创建一个变量，并给它命名（例如，x），然后再给它赋一个值，比如一个数字或某个文本（如，x=3）。在我们的例子中，引用x将会返回3。这在编程时是非常重要的，因为通过种方式，只需要赋值一次，就能够通过这个变量重复使用这个值，而不需要重复地输入这个值。甚至还可以在赋值后修改变量的值。正因为如此，它才被称为*变量*。你很快会理解这一点，所以不用太担心。

1. 给变量命名

在使用变量之前，首先需要给变量命名，变量的名字应该是能够解释其用途的简短词汇。例如，将一个保存用户姓名的变量命名为*myVariable*或x是不对的。你将来怎么可能记得这个变量的用途呢？给变量指定一个能够准确解释其用途的有意义的名称是比较好的方法。在我们的例子中，用*userName*甚至用*name*都是不错的名字。一定要选择一个你绝不会忘记其作用的名字。

注意 变量名的格式有很多，但我们的例子将使用一种称为驼峰拼写法的书写方式。当我们将多个单词拼接在一起来表示变量名称时，除了第一个单词的首字母之外，其余每个单词的首字母都大写。这是一种常用的变量命名方式，但是你也可以使用下划线或者其他字符来划分单词，甚至任何你想使用的方式。

在命名变量时应该注意一点，就是并非所有单词都能使用。一定要记住，变量名只能以字母（a~z）或下划线开头。变量名可以包含数字，但是数字不能作为它的第一个字符。同时，有一些单词是保留字，这意味着它们不能用作变量、函数或对象等的名称（稍后会详细讲述）。Mozilla开发者中心有一份完整清单，其中包含了所有不允许使用的单词：https://developer.mozilla.org/en/JavaScript/Reference/Reserved-Words。

2. 声明变量与赋值

我们现在理解了什么是变量，以及为什么需要使用变量。我们还知道了如何给变量正确命名，使它们的名称容易理解。下一步，我们将学习如何真正在JavaScript中使用变量，或者按照开发者的说法，如何声明变量和给变量赋值。

声明变量是创建变量的另一种说法。使用变量之前先声明变量是一种好的做法，因为这样我们才能够完全控制变量且知道它的存在。下面是声明变量的方法：

```
var userName;
```

这里需要注意两点。第一，在JavaScript中，变量是使用关键字var声明的。这表示var之后的单词为变量名称，而JavaScript会用这个名称声明一个变量。第二，每条语句以分号结束。在一些编程语言中，在每一条语句（代码块）之后添加分号是极其重要的。在JavaScript中有些不同，语句末尾不添加分号也是允许的，但是添加分号会更好一些，因为这样做代码会更加整洁，并且有可降低将来调试代码的难度。

在声明一个变量之后，通常最好给它赋一个值，否则它会被自动赋值为undefined，后面将对这个值进行介绍。给变量赋值非常简单，只需要使用等号，也叫赋值操作符，如下所示：

```
userName = "Rob Hawkes";
```

请注意，不需要使用var关键字，因为变量已经被声明过了。如果我们还没有声明*userName*变量，这个语句仍然能够产生预期效果，因为JavaScript语法是很宽松的。虽然我们可以不使用var关键字而声明一个变量（通过直接赋值），但是这不是一种好的做法，因为在JavaScript中它可能导致出现各种作用域问题。简而言之，在第一次使用变量时，应该先正确地使用var关键字声明这个变量。此外，你应该注意到，*Rob Hawkes*位于双引号之中，这是因为文本值（字符串）需要特殊处理。我们将在2.5.2节中介绍字符串及其他类型的值。

注意 在所有函数之外声明的变量是全局变量，它们可以被所有代码访问。而在函数之内声明的变量是局部变量，在创建它们的函数外部是访问不到的。作用域的问题很常见，所以如果遇到问题，一定要检查变量声明的位置是否正确。

实际上可以同时声明变量和给变量赋值，方法如下：

```
var userName = "Rob Hawkes";
var age = 34;
```

这是一种非常快捷的方法，而且更简洁。

此外，还可以给一个变量重新赋值，不用带var关键字。在前一个例子中，我的年龄（age）是错误的，可以这样将它修改为正确的值：

```
age = 24;
```

这就对了。

3. 访问变量

如果给一个变量赋值后却无法再取回它的值，那么将是毫无意义的。但是，访问变量的值是非常简单的——只需要使用所指定的变量名称即可。例如，如果希望在一个警告框中显示*userName*变量值，可以使用以下方法：

```
var userName = "Rob Hawkes";
alert(userName);
```

如果一切正常，就会看到一条警告信息，其中显示用户的姓名，如图2-2所示。

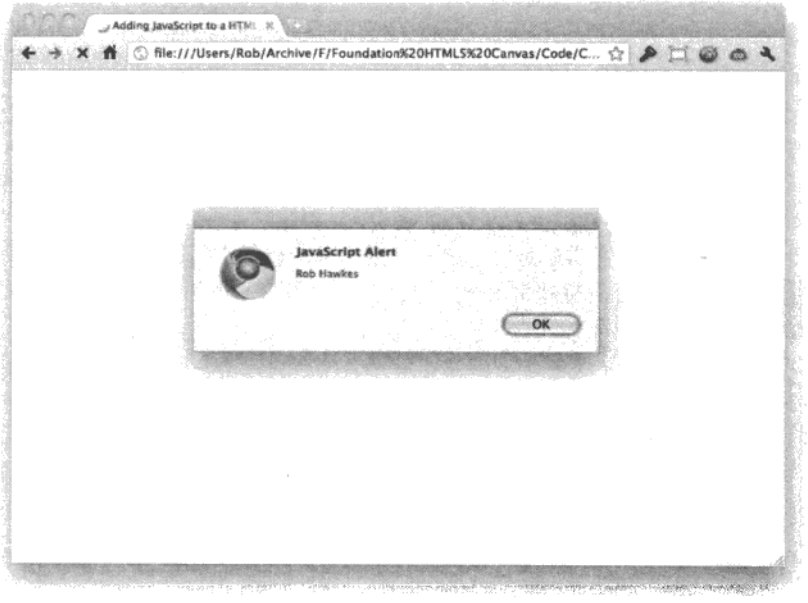


图2-2　使用变量输出用户的姓名

如果愿意，甚至可以同时显示用户的年龄：

```
var userName = "Rob Hawkes";
var age = 24;
alert(userName+" is "+age+" years old");
```

这是通过字符串拼接操作实现的，它能够输出如图2-3所示的消息。

图2-3 使用变量输出用户的姓名和年龄

这就是变量的强大之处。它们允许你随时访问代码中的值，并且允许组合使用，这样你就能够按照自己希望的方式显示这些值了。

4. 算术运算

变量的另一个用途是进行算术运算。JavaScript几乎能提供你所需要的各种算术功能。例如，下面的代码可以实现两个数的求和运算：

```
var myNum = 24+6;
alert(myNum); // 输出30
```

但是，这里的双斜杠有什么作用呢？此处是JavaScript的注释，注释是在代码中添加的一些说明，它们不会运行也不会显示。从现在开始，我将使用注释来说明例子的预期输出结果。一定要注意，同一行中双斜杠后面的内容为注释，它们将被JavaScript忽略。如果使用不当，注释有时候会导致一些错误，所以这是需要特别注意的。

下面的代码是求两个数的差(注意，可以直接在alert中输出运算结果，而不需要使用变量)：

```
alert(24-4); // 输出20
```

非常简单。如果要相乘，则为：

```
alert(2*5); // 输出10
```

如果要相除，则为：

```
alert(100/2); // 输出50
```

以此类推。这些都是常用运算，并且都使用与计算器或其他编程语言相同的运算符号。

为了增加算术式的可读性，可以考虑使用变量来保存数值。例如，使用下面的代码输出一天的秒数：

```
alert(24*60*60); // 输出86400
```

它的运算结果是正确的，但是可读性比较差。现在使用变量来实现相同的运算：

```
var hoursInDay = 24;
var minutesInHour = 60;
var secondsInMinute = 60;
alert (hoursInDay*minutesInHour*secondsInMinute); // 输出86400
```

实际上，这段代码的可读性更强一些。任何人看到这段代码都能够理解它计算的结果。这正是变量的价值所在，也是应该合理命名变量的原因所在。

2.5.2　数据类型

数据类型是编程语言中必不可少的，几乎所有JavaScript代码中都要使用数据类型。数据类型定义了你所使用的数据种类，并限制对这些数据的操作。理解每一种数据类型有利于避免将来出现不必要的错误和问题。例如，String和Number这两种数据类型的相加结果是不同的：

```
var myString = "24"+"6";
var myNum = 24+6;
alert(myString); // 输出"246"
alert(myNum); // 输出30
```

当两个字符串相加时，它们是连接在一起的，即所谓的拼接操作。即使字符串的内容是数字，其结果也是相同的，因为JavaScript将引号内的任意内容都视为字符串，而非数字。

1. 基本数据类型

基本数据类型是数据的最基本形态，之所以称为基本数据类型，是因为它们不能进一步细分为其他数据类型。基本数据类型有以下几种。

- **数值**：整数和浮点数（例如，2.04）。
- **布尔值**：真值，可为true或false。
- **字符串**：位于引号中的任意字符（例如，"Rob Hawkes"）。

2. 复合数据类型

这些数据类型更高级一些，它们可能包含一些基本数据类型或复合数据类型的值。例如，一个包含更多数组或对象的数组就是一种递归复合数据类型。

- **数组**：通常用于保存一组相关联的值。

❑ **对象**：JavaScript中所有对象的基础。

这些数据类型可能有些容易混淆。本章后面将会更详细地介绍数组和对象。

3. 其他数据类型

除了标准数据类型，还有两种表示不存在或空值的数据类型。

❑ **null**：表示什么也没有，或者没有值。

❑ **undefined**：变量在声明之后和赋值之前保存的值。

通常，在变量声明出错，或者变量未被正确赋值时可以看到它们；但是它们也是可以合法使用的。比如通过给一个变量赋值null来清除它的原始值。另外，它们可以用在条件语句中，检查一个变量是否声明过或者被赋过值。

2.6 条件语句

你是否想喝杯茶：是或否？这里我给你一个选择。我问你一个问题，然后你可以选择说："是，我想喝茶。"或者说："不，你想干什么？我不喜欢喝茶！"如果回答是，那么你会得到一杯茶。如果回答不，你就没有茶。选择就是这么简单。JavaScript的条件语句处理选择的方式差不多就是这样——它们实际上只是用代码描述的问题。

条件语句在JavaScript中非常有用，因为它们允许你处理一些有不同结果的情况。它们允许你做出决定，如判断一个变量是什么，或者事情是正确的（true）还是错误的（false）。

2.6.1 if 语句

最简单的条件语句是if语句。它允许你做出true或false的决定：如果是true，那么if语句中的代码就会运行，如果是false，就不会执行。

下面是一个非常简单的例子：

```
var age = 24;
if (age == 24) {
        alert("You're 24 years old"); // 只有在age为24时才会输出
};
```

我们先创建一个变量*age*，然后创建一个if语句，基于该变量进行true或false判断。在我们的例子中，我们检查*age*是否等于24，如果相等，就会显示一条警告信息。如果将变量*age*修改为30，警告信息就不会显示，因为*age*（现在为30）不等于24。我们的代码具有了一定的智能。

注意我们使用了两个等号，它是比较操作符。两个等号与一个等号（赋值操作符）是完全不同的，后者是用来给一个变量赋值的。如果混淆这两个符号的使用，就会出现问题。例如，在前一个例子中，如果if语句使用一个等号，那么语句就一定会执行：

```
var age = 24;
if (age = 24) {
        alert("You're 24 years old"); // 总是会输出
};
```

甚至可以将*age*修改为一个你知道肯定是错的值，但是这个语句仍然会执行，因为它的条件总为true，而且*age*变量在if语句中执行了赋值操作。稍后将对此进行更详细的解释。

if语句的另一个常见用法是使用Boolean数据类型明确地检查一个变量是否为真，其中Boolean值只能是true或false。下面是布尔值的使用示例：

```
var wouldLikeATea = true;
if (wouldLikeATea == true) {
        alert("Milk, 1 sugar. Thank you!"); // 在用户想喝茶时才输出
};
```

在这个例子中，我们用一个变量来表示用户是否想要喝一杯茶，true表示是，false表示否，都是用布尔值。然后，我们创建一个if语句，使用布尔值来检查用户是否想要喝一杯茶，如果是，那么输出一条警告信息。实际上，可以将这个例子简写为：

```
var wouldLikeATea = true;
if (wouldLikeATea) {
        alert("Milk, 1 sugar. Thank you!"); // 在用户想喝茶时才输出
};
```

注意我们删除了if语句中的双等号和true。之所以可以这样做是因为if语句默认是检查真值的。如果所检查的变量为true，那么语句内的代码就会执行。一定要注意，如果你使用这种方法来检查一个非布尔型变量，如字符串或数字，那么除0以外的值都会返回true。这是因为非布尔值只要包含数据就会返回true。例如，在下面的代码中，如果数字不为0则返回true，如果为0则返回false：

```
var myInteger = 7;
if (myInteger) {
        alert("This code executes.");
};

myInteger = 0;
if (myInteger) {
        alert("This code does not execute.")
};
```

2.6.2　比较运算符

比较运算不仅限于检查两个值是否相等，还可以执行很多其他的比较。在JavaScript中，通过修改条件语句所使用的比较运算符，可以实现各种类型的比较。表2-1显示了一组最常用的运算符。

表2-1　JavaScript中的比较运算符

运 算 符	名　称	说　明
a == b	相等	a与b相等时为true
a === b	相同	a与b相等且类型相同时为true
a != b	不相等	a不等于b时为true
a !== b	不相同	a不等于b或a与b类型不同时为true

（续）

运 算 符	名 称	说 明
a < b	小于	a小于b时为true
a > b	大于	a大于b时为true
a <= b	小于等于	a小于或等于b时为true
a >= b	大于等于	a大于或等于b时为true

> **注意** 条件语句中的两个等号（==），即比较运算符，用于检查两个值是否相等。赋值运算符（一个等号，=）只用来给变量赋值。在代码中将它们混淆使用是很常见的错误，所以一定要注意这个问题，否则会出现意想不到的错误。

2.6.3 在 if 语句中进行多重布尔值检查

除了比较运算符，还有逻辑运算符。这些运算符允许你在一个条件语句中进行多重检查。如果我们想要判断两个检查是否为true，可以使用与运算符：

```
var age = "24";
var userName = "Rob Hawkes";
if (age == 24 && userName == "Rob Hawkes") {
        alert("You're definitely Rob Hawkes"); // 如果两个检查均为true则输出
};
```

现在if语句只有在两个检查都返回true时才会执行，即*age*变量为24且*userName*变量为"Rob Hawkes"。如果这两个检查有一个返回false，if语句就不会执行。如果想要有一个检查为真时就执行if语句，需要使用或运算符（||）：

```
var age = "30";
var userName = "Rob Hawkes";
if (age == 24 || userName == "Rob Hawkes") {
        alert("You're either 24 or Rob Hawkes"); //如果任一检查为true则输出
};
```

使用这些逻辑运算符，你就能够组合检查和创建功能强大的条件语句。

2.6.4 else 和 else if 语句

如果你只考虑结果为true的情况，if语句就够用了，但如果你希望处理结果为false的情况，又该如何做呢？这时就需要使用到else和else if语句。

如果你还希望在if语句为false时执行一些操作，可以使用else语句：

```
var age = 30;
if (age == 24) {
        alert("You're 24 years old"); // 当age等于24时输出
} else {
        alert("You're not 24 years old"); // 当age不等于24时输出
};
```

当原始if语句为false时，还可以进一步执行其他检查。例如，可以使用else if语句：

```
var age = 30;
if (age == 24) {
        alert("You're 24 years old"); // 当age等于24时输出
} else if (age == 30) {
        alert("You're 30 years old"); // 当age等于30时输出
} else {
        alert("You're not 24 or 30 years old"); // 当age不等于24和30时输出
};
```

else if语句是另一种在问题之后再提问题的方法：是这个吗？不是。那么这个呢？不是。那么那个呢？等等。

2.7　函数

代码重复是编程时不可避免的问题。有时，是出于代码质量的考虑；有时，是为了简化程序员的工作。无论是什么原因，如果处理不当，代码重复就会造成严重的问题。可能会浪费编码时间，也可能会致使你前功尽弃，因为你可能会重复编写相同的代码50次。

现在，许多人的疑问可能是不理解为什么要重复代码。原因是什么呢？最简单的原因是有一些代码是会重复使用的，因为你可能会多次执行一个相似的操作。例如，一段输出由姓和名组成的完整姓名的代码：

```
var firstName = "Rob";
var lastName = "Hawkes";
var fullName = firstName+" "+lastName;
alert(fullName);

var anotherFirstName = "John";
var anotherLastName = "Smith";
var anotherFullName = anotherFirstName+" "+anotherLastName;
alert(anotherFullName);
```

> **注意**　这两种情况组合姓和名的方法实际上是相同的。唯一的区别是它们使用了不同的变量。不要认为只有这两个地方不同，而要想象一下，在代码中可能有几十处需要组合姓和名的地方。假设你需要修改姓名格式，如调换姓和名的位置，那么需要修改的地方会很多。而如果漏掉一处呢？问题的严重性不可想象！如果我们用一段特殊代码来处理姓名的格式，然后只修改这段代码并自动在所有地方更新，这样是不是很不错呢？这就是函数的功能。

2.7.1　创建函数

使用函数是非常简单的。只需要理解它们的使用方法。函数就像一个反复以相同方式重复执

行同一个操作的小型机器。可以给函数传递信息，也可以从函数获取信息。解释这个过程的最好方法是展示一个函数是如何工作的。

如果将前面使用的姓名格式重写为一个函数，那么这个函数是：

```
function formatName(firstName, lastName) {
        return firstName+" "+lastName;
};
```

关键字function是用来声明函数的，它使JavaScript知道后面的单词是函数名称，在这个例子中函数名称是formatName。括号里面的单词是参数，参数是用来声明传入函数中的值的变量。即使函数没有参数，也需要使用这对括号，只是没有参数时括号里面没有任何内容。函数中所有运行的代码都位于花括号内。

函数经常需要输出或返回一些值。这时，需要使用return关键字，它会返回位于它后面的任何值。关键字return也会退出当前函数，所以return之后的代码是不会执行的。

2.7.2 调用函数

函数在不被使用的时候是没有任何意义的，用开发人员的话说就是要调用它。通过调用函数，可以运行其中包含的代码，可以访问函数返回的值。在调用函数时，也可以使用一些输入值（实参）。下面，我们可以充分运用这些知识来调用新的formatName函数：

```
formatName(firstName, lastName);
```

或者，与其余代码结合在一起为：

```
var fullName = formatName("Rob", "Hawkes");
alert(fullName);

var anotherFullName = formatName("John", "Smith");
alert(anotherFullName);
```

调用函数与访问变量有些相似，需要使用创建函数时使用的名称。区别是，调用函数需要在函数名之后加上一对括号。在括号内，还需要加入函数的输入值；在我们的例子中，加入的是表示姓和名的变量。再次查看这个函数，就会理解它的作用了：

```
function formatName(firstName, lastName) {
        return firstName+" "+lastName;
};
```

在第一个例子中，我们调用formatName函数，给它传入名（Rob）和姓（Hawkes）。函数会将这些字符串作为输入实参，即*firstName*和*lastName*，它们都是一般的变量。我们的函数实际上很简单，仅仅返回格式化后的姓名，而不进行其他的处理。然后，返回值会存储在*fullName*变量中，就像我们从未使用函数一样。在第二个例子中，我们采用相同的方式调用函数，只是将输入变量替换为不同的姓和名。

使用这种方法有很多好处。首先，由于不需要一次次重复编写相同的代码，能够节省很多时间，我想你一定会感到非常满意。其次，如果我们需要修改姓名的格式化方式，此时只需要修改

一处代码，而不需要修改多处代码——这是非常非常好的优点。例如，可以轻松地将函数输出修改为颠倒位置后的姓名：

```
function formatName(firstName, lastName) {
        return lastName+" "+firstName;
};
```

2.8　对象

有些人认为对象属于基础类图书的高级主题，但是我并不认同。从长远看，对象可以简化我们的开发，那么既然从一开始就能够学习到正确的方法，为什么我要教你错误的习惯呢？我们将在游戏开发中大量使用对象。而且，它们实际上也不像人们想象得那么复杂。你很快就会理解它们的用法。

2.8.1　什么是对象

在开始学习如何创建和使用对象之前，先了解它们是什么，以及它们为什么好用很重要。技术手册中一般将对象描述为一组处理相关值和任务的属性（变量）和方法（函数）。对象也是一种数据类型，这意味着我们可以将它们赋值给变量。关于这个用法，将在后面内容中再介绍。

在JavaScript中理解对象的最佳方法是将它们想象为现实世界的对象，如汽车或火箭——我更喜欢火箭。JavaScript的对象拥有属性和方法，这和火箭一样。如果我们进一步分析，火箭的属性包括它拥有的引擎个数、推力、宇航员人数等，甚至也包括简单的颜色等。属性实际上是描述对象特征的值。另一方面，火箭的方法可能是诸如开启引擎和关闭引擎、打开宇航员逃生门或者发送消息到地面控制中心等。方法是指在对象上或由对象执行的某种操作。

那么，对象到底是什么呢？并且，为什么它们如此适合用来解释事物？虽然我不能将对象概括成几个字，但是我肯定可以用一句话来概括：JavaScript对象是模板，它们描述一些事物的特性，定义了事物能够执行的操作。

从我个人的经验来看，我认为真正理解对象的最佳方法是实际编写一个对象；现在就开始动手吧。

2.8.2　创建和使用对象

在JavaScript中创建对象的方法有很多，其中最简单的方法是：

```
var rocket = new Object();
```

这条语句的作用是创建空白对象模板Object的一个新实例（版本），并将它赋值给一个变量。从技术上讲，我们这里使用的是Object对象，但是我们将它称为普通对象（plain old object），以避免混淆。希望你不会混淆。

现在，除了JavaScript的一些内置方法，这个火箭对象不包含任何属性。这显然不够，我们需要有引擎、推力和宇航员。声明一个对象的描述性元素（属性）与创建变量一样简单：

```
var rocket = new Object();
rocket.engineCount = 2;
rocket.thrust = 5000;
rocket.astronautCount = 4;
rocket.colour = "red";
```

这里的模式是非常简单的，引用对象变量，使用点（.）运算符，然后输入我们喜欢的属性名称（与变量名一样），给它赋值。对象并不难懂，它们只是有些与众不同而已。

但是，这种方法有一个大问题，那就是我们的对象是独一无二的。我们从一个空对象模板创建它，手动声明它的所有属性，这些都是一次性的。如果需要创建多个火箭，那么必须一次次地复制相同的代码，而复制代码是一种糟糕的方法。这也意味着我们无法保证每一个火箭对象都拥有相同的属性和方法，例如：

```
var rocket = new Object();
rocket.engineCount = 2;

var anotherRocket = new Object();
anotherRocket.engineCount = 1;
anotherRocket.wings = 2;
```

这两个火箭对象表面上是相同的，但是第二个火箭有一个新属性wings。因为第一个火箭没有这个属性，所以我们就遇到一个致命问题。例如，如果输出第二个火箭的机翼数，那么就会得到“2”，但是如果在第一个火箭上执行相同的操作，就得到“undefined”。第一个火箭没有机翼（wings），因为没有给它们创建过。

那么，应该如何解决这种唯一且不可预见的对象问题呢？当然，方法就是用我们自己的火箭对象模板来创建对象，模板内预先声明了火箭的所有属性和方法。下面这个例子会更方便我们理解：

```
function Rocket(engineCount, thrust, wings) {
        this.engineCount = engineCount;
        this.thrust = thrust;
        this.wings = wings;
};
```

你可能会认为它是一个函数，没错——函数也是对象。可以按照通常采用的方式创建一个函数，然后将它作为一个对象使用。值得注意的是，一般函数和对象模板函数的主要区别在于声明它们变量的方式。在这里，我们声明的是属性，而不是变量，所以我们使用关键字this，而不使用var。关键字this表示将这个属性赋值给对象的当前（this）实例，这样不同的实例就拥有不同的属性值。这有些难以理解，但是不用担心，后文将详细阐述。

使用新的火箭对象模板非常简单：

```
var rocket = new Rocket(2, 5000, 4);
var anotherRocket = new Rocket(1, 2000, 2);
```

现在不仅代码更加整洁，而且完全不存在代码重复问题。另一个好处是我们能够确信所有火箭对象都拥有相同的属性：

```
alert(rocket.wings); //输出4
alert(anotherRocket.wings); //输出2
```

太棒了！但是这些属性都是静态的，应该如何使火箭对象执行一些实际操作呢？为此，我们需要增加一些方法，这实际上也是非常简单的。下面，我们添加一个方法来开启火箭引擎：

```
function Rocket(engineCount, thrust, wings) {
        this.engineCount = engineCount;
        this.enginesOn = false;
        this.thrust = thrust;
        this.wings = wings;

        this.turnEnginesOn = function() {
                this.enginesOn = true;
                alert("Engines are now on");
        };
};
```

对象方法实际上是一个赋值给对象属性的函数。要记住，函数也是对象，所以它们也是一种数据类型，这意味着它们也可以赋值给变量（和属性）。方法的名称就是方法所赋值的属性名称，之后调用方法时也会用到这个名称。

为实现我们的目的，我们声明了一个新的火箭属性，即*enginesOn*，它表示引擎的状态是开启（true）还是关闭（false）。在turnEnginesOn方法中使用这个属性，就能够开启火箭引擎，同时，还另外弹出一个提示框来表示动作执行（注意，这里的提示框仅仅用于测试）。

现在我们创建了火箭的turnEnginesOn方法，这个方法的实际使用很简单：

```
var rocket = new Rocket(2, 5000, 4);
rocket.turnEnginesOn();
```

对象方法也是函数，所以它们的调用与一般函数相同。理解这一点也就理解了方法的使用。你可能还需要一定的时间才会完全理解，但它们真的不难。

如果你还不能完全明白对象的使用，我完全能够理解，因为以前我也花了很长时间才完全掌握对象的使用。我希望经过这样的简单介绍，你至少能够理解游戏开发中对象使用的基本方法。请相信，对象在一些编程项目中是非常有用的。

2.9　数组

有时候，你难免需要在一个变量中存储多个值。我指的是，这些变量可能限制太多了，不是吗？幸好，JavaScript有一个完美特性，它允许你进行这种操作。这个特性就是Array对象。

数组（使用Array对象创建）实际上是一种容器，可以存储多个值。而重要的一点是它们是一种数据类型，这表示它们能够赋值给变量，方法与处理数字和字符串变量一样。所以，一个保存了Array对象的变量实际上可以包含无数个值（或元素）——这是非常好的特性。

数组的作用是在不需要创建大量变量的情况下就能够方便地存储许多相关数据，例如，太阳系的一系列行星。数组就像一个列表，使一些相关值能够保存到一个变量中。

2.9.1　创建数组

创建Array对象的方法有很多，其中最复杂的一种方法是：

```
var planets = new Array();
planets[0] = "Mercury";
```

第一行代码创建了一个空的Array对象，然后将它赋值给变量*planets*。第二行代码将一个字符串类型的星球名称赋值给数组的第一个元素。一定要注意，Array对象的第一个元素的索引值是0，而不是1。

要给这个数组添加其他一些元素，可以采用下面的方法：

```
planets[1] = "Venus";
planets[2] = "Earth";
planets[3] = "Mars";
```

也可以在创建Array对象时同时赋值，如：

```
var planets = new Array("Mercury", "Venus", "Earth", "Mars");
```

另外，如果你觉得还不够好，可以使用一种称为“字面量表示法”的方法——一种使用方括号的快捷方式：

```
var planets = ["Mercury", "Venus", "Earth", "Mars"];
```

每一种方法生成的结果都是相同的，你可以选择一种自己喜欢的方法来创建Array对象。

2.9.2 访问和修改数组

创建一个Array对象并将所有值保存在一个整洁的清单中是一种很好的做法，但是你应该如何取回这些值呢？很简单。事实上，我们已经看到了取回值的代码了。

下面是访问行星数组的第二个元素的方法：

```
var planets = ["Mercury", "Venus", "Earth", "Mars"];
alert(planets[1]); // 输出Venus
```

再提醒一次，数组元素的索引是从0开始的，所以第二个值的索引是1，不是2。那么如何访问数组的第4个元素呢？很简单：

```
alert(planets[3]); // 输出Mars
```

现在了解这些操作就已经足够了。数组还有许多其他操作，但是我们将留到需要使用时再介绍。

2.10 循环

我们已经知道函数是自动封装大段代码的好方法。然而，函数也有一个缺点，那就是使用一次就必须调用它们一次。例如，如果想要运行函数5次，那就需要调用5次。如果要运行100次，就需要调用100次。这样，编写100行代码来调用同一个函数是没什么意义的。这对JavaScript来说貌似有些奇怪，我们使用如此强大的函数和数组却做这样低效的事情。如果有一件事必须重复做100次，是不是有什么更简单的方法呢？ JavaScript并没有让我们失望，它拥有一组完美的特性来解决这个问题——循环。

举一个非常简单的例子，假设我们在数组中保存一组姓名，然后再执行formatName函数5次，一次处理一个姓名。如果不使用循环，需要编写下面的代码：

```
var names = [["Rob", "Hawkes"], ["John", "Smith"], ["Jane", "Doe"], ["Queen", "Elizabeth"],
["Steve", "Jobs"]];

var first = formatName(names[0][0], names[0][1]); // "Rob Hawkes"
var second = formatName(names[1][0], names[1][1]); // "John Smith"
var third = formatName(names[2][0], names[2][1]); // "Jane Doe"
var forth = formatName(names[3][0], names[3][1]); // "Queen Elizabeth"
var fifth = formatName(names[4][0], names[4][1]); // "Steve Jobs"
```

你可能想知道这是个什么数组。不用害怕，这只是一个多维数组，实际上就是一个数组位于另一个数组之中。主数组包含另外5个数组，而这5个数组分别包含一个人的姓和名。请先不要担心多维数组的概念，我们关注的是循环，先按照这个思路走。

虽然仅仅5次迭代并不是什么糟糕的事情，但是这种方法肯定不好或效率不高。所以，我们再用循环来试一次：

```
var names = [["Rob", "Hawkes"], ["John", "Smith"], ["Jane", "Doe"], ["Queen", "Elizabeth"],
["Steve", "Jobs"]];

for (var i = 0; i < names.length; i++) {
        var fullName = formatName(names[i][0], names[i][1]);
};
```

毫无疑问，这种方法不仅更加整洁，而且还更高效。这里使用的是for循环，它通常将一段代码重复执行指定的次数。括号内有三个重要的部分，每一部分都是由分号分隔的。下面让我们逐一分析每个部分：

```
var i = 0;
```

在第一部分中，我们声明了变量i，将其作为循环计数器，同时给它赋值0（记住：数组的第一个元素是0）。

```
i < names.length;
```

第二部分是在每次循环之前对计数器执行一次检查。在这个例子中，我们要检查计数器是否小于names数组的元素个数，即它的长度（length），这是每一个Array对象都拥有的属性。如果这个检查返回true，则将转到第三部分（最后一部分），否则将退出循环。

```
i++
```

最后一部分是在每次循环结束后运行的，在我们的例子中，计数器每次循环都使用自增运算符增加1；i++等价于使用i=i+1。

最终的结果是将计数器从0循环到4，这样就实现了用一行代码访问names数组的5个元素：

```
var fullName = formatName(names[i][0], names[i][1]);
```

注意，变量i的位置，它替换了前面没有循环的例子中的数字0～4。这样就简单很多了！

> 注意 循环还有其他形式，如while循环，但是在本书中，我们不会使用这些循环。它们和for循环一样简单，你可以自己花一些时间来学习。

2.11 定时器

顾名思义，定时器允许你在一定时间之后运行某段代码（有点像电饭锅定时器），或者在一定时间间隔内重复运行（如汽车转向灯）。这两种定时器都是非常有用的，而且实际上是本书后面介绍的游戏的主要组成部分。

2.11.1 设置一次性定时器

setTimeout方法允许我们在以毫秒为单位的特定延迟时间之后运行某段代码。它很容易设置，并且它支持的语法都是我们已经知道的：

```
function onTimeout() {
        alert("Ding dong!");
};
var timer = setTimeout(onTimeout, 3000);
```

setTimeout方法有两个参数：当定时器运行时你希望调用的代码或函数，以及运行定时器之前以毫秒为单位的延迟时间。在我们的例子中，我们在定时器中调用了一个函数，但是注意我们省略了括号。这是因为，如果保留括号，就会在定时器创建时立刻调用这个函数——这不是我们希望得到的结果。这个方式理解起来有些复杂，但是先不要担心。第二个参数是以毫秒为单位的延迟时间，在我们的例子中它的值为3000，即3秒钟。记住，1秒钟相当于1000毫秒，千万不要忘记进行时间单位换算。

如果一切正常，3秒钟之后你会看到一条警告信息：“Ding dong!”。

2.11.2 取消一次性定时器

有时候，你会希望停止一个运行的setTimeout方法，这通常是因为在设置定时器后又发生了一些要求停止定时器的事件。如果要取消由setTimeout创建的定时器，就需要使用clearTimeout方法。将保存这个定时器的变量作为参数传递给clearTimeout方法，就能够停止这个定时器：

```
clearTimeout(timer);
```

显然，如果取消定时器的时间晚了，定时器会正常运行。

2.11.3 设置重复定时器

很多时候一次性定时器并不能满足使用，你可能希望定时重复执行某个操作。这时，需要使用setInterval方法，设置这个方法与setTimeout相同：

```
function onInterval() {
        alert("Ding dong!");
};

var interval = setInterval(onInterval, 3000);
```

它与setTimeout方法的唯一区别是传递给setInterval方法的函数每隔3000毫秒运行一次，并且重复运行。所以一定要注意，在浏览器运行这个脚本时，每3秒钟会弹出一个提示框。我希望在实际使用中你最好使用其他提醒方式，而不要让警告提示框来打扰用户。

2.11.4　取消重复定时器

与clearTimeout相似，用类似的方法可以取消重复定时器，那就是clearInterval方法。它的使用方法与前者完全相同，只是传递的参数是保存setInterval方法返回值的变量：

```
clearInterval(interval);
```

为了避免无限循环（即从不停止），你可能需要在定时器完成指定任务之后调用clearInterval方法。毕竟，任由它一直运行会浪费资源，也可能惹人讨厌。

> **注意** 一定要注意，setTimeout和setInterval都是DOM window对象的方法。它们其实不是JavaScript的方法，但是可以通过JavaScript访问和操作这些方法。随着对JavaScript和DOM内容的进一步了解，你对此会更有体会。

2.12　DOM

在本章前面的内容中，我们已经接触到了DOM，它的作用是表示网页和HTML元素的原始内容结构。换言之，如果你想要通过JavaScript访问或操作内容，那么就会用到DOM。本书不会详细介绍DOM的方方面面，但是会介绍与游戏开发相关的内容。可以从网上或通过阅读其他图书来详细了解DOM，有一些图书是专门介绍DOM使用方法的。

2.12.1　HTML网页示例

在本节中，我们将接触许多HTML元素，所以先创建一个精简版的HTML5博客页面，相关的内容已经在第1章中做了介绍。这并不是一个特别有用的网页，但是它可以作为后面内容讲解的示例。

```
<!DOCTYPE html>

<html>
        <head>
                <title>The DOM</title>
                <meta charset="utf-8">
```

```
            <script type="text/javascript" src="http://ajax.googleapis.com➥
/ajax/libs/jquery/1/jquery.min.js"></script>
            <script type="text/javascript">
                $(document).ready(function() {
                    // JavaScript代放在这里
                });
            </script>
        </head>

        <body>
            <section id="blogArticles">
                <article>
                    <header>
                        <hgroup>
                            <h1><a href="/blog/first-postlink/">
Main heading of the first blog post</a></h1>
                            <h2>Sub-heading of the first blog
post</h2>
                        </hgroup>
                        <p>Posted on the <time pubdate datetime=➥
"2010-10-30T13:08">30 October 2010 at 1:08 PM</time></p>
                    </header>

                    <p>Summary of the first blog post.</p>
                </article>
            </section>
        </body>
</html>
```

2.12.2 使用纯 JavaScript 访问 DOM

在介绍更简单的jQuery方法之前，一定要先了解如何使用纯JavaScript访问DOM。我们使用document对象来实现这些操作，它是包含了所有HTML元素的网页的根元素。除了元素，document对象也提供了一些按照元素属性或标签名称访问特定元素的方法。例如，下面的语句可以访问id为*blogArticles*的第一个HTML元素：

```
document.getElementById("blogArticles");
```

我们所使用的方法是非常简单明了的，其他方法其实也是一样。显然，这个方法将通过其id属性获得一个HTML元素。你现在应该明白给方法命名为什么如此重要了吧：它可以直接明了地说明其作用。

另一个document对象方法允许你访问具有特定标签名称的所有元素。在我们的例子中，可以使用下面这个方法来访问p元素：

```
document.getElementsByTagName("p");
```

getElementsByTagName方法返回一个数组，其中包含所有匹配的HTML元素。在我们的例子中，它返回一个数组，其中包含header元素中的一个p元素和article元素中的另一个p元素。

现在，仅仅调用getElementById和getElementsByTagName的作用并不大。真正有用的是对这两个方法返回的HTML元素执行进一步的处理。当你使用DOM访问某个HTML元素时，你能够使用该元素的全部属性。有一些属性允许你访问元素的属性值，而其中有一个特殊属性允许

你访问元素所包含的内容。这个属性非常有趣，它就是innerHTML，使用这个属性可以访问HTML元素的内容：

```
var secondaryHeadings = document.getElementsByTagName("h2");
alert(secondaryHeadings[0].innerHTML);
```

这里所做的第一件事是搜索所有HTML h2元素，然后将得到的数组赋值给变量*secondaryHeadings*。由于数组元素的位置是从0开始的，所以我们访问页面中第一个（在我们例子中也是唯一一个）h2元素的innerHTML属性。最后，用一个提示框输出h2元素中的内容。这是我开始学习DOM时最喜欢的功能，但这可能只是我自己的体验。

2.12.3　使用 jQuery 访问 DOM

虽然使用纯JavaScript访问DOM有些复杂，但是难度并不大。而且，jQuery允许我们以一种简单但又极强大的方式访问DOM。例如，jQuery与getElementById方法等价的方法是：

```
$("#blogArticles");
```

看，非常简洁！仔细计算，它整整少了21个字符——很短。值得注意的是id名之前的井号(#)，这是因为jQuery使用与CSS相同的前缀来匹配元素——井号表示id名，点表示类名，要匹配标签名称什么也不用带。另外，要注意代码开头的美元符号（$），这是访问jQuery的快捷方式，它等价于调用jQuery("#blogArticles")。

所以，现在你可能觉得用jQuery重写getElementsByTagName示例是小事一桩了，但我还是把它写出来，非常简单：

```
$("p");
```

这就是全部代码。如此简短的代码能够实现如此多的功能，实在是令人难以置信，但是这7个字符确实能够做很多事情。

而对于innerHTML属性，用jQuery访问也很简单吗？答案是肯定的！

```
var secondaryHeadings = $("h2");
alert(secondaryHeadings.html());
```

只需要使用jQuery的html方法，它的作用与innerHTML属性完全相同。它们的主要区别是html方法总是返回第一个HTML元素的内容，所以你不需要像使用innerHTML那样引用数组索引。

2.12.4　操作 DOM

访问DOM的内容很有用，而修改这些内容则更加有用。最棒的是，我们已经知道怎么编写这样的代码了，就是使用jQuery的html方法或者纯JavaScript的innerHTML属性。

现在让我们修改节选的HTML5博客页面的h2元素内容：

```
var secondaryHeadings = $("h2");
secondaryHeadings.html("Now we've changed the content");
```

如果操作正确，应该能得到如图2-4所示的内容（注意标题已经改变）。

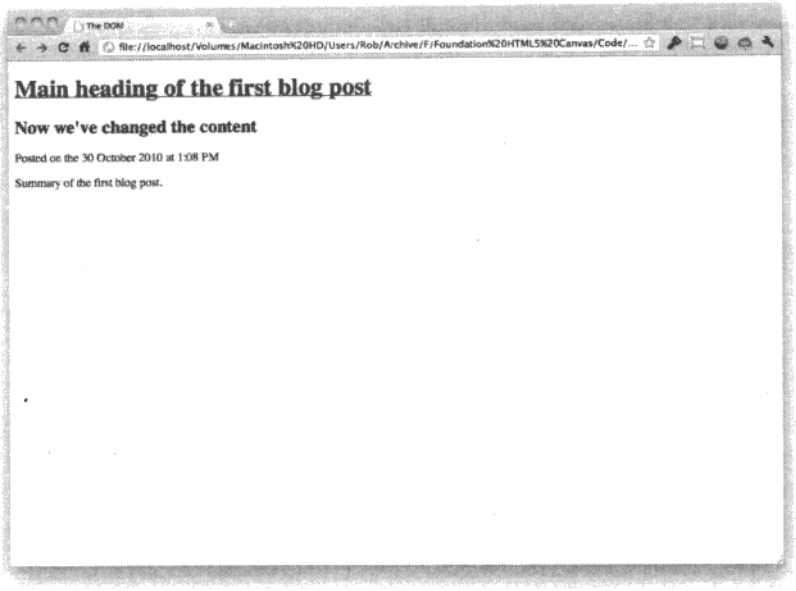


图2-4 使用jQuery操作DOM

一定要注意，我们并不是在编辑实际的HTML文件，而仅仅修改了它在浏览器的输出方式。我一定要强调这一点。如果你还不相信，那么请在浏览器上禁用JavaScript，然后再查看这个页面——标题会变回原来的值。

正如在本节开头所提到的，本节只介绍了DOM的皮毛。限于本书所涉及的范围，我无法对DOM的所有方面进行一一阐述，因为我必须关注我们真正会用到的功能。本章介绍的所有JavaScript知识也同样如此，不会涉及许多方面。如果你希望了解更多关于DOM或JavaScript的内容，建议你阅读一本专门介绍它们的图书，或者浏览W3Schools网站：www.w3schools.com。

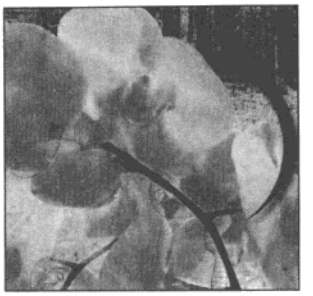
2.13 小结

本章涉及了相当多的内容，我向你推荐的知识就这么多了。在本章中，我尽可能地对我所了解的关于JavaScript的所有内容进行了阐述，其中有一些内容实际上应该用一整本书来专门进行介绍。当然，我们也了解了一些相关知识，如JavaScript的起源，为什么诸如jQuery这样的库会使我们的开发工作更简单，以及如何将JavaScript添加到HTML网页。同时还介绍了JavaScript的一些基础特性，如变量、数据类型、条件语句、函数、数组、循环、定时器、DOM和对象。细细列举，内容似乎很多，不是吗？

也许你现在应该休息一下。喝一杯茶，放松之余也总结一下我们所学的内容，然后再继续阅读本书。接下来，我们将学习如何使用HTML5 Canvas（这是我最喜欢的HTML5特性）。

第 3 章

Canvas基础知识



现在，你应该已经知道什么是Canvas以及为什么说它是HTML5中最炫的功能了吧。在本章中，我们将学习Canvas的特性，包括如何在HTML文档中引入Canvas以及在Canvas上绘制图形和各种对象。我们也将学习如何修改绘制在Canvas上的图形和对象，以及如何擦除它们。最后，将通过一个例子来学习如何将Canvas尺寸设置为浏览器窗口的大小，这是开发引人入胜的游戏的必备技能。希望在完成本章学习之后，你会更喜欢HTML5 Canvas以及它为你带来的各种机会。

3.1 认识 canvas 元素

与video和audio类似，canvas元素完全不需要任何外部插件就能够运行。你只需要使用HTML并使用2D渲染上下文API（2D rendering context API）编写一些JavaScript代码。即使你现在还不了解什么是2D渲染上下文API，也不用担心——你很快就会熟悉这方面知识。

canvas元素的用法很简单——我想说的是非常简单。开始只需要添加下面的代码：

```
<canvas width="500" height="500">
        <!--在此插入后备内容-->
</canvas>
```

我想我应该坦诚地告诉你，这段代码实际上并没有实现什么特殊的效果。它仅仅创建了一个新的空白canvas元素，还无法看到任何内容，因为还没有使用2D渲染上下文执行任何操作。我们很快就会在canvas上绘制一些图形，而绘制这些图形也是很简单的。

目前，在创建canvas元素时，需要特别注意的是width和height属性。这两个属性明确地定义了canvas元素的尺寸，从而相应地定义了2D渲染上下文的尺寸。如果不使用这些属性定义canvas元素的尺寸，那么2D渲染上下文会被设置为使用默认宽度和高度，分别是300和150像素。在本章后面的内容中，我们将学习如何创建一个能够动态修改尺寸并填充整个浏览器窗口的canvas元素。

> **注意** canvas元素的位置是由它在HTML文档的位置决定的。可以根据需要使用CSS移动，与移动其他HTML元素相同。

Canvas 的浏览器支持

大多数现代浏览器都支持canvas元素及其主要特性，但是毫无疑问，IE还不支持这些，至少在IE9之前的版本是不支持的。如果你接受这个现实，那么可以在canvas元素的后备内容中添加一条友好的消息，告诉使用IE的用户应该升级他们的浏览器。另外的方法是使用强大的ExplorerCanvas脚本，它是由Google的一些研究人员开发的。这个方法的好处在于只需要在网页上添加一个脚本，然后canvas元素就能够在IE9之前的版本上正常运行了。

如果对此感兴趣，可以从ExplorerCanvas网站[①]下载这个脚本，然后按照说明进行安装。

3.2 2D 渲染上下文

canvas元素并非Canvas中最强大的部分，真正的关键部分是2D渲染上下文，这是你真正绘制图形的地方。canvas元素的用途只是作为2D渲染上下文的包装器，它包含绘图和图形操作所需要的全部方法和丰富功能。理解这一点是很重要的，所以我再强调一下：绘图是在2D渲染上下文中进行的，而不是在canvas元素中进行。可以通过canvas元素访问和显示2D渲染上下文。我并不期望你现在就能够完全理解这一点，但是我希望当你开始使用Canvas进行实际开发时能够理解。

3.2.1 坐标系统

2D渲染上下文是一种基于屏幕的标准绘图平台。与其他的2D平台类似，它采用平面的笛卡儿坐标系统，左上角为原点(0,0)。向右移动时，*x*坐标值会增加；向下移动时，*y*坐标值会增加(见图3-1)。如果你想把图形绘制到正确的位置上，一定要理解这个坐标系统。

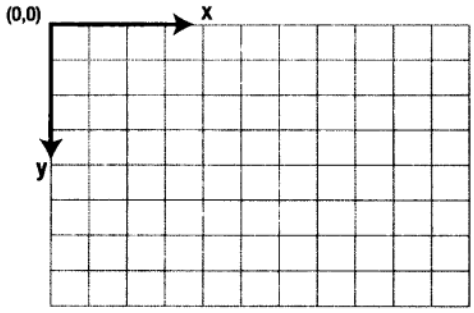


图3-1 2D渲染上下文的笛卡儿坐标系统

坐标系统的1个单位通常相当于屏幕的1个像素，所以位置(24,30)是向右24像素和向下30像

① http://code.google.com/p/explorercanvas/。

素的位置。有时候坐标系统的1个单位相当于2个像素（例如，在一些高分辨率显示器中），但是一般的经验法则是1个坐标单位等于1个屏幕像素。

3.2.2　访问2D渲染上下文

暂时不解释这个概念，先来使用一下。我们先创建只有一个空白canvas元素的简单HTML网页：

```
<!DOCTYPE html>

<html>
        <head>
                <title>Learning the basics of canvas</title>
                <meta charset="utf-8">

                <script type="text/javascript" src="http://ajax.googleapis.com
/ajax/libs/jquery/1/jquery.min.js"></script>

                <script type="text/javascript">
                        $(document).ready(function() {

                        });
                </script>
        </head>

        <body>
                <canvas id="myCanvas" width="500" height="500">
                        <!-- Insert fallback content here -->
                </canvas>
        </body>
</html>
```

如果现在运行，你不会看到任何内容，因此我们要访问2D渲染上下文，这样就可以开始绘图了。将下面的语句放到jQuery语句之中，方法与前一个JavaScript例子类似：

```
var canvas = $("#myCanvas");
var context = canvas.get(0).getContext("2d");
```

在这个例子中，我们将这个canvas元素赋值给一个变量，然后再通过调用getContext方法将得到的2D渲染上下文赋给另一个变量。必须强调一点，由于我们使用了jQuery，所以需要调用get方法才能访问DOM中的canvas元素，然后才能够访问Canvas的getContext方法。理解这一点并不难：你只需记住get方法本身与Canvas之间是没有任何关系的。

有了包含2D渲染上下文的变量之后，就可以开始绘制图形了。最令人激动的时刻到来了！在上下文变量声明语句之后添加下面这行代码：

```
context.fillRect(40, 40, 100, 100);
```

刷新页面，你会看到一些令人惊奇的变化，页面上出现一个黑色的矩形（见图3-2）。

你刚刚使用Canvas绘制了第一个元素。感觉不错吧？这个矩形是黑色的，因为Canvas所绘制元素的默认颜色是黑色。我们将在本章后面学习如何使用其他颜色。

图3-2 使用Canvas绘制对象

3.3 绘制基本图形和线条

正如你所看到的，绘制一个正方形是非常简单的。只需要使用一行代码，即调用fillRect方法：

```
context.fillRect(40, 40, 100, 100);
```

需要注意的一点是，调用的方法是fillRect，而不是fillSquare。我相信，大多数人都知道正方形实际上是一个等边矩形。如果还不知道，那么现在要记住，正方形实际上是各边相等的矩形。

创建一个矩形需要输入4个参数。前两个参数是正方形原点（左上角）的(x,y)坐标值，其余两个参数是矩形的宽度和高度。矩形宽度是(x,y)位置向右绘制的距离，而矩形高度是(x,y)位置向下绘制的距离。你现在就能明白，为什么理解坐标系统很重要，否则你可能会误认为高度是指从(x,y)位置向上绘制的距离。fillRect方法可以重写为以下形式，从而方便对参数的理解：

```
context.fillRect(x, y, width, height);
```

为了清楚起见，可以将正方形的宽度修改为200，保存文件，然后刷新页面（见图3-3）。

很惊奇吧，这是一个矩形。如果要在不同的位置绘制矩形呢？很简单，只需要修改(x,y)位置值。例如，将x坐标修改为200，y坐标修改为300（见图3-4）。

这正是Canvas的美妙之处。操作你所绘制的对象是非常简单的，只需要修改一些参数值。

> **注意** 有一个问题可能不太明显，如果你绘制的图形原点位于canvas元素之外，那么它将无法显示在屏幕上。只有当图形的原点或者某些部分位于canvas元素之内时，它才是可见的。

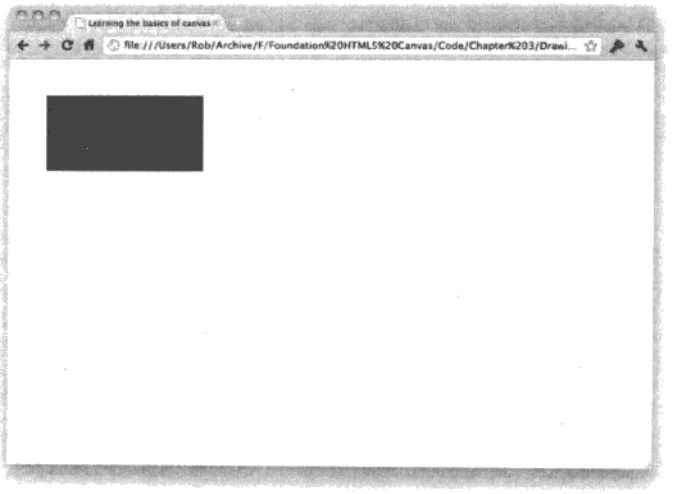


图3-3　绘制一个矩形

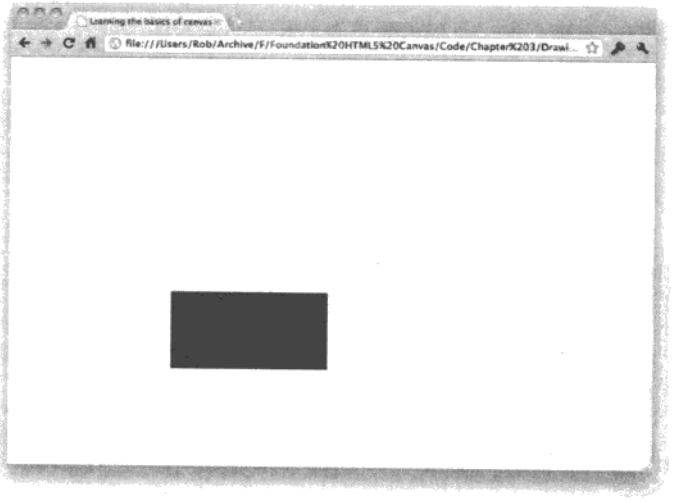


图3-4　在不同的位置绘制矩形

与fillRect相对应的方法是strokeRect。fillRect绘制一个矩形并给它填充颜色（在我们的例子中是黑色），strokeRect则绘制一个矩形并给它绘制边框，也就是用线条绘制出矩形的轮廓。如果将使用fillRect的例子修改为使用strokeRect，那么你就会明白我所说的意思了（见图3-5）。

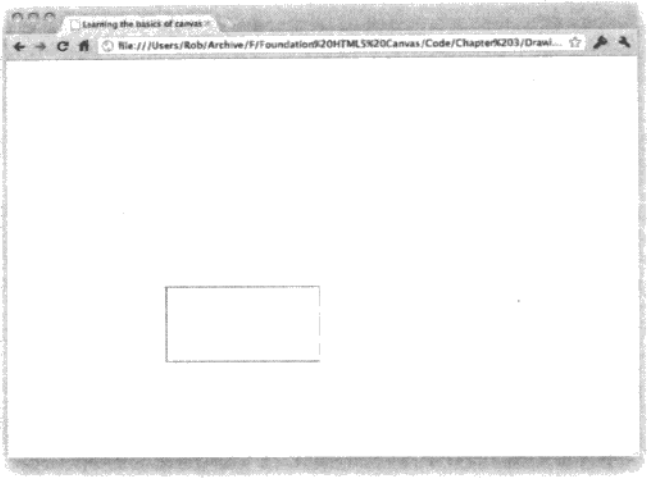


图3-5 绘制一个描边矩形

矩形现在加上了轮廓线，它实际上变成了中空的。由此可见，绘图是很有意思的，但是如何绘制一些更高级的图形呢，例如粗线条？没问题。

3.3.1 线条

绘制线条与绘制图形有一些区别。它们实际上称为路径。要绘制一个简单的路径，首先必须在2D渲染上下文中调用beginPath方法，意思实际上就是说："准备，要开始画路径了。"下一个调用的方法是moveTo，它会设置要绘制路径的原点坐标(*x*, *y*)。然后调用lineTo方法设置线条的终点坐标(*x*,*y*)，再调用closePath完成路径的绘制。最后，调用stroke绘制它的轮廓，显示线条。将全部步骤放到一起，就得到了下面的代码：

```
context.beginPath(); // 开始路径
context.moveTo(40, 40); // 设置路径原点
context.lineTo(340, 40); // 设置路径终点
context.closePath(); // 结束路径
context.stroke(); // 绘出路径轮廓
```

最后，我们得到一条直线（见图3-6）。

但是，直线并不一定是水平或垂直的，通过修改lineTo方法的坐标(*x*,*y*)参数，就能够绘制出斜线：

```
context.lineTo(340, 340);
```

结果如图3-7所示。

直线本身并没有什么特别的，但是通过组合，它们能够产生复杂且令人惊奇的图形。下一章将介绍路径的高级特性。现在，我们先用它来干点别的。先画一个圆形如何？肯定也是非常有趣的。

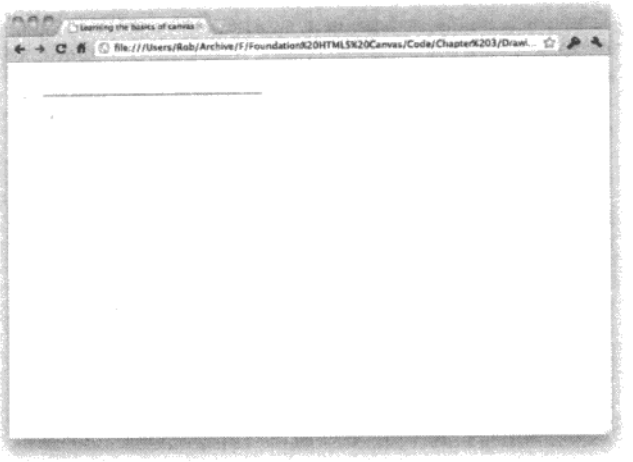


图3-6　绘制一条直线

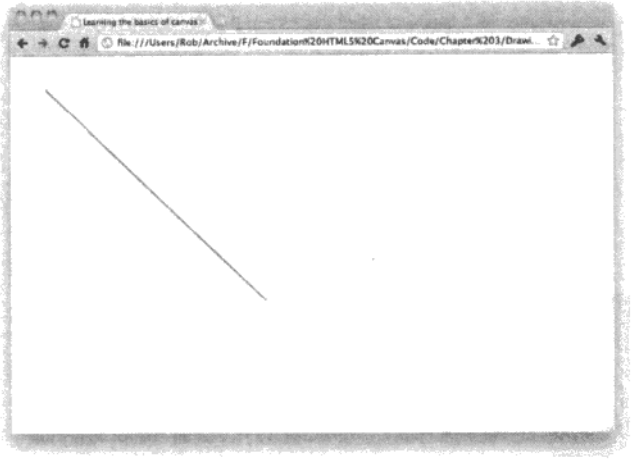


图3-7　绘制一条斜线

3.3.2　圆形

要理解圆形与矩形之间存在很大区别并不难。然而，认识这一点能够说明为什么在Canvas中绘制圆形与矩形也有很大区别。圆形是一个非常复杂的形状，因此Canvas 实际上并没有专门绘制圆形的方法。但是有一个方法可以绘制圆弧，圆弧实际上是圆形的组成部分——首尾相连的圆弧就是圆形。

这有点难懂，所以我们暂且不去解释，先在Canvas中创建一个圆形：

```
context.beginPath(); // 开始路径
context.arc(230, 90, 50, 0, Math.PI*2, false); // 绘制一个圆形
context.closePath(); // 结束路径
context.fill(); // 填充路径
```

你现在已经理解第一行及最后两行代码了，它们负责开始和结束路径（即圆弧），然后在它们完成时填充路径（fill是与stroke类似的方法）。最关键的是第二行代码，它包含绘制圆形所需要的全部信息。这似乎有些复杂，所以让我们来分析一下。

创建一个圆弧需要使用6个参数：圆弧原点的(*x*, *y*)坐标值（也是我们例子中的圆心）、圆弧半径、开始角度、结束角度和一个布尔值，如果圆弧按逆时针方向绘制，那么它为true，否则它为false。方法arc可以重写为下面更具可读性的形式：

```
context.arc(x, y, radius, startAngle, endAngle, anticlockwise);
```

前面三个参数都很简单，这里不过多解释。开始角度和结束角度参数表面上很简单，但是需要适当解释才能够很好地理解它们的使用方法。简言之，在Canvas中，一条弧线是由一条曲线定义的，它从与原点(*x*, *y*)距离为一个半径且角度为开始角度的位置开始。这条路径最后停在离原点(*x*, *y*)一个半径且角度为结束角度的位置上（见图3-8）。

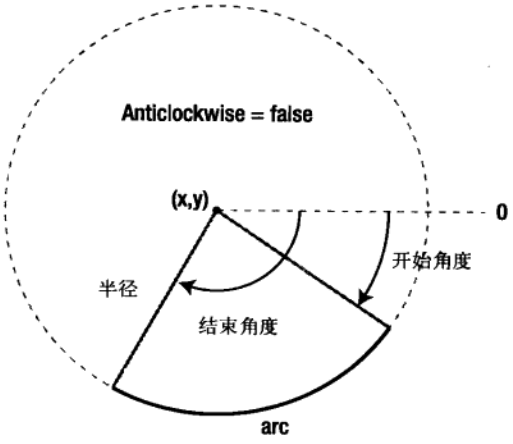


图3-8 绘制一条圆弧

一定要注意，Canvas中的角度是以弧度而不是角度为单位的。简单地说，360度（一个完整的圆）是2π（pi的2倍）弧度。

我们都知道如何将角度换算成弧度，它们可以按照以下公式进行换算（当然，这里是用JavaScript语句表示的）：

```
var degrees = 1; // 1度
var radians = degrees * (Math.PI / 180); // 0.0175弧度
```

本书中将采用弧度，这样就不需要进行角度到弧度的换算了。你可以使用图3-9作为参考，从而更容易以弧度来确定圆的角度。

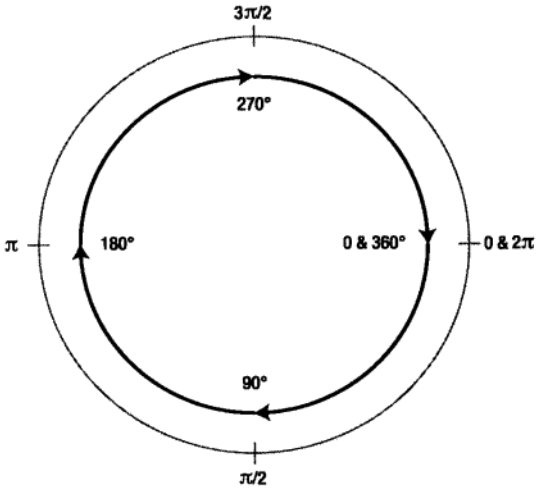


图3-9　角度与弧度之间的换算

对于更复杂的情况，可以使用前面介绍的公式进行换算。

那么，现在你对角度在Canvas中的用法有所了解了。让我们把注意力转移回画圆形的例子。在那个例子中，我们所画弧线的开始角度是0，结束角度是Math.PI*2（pi乘以2），它们就是圆的开始和结束角度。如果你还不确定，请对照图3-9。

> **注意**　要在JavaScript中使用pi的值，你需要使用Math对象，它是一个特殊对象，允许你完成各种强大的数学计算。我们还会在其他一些任务中使用这个对象，如生成随机数。

运行这个例子，会在浏览器上看到如图3-10所示的显示结果。

一个圆画成了！那么，如果想要画一个半圆，应该如何设置结束角度呢？请对照图3-9。没错，非常简单，其JavaScript代码如下：

```
context.arc(230, 90, 50, 0, Math.PI, false); //绘制一个半圆
```

如果一切正常，应该会在浏览器上看到一个半圆（见图3-11）。

> **注意**　虽然arc方法的第6个参数是可选的，但是如果不传入这个参数，Firefox会抛出一个错误。因此，最好保留这个参数，以便明确地指定弧线的绘制方向。

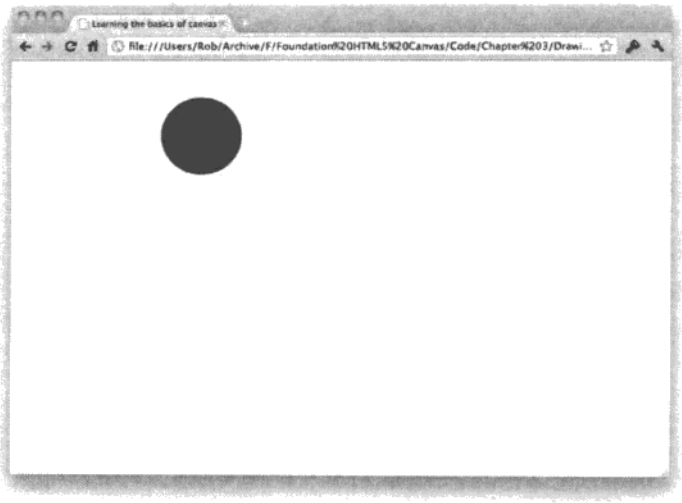


图3-10 绘制一个圆

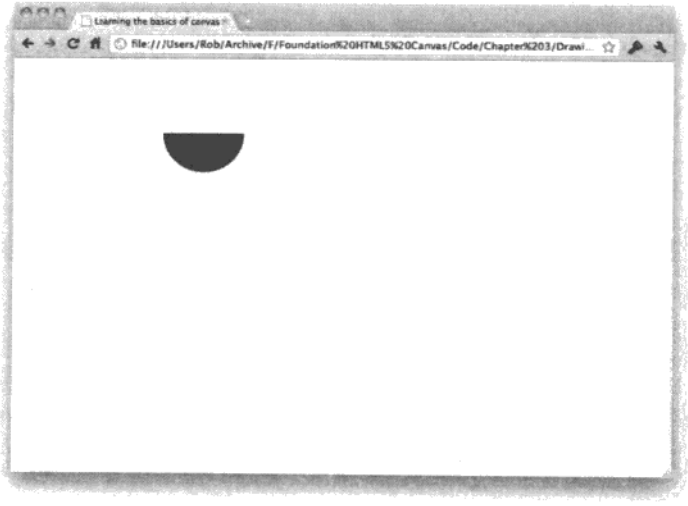


图3-11 绘制一个半圆

你还可以对角度进行任意调整，以创建1/4圆和任意饼形。然而，如果你想要了解这些图形的绘制方法，请另找时间进行尝试。我们需要继续关注更重要的方面，例如，修改图形的颜色！

3.4　样式

黑色太单调了，要是有一种方法能够修改图形和线条颜色该多好，有办法吗？这个方法容易吗？也是用一行代码就能实现吗？完全正确！我绝对没有说谎。让我们马上修改本章开头所创建的正方形的颜色。

```
context.fillStyle = "rgb(255, 0, 0)";
context.fillRect(40, 40, 100, 100);
```

通过设置2D渲染上下文的fillStyle属性，你就能够修改形状和路径的填充颜色。在前一个例子中，我们赋值了一个“rgb（红、绿、蓝）”颜色值，但是你也可以使用任何有效的CSS颜色值，如十六进制码（例如，#FF0000）或单词“red”。在这个例子中，颜色值设置为红色（纯红色，没有绿色和蓝色），最后正方形应该如图3-12所示。

图3-12　将填充颜色修改为红色

我说过这很简单，但是先不要太高兴，因为它还有一个问题，那就是在设置fillStyle属性之后，你所绘制的所有图形都会采用这个颜色。如果你接受这个结果，它就不是问题。但是如果你只希望修改一个对象的颜色，那么你一定要注意。有一个方法可以解决这个问题，就是当你在Canvas上绘制对象时，将fillStyle属性设置回黑色（或其他颜色），例如：

```
context.fillStyle = "rgb(255, 0, 0)";
context.fillRect(40, 40, 100, 100); // 红色正方形
context.fillRect(180, 40, 100, 100); // 红色正方形

context.fillStyle = "rgb(0, 0, 0)";
context.fillRect(320, 40, 100, 100); // 黑色正方形
```

结果如图3-13所示。

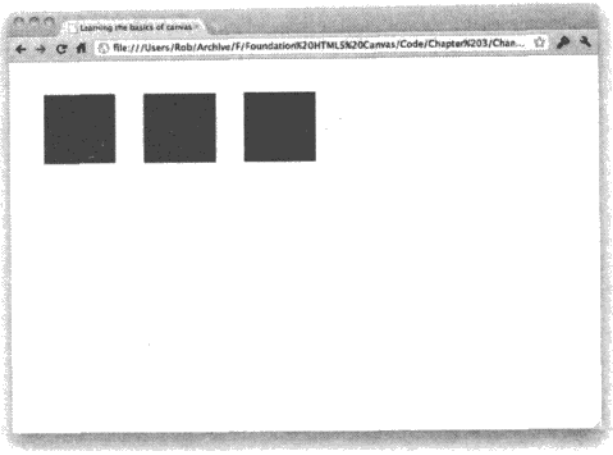


图3-13 将填充颜色改回黑色

还可以在描边图形和路径上使用strokeStyle属性实现变色效果。例如，下面的代码与前一个例子相同，唯一区别是它使用笔画描边而不是填充：

```
context.strokeStyle = "rgb(255, 0, 0)";
context.strokeRect(40, 40, 100, 100); // 红色正方形
context.strokeRect(180, 40, 100, 100); // 红色正方形

context.strokeStyle = "rgb(0, 0, 0)";
context.strokeRect(320, 40, 100, 100); // 黑色正方形
```

结果如图3-14所示。

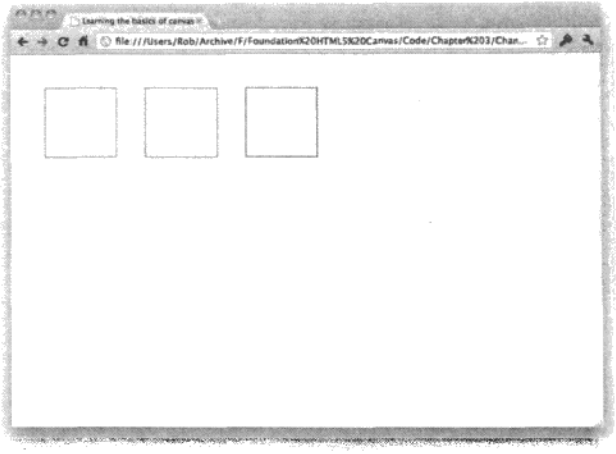


图3-14 修改描边颜色

> **注意**　完全可以同时使用fillStyle和strokeStyle为图形设置不同的填充和描边颜色。

这其中并没有什么复杂的地方，所有代码都非常简单。修改线条的颜色也是非常简单的：

```
context.strokeStyle = "rgb(255, 0, 0)";
context.beginPath();
context.moveTo(40, 180);
context.lineTo(420, 180); // 红色线条
context.closePath();
context.stroke();

context.strokeStyle = "rgb(0, 0, 0)";
context.beginPath();
context.moveTo(40, 220);
context.lineTo(420, 220); // 黑色线条
context.closePath();
context.stroke();
```

结果如图3-15所示。

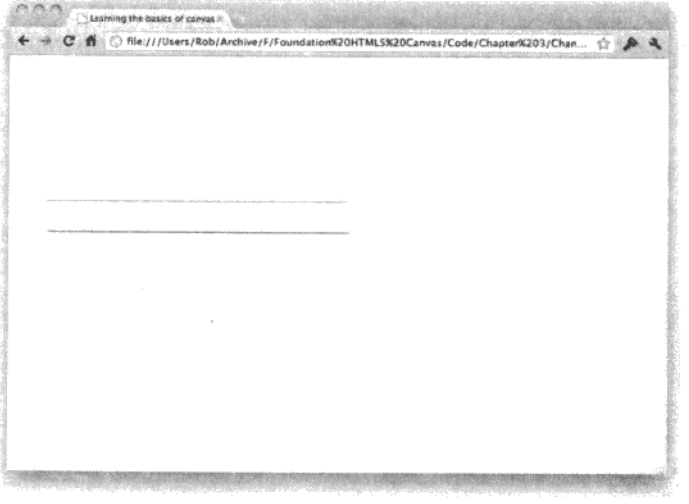


图3-15　修改线条的颜色

以上就是在Canvas中修改颜色的所有方法。

修改线宽

修改颜色很有意思，但是我们例子中的线条还有些细。Canvas有一个方法可以增加线宽，即2D渲染上下文的lineWidth属性。lineWidth属性的默认值为1，但是可以将它修改为任意值。例如，修改红线和黑线的宽度：

```
context.lineWidth = 5; // 加粗线条

context.strokeStyle = "rgb(255, 0, 0)";
context.beginPath();
context.moveTo(40, 180);
context.lineTo(420, 180); // 红线
context.closePath();
context.stroke();

context.lineWidth = 20; // 进一步加粗线条

context.strokeStyle = "rgb(0, 0, 0)";
context.beginPath();
context.moveTo(40, 220);
context.lineTo(420, 220); // 黑线
context.closePath();
context.stroke();
```

其结果是得到一条稍粗的红线和一条非常粗的黑线（见图3-16）。

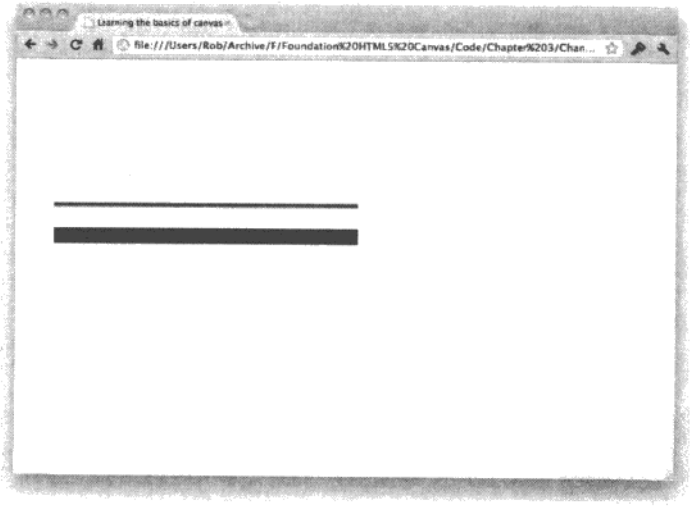


图3-16 修改线宽

lineWidth属性也会影响图形：

```
context.lineWidth = 5; // 加粗线条

context.strokeStyle = "rgb(255, 0, 0)";
context.strokeRect(40, 40, 100, 100); // 红色正方形
context.strokeRect(180, 40, 100, 100); // 红色正方形

context.lineWidth = 20; // 进一步加粗线条

context.strokeStyle = "rgb(0, 0, 0)";
context.strokeRect(320, 40, 100, 100); // 黑色正方形
```

你可能已经猜到，最终得到两个边框稍粗的红色正方形和一个边框非常粗的黑色正方形（见图3-17）。

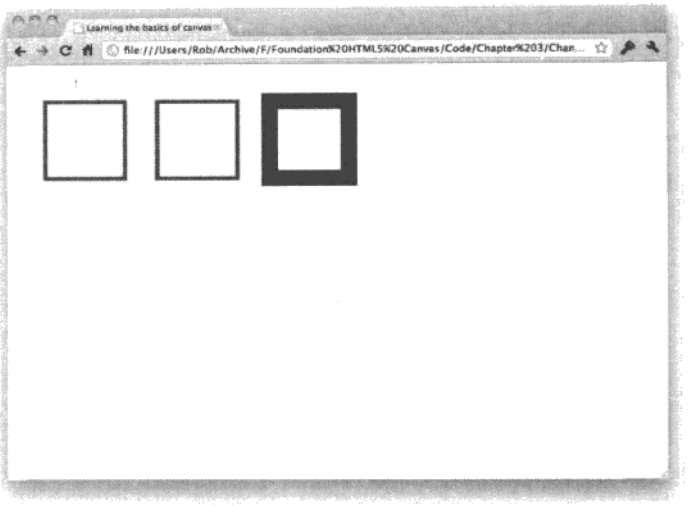


图3-17　修改图形的线宽

你现在已经掌握了基本的绘图方法，但是在继续学习真正强大的特性之前，我们还需要学习其他一些特性。

3.5　绘制文本

Canvas不仅能绘制图形，还能够显示文本。老实说，与使用传统的HTML元素（如p元素）创建文本相比，使用Canvas绘制文本通常并不是好方法，原因如下。

Canvas中的文本是以图像形式绘制的，这意味着它无法像HTML文档中的普通文字一样用鼠标指针选取——它实际上不是文本，只是像文本而已。如果你以前使用过微软画图程序，那么就会理解我的意思。一旦文字绘制之后，它就无法编辑，除非先擦除文字，再重新绘制。在Canvas中绘制文本的好处是你可以利用Canvas支持的强大转换和其他绘图功能。然而，我必须提醒你，除非你有充分理由不使用普通的HTML元素，否则一定不要在Canvas中创建文本。相反，你应该使用普通的HTML元素来创建文本，然后使用CSS定位到Canvas之上。关键是使用HTML来处理文本（内容），而使用Canvas来处理像素和图形。

下面是我想要介绍的Canvas绘制文本的方法，其实很简单：

```
var text = "Hello, World!";
context.fillText(text, 40, 40);
```

这就是你绘制文本所需要的代码。2D渲染上下文的fillText方法可接受4个参数（其中一个是可选的，所以我们现在省略了）。第一个参数是准备绘制的文本，第二个和第三个参数是文本原点（左下角）的(x,y)坐标值。我都说过很简单了。

由于字号太小无法看清楚，我现在还不准备展示输出效果，这是因为Canvas的默认文本设置是10px sans-serif（非常小）。所以现在让我们修改字号，同时也会介绍修改字体的方法。要实现这个操作，你需要设置2D渲染上下文的font属性，如下所示：

```
var text = "Hello, World!";
context.font = "30px serif"; // 修改字号和字体
context.fillText(text, 40, 40);
```

属性font可接受与CSS的font属性完全相同的字符串值。在前一个例子中，我们指定了字体的像素大小，然后是希望使用的字体。设置为serif表示计算机的默认字体是serif字体（与Times New Roman类似）。所有代码组合在一起将得到如图3-18所示的结果。

图3-18 绘制文本和修改字号

这样显示效果会好一些，能看清了。如果愿意，甚至可以将文本设置为斜体：

```
var text = "Hello, World!";
context.font = "italic 30px serif";
context.fillText(text, 40, 40);
```

这里所做的唯一修改就是将单词italic添加到font属性中，这样就得到如图3-19所示的结果。

除了font属性，还可以使用许多设置，如行高和备用字体系列。本书不会介绍这方面内容，但是如果你对在Canvas中使用文本感兴趣，建议你自己学习。

> **注意** 我希望你已经明白，Canvas基础知识是非常简单的。其原因是2D渲染上下文API所使用的方法和属性的命名方式都是简单易懂的。这些API也说明了为何我在第2章强调变量命名的重要性。

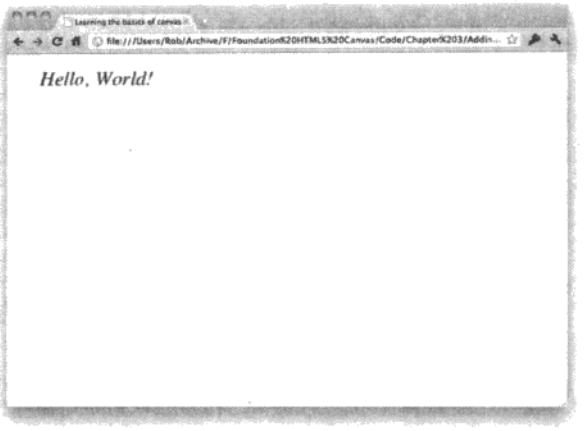


图3-19　绘制斜体文本

在继续之前，先让我们介绍如何创建描边文本——这也是很有用的：

```
var text = "Hello, World!";
context.font = "italic 60px serif";
context.strokeText(text, 40, 100);
```

这次使用的是strokeText方法，它的参数与fillText完全相同。字号过小会让文本难以辨别，所以在这个例子中，我们加大了字号，而原点也稍微向下移，所以文本不会超出屏幕顶部。最终得到的结果如图3-20所示。

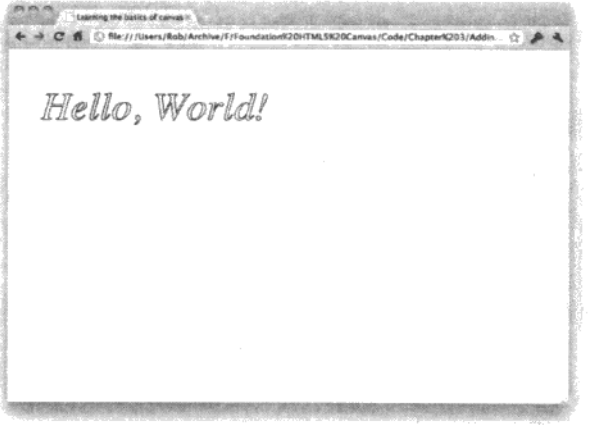


图3-20　绘制描边文本

我们并不常用描边文本，但是它在一些项目中是必不可少的。如果遇到这种情况，建议你尽量使用这个功能。

3.6 擦除 Canvas

在Canvas上绘制确实是很有趣的事情，但是当你画错了或者想要清除画布和绘制其他图形时，应该如何做呢？有两个方法可以使用：clearRect方法以及宽度/高度技巧。我们首先学习2D渲染上下文的clearRect方法。

假设你在Canvas上只画了一个正方形和圆形：

```
context.fillRect(40, 40, 100, 100);

context.beginPath();
context.arc(230, 90, 50, 0, Math.PI*2, false);
context.closePath();
context.fill();
```

然后，可以随时清除Canvas。要执行这个操作，只需要使用Canvas的原点坐标(x,y)、宽度和高度调用clearRect。如果Canvas宽500像素高500像素，那么可以按照以下方式调用clearRect：

```
context.clearRect(0, 0,.500, 500);
```

当运行时，它在浏览器上不会显示任何内容，因为刚刚清除了Canvas的所有内容。甚至，即使不知道Canvas尺寸，也可以使用jQuery的width和height方法来调用clearRect，方法如下：

```
context.clearRect(0, 0, canvas.width(), canvas.height());
```

完整的例子如下所示：

```
var canvas = $("#myCanvas");
var context = canvas.get(0).getContext("2d");

context.fillRect(40, 40, 100, 100);

context.beginPath();
context.arc(230, 90, 50, 0, Math.PI*2, false);
context.closePath();
context.fill();

context.clearRect(0, 0, canvas.width(), canvas.height());
```

我在这个例子中加入了原始的canvas变量，目的是为了提醒你，我们调用的是哪个对象的clearRect方法。

> **注意** canvas元素实际上提供了windth和height属性，所以你可以自己决定是使用jQuery方法，还是使用纯JavaScript方法来访问Canvas的尺寸。

但是，并不一定要清除整个Canvas，可以只清除Canvas的一个特定区域。例如，如果我们只想清除例子中的正方形，可以按以下方式调用clearRect：

```
context.clearRect(40, 40, 100, 100);
```

这样就剩下一个圆形（见图3-21）。

这种方法是通过修改clearRect的参数来清除一个特定区域。在我们的例子中，我们将准备擦除的区域的原点（左上角）移动到正方形的左上角(40, 40)，并将准备擦除的区域的宽度和高度设置为正方形的宽度和高度(100)。其结果是只将正方形所在的特定区域清除。按照以下方式修改clearRect的参数，就能够将圆形清除：

```
context.clearRect(180, 40, 100, 100);
```

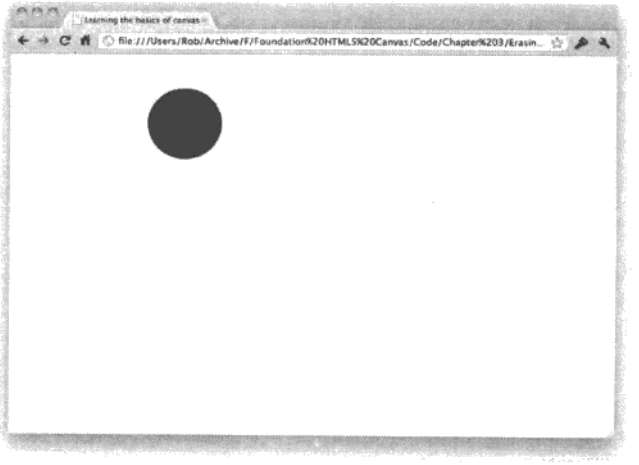


图3-21　擦除Canvas的特定区域

如果计算正确，画布中将只剩下正方形（见图3-22）。

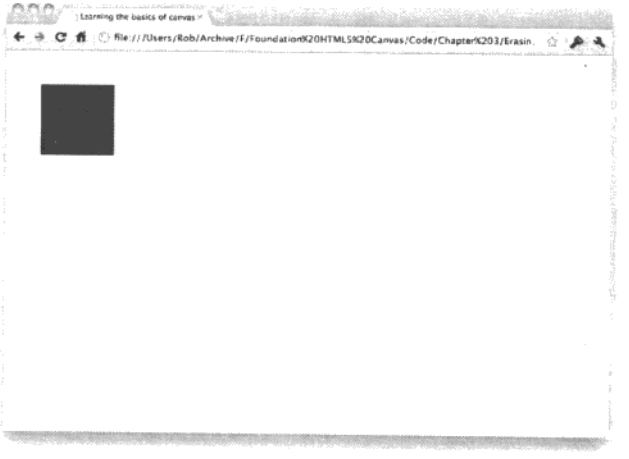


图3-22　擦除Canvas的特定区域

记住，弧形的原点是它的中心，所以为了获得clearRect方法所需要的正确原点，我们需要用原点的*x*和*y*坐标减去它的半径。

Canvas中的对象是可以被部分擦除的，虽然你可能并不需要这样做：

```
context.fillRect(40, 40, 100, 100);

context.beginPath();
context.arc(230, 90, 50, 0, Math.PI*2, false);
context.closePath();
context.fill();

context.clearRect(230, 90, 50, 50);
```

这个例子会切掉圆形的一部分（见图3-23）。

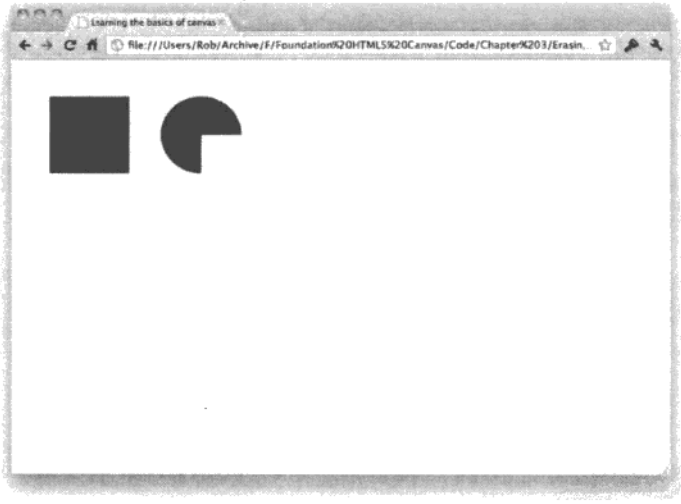


图3-23 擦除图形的一部分

我们可以先绘制一个基本形状，然后再去掉一部分，从而快速方便地绘制出一些复杂的形状。

宽度/高度技巧

如果只是想要擦除Canvas上的所有内容，并从零开始绘图，那么你可能要考虑使用宽度/高度技巧。老实说，这实际上并不是一种技巧，而是一种将Canvas重置为默认新状态的方法，但是关于它的文档很少。其依据是每当重新设置一个canvas元素的width和height属性时，Canvas都会自动清除内容并返回其原始状态。这种方法也有一些缺点，所以先看一个例子：

```
context.fillStyle = "rgb(255, 0, 0)";

context.fillRect(40, 40, 100, 100);

context.beginPath();
context.arc(230, 90, 50, 0, Math.PI*2, false);
context.closePath();
context.fill();
```

上面的代码仅仅是在Canvas上绘制出一个红色正方形和一个圆形。现在让我们来重置这个Canvas：

```
canvas.attr("width", canvas.width());
canvas.attr("height", canvas.height());
```

在这里，我们使用了jQuery的方法。需要修改canvas元素的width和height属性，所以可以使用jQuery的attr方法。我希望你现在已经知道操作的结果是什么了。实际上，我传入了希望修改的属性名称（width和height），后面紧跟它们相应的值（与之前相同的宽度和高度）。如果一切正确，你会看到一个空白的Canvas。

现在，将下面这行代码添加到使用宽度/高度技巧清除Canvas内容的代码之后：

```
context.fillRect(40, 40, 100, 100);
```

这肯定会绘制出一个红色正方形，对吗？（记住：之前设置了fillStyle属性。）那么，为什么它实际上绘制出了一个黑色正方形呢（见图3-24）？

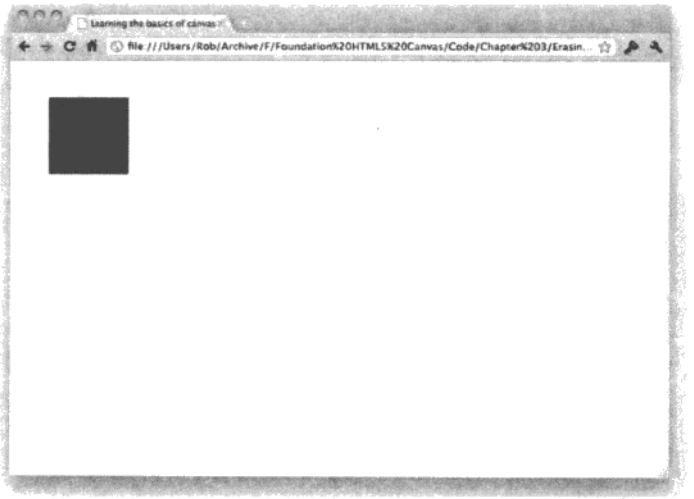


图3-24　使用宽度/高度技巧重置Canvas

宽度/高度技巧的缺点是，它会完全重置Canvas上的所有内容，包括样式和颜色。所以，只有准备完全重置Canvas，而不仅仅是清除内容时，你才能使用这种方法。

3.7 使 Canvas 填满浏览器窗口

到现在为止，我们使用的canvas元素一直采用固定的500像素的宽度和高度。这个尺寸没有问题，但是如果我们想要将它填满整个浏览器窗口，又该如何做呢？对于普通的HTML元素，可以将width和height属性设置为100%，然后一切就都满足要求了。然而，canvas元素并不支持这种方法，它会忽略百分比，将100%解释为100像素，200%解释为200像素，以此类推。因此，我们需要使用另一种方法。

最简单的方法是将canvas元素的宽度和高度精确设置为浏览器窗口的宽度和高度。可以使用window浏览器对象和jQuery方法获取窗口的宽度和高度：


```
var canvas = $("#myCanvas");
var context = canvas.get(0).getContext("2d");

canvas.attr("width", $(window).get(0).innerWidth);
canvas.attr("height", $(window).get(0).innerHeight);

context.fillRect(0, 0, canvas.width(), canvas.height());
```

我在这里使用了$(window).get(0).innerHeight，而不使用$(window).height()，原因是后者似乎并不支持所有浏览器。你会发现这种方法实际上并非完全正确，因为canvas元素和浏览器窗口旁边还会有一个白色空隙（见图3-25）。

图3-25 Canvas旁边的白色空隙

要解决这个问题，我们需要使用一些CSS。在文本编辑器中新建一个文件，将它命名为canvas.css，保存到HTML文档所在目录。在CSS文件中添加以下代码，然后保存：

```
* { margin: 0; padding: 0; }
html, body { height: 100%; width: 100%; }
canvas { display: block; }
```

第一行代码将所有HTML元素的margin和padding重置为0，从而删除图3-25所显示的白色边框；这一般称为CSS重置（reset）。还有其他更好的方法可以实现CSS重置，但是现在使用的这种方法已经满足我们的需要了。第二行代码并不是必需的，但是它可以保证html和body元素使用整个浏览器窗口的宽度和高度。最后一行代码将canvas元素从inline修改为block，这样我们才能够正确地设置宽度和高度，从而使之能够使用整个浏览器窗口的宽度和高度，而不会出现滚动条。

要在HTML文档中使用这个CSS文件，需要将下面的代码添加到head元素中引入jQuery的script元素之前：

```
<link href="canvas.css" rel="stylesheet" type="text/css">
```

这个元素将链接刚刚创建的CSS文件，然后运行其中的样式。结果就是canvas元素完美地填满浏览器窗口（见图3-26）。

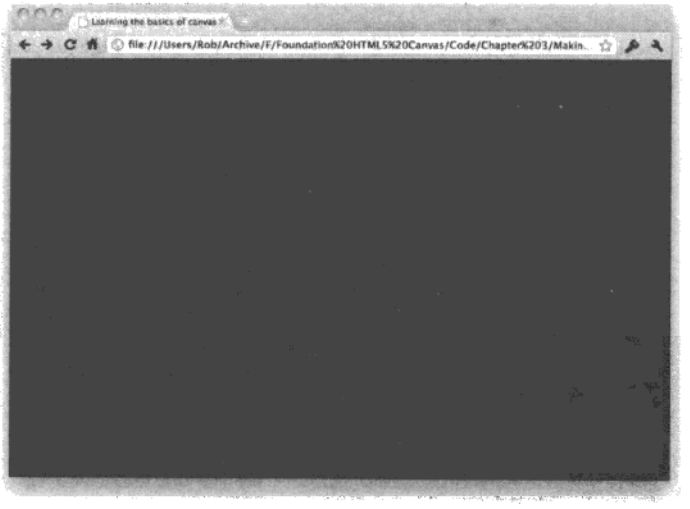


图3-26　使Canvas填满浏览器窗口

但是，还有问题要解决。如果你调整浏览器窗口大小，canvas元素仍然会保持原来的尺寸，这样，如果窗口缩小过多就会显示滚动条（见图3-27）。

为了解决这个问题，需要在调整浏览器窗口大小的同时调整canvs元素的大小。要是jQuery有一个当浏览器窗口大小调整时触发的resize方法该有多好，这个方法应该类似于DOM准备好时触发的ready方法。幸好，它还真有一个resize方法，而且完全满足我们的需要。

```
$(window).resize(resizeCanvas);
```

```
function resizeCanvas() {
        canvas.attr("width", $(window).get(0).innerWidth);
        canvas.attr("height", $(window).get(0).innerHeight);
        context.fillRect(0, 0, canvas.width(), canvas.height());
};

resizeCanvas();
```

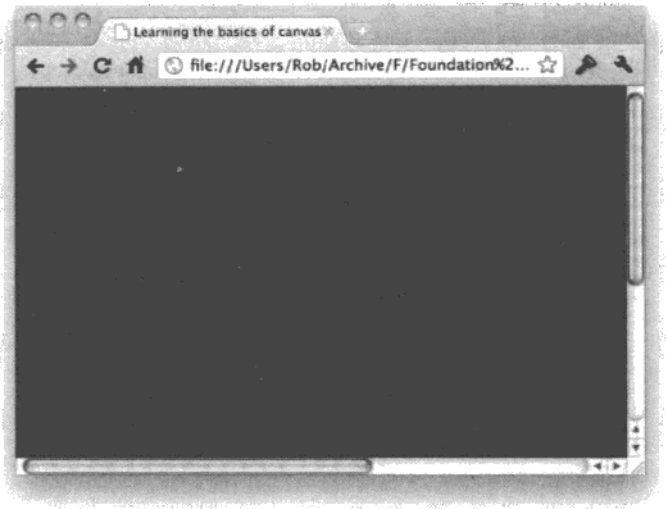


图3-27 调整浏览器窗口大小后出现的问题

这里并没有什么新代码，大多数代码都只有少量变化。主要增加的是jQuery的resize方法，它被设置为每当浏览器窗口发生大小变化时，就调用resizeCanvas函数。之前用来设置canvas元素宽度和高度的所有代码都移到这个函数之中，包括绘制与Canvas相同尺寸的矩形（记住：修改宽度和高度会重置Canvas，因此所有内容都必须重新绘制）。最后一个语句是调用resizeCanvas函数，在页面首次加载时执行一次初始化操作。

运行后，你会发现canvas元素可以完美地动态调整大小，不会出现滚动条。干得好！

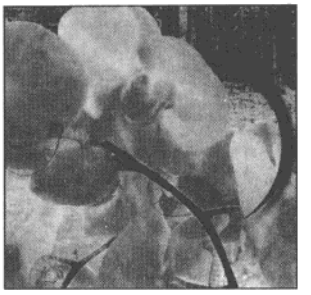
3.8 小结

本章介绍了各种有趣的内容，特别是全新的Canvas。你已经学习了如何使用canvas元素，如何绘制基本图形和路径，以及如何修改这些图形和路径的颜色。同时还学习了如何绘制文本，擦除Canvas，以及如何使Canvas填满浏览器窗口。本章内容确实很多，所以我认为你需要认真复习，慢慢吸收学到的新知识。

下一章内容将更精彩，你将学习到Canvas绘图的所有高级功能。

第4章

Canvas高级功能



在本章中，你将学习到Canvas提供的一些更高级的功能。你将看到在使用多种绘图样式时如何节省时间，以及如何转换和操作绘图来使其更激动人心。本章的中间部分将讲述如何创建阴影和渐变使图形看起来更真实有趣。本章最后会介绍使用高级路径创建特殊图形，以及如何将画布导出为图像，以便保存或留待将来使用。

本章内容非常精彩，我希望这些内容能够拓宽你的眼界，帮助你学会画布的高级功能。

4.1 保存和恢复绘图状态

在第3章中，我们经常在各种样式之间切换，甚至有时候会在不同颜色之间反复切换。这种重复是很麻烦的，它意味着如果你想要返回之前使用的一些样式，必须重写大量的代码。幸好，画布能够记住一些样式和属性，这样将来你就可以再次使用。这就是所谓的保存和恢复画布绘图状态。然而，问题是，如果要记住多个状态，操作起来可能令人困惑，因为你必须跟踪所有发生的变化。但是不用担心，听完我的讲解你就会完全清楚其中的奥妙。

4.1.1 画布绘图状态是什么

如果我走过来对你说："看看你的状态！"我不光可能会冒犯你，而且还直接查问了你看起来怎样（尽管我确信你的样子会很不错）。无论是在现实世界还是画布中，"状态"这个词都是用来描述事物在特定时刻所处的状况。重要的是要抓住与所描述时间直接关联的对象状态。例如，如果我要描述你昨天及今天的状态，那么它们一定是两个完全不同的状态——你今天的状态可能不如昨天（但我相信你不是）。简而言之，状态总在变化。

在画布中，绘图状态指的是描述某一时刻2D渲染上下文外观的整套属性，从简单的颜色值到复杂的变换矩阵（transformation matrix）及其他特性。我们将在本章后面介绍变换矩阵，所以请不必担心我刚刚提到的这些专业词汇。

用于描述画布绘图状态的全部属性为：变换矩阵、裁剪区域（clipping region）、`globalAlpha`、`globalCompositeOperation`、`strokeStyle`、`fillStyle`、`lineWidth`、`lineCap`、

lineJoin、miterLimit、shadowOffsetX、shadowOffsetY、shadowBlur、shadowColor、font、textAlign和textBaseline。在本章中，你将学习到大多数我们尚未接触过的属性。

有一点很重要，画布上的当前路径和当前位图（正在显示的内容）并不属于状态。我们更应该将状态看做2D渲染上下文属性的描述，而不是画布上显示的所有内容的副本。

4.1.2 保存绘图状态

在开始之前，一定要按照第3章开头的描述创建一个新的HTML网页。如果你想要使用自己的HTML，或者只是想阅读示例代码而不想自己编写，我也不反对。

现在，有个好消息是保存画布状态非常简单。你需要做的就是调用2D渲染上下文的save方法。仅此而已。不需要怀疑，我们马上就来看看简单的save方法：

```
var canvas = $("#myCanvas");
var context = canvas.get(0).getContext("2d");

context.fillStyle = "rgb(255, 0, 0)";
context.save(); // 保存画布状态
context.fillRect(50, 50, 100, 100); // 红色正方形
```

那么，当你保存绘图状态时，实际上发生了什么呢？可以肯定的是，它必须保存在某个地方，对吗？正确。2D渲染上下文会保存一个绘图状态栈，实际上它是一组之前保存的状态，其中最近保存的状态位于顶部——就像一叠纸。绘图状态的默认栈是空的，调用save方法，就会有一个新状态被放入（添加到）这个栈。这意味着，你完全可以多次调用save方法，将多个绘图状态逐一保存到栈中，其中最早的状态在底部。然而，这其中有一点不易理解，那就是你无法将任何绘图状态后移，因为这个过程是有严格顺序的。但是现在请不要担心——我们很快就会介绍多状态。现在，我们先来了解一下如何访问刚刚保存的状态。

4.1.3 恢复绘图状态

访问一个已有绘图状态与保存它一样简单，唯一的区别是这次调用的是restore方法。现在，如果你绘制另一个正方形，并且这次将fillStyle设置为蓝色，那么很快会看到画布绘图状态的好处：

```
context.fillStyle = "rgb(0, 0, 255)";
context.fillRect(200, 50, 100, 100); // 蓝色正方形
```

这里并没有执行任何特殊操作，唯一修改的是填充颜色（参见图4-1）。

但是，如果你想换回之前使用的红色填充颜色，该怎么做呢？我希望你不会考虑再次重写fillStyle属性并将它设置为红色！哦，你没这样想？太聪明了！没错，你将颜色设置为红色之后保存了绘图状态，所以它已经存在于栈中了，你只需要在现有代码之前调用restore，就可以恢复原先的状态：

```
context.restore(); // 恢复画布状态
context.fillRect(350, 50, 100, 100); // 红色正方形
```

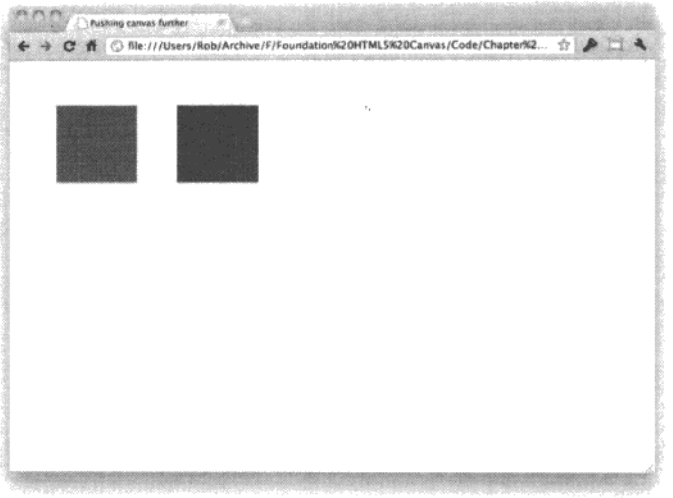


图4-1　在恢复绘图状态之前修改填充颜色

通过调用restore方法，你能够自动取出最后添加到栈中的绘图状态，并将它应用于2D渲染上下文，用所保存的状态覆盖全部现有的样式。这意味着，虽然你没有在代码中直接修改fillStyle属性，但是它将取得所保存的绘图状态的值——它会变成红色（参见图4-2）。如果只是修改颜色，效果可能还不够明显，但这个概念适用于所有能够保存到绘图状态中的画布属性。

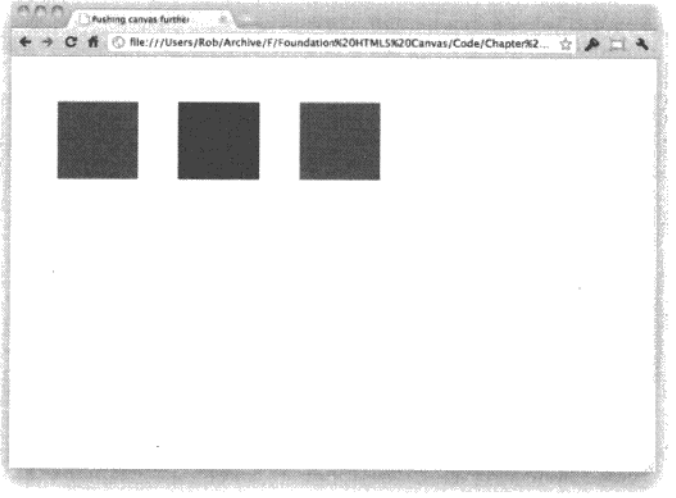


图4-2　恢复绘图状态

4.1.4 保存和恢复多个绘图状态

在本节开头，我曾提到过一次处理多个状态有一些复杂。但是，在学完前面的内容之后，我希望现在你已经理解该如何处理它了。实话说，如果理解了栈的概念，并且明白新增的项被添加到栈的顶部，并且它们是从栈顶部取回的，那么你就不会觉得它复杂了。栈实际上采用一种后进先出的机制，最近保存到栈的绘图状态将是后来第一个恢复的状态。

如果你修改前面的例子，在将fillStyle设置为蓝色后保存绘图状态，就会明白我的意思：

```
context.fillStyle = "rgb(255, 0, 0)";
context.save();
context.fillRect(50, 50, 100, 100); // 红色正方形

context.fillStyle = "rgb(0, 0, 255)";
context.save();
context.fillRect(200, 50, 100, 100); // 蓝色正方形

context.restore();
context.fillRect(350, 50, 100, 100); // 蓝色正方形
```

第三个正方形现在不是红色，而是蓝色。这是因为最后保存到栈的绘图状态是蓝色的fillStyle，所以它最先恢复（参见图4-3）。

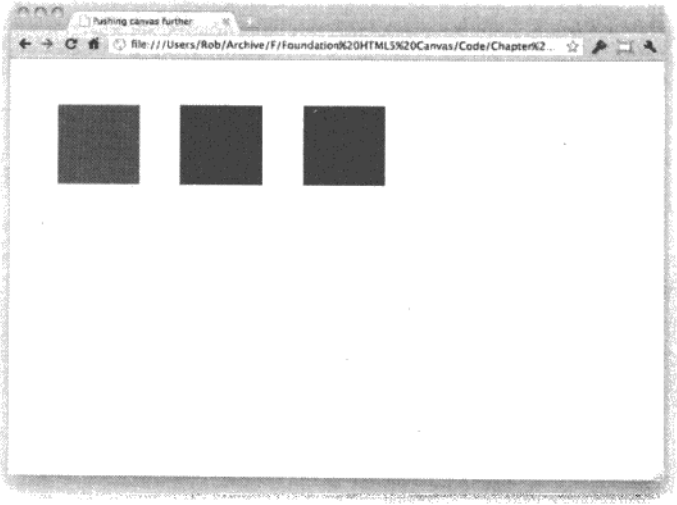


图4-3 保存多个绘图状态

另一个状态是红色的fillStyle，它仍然在栈中等待，你只需要再调用一次restore就能够恢复这个状态：

```
context.restore();
context.fillRect(50, 200, 100, 100); // 红色正方形
```

这会从栈返回最后一个状态，并将它删除，使栈变成空的（参见图4-4）。

关于绘图状态的保存和恢复还有很多其他内容，本节的目的只是介绍一些基础知识。从现在开始，你就能够理解后续章节关于绘图状态的使用方法了，因此能够有更充裕的时间学习其他更有趣的功能。

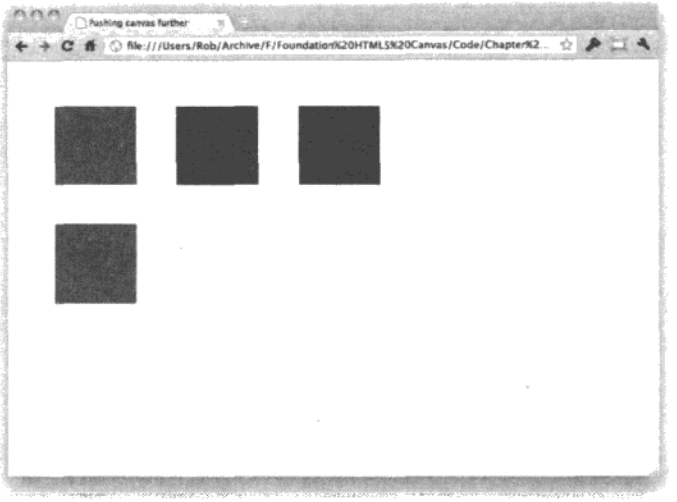


图4-4　恢复多个绘图状态

4.2　变形

到现在为止，你在画布中绘制的所有元素都是按照它应该出现的样子绘制的。例如，矩形是按照fillRect方法定义的位置和尺寸绘制的，并且它是用水平和垂直的线条绘制的；平淡无奇。但是，如果你想要画一些奇特的图形呢？如果想要旋转一个矩形呢？如果想要缩放图形呢？2D渲染上下文的变形功能能够帮助你实现所有这样的操作。它们支持的功能是非常强大的。

4.2.1　平移

最基本的操作就是平移，即将2D渲染上下文的原点从一个位置移动到另一个位置。在画布中进行平移使用的是translate方法时，实际上它移动的是2D渲染上下文的坐标原点，而不是所绘制的对象（参见图4-5）。

translate方法的调用方式如下：

```
context.translate(150, 150);
```

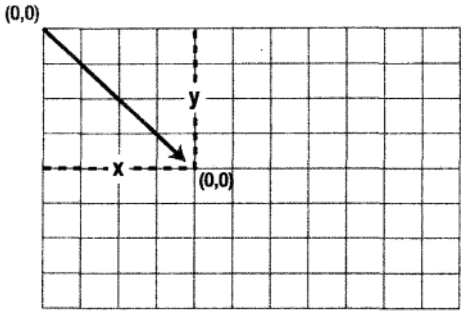


图4-5 translate方法移动2D渲染上下文的原点

两个参数是 (*x*, *y*)坐标值，表示把2D渲染上下文的原点移动多远。一定要注意，将来你所指定的(*x*, *y*)坐标值会加上原点的平移，原点最初的默认值是(0, 0)。例如，如果执行两次与上面例子完全相同的平移，那么实际上是将原点在*x*轴方向移动300个单位（0+150+150），在*y*轴方向也移动300个单位（0+150+150）。

通过移动2D渲染上下文的原点，画布中的所有对象都将移动相应的距离：

```
context.fillRect(150, 150, 100, 100);

context.translate(150, 150);

context.fillStyle = "rgb(255, 0, 0)";
context.fillRect(150, 150, 100, 100);
```

一般情况下，第二次调用fillRect时，所绘制的正方形的原点坐标是(150, 150)，但是由于执行了一次平移，这个正方形的原点现在变成(300, 300)（参见图4-6）。

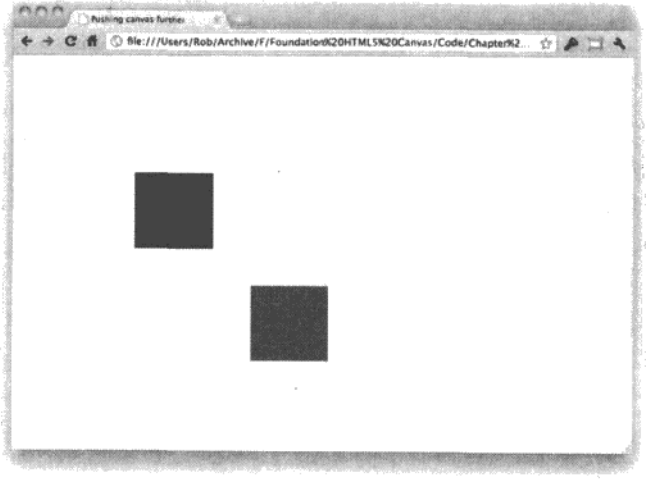


图4-6 平移会影响图形原点

一定要理解这其中的原理。红色正方形的原点仍然为(150, 150)，它只是看上去又平移了150像素，这是因为在黑色正方形绘制之后，2D渲染上下文的原点已经平移了150像素。如果你希望红色正方形仍然出现在点(150, 150)原来的位置（即黑色正方形所在位置），那么可以直接将它的原点设为(0, 0)：

```
context.translate(150, 150);
context.fillStyle = "rgb(255, 0, 0)";
context.fillRect(0, 0, 100, 100);
```

这是因为你已经将2D渲染上下文移动到位置(150, 150)，所以从现在开始，所有在点(0, 0)绘制的图形实际上都显示在点(150, 150)上（参见图4-7）。

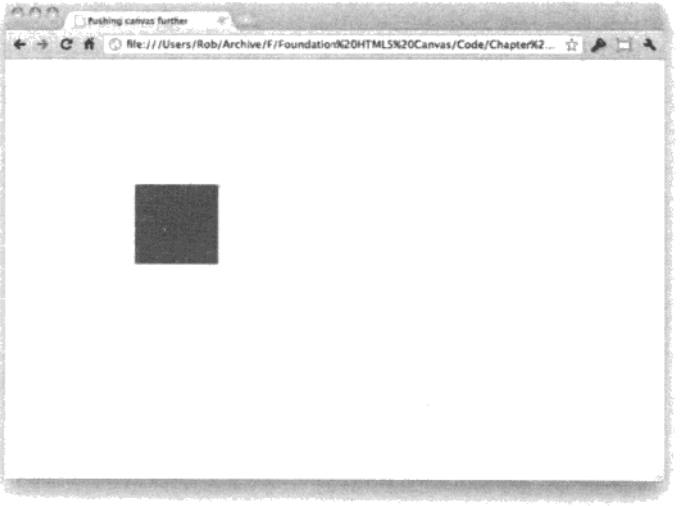


图4-7　平移后保持原点

> **注意**　每一种变形方法，包括平移，都会影响方法执行后所绘制的所有元素。这是因为它们都是直接在2D渲染上下文上操作的，而不是只针对所绘制的图形。这与你修改了fillStyle等属性的效果一样，新的颜色会影响后来绘制的所有元素。

4.2.2　缩放

另一个变形方法就是缩放（scale），顾名思义，它是调整2D渲染上下文的尺寸。它与平移的区别在于(*x*, *y*)参数是缩放倍数，而不是像素值。

```
context.scale(2, 2);
context.fillRect(150, 150, 100, 100);
```

这个例子将2D渲染上下文的*x*和*y*方向都乘以2。通俗地说，2D渲染上下文及其绘制的所有对象现在都变成2倍尺寸（参见图4-8）。

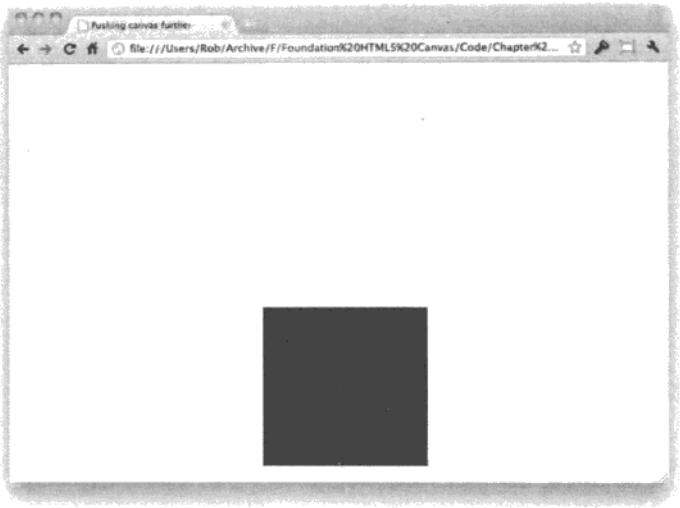


图4-8 缩放2D渲染上下文

单独使用scale将使所有绘图内容变大，而且它也会使一些对象被画在一些不恰当的位置上。例如，放大2倍实际上意味着现在1个像素变成2个像素；所以如果你绘制了一个*x*为150像素的图形，现在它看起来像是变成*x*为300像素了。如果这不符合你的要求，或者你只想要缩放一个图形，可以组合使用scale和translate方法。

```
context.save();
context.translate(150, 150);
context.scale(2, 2);
context.fillRect(0, 0, 100, 100);
```

在这个例子中，首先保存画布的状态，再将原点平移到(150, 150)。然后，将画布放大两倍，在位置(0, 0)绘制一个正方形。因为已经将2D渲染上下文平移到(150, 150)，所以这个正方形会被绘制在正确的位置，并同时放大两倍（参见图4-9。）

问题是，从现在开始绘制的其他图形都将平移150像素并在两个方向同时放大两倍。幸好，你已经完成了前面一半的工作：在执行变形之前保存了绘图状态。剩下一半工作是恢复之前保存的绘图状态。

```
context.restore();
context.fillRect(0, 0, 100, 100);
```

在恢复绘图状态之后，后面绘制的所有图形都不会出现变形效果（参见图4-10）。没错！我说过，保存和恢复绘图状态使你能够画出漂亮的图形。

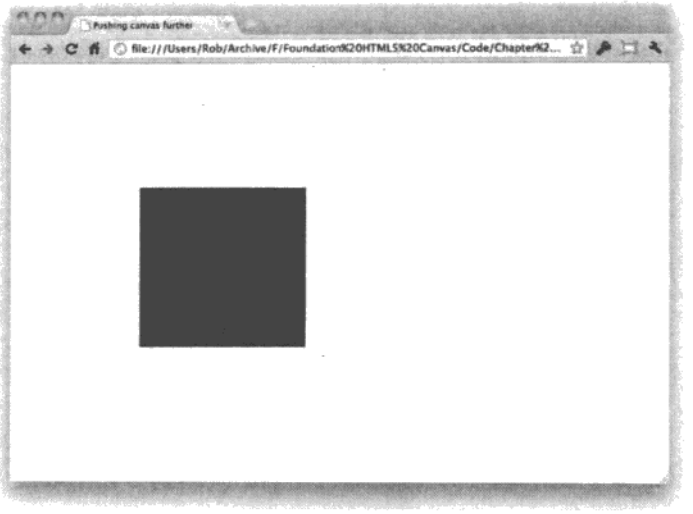


图4-9　保持原点同时进行缩放

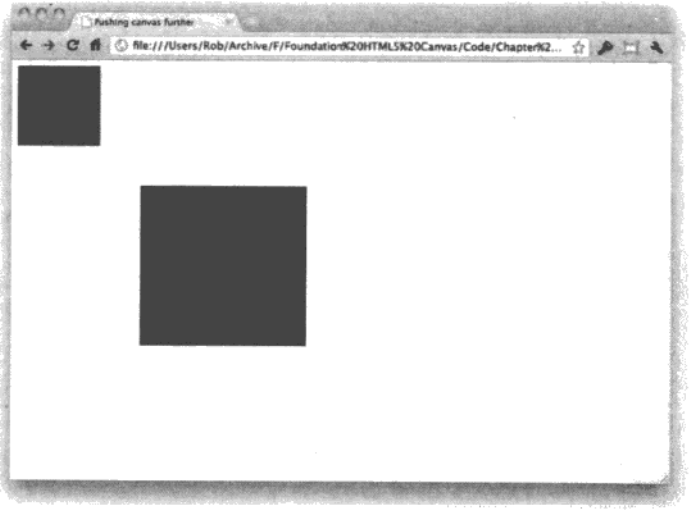


图4-10　在执行平移后恢复绘图状态

4.2.3　旋转

如果要我选择一个最喜欢的变形功能，我肯定会选择rotate方法。通过旋转角度来打破正

方形像素的概念，这一直令我惊奇，但可能只是我才有这样的想法而已。就是觉得很有意思！

到现在为止，我们介绍的变形方法的共同特点是它们都很容易调用。rotate方法也不例外，你只需要传入以弧度为单位的2D渲染上下文旋转角度值即可：

```
context.rotate(0.7854); // 旋转45度 (Math.PI/4)
context.fillRect(150, 150, 100, 100);
```

然而，这个旋转的结果可能并不是你所期望的。为什么正方形会旋转到浏览器边界以外呢？(参见图4-11)。

图4-11 旋转画布可能导致图形出现在一些奇怪的位置上

出现这种结果，是因为rotate方法是把2D渲染上下文绕其原点(0, 0)进行旋转的，在前面这个例子中，原点是屏幕的左上角。因此，你所绘制的正方形本身是不会旋转的，它现在实际上是以45度角绘制到画布中。图4-12可以帮助理解这一点。

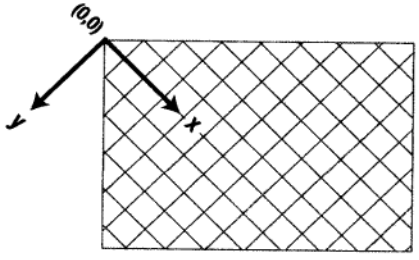


图4-12 旋转后的2D绘图上下文

当然，如果你只想旋转所要绘制的图形，那么这样肯定不行。这时，仍然还需要使用translate方法。要实现所期望的效果，需要将2D渲染上下文的原点平移到正在绘制的图形的中心。然后，再对画布执行一次旋转，接着在当前位置绘制图形。这个过程描述起来有些复杂，所以让我们用示例代码来演示这个过程：

```
context.translate(200, 200); // 平移到正方形中心
context.rotate(0.7854); // 旋转45度角
context.fillRect(-50, -50, 100, 100); // 以旋转点为中心绘制一个正方形
```

这样你会得到一个旋转45度角的有趣正方形，它正位于你想要的位置（参见图4-13）。

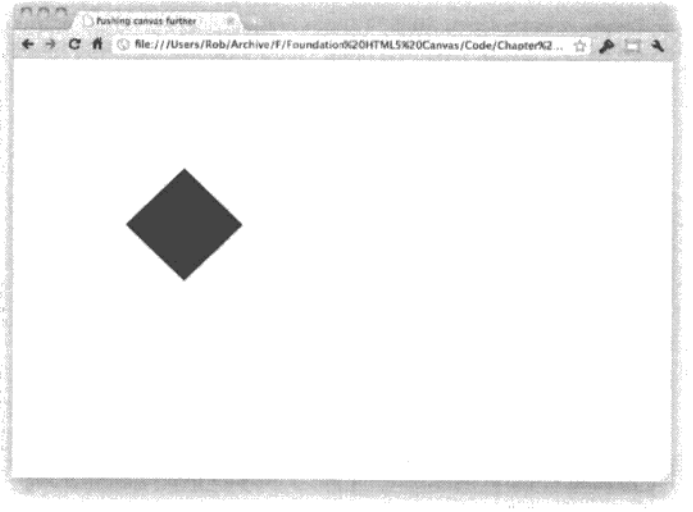


图4-13　以原点为中心旋转一个图形

注意　执行变形的顺序是极为重要的。例如，如果在执行平移之前将画布旋转45度，那么你会在45度角上进行平移。WHATWG规范中有一个例子指出，如果一个缩放变形操作先将你绘制的任何图形的宽度放大2倍，随后再旋转90度，那么当你绘制一个宽度是高度2倍的矩形时，这个缩放变形操作会把你所绘制的矩形变成一个正方形。仔细想想，你就能明白它的意思，它强调了考虑变形顺序的必要性。如果绘图时出现错误，那么请先检查顺序！

4.2.4　变换矩阵

现目前为止，你所使用的所有变形方法都会影响一个东西，那就是变换矩阵。我们不讨论一

些非必要的细节（这些细节信息并不重要），变换矩阵就是一组数字，它们各自描述一个稍后将会介绍的特定变形类型。矩阵分成多个列和行；在画布中，你使用的是一个3×3矩阵——3列和3行（参见图4-14）。

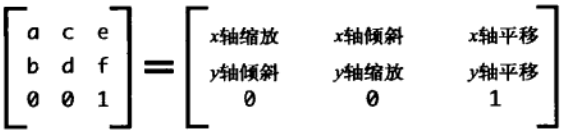


图4-14 2D渲染上下文的变换矩阵

你可以忽略最后一行，因为你不需要也不能修改它的值。最重要的是第一行和第二行，其中包含的数字值对应画布中使用的*a*至*f*。你可以看到，图4-14中每一个数字值都对应一种特定的变形。例如，*a*表示在*x*轴的缩放倍数，*f*表示在*y*轴的平移。

现在，在学习如何手动处理变换矩阵之前，我先说明一下这个矩阵的默认值。一个新的2D渲染上下文将包含一个全新的变换矩阵，即单位矩阵（identity matrix）（参见图4-15）。

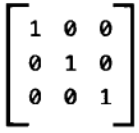


图4-15 单位矩阵

除了左上角至右下角的主对角线以外，这个特殊矩阵的每个值都设置为0。这样设置的唯一原因是它更适合进行计算，但是可以确定的是，单位矩阵表示完全未执行过变形。全面理解单位矩阵的含义并不是很重要，重要的是要知道变换矩阵中的默认值是什么。

操作变换矩阵

本节要介绍的最后两个方法是`transform`和`setTransform`。它们能够帮助我们操作2D渲染上下文的变换矩阵。我们已经了解了足够多的基本概念，所以现在让我们使用`transform`执行一个平移和缩放，然后再绘制一个正方形，以此说明它的作用：

```
context.transform(2, 0, 0, 2, 150, 150);
context.fillRect(0, 0, 100, 100);
```

`transform`方法有6个参数，分别对应变换矩阵的每一个值，第一个表示*a*，最后一个表示*f*。在这个例子中，你想将画布的尺寸放大2倍，所以将第1个和第4个参数设置为2，即*a*和*d*——分别对应*x*轴缩放和*y*轴缩放。可以理解。而如果要平移画布原点呢？没错：你需要设置第5个和第6个参数，即*e*和*f*——分别对应*x*轴平移和*y*轴平移（参见图4-16）。

希望你现在已经理解了它的使用方法，手动操作变换矩阵其实并不复杂。只要理解每一个值的意义，就能够执行正确的操作。现在让我们用变换矩阵执行一些更高级的变形——旋转！

不使用`rotate`方法执行旋转变形似乎有些复杂，但是如果你听我讲下去，很快就能明白这样做的意义：

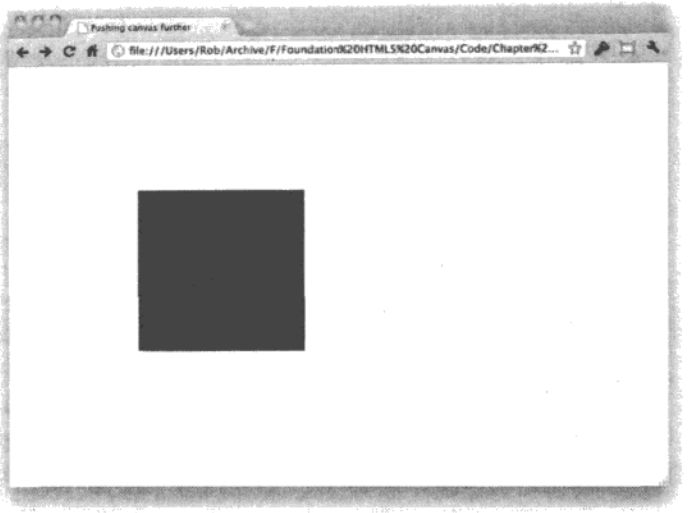


图4-16　使用变换矩阵进行缩放平移

```
context.setTransform(1, 0, 0, 1, 0, 0); // 单位矩阵
var xScale = Math.cos(0.7854);
var ySkew = -Math.sin(0.7854);
var xSkew = Math.sin(0.7854);
var yScale = Math.cos(0.7854);
var xTrans = 200;
var yTrans = 200;

context.transform(xScale, ySkew, xSkew, yScale, xTrans, yTrans);
context.fillRect(-50, -50, 100, 100);
```

首先，你需要调用setTransform方法。这是第二个操作变换矩阵的方法，它的作用是将矩阵重置为单位矩阵，然后按照6个参数执行变形。在这个例子中，使用它来重置变换矩阵，从而保证你操作的是一个原始状态的变换矩阵。然后，为一些变量赋值，它们是调用transform方法所使用的参数。有了这些作为参数的变量，就能够使整个过程变得更加简洁和清晰，而且更容易理解。

需要指出的是，transform方法实际上是将现有的变换矩阵乘以你所指定的值，而不是直接设置变换矩阵的值。这意味着其中会有一个累积效应。如果你多次调用transform，那么每一次变形都是应用到前一个变形所得到的变换矩阵。我承认这有一些复杂，但是我希望它能够帮助你理解变形的工作方式。

你可能注意到了，我们又一次使用到Math对象。在这个例子中，使用它来返回一些必要值，缩放和倾斜变形将使用这些值来生成旋转效果。因为掌握这一点非常重要，所以在此重复一遍：使用变换矩阵进行旋转是倾斜和缩放的组合效果。为此，你需要给三角函数cos（余弦）和sin（正弦）传入以弧度为单位的角度值。

> **注意** 我们还会在本书中使用正弦和余弦等三角函数。如果你希望学习更多关于它们的使用方法和作用，我强烈建议你找些书来复习一下这些函数。但是，我无法在这里逐一解释每一个概念。变换矩阵的维基百科页面包含更丰富的信息：http://en.wikipedia.org/wiki/Transformation_matrix。

最后，将所有代码编写出来，你会得到下面的结果 —— 一个漂亮的旋转后的正方形（参见图4-17）。

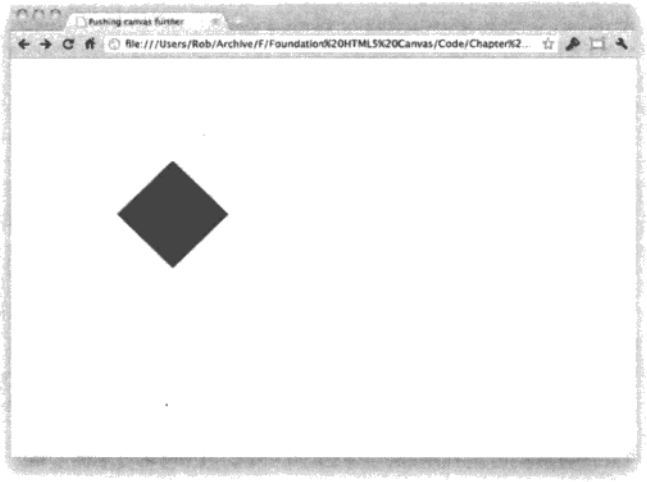


图4-17 使用变换矩阵进行旋转

效果不错，不是吗？想想吧，这些旋转效果完全是你赤手空拳做出来的，根本没用rotate方法！严格地说，这三个核心的变形方法在大多数时间都能够满足要求，但是即便不满足要求，理解变换矩阵也能够帮助你解决问题。

4.3 合成

上一节确实包括许多内容，祝贺你学完了这些知识。在这一节中，我们将学习一些较为轻松的内容——合成，它们不复杂，而且还很有趣。简而言之，组合就是将多个可视化元素组合成为一个可视化元素。它广泛应用于各行各业，从图形设计到好莱坞电影制作（绿色背景的场景是后期与另一个图像合成的）。

在画布中绘制的所有东西都是已经合成的，这意味着绘制的所有内容都会与已经绘制的现有元素合并在一起。这实际上都是基本合成，只是将一些内容叠加到另一些内容之上。我马上要介绍这些方面的合成，但是现在我们先了解一下画布中最简单的合成方法，即globalAlpha属性。

> **注意**　本节将介绍的两个全局合成属性都会影响到2D渲染上下文的绘图效果。一定要明确一点，那就是修改全局合成属性会影响到修改之后所绘制的全部内容。

4.3.1　全局阿尔法值

在画布上进行绘图之前，它会应用一个与globalAlpha属性相匹配的阿尔法值。赋给globalAlpha的值必须在0.0（全透明）与1.0（不透明）之间，默认值是1.0。简单地说，globalAlpha属性会影响将要绘制的对象的透明度。例如，可以按照以下方式绘制一个半透明的正方形：

```
context.fillStyle = "rgb(63, 169, 245)";
context.fillRect(50, 50, 100, 100);
context.globalAlpha = 0.5;
context.fillStyle = "rgb(255, 123, 172)";
context.fillRect(100, 100, 100, 100);
```

由于我们是在绘制了蓝色正方形后才设置globalAlpha属性的，所以只有粉红色正方形才会受到阿尔法值的影响。结果是后面蓝色正方形的一小块稍稍透过前面的粉红色正方形显示出来（参见图4-18）。

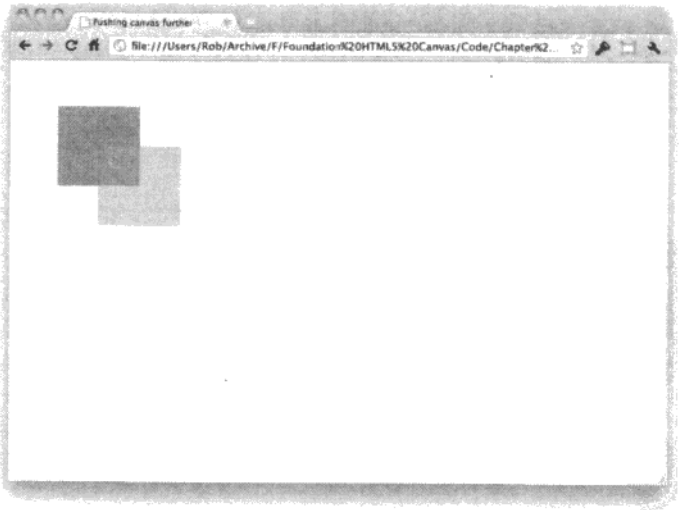


图4-18　设置globalAlpha属性

现在，通过给fillStyle设置一个包含小于1的阿尔法值的rgba值，也可以得到相同的效果。不同之处是，globalAlpha设置的是全局阿尔法值，这个值会在后续应用rgba颜色值等阿尔法值时被参照。例如，如果globalAlpha为0.5，你又应用了一次fillstyle（它带有一个阿尔法值

为0.5的rgba），那么结果的阿尔法值实际上就是0.25。2D渲染上下文的全局阿尔法值（0.5）充当了计算其他阿尔法值的基数（0.5 * 0.5 = 0.25）。

4.3.2 合成操作

正如本节开头介绍的，即使全新的2D渲染上下文也会在一开始就使用合成。你可能没有注意到这一点，因为此时使用的合成方法能得到你预期的结果：一个图形叠加到另一图形之上。这种合成称为源覆盖于目标之上（source over destination），源是绘制的新图形，而目标则是可能已经绘制了图形的2D渲染上下文。我们知道，这是因为2D渲染上下文的globalCompositeOperation属性的默认值是source-over，并且这个属性定义了对2D渲染上下文上所有绘制图形执行的合成类型（11种可选方法之一）。必须指出的是，根据赋值顺序的不同，globalCompositeOperation的所有值可能会涉及源或目标的其中一个（取决于顺序），而不会同时涉及两者。例如，“source-over”是（源覆盖于目标之上）的简称；目标是隐含的，因为它不需要在值中指定（源必须绘制在某些东西之上）。

让我们先了解一下globalCompositeOperation支持的11种选择。使用下面的代码作为模板，你可以学习每一种合成操作。其中蓝色正方形是目标，而粉红色正方形是源。我们只使用蓝色和粉红色而不使用其他颜色的原因是它们能够更好地显示合成操作的效果：

```
context.fillStyle = "rgb(63, 169, 245)";
context.fillRect(50, 50, 100, 100);
context.globalCompositeOperation = "source-over";
context.fillStyle = "rgb(255, 123, 172)";
context.fillRect(100, 100, 100, 100);
```

> **注意** 有一些浏览器不支持全部的globalCompositeOperation值，所以为了便于讲述，我将使用插图来显示它们的运行结果，而不使用浏览器中的截图。我将使用WHATWG规范作为每一个操作的显示效果的依据。

1. source-over

这是默认值，它表示绘制的图形（源）将画在现有画布（目标）之上：

```
context.globalCompositeOperation = "source-over";
```

效果与目前学习到的绘图效果是完全相同的（参见4-19）。

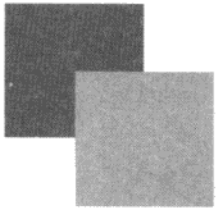


图4-19 source-over合成操作

2. destination-over

这个操作的值与前一个值相反，所以现在目标绘制在源之上：

```
context.globalCompositeOperation = "destination-over";
```

效果与前一个操作恰好相反（参见图4-20）。

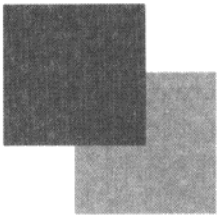


图4-20　destination-over合成操作

3. source-atop

这个操作会将源绘制在目标之上，但是在重叠区域上两者都是不透明的。绘制在其他位置的目标是不透明的，但源是透明的（参见图4-21）。

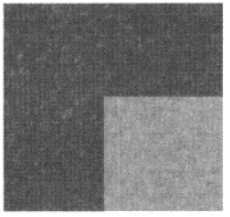


图4-21　source-atop合成操作

4. destination-atop

这个操作与source-atop相反，目标绘制在源之上，其中在重叠区域上两者都是不透明的，但绘制在其他位置的源是不透明的，而目标变成透明的（参见图4-22）。

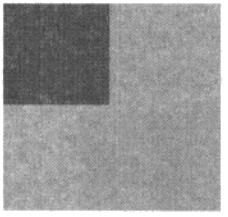


图4-22　destination-atop合成操作

5. source-in

在源与目标重叠的区域只绘制源。而不重叠的部分都变成透明的（见图4-23）。

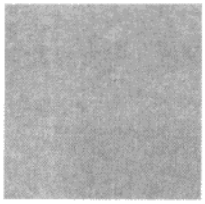


图4-23 source-in合成操作

6. destination-in

这个操作与source-in相反，在源与目标重叠的区域保留目标。而不重叠的部分都变成透明的（参见图4-24）。

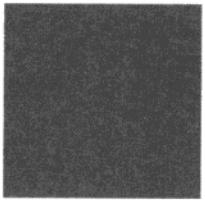


图4-24 destination-in合成操作

7. source-out

在与目标不重叠的区域上绘制源。其他部分都变成透明的（参见图4-25）。

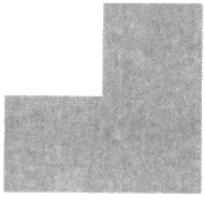


图4-25 source-out合成操作

8. destination-out

在与源不重叠的区域上保留目标。其他部分都变成透明的（参见4-26）。

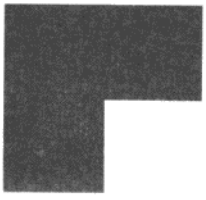


图4-26 destination-out合成操作

9. `lighter`

这个值与顺序无关，如果源与目标重叠，就将两者的颜色值相加。得到的颜色值的最大取值为255，结果就是白色（参见图4-27）。

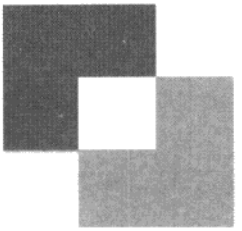


图4-27　lighter合成操作

10. `copy`

这个值与顺序无关，只绘制源，覆盖掉目标（参见图4-28）。

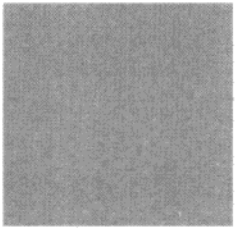


图4-28　copy合成操作

11. `xor`（异或）

这个值与顺序无关，只绘制出不重叠的源与目标区域。所有重叠的部分都变成透明的（参见图4-29）。

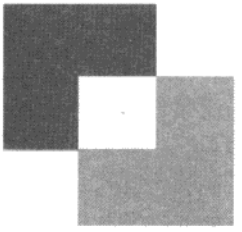


图4-29　xor合成操作

总之，这些合成操作使你能够在需要绘制一些复杂图形的情况下实现一些有趣的效果。有一些操作（如“destination-out”）在擦除画布上一些非矩形区域时是很有用的；例如，使用圆作为源。

4.4 阴影

所有人都喜欢好看的阴影效果，它们可能是Adobe Photoshop中使用最广泛的效果了，并且也经常在Web和图形设计中使用。如果操作正确，它们实际上确实能够增加图像真实感。然而，如果操作不当，它们也可能完全毁掉一个图像。在此，我并非要讲述个人对图形设计的看法，而是想教会你如何掌握画布的使用方法。

在画布中创建阴影效果是相对较简单的，它可以通过4个全局属性进行控制。这些属性是shadowBlur、shadowOffsetX、shadowOffsetY和shadowColor。我们马上开始逐一讲解这些属性。默认情况下，2D渲染上下文是不会绘制阴影效果的，因为shadowBlur、shadowOffsetX和shadowOffsetY都设置为0，而shadowColor则设置为透明黑色。创建阴影效果的唯一方法是将shadowColor修改为不透明值，同时将shadowBlur、shadowOffsetX或shadowOffsetY都设置为非0值：

```
context.shadowBlur = 20;
context.shadowColor = "rgb(0, 0, 0)";
context.fillRect(50, 50, 100, 100);
```

在这个例子中，给阴影设置了20像素的模糊值，并将它的颜色设置为完全不透明的黑色。阴影的偏移值在*x*轴和*y*轴方向仍然保持为默认值0。需要特别指出的是，即使使用了不透明的黑色，但由于采用了模糊效果，这个阴影在边界上仍然有些透明效果（参见图4-30）。

图4-30 基本阴影效果

修改shadowBlur、shadowOffsetX或shadowOffsetY属性，就能够创建不同的阴影效果：

```
context.shadowBlur = 0;
context.shadowOffsetX = 10;
context.shadowOffsetY = 10;
context.shadowColor = "rgba(100, 100, 100, 0.5)"; // 透明灰色
context.fillRect(200, 50, 100, 100);
```

将模糊修改为0，创建清晰的阴影效果，而稍微向右下偏移，就得到一个不同的阴影效果。使用4.3节中提到过的rgba颜色值将shadowColor设置为透明浅灰色，就能够实现更炫的效果（参见图4-31）。

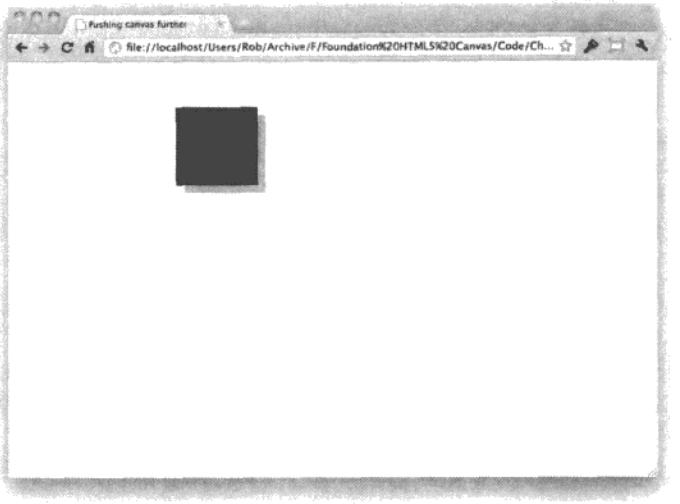


图4-31　创建不同的阴影效果

画布的阴影支持所有图形，所以完全可以在所绘制的圆形或其他图形上创建阴影效果。甚至可以将颜色修改为任意奇特的值，尽管我肯定会置疑你的动机：

```
context.shadowColor = "rgb(255, 0, 0)"; //红色
context.shadowBlur = 50;
context.shadowOffsetX = 0;
context.shadowOffsetY = 0;
context.beginPath();
context.arc(400, 100, 50, 0, Math.PI*2, false);
context.closePath();
context.fill();
```

这段代码会得到一个非常漂亮的圆形，它后面有一个鲜红色阴影效果（参见图4-32）。

通过组合使用各种模糊和颜色值，我们就能够实现一些与阴影完全无关的效果。例如，使用模糊黄色阴影在一个对象周围创建出光照效果，如太阳或发光体。

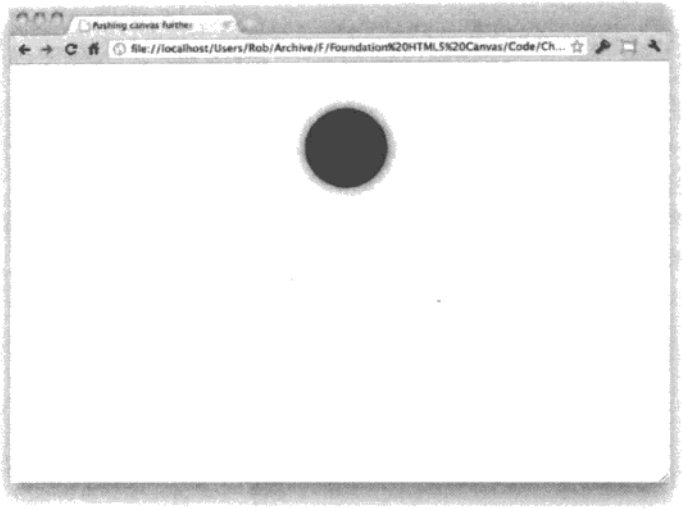


图4-32 在圆形上绘制阴影

4.5 渐变

有时候，简单的颜色并不够用，或者你确实需要给图形颜色增加额外的真实元素。无论是何种情况，画布的渐变颜色肯定都是一种可以考虑的方法，fillStyle和strokeStyle都可以接受以CanvasGradient对象表示的渐变颜色值。

画布支持两种类型的渐变：线性渐变和放射渐变。每一种渐变在2D渲染上下文中都有对应的创建方法，线性渐变对应的是createLinearGradient，而放射渐变对应的是createRadialGradient（如果你没看就猜到了，那奖励你5分）。这两种方法都返回一个CanvasGradient对象，可以使用CanvasGradient对象本身的addColorStop方法对它进行进一步处理。下面让我们创建一个基本的线性渐变，以了解它们的使用方法。

```
var gradient = context.createLinearGradient(0, 0, 0, canvas.height());
gradient.addColorStop(0, "rgb(0, 0, 0)");
gradient.addColorStop(1, "rgb(255, 255, 255)");

context.fillStyle = gradient;
context.fillRect(0, 0, canvas.width(), canvas.height());
```

这里，我们首先使用createLinearGradient创建一个新的CanvasGradient对象，然后将它赋值给一个变量，这样在将来才能够再次访问这个对象。createLinearGradient方法有4个参数：渐变起点的(*x*, *y*)坐标，渐变终点的(*x*, *y*)坐标。起点和终点描述了所绘制渐变效果的长度、位置和方向。在这个例子中，渐变是从画布的左上角开始的，然后延伸到左下角。线性渐变的绘

制方向与起点和终点所确定的直线是正交的，所以在这个例子中，渐变是从顶部延伸到底部的。

定义一个CanvasGradient对象还不够，我们还需要给它指定一种颜色。因此，我们两次调用CanvasGradient对象（此对象已经保存到一个变量中了）的addColorStop方法。addColorStop方法有两个参数：颜色的偏移值（0表示渐变起点，1表示终点），以及该偏移的颜色值。与fillStyle相同，这个颜色值可以是任何CSS颜色值。在这个例子中，渐变是从起点的黑色（偏移值为0）变化到终点的白色（偏移值为1）。

最后，这个渐变会作为一个颜色值被应用到fillStyle属性中，从而在整个画布上显示从黑色到白色的渐变效果（参见图4-33）。

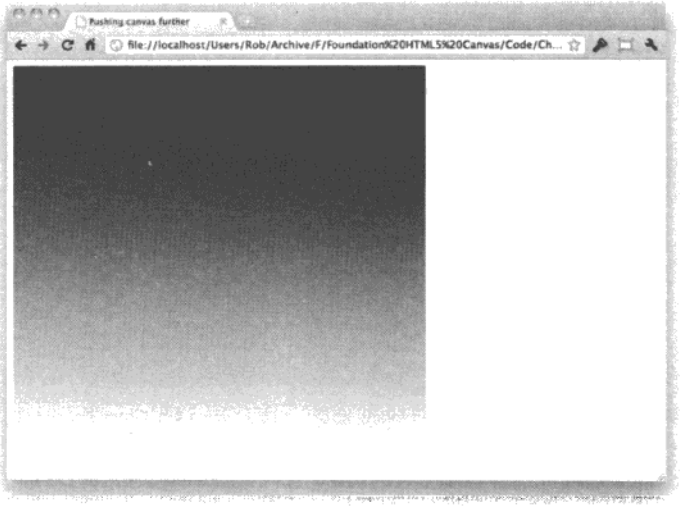


图4-33　绘制一个线性渐变

创建放射渐变的过程则有些不同。事实上，如果不能真正了解放射渐变的工作方式，那么你就会觉得放射渐变其实是很混乱的，特别是它的效果在不同浏览器上是不相同的（目前的情况）。幸好，现在我们只是分析它的创建过程，从而使你掌握它的使用方法，所以让我们马上开始吧。

放射渐变是使用createRadialGradient方法创建的。这个方法需要6个参数——前3个参数描述一个圆（开始圆），后3个参数描述另一个圆（结束圆）。这两个圆本身不仅描述了方向及渐变的起止位置，而且还描述了渐变的形状。用于描述每一个圆的3个参数是圆心坐标(*x*, *y*)和半径。这些参数可以用字符串描述为：

```
createRadialGradient(x0, y0, r0, x1, y1, r1);
```

实际的渐变效果是连接两个圆周的锥体，其中开始圆之前的锥体部分显示偏移值为0的颜色，而结束圆之后的锥体部分则显示偏移值为1的颜色。我承认，理解这个概念有一些难度，所以我用一个图片来说明，希望你能够理解它的原理（参见图4-34）。

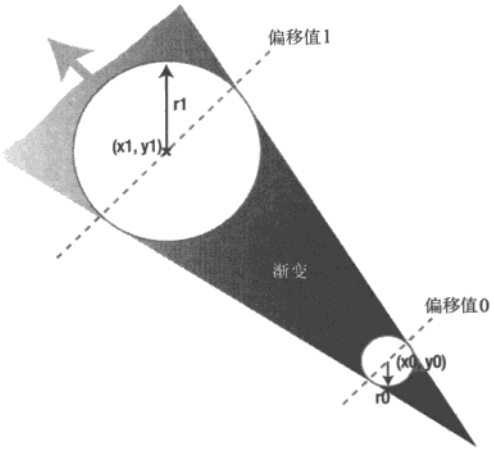


图4-34 画布中放射渐变工作原理的图形表示

使用画布代码编写的结果如下：

```
var gradient = context.createRadialGradient(300, 300, 10, 100, 100, 50);
gradient.addColorStop(0, "rgb(0, 0, 0)");
gradient.addColorStop(1, "rgb(150, 150, 150)");

context.fillStyle = gradient;
context.fillRect(0, 0, canvas.width(), canvas.height());
```

开始圆位于坐标位置(300, 300)，半径为10；结束圆的坐标位置为(100, 100)，半径为50。最终得到的锥体基本上与图4-34相似，开始圆为黑色（偏移值0），慢慢褪色为结束圆的灰色（偏移值1）（参见图4-35）。

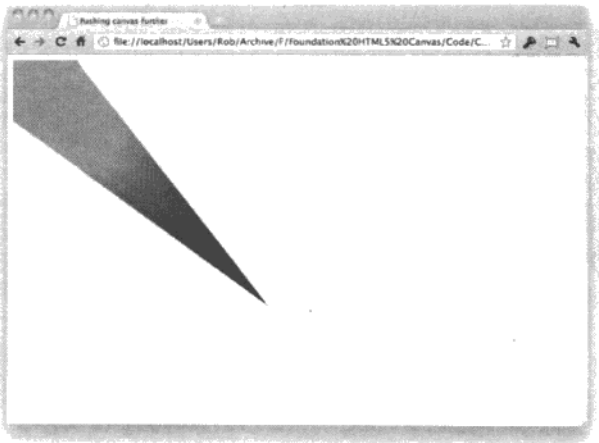


图4-35 绘制一个放射渐变锥体

现在我可以坦白地说，这并不是我想象的放射渐变效果；我想象的效果应该类似于图4-36。

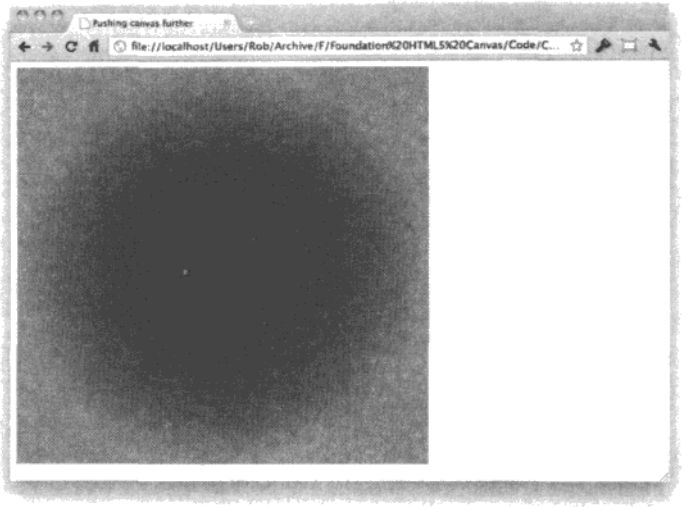


图4-36　绘制一个更常见的放射渐变

这种放射渐变其实非常容易实现，只需要将开始圆和结束圆放置在同一个位置即可。仅此而已！

```
var canvasCentreX = canvas.width()/2;
var canvasCentreY = canvas.height()/2;

var gradient = context.createRadialGradient(canvasCentreX, canvasCentreY, 0,
canvasCentreX, canvasCentreY, 250);
gradient.addColorStop(0, "rgb(0, 0, 0)");
gradient.addColorStop(1, "rgb(150, 150, 150)");

context.fillStyle = gradient;
context.fillRect(0, 0, canvas.width(), canvas.height());
```

通过将两个圆叠放在一起，我们就可以将渐变锥体环绕360度，只要两个圆的大小不同，那么渐变效果就会从较小的圆延伸到较大的圆上。

> **注意**　虽然渐变的效果很漂亮，但是使用cavans来创建渐变并非总是最好的方法，特别是在将它们作为背景使用时。我们应该考虑使用其他的方法，如专门用来完成这个任务的CSS3渐变背景。

4.6　复杂路径

在上一章中，通过学习了如何绘制直线和弧线（为了创建圆），也对路径进行了概括介绍。

画布中的路径不仅限于线条和圆，实际上还可以用它们来创建各种奇妙的图形。

单独一条直线路径可以绘制成一条漂亮的线条。在第3章中，我们还未涉及的是如何一次绘制多个线条，以及将它们组合成一个图形。事实上，我们已经知道实现这个效果的代码了，所以下面将直接介绍如何将多个路径连接在一起。

```
context.beginPath();
context.moveTo(100, 50);
context.lineTo(150, 150);
context.lineTo(50, 150);
context.closePath();
context.stroke();
context.fill();
```

你应该能够读懂所有这些代码——它先开始一条路径，将原点移到当前路径，从当前路径原点绘制一条线到一个指定点，再绘制一条线到另一个点，然后再继续。那么我们在这里做了什么呢？我们刚刚做的就是将多个路径连接在一起，我们只需要不停地画线。这确实非常简单；每次调用moveTo或lineTo都会给所谓子路径(sub path)增加一个相应的(x, y)坐标值。事实上，moveTo会创建一条全新的子路径，而lineTo只是沿着一条已有的子路径继续画线。这条子路径会记录我们所添加的最后一个坐标值，因而可以连续多次调用lineTo方法。lineTo的每次调用都是从子路径的最后一个坐标值（由moveTo或lineTo调用留下）开始画线，绘制一条线连接lineTo参数所定义的坐标值，然后再将子路径更新到新的坐标值。

绘制三角形的最后一步是调用closePath方法，它会画一条线连接子路径的最后一个点和第一个点——封闭路径。它也会将起点和终点添加到子路径上，这两个点现在具有相同的坐标值(参见图4-37)。

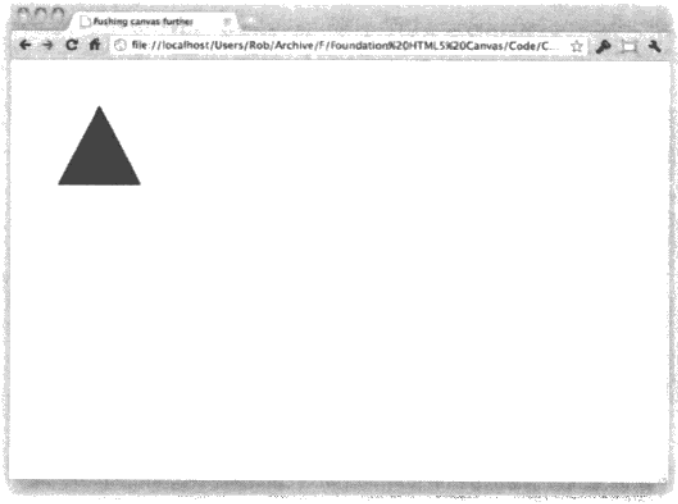


图4-37 通过连接路径绘制一个三角形

贝塞尔曲线

要在画布中绘制一条曲线，我们可以使用arc方法或arcTo方法（在两点间绘制一条弧线），但是这些弧线只是一条具有相同半径的曲线。要创建一条更复杂的曲线，需要使用贝塞尔曲线（Bézier curve）方法：quadraticCurveTo或bezierCurveTo。

> **注意**　不要被贝塞尔曲线的名称误导，它们都是贝塞尔曲线，即使其中一个方法的名称不包含Bézier字样。实际上，quadraticCurveTo是一种二次贝塞尔曲线，而bezierCurveTo是三次贝塞尔曲线。现在，我相信你对此应该不会再感到意外了。

这两种贝塞尔曲线都是通过控制点将一条直线变成曲线。二次贝塞尔曲线只有一个控制点，这意味着线条中只有一次弯曲；而三次贝塞尔曲线则有两个控制点，这意味着一条线中会有两次弯曲。通过图4-38，我们可以直观地了解这两种曲线的效果，左边是二次贝塞尔曲线，右边是三次贝塞尔曲线。

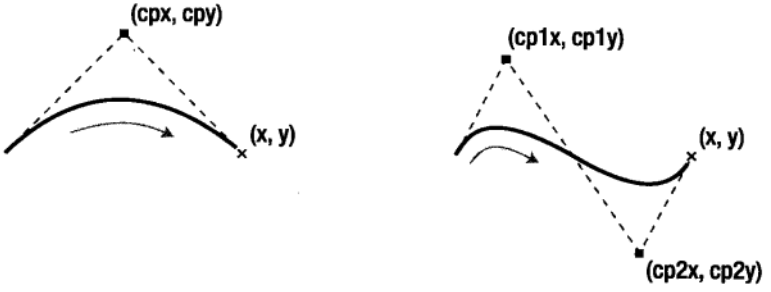


图4-38　贝塞尔曲线的构造

要在画布中创建这两种曲线，只需要直接调用quadraticCurveTo或bezierCurveTo。让我们先尝试使用quadraticCurveTo：

```
context.lineWidth = 5;
context.beginPath();
context.moveTo(50, 250);
context.quadraticCurveTo(250, 100, 450, 250);
context.stroke();
```

除了quadraticCurveTo方法，这段代码的其他语句你都理解。这个方法有4个参数：控制点的(*x*, *y*)坐标值（图4-38中的cpx和cpy），以及路径目标点的(*x*, *y*)坐标。控制点在水平（*x*）方向上位于线条的中心，在垂直（*y*）方向上稍微偏上（如图4-38所示），图4-39所示的就是这条漂亮的曲线。

然后，创建三次贝塞尔曲线也很简单：

```
context.lineWidth = 5;
context.beginPath();
```

```
context.moveTo(50, 250);
context.bezierCurveTo(150, 50, 350, 450, 450, 250);
context.stroke();
```

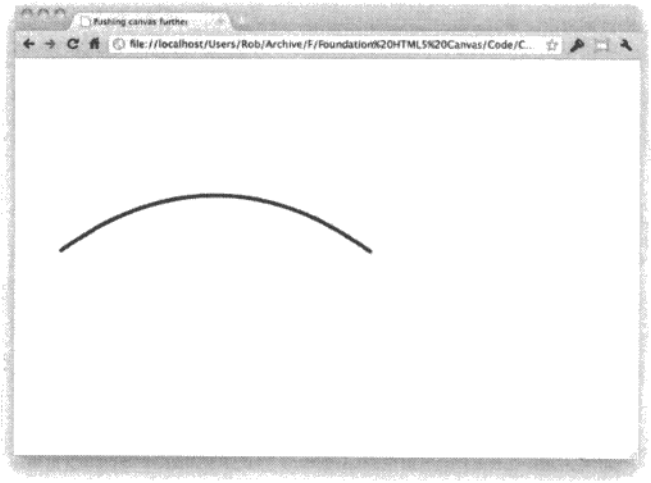


图4-39 二次贝塞尔曲线

bezierCurveTo函数有6个参数：第一个控制点的(*x*, *y*)坐标值（如图4-38中的cp1x和cp1y），第二个控制点的(*x*, *y*)坐标值（cp2x和cp2y），以及路径目标点的(*x*, *y*)坐标。两个控制点都位于如图4-38所示的位置，屏幕上显示了两次弯曲的曲线（参见图4-40）。效果很不错，不是吗？

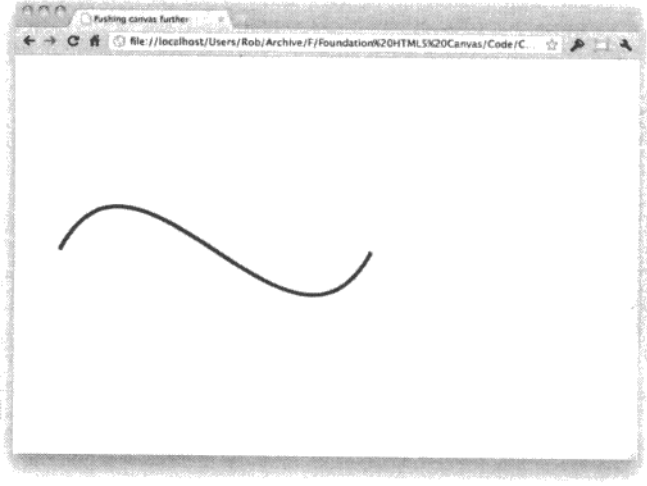


图4-40 三次贝塞尔曲线

现在绘制的贝塞尔曲线效果已经很不错了，但是在实践中它们一般不单独使用。贝塞尔曲线的最大用处是组合及附加到其他路径上，从而创建一些非常复杂的图形。你不再受限于只能创建直线和简单弧线的路径了！

> **注意**　如果你现在还不理解实际上应该如何处理复杂图形的所有坐标位置，不要担心。Adobe Illustrator有一个名为Ai->Canvas的插件（http://visitmix.com/labs/ai2canvas/），它允许你将绘制好的矢量图转换成画布代码，这意味着你完全不需要计算这些坐标。

4.7　将画布导出为图像

到目前为止，我们在画布中绘制的图形都仅限于在画布中使用，而无法在其他位置使用。这实际上是不理想的，特别是如果你希望导出漂亮的画布绘图作品，并将他保存在其他位置。先别着急！其实画布还有一个很有用的toDataURL方法。这个简单的方法能够将画布绘图转换为一个数据URL，我们可以通过它在浏览器上显示一个图像。这是非常棒的方法！

> **注意**　Mozilla Firefox浏览器本身支持右键单击canvas元素，然后将它另存为图像。它所使用的方法与在代码中使用toDataURL是完全相同的。

这个方法实际上是很简单的，所以让我们马上尝试使用这个方法来导出一个基本图形。我将在后面详细讲述它的工作原理。

```
context.save();
context.fillRect(50, 50, 100, 100);

context.fillStyle = "rgb(255, 0, 0)";
context.fillRect(100, 100, 100, 100);

context.restore();
context.fillRect(150, 150, 100, 100);

var dataURL = canvas.get(0).toDataURL();
```

这段代码将绘制一系列相互叠加的正方形，然后将图像数据URL赋值给dataURL变量。你会看到这三个正方形在浏览器的显示效果（参见图4-41），但是现在它仍然是前面介绍的画布图像，而不是导出的图像。下面马上介绍如何显示导出的图像。

这个例子中最关键的部分是dataURL变量，下面是你刚刚存储到此变量中的值的一个片段：

```

QQ4dNxYDwPkD3//KPZMDRItA0ovI8tINS81iG8T/fd/3H38IECBAgACBtwX++/bre3sCBAgQIEDgLwGD7jsgQI
AAAQIBAgY9oEQRCBAgQICAQfcNECBAgACBAAGDHlCiCAQIECBAwKD7BggQIECAQQ8oUQQCBAgQIGPfQME
CBAgQCBAwKAHlCgCAQIECBAw6L4BAgQIECAQQA0oUgQABAgQIGHTfAAECBAgQCBAw6AElikCAAAECBAy6b4
AAAQIECAQIGPSAEkUgQIAAAQIG3TdAgAABAgQCBAx6QIkiECBAgAABg+4bIECAAAECAQIGPaBEEQgQIECAgEH3
DRAgQIAAgQABgx5QoggECBAgQMCg...
```

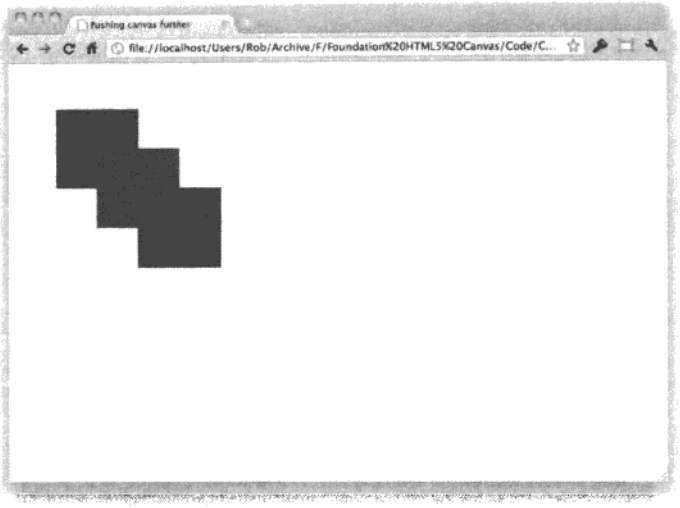


图4-41 准备将要导出的画布

实际的输出比这些要长得多，但是实际上只有前面4个单词是我们现在关心的。前面3个单词是data:image/png；它们表示后续内容是一个PNG格式的图像的数据URL。第4个单词是base64，它表示数据采用base64编码格式。这种格式经常用于向使用文本数据的系统传输二进制数据（例如图像）。实际上，在base64后面的所有数字、字母和符号是以文本表示的画布图像。

> **注意** Canvas规范支持使用toDataURL方法导出其他类型的图像。然而，PNG支持是默认的要求，而各个浏览器制造商可以自行决定是否支持其他格式的图像。

这是一个不寻常的做法，但是如果复制dataURL变量中的字符串，然后粘贴到现代浏览器的地址栏（只要不超过输入URL长度限制），那么你就会看到在画布中绘制的图像。然后，在需要时，可以右键单击图像，将它保存到桌面。或者，你可以在例子中用生成的图像替换canvas元素：

```
var dataURL = canvas.get(0).toDataURL();
var img = $("<img></img>");
img.attr("src", dataURL);

canvas.replaceWith(img);
```

这段代码使用jQuery创建了一个全新的HTML img元素，然后将图像数据赋值给它的src属性。最后，使用jQuery的replaceWith方法将canvas元素替换成刚刚创建的img元素。这样，我们就得到一个图像，其内容与画布上绘制的内容完全相同。你可以通过右键单击图像，然后查看是否有“图片另存为...”或类似选项，从而确定它是否真是一个图像（参见图4-42）。

> **注意**　需要指出的是，base64数据比它表示的原始二进制图像数据大50%。如果你处理的是少量较小的图像，那么这就不是问题，但是如果你处理的是大图像和大量图像，它就会有一些问题。

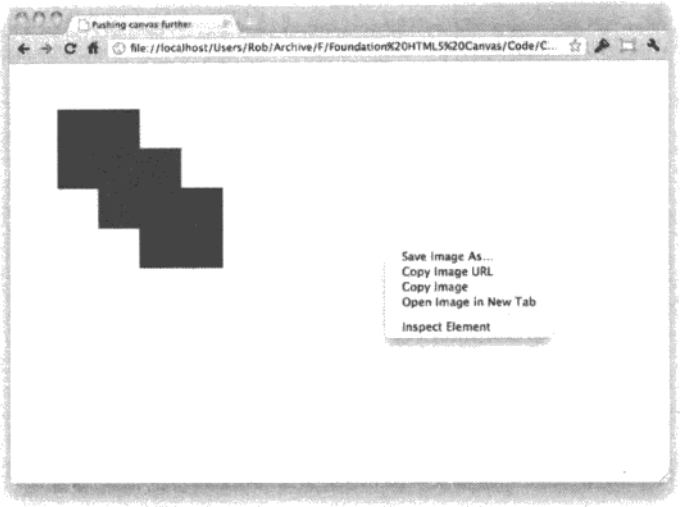


图4-42　将保存的画布图像数据保存为一个HTML图像元素

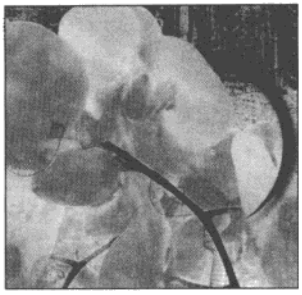
实际上，你可以自由决定如何使用这些图像数据，但是首先需要知道如何将画布导出为图像。你甚至可以使用画布随意绘制一个图像，然后导出图像，使用它作为CSS背景。

4.8　小结

本章内容很多，但是我希望你的学习之旅充满了快乐。你已经学习了绘图状态，以及如何使用它来节省大量开发时间。你还学习了变形和合成，通过这些，你可以创建一些非常复杂的图形。另外，你还学习了如何使用阴影和渐变来提高绘图作品的逼真度。然后，学习了如何超越简单的图形，使用高级路径创建一些更复杂的图形。本章最后介绍了将画布导出为图像的方法，从而可以将图像保存到计算机中或者在浏览器的其他元素中使用。

第5章

处理图像和视频



本章将介绍在Canvas中使用图像的知识，包括加载图像和处理图像中的单个像素。Canvas的这个功能可以用来创建一些炫丽的效果。本章还将教会你一般图像处理的知识。

本章最后专门介绍如何使用Canvas处理HTML5视频——为视频添加一些动画特效。这是非常令人兴奋的功能！

5.1 加载图像

上一章讲述了如何将画布导出为图像，将它保存到本地和与他人共享。现在，我们将学习如何实现完全相反的操作：将图像加载到画布中。介绍这个功能的主要原因是，它使我们能够用2D渲染上下文方法对原本不是在画布中创建的图像进行处理。我们还可以使用几种特殊的像素处理方法，对图像执行一些有趣的特殊操作，这将在下一节介绍。

> **注意** 在画布中进行像素处理实际上并不要求真加载图像，如照片。相反，画布本身就是作为图像进行处理的，这意味着你在上面绘制的所有内容都可以使用本章介绍的方法进行处理。

将图像加载到画布中实际上与绘制图像一样简单——只涉及一个方法。在调用drawImage方法时，至少需要三个参数：所绘制的图像和图像绘制位置的(*x*, *y*)坐标。这个方法的完整形式是：

```
context.drawImage(image, x, y);
```

参数image可以是HTML img元素、HTML5 canvas元素或HTML5 video元素。不仅局限于图像让画布的前景更加光明。在本章最后，我们还将学习如何在画布中使用和操作HTML5 video元素。

> **注意** 实际上，drawImage方法有两种调用方式，这两种方式所使用的参数个数是不同的。我们将在下一节详细介绍这两种方式。

首先，让我们使用与HTML文件位于相同目录的一个图像（如图5-1所示），将一个HTML img元素绘制到画布中。

```
var image = new Image();
image.src = "example.jpg";
$(image).load(function() {
        context.drawImage(image, 0, 0);
});
```

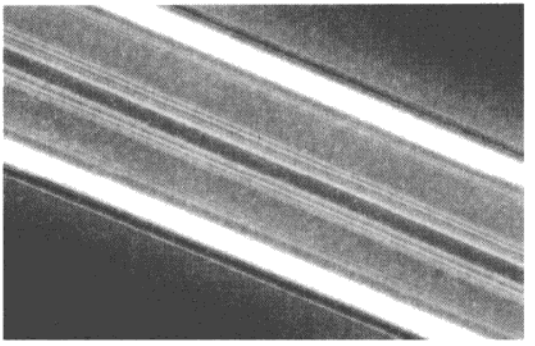


图5-1　本章所使用的图像示例

这里所做的第一件事是使用Image类为HTML img元素赋一个空的DOM对象。然后，通过把它的src属性设置为一个有效的图像文件路径，就可以将该图像加载到图像元素中，这就好像是设置了HTML img元素的src属性。实际上这创建了一个普通的HTML img元素，但是并没有将它显示在浏览器上。如果只希望给画布传递一个图像，而实际上不将它添加到HTML代码中，那么就可以使用这种方法。如果你就是想要看到这个HTML图像，那么完全可以跳过这些步骤，将image变量的值赋给现有HTML img元素的DOM对象。

> **注意**　在这个例子中，我们使用的是本地存储的图像文件，但是只要愿意，你也可以轻松地加载其他网站的图像。然而，使用外部图像有一些限制，所以在阅读5.4节的内容之前，我强烈建议你先不要这样做。现在，你只需要知道在使用外部图像时，画布会限制一些特定的功能就可以了。

无论使用哪一种方法，现在我们都应该能够访问图像的DOM对象了。最后要做的一件事是将这个图像对象传递给2D渲染上下文的drawImage方法，但是在这之前，我们需要确认这个图像已经完全加载。为此，可以使用jQuery的load方法，它是在一个元素完全加载后触发load事件时调用的方法。你可能还记得，第2章讨论了各种在页面加载后再加载JavaScript的方法。其中一种方法就是在window DOM对象上调用load事件，而这里使用的事件正是同一个事件。前面没有采用这种方法而现在却采用它的原因是，load事件只有当所有内容（包括图像）完成加载之后才会触发。直接将jQuery的load方法赋给图像对象，就可以保证我们只需要等待这个图像的

加载完成，而不需要等待其他内容的加载。

现在，我们知道这个图像在什么时候完成加载；我们将drawImage方法置于load事件被触发之后运行的回调事件中。drawImage方法的参数就是刚刚创建的图像对象，以及绘制图像的原点(*x*, *y*)坐标值。

如果一切正常，我们就能够将图像绘制到画布上，尽管图像被剪掉一部分（参见图5-2）。然而，不需要担心，因为剪掉的原因是画布小于所绘制的图像尺寸，而图像是以完整尺寸绘制的。在这个例子中，画布的宽度和高度都是500像素，而示例图像宽度为1024像素，高度为683像素。

图5-2 将完整尺寸的图像绘制到画布上

正如你所见，将图像加载到画布上并不难。然而，无法看到另一半图像很让人失望，所以让我们看看如何使它适合画布的尺寸。

5.2 调整和裁剪图像

我们现在知道调用drawImage方法的第一种方式，即将完整尺寸的图像绘制到画布上，但超过画布边界的部分被剪掉了。为了解决这个问题，需要调整图像大小或者控制图像的裁剪。通过drawImage方法的最后两种调用方式都能够完成这两个任务（也许这并不令人感到意外）；第一种调用可以调整图像大小，第二种可以同时调整和裁剪图像。drawImage的所有调用方式的唯一区别是所使用参数的个数和类型不同。

5.2.1 调整图像大小

实际上，调整图像大小与绘制完整尺寸的图像一样简单，只需要传入希望绘制的图像宽度和高度。用代码来表示，带有调整大小的参数的drawImage方法如下所示：

```
context.drawImage(image, x, y, width, height);
```

的确非常简单。

将前一个例子的drawImage方法修改为以下形式，图像就能够被调整为在画布中完全显示（参见图5-3）：

```
context.drawImage(image, 0, 0, 500, 333);
```

其中，宽度为500像素，与画布的宽度相等。而333像素的高度是按照原始图像的高宽比（高度与宽度的比例）计算得来的。要计算这个高宽比，只需要用高度除以宽度，对于原始图像（宽1024像素，高683像素），计算得到的高宽比为0.666992188（683÷1024）。然后，用宽度乘以这个比例就可以计算出调整后的图像高度。例如，假设调整后的宽度为500像素，那么高度就是500×0.666992188，结果为333。如果已知高度，可以计算出调整后图像的宽度。这时，只需要用调整后的高度除以高宽比即可。例如，通过计算可以得到这个例子的调整后宽度为500（333÷0.666992188）。

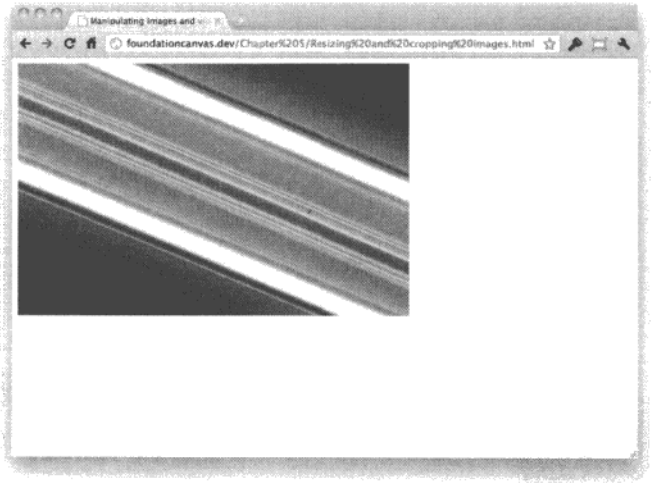


图5-3　调整图像使之在画布中完全显示

如果要绘制完整的图像，那么调整大小是很有用的，但是有时候我们需要进一步控制图像绘制的部分，那么它就缺少足够的支持了。这时，我们需要使用裁剪功能。

5.2.2　裁剪图像

裁剪的目的是将图像剪切为较小尺寸，这通常是因为我们只需要使用被裁剪对象的一部分。这是在摄影中常用到的一种技术，目的是突出照片的某个特定区域。裁剪画布所采取的方法与流行的照片编辑应用程序（如Adobe Photoshop）是完全相同的：划定一个希望保留的矩形区域，然后将矩形以外的全部内容删除。

裁剪是drawImage方法的最后一种用法，它总共有9个参数：源图像、源图像的裁剪区原点坐标(*x*,*y*)、源图像的裁剪区宽度和高度、在画布（目标）上绘制图像的原点坐标(*x*,*y*)及在画布上绘制图像的宽度和高度。用代码表示，这些参数如下所示（*w*表示宽度，*h*表示高度）：

```
context.drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh);
```

想要用图形精确描述所有这些参数的作用可能有些难，所以图5-4应该能够对理解它们有一定的帮助。

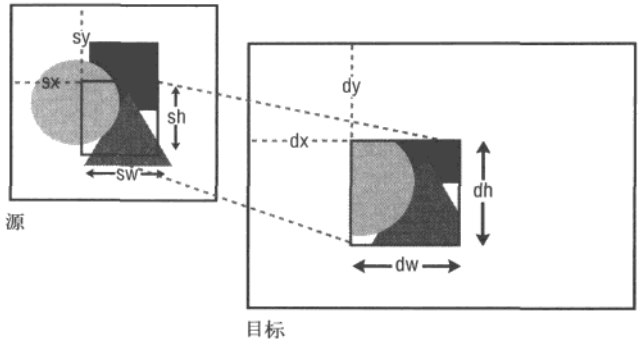


图5-4 在画布中裁剪图像

我们可以从前面的图像中裁剪出一小部分，然后将它绘制到画布中：

```
context.drawImage(image, 0, 0, 250, 250, 0, 0, 250, 250);
```

在这个例子中，我们从源图像的左上角(0, 0)开始裁剪出250像素的正方形，然后以相同的宽度和高度将它绘制到画布的左上角（参见图5-5）。

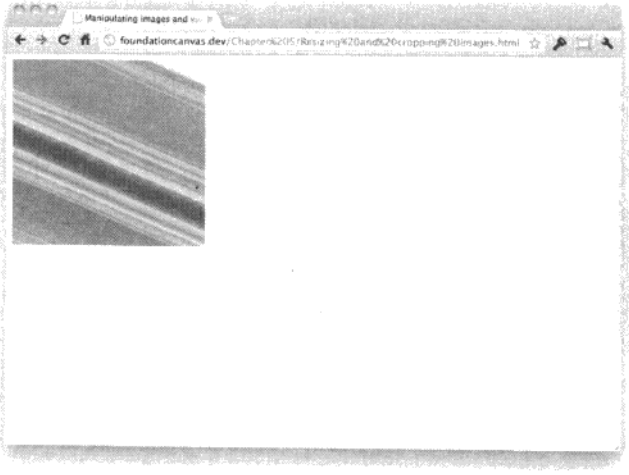


图5-5 裁剪图像而不改变其尺寸

在将裁剪的图像绘制到画布时，还可以调整它的尺寸，例如：

```
context.drawImage(image, 0, 0, 250, 250, 0, 0, 500, 500);
```

这段代码实际上与前一个例子是完全相同的，只是所绘制的图像不再保留裁剪区域的原始尺寸，而是将它放大两倍（参见图5-6）。

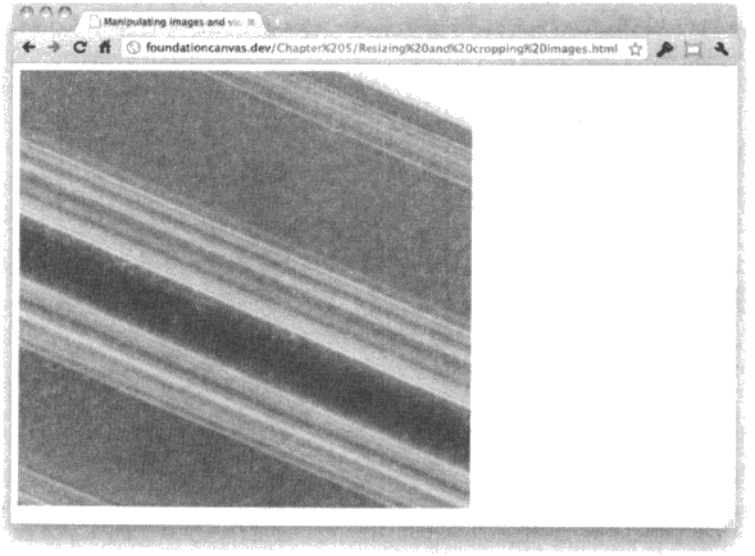


图5-6　裁剪图像同时调整图像尺寸

5.2.3　阴影

简单强调一下在进行裁剪时的阴影效果，这是很重要的。简言之，在调整图像尺寸时，阴影效果应该也显示得很好（参见图5-7）。

```
context.shadowBlur = 20;
context.shadowColor = "rgb(0, 0, 0)";

var image = new Image();
image.src = "example.jpg";
$(image).load(function() {
        context.drawImage(image, 50, 50, 300, 200);
});
```

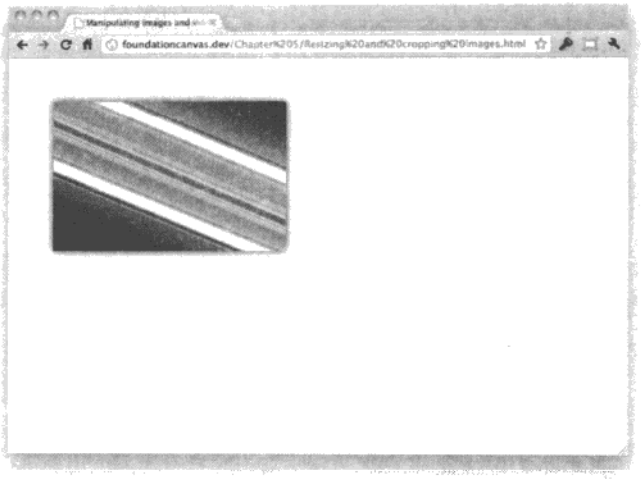


图5-7 在调整图像尺寸时阴影效果显示得很完美

然而，在一些浏览器中，对图像进行裁剪时阴影效果似乎会完全消失（参见图5-8）。


```
context.shadowBlur = 20;
context.shadowColor = "rgb(0, 0, 0)";

var image = new Image();
image.src = "example.jpg";
$(image).load(function() {
        context.drawImage(image, 0, 0, 250, 250, 50, 50, 250, 250);
});
```

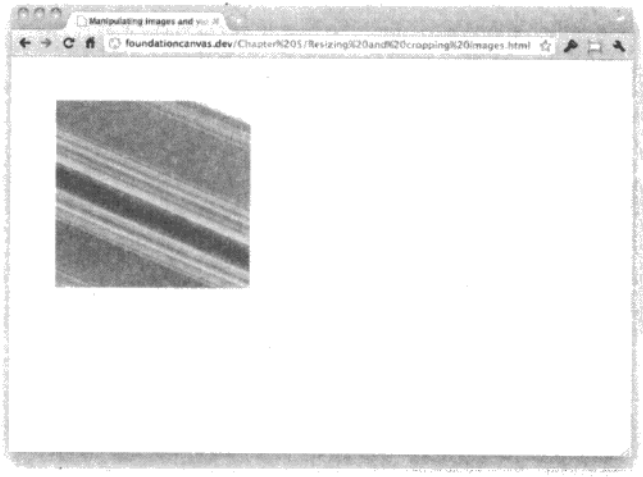


图5-8 在一些浏览器中裁剪图像会丢失阴影效果

这个问题只出现在当前版本的WebKit浏览器中，如Safari和Chrome。官方规范规定了图像在绘制到画布时应当支持阴影效果，只是有些浏览器还没有完全支持这一点。

这就是关于在画布中调整和裁剪图像的全部内容。如果希望执行更多的操作，可以使用2D渲染上下文的变形功能，我们马上开始介绍这部分内容。

5.3　图像变形

正如5.1节中介绍的，在画布中绘制图像之后，我们就可以对它执行所有的2D渲染上下文方法。我们在第4章中学过，变形作为一组方法使我们能够在图像上做出一些非常漂亮的效果。我们已经知道了变形的工作原理，所以现在继续学习如何使用它们来操作图像。

5.3.1　平移

这是到目前为止最简单的图像变形方法：

```
context.translate(100, 100);

var image = new Image();
image.src = "example.jpg";
$(image).load(function() {
        context.drawImage(image, 0, 0, 500, 500, 0, 0, 300, 300);
});
```

它在绘制图像之前将画布平移，代码是我们已经熟悉的（参见图5-9）。

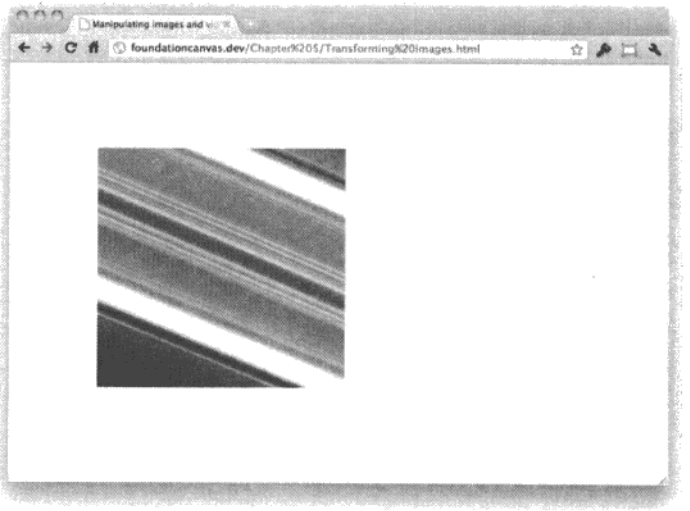


图5-9　平移图像

5.3.2 旋转

以前，在浏览器中旋转图像是很难实现的，但是利用画布这个操作变得很容易（参见图5-10）。

```
context.translate(250, 250);
context.rotate(0.7854); // 旋转45度

var image = new Image();
image.src = "example.jpg";
$(image).load(function() {
        context.drawImage(image, 0, 0, 500, 500, -150, -150, 300, 300);
});
```

同样，这段代码中并没有什么新东西。其中最有意思的是这些方法可用来处理之前平淡无奇的静态图像。

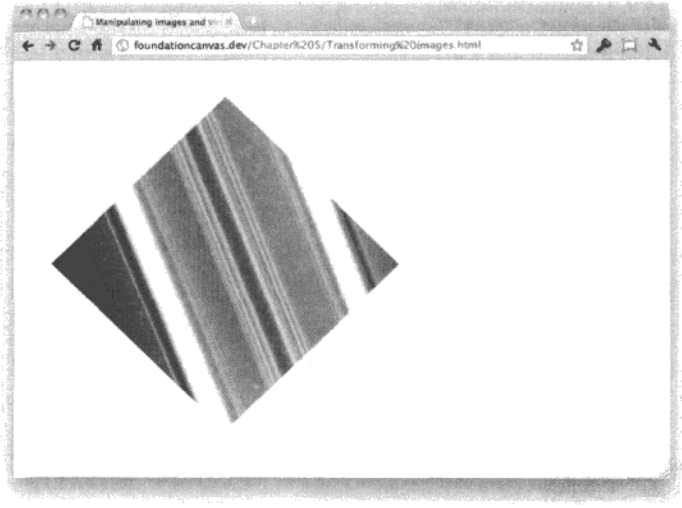


图5-10 旋转图像

5.3.3 缩放与翻转

所有的变形方法中最随机的一个就是完全翻转图像。例如，通过各种方式对同一图像进行翻转，可以创建出万花筒效果（参见图5-11）。

```
var image = new Image();
image.src = "example.jpg";
$(image).load(function() {
        // 左上角
        context.translate(50, 50);
        context.drawImage(image, 0, 0, 500, 500, 0, 0, 200, 200);
```

```
        // 左下角
        context.setTransform(1, 0, 0, 1, 0, 0); // 重置变换矩阵
        context.translate(50, 450);
        context.scale(1, -1);

        context.drawImage(image, 0, 0, 500, 500, 0, 0, 200, 200);

        // 右下角
        context.setTransform(1, 0, 0, 1, 0, 0);
        context.translate(450, 450);
        context.scale(-1, -1);

        context.drawImage(image, 0, 0, 500, 500, 0, 0, 200, 200);

        // 右上角
        context.setTransform(1, 0, 0, 1, 0, 0);
        context.translate(450, 50);
        context.scale(-1, 1);

        context.drawImage(image, 0, 0, 500, 500, 0, 0, 200, 200);
});
```

这段代码太长了可能有点难以理解，但是其过程实际上很简单。它所执行的操作就是在4个不同位置绘制同一个图像，每一个都具有不同的缩放因子。如果使用负数缩放因子，就会使图像翻转。一定要记住，当图像翻转时，原点会转到图像右边，所以你必须移动原点进行补偿，以便从右到左进行绘制。例如，右上角的图像是在位置(450, 50)上绘制的，因为它已经在*x*轴方向翻转，这意味着现在它是从*x*轴450像素位置画到*x*轴250像素位置（从右到左）。

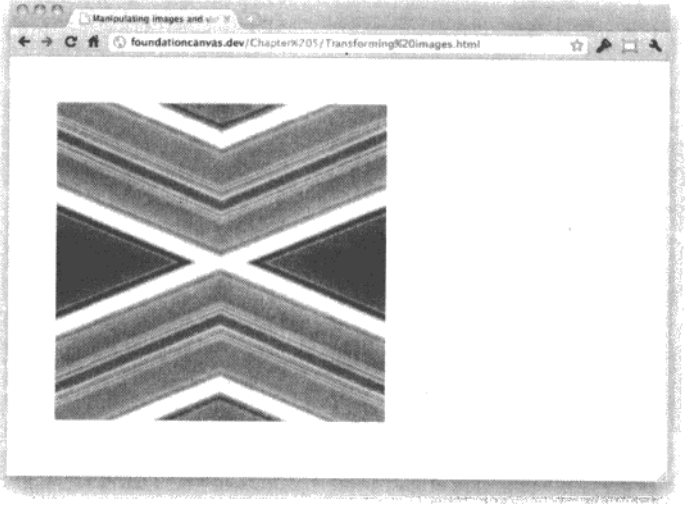


图5-11　使用负值缩放变形翻转图像

这个过程有点违反直觉，但是这确实是一种能够创建有趣图像效果的简单方法。例如，它完

全可以用来在画布中绘制出人造的反射效果。

5.4 访问像素值

虽然调整尺寸、裁剪和变形可用来创建有趣的图像效果，但画布还有另一个更强大的特性：像素处理。通过访问2D渲染上下文的各个像素，我们就能够得到每一个像素的颜色和阿尔法值等信息。我们还能够修改每一个像素的颜色，使之显示出截然不同的效果，后续几节将介绍这个功能。

在画布中访问像素的方法是getImageData。这个方法有4个参数：要访问的像素区域原点坐标(x, y)、像素区域的宽度和高度（参见图5-12）。它可以用代码表示为：

```
context.getImageData(x, y, width, height);
```

调用getImageData不会出现任何可见的效果，但是它会返回一个2D渲染上下文ImageData对象。这个ImageData对象包含3个属性：width表示所访问像素区域的宽度，height表示像素区域的高度，data是一个包含所访问区域中全部像素信息的CanvasPixelArray。

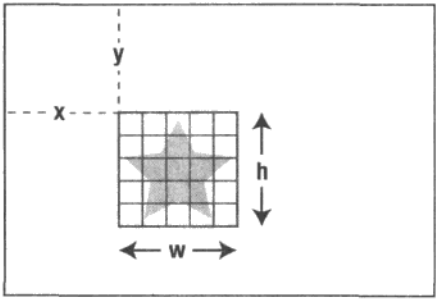


图5-12 调用getImageData方法的示意图

我认为width和height属性不需要多做解释了，此处我们真正关注的是data属性。data属性存储的是一个CanvasPixelArray，它是一个JavaScript一维数组。每一个像素用4个整数值表示，范围从0至255，分别表示红（r）、绿（g）、蓝（b）和阿尔法值（a）（参见图5-13）。所以，数组的前4项（0~3）是第一个像素的颜色值，接下来4项（4~7）是第二个像素的颜色值，以此类推。CanvasPixelArray在这里是关键，所以一定要正确理解它的工作原理。

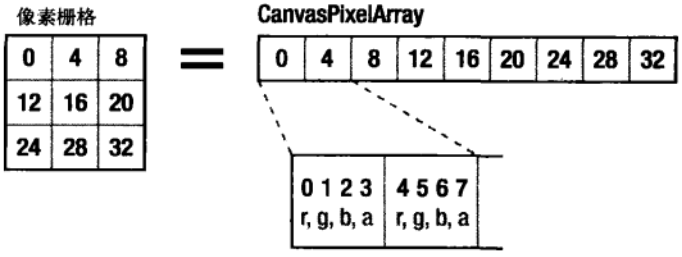


图5-13 3×3区域的CanvasPixelArray

在详细解释之前，我们先看一个简单示例。我们使用图5-13所定义的索引数字来访问CanvasPixelArray中第一个像素的RGBA值。

```
var imageData = context.getImageData(0, 0, 3, 3); // 3×3栅格
var pixel = imageData.data; // CanvasPixelArray

var red = pixel[0];
var green = pixel[1];
var blue = pixel[2];
var alpha = pixel[3];
```

CanvasPixelArray本身绝对不知道所访问的像素区域的尺寸。相反，返回的数组实际上只是一长串RGBA颜色值，它的长度等于所访问区域的像素个数乘以4（每个像素有4个颜色值）。例如，如果访问一个宽度和高度均为3个像素的像素栅格，那么CanvasPixelArray的长度就是36（3×3×4）；宽度和高度为200时，则长度为160 000 (200×200×4)，以此类推。

CanvasPixelArray中的像素排列顺序很简单：左上角像素位于数组开头（从位置0红色到位置3阿尔法值），而右下角像素位于数组末尾。这意味着，在所访问的区域中，每一行像素是从左到右访问的，直至到达行尾，然后再同样从左到右访问下一行（参见图5-13的栅格）。

所以，如果CanvasPixelArray只是一长串颜色值，而不知道像素区域的尺寸，那么应该如何从数组访问一个具体像素呢？在图5-13所示的例子中，应该如何访问(*x*, *y*)坐标位置为(2, 2)的中心像素呢？通过查看图5-13，我们很容易发现它从数组索引16开始，但是如果没有这个图，我们应该如何确定呢？如果你现在不知道确定方法，请不要担心，因为在开始学习画布时我也全然不知道如何计算。很让人头痛！幸好，一些聪明的人已经帮我们计算出一个公式，我们可以用这个公式准确地计算出你需要从CanvasPixelArray中访问的像素，而且它非常简单：

```
var imageData = context.getImageData(0, 0, 3, 3); // 3×3栅格

var width = imageData.width;
var x = 2;
var y = 2;

var pixelRed = ((y-1)*(width*4))+((x-1)*4);
var pixelGreen = pixelRed+1;
var pixelBlue = pixelRed+2;
var pixelAlpha = pixelRed+3;
```

现在，我们最关注的地方是计算像素红色值索引位置的公式。我们拆解分析这个公式，以了解它的计算原理：

```
(y-1)
```

因为我们使用非0坐标值定义像素的(*x*,*y*)坐标位置，所以需要将坐标值减1。它的作用只是将画布所使用的坐标系统转换为数组所使用的从0开始的坐标系统。

```
(width*4)
```

这会得到图像中每一行的颜色值个数。通过将(y-1)的结果与这个数相乘，就能够得到所

访问行的开头位置的数组索引值（y 坐标位置）。在这个例子中，索引值是12，这对应图5-13的第二行。

```
(x-1)*4
```

这里我们对y 坐标位置重复相同的计算——将它转换成从0开始的坐标系统。然后，将列（x 坐位置）乘以4，得到所访问列的前一行颜色值个数。

将列索引值与行索引值相加，最终可以得到所访问像素的第一个颜色（红色）的索引值。在这个例子中，它应该是16（参见图5-14）。

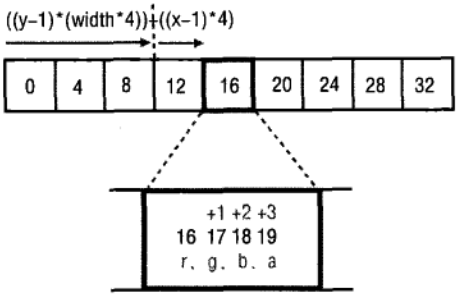


图5-14 访问CanvasPixelArray中的像素

一旦得到红色像素的索引值，其他部分就很简单了。只需要给红色索引值分别加上1、2或3，就可以得到其他三种颜色——绿、蓝和阿尔法值。

我相信，这一步不难理解，我希望通过这样的解释，你已经能够理解访问画布像素的方法和原因了。

在继续学习其他内容之前，我们来创建一个有趣的颜色拾取器。

```
var image = new Image();
image.src = "example.jpg";
$(image).load(function() {
        context.drawImage(image, 0, 0, 500, 333);
});

canvas.click(function(e) {

        var canvasOffset = canvas.offset();
        var canvasX = Math.floor(e.pageX-canvasOffset.left);
        var canvasY = Math.floor(e.pageY-canvasOffset.top);

        var imageData = context.getImageData(canvasX, canvasY, 1, 1);
        var pixel = imageData.data;
        var pixelColor = "rgba("+pixel[0]+", "+pixel[1]+", "+pixel[2]+", "+pixel[3]+")";

        $("body").css("backgroundColor", pixelColor);
});
```

我将跳过前几行代码，因为你已经知道它们的作用了。我们要关注的是jQuery的click方法，它是在指定元素上发生鼠标点击事件时调用的。在这里，元素就是画布。click方法中的回调函

数会传递给你一个包含事件信息的参数，这里是e。这个参数包含了相对于整个浏览器窗口的鼠标点击位置的(x, y)坐标，它可用来处理画布上发生的点击事件。

通过使用jQuery的offset方法，我们就能够得到画布与浏览器窗口顶部和左边的像素距离。然后，用鼠标点击位置的x 坐标（pageX）减去画布的左侧偏移量，就可以得到点击位置在画布上的x坐标。如果对鼠标点击位置y 坐标和顶部偏移量进行相同的计算，将得到鼠标点击位置相对于画布原点的(x, y)坐标值（参见图5-15）。

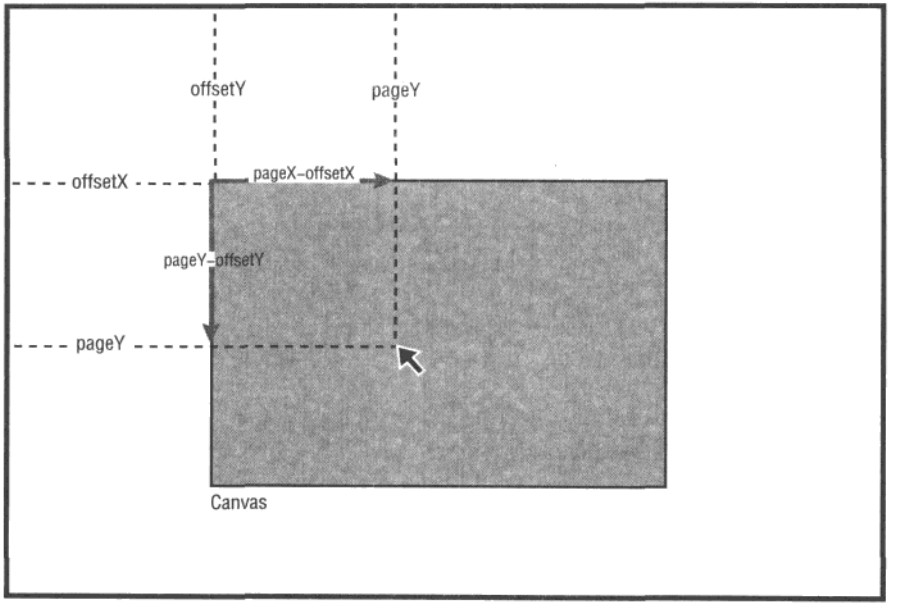


图5-15　找到鼠标点击位置在画布中的(x, y)坐标值

现在，我们得到了点击位置在画布中的(x, y)位置，下一步是查询该点的颜色值。为此，我们将canvasX和canvasY传入getImageData方法。我们只需要一个像素的数据，这就是把getImageData调用的宽度和高度都设为1的原因；这样可以保持数据尽可能小。

一旦得到ImageData对象，就可以将它保存在一个变量中，然后访问data属性中的CanvasPixelArray。由于只得到一个像素的数据，所以检索颜色值就简单到只需访问CanvasPixelArray中的前4个索引。我们将修改整个网页的CSS背景，所以要用这些值创建一个表示CSS RGBA颜色值的字符串。

最后一步是将这个CSS颜色值传递给jQuery的css方法，它可以修改HTML body元素的background-color CSS属性。如果一切正常，这会把网页的背景颜色设置为你在画布中点击的那个像素的颜色（参见图5-16）。

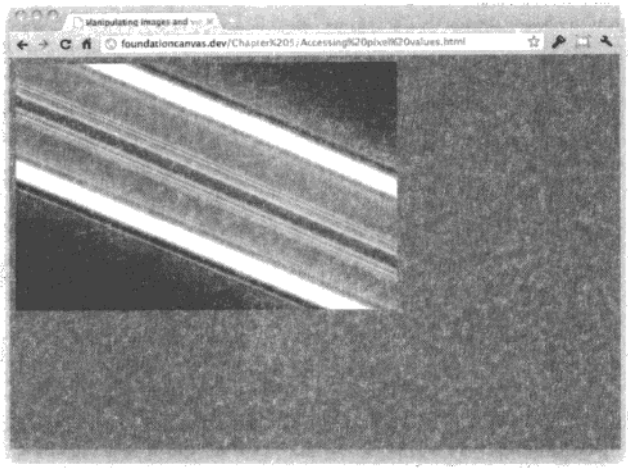


图5-16 根据画布像素颜色修改背景颜色

安全问题

如果在自己的计算机上操作这些例子，而不是将它上传到Web服务器上，那么你可能不会看到任何结果或者会遇到一个安全错误。这是因为，如果图像与控制画布的JavaScript不在同一个位置，那么画布对于访问这个图像的像素级数据会有严格的限制。解决这个问题的最简单方法就是将这些例子上传到一个Web服务器上，或者上传到一个本地开发环境中，如Mac的MAMP或Windows的WAMP。这种解决方法的关键在于JavaScript和所访问的图像必须通过相同的域名访问（如rawkes.com）。你可能已经注意到截图了，我是在一个本地开发环境运行这些例子的，域名是foundationcanvas.dev。

按照一般的经验，如果执行像素操作时出现问题，那么一定要确认所有内容都是位于同一个域名下。

5.5 从零绘制图像

上一节内容很多，但学习它是非常值得的，因为我们现在可以开始制作一些真正漂亮的图像了，例如从创建像素开始制作自己的图像。

要创建一些像素，需要调用2D渲染上下文的`createImageData`方法。通过传入宽度和高度，它会返回一个包含所有常规属性的`ImageData`对象：`width`、`height`和（最重要的）`data`。`data`属性所包含的`CanvasPixelArray`将保存新的像素，此时它们是不可见的，因为它们都被设置为透明黑色。

在下一个例子中，我们将创建一个包含200×200透明像素区域的`ImageData`对象，然后将它们全部修改成红色。

```
var imageData = context.createImageData(200, 200);
var pixels = imageData.data;
```

变量pixels仅用作访问CanvasPixelArray中的像素的快捷方式。

修改颜色值与查询颜色值一样简单：都是读写CanvasPixelArray中的颜色值。如果想将所有像素修改为红色，那么需要使用for循环语句遍历每一个像素。

```
var numPixels = imageData.width*imageData.height;

for (var i = 0; i < numPixels; i++) {
        pixels[i*4] = 255; // 红
        pixels[i*4+1] = 0; // 绿
        pixels[i*4+2] = 0; // 蓝
        pixels[i*4+3] = 255; // 透明度
};
```

变量numPixels保存了ImageData对象中的像素个数，它就是for循环的执行次数。在每一次循环过程中，我们都使用一个简单算法给每个像素赋予颜色值。每个像素都有4个颜色值，所以将像素个数乘以4就能够得到该像素的红色颜色值在CanvasPixelArray中的索引位置。然后，就可以将红色颜色值设置为255（全色），绿色和蓝色设置为0，而阿尔法值设置为255，这样它就变成不透明的了。非常简单！

按照目前情况，我们所做的就是创建一个ImageData，然后将像素修改为红色。现在画布上还看不见任何效果，因为我们还没有将新像素画到上面。为此，我们需要调用2D渲染上下文的putImageData方法。这个方法可以接受3个或7个参数：ImageData对象、绘制像素数据的原点坐标(x, y)、所谓脏矩形（dirty rectangle）的原点坐标(x, y)、脏矩形的宽度和高度。在这个例子中，你暂时可以不考虑脏矩形的用途，它的作用只是定义ImageData对象中需要绘制的像素。

```
context.putImageData(imageData, 0, 0);
```

这样会在画布原点绘制新的红色像素（参见图5-17）。

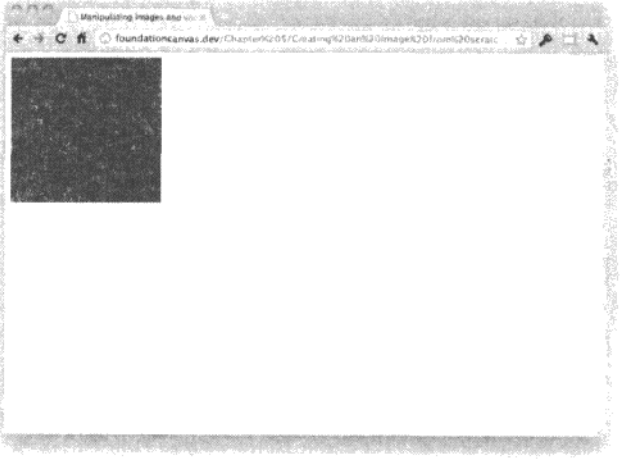


图5-17　从零开始创建和绘制像素

5.5.1 随机绘制像素

只有红色像素似乎太单调，让我们更进一步，绘制一些完全随机的颜色。这也很简单。

```
for (var i = 0; i < numPixels; i++) {
        pixels[i*4] = Math.floor(Math.random()*255); // 红色
        pixels[i*4+1] = Math.floor(Math.random()*255); // 绿色
        pixels[i*4+2] = Math.floor(Math.random()*255); // 蓝色
        pixels[i*4+3] = 255; // 透明度
};
```

通过修改前一个例子中设置颜色值的代码，我们可以插入0至255之间的随机数。我们仍然保持阿尔法值为255，否则有一些像素会变成透明的。注意，我们使用了Math.floor来向下舍入产生的随机数（例如，150.456会变成150）。

结果，我们得到一些杂乱的像素点（参见图5-18）。

> **注意** Math.random可以产生0到1之间的随机小数。将它与另一个数字相乘，就可以得到0与该数字（乘数）之间的随机数。例如，Math.random()*255将得到0与255之间的一个随机数。

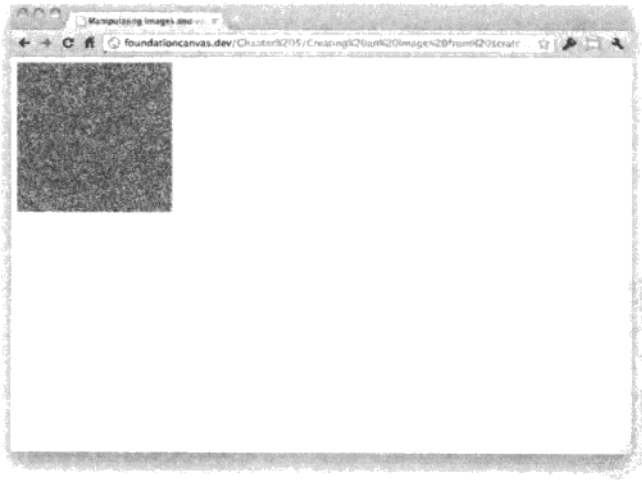


图5-18 随机设置在画布上绘制的像素的颜色

5.5.2 创建马赛克效果

但是，杂乱的像素并不是画布的最佳用途。那么创建一个马赛克效果呢？肯定更有意思一些。它的实现方法是，创建一个新像素区域，然后将它分割到一个栅格中，并为栅格每个片段设置随机颜色。最复杂的部分是计算出每个像素应该落到哪个片段，这样相同的片段就可以设置相同的

颜色。在图5-19中，我们会看到每个片段实际上是由许多像素构成的。

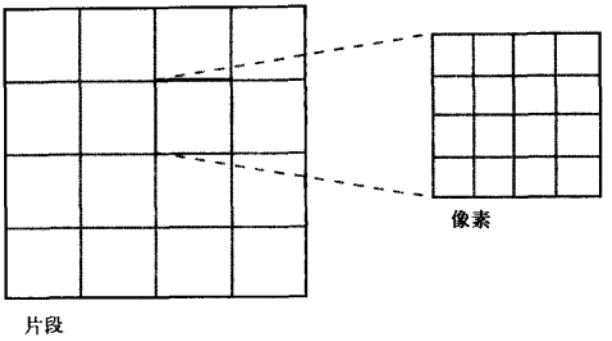


图5-19　将画布分割到像素片段栅格中

稍后，我会介绍如何计算出每个片段的像素。现在，先来做一些基础性工作。

```
var imageData = context.createImageData(500, 500);
var pixels = imageData.data;

// 马赛克块的个数
var numTileRows = 4;
var numTileCols = 4;

// 每个块的尺寸
var tileWidth = imageData.width/numTileCols;
var tileHeight = imageData.height/numTileRows;
```

前两行代码现在你应该很熟悉了，它们创建了一个500×500像素的ImageData对象，然后将CanvasPixelArray保存在一个变量中。后面的代码是定义两个变量，用于声明像素区域划分的片段数，其中包括每行每列的马赛克数。从现在起，我们将片段称为块（tile），因为这个词更能说明它们的实际作用。最后两行代码是根据ImageData对象的尺寸和各行各列的块数计算出每个块的宽度和高度（以像素为单位）。

现在，我们有了足够信息，可以开始遍历这些块和修改像素的颜色值。

```
for (var r = 0; r < numTileRows; r++) {
        for (var c = 0; c < numTileCols; c++) {
                // 为每个块设置像素的颜色值
                var red = Math.floor(Math.random()*255);
                var green = Math.floor(Math.random()*255);
                var blue = Math.floor(Math.random()*255);
        };
};
```

这是一个嵌套循环；第一个循环遍历每一行的块，第二个循环遍历当前行的每一列块（参见图5-20左边的栅格）。每一个块都赋了新的颜色值，这些值都是0至255的随机数。到现在为止，所有代码都是非常基础的。

现在，在列循环中颜色值的下方，我们要声明另外两个循环：

```
for (var tr = 0; tr < tileHeight; tr++) {
        for (var tc = 0; tc < tileWidth; tc++) {

        };
};
```

根据之前计算的块尺寸，这些循环遍历的次数与每个块中的像素个数相同。变量tr和tc表示当前访问块的像素行（基于块的高度）和像素列（基于块的宽度）（参见图5-20右边的栅格）。在这个例子中，每一个块的宽和高都是125像素，所以tr将会循环125次，而在每一次循环中，tc将会再循环125次。

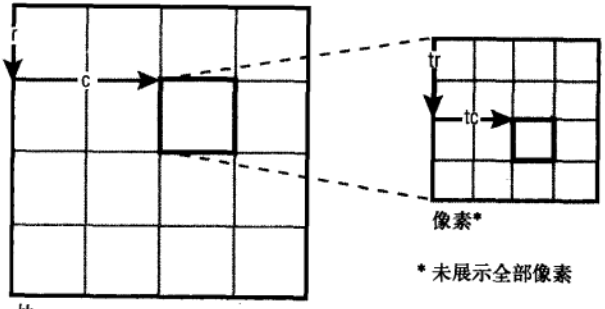


图5-20 循环每一个块和块中每一个像素

然而，我们现在仍然还无法访问每一个块中的实际像素。我们现在得到的是所访问的块的行和列（变量r和c），以及你在该块中所处的像素的行和列（变量tr和tc）对于它们本身而言，这些变量并不足以用来访问CanvasPixelArray中的像素。为此，需要将它们转换为以0开始的像素位置坐标(*x*, *y*)，就像是没有块存在时那样。这似乎有些难以置信，但是请相信我。

将下面的代码添加到第二个循环中，然后我将解释会出现什么结果，这事实上是很简单的：

```
var trueX = (c*tileWidth)+tc;
var trueY = (r*tileHeight)+tr;
```

这两个变量可以计算出像素的真实位置。例如，要计算*x*轴位置，首先要将当前块的列数（2）乘以每个块的宽度（125），这样就得到所访问块的左边缘的*x*坐标位置（2×125=250）。然后，再加上所访问的块中像素的列数（例如，10），这样就得到没有块时的*x*轴确切坐标（250+10=260）。对y轴重复这个过程，就可以得到开始修改像素颜色值的位置坐标(*x*, *y*)。

将下面的代码加到trueX和trueY的赋值语句后面：

```
var pos = (trueY*(imageData.width*4))+(trueX*4);

pixels[pos] = red;
pixels[pos+1] = green;
pixels[pos+2] = blue;
pixels[pos+3] = 255;
```

这里并没有出现新代码，它只是访问像素的红色颜色值，然后使用之前设置的颜色值进行赋

值。因为这里从0开始计算，所以必须将trueX和trueY减1，就像前面第一次看到这个公式时的做法一样。

最后一步是将像素绘制到画布上，所以要将下面的putImageData调用放到4个循环之外：

```
context.putImageData(imageData, 0, 0);
```

如果一切正常，画布上就会出现生动的马赛克效果（参见图5-21）。

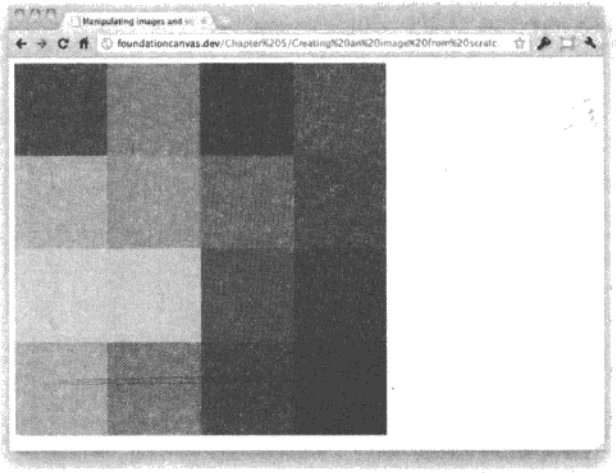


图5-21　完成的马赛克效果

通过修改每行和每列的块数，还能创建出更有趣的效果（参见图5-22）。

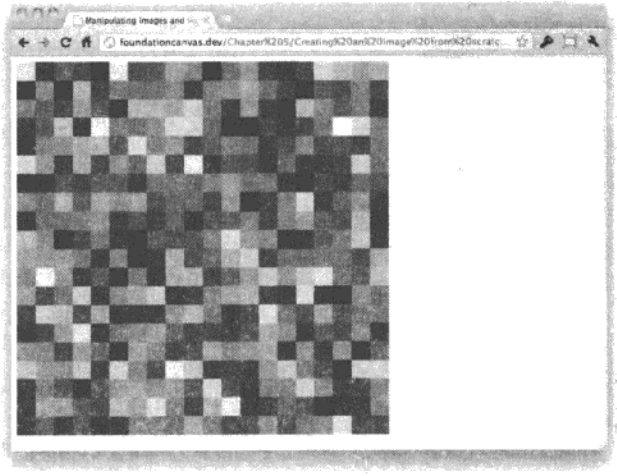


图5-22　修改块的个数，创建出不同的马赛克效果

我想你现在可以休息一下，做个深呼吸——因为你在本节学到了非常丰富的知识！

5.6 基本图像效果

修改像素的颜色值并不意味着必须从零开始创建整个图像，已经存在的图像也是可以修改的。有一个例子就是基本照片处理——通过修改图像中的像素来修改它的显示效果。这种效果在画布中实现是很简单的，特别是现在你已经掌握了像素的操作方法。

5.6.1 反转颜色

这个效果将反转图像的颜色值，这会使它看起来有些奇怪（找不到更适合的词来形容）。基本方法就是用255减去像素现在的颜色值（150），所得的就是反转后的颜色（255–150=105）。让我们尝试一些不同的操作，然后看看最后的代码，这里并没有新知识。

```
var image = new Image();
image.src = "example.jpg";
$(image).load(function() {
        context.drawImage(image, 0, 0, 1024, 683, 0, 0, 500, 500);

        var imageData = context.getImageData(0, 0, canvas.width(), canvas.height());
        var pixels = imageData.data;
        var numPixels = pixels.length;

        context.clearRect(0, 0, canvas.width(), canvas.height());

        for (var i = 0; i < numPixels; i++) {
                pixels[i*4] = 255-pixels[i*4]; // 红色
                pixels[i*4+1] = 255-pixels[i*4+1]; // 绿色
                pixels[i*4+2] = 255-pixels[i*4+2]; // 蓝色
        };

        context.putImageData(imageData, 0, 0);
});
```

前面几行代码创建了一个新的Image对象，然后加载上一节使用的示例图像。等到图像加载完成之后，将图像绘制到画布上，并将包含所有像素的ImageData对象保存到一个变量中。在保存了像素之后，清除画布，开始循环处理原始图像中所有的像素。

在循环中，我们使用与上一节中第一个例子（红色正方形）相同的方法来访问像素，并用255减去当前值。不需要对阿尔法值进行任何处理，因为我们要保留原始图像中的这个值。

最后是将像素绘制回画布，这样就得到一个反转颜色的图像（参见图5-23）。很有趣！

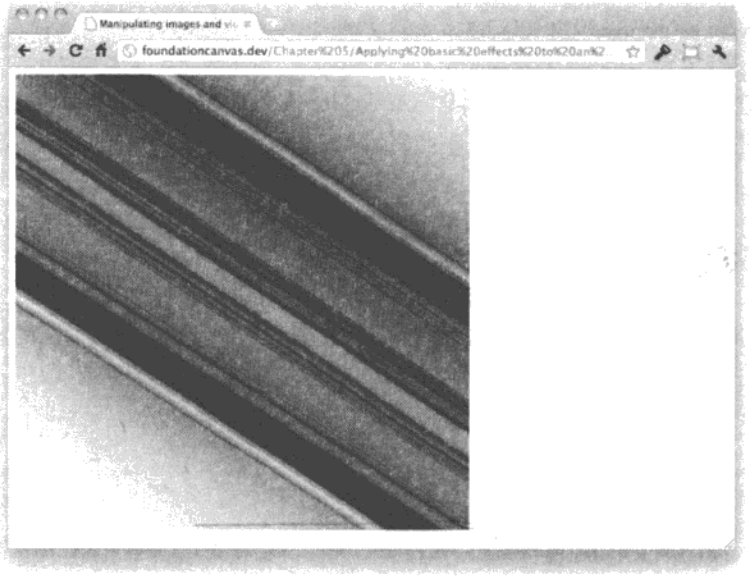


图5-23　反转图像的颜色

5.6.2　灰度

另一个有趣的效果是灰度，这也许是更有用的一种效果。将彩色图像变为灰色（有时候也称为黑白色；但是这种说法并不准确），除了访问和修改颜色值，实现代码与反转颜色例子中的代码完全相同。

```
for (var i = 0; i < numPixels; i++) {
        var average = (pixels[i*4]+pixels[i*4+1]+pixels[i*4+2])/3;
        pixels[i*4] = average; // 红色
        pixels[i*4+1] = average; // 绿色
        pixels[i*4+2] = average; // 蓝色
};
```

将彩色转换为灰度要求计算出现有颜色值的平均值，即将它们加在一起然后除以颜色个数。这个平均颜色将作为三种颜色（红、绿和蓝）的值。其结果是将每一种颜色转换为灰度（参见图5-24）。

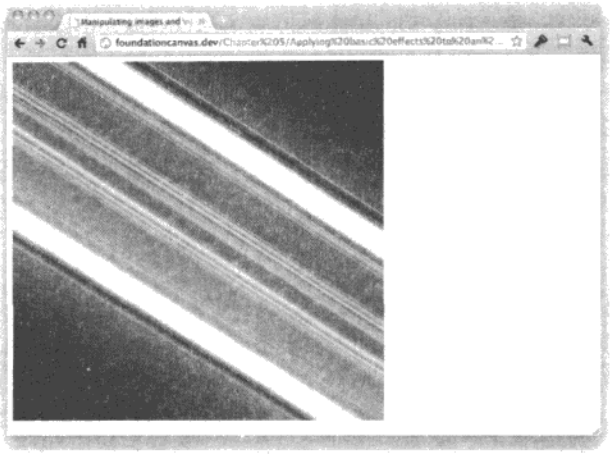


图5-24 将图像转换为灰度

5.6.3 像素化

你是否曾经看到过新闻或文件中人物脸孔被像素化的情况？这是一种强大的特效，它可以将图像变得不可识别，但并不真正删除整个部分。实际上重新在画布上创建会相对简单一些，只需要将图像按栅格分割，或者对每个片段的颜色取平均值，或者选取每个片段的颜色。我们将使用的代码与上一节马赛克的例子很相似。

```
var image = new Image();
image.src = "example.jpg";
$(image).load(function() {
        context.drawImage(image, 0, 0, 1024, 683, 0, 0, 500, 500);

        var imageData = context.getImageData(0, 0, canvas.width(), canvas.height());
        var pixels = imageData.data;

        context.clearRect(0, 0, canvas.width(), canvas.height());

        var numTileRows = 20;
        var numTileCols = 20;

        var tileWidth = imageData.width/numTileCols;
        var tileHeight = imageData.height/numTileRows;

        for (var r = 0; r < numTileRows; r++) {
                for (var c = 0; c < numTileCols; c++) {

                };
        };
});
```

循环之前的代码都是一样的。访问图像，等待图像加载，将它绘制到画布中，保存ImageData

对象，从画布清除该图像，然后给分割的图像赋值确定块（片段）的数量和尺寸。

这两个循环的工作方式与马赛克的例子是一样的：第一个循环处理每一行块，第二个循环则处理当前行中的每一个块。而新的代码位于循环中；访问颜色值和创建像素化效果。

这里将使用第二种方法来获取像素化效果的颜色值，为每一个块选择一种颜色。最简单的方法是使用块的中心位置像素，将以下代码添加到第二个循环中，就可以得到这个信息：

```
var x = (c*tileWidth)+(tileWidth/2);
var y = (r*tileHeight)+(tileHeight/2);

var pos = (Math.floor(y)*(imageData.width*4))+(Math.floor(x)*4);
```

前两行将得到当前块中心像素从0开始表示的(*x*, *y*)坐标。这个计算方法与马赛克例子非常相似，先找到块边缘的(*x*, *y*)坐标位置，然后加上一半宽度或高度，从而确定中心。然后将(*x*, *y*)坐标传入标准公式，这样就得到CanvasPixelArray中该像素的索引值。但你可能注意到了，(*x*, *y*)坐标值在Math对象的floor方法中进行了取整处理。其原因是，除非(*x*, *y*)是整数，否则这个返回的索引将是错误的，所以我们使用floor方法将值取整为下一个最小整数（例如，3.567取整后变成3）。

最后，我们得到了访问颜色值和绘制像素化效果所需要的全部信息。将下面的代码插入到变量pos的声明语句之后。

```
var red = pixels[pos];
var green = pixels[pos+1];
var blue = pixels[pos+2];

context.fillStyle = "rgb("+red+", "+green+", "+blue+")";
context.fillRect(x-(tileWidth/2), y-(tileHeight/2), tileWidth, tileHeight);
```

这里没有新代码，它只是访问红色、绿色和蓝色值，然后使用这些值来设置fillStyle。最后一步是在块的位置上绘制一个正方形，它是使用所访问的颜色填充的。结果是将示例图像变成一个独特的像素化效果（参见图5-25）。

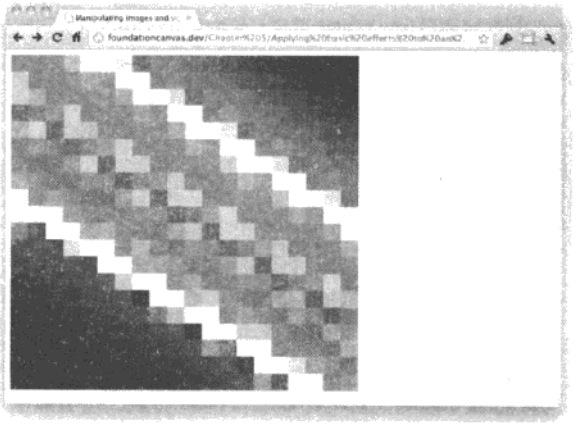


图5-25　使用画布实现图像像素化

我们可以进一步将正方形修改为圆形（参见图5-26）。

```
context.beginPath();
context.arc(x, y, tileWidth/2, 0, Math.PI*2, false);
context.closePath();
context.fill();
```

现在效果更酷了，我都情不自禁地要叫出来了！

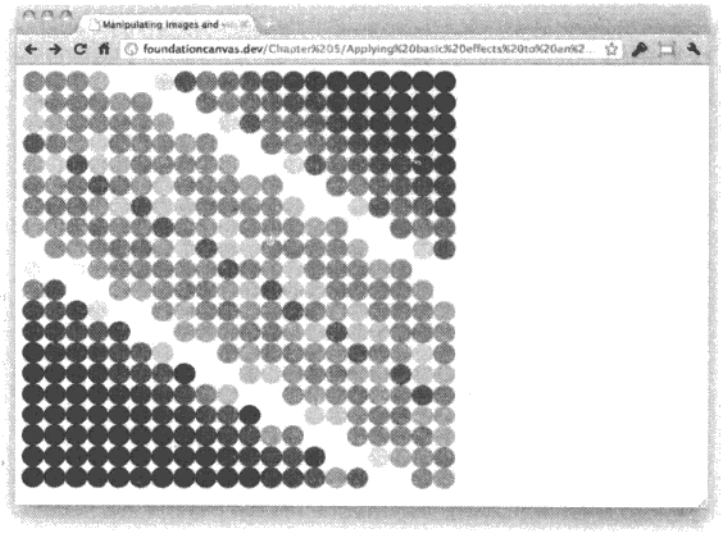


图5-26 在画布中使用圆形替换正方形实现图像像素化

5.7 视频处理

5.1节中介绍过drawImage方法如何接受一个HTML5 video元素作为输入。现在是时候学习在画布中处理图像这部分有趣且令人兴奋的内容了。如果你现在仍然认为视频处理很难实现，那么我会告诉你：你错了，它与图像处理几乎完全一样！然而，它的效果更有意思。

5.7.1 创建 HTML5 video 元素

在开始之前，我们需要创建一个HTML5 video元素。将下面的代码插入到HTML页面的body元素之中：

```
<video id="myVideo" width="500" height="281" controls>
        <source src="example.mp4" type="video/mp4"></source>
        <source src="example.ogg" type="video/ogg"></source>
</video>
```

你可以使用代码提供的示例视频（由James Watson提供），也可以使用自己的视频。如果使用自己的视频，一定要修改video元素的宽度和高度，而且如果希望在主流浏览器上访问，一定要将它编码为MPEG-4和OGG格式。

按照这种情况，网页中唯一需要的代码是第3章学习2D渲染上下文时所使用的基本模板，而video元素则是body元素中唯一的代码。如果一切正常，运行结果将会如图5-27所示。我们应该能够播放视频，因为我们在定义video元素时使用了controls属性。

图5-27　在网页上添加一个HTML5视频

5.7.2　使用 HTML5 video API

虽然这确实很好（HTML5视频本身就不错），但是我们将要实现的功能会更强大。下一步是创建两个HTML button元素，在其中使用JavaScript进行视频控制。在video元素下面插入以下代码：

```
<div>
        <button id="play">Play</button>
        <button id="stop">Stop</button>
</div>
```

在JavaScript中使用按钮进行视频控制相对简单，我们只需要访问HTML5 video元素的API。为此，需要将下面的代码插入到script元素的文档就绪函数中。

```
var video = $("#myVideo");

$("#play").click(function() {
        video.get(0).play();
});
```

```
$("#stop").click(function() {
        video.get(0).pause();
});
```

第一行代码将video元素的jQuery对象赋给video变量。然后是对jQuery的click方法的两个调用，每一个调用都有一个回调函数。每一个回调函数都会调用一个HTML5 video元素API方法；其中一个调用play，另一个调用pause。这两个方法都属于video DOM对象，所以在此之前我们需要调用jQuery的get方法，方式与访问canvas元素的2D渲染上下文一样。在JavaScript中开始和停止视频，只需要调用play和pause（参见图5-28），非常简单！

图5-28 通过JavaScript API控制HTML5视频

5.7.3 设置画布

鉴于你将使用与上一节几乎相同的像素化效果，所以实际上没必要看到原始视频。所以，让我们来个“一石二鸟”，将video元素置于与原始视频具有相同尺寸的画布之中。

```
<canvas id="myCanvas" width="500" height="281">
        <video id="myVideo" width="500" height="281" controls="true">
                <source src="example.mp4" type="video/mp4"></source>
                <source src="example.ogg" type="video/ogg"></source>
        </video>
</canvas>
```

创建JavaScript变量，保存画布的一些对象。

```
var canvas = $("#myCanvas");
var context = canvas.get(0).getContext("2d");
```

这时，使用支持HTML5 Canvas的浏览器将看不到任何内容，而使用不支持HTML5 Canvas但支持HTML5视频的浏览器将能够看到视频。现在，我们还不能看到任何内容，但这正是我们想要的结果（参见图5-29）。

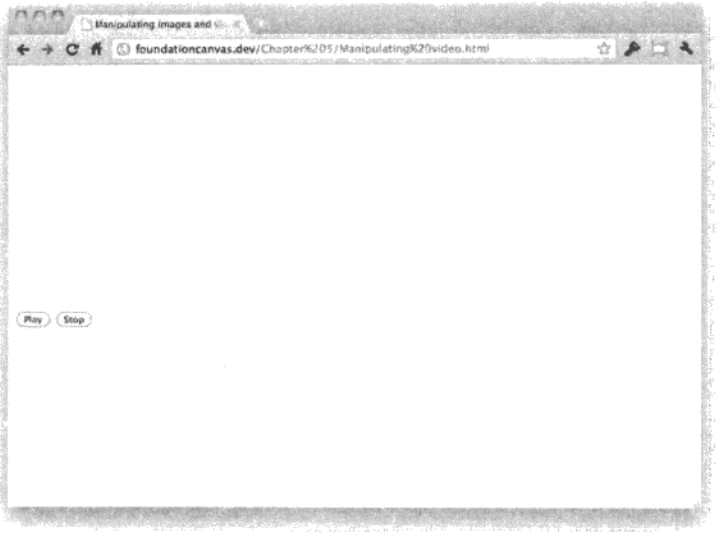


图5-29　将HTML5 video置于canvas元素之中

在准备操作视频时，还需要做一些准备工作。将下面的代码插入到context变量下方：

```
video.bind("play", function() {
        drawCanvas();
});

function drawCanvas() {
        if (video.get(0).paused || video.get(0).ended)
                return false;
};
```

首先，我们使用jQuery的bind方法创建一个回调函数，它会在视频开始播放时运行，这是通过监听play事件实现的。回调函数将调用一个自定义函数。这个函数包含所有创建像素化效果和绘制画布内容的功能。

第二步是创建drawCanvas函数。通过一个条件语句，检查视频是否处于暂停或停止状态，如果是，则调用return停止函数执行。视频状态是通过HTML5 video API的paused和ended属性值进行检查的。执行这种检查是很好的做法，因为它可以避免执行不必要的操作，例如，对实际上不再播放的视频执行像素化操作。

现在，我们对视频执行像素化操作。这是最有趣的部分！将下面的代码插入到条件语句中。

```
context.drawImage(video.get(0), 0, 0, 500, 281);
```

```
var imageData = context.getImageData(0, 0, canvas.width(), canvas.height());
var pixels = imageData.data;

context.clearRect(0, 0, canvas.width(), canvas.height());

var numTileRows = 36;
var numTileCols = 64;

var tileWidth = imageData.width/numTileCols;
var tileHeight = imageData.height/numTileRows;

for (var r = 0; r < numTileRows; r++) {
        for (var c = 0; c < numTileCols; c++) {
                var x = (c*tileWidth)+(tileWidth/2);
                var y = (r*tileHeight)+(tileHeight/2);

                var pos = (Math.floor(y)*(imageData.width*4))+(Math.floor(x)*4);

                var red = pixels[pos];
                var green = pixels[pos+1];
                var blue = pixels[pos+2];

                context.fillStyle = "rgb("+red+", "+green+", "+blue+")";
                context.fillRect(x-(tileWidth/2), y-(tileHeight/2), tileWidth,
tileHeight);
        };
};
```

除了第一行和各行各列的块数，这段代码的其余部分与上一节的像素化代码都是相同的。第一行调用了drawImage，区别是现在它的第一个参数是HTML5 video元素。这是因为drawImage将HTML5 video元素的当前帧绘制到画布上，如果只调用一次（它将是一个静态图像），是没有用的，但是如果与视频播放相同的速度调用，那么效果就不一样了。你很快会看到它的效果。

要计算各行各列的块数，首先需要计算视频的比例（281÷500=0.562），然后任意指定每行的块数（例如，64），再使用这个比例来计算每列的块数（64×0.562=35.968=36）。这样就得到了全部的正方形块。非常简单！

实现这个效果的最后一步是在drawCanvas()方法之后添加一个定时器，与视频的帧播放速度同步重复调用这个方法。方法是在drawCanvas函数的两个循环之后添加一个setTimeout调用。

```
setTimeout(drawCanvas, 30);
```

这个定时器重复速度非常快，但因为drawCanvas前面进行了视频检查，所以它只会在视频真正播放时才会运行。

如果一切正常（我相信这是肯定的），在单击Play按钮时，画布上会显示一个像素化的视频（参见图5-30）。

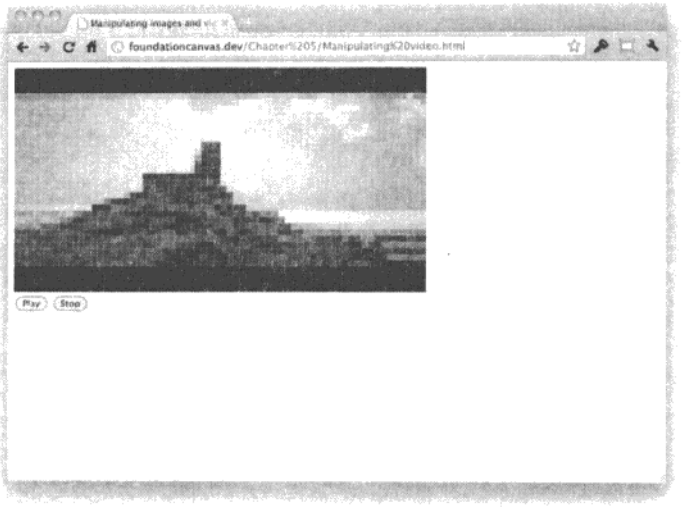


图5-30　在HTML5视频上完成的像素化效果

单击Stop按钮会暂停视频播放和像素化效果，再次单击Play按钮会在视频暂停的位置继续播放。甚至，我们还可以像上一节的像素化例子一样将矩形修改为圆形（参见图5-31）。

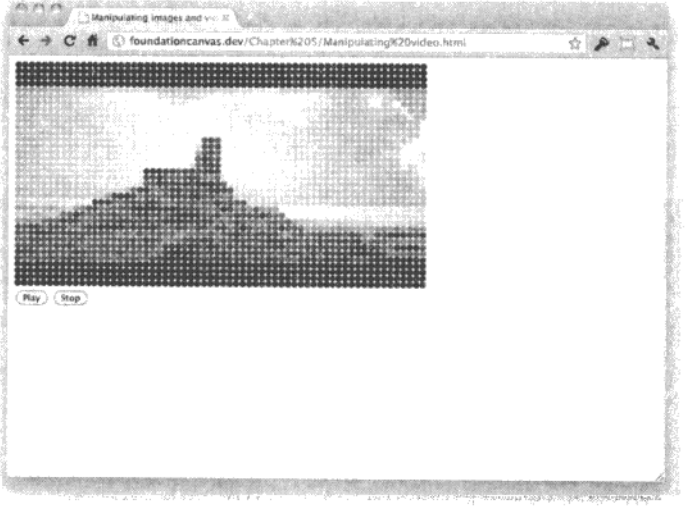


图5-31　使用圆形替换正方形实现像素化

祝贺你！你在本节学习到了非常有用的特性。你现在肯定深深体会到这种视频处理是非常强大的。

5.8 小结

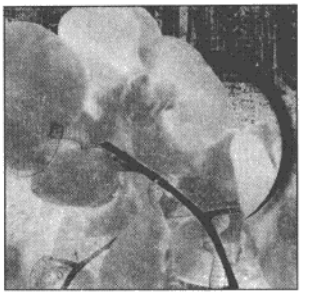
在一定程度上，本章内容确实有点复杂。在短短一章中学到这么多知识，你一定感到非常自豪。实际上，内容只是一方面，关键是你学会了思路。

你已经学习了如何将一个图像加载到画布中，以及对图像进行尺寸调整和裁剪，使它们按照你希望的方式显示。还学习了如何实现图像变形，创建一些特殊的效果。而且，你还学习了更高层次的知识——像素处理，这部分内容在一开始是很难掌握的。如果对这部分内容还不自信，那么我希望你能再复习一遍像素处理的方法。

本章重点是介绍如何使用像素处理来创建你自己的图像，以及使用它来修改现有图像。最后创建的项目说明了如何修改HTML5视频，这是一个最令人激动的画布特性。

本章内容繁多，但是你在本章学到的技术可能会是所有内容中最精彩的部分。从现在起，我们将开始学习如何在静态的画布绘图中添加一些动画，学习制作超炫游戏所需要的各方面知识。

第 6 章



制作动画

本章将介绍如何通过JavaScript的强大功能和你的头脑制作动画。首先，学习在画布中实现动画的工作原理。然后，你将亲自动手创建第一个动画循环。通过实践，你会发现一种记住你需要创建的动画内容的方法，并探索多种方法来使形状产生逼真的动画效果。最后，本章将讨论如何为形状赋予一点智能，让它们在画布的边界处能反弹回来，而不是在视野中永远消失。

6.1 画布中的动画

实际上，画布中的动画与一般的动画在理论上并没有太大的区别。深入研究动画的本质可以发现：动画只是一连串的图像，每个图像之间的差别非常微小，并且它们以极快的速度连续显示。其中的奥妙就在于，每秒钟显示的图像非常多，人的肉眼通常认为自己看到的是一个正在运动的物体，而不是一张张连续显示的静态图像。其原理就像你在学校读书时翻书一样，如果你翻书的速度足够快，一些小图形看上去就好像在运动。

当然，你不能在画布中翻页，因此需要采用其他方法来创建动画效果。仔细思考一下，你就会发现当使用代码创建动画时，只有一张纸可用，这张纸就是计算机屏幕。这意味着不可能载入一张张图像并通过手工翻页的方式来创建动画，因此需要寻找其他途径解决这个问题。连续载入一系列图像其实和翻页的效果是相同的，首先在屏幕上绘制一些对象（第一个页面），接着清除屏幕上的对象（第一页与第二页之间的过渡），然后快速在屏幕上绘制其他对象——更新图像（第二个页面）。这和翻书的原理其实是相同的，只是执行方式略有不同而已。

现在，你已经了解了在画布中创建动画所需的有关知识，这非常重要。你知道如何在屏幕上绘制对象，以及如何从屏幕上清除对象。这部分内容比较简单。较难的部分是如何使整个过程自动化，使动画在每秒钟能够发生很多次。另外，准确记忆你需要绘制的动画内容及其位置也比较难。当然，虽然我说这部分内容比较难，但其实当你真正学习起来就会觉得并不难。

6.2 创建动画循环

动画循环是创建动画效果的基础，创建动画循环其实非常简单。需要说明的是，虽然这里我们称它为循环，但其实它和循环并没有什么联系，其工作原理也不同。这里之所以称作循环，是因为它在重复发生，稍后你将看到这种效果。

动画循环的三要素是：更新需要绘制的对象（如移动对象的位置）、清除画布、在画布上重新绘制对象（如图6-1所示）。本章即将创建的动画与你执行任务的先后顺序并没有实际关系，但请务必注意：在清除对象之前不要绘制对象，否则你将看不到任何对象！

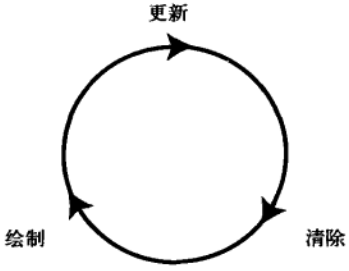


图6-1 画布中的典型动画循环

6.2.1 循环

下面我们开始创建动画循环。

```
var canvas = $("#myCanvas");
var context = canvas.get(0).getContext("2d");

var canvasWidth = canvas.width();
var canvasHeight = canvas.height();

function animate() {
        setTimeout(animate, 33);
};

animate();
```

在以上代码中，其实你只需关注animate函数，它现在非常简单。实际上，animate函数使用setTimeout方法设置了一个定时器，setTimeout方法每隔33毫秒调用一次animate函数，但这常常会创建一个无限循环。在循环外部调用animate函数即可启动该循环。也许你并不希望循环一直运行下去，因为这会消耗不必要的计算机资源，所以最好添加一个结束开关。

在canvas元素之后添加以下按钮代码：

```
<div>
        <button id="startAnimation">Start</button>
```

```
        <button id="stopAnimation">Stop</button>
</div>
```

然后在animate函数上方添加处理按钮的逻辑。

```
var playAnimation = true;

var startButton = $("#startAnimation");
var stopButton = $("#stopAnimation");

startButton.hide();
startButton.click(function() {
        $(this).hide();
        stopButton.show();

        playAnimation = true;
        animate();
});

stopButton.click(function() {
        $(this).hide();
        startButton.show();

        playAnimation = false;
});
```

以上代码的逻辑非常简单：让playAnimation变量保存一个布尔值，用于停止或播放动画循环。jQuery代码为每个按钮添加了单击事件，用于隐藏刚刚单击过的按钮并显示另外一个按钮，然后将playAnimation变量设置为正确的值。启动按钮与其他按钮略有不同，因为动画停止以后，需要再次手动启动循环。为此，只需在代码中添加一个额外的animate函数调用即可。

这些实际上都还不会对动画循环产生影响。要想控制播放，需要在setTimeout方法前面添加一个条件语句。

```
if (playAnimation) {
        setTimeout(animate, 33);
};
```

如果playAnimation变量保存了一个false值，那么动画循环将会停止运行。尝试运行一下吧，虽然你不会看到任何动画，但该按钮应该能够提供启动和停止功能。

> **注意**　为什么在动画循环中使用33毫秒作为时间间隔呢？动画在每秒钟需要的帧数通常介于25到30帧之间。1秒是1000毫秒，因此用1000除以30得到33毫秒。当然，也可以将它设置为不同的数值来使动画加速或减速。可以根据自己的需要，将动画效果的时间间隔调整为30或40毫秒。

6.2.2 更新、清除、绘制

现在已经建立了基本动画循环，接下来可以开始添加前面提到的更新、清除和绘制过程了（如图6-1所示）。我们建立一个简单的动画，使一个正方形每帧向右移动1像素。首先，需要在animate函数外部建立一个变量，用于保存当前正方形的*x*位置：

```
var x = 0;
```

现在你有了一种记住正方形在每个循环中所处位置的方式，可以一次性添加这三个过程（更新、清除、绘制）。在animate函数内部的setTimeout前面添加以下代码：

```
x++;
context.clearRect(0, 0, canvasWidth, canvasHeight);
context.fillRect(x, 250, 10, 10);
```

第一行代码用于更新正方形的*x*位置，每循环一次该值增加1，变量后面的两个加号表示在现有值上增加1。该行代码也可通过以下方式来表示：

```
x = x + 1;
```

第二行代码是清除过程，可以将画布上的对象有效地擦除干净，为第三行代码绘制正方形做好准备。第三行代码的作用是实现最后一个过程。将变量*x*添加到fillRect方法调用中，说明绘制正方形始终从当前的*x*位置开始，该值始终在增加（如图6-2所示）。

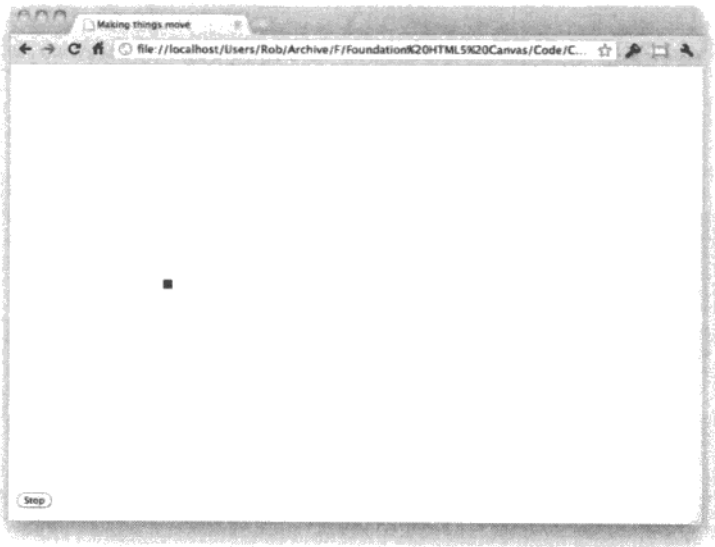


图6-2 通过基本动画移动一个正方形

如果一切正常，你应该看到一个黑色的小正方形在画布上移动。单击Stop按钮，该正方形

将停止运动；单击Start按钮，该正方形会重新移动。现在你已经在画布中创建了第一个动画。棒极了！

6.3　记忆要绘制的形状

在创建动画循环时，需要克服的一个主要问题是：准确记忆要绘制的对象的内容及位置。上一节已经简要涉及了该内容，我们采用的方法是：在循环外部设置一个变量来保存需要绘制的对象的位置值，并通过fillRect调用和在循环内部使用位置变量，就可以获取需要绘制的对象的内容及位置信息。但是，如果需要绘制多个动画形状该怎么办呢？甚至在创建循环时你都不知道需要创建多少种动画形状又该怎么办？因此，我们需要采用一种更好的方法。

6.3.1　错误的方法

你也许试图在动画循环外部使用独立的变量来存储每种形状的位置值。为什么不可以呢？既然这种方法适用于一种形状，那么为什么不适用于多个形状呢？你试图采用的方法是正确的，这样做可以达到目的，但这种方法非常笨拙，需要复制大量代码，并且以后修改这些形状也会非常复杂。

以下代码展示了如何通过修改上一节的代码来使多个形状产生动画效果。

```
var firstX = 0;
var secondX = 50;
var thirdX = 100;

function animate() {
        firstX++;
        secondX++;
        thirdX++;

        context.clearRect(0, 0, canvasWidth, canvasHeight);

        context.fillRect(firstX, 50, 10, 10);
        context.fillRect(secondX, 100, 10, 10);
        context.fillRect(thirdX, 150, 10, 10);

        if (playAnimation) {
                setTimeout(animate, 33);
        };
};
animate();
```

这里特意把这段代码单独列出来，而忽略了其他代码（例如，按钮事件），因为目前我们只需关注这部分内容。

这个例子和前面例子的唯一不同之处在于代码中使用了3次位置变量，并3次调用了fillRect方法。但是，没有绝对的理由能够说明这段代码不起作用，代码的运行结果如图6-3所示。

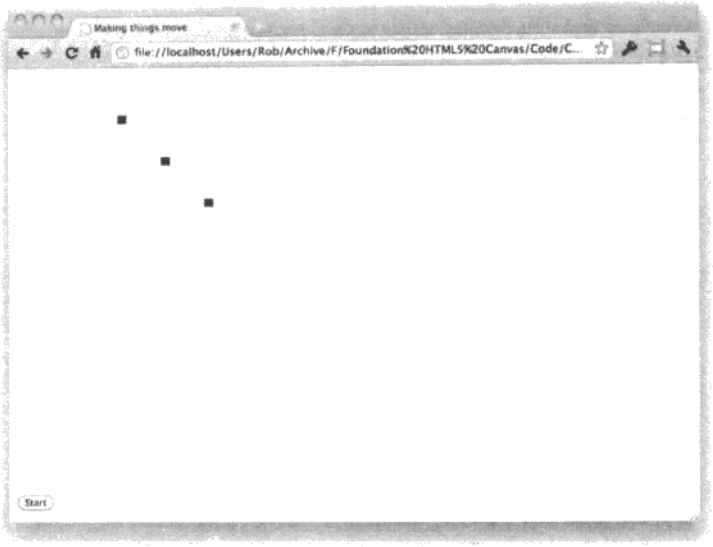


图6-3 使用错误的方法记忆多个形状

这种方法在此处也许很管用，但是，如果需要绘制上百个形状该怎么办呢？你会一本正经地告诉我，你会编写上百个位置变量并上百次调用`fillRect`方法吗？当然不会！因为代码能帮你自动地让对象产生动画效果，这才是使用代码创建动画的真正原因。

6.3.2 正确的方法

既然使用上面的方法创建多个变量既冗长又复杂，那我们该如何使其简化呢？简而言之，可以考虑使用对象和数组来实现。实际上，需要解决的问题有两个：第一，不管形状的数量有多少，首先考虑如何存储每个形状的位置值；第二，在不复制代码的情况下如何绘制每个形状。

第一个问题的解决方法非常简单，因此我们将直接介绍第二个问题的解决方法。你已经知道了每个形状所需的位置数据，即*x*的值。但是，我们还需要进一步深入研究，考虑同时使用*y*的值。如果知道每个形状具有两个相同类型的位置值，就可以通过创建JavaScript类来创建形状对象，代码如下所示：

```
var Shape = function(x, y) {
        this.x = x;
        this.y = y;
};
```

如果你忘记了对象的工作原理，请参考本书第2章的内容。这个概念非常重要，你必须掌握它，本书在后面讨论游戏的创建时也经常需要使用它。简言之，通过使用对象的属性（例如，火

箭上的机翼数目）和方法（例如，启动火箭的发动机），可以为某些对象定义模板。

但是，仅仅定义对象还不够，还需要通过某种方式来存储对象，这样就不必手动引用上百种形状了。为此，可以在数组中存储形状对象，这样能够把它们按顺序存储在同一个变量中：

```
var shapes = new Array();

shapes.push(new Shape(50, 50));
shapes.push(new Shape(100, 100));
shapes.push(new Shape(150, 150));
```

向数组中添加形状对象时，可以使用Array对象的push方法。这听起来也许很复杂，其实该方法就是在数组的末尾添加对象。在本例中，该对象是一个形状对象。其功能与以下代码完全相同：

```
var shapes = new Array();

shapes[0] = new Shape(50, 50);
shapes[1] = new Shape(100, 100);
shapes[2] = new Shape(150, 150);
```

使用push方法的好处是，无须知道数组中最后一个元素的序号，该方法将会自动在数组的末端添加对象。这太好了！

现在就得到了一组形状，每个形状都有不同的x和y值，这些值存储在一个数组中（这个数组已经被赋给shapes变量）。接下来的任务是如何将这些形状从数组中取出来并更新它们的位置（使它们产生动画效果），然后绘制这些形状。为此，需要在动画循环内部设置一个for循环：

```
function animate() {
        context.clearRect(0, 0, canvasWidth, canvasHeight);

        var shapesLength = shapes.length;
        for (var i = 0; i < shapesLength; i++) {
                var tmpShape = shapes[i];
                tmpShape.x++;
                context.fillRect(tmpShape.x, tmpShape.y, 10, 10);
        };
        if (playAnimation) {
                setTimeout(animate, 33);
        };
};
```

for循环遍历了数组中的每个形状，并将形状赋给了tmpShape变量，因此很容易访问它。现在你已经有了对数组中当前形状的引用，下面只需更新形状的x属性，然后使用x和y属性在当前位置绘制形状就可以了。

以下是完整的代码，它和上一节示例所产生的动画效果是完全相同的（如图6-4所示）：

```
var canvasWidth = canvas.width();
var canvasHeight = canvas.height();
```

```
var playAnimation = true;

var startButton = $("#startAnimation");
var stopButton = $("#stopAnimation");

startButton.hide();
startButton.click(function() {
        $(this).hide();
        stopButton.show();

        playAnimation = true;
        animate();
});

stopButton.click(function() {
        $(this).hide();
        startButton.show();

        playAnimation = false;
});

var Shape = function(x, y) {
        this.x = x;
        this.y = y;
};

var shapes = new Array();

shapes.push(new Shape(50, 50, 10, 10));
shapes.push(new Shape(100, 100, 10, 10));
shapes.push(new Shape(150, 150, 10, 10));

function animate() {
        context.clearRect(0, 0, canvasWidth, canvasHeight);

        var shapesLength = shapes.length;
        for (var i = 0; i < shapesLength; i++) {
                var tmpShape = shapes[i];
                tmpShape.x++;
                context.fillRect(tmpShape.x, tmpShape.y, 10, 10);
        };
        if (playAnimation) {
                setTimeout(animate, 33);
        };
};

animate();
```

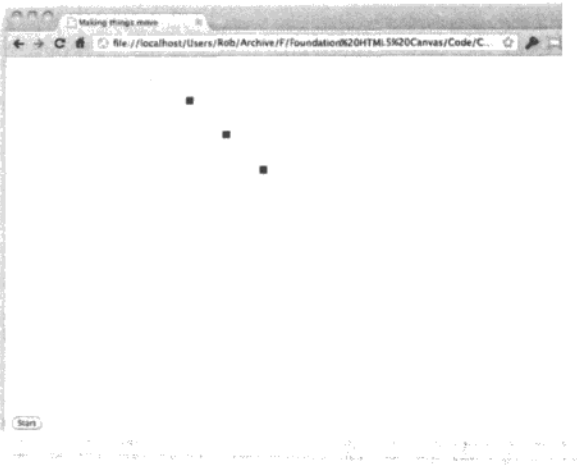


图6-4　使用正确的方法记忆多个形状

6.3.3　随机产生形状

现在已经可以采用一种快速而简单的方法来创建形状了，接下来就是如何产生随机形状。因为正在使用对象，所以解决这个问题非常简单。首先，需要更改Shape类来定义形状的宽度和高度：

```
var Shape = function(x, y, width, height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
};
```

然后，为每个形状随机选取起始位置和大小，并使用以下代码替换shapes.push语句：

```
for (var i = 0; i < 10; i++) {
        var x = Math.random()*250;
        var y = Math.random()*250;
        var width = height = Math.random()*50;
        shapes.push(new Shape(x, y, width, height));
};
```

width和height变量的赋值语句是一个双赋值语句，看上去也许有些奇怪，但它可以为这两个变量赋相同的值。另外，还需要更改对fillRect方法的调用，以便采用新的宽度和高度：

```
context.fillRect(tmpShape.x, tmpShape.y, tmpShape.width, tmpShape.height);
```

就这么简单。使用类和对象的优点在于，无须编辑太多的代码来添加或删除新特性。如果需要通过新示例来验证该特性，可以选择10个不同大小和位置的形状，并让它们在浏览器上移动（如图6-5所示）。

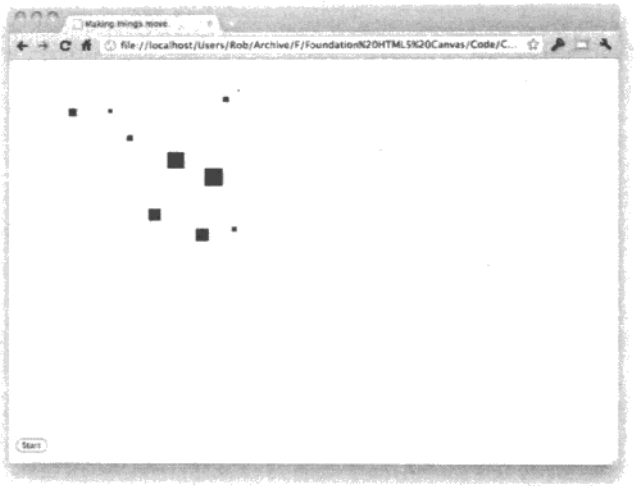


图6-5 随机设置图形的大小和位置

6.4 改变方向

到目前为止，我们已经介绍了如何创建动画，但还没有讨论如何控制形状动画的方式。我觉得只有直线动画会让人觉得很枯燥，不知你是否也这样想。

你已经知道了如何让一个形状向右移动（把*x*的值增加），但是如果需要改变运动的速度又该如何实现呢，或者如何改变动画的方向呢？非常简单：只需要增加（或减少）*x*和*y*值就可以了。如果你使用与前面示例相同的代码，按以下方式修改`tmpShape.x++`语句，就可以非常方便地使形状沿着向右的对角线方向运动：

```
tmpShape.x += 2;
tmpShape.y++;
```

与前面代码的不同之处在于，上面的代码将*x*值每次增加2，而不是增加1，并将*y*值每次增加1。这样产生的效果是，在每次动画循环中，每个形状向右移动2像素，并向下移动1像素，即为向右的对角线方向移动（如图6-6所示）。

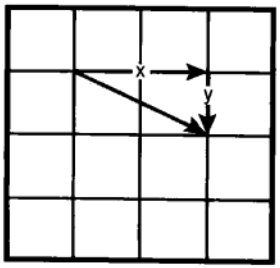


图6-6 让形状沿着对角线运动

或者还可以实现一些非常有趣的效果。例如，在每个动画循环中将*x*和*y*值设为随机值。这样产生的动画效果就具有不可预测性和无序性，形状将表现为不规则的运动形式。这听上去可能有不太合理，但这种方法可以让对象的运动更加生动自然：

```
tmpShape.x += Math.random()*4-2;
tmpShape.y += Math.random()*4-2;
```

以上代码的作用是产生一个介于0到4之间的随机数（Math.random产生一个0到1之间的数，然后将该数乘以4），然后减去2得到一个介于-2到2之间的随机数。通过这种方法，形状可以向右运动（*x*值为正数）、向左运动（*x*值为负数）、向上运动（*y*值为正数）和向下运动（*y*值为负数）。

如果你在浏览器中尝试该方法，形状将会前后随机运动，并出现摆动现象。太神奇了！

6.5　圆周运动

形状不一定始终沿着直线运动。如果你需要的动画效果是沿着圆周运动，例如，沿着圆形轨道运行（如图6-7所示），该如何实现呢？好消息是，这是完全可以实现的，并且不需要使用太多代码。坏消息是，这里需要使用三角函数的相关知识，可能需要你稍微动一下脑筋。

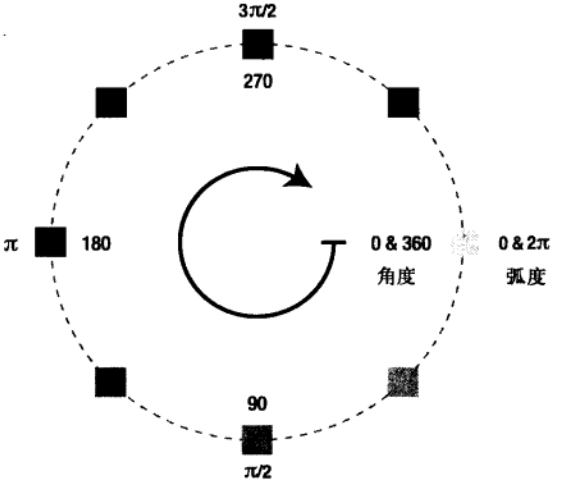


图6-7　使形状动画沿着圆形轨道运动

概念非常简单：将一个形状放在圆周的边缘处（它的周长上），以圆周的任意位置作为起点。但为了简单起见，可以将形状放在周长上角度为0弧度的位置，该位置位于右手边（如图6-7所示）。在每次动画循环中，只需要增加位于圆周上的形状的角度，就可以使形状沿着圆周运动。这非常简单，接下来我们具体讨论如何实现。

6.5.1 三角函数

需要解决的问题是：如何计算位于圆周上的形状的(x,y)坐标值（如图6-8所示）。这听上去也许很深奥，但需要解决的问题其实很简单。当然，只有用正确的方式来考虑需要解决的问题，才会觉得它容易。

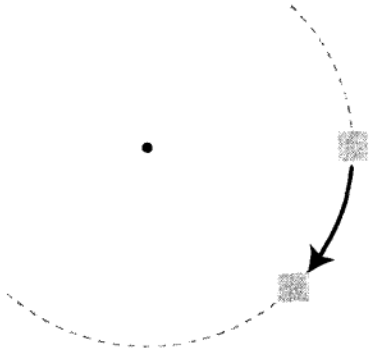


图6-8 确定圆周上某个位置对应的坐标值

在解决问题之前，首先需要知道圆的实际大小。可以选择任意大小的圆周，毕竟，这里只是示例，所以实际大小并不重要。重要的是可以通过半径（从圆心到圆周的长度）来描述圆的大小。如果画出运动轨道所在圆周的半径，那么你会发现形状移动的角度遵循一种有趣的模式（如图6-9所示）。

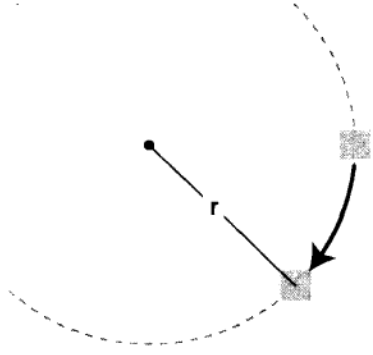


图6-9 画出半径，突出了一种有趣的模式

发现了吗？如果你认真地看，或者稍微思考一下，也许就会发现这种模式。如果幸运的话，你会发现三角形的边存在一些规律。如果没有发现规律，也不要紧。如果你是第一次接触这种问题，那么稍微发挥一下想象力就可以了。图6-10中有这里要讨论的三角形。

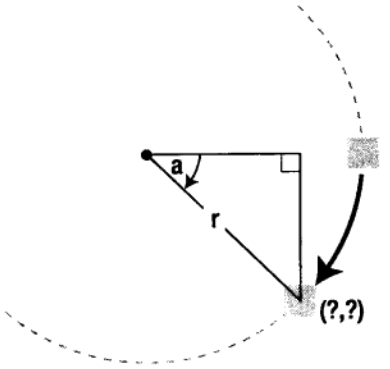


图6-10　半径是圆周内直角三角形的一条边

好，圆周中包含了一个三角形。但它有何用处呢？这个三角形能够提供一些准确的信息，帮助你计算形状沿圆周移动到新位置处的(*x*,*y*)坐标值。更具体地说，现在得到了一个三角形和两个角度（沿圆周转动的角度和三角形的90度直角），接下来可以构造一些基本三角形来计算你需要的值。这也体现了数学的重要作用。但是，在真正解决问题之前，我还要简要解释一下三角函数的原理。

三角函数的基本要点是：如果已知一个三角形的一个角是90度，并且已知另外一个角，那么就可以计算三角形的边长之间的比值。然后，可以通过该比值来计算边的长度，边的长度单位是任意的，本示例中边的单位是像素。因此，你需要知道三角形的哪条边是需要计算的长度，因为它们分别对应着不同的三角函数规则。这三条边分别是斜边（最长的边）、邻边（与除直角以外的已知角相邻的边）和对边（与已知角相对的边）。图6-11详细标注了这些边。

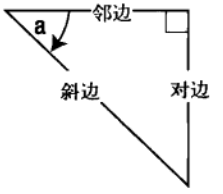


图6-11　描述直角三角形的三条边

要计算边之间的比值，需要用3种三角函数：正弦函数（sin）、余弦函数（cos）或正切函数（tan）。正弦函数是对边与斜边的比值，余弦函数是邻边与斜边的比值，正切函数是对边与邻边的比值（如图6-12所示）。你也许听过把这些函数叫做SOH-CAH-TOA，其实这就是代表正弦-对边-斜边、余弦-邻边-斜边、正切-对边-邻边。通过把三角形中的已知角代入正确的函数，可以计算出所需的比值来。

$$\sin(a) = \frac{\text{对边}}{\text{斜边}} \quad \cos(a) = \frac{\text{邻边}}{\text{斜边}} \quad \tan(a) = \frac{\text{对边}}{\text{邻边}}$$

图6-12 SOH-CAH-TOA公式

在此，我们需要知道三角形的邻边和对边的长度，它们分别代表*x*和*y*的位置（如图6-13所示）。要计算这些边的长度，首先需要在对应的三角函数中通过已知角计算比值。在JavaScript中，可以使用Math对象来计算这些比值：

```
var angle = 45;
var adjRatio = Math.cos(angle*(Math.PI/180)); // 余弦-邻边-斜边
var oppRatio = Math.sin(angle*(Math.PI/180)); // 正弦-对边-斜边
```

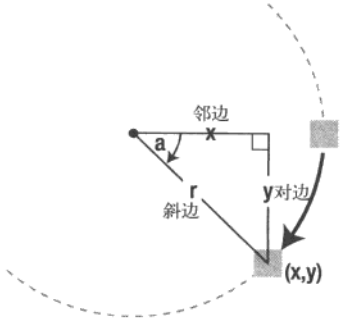


图6-13 通过三角函数计算三角形对应边的比值

你会注意到，Math对象的cos和sin方法中执行了一些简单的计算过程。这种计算是为了将角从角度转换为弧度，因为JavaScript使用的单位是弧度（请参考第3章）。如果你在开始就使用弧度制，就不需要做任何转换了。

得到这些比值仅仅完成了一半的工作量。另外一半工作才是最终我们需要得到的答案，将这些比值与斜边（因为它是半径，所以长度已知）的长度相比较，如图6-14所示。最终的答案可以由半径乘以该比值得到，即：

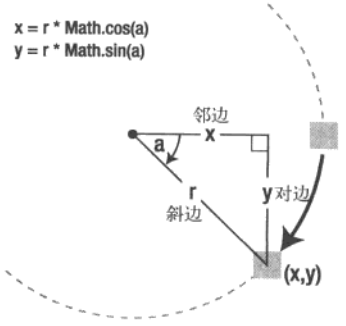


图6-14 通过每条边的比值计算坐标值

```
var radius = 50;
var x = radius * adjRatio;
var y = radius * oppRatio;
```

6.5.2　综合运用

既然你能够计算位于圆周上某个角度的形状对应的(*x*,*y*)坐标值，那么把这些结果综合应用于当前的示例就非常简单了。第一步是更新Shape类，并向其中添加几个新属性：

```
var Shape = function(x, y, width, height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;

        this.radius = Math.random()*30;
        this.angle = 0;
};
```

这两个属性用于设置起始角度和计算圆周的随机半径（介于0~30之间）。倒数第二步是使用以下代码替换动画循环中的现有代码，从而更新形状：

```
var x = tmpShape.x+(tmpShape.radius*Math.cos(tmpShape.angle*(Math.PI/180)));
var y = tmpShape.y+(tmpShape.radius*Math.sin(tmpShape.angle*(Math.PI/180)));

tmpShape.angle += 5;
if (tmpShape.angle > 360) {
        tmpShape.angle = 0;
};
```

前两行代码没有什么新内容，它们分别用于计算位于圆周上当前角度的形状所对应的*x*和*y*值，其中圆周是通过半径来定义的。这里的*x*和*y*值能够提供坐标值(假设圆周中心的坐标为(0,0))，因此，当将*x*和*y* 值添加到形状中对应的点（*x*, *y*）时，就可以把形状移动到正确的位置。注意，形状对象中定义的点（*x*, *y*）现在引用的是圆周的中心——形状围绕它旋转的点，而不是形状的起点。最后几行代码用于在每个动画循环中增加角的度数，如果角度超过360度（一个完整的圆），则将角度重新设置为0度。

最后，将新的*x*和*y* 变量添加到fillRect方法中：

```
context.fillRect(x, y, tmpShape.width, tmpShape.height);
```

如果一切运行正常，就可以选择不同的形状，让它们沿着不同的圆周运动（如图6-15所示）。你也许永远也不会想到，这么简单的效果居然需要这么复杂的方法来实现。

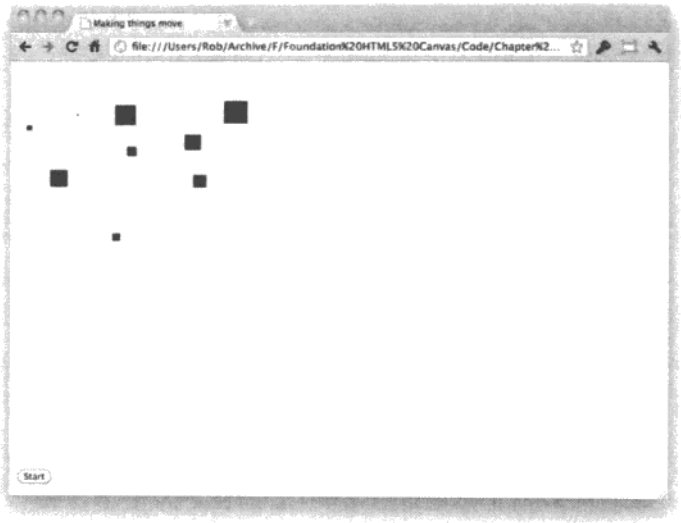


图6-15 选择不同的形状围绕圆形轨道运动

本节可能有些难度，以下是完整的代码供你参考。

```
var canvas = $("#myCanvas");
var context = canvas.get(0).getContext("2d");

var canvasWidth = canvas.width();
var canvasHeight = canvas.height();

var playAnimation = true;

var startButton = $("#startAnimation");
var stopButton = $("#stopAnimation");

startButton.hide();
startButton.click(function() {
        $(this).hide();
        stopButton.show();

        playAnimation = true;
        animate();
});

stopButton.click(function() {
        $(this).hide();
        startButton.show();
        playAnimation = false;
});

var Shape = function(x, y, width, height) {
        this.x = x;
```

```
        this.y = y;
        this.width = width;
        this.height = height;

        this.radius = Math.random()*30;
        this.angle = 0;
};

var shapes = new Array();

for (var i = 0; i < 10; i++) {
        var x = Math.random()*250;
        var y = Math.random()*250;
        var width = height = Math.random()*30;
        shapes.push(new Shape(x, y, width, height));
};

function animate() {
        context.clearRect(0, 0, canvasWidth, canvasHeight);

        var shapesLength = shapes.length;
        for (var i = 0; i < shapesLength; i++) {
                var tmpShape = shapes[i];

                var x =
tmpShape.x+(tmpShape.radius*Math.cos(tmpShape.angle*(Math.PI/180)));
                var y =
tmpShape.y+(tmpShape.radius*Math.sin(tmpShape.angle*(Math.PI/180)));

                tmpShape.angle += 5;
                if (tmpShape.angle > 360) {
                        tmpShape.angle = 0;
                };
                context.fillRect(x, y, tmpShape.width, tmpShape.height);
        };
        if (playAnimation) {
                setTimeout(animate, 33);
        };
};

animate();
```

6.6　反弹

也许你还没有注意到，到目前为止我们使用的示例其实都没有边界。也就是说，当形状移动到画布的边界处时，什么都没发生，它们只是消失在我们的视野中，再也看不见了。这也许是你需要的效果。例如，如果你只是创建一段简短的动画，并且动画在到达边界之前就会停止，或者你希望形状移动到画布之外。但是，如果你不需要这种行为怎么办？如果你希望形状能够感知周围的环境，或者在边界处反弹回来怎么办呢？这种行为可以避免机械性的动画，使动画更加自然和随机。

在学习如何实现这种行为之前，先用本章讨论的技术编写如下代码：

```
var canvasWidth = canvas.width();
var canvasHeight = canvas.height();

var playAnimation = true;

var startButton = $("#startAnimation");
var stopButton = $("#stopAnimation");

startButton.hide();
startButton.click(function() {
        $(this).hide();
        stopButton.show();

        playAnimation = true;
        animate();
});

stopButton.click(function() {
        $(this).hide();
        startButton.show();

        playAnimation = false;
});

var Shape = function(x, y, width, height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
};

var shapes = new Array();

for (var i = 0; i < 10; i++) {
        var x = Math.random()*250;
        var y = Math.random()*250;
        var width = height = Math.random()*30;
        shapes.push(new Shape(x, y, width, height));
};

function animate() {
        context.clearRect(0, 0, canvasWidth, canvasHeight);

        var shapesLength = shapes.length;
        for (var i = 0; i < shapesLength; i++) {
                var tmpShape = shapes[i];
                context.fillRect(tmpShape.x, tmpShape.y, tmpShape.width,
tmpShape.height);
        };

          if (playAnimation) {
                  setTimeout(animate, 33);
          };
};
```

```
animate();
```

这些代码建立了一个完整的动画循环，该循环将遍历10个随机生成的形状。实际上，代码并没有在视觉上移动任何形状，因为我们没有修改动画循环中形状的属性（如，增加x的值将形状向右移动）。

使形状感知画布边界的过程其实非常简单。假设一个形状在每个循环中向右移动1像素。一旦该形状移动到画布的右边界处（假设是500像素），它将会继续移动，并且x值仍在增加，但我们就无法在画布上看到它了。其实你希望该形状发生的行为是：形状在画布的右边界处反弹回来，就好像边界处有一堵墙一样。为此，你需要检查形状是否超过了画布的右边界，如果已经到达边界处，则反向改变形状运动的方向，这样它就会反弹回来。

图6-16描述了这种效果的基本概念，运动的球上的数字代表每个动画循环。左边的图显示了没有边界时形状的正常反应，右边的图显示了存在边界时形状会如何反应。

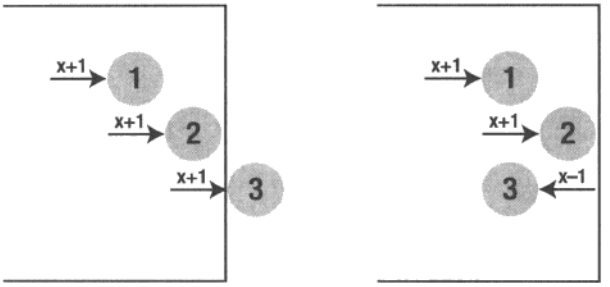


图6-16　形状从边界弹回的示意图

计算一个形状是否超过画布的右边界其实就是检查形状的x位置是否超过了画布的宽度。如果形状的x位置大于画布的宽度，那么形状必然会超出右边界。同样，检查形状是否超过画布的左边界也可以采用这种方法。其中，形状的左边界的位置对应的x值为0。检查形状的x位置是否小于0，就可以确定形状是否位于画布的左边界之外。当然，也可以使用同样的方法检查形状是否位于画布的上边界和下边界。具体做法是，检查 y 值是否小于上边界0，并检查 y 是否大于画布的高度（下边界）。

综合运用这些方法，可以创建一组简单的逻辑：让形状在画布的边界处弹回。第一步是向Shape类中添加一些新属性，它们将用于定义形状是否碰到边界及反弹的路径方向：

```
this.reverseX = false;
this.reverseY = false;
```

默认情况下，这些属性的值为false，在本示例中，这表明形状将一直向右下方运动。下一步是添加逻辑关系来检查形状是否超出了画布边界。在动画循环的fillRect调用下面插入以下代码：

```
if (tmpShape.x < 0) {
        tmpShape.reverseX = false;
} else if (tmpShape.x + tmpShape.width > canvasWidth) {
        tmpShape.reverseX = true;
```

```
};

if (tmpShape.y < 0) {
        tmpShape.reverseY = false;
} else if (tmpShape.y + tmpShape.height > canvasHeight) {
        tmpShape.reverseY = true;
};
```

当形状即将到达边界之外时，这些检查将反向改变形状的运动路线。但是，设置布尔值并不能实际改变形状的具体运动方向，因此，需要另外进行一些检查。此时，需要将它们放在fillRect调用的上面：

```
if (!tmpShape.reverseX) {
        tmpShape.x += 2;
} else {
        tmpShape.x -= 2;
};

if (!tmpShape.reverseY) {
        tmpShape.y += 2;
} else {
        tmpShape.y -= 2;
};
```

这将会产生一些神奇的效果。如果形状在x轴上没有反转，那么这些检查将会使形状向右移动（通过增加x位置）。如果形状在x轴上反转，那么这些检查将会使形状向左移动（通过减少x位置）。同样，在y轴上也可以执行相同的检查。

由于有了这些相对简单的逻辑检查，你可以使一组形状移动到画布的边界时反弹回来（如图6-17所示）。甚至可以更改Shape类表示反转方向的属性的默认值，从而改变形状的运动方式。

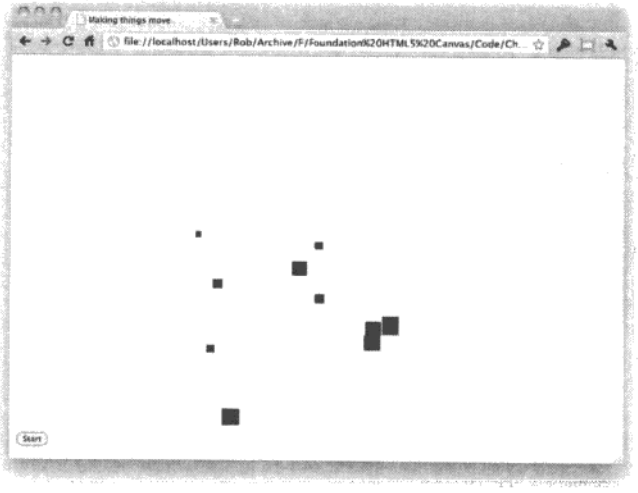


图6-17 一组在边界处弹回的形状

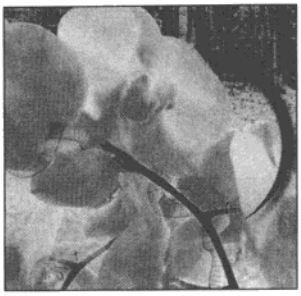
第9章的侧卷轴游戏（side-scroller game）中实现的一些非常出色的功能其实就是使用类似简单的检查来完成的。

6.7　小结

本章不仅专门学习了画布的有关知识，更多关注的是理论知识和动画技术。通过学习，你已经了解了动画的工作原理，包括循环以及记忆形状的属性。另外，还学习了如何移动形状、如何改变动画的方向，以及如何使形状围绕圆周运动。最后，本章还介绍了如何让形状在画布的边界处弹回，并展示了形状运动时的基本智能和物理性质。

通过本章的学习，希望大家能认识到，一些有趣而复杂的视觉效果可以通过非常简单的理论来实现，且不需要多少代码。动画的本质（特别是在画布中）常常是采用最简单的选项，只要它的视觉和感觉效果达到你的目标就可以了。这也正是我们将在下一章要探讨的原则。下一章将介绍利用物理知识实现高级动画。太棒！

第7章



实现高级动画

本章将学习如何通过物理知识把动画效果进一步提高，并复习一些与现实中物体运动有关的物理概念，以及如何在JavaScript中应用这些概念。最后，本章将探讨物体碰撞检测的一些理论，并讨论物体碰撞后相互弹开的原因。

这是在开始设计游戏之前学习的最后一章内容，可能会有一些难度，但我相信你一定会对它感兴趣；而且，这些方法产生的神奇效果将会给你留下难忘的印象。

7.1 物理常识

如果你和我有类似的想法，那么当听到“物理”这个词时，应该立刻会联想到中学时学习的那些乏味的数学和理科课程。我认为这些课程存在的主要问题是，它们没有让我们对数学和物理产生足够的兴趣。为什么要关注苹果从树上落下来的时间？计算圆球扔向空中所形成的轨迹有何意义？在物理中，了解汽车加速、减速或移动有何用处？过去我一直对这些问题迷惑不解，但在课堂上所学习的示例却从来没有涉及这些问题，更别说将这些概念应用到现实生活中了。我觉得自己完全是被动地接受数学和理科教育，没有特定的学习目标。

10年转眼即过，事情发生了奇怪的转变。很久以前那个年轻的我虽然对数学和理科感兴趣，但却从来不知道为什么需要它们。而现在，我已经深深迷恋上了它们。更具体地说，我已经对物理产生了浓厚的兴趣，并痴迷于它无限的可能性。这不是撒谎，我虽然不是物理专家，但觉得它真的很有趣，并且我很希望你也对它产生兴趣。其中的秘密就是将它运用于现实生活中；也就是说，你要带着特定的目标去学习。对你而言，这种目标就是学习如何使用JavaScript来使物体产生更逼真的动画效果。看，这听上去比那些古板的数学课程要有趣多了！

因此，我们将详细介绍一些物理学方面的知识，首先让我们从物理学的一般概念开始。

7.1.1 什么是物理学

简而言之，物理学是一门研究物质和能量的自然科学。具体来说，物理学的目的是研究自然世界的原理，旨在理解事物以某种状态存在的方式及原因。目前我们关注的是一些直接与物体运

动有关的物理学知识，研究哪些因素会影响物体的运动。物理学的研究分支非常多，考虑到实际情况，我们将从基本的概念开始介绍。

7.1.2　物理学对创建动画有何作用

理解了物理学，你就能够创建更加逼真和具有动态效果的动画。也许现在你还无法明显感受到这一点。无论是使用速度、加速度这些简单概念，还是构建一个包含不同方向作用力的完善系统，通过学习物理学，我们才能真实而全面地展现动画效果。物理学概念不仅是制作逼真动画的基础，它们还是在各种动画（从游戏和数据可视化一直到卡通和电影）中经过反复验证的概念。

7.1.3　基本概念

学习物理的关键是了解一些基本术语和单位，因为在一些较复杂的概念中通常要使用它们。虽然你不一定会用到所有术语，但简要了解它们有益于日后学习。这样，在后面的代码中用到这些术语时，你会更明确它们的意义。

1. 力

力是作用于物体的推力或拉力。它会使物体改变速率、方向或形状。力既有数量（大小或距离）也有方向，这说明可以把力看作矢量（vector）。力的单位是牛顿（N）。

2. 矢量

矢量是具有大小和方向的量（如力）。矢量可以用图形来表示，通常用一条从原点出发并指向终点的线段来表示。其中，长度代表矢量的大小，箭头表示矢量的方向。

3. 质量

质量是一种阻碍物体在力的作用下加速的物理量，也用来衡量惯性。当力作用于物体时，质量将直接影响加速度的大小。例如，相同的力作用于质量不同的两个物体，则质量较大的物体的加速度小于质量较小的物体的加速度。质量的单位是千克（kg）。

4. 重力

物体的质量受另一个物体引力的作用而产生的力称为重力。重力是物体存在重量的原因。重力的计算方法是：将物体的质量乘以地球引力。因此，地球表面的物体比高空中相同的物体要重一些——高空中的地球引力稍微小一些。在地球表面，重力通常使用质量的单位（kg），但实际应该使用力的单位：牛顿（N）。

5. 摩擦力

摩擦力是一种阻止一个物体沿着另一个物体表面运动的力。正是由于摩擦力的存在，相对于地毯之类的物体，冰才显得更光滑一些。

6. 速度

速度是指物体运动的方向和速率。它是一个矢量，通常用米/秒（m/s）作为单位，并附加一

个方向（例如，向西）。平均速度的计算方法是：用物体的速率除以物体运动的时间。

7. 速率

速率是指物体运动速度的大小，表示物体在一段时间内通过的距离。它没有方向（标量），通常用米/秒（m/s）、千米/小时（km/h）、英里/小时（mph）作为单位。平均速率的计算方法是：用物体通过的距离除以物体运动的时间。

8. 加速度

加速度是指物体的速度随时间变化的比值，它既有大小又有方向。加速度通常指速度增加，减速度通常指速度减慢。一个物体可以有速率但可以无加速度，因为加速度是相对于物体的先前速度而言的。例如，如果物体在匀速运动，它没有加速（速度没有变得更快），但它仍然具有速度。加速度是一个矢量，它通常用米/秒^2（m/s^2）作为单位。加速度是运用牛顿第二运动定律计算出来的，我们将在下一节具体介绍。

7.1.4 牛顿运动定律

艾萨克·牛顿发现用三条简单定律来描述运动的性质是一种绝妙的想法。利用这三条定律可以计算出力是如何影响物体的运动的，当需要创建自然流畅的动画时，了解力对物体运动状态的影响是非常有用的。所以，我们要感谢艾萨克·牛顿！

1. 第一运动定律

如果物体不受力的作用，那么它将保持匀速直线运动状态。如果物体处于静止状态，那么它将保持这种状态。简言之，只有外力作用于物体时，物体才会运动。

2. 第二运动定律

力作用于具有一定质量的物体时，将会产生与力的方向相同的加速度，并且加速度的大小与力的大小成正比（随着力的增大而增加），与物体的质量成反比（随着质量的增大而减小）。这就是非常著名的公式：F=ma，即力等于质量乘以加速度。另外，这个公式也可以用于计算物体的质量和加速度。

3. 第三运动定律

任何作用力都有一个与之相等的反作用力。还记得吗？这也许是牛顿三大定律中最著名的一条。它实际说明的是，如果一个物体（A）作用于另外一个物体（B），那么物体B将会以大小相等、方向相反的力作用于物体（A）。但是，这并不能说明两个物体将会获得相同的加速度，因为我们通过牛顿第二运动定律已经知道，只有两个物体的质量相同时才会获得相同的加速度。

当然，还有很多物理知识需要复习，但我担心太多复杂的内容会把你吓跑。其实，物理学中有很多有用的概念和思想，它们可以帮助你寻求更加逼真的动画及模拟效果。我们在第8章和第9章中讨论创建游戏时，将会使用这里介绍的有关概念。接下来介绍如何把这些概念运用于动画中。

> **注意**　如果你需要学习更多相关的物理学知识，请查找维基百科网页的相关学科领域[①]。有些概念只需要少量时间就可以理解，因此我的建议是，尝试把你所学到的知识在JavaScript中实际应用。我一直认为只有亲自把相关概念运用到现实之中，才会真正领悟其中的真谛。

7.2　运用物理知识创建动画

在运用物理学核心概念创建一些神奇的动画效果之前，首先需要掌握这些概念，这非常重要。本节将创建一个简单的太空场景，其中在太空深处有一个动态的小行星群。这可不是骗你，其中的场景将非常奇妙。

7.2.1　准备工作

如果你还没有这样做，那么现在请你把JavaScript代码从HTML页面分离出来，并保存为一个单独的JavaScript文件。在第2章中，我们已经讨论了如何将JavaScript代码保存为一个单独的文件，这有助于保持文件的条理性。本节开头的JavaScript代码并没有什么特别之处；实际上，它与上一章使用的代码几乎完全相同。主要的不同之处在于，上一章使用的对象是形状，而本章使用的对象是小行星。

将以下代码添加到一个外部JavaScript文件中，并用一个可读性强的名称为其命名，如asteroids.js：

```
$(document).ready(function() {
        var canvas = $("#myCanvas");
        var context = canvas.get(0).getContext("2d");

        var canvasWidth = canvas.width();
        var canvasHeight = canvas.height();

        $(window).resize(resizeCanvas);

        function resizeCanvas() {
                canvas.attr("width", $(window).get(0).innerWidth);
                canvas.attr("height", $(window).get(0).innerHeight);
                canvasWidth = canvas.width();
                canvasHeight = canvas.height();
        };

        resizeCanvas();

        var playAnimation = true;

        var startButton = $("#startAnimation");
        var stopButton = $("#stopAnimation");
```

① http://en.wikipedia.org/wiki/Index_of_physics_articles。

```
        startButton.hide();
        startButton.click(function() {
                $(this).hide();
                stopButton.show();
                playAnimation = true;
                animate();
        });

        stopButton.click(function() {
                $(this).hide();
                startButton.show();
                playAnimation = false;
        });

        var Asteroid = function(x, y, radius) {
                this.x = x;
                this.y = y;
                this.radius = radius;
        };

        var asteroids = new Array();

        for (var i = 0; i < 10; i++) {
                var x = 20+(Math.random()*(canvasWidth-40));
                var y = 20+(Math.random()*(canvasHeight-40));
                var radius = 5+Math.random()*10;

                asteroids.push(new Asteroid(x, y, radius));
        };

        function animate() {
                context.clearRect(0, 0, canvasWidth, canvasHeight);
                context.fillStyle = "rgb(255, 255, 255)";

                var asteroidsLength = asteroids.length;
                for (var i = 0; i < asteroidsLength; i++) {
                        var tmpAsteroid = asteroids[i];

                        context.beginPath();
                        context.arc(tmpAsteroid.x, tmpAsteroid.y,
tmpAsteroid.radius, 0,
Math.PI*2, false);
                        context.closePath();
                        context.fill();
                };
                if (playAnimation) {
                        setTimeout(animate, 33);
                };
        };
        animate();
});
```

下一步就是构造CSS文件，让画布的尺寸与浏览器窗口同宽。另外，还要使用CSS来移动Start和Stop按钮，因为如果不这样做，当画布占据整个窗口时，这些按钮将位于浏览器之外。

将包含以下代码的外部CSS文件和JavaScript文件放在相同的目录下，并用一个可读性强的名称为其命名，如canvas.css：

```
* { margin: 0; padding: 0; }
html, body { height: 100%; width: 100%; }
canvas { display: block; }

#myCanvas {
        background: #001022;
}

#myButtons {
        bottom: 20px;
        left: 20px;
        position: absolute;
}

#myButtons button {
        padding: 5px;
}
```

最后，你需要建立一个HTML文件将所有文件整合到一起。这与上一章建立的HTML文件完全相同，不过文件要小得多，因为所有的JavaScript代码都在一个外部文件中。注意这里新加了一个script元素调用JavaScript文件。如果你的文件名不叫asteroids.js，就需要更改调用文件的名称。

将包含以下代码的HTML文件和其他文件放在相同的目录下，并命名为index.html：

```
<!DOCTYPE html>

<html>
        <head>
                <title>Implementing advanced animation</title>
                <meta charset="utf-8">
                <link href="canvas.css" rel="stylesheet" type="text/css">
                <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/
jquery/1/jquery.min.js"></script>
                <script type="text/javascript" src="asteroids.js"></script>
        </head>

        <body>
                <canvas id="myCanvas" width="500" height="500">
                        <!-- 在此插入后备内容
                </canvas>
                <div id="myButtons">
                        <button id="startAnimation">Start</button>
                        <button id="stopAnimation">Stop</button>
                </div>
        </body>
</html>
```

如果使用现代浏览器加载该HTML页面，应该会看到一个蓝色的宽大背景（画布），在其左下角有一个Stop按钮和一群四散分布的小行星（如图7-1所示）。看到了吧？这太好了。现在，你

已经做好了成为一个物理大师的准备了。此刻，你应该能够感到自己正在变得越来越聪明！

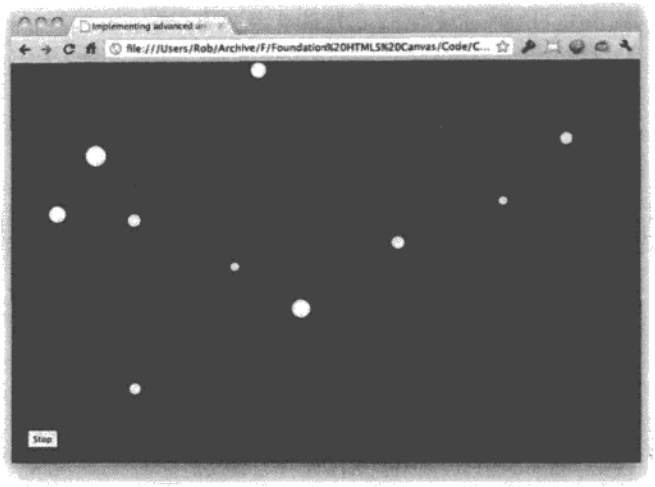


图7-1 建立基本的太空背景

最后再给你提个醒：单击Stop按钮可以让背景中的动画停下来，这你说了算。

7.2.2 速度

第6章介绍了通过增加或减少形状的 x 和 y 位置来移动形状。令人兴奋的是，同样的方法也可以赋予每个形状速度。请记住，速度包括物体的速率和方向。速率是指像素移动的数目，方向是指向左和向右（x）、向上和向下（y）。因此，你已经不知不觉使用了一些物理学知识！

上一章中的速度存在的问题是，它们要么是完全随机的，要么是完全相同的。因此，我们可以将这两种情况中和一下，让每颗小行星采用不同的飞行速度（请注意，在命名本章中的对象时，应该用asteroids替换上一章中的shapes）。为此，你需要在Asteroid类中定义两个新属性，代码如下：

```
var Asteroid = function(x, y, radius, vX, vY) {
        this.x = x;
        this.y = y;
        this.radius = radius;

        this.vX = vX;
        this.vY = vY;
};
```

通过添加vX和vY属性，现在每颗小行星可以拥有各自不同的速度。注意类函数中参数vX和vY的设置方法，当你创建一颗新的小行星时，可以通过这两个参数来设置速度。那么，接下来需要为每颗小行星设置不同的速度，速度定义了每个动画循环中小行星移动的像素数目。

为了在循环中创建所有的小行星，需要在radius变量下面添加以下代码：

```
var vX = Math.random()*4-2;
var vY = Math.random()*4-2;
```

另外，还需要使用以下代码替换radius变量下一行的代码，以便把新的速度作为参数传递给Asteroid类：

```
asteroids.push(new Asteroid(x, y, radius, vX, vY));
```

在本示例中，在*x*轴和*y*轴上同时将速度设置为一个介于–2到2之间的随机数。你已经知道，Math.random方法将会产生一个介于0到1之间的小数。因此，为了得到一个介于–2到2之间的数，需要分两步完成。第一步，将随机数乘以4，得到一个介于0到4之间的随机数。第二步非常简单，只需将该随机数减去2，这样将得到一个介于–2（0减2）到2（4减2）之间的数。你可以使用该方法计算介于任意范围内的随机数。

仅对代码进行这些修改，还不能改变小行星的速度。你还需要使用新的速度属性来更新每颗小行星的*x*和*y*位置。在动画循环中的tmpAsteroid变量声明下面添加以下代码：

```
tmpAsteroid.x += tmpAsteroid.vX;
tmpAsteroid.y += tmpAsteroid.vY;
```

这与在上一章中所做的工作基本上是相同的，小行星的当前位置增加了一个确定的像素数。不同之处在于，现在每颗小行星各自有了不同的速度，这说明它们将会以不同的速率（每个循环的像素数）和方向运动。有些动画看上去非常自然和流畅，其秘诀就在于实现了这种不同速度的运动。

刷新或加载该HTML文件，你应该会看到一群类似于小行星的物体在画布上运动（如图7-2所示）。继续刷新页面，可以看到这些小行星将从不同的位置出发，并以不同的速度运动。这太奇妙了！目前画布中还没有边界，当小行星运动到屏幕的边界快要消失时，你可以单击Stop按钮来留住它们。

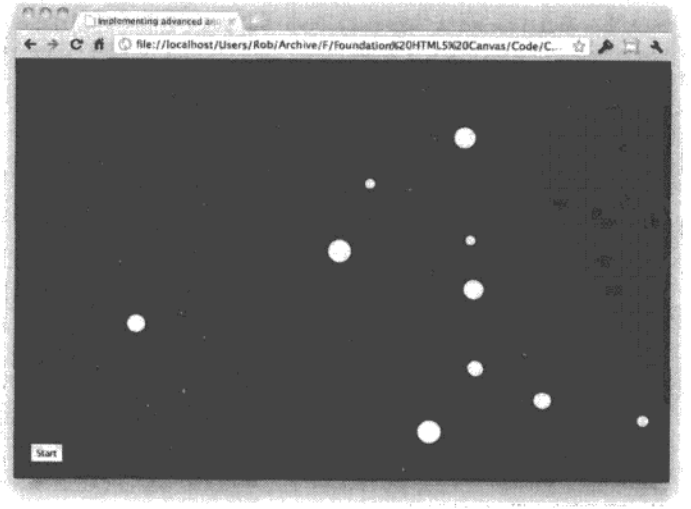


图7-2　建立小行星群

7.2.3 添加边界

在继续介绍相关内容之前，为了防止某些极端的小行星运动到画布之外，我们可以在画布上添加一个边界。所幸，实现该功能的代码与上一章我们使用的代码非常相似，要使用新的速度逻辑，只需对代码稍做修改即可。在动画循环中添加以下代码，防止小行星飞出画布之外：

```
if (tmpAsteroid.x-tmpAsteroid.radius < 0) {
        tmpAsteroid.x = tmpAsteroid.radius;
        tmpAsteroid.vX *= -1;
} else if (tmpAsteroid.x+tmpAsteroid.radius > canvasWidth) {
        tmpAsteroid.x = canvasWidth-tmpAsteroid.radius;
        tmpAsteroid.vX *= -1;
};

if (tmpAsteroid.y-tmpAsteroid.radius < 0) {
        tmpAsteroid.y = tmpAsteroid.radius;
        tmpAsteroid.vY *= -1;
} else if (tmpAsteroid.y+tmpAsteroid.radius > canvasHeight) {
        tmpAsteroid.y = canvasHeight-tmpAsteroid.radius;
        tmpAsteroid.vY *= -1;
};
```

在画布上绘制小行星之前，以上代码中的两个条件语句用于检查每颗小行星的位置。如果小行星的边界位于某个边界之外，那么它将向边界内部运动，并且其速度也改变为相反的方向。如果不改变小行星的运动方向，那么它要么停下来，要么完全飞出边界之外。因为这里用圆来代表小行星，因此 (x,y) 坐标位于圆形中心点。为此，你需要加上或减去圆的半径来计算边界处的x或y位置。

刷新该页面，你应该会看到一群小行星在四处飘荡，并在浏览器的边缘处弹回。现在，既然你已经学会了如何全面控制那些极端的小行星，接下来我们再讨论一些其他物理知识。

7.2.4 加速度

我们已经在7.1节中说过，加速度是速度在一段时间内的变化，也称为速率的增加。在小行星动画中添加加速度非常简单。实际上，这和添加速度几乎完全相同，因为加速度也包含大小和方向；大小指加速小行星的像素数目，方向指加速度沿 x 轴或 y 轴方向。

你需要让每个小行星拥有不同的加速度，因此第一步需要在Asteroid类中创建所需的属性，然后在构建每颗小行星时使用这些属性。在Asteroid类中添加以下代码：

```
this.aX = aX;
this.aY = aY;
```

就像前面对速度参数所做的操作一样，请务必向类函数中添加aX和aY参数。以下是Asteroid类最终的代码：

```
var Asteroid = function(x, y, radius, vX, vY, aX, aY) {
        this.x = x;
        this.y = y;
```

```
        this.radius = radius;

        this.vX = vX;
        this.vY = vY;
        this.aX = aX;
        this.aY = aY;
};
```

下一步是在创建小行星时使用这些新属性，因此在循环中创建小行星，并在速度变量之后添加以下代码：

```
var aX = Math.random()*0.2-0.1;
var aY = Math.random()*0.2-0.1;
```

通过以上两行代码，小行星将获得一个介于-0.1到0.1之间的加速度。你很快就会明白为什么用这么小的数值。

在循环中所做的最后一件事，是在new Asteroid调用中添加新的aX和aY变量作为最后面的参数，代码如下所示：

```
asteroids.push(new Asteroid(x, y, radius, vX, vY, aX, aY));
```

仅对代码做这些修改，还无法看到加速度的效果，因为你还需要将加速度应用到每个具体的小行星。应用加速度就和把加速度添加到物体的当前速度一样，非常简单。毕竟，加速度是物体速度的变化情况；也就是说，它是先前速度与当前速度之间的差值。

通过在动画循环中添加以下代码，将加速度应用到每个小行星。以下代码需要放在每颗小行星的速度代码（*x*和*y*位置）之后：

```
tmpAsteroid.vX += tmpAsteroid.aX;
tmpAsteroid.vY += tmpAsteroid.aY;
```

所有这些步骤都是通过加速度为每颗小行星增加速度，单位为像素。这并不会影响小行星的当前动画循环，但这意味着小行星在随后的循环中将会改变速度。

在让小行星加速之前，还需要在边界检查中添加几行代码。这样，当小行星碰到窗口的边缘时，边界检查将改变速度的方向，使物体沿着相反的方向运动。但加速度没有改变，因此当某颗小行星改变方向时，其加速度将逐步使小行星恢复为原来的方向。如果现在刷新浏览器，你就会明白我的意思。浏览器上显示的场景非常神奇，但并不是我们所期望的结果。

所幸，这个问题并不难解决，只需在改变速度方向的同时也改变加速度的方向就可以了。这将产生一系列边界检查，代码如下所示：

```
if (tmpAsteroid.x-tmpAsteroid.radius < 0) {
        tmpAsteroid.x = tmpAsteroid.radius;
        tmpAsteroid.vX *= -1;
        tmpAsteroid.aX *= -1;
} else if (tmpAsteroid.x+tmpAsteroid.radius > canvasWidth) {
        tmpAsteroid.x = canvasWidth-tmpAsteroid.radius;
        tmpAsteroid.vX *= -1;
        tmpAsteroid.aX *= -1;
};
```

```
if (tmpAsteroid.y-tmpAsteroid.radius < 0) {
        tmpAsteroid.y = tmpAsteroid.radius;
        tmpAsteroid.vY *= -1;
        tmpAsteroid.aY *= -1;
} else if (tmpAsteroid.y+tmpAsteroid.radius > canvasHeight) {
        tmpAsteroid.y = canvasHeight-tmpAsteroid.radius;
        tmpAsteroid.vY *= -1;
        tmpAsteroid.aY *= -1;
};
```

再次刷新页面，你就会看到期望的加速效果了。实际上，如果代码运行正常，小行星将永远处于加速状态，最终它们将以近似于光速的速度在屏幕上飞行。这太不现实了！要想改变小行星一直处于加速的状态，需要为每个小行星设置一个最大速度——限制为某种宇宙速度。

用以下代码替换前面的代码（把加速度添加到速度中的那段代码）：

```
if (Math.abs(tmpAsteroid.vX) < 10) {
        tmpAsteroid.vX += tmpAsteroid.aX;
};

if (Math.abs(tmpAsteroid.vY) < 10) {
        tmpAsteroid.vY += tmpAsteroid.aY;
};
```

以上代码的功能是，如果每个循环中小行星的速度小于10像素，就把加速度应用于该小行星。这种简单的检查可以限制小行星实际可以达到的速度，使小行星变得更易于控制。还需要重点注意的是，Math.abs方法将一个数转化成了绝对值数，这种方法主要用于删除数值前面的符号，例如，删除负数前面的符号。使用绝对值数意味着你仅处理正数，这样可以减少条件语句中的判断次数。

最后一次刷新页面，你应该看到一群小行星最终变得非常听话，它们的速度将逐步达到最大值。现在，你已经更进一步掌握了物理学的相关知识了。好极了！

> **注意** 我没有详细解释力，力实质上是沿特定方向的加速度。例如，如果需要模拟重力，你可以沿 y 轴的正向（向下）创建均匀的加速度。计算并应用精确的力的内容超出了本书的讨论范围，但我极力推荐你阅读Keith Peters写的《Flash ActionScript 3.0动画高级教程》。虽然它的内容介绍的是ActionScript，但其中包含了大量有用的公式，并运用物理学原理对高级动画进行了深入分析。

7.2.5 摩擦力

从技术上说，摩擦力也是一种力，可以非常精确地计算摩擦力，然后将它作用于物体，使物体降低速度。但是，这里我将透露一个运用物理学制作动画的小技巧：如果精确的计算非常复杂且需要耗费一些不必要的时间，那么我们可以模仿摩擦力！显然，如果你对动画的仿真度要求很高，那么这种方法就不太适用了。但是在多数情况下，尤其是对游戏而言，模仿的摩擦力产生的

效果和实际效果几乎看不出差别来。模仿物理量的优点在于计算时间更少且易于理解。

就摩擦力而言，你需要用它来降低物体的运动速度。正常情况下，必须根据物体及其经过的物体表面来计算真实的摩擦力；但是如果你采用模仿摩擦力的方法，就可以仅用物体的速度乘以一个摩擦系数来实现。对非专业人员来说，这两种方法产生的效果是相同的：计算正确的摩擦力并将它作用于物体，将会使物体减速；通过速度乘以一个摩擦系数来模仿摩擦力也会使物体减速。例如，如果物体的速度是2个像素，摩擦系数为0.9，最后的速度将会等于当前的速度乘以该系数，即等于1.8。在每个循环中使用相同的摩擦系数，物体的速度将很快趋近于0（假设该摩擦力使速度降低的幅度大于加速度使速度增加的幅度）。

如果需要在小行星示例中演示这种摩擦力，那么只需要在加速度代码后面添加以下代码即可：

```
if (Math.abs(tmpAsteroid.vX) > 0.1) {
        tmpAsteroid.vX *= 0.9;
} else {
        tmpAsteroid.vX = 0;
};
if (Math.abs(tmpAsteroid.vY) > 0.1) {
        tmpAsteroid.vY *= 0.9;
} else {
        tmpAsteroid.vY = 0;
};
```

以上代码把每颗小行星的速度都乘以0.9，其结果作为一个全局变量值。摩擦力将使每颗小行星逐渐减速，它们就好像在沿着台球桌往高处运动。当物体的速度降低到一定值时，条件语句用于取消摩擦力，防止摩擦力占用系统资源。为此，当速度取非常小的数值时（速度非常小，看上去好像处于静止状态），可以将速度值设置为0（使小行星停止运动）。

刷新页面查看结果，你会发现小行星的运动速度比先前更慢了。但是，由于加速度仍然在起作用，小行星永远不会完全停止下来。当然，也可以让小行星完全停止运动，但必须完全删除代码中的加速度。

在本节中，你已经学习了很多与动画有关的物理知识。有些书籍专门介绍有关动画的物理知识，如果它们采用了合适的介绍方法，你将会学到很多有用知识。如果不是，也不用担心，因为我花了很少时间就通过编写代码掌握相关的物理知识。其秘诀就在于反复编写、思考代码，更改代码中的值，再查看运行结果。为了真正理解这些内容，反复阅读这两节无疑会对你大有帮助。

如果你已经做好了准备，我们将继续讨论其他控制运动的方法。

7.3　碰撞检测

到目前为止，你已经在小行星中运用了一些有关运动的概念，让一群小行星在一个美丽的太空场景中四处漂动。但不足之处在于，所有的小行星在运行过程中会相互穿越，而没有相互碰撞。所幸的是，我们很快就可以解决这个问题，方法就是所谓的碰撞检测。在动画中采用碰撞检测可以创建更加逼真的动画。

碰撞检测有两个关键步骤：第一步，计算两个物体（小行星）是否发生重叠（碰撞）；第二

步，搞清楚这些物体以什么方式相互分离才显得更加逼真。综合考虑这两个关键步骤以后，你就可以创建更加形象逼真的动画效果，这种效果有点像台球相互碰撞后产生的运动状态。

在开始学习之前，首先需要把最后一个示例中有关摩擦力的代码删除，并将每颗新的小行星的加速度设为0。因为这些小行星在一个不存在摩擦力的空间中运动，如果我们保留摩擦力就显得不切合实际了。如果你不想完全删除这些代码，也可以把它们注释掉。

7.3.1 碰撞检测

从理论上说，这种现象非常简单：如果两个物体相互重叠，它们就会发生碰撞。这与它们是重叠1个像素还是很多像素没有关系，这是很显然的。当发生碰撞的物体是矩形时，检测碰撞相对要简单一些，在JavaScript中可以通过以下代码实现：

```
if (!(rectB.x+rectB.width < rectA.x) &&
        !(rectA.x+rectA.width < rectB.x) &&
        !(rectB.y+rectB.height < rectA.y) &&
        !(rectA.y+rectA.height < rectB.y)) {
        // 这两个物体重叠了
};
```

前两行代码用于检查所选矩形（A）（如图7-3所示）的左侧和右侧是否存在空隙。后两行代码用于检查所选矩形（A）（如图7-4所示）的上侧和下侧是否存在空隙。如果这些检查中有一个返回的结果为真，那么这两个矩形之间必定存在空隙，因此它们不可能相互重叠。在我们的例子中，需要检查它们之间是否出现重叠，因此检查所有这些条件是否都为假。它们的原理都是相同的。

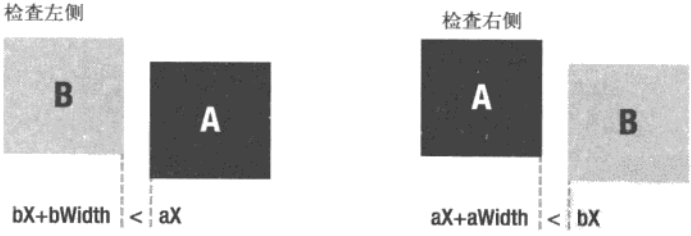


图7-3 检查左侧和右侧的空隙

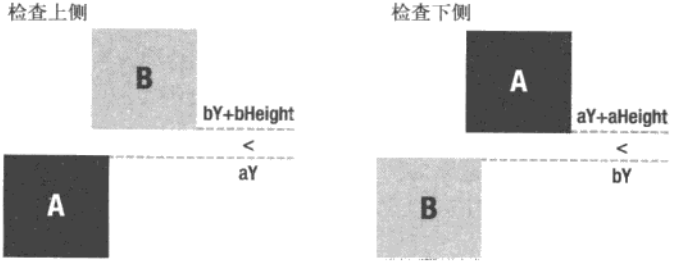


图7-4 检查上侧和下侧的空隙

小行星的示例中使用的是圆，因此检查矩形的方法用不上。否则，你会得到一些错误的结果。如图7-5所示，通过上面的检查方法，可能会发现矩形是重叠的，但圆并没有重叠，因此需要采用不同的检查方法。

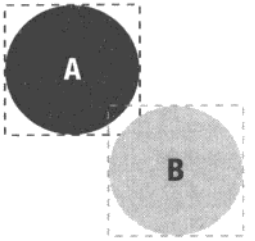


图7-5　矩形的检查方法对圆将失效

检查重叠的圆的方法是：计算两个圆的圆心之间的距离。更具体地说，需要检查两个圆心之间的距离是否小于这两个圆的半径之和。两个圆的半径之和是这两个物体不发生碰撞的最小距离（如图7-6所示），因此如果圆心之间的距离小于最小距离，那么两个圆必定是重叠的。这种检查方法很简单，用代码实现也相对简单。虽然这种方法非常简单，但由于涉及一些数学运算，所以在计算上会产生更大的开销。

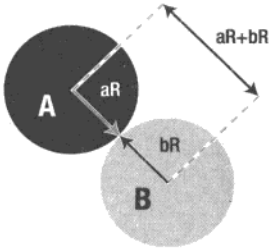


图7-6　圆心之间的距离小于它们的半径之和，圆将相互接触

由于需要检查的圆的位置具有不确定性，因此计算两个圆的圆心之间的距离还需要一些技巧，而不只是简单地比较它们之间的*x*和*y*位置。所幸，三角函数可以帮助我们解决这个问题。你将用到勾股定理，勾股定理是一个方程，如果知道一个直角三角形的两条边，就可以通过勾股定理计算出第三条边的长度。这种方法非常棒。但我不准备详细介绍该方程，如果你需要深入了解它，请阅读维基百科网页的相关内容[①]。

计算两个圆之间的距离时，总是需要构造一个直角三角形（如图7-7所示）来计算斜边。为了计算斜边，首先需要计算两个圆在*x*和*y*轴上的距离（在图7-7中分别用dX和dY表示）。

① http://en.wikipedia.org/wiki/Pythagorean_theorem。

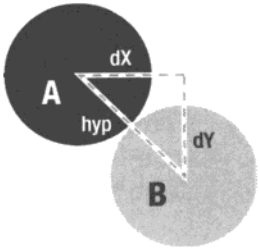


图7-7 计算两点之间的距离

dX和dY分别对应于直角三角形的两条直角边的长度。总之，这些长度可用于计算斜边的长度，只需要将这两个值代入勾股定理的方程即可（如图7-8所示）。

$$hyp = \sqrt{dX^2 + dY^2}$$

图7-8 勾股定理

现在，我们开始学习JavaScript代码的实现，首先打开前面小行星示例的代码，找到动画循环部分。在tmpAsteroid变量声明下面添加以下代码：

```
for (var j = i+1; j < asteroidsLength; j++) {
        var tmpAsteroidB = asteroids[j];

        var dX = tmpAsteroidB.x - tmpAsteroid.x;
        var dY = tmpAsteroidB.y - tmpAsteroid.y;
        var distance = Math.sqrt((dX*dX)+(dY*dY));
        };
};
```

第二个循环用于确保与其他所有小行星都比较一遍，以判断与它们之间是否重叠。注意，循环中的j变量设置为i+1；这是一种小技巧，可减少循环的执行次数。如果将j设置为0，那么就会把前面循环中已经检查过的小行星进行重复检查，这就显得有些混乱了。

dX和dY变量用于计算*x*和*y*轴上的距离，然后将这些距离代入勾股定理方程中计算distance变量的值。注意，这里使用了Math.sqrt方法来计算方程的平方根。distance变量中最后存储的结果即为两颗小行星圆心之间的真实像素距离。

算出了距离，判断小行星是否相互接触就轻而易举了。只需要计算小行星之间的半径之和，然后检查距离是否小于半径之和。将以下代码添加到第二个循环中的distance变量下面：

```
if (distance < tmpAsteroid.radius + tmpAsteroidB.radius) {
};
```

代码的复杂度与判断两个圆是否重合相似。而最有趣的部分是，既然你已经知道了两个圆相互接触，那么接下来该如何处理它们呢？这正是接下来要学习的内容。

7.3.2　弹开物体

坦率地说，接下来介绍的这部分内容稍微有些复杂，因此你要耐心学习。我也希望能够通过更多章节来详细解释这些概念，但这样一来就没有时间介绍有关Canvas的内容了，因此只能对相关物理概念做简要介绍。不过请放心，在必要的时候，我会对有关概念进行详细解释，以便你拓宽视野和了解更多信息。我更侧重向你展示如何实现这些重要的运动概念，为此有时候会忽略某些细节问题。

> **注意**　如果你需要了解更多有关碰撞检测和弹开对象的信息，建议你阅读Keith Peters编写的《Flash ActionScript 3.0动画教程》。正如我在上一节提到的另一本书一样，这本书也涉及很多使用物理知识创建动画的经典理论和代码示例。虽然这本书以ActionScript为基础，但其中仍然有很多概念和代码可以非常方便地移植到JavaScript中。

到目前为止，你已经知道了如何使物体在完全水平的表面上相互弹开。但是，对于以完全不同的角度和速度运动的圆形物体，这种方法就不行了。在水平表面上，可以通过反转速度（与表面垂直）的方向使物体弹回，因此反转y轴可以实现上下弹跳，反转x轴可以实现左右弹跳（如图7-9所示）。

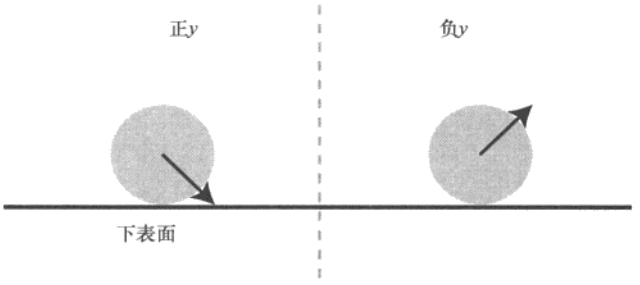


图7-9　底部平面上的反弹

现在的问题是，图7-9中的这种运动现象只会在极其特殊的环境下才会发生。实际上，两个圆发生碰撞后反向改变速度这种理想环境是不太可能出现的。例如，只有两个圆发生碰撞时正好完全位于一条直线上，并且它们的中心点之间没有任何角度（如图7-10所示）。在多数情况下，两个圆都会以一定角度发生碰撞，仅仅反向改变速度是行不通的（如图7-11所示）。此时，两个圆应该以正确的角度相互弹开（如图7-11所示，左侧的圆将直接向下运动）。但是，如果使用当前的方法，圆将会以奇怪的角度相互弹开。例如，图7-11中左侧的圆将会向相反方向弹开，这显然不符合逻辑。实际情况可能比这还要复杂得多，但我将尽量使问题简化，并介绍一些必要的基础知识。

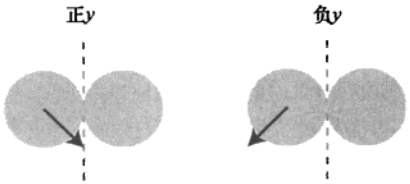


图7-10 两个圆之间发生的一种完全正面的碰撞

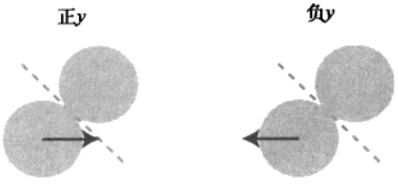


图7-11 两个圆之间发生的一种不切实际的碰撞

为了解决上述问题，需要通过一种方法来计算圆以未知角度发生碰撞后的位置和速度。下面提供的解决方法从概念上看很简单，但是理解它的代码可能稍微需要一些时间。下面我将对此进行解释。

如果两个圆发生碰撞，首先需要知道它们之间的角度（如图7-12所示）。为此，可以使用前面已经计算过的两个距离值（dX和dY），并将它们代入反正切函数中。如果已经知道直角三角形某个角的对边和邻边（它们分别对应于dX和dY），通过反正切函数，就可以计算出这个角的度数。

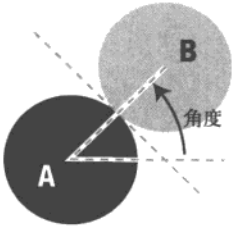


图7-12 计算两个圆之间的角度

JavaScript提供了反正切函数，但我们不准备使用该函数，因为它可能会产生一些潜在的错误结果。取而代之，我们将使用Math.atan2函数，这个函数更加可靠一些，能够准确计算你需要的结果。

下面我们给出使用Math.atan2函数的相关代码，在前面创建的空条件语句（检查两个圆周是否发生碰撞的语句）中添加以下代码：

```
var angle = Math.atan2(dY, dX);
var sine = Math.sin(angle);
var cosine = Math.cos(angle);
```

第一行代码用于计算两个圆之间的角度，接下来两行代码用于计算dX和dY分别与斜边（变量distance）相比所得的比值。后面需要多次使用这些值，所以可以先将它们计算出来。

既然已经知道了两个圆之间的角度，接下来可以依次把每个圆的速度进行旋转，这样它们发生的碰撞就类似于前面我提到的那种极少发生的理想碰撞了（如图7-10所示）。因此，速度就会从图7-12所示的状态转化成图7-13所示的状态，从而使圆的碰撞问题转化成了一种简单的反向改变速度的问题。所有这些听上去很简单，接下来将介绍如何用代码来实现这一步。

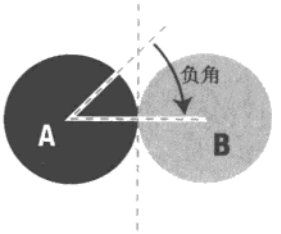


图7-13　旋转两个圆

下面一些公式稍微复杂一些，但是，这些公式完成了把每个圆的位置和速度进行旋转的所有工作。解释这些公式需要花费大量时间，而且效果也许并不明显，因此最好还是先相信我的话。如果你想进一步了解这些公式的原理，我建议你以后再深入学习，现在可以暂时忽略它们，可别说我没有提醒你哦！

在前面你声明的cosine变量下面插入以下代码：

```
var x = 0;
var y = 0;

var xB = dX * cosine + dY * sine;
var yB = dY * cosine - dX * sine;

var vX = tmpAsteroid.vX * cosine + tmpAsteroid.vY * sine;
var vY = tmpAsteroid.vY * cosine - tmpAsteroid.vX * sine;

var vXb = tmpAsteroidB.vX * cosine + tmpAsteroidB.vY * sine;
var vYb = tmpAsteroidB.vY * cosine - tmpAsteroidB.vX * sine;
```

得到的结果就是每个圆在旋转之后的位置和速度，旋转之后两个圆之间就没有角度了。我知道这可能很难掌握，但先别管它，我们离完成只有几步之遥了。

把每颗小行星的速度改为反向运动是一个非常简单的过程，只需在vYb变量下面插入以下代码即可：

```
vX *= -1;
vXb *= -1;
```

然后，需要让小行星相互分离，确保它们不再粘在一起，可以在反向改变小行星速度的代码下面插入以下代码：

```
xB = x + (tmpAsteroid.radius + tmpAsteroidB.radius);
```

最后，需要把这些小行星旋转到它们原来所在的位置，并使用新的速度。为此，代码与原先旋转小行星的代码基本相反。

在反向改变小行星位置和速度的代码下面插入以下代码：

```
tmpAsteroid.x = tmpAsteroid.x + (x * cosine - y * sine);
tmpAsteroid.y = tmpAsteroid.y + (y * cosine + x * sine);

tmpAsteroidB.x = tmpAsteroid.x + (xB * cosine - yB * sine);
tmpAsteroidB.y = tmpAsteroid.y + (yB * cosine + xB * sine);


tmpAsteroid.vX = vX * cosine - vY * sine;
tmpAsteroid.vY = vY * cosine + vX * sine;

tmpAsteroidB.vX = vXb * cosine - vYb * sine;
tmpAsteroidB.vY = vYb * cosine + vXb * sine;
```

这样，你就完成了相关代码的编写工作。请刷新页面，查看你创建的动态小行星群的效果。如果我是你，我会感到无比自豪，因为你在本节已经学习了很多有用的知识。

你也许会注意到，有些小行星不管撞上多大的小行星，它们弹回的速度看上去都非常快。这是因为在反向改变小行星的速度时采用的是一种类似于两个台球相撞的效果。接下来我们进一步简要介绍如何让每颗小行星实现动量守恒，这样它们弹回的速度就会显得更加逼真。

7.3.3 动量守恒

在前面的示例中，小行星在发生相互碰撞前后的速度大小是相同的，唯一不同之处在于速度的方向倒转。现在的动画效果已经非常不错了，但是正如前文所述，在碰撞和弹开的效果上还不够逼真。为此，需要通过一个过程来使小行星保持动量守恒。

动量守恒过程通过两颗小行星的当前速度来计算它们发生碰撞后各自新的速度。通过动量守恒，速度（或质量）较大的小行星与速度（或质量）较小的小行星发生碰撞之后，较小的小行星将被弹回。当质量和速度相似的两个小行星发生碰撞，如果它们发生的是正面相碰，那么就像两个台球迎面相撞一样，两个小行星都会因碰撞而停止运动。

但是，在进一步学习有关内容之前，你需要在小行星中添加质量属性，因为质量是计算动量的基本条件。

为了添加质量属性，需要用以下代码替换Asteroid类代码：

```
var Asteroid = function(x, y, radius, mass, vX, vY, aX, aY) {
        this.x = x;
        this.y = y;
        this.radius = radius;
        this.mass = mass;

        this.vX = vX;
        this.vY = vY;
```

```
        this.aX = aX;
        this.aY = aY;
};
```

创建小行星时，为了给每颗小行星添加质量属性，需要将循环代码改成以下形式：

```
for (var i = 0; i < 10; i++) {
        var x = 20+(Math.random()*(canvasWidth-40));
        var y = 20+(Math.random()*(canvasHeight-40));

        var radius = 5+Math.random()*10;
        var mass = radius/2;

        var vX = Math.random()*4-2;
        var vY = Math.random()*4-2;
        var aX = 0;
        var aY = 0;

        asteroids.push(new Asteroid(x, y, radius, mass, vX, vY, aX, aY));
};
```

注意mass变量和新的小行星对象中添加的mass参数。

现在每颗小行星都有了质量属性，接下来可以用小行星的质量乘以速度来计算它的动量。当然，我并不打算介绍动量的完整定义，但你至少应该知道这些概念。

我不打算在本节用过多的篇幅来解释这些概念，因为这的确是一个非常复杂的问题，需要花费一定的时间才能正确掌握。我的目标是告诉你如何让动画产生这种逼真的效果，而不是解释公式的原理。

因此，我们忽略这些复杂的问题，继续关注动量守恒。

接下来，你需要注释掉或删除一段代码(发生碰撞后，反向改变每个小行星速度的那段代码)，然后在对应位置插入以下代码：

```
    var vTotal = vX - vXb;
    vX = ((tmpAsteroid.mass - tmpAsteroidB.mass) * vX + 2 * tmpAsteroidB.mass * vXb)
/(tmpAsteroid.mass + tmpAsteroidB.mass);
    vXb = vTotal + vX;
```

这看上去有些费解，但我们已经完成了所有的代码。刷新浏览器并查看一下效果吧。现在的动画效果是不是变得更逼真了？

以上代码的基本思想是：首先计算两颗小行星的总速度（或相对速度），然后利用动能公式计算第一颗小行星的新速度；再把第一颗小行星的新速度与前面已经计算过的相对速度相加来计算第二颗小行星的新速度。

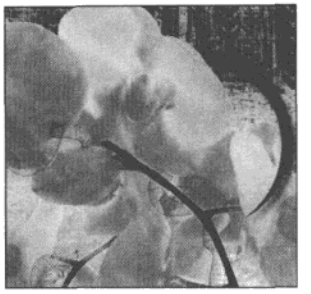
除非你真的需要理解这些公式，否则不需要急于了解它们的原理。这段代码我已经使用很长一段时间了，但直到现在为止，我还没有完全理解它。这些复杂的数学问题还是留给那些聪明的人去解决吧。

7.4 小结

到目前为止，本章可能是最难的一章了，祝贺你学完了它。你已经学习了与动画有关的物理知识及其概念。另外，你还学会了如何在JavaScript中运用这些概念，以及如何通过它们创建视觉上更加逼真的动画效果。最后，你学习了如何在物体之间创建碰撞效果，以及如何让这些碰撞效果变得更加栩栩如生。通过本章的学习，你真的可以说是满载而归。

还有一个好消息，你已经学完了本书的全部理论知识。从现在开始，你将真正开始学习如何创建一些有趣的游戏。在创建游戏的过程中，你将综合运用相关技术创建一些绚丽的动画效果，这才是真正有趣的内容。

第8章



太空保龄球游戏

你即将创建一个太空保龄球游戏——这很可能是你在画布中创建的第一个游戏。本章的思路是综合运用前几章介绍的技巧来创建一个易于操作的小游戏，虽然简单，但它却非常有趣和令人着迷。

我必须强调的是，本书的主要目的是介绍如何在画布中采用一些基本技巧来创建动画和交互式游戏，这一点在本章和下一章体现得尤为明显。这意味着本书在选取示例时有意采用那些看上去很简单的游戏。这样可以避免一些过于复杂的问题。当你开始学习动画和编码的相关知识时，没有必要花费太多的精力来设计复杂的对象。画布以及你即将创建的游戏的优点在于你很容易对它们进行修改。例如，你完全可以用自己的方法把小行星画得更灿烂一些，并用自己的代码来替换本书中的代码。实际上，我很喜欢你这么做！

本章的所有代码都可以通过Friend of ED出版社的网站（www.friendsofed.com）[①]下载，本书其他章节的代码也可以通过该网站下载。下载的这些代码可用于检查你自己所编写的代码是否正确。但我仍然建议你手工输入所有这些代码，而不是复制和粘贴下载的代码。因为手工输入代码有助于更好地学习相关内容。

8.1 游戏概述

太空保龄球游戏在窗口的上部包含一个圆形平台，一群静止的小行星分布在圆形平台上，另外一个稍大一些的小行星放在离圆形平台较远的位置，作为玩家投掷时使用的圆球（如图8-1所示）。游戏的思路是：将较大的小行星扔向其他小行星，并尽可能多地撞开位于圆形平台上的小行星。当圆形平台上所有的小行星都被撞离平台时，你就在游戏中获胜了。游戏的分数是根据你撞开所有小行星所用的撞击次数来计算的。这个游戏非常简单，但很有趣，因为它是根据一些非常经典的游戏而创建的，例如保龄球和弹球游戏。

① 现在的网址已更换为http://www.apress.com/。——译者注

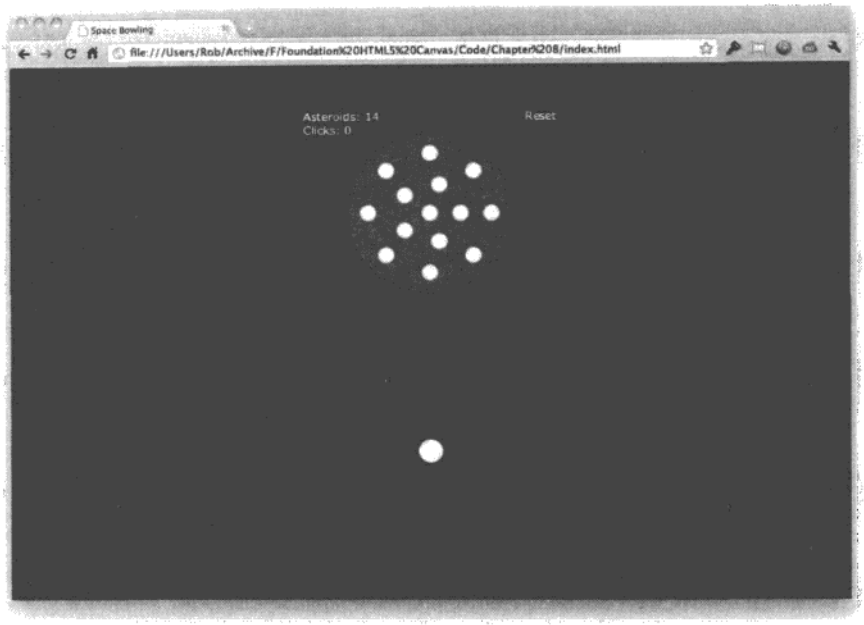


图8-1 太空保龄球游戏

需求

创建该游戏将涉及很多本书中已经介绍过的技巧。你需要使用已学到的所有知识，例如JavaScript、在画布中绘制图形的技巧、动画以及物理学知识。你甚至还会使用一些还没有详细介绍的技术，例如构建用户界面以及使用鼠标输入在画布上绘制对象。通过创建太空保龄球游戏，你将真正体会到前面所学技术的强大威力。

我确信你已经跃跃欲试了，那么让我们开始吧！

8.2 核心功能

在创建游戏之前，首先需要构建一些基本框架。就创建太空保龄球游戏而言，这些框架就是基本的HTML、CSS以及JavaScript代码（作为将来要添加的高级代码的基础）。

8.2.1 构建 HTML 代码

在浏览器中创建游戏的优点在于可以使用一些构建网站的常用技术。也就是说，可以使用HTML语言来创建游戏的用户界面（UI）。现在的界面看上去不太美观，这是因为我们还没有使

用CSS来设计用户界面的样式，但目前内容的原始结构是最重要的。

在你的计算机上为该游戏创建一个新目录，新建一个index.html文件，在其中加入以下代码：

```
<!DOCTYPE html>

<html>
        <head>
                <title>Space Bowling</title>
                <meta charset="utf-8">

                <link href="game.css" rel="stylesheet" type="text/css">

                <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/➥
jquery/1/jquery.min.js"></script>
                <script type="text/javascript" src="game.js"></script>
        </head>

        <body>
                <div id="game">
                        <div id="gameUI">
                                <div id="gameIntro">
                                        <h1>Space bowling</h1>
                                        <p>This is an awesome game.</p>
                                        <p><a id="gamePlay" class="button"href="">➥
Play</a></p>
                                </div>
                                <div id="gameStats">
                                        <p>Asteroids: <span
id="gameRemaining"></span></p>
                                        <p>Clicks: <span
class="gameScore"></span></p>
                                        <p><a class="gameReset"
href="">Reset</a></p>
                                        </div>
                                        <div id="gameComplete">
                                                <h1>You win!</h1>
                                                <p>Congratulations, you completed
the game in <span class="gameScore"></span> clicks.</p>
                                                <p><a class="gameReset button"
href="">Play again</a></p>
                                        </div>
                                </div>
                                <canvas id="gameCanvas" width="350" height="600">
                                        <!-- 在此插入后备内容
                                </canvas>
                        </div>
                </body>
</html>
```

我不打算过多解释这些HTML代码，因为它们比较简单，但你只要知道这就是游戏所需的所

有标记即可。

head元素中的代码导入了游戏的CSS和JavaScript文件。而body元素的代码描述了游戏的UI和画布。

游戏UI被分为几个单独的部分，这是为了便于使用JavaScript与用户进行交互和控制特定区域的可见性。例如，UI的三个主要区域分别是介绍屏幕、统计屏幕和游戏完成屏幕。在最终的游戏中，某一时刻只会显示其中的某个区域：在开始时显示介绍屏幕，在游戏过程中显示统计屏幕，在游戏结束时显示完成屏幕。

从目前情况看，游戏并没有什么看点，如图8-2所示。此刻你只能看到原始的UI内容，其中包含一些链接，我们即将把它们转变成精致美观的按钮；还包含在玩游戏时用于统计信息的占位符文本。画布元素实际上也包含在该页面上，但目前还无法看到。如果你仔细观察，可以发现窗口右边的滚动条，这表明其中实际上已经包含了画布元素。

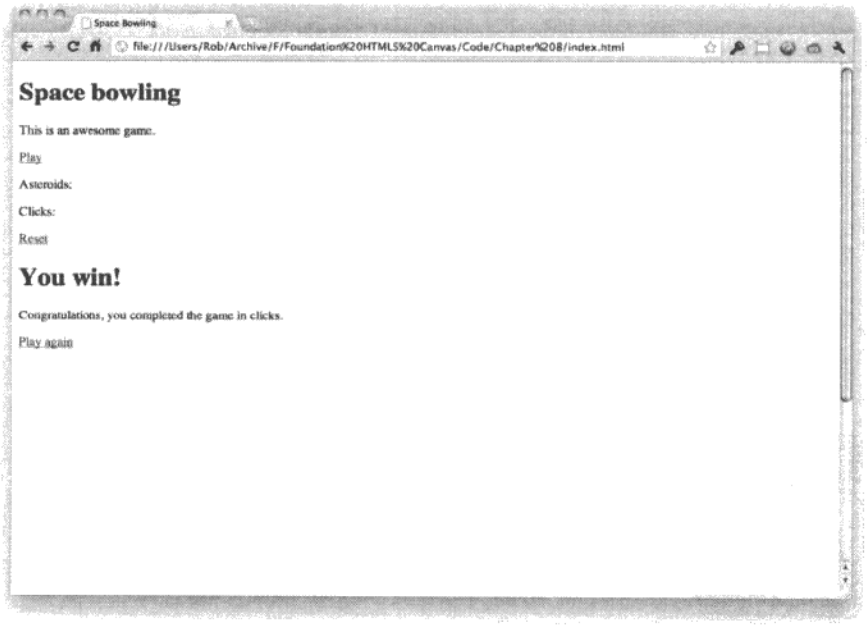


图8-2 使用HTML创建游戏

注意 需要重点指出的是，游戏UI的标记必须在canvas元素前面，这是因为需要在画布元素上方显示UI。如果把UI代码放在画布元素之后，那么就需要采用一些额外的CSS技术让UI显示在画布上方。

8.2.2　美化界面

现在HTML已经就绪，接下来需要采用一些CSS技术使界面看上去更有游戏的感觉。即将使用的CSS其实非常简单，但我仍然会简要介绍一下该代码段将会产生的效果。

创建一个名称为game.css的新文件，并把它和HTML文件放在相同的目录下。在该CSS文件中插入以下代码：

```
* { margin: 0; padding: 0; }
html, body { height: 100%; width: 100%; }
canvas { display: block; }

body {
        background: #000;
        color: #fff;
        font-family: Verdana, Arial, sans-serif;
        font-size: 18px;
}

h1 {
        font-size: 30px;
}

p {
        margin: 0 20px;
}
a {
        color: #fff;
        text-decoration: none;
}

a:hover {
        text-decoration: underline;
}

a.button {
        background: #185da8;
        border-radius: 5px;
        display: block;
        font-size: 30px;
        margin: 40px 0 0 45px;
        padding: 10px;
        width: 200px;
}

a.button:hover {
        background: #2488f5;
        color: #fff;
        text-decoration: none;
}

#game {
        height: 600px;
```

```
        left: 50%;
        margin: -300px 0 0 -175px;
        position: relative;
        top: 50%;
        width: 350px;
}

#gameCanvas {
        background: #001022;
}

#gameUI {
        height: 600px;
        position: absolute; /*将UI置于画布的上方*/
        width: 350px;
}

#gameIntro, #gameComplete {
        background: rgba(0, 0, 0, 0.5);
        margin-top: 100px;
        padding: 40px 0;
        text-align: center;
}

#gameStats {
        font-size: 14px;
        margin: 20px 0;
}

#gameStats .gameReset {
        margin: 20px 20px 0 0;
        position: absolute;
        right: 0;
        top: 0;
}
```

CSS文件的前三行用于重设样式，并使HTML文档适合浏览器窗口的尺寸。接下来的5个规则（body、h1、p、a和a:hover）用于定义文档的背景颜色，同时定义了游戏中的文本和超链接的默认样式。随后的两个规则（a.button和a.button:hover）用于定义游戏中按钮的样式，很快就能看到效果。之所以使用类来定义按钮型链接，是因为使用相同的样式来定义所有按钮会更方便一些。

游戏的所有HTML代码都包含在一个div元素中。#game CSS规则用于定义容器的特定尺寸，并通过一些巧妙的CSS位置和页边距策略让所有对象在浏览器窗口中居中显示。画布元素的样式是通过#gameCanvas规则来定义的，但它只是把游戏的背景颜色改变为深蓝色。

UI的样式是通过各种CSS规则来定义的，其中包括#gameUI、#gameIntro、#gameComplete、#gameStats和#gameStats .gameReset。每条规则都会影响HTML文档中单个UI元素的样式。例如，#gameUI使用绝对位置值将UI移动到canvas元素的上方。CSS文件中的其余规则改变的是文本的样式，并对位置进行细微调整。

总而言之，通过使用CSS文件可以令游戏的外观变得更美观一些，虽然还不够完美（如图8-3所示）。其他一些问题还需要在JavaScript代码中解决。

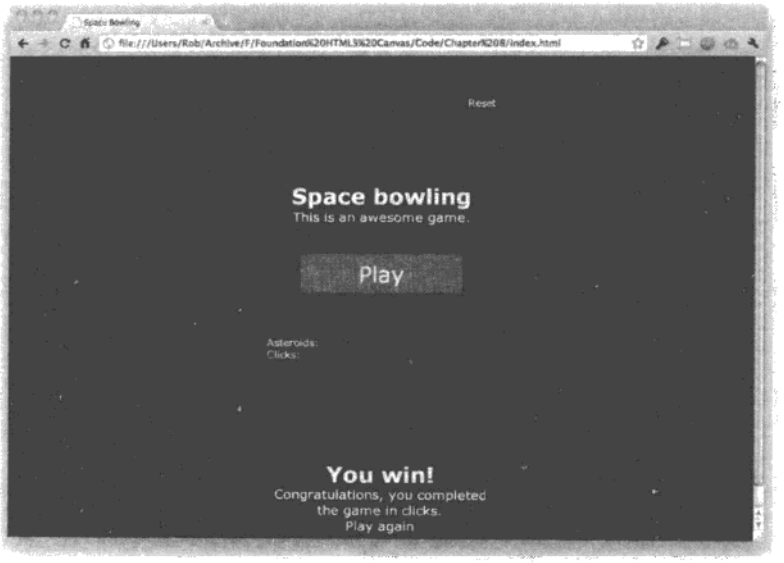


图8-3　使用CSS设置UI的样式

8.2.3　编写 JavaScript 代码

在添加一些有趣的游戏逻辑之前，首先需要用JavaScript实现核心功能。

创建一个名为game.js的新文件，并把它和HTML文件放在相同的目录下。在该js文件中插入以下代码：

```
$(document).ready(function() {
        var canvas = $("#gameCanvas");
        var context = canvas.get(0).getContext("2d");

        // 画布尺寸
        var canvasWidth = canvas.width();
        var canvasHeight = canvas.height();

        // 游戏设置
        var playGame;

        // 重置和启动游戏
        function startGame() {
                // 初始游戏设置
                playGame = false;
```

```
            // 开始动画循环
            animate();
        };

        // 初始化游戏环境
        function init() {

        };
        // 动画循环，游戏的趣味性就在这里
        function animate() {
            // 清除
            context.clearRect(0, 0, canvasWidth, canvasHeight);

            if (playGame) {
                // 33毫秒后再次运行动画循环
                setTimeout(animate, 33);
            };
        };
        init();
});
```

除了几个函数之外，你应该对这些代码非常熟悉了。前四个变量用于保存游戏的画布元素，访问2D渲染上下文，并指定画布的尺寸。第五个变量playGame已经声明，但还没有赋值，后面将对它赋值，用于确定是否运行动画代码。

startGame函数是一个占位符，当玩家在UI中单击“Play”、“Reset”或“Play again”链接时，该占位符用于重置或启动游戏。从目前情况看，该函数仅仅将PlayGame变量设置为false，禁止当前运行动画。另外，startGame函数还将调用animate函数，animate函数将每秒帧数的超时值设置为基础值30，暂时还没有其他功能。最后一个函数是init，当第一次载入游戏时，该函数将完成游戏的所有初始化设置。现在该函数还是空的，当我们准备好JavaScript代码后将调用该函数。

现在已经构建了游戏的核心功能，下面我们将介绍更精彩的内容！

8.3 激活用户界面

到目前为止，游戏的界面还比较难看，并且UI也还没有实现任何功能。好在，修改界面和激活UI都非常简单！

在startGame函数上面插入以下代码：

```
var ui = $("#gameUI");
var uiIntro = $("#gameIntro");
var uiStats = $("#gameStats");
var uiComplete = $("#gameComplete");
var uiPlay = $("#gamePlay");
var uiReset = $(".gameReset");
var uiRemaining = $("#gameRemaining");
var uiScore = $(".gameScore");
```

这些变量保存着各种UI元素，后面我们将使用这些变量。此刻，它们仅作为UI中DOM元素的快捷方式。

为了整理当前的初始化屏幕（如图8-3所示），需要隐藏所有不必要的UI元素并激活按钮，以便真正启动游戏和改变屏幕。在init函数中插入以下代码：

```
uiStats.hide();
uiComplete.hide();

uiPlay.click(function(e) {
        e.preventDefault();
        uiIntro.hide();
        startGame();
});

uiReset.click(function(e) {
        e.preventDefault();
        uiComplete.hide();
        startGame();
});
```

以上代码的前两行用于隐藏UI的统计和完成屏幕，这立刻使游戏界面变得更美观起来（如图8-4所示）。随后的两行代码是单击事件监听器：一个用于监控第一个屏幕上的Play按钮；另一个用于监控游戏屏幕上的重置按钮和游戏的结束屏幕。这两个事件监听器都隐藏了相关的UI屏幕，并运行startGame函数启动游戏。

图8-4　使用JavaScript隐藏部分UI

当游戏开始时，还需要显示统计屏幕，因此在startGame函数的顶部插入以下代码：

```
uiScore.html("0");
uiStats.show();
```

第一行代码通过将分数设置为字符串“0”来重置分数。第二行代码确保游戏开始时能够显示统计屏幕（如图8-5所示）。在游戏中，当你使用UI的其他部分时，还需要继续设置和更改它们的值。

图8-5 重置游戏的分数

8.4 创建游戏对象

许多游戏都是由具有特定目的的对象来定义的：不论是俄罗斯方块中的方块、国际象棋的棋子，还是超级玛丽中的玛丽，它们都是具有特定目的的对象。在计算机游戏中，没有对象的游戏是很少的。在太空保龄球游戏中，有两个主要对象：小行星（包括较大的那个小行星）和放置小行星的圆形平台。当把游戏分解成一些核心对象和组件时，你就会觉得游戏看上去变得更简单了，这太神奇了。

尤其是当你准备在游戏中使用多个对象时，最好将对象定义为JavaScript类。因为这可以确保多个对象都建立在相同的代码基础上，并且具有相同的属性和方法。你即将在游戏中创建一个小行星类，而不需要为平台创建类。这是因为平台完全是一个独立的对象，它在游戏过程中只会出现一次，所以不需要使用类，只有创建多个平台时才需要使用类。

8.4.1 创建平台

在创建小行星之前，首先需要建立放置小行星的圆形平台（参见图8-1）。平台是圆形的，它

是由一些特定的变量定义的，请将把这些变量添加到位于代码顶部的playGame变量的下面：

```
var platformX;
var platformY;
var platformOuterRadius;
var platformInnerRadius;
```

这些变量将存储平台的原点(*x*,*y*)、外半径（整个平台区域）以及内半径（实际放置小行星的区域）。目前，这些平台变量还没有任何值，因此在startGame函数中的playGame变量下面添加以下代码：

```
platformX = canvasWidth/2;
platformY = 150;
platformOuterRadius = 100;
platformInnerRadius = 75;
```

以上代码把平台的原点放在*x*轴的正中间，在*y*轴上距离画布顶部150像素。外半径长度设置为100像素（注意：半径是圆周直径的一半），内半径长度稍微小一些，以便在平台的边缘和小行星所在位置之间留一点间距。

最后，需要在画布上绘制平台，因此在animate函数中的clearRect调用下面添加以下代码：

```
context.fillStyle = "rgb(100, 100, 100)";
context.beginPath();
context.arc(platformX, platformY, platformOuterRadius, 0, Math.PI*2, true);
context.closePath();
context.fill();
```

在游戏UI上单击Play按钮，将会看到该平台（如图8-6所示）。很兴奋吧。

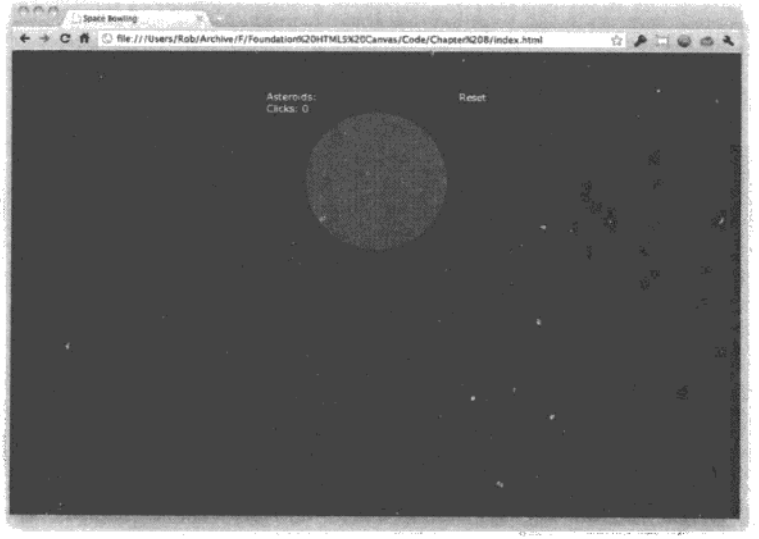


图8-6　创建小行星平台

8.4.2 创建小行星

接下来需要创建小行星代码，并为创建动画做好准备。你需要使用的大部分代码与上一章学习的代码几乎完全相同，因此对这些内容我将不做过多解释。

你需要记忆所有小行星的运动状态并进行碰撞检测，因此需要在JavaScript顶部的平台变量下面声明一个asteroids变量：

```
var asteroids;
```

在游戏环境中，你将使用该变量来存储一个包含所有小行星的数组。

仅仅声明小行星数组是不行的，你还需要在数组中放置小行星，因此下一步工作是声明小行星类，该类是建立在上一章的asteroids类的基础上。在startGame函数上面添加以下代码：

```
var Asteroid = function(x, y, radius, mass, friction) {
        this.x = x;
        this.y = y;
        this.radius = radius;
        this.mass = mass;
        this.friction = friction;
        this.vX = 0;
        this.vY = 0;
        this.player = false;
};
```

需要重点注意的是，与上一章中的asteroids类相比，这里的小行星类还是略有不同的。最显著的区别是，这里声明的类添加了friction和player属性，分别用于定义每个小行星产生的摩擦力大小以及该小行星是否是玩家使用的较大小行星，player的默认值是false。另外，还需要注意，这里没有包含加速度属性，这是因为游戏中使用的运动非常简单，没有必要使用加速度。很快你就会完全理解其中的原因。

在创建所有小行星并把它们放在平台上之前，需要建立空的asteroids数组，因此需要在startGame函数中的平台变量下面添加以下代码：

```
asteroids = new Array();
```

需要用startGame函数来为这些变量赋值，因为玩家每次重新启动游戏时（玩家获胜、再次开始游戏或单击Reset链接），你都希望刷新和重写这些变量的值。

为了让这些小行星整齐地排列成环形（如图8-1所示），你需要动动脑筋。在startGame函数中的asteroids变量下面添加以下代码：

```
var outerRing = 8; // 外圈上的小行星数目
var ringCount = 3; // 圈数
var ringSpacing = (platformInnerRadius/(ringCount-1)); // 每个圈之间的距离

for (var r = 0; r < ringCount; r++) {
        var currentRing = 0; // 当前圈上的小行星数目
        var angle = 0; // 每颗小行星之间的角度
        var ringRadius = 0;
```

```
        // 这是最里面的圈吗?
        if (r == ringCount-1) {
                currentRing = 1;
        } else {
                currentRing = outerRing-(r*3);
                angle = 360/currentRing;
                ringRadius = platformInnerRadius-(ringSpacing*r);
        };
};
```

我已经在以上代码中添加了一些注释，但还需要作一些说明。

前三个变量分别用于声明你需要在最外圈上放置的小行星的数目、圈数以及每个圈之间的距离。当计算在每圈上放置多少颗小行星时，需要使用这些变量。

这些变量后面是第一个循环，它对平台上放置的每圈小行星执行循环。在每个循环中，currentRing变量的值被设为默认值0。在下面的条件语句中，currentRing变量用于存储需要在当前圈上放置的小行星数目。同样，angle和ringRadius变量的默认值也设为0。

条件语句首先检查当前是否为最后一圈小行星（即中心点），如果是，那么就把小行星的数目设为1。如果不是，则小行星数目的计算方法为：最外层的小行星数目减去3乘以当前的循环数。虽然这种计算方法有些随意，但效果却很好。如果你能提供更好的计算方法，当然也可以更改相关代码。但总体思想是：随着圈距离中心的位置越来越近，你希望每道圆环上的小行星数目变得越来越少，否则，你就无法在圆形平台上合理放置这些小行星。

确定了当前圈上应该放置的小行星数目以后，还需要计算每颗小行星之间的角度，以便使它们均匀地分布在圈上。角度的计算方法为：将圆周的总角度（360度）除以需要放置的小行星的数目（如图8-7所示）。这里使用角度制是为了便于理解代码，其实使用弧度制也很容易理解。

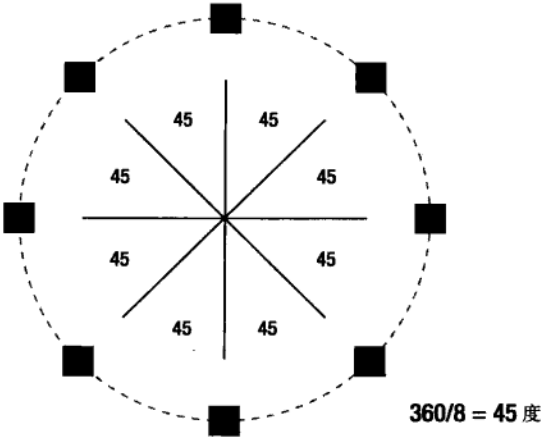


图8-7　计算每颗小行星之间的角度

另外，计算每圈的半径也很重要，因为你希望这些圈能够均匀地分布在圆形平台上。可以用前面定义的platformInnerRadius变量计算半径，具体计算方法为：先将每个圈之间的距离乘

以当前的循环数，然后用platformInnerRadius减去该乘积即可。这种计算方法会使每个圈变得越来越小，同时还可以让这些圈均匀地分布在圆形平台上（如图8-8所示）。

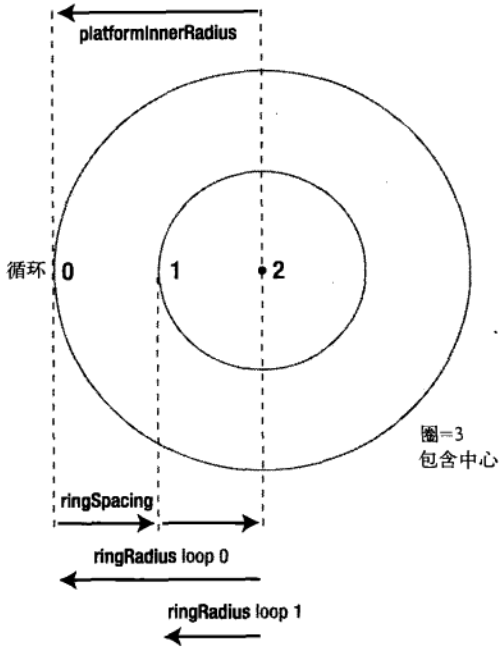


图8-8 计算每个圆环的半径

现在，创建所有小行星所需要的条件已经都具备了，接下来是第二个循环，该循环将依次遍历每颗小行星。在条件语句下面添加以下代码（**在圈循环内部**）：

```
for (var a = 0; a < currentRing; a++) {
        var x = 0;
        var y = 0;

        //这是最里面的圈吗?
        if (r == ringCount-1) {
              x = platformX;
              y = platformY;
        } else {
              x = platformX+(ringRadius*Math.cos((angle*a)*(Math.PI/180)));
              y = platformY+(ringRadius*Math.sin((angle*a)*(Math.PI/180)));
        };

        var radius = 10;
        var mass = 5;
        var friction = 0.95;
        asteroids.push(new Asteroid(x, y, radius, mass, friction));
};
```

对当前圈上的每一颗小行星而言，循环可以通过一个简单的条件语句来计算该小行星的位置。如果当前的小行星位于最里面的圈上，那么其位置应该放在平台的中心，因此可以把小行星的原点（x, y）放在平台的中心。否则，应该通过当前圈所在的圆周来计算小行星的位置。这听上去也许有些蹊跷，但是在6.5节中你已经使用了该代码。唯一不同之处在于，第6章中增加的角度是5，这里增加的角度是基于刚才计算的结果。每颗小行星之间需要存在一定的角度，以便它们均匀地分布在圈上。

最后，声明小行星的半径、质量以及摩擦力，并将小行星添加到先前创建的asteroids数组中。小行星已经各就各位了！当给这些小行星添加动画时，如何显示它们呢？这是下一节讨论的问题。

8.4.3　创建玩家使用的小行星

现在只剩下最后一个对象需要创建了，那就是玩家投向其他小行星时使用的较大的小行星。这个小行星实际上和其他小行星很相似，但我们对它的处理方式略有不同，所以需要单独创建它。首先，在JavaScript代码的顶部建立一些变量，在asteroids变量声明下面添加以下代码：

```
var player;
var playerOriginalX;
var playerOriginalY;
```

需要使用player变量，以便在游戏中单独访问玩家使用的小行星。很快你就会明白playerOriginalX和playerOriginalY变量的用途，但现在只需要知道这些变量将用于存储玩家使用的小行星的初始位置即可。

接下来就是实际创建玩家使用的小行星了。为此，在startGame函数中的asteroids变量下面添加以下代码：

```
var pRadius = 15;
var pMass = 10;
var pFriction = 0.97;
playerOriginalX = canvasWidth/2;
playerOriginalY = canvasHeight-150;
player = new Asteroid(playerOriginalX, playerOriginalY, pRadius, pMass, pFriction);
player.player = true;
asteroids.push(player);
```

以上这些代码都非常简单：首先，声明半径（该半径比其他小行星的半径大一些）、质量（同样，大于其他小行星的质量）、摩擦力以及玩家的初始位置（位于屏幕的底部）；然后，用这些变量创建一个新的小行星对象。该操作的不同之处在于，把新的小行星对象赋值给player变量，这样就可以在后面再次使用player变量。可以直接使用player变量来更改小行星的player属性，因为这就是玩家使用的小行星（咳，我又多嘴了）。

最后，还需要把玩家使用的小行星添加到asteroids数组中，以便在下一节的动画和碰撞检测中使用它。

玩家使用的小行星将与其他小行星一起在animate函数中绘制，我们很快就会介绍这部分内容。

8.4.4 更新 UI

现在已经创建了所有的小行星，接下来应该更新UI来显示平台上放置的小行星数量。幸运的是，这非常简单，只需要添加一行代码即可。在刚才创建的循环后面插入以下代码：

```
uiRemaining.html(asteroids.length-1);
```

以上代码将通过asteroids数组中的小行星数目来更新分配给uiRemaining变量的HTML元素，因为你不希望包含玩家使用的小行星，所以需要将数组的长度减1（如图8-9所示）。这非常简单，但需要指出的是，现在你实际上还看不到这些小行星。

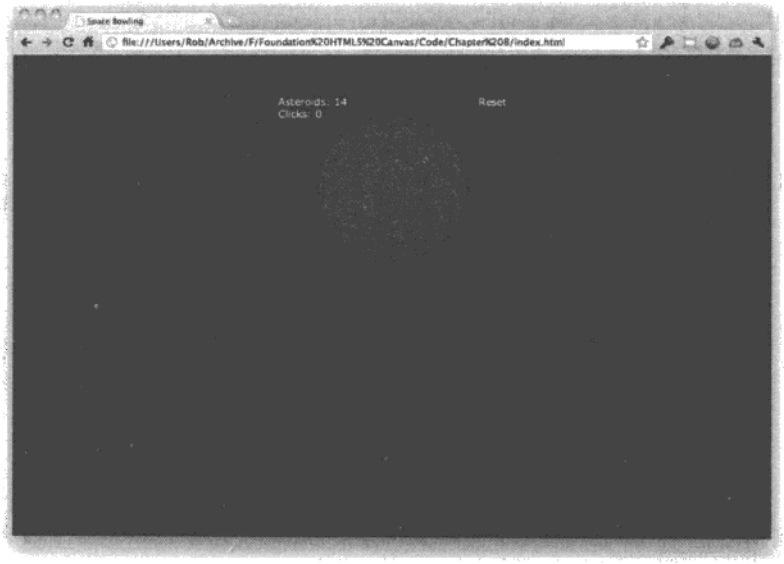


图8-9 显示保留的小行星数目

8.5 让对象运动起来

到目前为止，你已经建立了游戏的所有核心功能，但还没有显示任何小行星或动画。这根本不能算是游戏，因此我们要做一些改变。本节即将实现的一些操作都是你已经接触过的知识。

首先是建立两个循环，上一章你通过这两个循环对小行星实施移动和碰撞检测。这些循环将实现小行星的移动和相互碰撞，而这正是你需要它们在平台上产生的运动（通过向它们投掷玩家使用的小行星）。

在animate函数中绘制平台的代码下面插入以下代码：

```
context.fillStyle = "rgb(255, 255, 255)";
```

```
var asteroidsLength = asteroids.length;
for (var i = 0; i < asteroidsLength; i++) {
        var tmpAsteroid = asteroids[i];

        for (var j = i+1; j < asteroidsLength; j++) {
                var tmpAsteroidB = asteroids[j];
        };

        context.beginPath();
        context.arc(tmpAsteroid.x, tmpAsteroid.y, tmpAsteroid.radius, 0, Math.PI*2,
true);
        context.closePath();
        context.fill();
};
```

除了将fillStyle属性设为白色之外，以上代码没有什么特别之处，因为你在前面将平台的fillStyle设为灰色了。此时，如果在浏览器中查看游戏并单击Play，就可以看到一些小行星（如图8-10所示）。这看上去有点像游戏了。

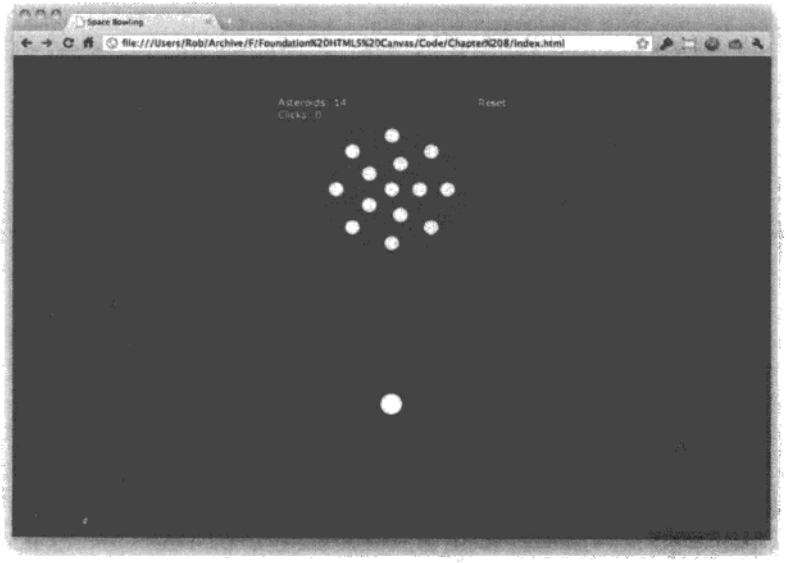


图8-10　显示小行星

目前，还没有任何运动或动画效果。要实现这些效果，首先需要添加上一章使用的碰撞检测代码，为此，在tmpAsteroidB变量下面添加以下代码：

```
var dX = tmpAsteroidB.x - tmpAsteroid.x;
var dY = tmpAsteroidB.y - tmpAsteroid.y;
var distance = Math.sqrt((dX*dX)+(dY*dY));
```

```
if (distance < tmpAsteroid.radius + tmpAsteroidB.radius) {
        var angle = Math.atan2(dY, dX);
        var sine = Math.sin(angle);
        var cosine = Math.cos(angle);

        //旋转小行星的位置
        var x = 0;
        var y = 0;

        //旋转小行星B的位置
        var xB = dX * cosine + dY * sine;
        var yB = dY * cosine - dX * sine;

        //旋转小行星的速度
        var vX = tmpAsteroid.vX * cosine + tmpAsteroid.vY * sine;
        var vY = tmpAsteroid.vY * cosine - tmpAsteroid.vX * sine;

        //旋转小行星B的速度
        var vXb = tmpAsteroidB.vX * cosine + tmpAsteroidB.vY * sine;
        var vYb = tmpAsteroidB.vY * cosine - tmpAsteroidB.vX * sine;

        //保持动量
        var vTotal = vX - vXb;
        vX = ((tmpAsteroid.mass - tmpAsteroidB.mass) * vX + 2 * tmpAsteroidB.mass *
vXb) /(tmpAsteroid.mass + tmpAsteroidB.mass);
        vXb = vTotal + vX;

        //将小行星分开
        xB = x + (tmpAsteroid.radius + tmpAsteroidB.radius);

        //转回小行星的位置
        tmpAsteroid.x = tmpAsteroid.x + (x * cosine - y * sine);
        tmpAsteroid.y = tmpAsteroid.y + (y * cosine + x * sine);

        tmpAsteroidB.x = tmpAsteroid.x + (xB * cosine - yB * sine);
        tmpAsteroidB.y = tmpAsteroid.y + (yB * cosine + xB * sine);

        //转回小行星的速度
        tmpAsteroid.vX = vX * cosine - vY * sine;
        tmpAsteroid.vY = vY * cosine + vX * sine;

        tmpAsteroidB.vX = vXb * cosine - vYb * sine;
        tmpAsteroidB.vY = vYb * cosine + vXb * sine;
};
```

这些代码看上去很复杂，但它和上一章使用的代码完全相同。让我们快速浏览一下代码：该段代码用于检查每颗小行星，查看它们是否相互重叠（碰撞），如果是，则改变每个小行星的速度，从而产生更加逼真的弹开效果。

即使现在运行动画循环（尚未运行），也不会出现任何移动效果，因为还没有更新每颗小行星的位置。为此，还需要在画布上（在绘制每个小行星的代码之前）添加位置和摩擦力代码：

```
//计算新位置
tmpAsteroid.x += tmpAsteroid.vX;
tmpAsteroid.y += tmpAsteroid.vY;

//摩擦力
if (Math.abs(tmpAsteroid.vX) > 0.1) {
        tmpAsteroid.vX *= tmpAsteroid.friction;
} else {
        tmpAsteroid.vX = 0;
};
if (Math.abs(tmpAsteroid.vY) > 0.1) {
        tmpAsteroid.vY *= tmpAsteroid.friction;
} else {
        tmpAsteroid.vY = 0;
};
```

同样，这些代码也很常见。首先，根据每颗小行星的速度来更新它们的位置；然后运用摩擦力（在下面的动画循环中需要考虑摩擦力）。如果小行星的速度小于0.1，则自动将其速度设为0，防止小行星继续运动。这正好也可以防止摩擦力代码始终处于运行状态。

如果希望查看这段代码的运行效果，在创建玩家使用的小行星时，你还需要为其赋一个负值（例如，−25），并在`startGame`函数中将`playGame`的值改为`true`。即将产生的效果是：玩家使用的小行星将飞向平台，并撞击其他小行星，使其他小行星也产生相互碰撞（如图8-11所示）。这场景太棒了！注意，请删除刚才手动为玩家使用的小行星设置的速度，并把`playGame`的值改回为`false`；否则，在后面会产生一些问题。

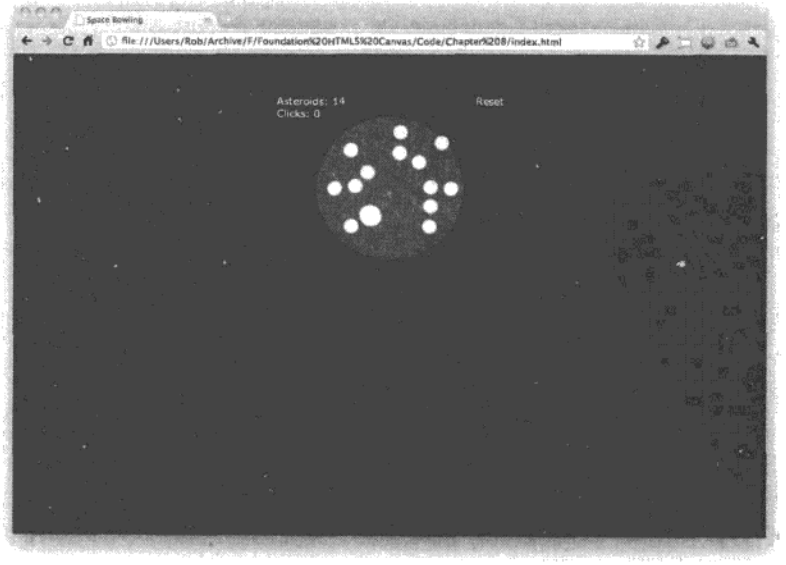


图8-11　在小行星中添加物理属性并产生运动效果

在对象的运动和碰撞检测方面，这就是我们需要介绍的全部内容。但你仍然无法在比赛中确定输赢，因为还没有实现计分系统。别担心，很快你就会解决这个问题了。现在，我们先介绍如何通过检测鼠标输入来实现用户交互功能。

8.6 检测用户交互

如前文所述，本游戏的目的是把玩家使用的小行星投向其他小行星，把它们撞离平台。在本节中，首先你将学习如何检测鼠标输入，然后学习如何在游戏中使用鼠标输入来投掷玩家使用的小行星。

投掷玩家使用的小行星的方法是：先在该小行星上向下按住鼠标左键不放，然后把鼠标拖曳到小行星的后面，并同时保持向下按住鼠标不放。释放鼠标按钮时，鼠标的最终位置将用于计算玩家使用的小行星的速度（位置越远，玩家使用的小行星的速度越大）和角度，最终该小行星将飞出去，目标是把其他小行星撞离圆形平台。图8-12可以帮助你理解该过程。

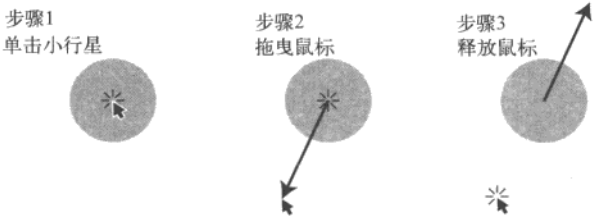


图8-12 玩家使用的小行星的用户界面

8.6.1 建立事件监听器

在计算鼠标位置和速度之前，首先需要建立鼠标事件监听器以及所有即将使用的变量。

在JavaScript文件顶部的playerOriginalY变量下面添加以下变量声明代码：

```
var playerSelected;
var playerMaxAbsVelocity;
var playerVelocityDampener;
var powerX;
var powerY;
```

以上代码中的每个变量都有特定用途，你很快就会明白它们的用处，但这里我先对它们做一些简要说明。playerSelected变量将用于存储一个布尔型的true或false值，以便确定玩家使用的小行星当前是否被选中。playerMaxAbsVelocity变量（用于限制玩家使用的小行星的最快速度）和playerVelocityDampener变量（用于微调速度的计算）用于存储玩家使用的小行星的速度。powerX和powerY中保存的位置（x, y）用于确定该小行星的速度，并在画布上绘制一条直线来表示该速度。

你需要为这些变量设置一些默认值，在startGame函数中的asteroids变量下面设置它们：

```
playerSelected = false;
playerMaxAbsVelocity = 30;
playerVelocityDampener = 0.3;
powerX = -1;
powerY = -1;
```

以上代码中的速度值是随意设置的，但powerX和powerY（在玩家开始游戏后）被重置为特定的值。因为鼠标移动时不可能产生-1这种位置值，所以把两个变量的值设为-1；你没必要对位于跟踪元素范围之外的鼠标移动进行检测。另外，你也不能把它们的值设为0，因为玩家有可能把鼠标移到位置（0, 0）。

为了建立事件监听器，需要在startGame函数中的animate调用上面添加以下代码：

```
$(window).mousedown(function(e) {

});

$(window).mousemove(function(e) {

});

$(window).mouseup(function(e) {

});
```

显然，这些代码不会产生任何视觉效果，但它创建了3个主要的事件监听器，你需要通过它们来检测玩家在什么时候按下鼠标左键（mousedown）、移动鼠标（mousemove）和释放鼠标（mouseup）。

> **注意**　在鼠标事件监听器中使用的是window对象，而不是canvas元素，这是因为还需要检测位于可见画布之外的鼠标运动情况。如果只检测画布上的鼠标输入，当玩家把鼠标移动到画布之外时，游戏就会停止更新鼠标位置，这会让玩家很恼火。

8.6.2　选中玩家使用的小行星

基于鼠标输入计算速度的大部分代码都来自5.4节（当时是用鼠标取得像素值）。唯一不同之处在于，这里需要在代码的顶部再添加少量逻辑语句。

在mousedown事件监听器中添加以下代码，稍后我会解释它们的作用：

```
if (!playerSelected && player.x == playerOriginalX && player.y == playerOriginalY) {
        var canvasOffset = canvas.offset();
        var canvasX = Math.floor(e.pageX-canvasOffset.left);
        var canvasY = Math.floor(e.pageY-canvasOffset.top);

        if (!playGame) {
                playGame = true;
                animate();
```

```
        };

        var dX = player.x-canvasX;
        var dY = player.y-canvasY;
        var distance = Math.sqrt((dX*dX)+(dY*dY));
        var padding = 5;

        if (distance < player.radius+padding) {
                powerX = player.x;
                powerY = player.y;
                playerSelected = true;
        };
};
```

只有当玩家使用的小行星没有被选中，并且玩家使用的小行星位于起点位置时，条件语句后的代码才会执行。当玩家使用的小行星仍然在第一次投掷后运动时，这可以防止玩家再次投掷该小行星。

计算画布上玩家鼠标的相对位置所使用的代码与5.4节使用的代码完全相同，具体原理请参考图5-15。

在第一个条件语句中，还有另外一个条件语句用于检查游戏是否正在进行，如果没有，则启动游戏并运行动画循环。这意味着当屏幕上没有对象运动时，你不会浪费宝贵的系统资源来运行游戏动画。

最后一组代码用于计算玩家使用的小行星与鼠标之间的距离。如果距离小于玩家使用的小行星的半径，那么就可以确定玩家已经单击了它（这与前面的碰撞检测使用的方法实际上是相同的）。为了让玩家容易选中该小行星，这里使用了padding变量，如果你觉得还不够满意，还可以进一步增加padding的值。确定玩家已经单击了该小行星之后，就可以把powerX和powerY变量设置为玩家使用的小行星的原点（使速度变为0，后面将会计算该值），并把playerSelected变量的值设为true，从而把选中该小行星的消息通知给游戏的其他对象。

8.6.3 增加力度

单击小行星以后，玩家可以通过移动鼠标的位置来改变该小行星的力度和角度。实现该功能的大部分代码与刚才创建的mousedown事件监听器代码非常相似。

在mousemove事件监听器代码中添加以下代码：

```
if (playerSelected) {
        var canvasOffset = canvas.offset();
        var canvasX = Math.floor(e.pageX-canvasOffset.left);
        var canvasY = Math.floor(e.pageY-canvasOffset.top);

        var dX = canvasX-player.x;
        var dY = canvasY-player.y;
        var distance = Math.sqrt((dX*dX)+(dY*dY));

        if (distance*playerVelocityDampener < playerMaxAbsVelocity) {
                powerX = canvasX;
```

```
                powerY = canvasY;
        } else {
                var ratio = playerMaxAbsVelocity/(distance*playerVelocityDampener);
                powerX = player.x+(dX*ratio);
                powerY = player.y+(dY*ratio);
        };
};
```

首先你将注意到的是，检查playerSelected的条件语句应该为真。这说明，只有当你确定玩家已经单击了该小行星，并且当前正在向后拖曳鼠标时，该条件语句后面的代码才会运行。

条件语句内部的三行代码用于计算画布上的鼠标的相对位置。

与mousedown事件监听器的用法很相似，然后你将计算鼠标与玩家使用的小行星的原点位置之间的距离。此时，你希望通过这个距离来表示速度的力度（速度的大小）。为此，首先需要使用前面定义的playerVelocityDampener变量来缓冲该距离（如图8-13所示）。通过这种方法，玩家可以把鼠标拖曳得更远一些（以便看到拖出来的力度线），但最终速度会小一些，以便显得更真实。

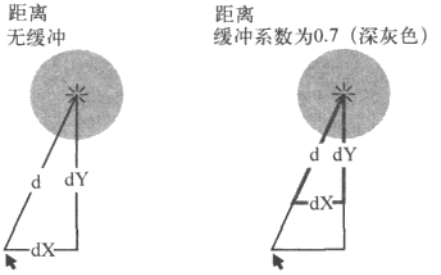


图8-13　使速度得到缓冲

只要缓冲后的速度值小于playerMaxAbsVelocity中设置的最大速度值，那么把鼠标在画布上的（x, y）位置分别作为powerX和powerY的值就是安全的。否则，速度大小就超出了最大允许值，此时你需要把它的值减少到最大允许值，并保持正确的角度。通过计算最大速度值与当前速度值（distance）之间的比值，可以获得该值，然后使用该值来缓冲dX和dY值（每个坐标轴上鼠标和玩家使用的小行星之间的距离），最后把小行星的（x,y）原点值加上缓冲后的值即可。

8.6.4　让玩家使用的小行星动起来

既然已经解决了所有与速度有关的难题，接下来需要使用powerX和powerY的值来计算最终的速度，并把该速度应用于玩家使用的小行星。简单。

在mouseup事件监听器中添加以下代码：

```
if (playerSelected) {
        var dX = powerX-player.x;
        var dY = powerY-player.y;
```

```
        player.vX = -(dX*playerVelocityDampener);
        player.vY = -(dY*playerVelocityDampener);
};

playerSelected = false;
powerX = -1;
powerY = -1;
```

这里的代码比其他鼠标监听器的代码要简单得多。此处只需计算力的位置与玩家使用的小行星之间的距离（将坐标位置转化成速度），然后将该结果应用于玩家使用的小行星，使它产生一个负的速度，同时该速度还受速度缓冲系数的影响。因为你希望该小行星沿着拖曳的相反方向运动，所以你需要使用一个负的速度，这与弹弓的原理非常类似（参见8-12）

最后，所以还需要把playerSelected的值设为false（因为小行星现在不再处于处于被选中状态了），并重新设置powerX和powerY的坐标值。

8.6.5 可视化用户输入

现在所有的对象都可以正常工作了，但游戏的界面还不够友好。实际上，玩家还无法通过某种方式来看到他们在小行星上施加的速度大小，也无法看到该小行星的运动角度。为此，可以通过一条简单的直线来表示速度。

在animate函数中绘制平台的代码后面添加以下代码：

```
if (playerSelected) {
        context.strokeStyle = "rgb(255, 255, 255)";
        context.lineWidth = 3;
        context.beginPath();
        context.moveTo(player.x, player.y);
        context.lineTo(powerX, powerY);
        context.closePath();
        context.stroke();
};
```

以上代码将在玩家使用的小行星和速度的位置之间绘制一条直线（至少是速度的可视化表示）。因为条件语句中playerSelected的值为true，所以只有选中玩家使用的小行星时，才会绘制该直线。由此可见，为了实现某些奇妙的特性，函数和游戏代码需要密切配合，互相通信。

在浏览器中刷新游戏，你将会看到一个完全交互式的游戏界面，你可以把玩家使用的小行星投掷到任何地方。现在还无法获得游戏的分数，而只是让玩家使用的小行星飞出去并把其他小行星从平台上弹开。很好玩！

目前，你只能把玩家使用的小行星投掷一次，并且其他小行星在画布的边缘处会消失，因此游戏还有欠缺！为此，在玩家每次投掷小行星以后，需要重置游戏状态并对玩家使用的小行星进行边界检测。

8.7 重置 player

在每次投掷出玩家使用的小行星以后，为了重置游戏的状态，需要把该小行星移动到起点位置，并把它的速度重新设置为0。这样，在玩家再次选中该小行星之前，它将一直保持静止状态。你需要在不同的代码位置重置player，因此，最好把所有代码封装在一个函数中。

在startGame函数前面添加以下代码：

```
function resetPlayer() {
        player.x = playerOriginalX;
        player.y = playerOriginalY;
        player.vX = 0;
        player.vY = 0;
};
```

以上代码不需要过多解释，因为它们的作用前面已经描述过了——首先重新设置位置，然后把速度设为0。

只有当玩家使用的小行星离开画布的范围之外，或者被投掷以后停止运动时，才需要重新设置玩家使用的小行星。所幸，我们已经把重置代码封装在一个函数中了，因此只需要在animate函数中绘制每颗小行星的代码上面添加以下代码即可：

```
if (player.x != playerOriginalX && player.y != playerOriginalY) {
        if (player.vX == 0 && player.vY == 0) {
                resetPlayer();
        } else if (player.x+player.radius < 0) {
                resetPlayer();
        } else if (player.x-player.radius > canvasWidth) {
                resetPlayer();
        } else if (player.y+player.radius < 0) {
                resetPlayer();
        } else if (player.y-player.radius > canvasHeight) {
                resetPlayer();
        };
};
```

在以上代码中，一组条件语句首先将检查玩家使用的小行星是否已经离开了起始位置，具体实现方法是：查看小行星的起始位置与当前位置是否不同。如果该小行星已经运动了，那么再使用一组条件语句来执行各种检查。首先检查该小行星的速度是否减小为0，此时该小行星将停止运动。其他检查用于查看该小行星是否移动到画布的边界之外。如果这些检查中任何一个条件为真，那么就需要重置玩家使用的小行星，以便玩家再次进行投掷。

> **注意** 你可能会把重置player的条件语句组合成一个更大的语句，这是由你自己决定的。但我不希望把它们混淆在一起。

刷新浏览器，你将发现，当完成第一次投掷以后，玩家使用的小行星将会重新移动到它的起始位置（如图8-14所示）。

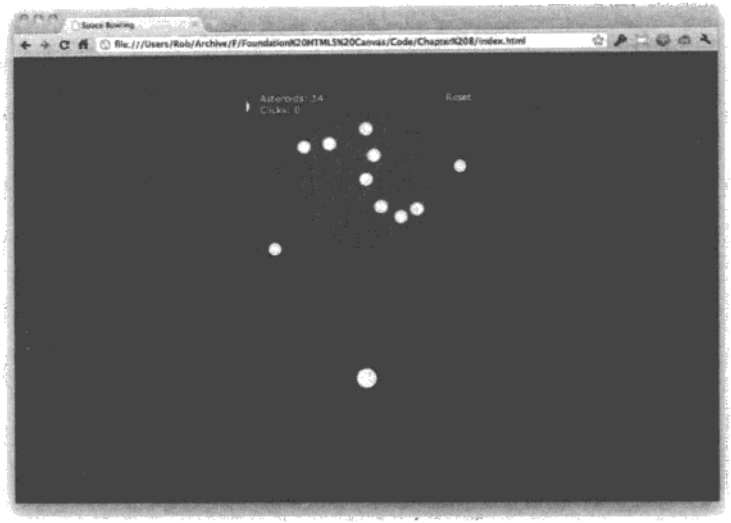


图8-14 重置玩家使用的小行星

8.8 玩家获胜

到目前为止，游戏还没有什么实质意义，因为你还没有给出游戏获胜的标准。因此，你需要实现某种形式的计分系统。在这个游戏中，可以根据玩家把所有小行星弹离平台之外所需要使用的投掷（单击）次数来设计计分系统。为此，你需要知道已经有多少小行星被弹出平台，以及玩家已经投掷了多少次（分数）。在每次投掷以后，都更新UI来显示玩家的当前分数。当所有的小行星都被弹出平台之后，游戏将结束，并显示玩家的最后得分，即他们获胜。

8.8.1 更新分数

首先需要创建存储分数的变量，在JavaScript代码顶部的powerY变量下面添加以下声明：

```
var score;
```

在startGame函数顶部的powerY变量下面添加以下代码，把默认分数设置为0：

```
score = 0;
```

玩家每次单击小行星时，都需要更新并显示分数，为此，在mouseup事件监听器中的player.vY属性下面添加以下代码：

```
uiScore.html(++score);
```

这样，分数将每次增加1，并同时更新分数的UI元素。如果你试着玩该游戏并注意观察位于画布左上角的分数（“Clicks”标签），将会看到该效果（如图8-15所示）。

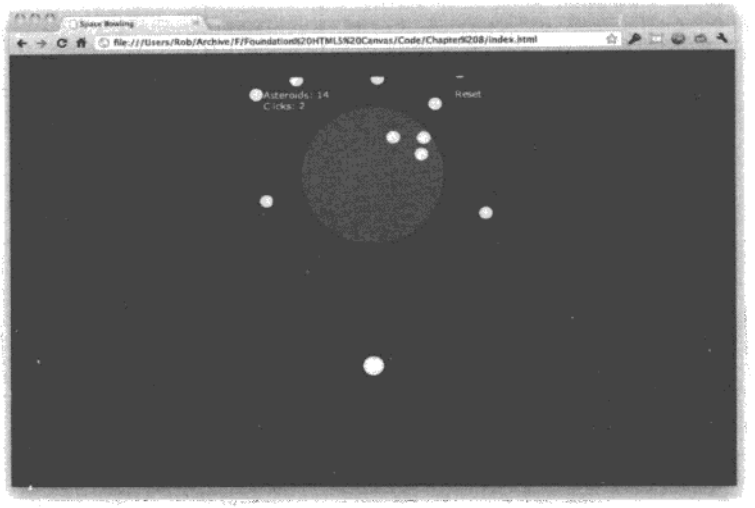


图8-15　每次单击时更新分数

8.8.2　从平台上删除小行星

为了在游戏中获胜，必须把平台上所有的小行星都弹出去，所以需要通过一种方法来判断所有的小行星是否都已经离开平台。这种方法包含两个步骤，第一步是通过一个简单的检查来查看每颗小行星是否位于平台上，第二步是通过一个例程（routine）来跟踪平台上的小行星数目。如果第二步返回的值为0（已经没有小行星留在平台上），那么就能判断游戏结束，并进入游戏的下一个屏幕。

当然，也可以采用其他方法来跟踪平台上的小行星数目。在下面你将使用的方法中，当任何一颗小行星从平台上掉落时，我们就从asteroids数组中把它删除。这样，只需检查asteroids数组的长度就可以判断平台上剩下的小行星数目。

但是，这里还存在一个问题：你还不能立刻从数组中把小行星删除，因为这会破坏animate函数中的小行星循环。由于循环需要运行一定的次数（小行星的数目），所以如果在循环中途删除一颗小行星，那么就会减少循环次数。因此，不能直接删除小行星。

为了解决这个问题，需要创建一个新的临时数组，在动画循环中用于存储从平台上掉下来的小行星，然后在动画循环的末尾处用该数组从原始asteroids数组中删除小行星。通过代码示例来解释该操作也许更容易理解一些。

在animate函数中的steroidsLength变量上面添加以下代码：

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

这是一个临时数组，用于存储从平台上掉落并消失的小行星。虽然这有些麻烦，但效果很好。接下来的步骤就是检查循环中当前存在的小行星是否已经从平台上掉落。

在animate函数的摩擦力代码下面添加以下代码：

```
if (!tmpAsteroid.player) {
        var dXp = tmpAsteroid.x - platformX;
        var dYp = tmpAsteroid.y - platformY;
        var distanceP = Math.sqrt((dXp*dXp)+(dYp*dYp));
        if (distanceP > platformOuterRadius) {
                if (tmpAsteroid.radius > 0) {
                        tmpAsteroid.radius -= 2;
                } else {
                        deadAsteroids.push(tmpAsteroid);
                };
        };
};
```

以上代码中有几个关键步骤。第一步是执行初步检查，确保当前处理的小行星不是玩家使用的小行星（被投掷的小行星），你当然不希望删除该小行星。第二步是计算小行星与平台中心的距离。第三步是检查小行星的中心是否位于平台边界之外。

在第三步检查中还包含另外一组代码来处理那些位于平台之外的小行星。仅仅让小行星突然消失在平台的边界处还不够逼真，我们可以让它逐渐消失，这样看上去就更像是从真实的平台上掉下来。为了实现该目标，只需要使用一个简单的检查就可以了。如果小行星的半径大于0，那么它将仍然可见，因此，我们让小行星的半径减少2像素。这样，每个动画循环都会让半径减少2像素，最终会形成小行星逐渐消失的效果。如果小行星的半径是0或小于0（经过一些循环之后），就可以确认该小行星已经从平台上掉落，并将其添加到deadAsteroids数组中。

运行游戏，你会看到，当小行星从平台上掉下去时，它在逐渐变小并消失（如图8-16所示）。

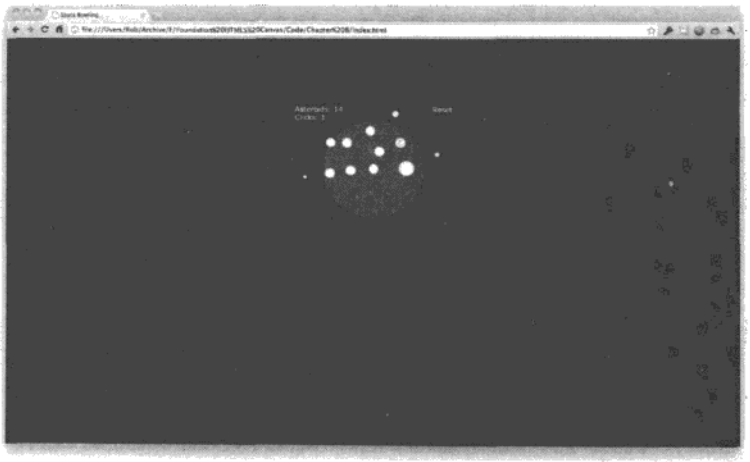


图8-16 让小行星离开平台时产生一种逐渐消失的效果

接下来的任务是从原始的asteroids数组中删除掉落的小行星，当平台上不存在小行星时，让玩家知道游戏已经结束。

在animate函数末尾的setTimeout调用的条件语句上面添加以下代码：

```
var deadAsteroidsLength = deadAsteroids.length;
if (deadAsteroidsLength > 0) {
        for (var di = 0; di < deadAsteroidsLength; di++) {
                var tmpDeadAsteroid = deadAsteroids[di];
                asteroids.splice(asteroids.indexOf(tmpDeadAsteroid), 1);
        };
};
```

这段代码运行在小行星循环之外。因此，在每次更新、碰撞检测和移动小行星之后它才开始运行。首先，它检查是否存在消失的小行星需要处理。如果不存在，就可以开始玩游戏了，让玩家迈出比赛获胜的第一步。

如果存在一些消失的小行星需要处理，可以循环遍历这些小行星，并把每颗小行星赋给临时变量tmpDeadAsteroid。因为deadAsteroids数组中的小行星对象与asteroidsArray数组中的对象是相同的，所以可以使用一个叫做indexOf的特殊数组方法来计算原始asteroids数组中消失的小行星的索引，最后通过这些索引，可以删除消失的小行星。这也许让你觉得有些迷惑，所以需要仔细思考一下。在前面的代码清单中，你已经把tmpAsteroid变量中存储的小行星添加到deadAsteroids数组中。deadAsteroids数组中的小行星其实并不是副本，它实际指向原始asteroids数组中的小行星。坦率地说，我并不打算让你完全理解这些原理，因为这有些复杂。你只需要记住这些要点就可以了。

为了从asteroids数组中删除小行星，需要使用Array对象的splice方法。通过它，你可以从一个数组中删除一个或多个元素，删除的元素从第一个参数定义的索引位置开始，并一直持续到第二个参数定义的索引位置为止。

你需要使用的起始索引是你希望删除的小行星的索引位置。为此，需要使用Array对象的indexOf方法。这个方法可以有效地从数组中搜索特定项，如果查找到满足条件的项，indexOf方法将返回该项在数组中对应的索引位置。在这里，你希望查找满足条件的小行星，因此在indexOf方法中可以把tmpDeadAsteroid变量作为参数。因为你只需要删除一个小行星，所以调用splice的第二个参数可以设为1。

综上所述，这些调用方法提供了充足的信息，你可以利用这些信息从原始的asteroids数组中准确删除那些消失的小行星。

最后，你还需要更新UI中剩下的小行星数目，当平台上不存在小行星时，还要通过某种方式来表示玩家在游戏中获胜。

在上一段代码中的for循环后面添加以下代码：

```
var remaining = asteroids.length-1; // 不算玩家投掷的小行星
uiRemaining.html(remaining);

if (remaining == 0) {
        //获胜!
        playGame = false;
        uiStats.hide();
        uiComplete.show();
```

```
        $(window).unbind("mousedown");
        $(window).unbind("mouseup");
        $(window).unbind("mousemove");
};
```

第一行代码声明了一个变量，用于存储平台上剩下的小行星数目。注意，需要把小行星的数目减1，因为你不能把玩家使用的小行星也统计在内。然后，使用该变量更新游戏的UI，并显示游戏中剩余的小行星数目。

为了确认玩家是否已经获胜，只需执行一个条件语句来检查剩余的小行星数目是否为0。如果为0，需要停止游戏，隐藏游戏的统计UI，并显示游戏结束屏幕。最后三行代码仅用于删除鼠标事件监听器，除非玩家需要重新开启游戏，否则将不再需要使用它们。

刷新并试试游戏吧（如图8-17所示）。经过多次尝试，我的最好成绩是2次单击！

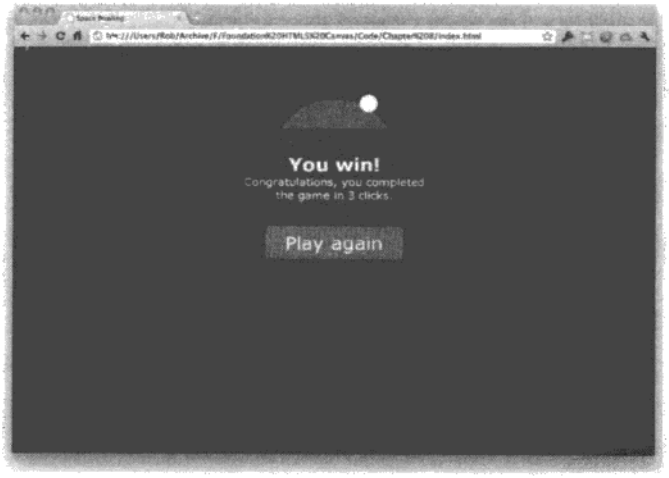
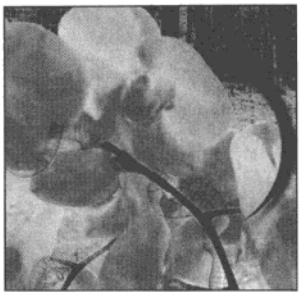


图8-17 结束的游戏界面

8.9 小结

你完成了画布中的第一个游戏。这非常神奇，不是吗？如果我是你，我一定会感到非常自豪。实际上，我也为你感到骄傲。本章的内容有一定难度，但我的初衷是让你充分了解如何把游戏的各个部分组合在一起，从而产生一些有趣的交互式游戏效果。如果你以前没有意识到，那么我相信你现在一定意识到了在游戏编程中需要付出怎样的努力。

本章需要掌握的要点是，通过检测用户的鼠标输入，创建并操纵用户界面，以及在游戏中创建玩家获胜的场景。在下一章要创建的游戏中，你还会用到包含这些方法在内的其他更多方法。与本章不同的是，在下一章的游戏中，玩家将要通过键盘来控制一个火箭。那就更棒了！



第 9 章

躲避小行星游戏

本章将创建一个躲避小行星游戏：在创建游戏的过程中，需要用到上一章的相关知识，同时本章还会介绍一些新的技巧。另外，躲避小行星游戏与上一章中的游戏在操控方面有很大的区别，玩家拥有更多的控制能力。

本章将重点介绍的两个方面是：如何使用JavaScript来检测键盘输入，以及如何在游戏中使用和处理HTML5音频。希望你能够喜欢！

9.1 游戏概述

顾名思义，躲避小行星游戏的目标是非常明显的：当小行星向你冲来时，让火箭飞行和生存的时间尽可能长一些（如图9-1所示）。如果你碰上某颗小行星，游戏将结束，游戏的分数是通过火箭生存的时间来计算的。与保龄球游戏一样，虽然躲避小行星游戏不是大型游戏，但却非常让人着迷。

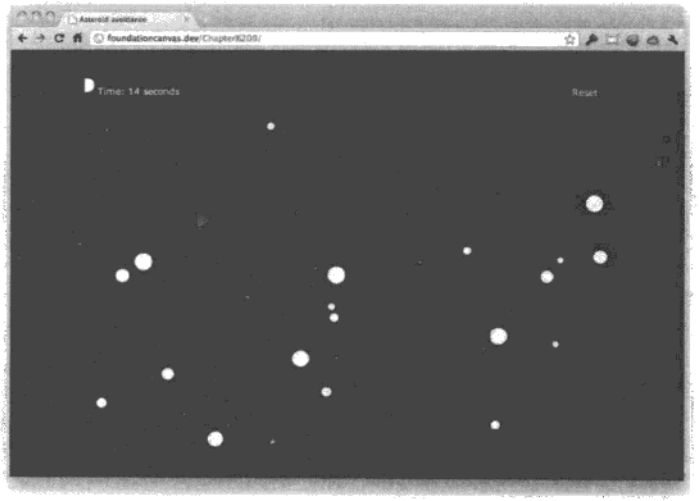


图9-1 躲避小行星游戏

躲避小行星游戏与你在上一章创建的太空保龄球游戏有很大的区别。最主要的区别是躲避小行星游戏是一个“横向卷轴式”游戏，或者说它至少类似于这样的游戏。上一章创建的游戏更多体现的是一种静态场景，而本章创建的游戏将会侧重于动态场景。

这两个游戏的另一个主要区别体现在用户输入方面。在保龄球游戏中，玩家通过鼠标输入，而这里的游戏将采用键盘输入。当然，这里的游戏也可以通过鼠标进行控制，但我认为你应该学习其他一些游戏交互方式，这很有必要。

需求

躲避小行星游戏将重用上一章使用的大部分核心逻辑：动画循环、一些游戏对象和主要函数。另外，除了一些明显的变化之外，躲避小行星游戏中使用的HTML和UI与上一章非常相似。

本章将学习一些新技巧，其中包括检测键盘输入，以及如何在游戏中使用HTML5音频来创建声音效果。我们将把这些新技巧与本书已经介绍的其他技巧结合起来，例如，在画布中绘制图形、动画、物理学知识和游戏逻辑。

9.2 核心功能

如前文所述，本游戏使用的大部分核心逻辑都基于上一章中的保龄球游戏逻辑。这看上去好像是在偷懒，其实正是由于这种代码重用技术才加快了游戏开发的进程，并且最终可以通过这些代码重用技术来创建一个健壮的引擎，以后就可以使用该引擎作为游戏开发的基础。你很难找到一款功能完全不同的游戏。既然这样，为什么要做重复工作呢？

9.2.1 创建 HTML 代码

本游戏使用的HTML和上一章游戏所使用的HTML代码几乎完全相同，因此我们直接给出代码。在你的计算机中创建一个新的文件目录，新建一个index.html文件并插入以下代码：

```
<!DOCTYPE html>

<html>
        <head>
                <title>Asteroid avoidance</title>
                <meta charset="utf-8">

                <link href="game.css" rel="stylesheet" type="text/css">

                <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js"></script>
                <script type="text/javascript" src="game.js"></script>
        </head>

        <body>
                <div id="game">
                        <div id="gameUI">
                                <div id="gameIntro">
```

```
                                        <h1>Asteroid avoidance</h1>
                                        <p>Click play and then press any key to
start.</p>
                                        <p><a id="gamePlay" class="button"
    href="">Play</a></p>
                                </div>
                                <div id="gameStats">
                                        <p>Time: <span class="gameScore"></span>
seconds</p>
                                        <p><a class="gameReset"
href="">Reset</a></p>
                                </div>
                                <div id="gameComplete">
                                        <h1>Game over!</h1>
                                        <p>You survived for <span
class="gameScore"></span>
    seconds.</p>
                                        <p><a class="gameReset button" href="">Play
    again</a></p>
                                </div>
                        </div>
                        <canvas id="gameCanvas" width="800" height="600">
                                <!-- 在此插入后备代码 -->
                        </canvas>
                </div>
        </body>
    </html>
```

以上代码非常简单。只是页面的标题、内容和画布的大小发生了改变。除了这些不同之处以外，本游戏的其他内容和上一章非常相似（如图9-2所示）。

图9-2　使用HTML创建游戏标记

9.2.2 美化界面

创建一个名为game.css的新文件，并把它和HTML文件放在相同的目录下。在该CSS文件中插入以下代码：

```
* { margin: 0; padding: 0; }
html, body { height: 100%; width: 100%; }
canvas { display: block; }

body {
        background: #000;
        color: #fff;
        font-family: Verdana, Arial, sans-serif;
        font-size: 18px;
}

h1 {
        font-size: 30px;
}

p {
        margin: 0 20px;
}

a {
        color: #fff;
        text-decoration: none;
}

a:hover {
        text-decoration: underline;
}

a.button {
        background: #185da8;
        border-radius: 5px;
        display: block;
        font-size: 30px;
        margin: 40px 0 0 270px;
        padding: 10px;
        width: 200px;
}

a.button:hover {
        background: #2488f5;
        color: #fff;
        text-decoration: none;
}

#game {
        height: 600px;
        left: 50%;
        margin: -300px 0 0 -400px;
        position: relative;
        top: 50%;
        width: 800px;
```

```
}

#gameCanvas {
        background: #001022;
}

#gameUI {
        height: 600px;
        position: absolute; /* 将UI放在画布上方*/
        width: 800px;
}

#gameIntro, #gameComplete {
        background: rgba(0, 0, 0, 0.5);
        margin-top: 100px;
        padding: 40px 0;
        text-align: center;
}

#gameStats {
        font-size: 14px;
        margin: 20px 0;
}

#gameStats .gameReset {
        margin: 20px 20px 0 0;
        position: absolute;
        right: 0;
        top: 0;
}
```

和前面的HTML代码一样，这里的CSS代码也没有发生多大变化。唯一的区别是CSS规则的#game和#gameUI尺寸发生了改变。这将调整UI的视觉风格，使用户界面合理地呈现在一个更大的画布上（如图9-3所示）。

图9-3　使用CSS设置UI的样式

9.2.3 编写 JavaScript 代码

和保龄球游戏一样，本游戏的最后一个阶段是添加JavaScript代码。创建一个名为game.js的新文件，并把它和HTML及CSS文件放在相同的目录下。在该js文件中插入以下代码：

```
$(document).ready(function() {
        var canvas = $("#gameCanvas");
        var context = canvas.get(0).getContext("2d");

        //画布尺寸
        var canvasWidth = canvas.width();
        var canvasHeight = canvas.height();

        //游戏设置
        var playGame;

        //游戏UI
        var ui = $("#gameUI");
        var uiIntro = $("#gameIntro");
        var uiStats = $("#gameStats");
        var uiComplete = $("#gameComplete");
        var uiPlay = $("#gamePlay");
        var uiReset = $(".gameReset");
        var uiScore = $(".gameScore");

        //重置和启动游戏
        function startGame() {
                //重置游戏状态
                uiScore.html("0");
                uiStats.show();

                //初始游戏设置
                playGame = false;

                //开始动画循环
                animate();
        };
        //初始化游戏环境
        function init() {
                uiStats.hide();
                uiComplete.hide();

                uiPlay.click(function(e) {
                        e.preventDefault();
                        uiIntro.hide();
                        startGame();
                });

                uiReset.click(function(e) {
                        e.preventDefault();
                        uiComplete.hide();
                        startGame();
```

```
            });
        };
        //动画循环，游戏的趣味性就在这里
        function animate() {
            //清除
            context.clearRect(0, 0, canvasWidth, canvasHeight);

            if (playGame) {

            //33毫秒后再次运行动画循环
            setTimeout(animate, 33);
        };
        };
        init();
});
```

虽然这些代码看上去有些复杂，实际上它却和上一章的逻辑代码几乎是相同的。唯一的区别是现在的代码已经包含了核心UI逻辑，而不是在一个独立的部分添加这些逻辑代码。不管怎样，这里的UI代码和前面是相同的，因此你应该很容易理解。

在你最喜欢的浏览器中运行该游戏，应该会看到一个更加美观的UI（如图9-4所示）。另外，你还可以单击Play按钮来显示游戏的主窗口，尽管它看上去也许还有些单调（如图9-5所示）。

图9-4　使用JavaScript隐藏部分UI

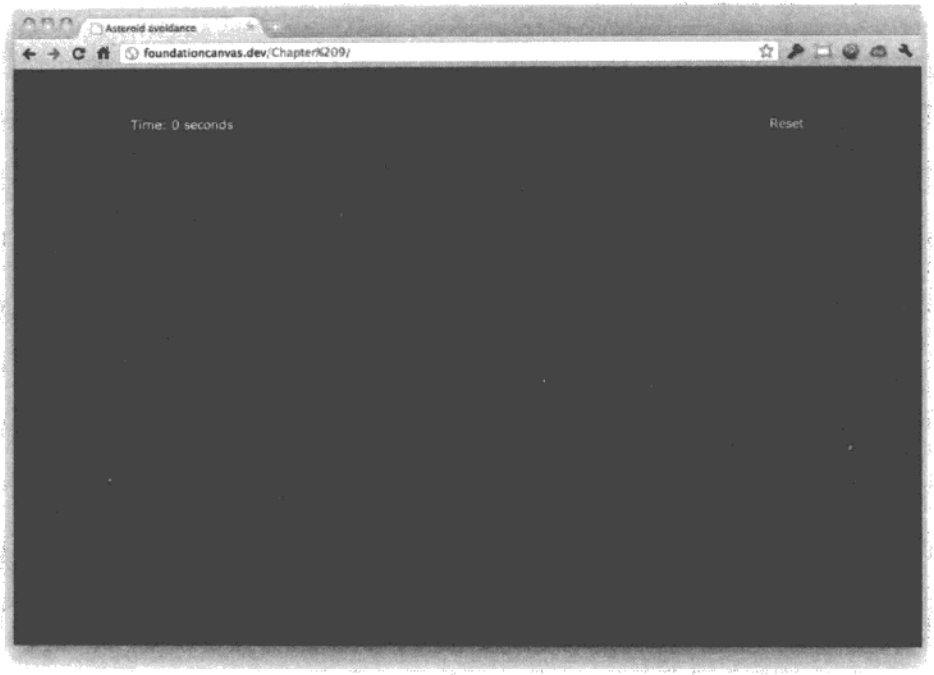


图9-5 查看当前空白的游戏主窗口

9.3 创建游戏对象

躲避小行星游戏使用两个主要对象：小行星和玩家使用的火箭。我们将使用JavaScript类来创建这些对象。你也许会觉得奇怪，既然玩家只有一个，为什么还要通过一个类来定义它呢？简而言之，如果你以后需要在游戏中添加多个玩家，通过类创建玩家就会更容易一些。

9.3.1 创建小行星

通过类创建游戏对象意味着你可以在其他游戏中非常方便地重用和改变它们的用途。例如，在躲避小行星游戏中，除了少数变化之外，所使用的小行星类与保龄球游戏中的小行星类几乎完全相同。

第一步是声明主要变量，我们将使用这些变量来存储所有的小行星。同时，还需要声明另外一个变量，用于计算游戏中应该存在的小行星数目。在JavaScript代码顶部的`playGame`变量下面添加以下代码：

```
var asteroids;
var numAsteroids;
```

稍后你将会为这些变量赋值，但现在我们只建立小行星类。在startGame函数上面添加以下代码：

```
var Asteroid = function(x, y, radius, vX) {
        this.x = x;
        this.y = y;
        this.radius = radius;
        this.vX = vX;
};
```

学习了上一章游戏的创建过程，你应该非常熟悉以上代码，这里的代码要简短得多。因为在本游戏中你不需要利用太多物理学知识使小行星产生动画效果，所以不需要使用质量或摩擦力的任何属性。另外，这里仅存在一个速度属性，这是因为小行星只需要从右向左运动，即只需要x速度。这里不需要y速度，所以就省略了。

在开始创建所有小行星之前，需要建立数组来存储这些小行星，并声明实际需要使用的小行星数目。在startGame函数中的playGame变量下面添加以下代码：

```
asteroids = new Array();
numAsteroids = 10;
```

你也许认为10个小行星是一个很小的数目，但是当这些小行星在屏幕上消失时，你将重复使用它们，所以在游戏中你实际看到的小行星数目可以有无穷多个。你可以把这里的小行星数目看做屏幕上某一时刻可能出现的小行星总数。

创建小行星的过程其实就是一个创建循环的过程，循环的次数就是你刚才声明的小行星数目。在你刚才赋值的numAsteroids变量下面添加以下代码：

```
for (var i = 0; i < numAsteroids; i++) {
        var radius = 5+(Math.random()*10);
        var x = canvasWidth+radius+Math.floor(Math.random()*canvasWidth);
        var y = Math.floor(Math.random()*canvasHeight);
        var vX = -5-(Math.random()*5);

        asteroids.push(new Asteroid(x, y, radius, vX));
};
```

为了让每颗小行星的外观都与众不同，并且使游戏看上去更有趣一些，可以把小行星的半径设为一个介于5到15像素之间的随机数（5加上一个介于0到10之间的随机数）。虽然x速度的值介于-5到-10之间，但你也可以采用同样的方法来设置它（-5减去一个0到5之间的数）。因为你需要让小行星按从右向左的方向运动，所以使用的是一个负的速度值，这说明x的位置将随着时间的推移而减小。

计算每颗小行星的x位置看上去有些复杂，但其实非常简单。在开始启动游戏的时候，如果让所有的小行星全部显示在屏幕上，就让人觉得有些太奇怪了。因此在游戏开始之前，最好把小行星放在屏幕的右侧，当游戏开始时才让它们按从右向左的顺序穿过屏幕。

为此，首先需要把x位置设为canvas元素的宽度，然后加上小行星的半径。这意味着如果你现在画出小行星，那么它应该位于屏幕的右侧。如果仅仅这样做，那么所有的小行星将会排成一

行，所以下一步我们需要把*x*位置加上一个介于0到画布宽度之间的随机值（如图9-6所示）。与*x*位置相比，*y*位置简单一些，它只是一个介于0到画布高度之间的随机值。

这样可以产生一个与画布尺寸相同的方框，方框中随机分布着一些小行星。当游戏开始时，这些小行星将穿过可见的画布。相信我，实现这种效果其实并不复杂。

最后一步是把一个新的小行星推送到数组中，做好移动和绘制小行星的准备。

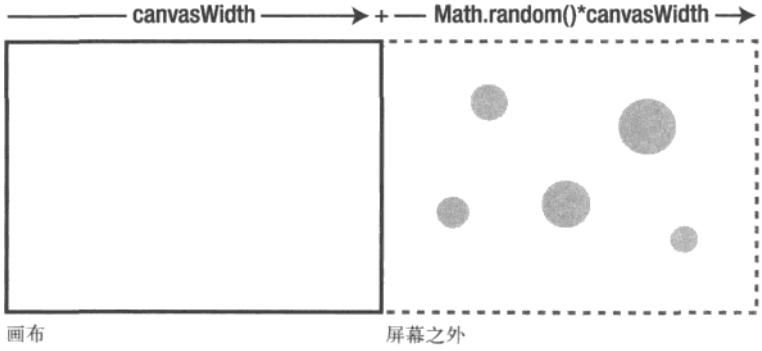


图9-6 让小行星停留在屏幕之外

9.3.2 创建玩家使用的火箭

在上一章的游戏中，玩家控制的仅仅是一个小行星。这太单调了。在本游戏中，玩家将控制一个奇妙的火箭飞船，因此游戏要有趣得多！

让我们直接进入主题吧。首先声明用于建立玩家的初始化变量。在JavaScript顶部的numAsteroids变量下面添加以下代码：

```
var player;
```

该变量将用于存储玩家对象的引用，但现在我们还没有定义玩家对象。在Asteroid类下面添加以下代码：

```
var Player = function(x, y) {
        this.x = x;
        this.y = y;
        this.width = 24;
        this.height = 24;
        this.halfWidth = this.width/2;
        this.halfHeight = this.height/2;

        this.vX = 0;
        this.vY = 0;
};
```

通过上一章对Asteroid类的学习，你应该熟悉以上代码的某些部分，例如位置和速度属性。其余属性用于描述玩家使用的火箭的尺寸，包括整个尺寸和一半的尺寸。绘制火箭和执行碰撞检测时，你需要使用这些尺寸。

最后一步是创建一个新的玩家对象。为此，在startGame函数中的numAsteroids变量下面添加以下对象：

```
player = new Player(150, canvasHeight/2);
```

通过以上代码，玩家的位置将垂直居中，并且距离画布左边界150像素。

现在还不能看到任何效果，稍后当你开始着手移动所有的游戏对象时，将会从视觉上看到这种效果。

9.4　检测键盘输入

在保龄球游戏中，你是通过鼠标输入来控制游戏的，而本游戏将使用键盘来控制游戏。更确切地说，你将使用方向箭来四处移动玩家使用的火箭。如何才能实现这种控制呢？这比控制鼠标输入更难吗？不，其实非常简单。下面我来教你怎么做。

9.4.1　键值

在处理键盘输入时，首先需要知道哪一个按键被按下了。在JavaScript中，普通键盘上的每一个按键都分配了一个特定的键值（key code）。通过这些键值，可以唯一确定按下或释放了哪个键。稍后你将学习如何使用键值，现在我们首先需要了解每个按键所对应的数值。

例如，键*a*到*z*（无论在什么情况下）对应的键值分别是从65到90。箭头键对应的键值是从37到40，其中左箭头的键值是37、上箭头的键值是38、右箭头的键值是39、下箭头的键值是40。空格键的键值是32。

> 如果需要了解更多有关键值的信息，建议你查看QuirksMode网站上的文章 [www.quirksmode.org/js/keys.html]。

在躲避小行星游戏中，你需要重点关注的是箭头键，因此在JavaScript代码顶部的player变量下面添加以下代码：

```
var arrowUp = 38;
var arrowRight = 39;
var arrowDown = 40;
```

以上代码为每个箭头对应的键值分别分配了一个变量。这种方法称作枚举（enumeration），它是对值进行命名的过程。这主要是为后面的工作提供方便，因为通过这些名称你能很容易确定变量引用的是哪个箭头键。

请注意，为什么没有为左箭头声明一个变量呢？因为你不会手动地让玩家向后移动。相反，当玩家没有按任何按键时，就会表现为向后移动的状态。稍后你就会明白其中的道理。

9.4.2　键盘事件

在向游戏中添加键盘交互效果之前，首先需要确定玩家在何时按下或释放某个按键。为此，

需要使用keydown和keyup事件监听器，这与上一个游戏中使用的mousedown和mouseup事件监听器非常类似。

在startGame函数中的animate函数调用上面(在创建所有小行星的循环下面)添加以下代码：

```
$(window).keydown(function(e) {

});

$(window).keyup(function(e) {

});
```

这看上去与上一章使用的鼠标事件监听器很相似，不是吗？按下某个按键时将触发第一个监听器，释放某个按键时将触发第二个监听器。非常简单。

稍后我们将在这些事件监听器中添加一些有用的代码，但首先需要在重新设置游戏时删除这些监听器，这能防止玩家由于无意按下某个按键而启动游戏。在uiReset.click事件监听器中的startGame调用上面添加以下代码：

```
$(window).unbind("keyup");
$(window).unbind("keydown");
```

接下来，还需要添加一些在激活玩家后用到的属性。在Player类的末尾添加以下代码：

```
this.moveRight = false;
this.moveUp = false;
this.moveDown = false;
```

通过这些属性，你可以知道玩家的移动方向，这些属性值的设置取决于玩家按下了哪个按键。现在你是不是已经理解了其中的所有道理呢？

最后，需要向键盘事件监听器中添加一些逻辑。首先，在keydown事件监听器中添加以下代码：

```
var keyCode = e.keyCode;

if (!playGame) {
        playGame = true;
        animate();
};
if (keyCode == arrowRight) {
        player.moveRight = true;
} else if (keyCode == arrowUp) {
        player.moveUp = true;
} else if (keyCode == arrowDown) {
        player.moveDown = true;
};
```

并在keyup事件监听器中添加以下代码：

```
var keyCode = e.keyCode;

if (keyCode == arrowRight) {
        player.moveRight = false;
} else if (keyCode == arrowUp) {
        player.moveUp = false;
} else if (keyCode == arrowDown) {
        player.moveDown = false;
};
```

以上代码的作用非常明显，但我还需要作一些说明。在两个监听器中，第一行的作用都是把按键的键值赋给一个变量。然后在一组检查语句中使用该键值来判断是否按下了某个箭头键，如果按下了箭头键，判断是哪个箭头键。这样，我们就可以启动（如果按下了该键）或禁用（如果释放了该键）玩家对象的对应属性。

例如，如果按下了向右的箭头键，那么玩家对象的moveRight属性将被设为true。如果释放了该方向键，则moveRight属性将被设为false。

> **注意**　如果玩家一直按住某个按键，那么将触发多个keydown事件。因此，代码要具备处理多个被触发的keydown事件的能力，这一点非常重要。在每个keydown事件之后不一定总是一个keyup事件。

另外还要注意的是，在keydown事件监听器中是如何通过一个条件语句来查看游戏当前是否正在进行的。如果玩家没有做好游戏准备，该语句将阻止游戏运行。只有玩家按下键盘上的某个键时，才会启动游戏。方法很简单，但却非常有效。

游戏中的键盘输入非常多，我们不可能一一列举。在下一节中，我们将通过这些输入来控制玩家沿着正确的方向运动。

9.5　让对象运动起来

现在你已经做好了实现游戏对象动画的所有准备。当你实际看到游戏效果时，这一切会变得更有趣。好极了！

第一步是更新所有游戏对象的位置。我们从更新小行星对象的位置开始，在animate函数中画布的clearRect方法下面添加以下代码：

```
    var asteroidsLength = asteroids.length;
    for (var i = 0; i < asteroidsLength; i++) {
            var tmpAsteroid = asteroids[i];

            tmpAsteroid.x += tmpAsteroid.vX;

            context.fillStyle = "rgb(255, 255, 255)";
            context.beginPath();
            context.arc(tmpAsteroid.x, tmpAsteroid.y, tmpAsteroid.radius, 0, Math.PI*2,
true);
            context.closePath();
            context.fill();
    };
```

这些代码非常简单，与上一章没有什么区别。主要是遍历每一颗小行星，并根据速度来更新它的位置，然后在画布上绘制小行星。

刷新浏览器查看效果（记住按下某个按键启动游戏）。应该能够看到某颗小行星带穿越屏幕的场景（如图9-7所示）。注意它们是如何消失在屏幕左侧的。下一节将学习如何在横向滚动的屏幕上阻止它们的运动。

迄今为止，假设这些小行星都实现了我们的预期效果。接下来还需要更新并显示玩家！在animate函数中刚才添加小行星代码的下面再添加以下代码：

```
player.vX = 0;
player.vY = 0;

if (player.moveRight) {
        player.vX = 3;
};

if (player.moveUp) {
        player.vY = -3;
};

if (player.moveDown) {
        player.vY = 3;

};
player.x += player.vX;
player.y += player.vY;
```

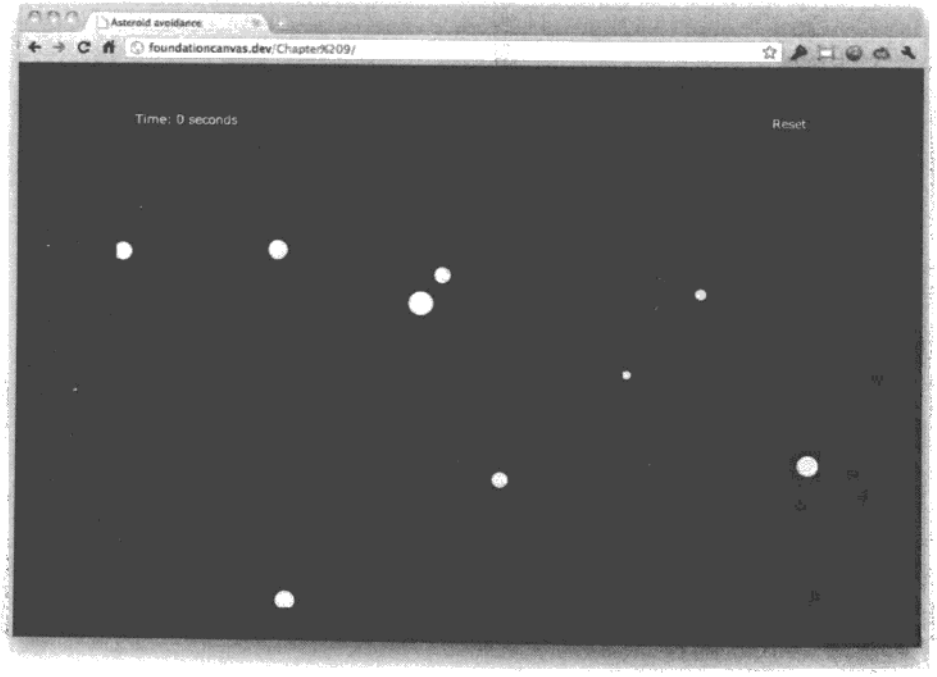


图9-7 绘制小行星带和实现动画

以上代码将更新玩家的速度，并将速度设置为一个特定的值，该值由玩家移动的方向来确定。如果玩家需要向右移动，那么速度值为*x*轴上的3像素。如果玩家需要向上移动，那么速度值为*y*轴上的-3像素。同样，如果玩家需要向下移动，那么速度值即为*y*轴上的3像素。这非常简单。另外，还需要注意如何在代码的开始处重置速度值。如果玩家没有按下任何按键，该语句将阻止玩家移动。

最后，还需要根据速度来更新玩家的*x*和*y*位置。现在你还看不到任何效果，但你已经做好了在屏幕上绘制火箭的所有准备工作。

在刚才添加的代码下面直接添加以下代码：

```
context.fillStyle = "rgb(255, 0, 0)";
context.beginPath();
context.moveTo(player.x+player.halfWidth, player.y);
context.lineTo(player.x-player.halfWidth, player.y-player.halfHeight);
context.lineTo(player.x-player.halfWidth, player.y+player.halfHeight);
context.closePath();
context.fill();
```

你知道以上代码的作用吗？很明显，你正在绘制一条填充路径，但你能告诉我绘制的路径是什么形状吗？你也可以不思考，直接查看图9-8，这些图形看上去很普通。是的，它只是一个三角形而已。

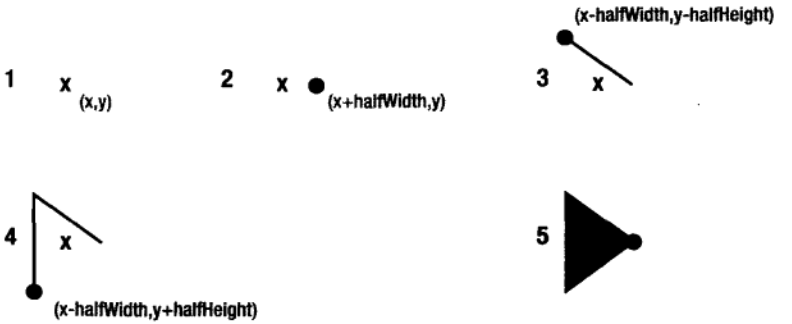


图9-8　绘制玩家火箭上的三角形

如果你仔细查看图9-8，会发现玩家对象的尺寸属性的作用。知道了玩家对象的宽度值和高度值的一半，就可以构建一个动态三角形，它能随着尺寸值的变化变大或缩小。方法很简单，但效果却很好。

在浏览器中查看游戏的效果，应该能够看到玩家使用的火箭（如图9-9所示）。试着按下箭头键。看到火箭移动了吗？现在的游戏效果已经非常棒了。

这里可以只使用运动逻辑，但游戏会显得有些单调。我们不妨在火箭上再添加一团闪动的火焰！在Player类的末尾添加以下代码：

```
this.flameLength = 20;
```

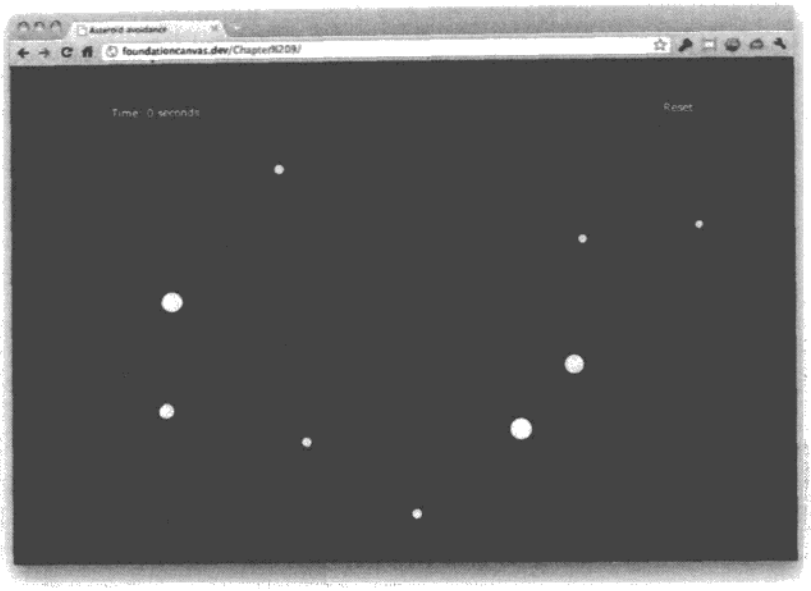


图9-9 绘制玩家火箭并实现动画

以上代码用于确定火焰的持续时间，稍后我们还需要添加更多代码。现在先在animate函数中绘制火箭的代码前面添加以下代码：

```
if (player.moveRight) {
        context.save();
        context.translate(player.x-player.halfWidth, player.y);

        if (player.flameLength == 20) {
                player.flameLength = 15;
        } else {
                player.flameLength = 20;
        };

        context.fillStyle = "orange";
        context.beginPath();
        context.moveTo(0, -5);
        context.lineTo(-player.flameLength, 0);
        context.lineTo(0, 5);
        context.closePath();
        context.fill();

        context.restore();
};
```

条件语句用于确保只有当玩家向右运动时才绘制火焰，因为如果在其他时间也能看到火焰，看上去就不符合常理了。

我们使用画布的translate方法来绘制火焰，因为在后面调用save方法来保存画布的绘图状态时，translate方法可以节约一些时间。现在已经存储了绘图上下文的原始状态，接下来就可以调用translate方法，并把2D绘图上下文的原点移到玩家使用的火箭的左侧（图9-10中用*x*标记了平移后的原点）。

图9-10　绘制火箭的火焰

现在已经移动了画布的原点，接下来的任务就非常简单了。只需要对存储在玩家对象的flameLength属性中的值执行循环（使火箭呈现闪烁效果），并把填充颜色改为橙色，然后从新的起点绘制一个长度与flameLength属性相同的三角形（参见图9-10右侧的图）。最后还需要调用restore方法，将原始绘图状态恢复到画布上。

刷新浏览器看看刚才的劳动成果。当按下向右的箭头键时，火箭上应该出现了一团闪烁的火焰（如图9-11所示）。

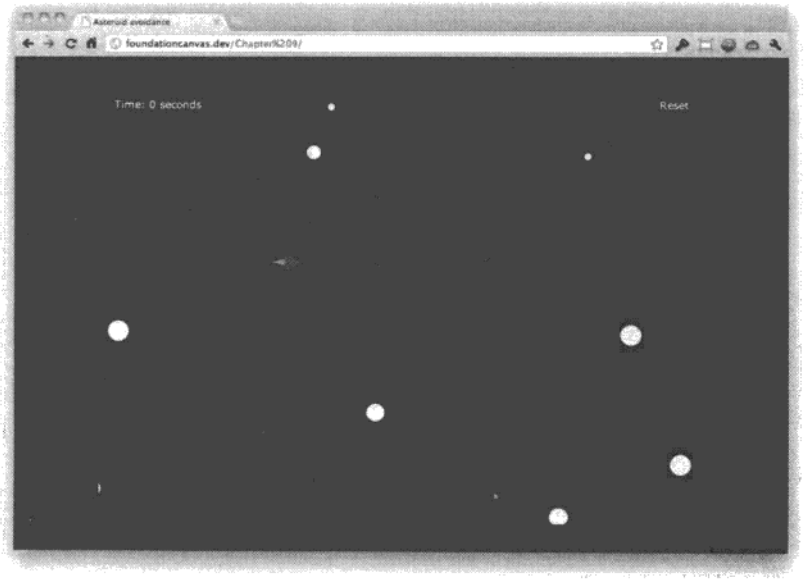


图9-11　绘制并激发火箭的火焰

现在的游戏变得更加有趣了。但你并没有在游戏中使用任何高级物理知（高级物理知不一定总能让游戏变得更加奇妙），所以接下来需要做好准备，我们将使游戏产生一种逼真的横向卷轴效果。

9.6 假造横向卷轴效果

我说的是假造吗？我真这么说了！是的，虽然这个游戏看上去好像是横向卷动的，但实际上你并没有穿越在游戏世界中。相反，你将循环利用所有在屏幕上消失的对象，并让它们重新显示在屏幕的另一侧。这样就会产生一种始终穿越在永无止境的游戏世界中的效果。听起来好像有些奇特，其实它只是一种横向卷动效果而已。

9.6.1 循环利用小行星

让游戏产生一种永无止境的穿越效果其实并不难。实际上非常简单！在animate函数中刚才绘制每颗小行星的代码上面添加以下代码：

```
if (tmpAsteroid.x+tmpAsteroid.radius < 0) {
        tmpAsteroid.radius = 5+(Math.random()*10);
        tmpAsteroid.x = canvasWidth+tmpAsteroid.radius;
        tmpAsteroid.y = Math.floor(Math.random()*canvasHeight);
        tmpAsteroid.vX = -5-(Math.random()*5);
};
```

这点代码就够了。这段代码的作用是检查小行星是否移动到画布的左边界之外，如果是，则重置该小行星，并将它重新移回到画布的右侧。你已经重新利用了该小行星，但它看上去却像是一颗全新的小行星。

9.6.2 添加边界

现在，玩家火箭可能会自由地在游戏中飞越，也可能会停止不动（试图飞越画布的右侧时）。为了解决这个问题，需要在适当的位置设置一些边界。在绘制火箭火焰的代码上面（正好在设置新的玩家位置的代码下面）添加以下代码：

```
if (player.x-player.halfWidth < 20) {
        player.x = 20+player.halfWidth;
} else if (player.x+player.halfWidth > canvasWidth-20) {
        player.x = canvasWidth-20-player.halfWidth;
}

if (player.y-player.halfHeight < 20) {
        player.y = 20+player.halfHeight;
} else if (player.y+player.halfHeight > canvasHeight-20) {
        player.y = canvasHeight-20-player.halfHeight;
};
```

你也许能猜出以上代码的作用。它主要执行一些标准的边界检查，就像上一章游戏中的边界检查一样。这些检查查看玩家是否位于画布边界20像素之内，如果是，则阻止它们沿着该方向进一步移动。我认为在画布的边界处预留20像素的空隙视觉效果更佳，但也可以把这个值再改小一点，以便玩家能够向右移动到画布的边缘处。

9.6.3　让玩家保持连续移动

目前，如果玩家没有按下任何按键，火箭将停止移动。当所有的小行星正在飘荡时，火箭突然停止移动不太符合常理。因此可以在游戏中添加一些额外的运动，当玩家不再向前移动时，可以让它们继续向后移动。

在animate函数中把改变玩家vX属性的代码段更换为以下代码：

```
if (player.moveRight) {
        player.vX = 3;
} else {
        player.vX = -3;
};
```

这段代码只是在条件语句中添加了一段额外代码，即当玩家不需要向右移动时，把玩家的vX属性设为-3。你总结一下就会发现，这与大部分游戏逻辑都是相同的。

在浏览器中运行该游戏，现在的游戏看上去更逼真了！

9.7　添加声音

这也许是游戏中最酷的一部分，因为在保龄球游戏中我们没有涉及该内容。在游戏中添加一些简单的声音非常有趣，游戏也会变得更加引人入胜。你也许觉得在游戏中添加音频是非常困难的，但使用HTML5音频来实现却是一件轻而易举的事！下面我们来看看。

首先需要在游戏的HTML代码中声明所有的HTML5音频元素。直接在index.html文件中的canvas元素下面添加以下代码：

```
<audio id="gameSoundBackground" loop>
        <source src="sounds/background.ogg">
        <source src="sounds/background.mp3">
</audio>
<audio id="gameSoundThrust" loop>
        <source src="sounds/thrust.ogg">
        <source src="sounds/thrust.mp3">
</audio>
<audio id="gameSoundDeath">
        <source src="sounds/death.ogg">
        <source src="sounds/death.mp3">
</audio>
```

如果你掌握了第1章HTML5音频部分的内容，那么应该对以上代码非常熟悉了。如果你还没有掌握该内容，也不用着急，因为它非常简单。这里声明了3个独立的HTML5 audio元素，并且为每个audio元素定义了一个唯一的id属性，后面将用到这些id属性。循环播放的声音还需要定义一个loop属性。

> **注意**　在撰写本书的时候，并非所有的浏览器都支持loop属性。由于它是规范的一部分，因此越来越多的浏览器将会全面支持该属性。如果需要采用一种变通的方案，可以在音频播放结束时添加一个事件监听器，并再次播放。

这3种声音都是背景音乐，火箭开始移动时使用推进器的声音，最后玩家死亡时使用深沉的轰鸣声。为了与大多数浏览器兼容，每种声音都需要两个版本的文件，因此也需要包含两个source元素：一个是mp3版本，另一个是ogg版本。

> **注意** 可以从Friends of ED网站下载本书的所有代码和相关资源。其中包括躲避小行星游戏中使用的声音文件。

在HTML文件中只需要完成这些任务就可以了，接下来我们回到JavaScript文件中，并在JavaScript文件顶部的uiScore变量下面添加以下代码：

```
var soundBackground = $("#gameSoundBackground").get(0);
var soundThrust = $("#gameSoundThrust").get(0);
var soundDeath = $("#gameSoundDeath").get(0);
```

这些变量使用HTML文件中声明的id属性来获取每个audio元素，这与在游戏中获取canvas元素非常相似。接下来将使用这些变量访问HTML5音频API并控制声音。

这些内容无需过多解释，紧接着我们转入keydown事件监听器中，在把playGame设置为true的代码后面添加以下代码：

```
soundBackground.currentTime = 0;
soundBackground.play();
```

现在，你已经在游戏中添加了HTML5音频，并且可以非常方便地控制它。很酷吧？以上代码的作用是访问与背景音乐相关的HTML5 audio元素，并且可以通过HTML5音频API直接控制它。因此，通过更改currentTime属性，可以重置音频文件播放的起始位置；另外，通过调用play方法，可以播放该音频文件。真的很简单！

载入并运行游戏，现在当你开始移动火箭时，应该能听到一些美妙的背景音乐。

> 本游戏中使用的所有音频文件都来自FreeSound网站。背景音乐由用户asdftekno提供[www.freesound.org/samplesViewSingle.php?id=48546]；推进器声音由用户nathanshadow提供[www.freesound.org/samplesViewSingle.php?id=22455]；死亡的声音由用户HerbertBoland提供[www.freesound.org/samplesViewSingle.php?id=33637]。各种创意共享必胜！

下一步是控制推进器的声音（当玩家移动火箭时）。我希望你已经猜到了如何去实现，其实这与实现背景音乐一样简单。

在keydown事件监听器中player对象的moveRight属性设置代码下面添加以下代码：

```
if (soundThrust.paused) {
        soundThrust.currentTime = 0;
        soundThrust.play();
};
```

第一行代码用于检查是否正在播放推进器声音，如果是，则禁止在游戏中再次播放它。这可以防止该声音在播放的过程中被中途切断，因为每秒钟可能会触发多次keydown事件，而你当然也不希望每次触发keydown事件时都再次播放推进器声音（这将是一件糟糕的事情）。

当玩家停止移动时，你也许不希望推进器声音继续播放，为此，在keyup事件监听器中player对象的moveRight属性设置代码下面添加以下代码：

```
soundThrust.pause();
```

就这么简单，音频API太方便了，通过它访问和操纵音频非常简单。

在继续下一步之前（下一节将添加死亡的声音），我们还需要考虑一个问题：如果玩家重置游戏，我们需要如何确保停止播放声音。为此，在init函数的uiReset.click事件处理程序中的startGame调用上面（读起来有点拗口吧）添加以下代码：

```
soundThrust.pause();
soundBackground.pause();
```

当游戏重置时，以上两行代码可以确保停止播放推进器声音和背景音乐。因为死亡的声音不需要进行循环，并且你希望在游戏结束时才播放它，所以暂时不需要考虑死亡的声音。

9.8　结束游戏

现在的游戏已经逐渐成型了。实际上，它就快完成了。接下来唯一要做的就是实现某种计分系统，并通过某种方法来结束游戏。首先解决计分系统问题，稍后介绍如何结束游戏。

9.8.1　计分系统

由于这个游戏的玩法与上一个游戏完全不同，所以需要使用不同的计分标准。在游戏中，鉴于玩家试图生存尽可能长的时间，所以把存活时间作为计分标准显然是一个不错想法。不是吗？很好，很高兴你能认同我的看法。

我们需要通过某种方法来计算游戏从开始到现在所持续的时间。这正好是JavaScript计时器的强项，但在构建计时器之前需要声明一些变量。在JavaScript代码顶部的player变量下面添加以下代码：

```
var score;
var scoreTimeout;
```

这些变量将用于存储分数（已经过去的秒数）和对计时器操作的引用，以便根据需要来开始或停止计时器。

另外，在游戏开始或重置时也需要重新设置分数。为此，在startGame函数顶部的numAsteroids变量下面添加以下代码：

```
score = 0;
```

为了便于管理得分计时器，我们创建一个名为timer的专用函数。在animate函数上面添加以下代码：

```
function timer() {
        if (playGame) {
                scoreTimeout = setTimeout(function() {
                        uiScore.html(++score);
                        timer();
                }, 1000);
        };
};
```

以上代码现在还不会起作用，但它会检查游戏是否开始，如果游戏已经开始，它就把计时器的时间间隔设置为1秒，并把该计时器赋给scoreTimeout变量。在计时器中，score变量的值在增加，同时计分UI也在更新，这与保龄球游戏中的更新方法相似。然后，计时器自身将调用timeout函数来重复整个过程，这意味着游戏结束时计时器才会停止计时。

现在还没有调用timer函数，所以它还不会发挥作用。当游戏开始时，需要调用该函数，因此在keydown事件监听器中的animate函数调用下面添加以下代码：

```
timer();
```

只要玩家开始游戏，以上代码就会触发计时器。在浏览器中查看效果，在游戏界面的左上角可以看到分数在不断增加（如图9-12所示）。

但遗憾的是，这里还存在一个问题——如果你重置游戏，分数有时候会显示为1秒钟。这是因为当你重置游戏时，分数计时器仍然在运行，但它实际在你重置游戏之后才运行（将重置分数由0更改为1）。为了解决这个问题，需要在重置游戏时先清除计时器。幸运的是，JavaScript有特定的函数可以实现该操作。

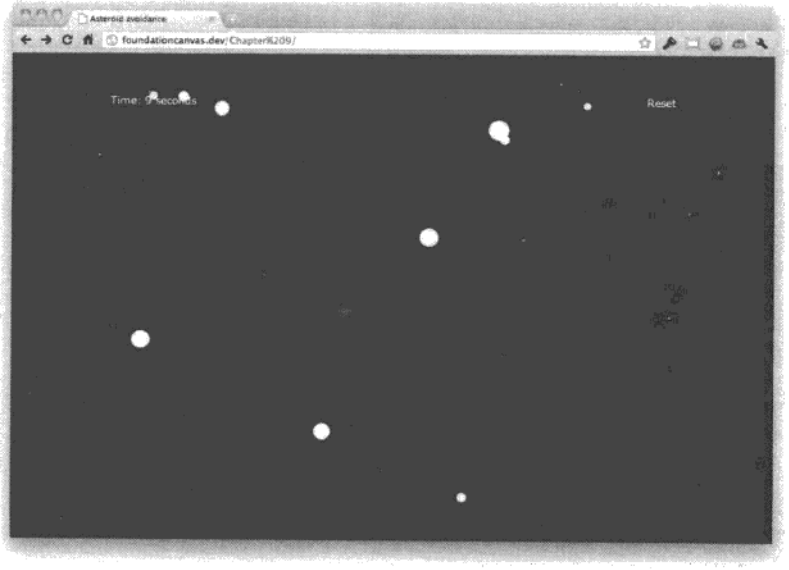


图9-12 在游戏中添加一个计时器

在init函数的uiReset.click事件监听器的startGame调用上面添加以下代码：

```
clearTimeout(scoreTimeout);
```

顾名思义，以上代码的作用显而易见。通过这个独立的函数可以获取scoreTime变量中的分数计时器，并且阻止计时器的运行。再次运行游戏，你可以发现通过这行简单的JavaScript代码已经成功解决了上面遇到的问题。

9.8.2　杀死玩家

如果小行星无法伤害你，那么躲避小行星就没有任何意义了，因此我们需要添加一些功能来杀死玩家（当玩家碰到小行星时）。这里使用的逻辑其实就是上一章已经反复使用过的小行星碰撞检测逻辑：圆周碰撞检测。

在这里发现明显的问题了吗？在火箭是三角形的情况下，你能执行圆周碰撞检测吗？简单地说，你不能执行圆周碰撞检测，或者说至少没那么容易。但这里将忽略一些细节问题，也就是说，把玩家火箭的一小部分区域作为碰撞检测区域（如图9-13所示）。在实际中如果你幸运一些，这种检测有助于躲避小行星。

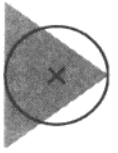


图9-13　在三角形上使用圆周碰撞检测

为了简化代码，我认为这样做是值得的。毕竟，这只是一个供娱乐的小游戏而已，因此不需要追求绝对的真实。

因此，只需在animate函数中绘制每颗小行星的代码上面添加以下代码即可：

```
var dX = player.x - tmpAsteroid.x;
var dY = player.y - tmpAsteroid.y;
var distance = Math.sqrt((dX*dX)+(dY*dY));

if (distance < player.halfWidth+tmpAsteroid.radius) {
        soundThrust.pause();

        soundDeath.currentTime = 0;
        soundDeath.play();

        //游戏结束
        playGame = false;
        clearTimeout(scoreTimeout);
        uiStats.hide();
        uiComplete.show();

        soundBackground.pause();

        $(window).unbind("keyup");
        $(window).unbind("keydown");
};
```

你应该能很快认出以上代码中的距离计算方法：它们与上一章反复使用的距离计算方法完全相同。你将通过它们来计算玩家火箭与当前循环中的小行星之间的像素距离。

下一步是判断火箭是否与小行星发生碰撞，可以通过查看上面计算的像素距离是否小于小行星半径加上火箭碰撞圆周的半径之和。这里使用的火箭碰撞圆周的半径是火箭宽度的一半，你也可以随意改变它。

如果火箭与小行星发生碰撞，就要杀死玩家。杀死玩家并结束游戏的过程非常简单，但我还需要逐行对它进行解释。

前三行代码停止播放推进器声音，并重置和播放死亡的声音。开始播放死亡的声音时，需要把`playGame`设置为`false`来结束整个游戏，并通过前面已经使用过的`clearTimeout`函数来停止计分计时器。

此时，所有的游戏逻辑都已经停止，因此可以隐藏统计界面，并显示游戏结束界面。这与保龄球游戏是完全相同的。

显示游戏结束界面时，需要停止播放背景音乐，并最终释放键盘事件处理程序，从而防止玩家由于无意按下某个按键而启动游戏。

还在听我说话吗？在浏览器上试一试，故意撞上某个小行星看看。你已经完成了第二个游戏（如图9-14所示），你做得很出色！

图9-14 完成的躲避小行星游戏……完成了吗？

9.9　增加游戏难度

好的，其实我们要创建的游戏还没有完成。让我们在游戏中再添加一些功能，即增加游戏的难度，玩家要想存活更长的时间就变得更加困难了。这听上去很有趣。

我们还是直入主题吧，在timer函数中的uiScore.html下面添加以下代码：

```
if (score % 5 == 0) {
        numAsteroids += 5;
};
```

以上代码看上去是否像一个普通的条件语句？其实不是。注意其中的百分比符号。对内行人而言，它是求模运算符；但在外行人看来，它也许只是百分比符号。

求模可以计算一个数是否能被另一个数完全整除，它将返回两个数相除所得的余数。例如，2%2等于0（2被2整除得1，余数为0），4%2等于0（4被2整除得2，余数为0）。另外一种情况是，5%2等于1（5被2整除得2，余数为1）。

你可以通过求模计算来执行周期性的操作，例如，每隔5秒发生一次。因为你可以把模5运算运用于某个数，如果它返回的结果为0，那么该数一定能够被5整除。这听上去也许有些复杂，但如果你稍微思考一下，就会发现它其实很简单。

在本游戏中，我们使用模5运算来确保某个代码段每隔5秒钟执行一次。这个代码段的作用是，每隔5秒钟就向游戏中添加5颗小行星。实际上，这里并没有增加小行星，增加的只是小行星的数目。

添加小行星很简单，在animate函数中绘制玩家火箭的代码下面添加以下代码：

```
while (asteroids.length < numAsteroids) {
        var radius = 5+(Math.random()*10);
        var x = Math.floor(Math.random()*canvasWidth)+canvasWidth+radius;
        var y = Math.floor(Math.random()*canvasHeight);
        var vX = -5-(Math.random()*5);

        asteroids.push(new Asteroid(x, y, radius, vX));
};
```

以上代码检查每个循环中的小行星数目，如果数目没有达到要求，它将继续向游戏中添加新的小行星，直到小行星的数目达到要求为止。这里使用的是while循环，while循环是另一种JavaScript循环，它将一直运行，直到括号中的条件变为false为止。在本游戏中，while循环将一直运行，直到小行星数组的长度大于或等于numAsteroids变量为止。

再次在浏览器中启动游戏，你会发现，当存活的时间越来越长时，游戏中的小行星就会越来越多（如图9-15所示）。现在，你已经真正完成了游戏。我保证！

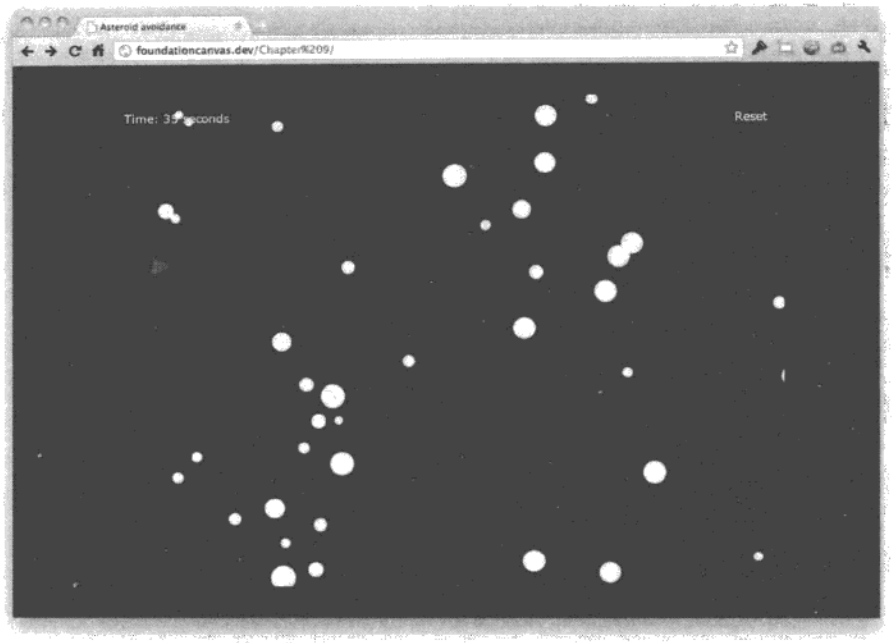


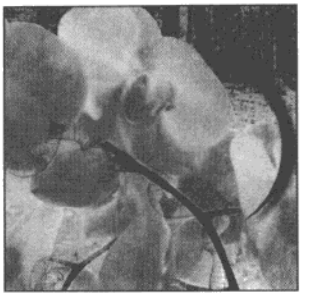
图9-15 真正完成的躲避小行星游戏

9.10 小结

这个游戏也许比上一章中的游戏简单一些，但它却教给你一些在JavaScript和HTML5中创建游戏的宝贵技巧。其中最有价值的知识是学习如何检测键盘输入，以及如何在游戏中使用和操纵HTML5音频。

现在你已经完成了两个游戏，这两个游戏的玩法是截然不同的。我希望我已经向你演示清楚了如何构建一些简单代码并在其中添加一些逻辑，就能创建非常有趣和迷人的游戏。另外，通过躲避小行星游戏，我希望你还能认识到不一定非要使用复杂的物理学知识才能创建有趣的游戏。

本章就到这里。下一章将展望Canvas的未来。



第 10 章

未来的Canvas

在最后一章中，我将带你一起展望动画和游戏开发技术的未来。另外，还将重点讨论与Canvas及其应用前景有关的问题，并将Canvas技术与Flash及其他技术进行比较。

10.1 Canvas 与 SVG

第1章曾提到过，大家通常认为SVG是HTML5的组成部分，其实SVG是一种完全不同的技术。了解SVG技术是很重要的，因为它所提供的功能与Canvas非常相似。另外，我们还需要重点区分SVG与Canvas的不同之处，因为这些差异将决定你会使用哪种技术。

SVG是可以在浏览器中使用的另一种二维绘图平台，它不需要使用像Adobe Flash这样的插件。正如第1章所提到的，SVG代表Scalable Vector Graphics（可缩放矢量图），它的名称从某种程度上说明了它的主要特性——矢量图。相比较而言，Canvas使用的是位图（bitmap graphics），稍后我将详细解释它们的区别。

SVG是通过DOM元素来创建图形的，图形的每个部分都是在单独的DOM元素中绘制而成的。例如，以下代码是SVG中矩形的定义：

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1" height="100" width="200">
        <rect x="50" y="50" width="200" height="100" fill="#000000"/>
</svg>
```

你是否注意到了形状的所有属性是如何在DOM元素中进行定义的？这是SVG与Canvas的最显著区别，Canvas中的所有对象都是在单个元素中绘制的，它使用的是JavaScript代码，而不是DOM元素。

这种在DOM中绘制形状的方法有很多优点，最有价值的一点是绘制完形状以后还可以对它进行访问和编辑。而这种操作无法在Canvas中实现，因为在Canvas中需要先清除画布上的形状，然后再重新绘制。在SVG中不需要擦除任何对象，只需对DOM中的每种形状的属性进行编辑即可，并且形状会自动进行更新。

比较这两种技术的好办法是把Canvas想象成微软的画图程序。在微软的画图程序中，如果需要编辑或更改任何对象，必须先全部擦除它，然后重新开始绘制。相比较而言，SVG更像Adobe

Illustrator，其中的对象都是可选择和可编辑的，并且可以随意调整大小。

> **注意** 当前的主流浏览器通常不支持内嵌的SVG。所幸，现在所有的主流浏览器开发商都在积极致力于提供这项支持，我们将拭目以待。更多详细信息可以参考“When can I use”站点：http://caniuse.com/#feat=svg-html5。

10.1.1 可访问性

SVG的另一个显著优势是它的可访问性。在Canvas中，你绘制的所有对象都是使用像素创建的，因此Canvas完全不知道它上面实际画了什么。例如，你也许用`arc`方法绘制了一个圆，虽然用肉眼看上去是一个圆，但Canvas只知道你刚才更改了特定区域的像素颜色。但在SVG中，可以通过遍历DOM元素获取每种形状及其数据。因此，可以在SVG中实现撤销操作，还可以获取图形的形状信息及其属性（如大小、位置等）。

10.1.2 位图与矢量图

正如我们之前所讨论的，SVG是一种矢量图，而Canvas是一种位图。但这实际又意味着什么呢？简而言之，它们有很多区别！

位图（又名光栅图）的像素是按1:1格式存储的（如图10-1所示）。这意味着如果你在位图中绘制一个像素的方块，那么它只能保存一个像素。如果你试图改变该像素（如改变大小），就会产生像素化（pixellated）和失真现象。因为位图无法识别自己所表示的形状类型。如果把1个像素的方块放大成5个像素的方块，那么一个像素的方块就会拉伸为原来的5倍，因此新的5个像素的方块看上去就变得模糊了。我确信你以前一定注意过这种现象，或许当你重新保存JPEG图像时就遇到过类似情况。

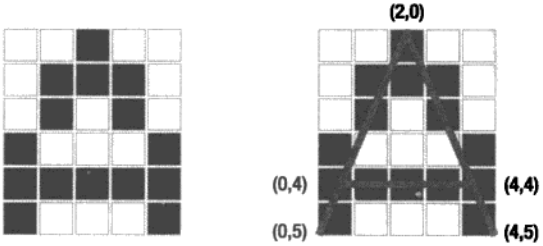


图10-1 位图与矢量图

相反，矢量图是以坐标的形式存储的，坐标描述了你希望绘制的形状（如图10-1所示）。这种存储方法的优点是，在实际需要在屏幕上展示矢量图之前，你不必考虑像素方面的问题。这意味着可以根据需要随意调整矢量图的大小，但以像素形式输出的图形看上去仍然很清晰。其中的

实现过程非常简单：虽然形状的坐标调整了大小，但每个坐标之间的比例关系仍然保持不变，因此在显示图形时，新的坐标将转化为像素。

Canvas和SVG有各自的优点和缺点。例如，如果你需要在将来插入或删除绘制的形状，那么SVG在可访问性和可视化数据方面将表现得更出色一些(因为它使用了DOM元素)。而Canvas在像素级的细节绘制方面则表现得更快和更完善一些（例如，处理图像和视频）。在决定使用哪种技术之前，你应该对它们进行权衡。这应该不会太难，毕竟这还算不上是一个生死攸关的决策。

> **注意**　如果你要尝试使用SVG，那么我强烈推荐你使用Raphael JavaScript库，其网址为：http://raphaeljs.com。采用这种方式尝试SVG，你就不必纠缠于设置一些环境和创建单个DOM元素了。

10.2　Canvas 与 Flash

现在你可能经常会听到一些评论，有人说Canvas是Flash杀手，有人说Flash比Canvas要好。实际上，这些说法都是片面的。因为这两种技术之间其实并没有什么冲突，它们完全是为不同的目标而服务的！

Flash已经存在很多年了，现在它正逐步用于解决Web动画和多媒体问题。过去，浏览器（或嵌入式视频和音频）中没有提供绘制二维图形的原生方法。通过在多数浏览器和平台上提供一种插件，Flash解决了以上问题，这可以说是雪中送炭！

从那以后，Flash开始变得越来越强大。实际上，现在Flash已经发展成为一种稳定的平台，通过它可以创建快速动画图形、对视频进行编码，以及创建复杂的游戏。另外，它还可以实现其他更多特性，例如网络通信以及与外围设备进行集成等。显然，这里我省略了很多特性，但我想你能明白这一点。

为了解决浏览器中需要一种原生方法来绘制二维图形的问题，我们创建了Canvas，并且这种方法不需要在浏览器中安装插件。它使用的是现有的一些技术，例如HTML和JavaScript。简而言之，Canvas为在浏览器中绘制图形和制作动画提供了一种可选方案。仅此而已！

我在这里想表达的观点是，创建Canvas和创建Flash的出发点是不同的。它们解决了不同的问题，因此它们之间不存在冲突。我们没必要对它们进行比较，只需要在工作中选择合适的工具就可以了。

10.2.1　JavaScript 开发人员可以借鉴 Flash

我的朋友Seb Lee-Delisle帮助我全面认识了Flash和Canvas。他说Canvas正在经历多年前Flash所经历的发展阶段（的确如此），因此JavaScript开发人员可以借鉴Flash开发人员的经验。

他的观点是完全正确的。Canvas在当前还不是主流技术，因此使用它开发产品的人还不多。Canvas的这种状况表明，JavaScript开发人员目前仍然处在试验阶段，他们正在尝试学习Canvas的实际功能。这些尝试和学习的过程与以前的开发人员推广Flash是非常相似的。

Seb的看法很简单：不要总是试图区分这些技术，要综合运用并分享你的知识。如果你能做到这一点，那么我敢断言，Canvas的发展前景将会变得更加美好。

10.2.2 Canvas 没有像 Flash 那样用户友好的编辑器

支持Flash的观点认为，由于没有应用程序为Canvas提供用户友好的编辑界面（像Flash IDE那样），所以Canvas很难使用。幸运的是，现在的情况已经改变了。

现在有很多可供Canvas开发人员选择的工具，通过这些工具，开发动画和游戏的过程将会变得更加容易。可选工具包括Radi（http://radiapp.com，这是一种创建HTML5动画的专用应用程序，如图10-2所示）和ai2canvas（一个Adobe Illustrator插件）。

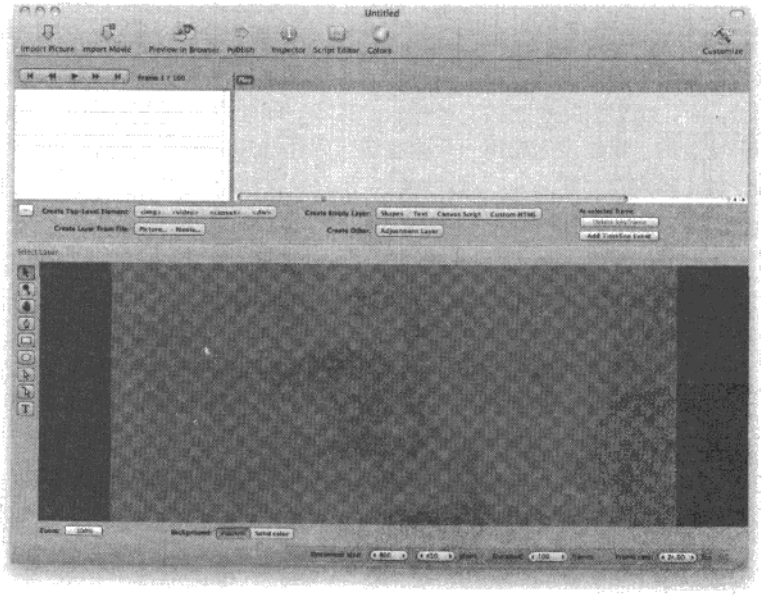


图10-2 Radi HTML5动画应用程序

坦率地说，即使Flash本身提供一种选项直接把结果导出到Canvas中，我也不会觉得奇怪。当然Adobe肯定不会采纳这种思想。

总之，如果我们理性思考一下就会发现，其实Flash和Canvas之间并不存在分歧。因此下次如果有人问你为什么应该使用Canvas而不使用Flash，你可以告诉他们不应该这么想。当然，这种回答一定会让他们觉得很奇怪！

10.3　Canvas与性能

Canvas最明显的局限性之一就是性能问题。这可不是撒谎，目前它是Canvas中的突出问题。因为在多数浏览器中需要使用CPU来呈现Canvas，这意味着处理器资源将会很快被耗尽，直至Canvas停止运行。当然，只有当你试图创建一个非常复杂的动画或绘制大量对象时才会发生这种情况，但这仍然是一个亟待解决的问题。

所幸，有一种称作硬件加速（hardware acceleration）的技术可以解决此问题。更令人兴奋的消息是，你现在就可以使用这种解决方案来尝试解决此问题。令人惊讶的是，IE9居然成为了支持Canvas硬件加速的先行者，Windows中的其他浏览器也纷纷紧随其后。遗憾的是，目前Mac的浏览器还不支持这种硬件加速技术，但我认为这种状况很快就会改变。

> **注意**　虽然IE9在微软系列浏览器中的性能是最完善的，但Windows XP却不支持它。这意味着那些使用Windows旧版本的用户将无法体验这种新兴的Web技术。但是，这些用户是否应该思考对XP进行升级呢，目前这个问题还存在争议。

硬件加速是把Canvas的绘制过程委派给图形卡上的图形处理器（GPU）进行处理。这样，CPU就被释放出来了，而让GPU去处理一些它最擅长的任务，即处理与图形相关的任务。硬件加速产生的效果是非常显著的。我建议你先用一个没有进行硬件加速的浏览器播放Canvas动画，然后再用已经进行硬件加速的浏览器试一下。通过比较，你会发现使用硬件加速的浏览器播放的动画看上去会更流畅。

测试性能

当然，硬件加速也不是万能的，它不能解决所有性能问题。我们还可以通过一些非常简单的方法来解决一些最常见的性能问题。例如，碰撞检测例程中过多的循环就会产生各种性能问题。另一个例子是，如果你通过一个任意数来代替过度复杂的数学计算就能获取同样的效果（例如，在第7章中使用的摩擦力），那么也能解决性能问题。

在这里，经验丰富的Flash开发人员的技巧就可以发挥作用了。这些人亲身经历了压缩每一个动画循环来提高系统性能的艰难岁月。从他们那里你还能学到一些折中技巧，一些小小的性能技巧有时却能产生非常显著的效果。

如果你想亲自测试代码的性能，我建议你查看jsPerf网站（http://jsperf.com）。通过使用该测试工具，可以对实现各种相同功能的方法进行测试，最终能确定哪种方法更快一些。虽然这种工具并不是特别简单，但它却为你提供了一种有效的途径，及时帮助你确定是否可以更进一步简化代码。

10.4　Canvas游戏和动画库

到目前为止，通过本书的学习你应该能够利用所有基础知识从零开始创建自己的游戏和动画。但是如果你不希望从零创建游戏该怎么办呢？如果你不希望每天都纠缠于动画循环这样的任

务该怎么办呢？这就是产生HTML5新动画和游戏库的背景环境。

通过这些游戏和动画库（例如Impact，网址为：http://impactjs.com，如图10-3所示），你可以把游戏和动画组合在一起，而无需考虑细节问题。从本质上说，这些库的工作原理与jQuery是完全相同的：它们消除了复杂性，使你能够更加专注于需要解决的问题，而无需考虑如何去实现它。

图10-3 Impact游戏库

在撰写本书的时候，其实已经有很多游戏和动画库可供选择（https://github.com/bebraw/jswiki/wiki/Game-Engines）。除了Impact之外，另一个最著名的库是Easel（其网址为：http://easeljs.com/，如图10-4所示）。Easel库其实并不是专业的游戏库，而是一种采用Canvas元素创建动画时使用的通用辅助库。

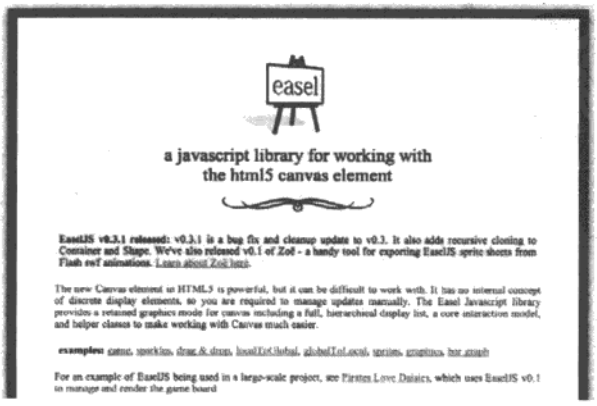


图10-4 Easel动画库

如果你是一位Flash开发人员，那么很快就会发现，Easel使用的很多概念都来自ActionScript，如显示对象和场景（stage）等。这并不是偶然现象。Grant Skinner是一位非常出色的Flash技术开发人员，他认为Easel“总体上是基于Flash的显示列表（display list），因此JS和AS3开发人员应该很容易掌握它”。

这些库使用起来非常方便，但它们不可能完全满足所有需求。因此，我建议不要急于依赖这些库来完成所有任务。应该首先学习动画和游戏开发的核心概念，为处理问题打好坚实基础，这样才能尝试解决一些库可能无法支持的任务。

10.5　三维图形

到目前为止，我们在Canvas中创建的对象都是二维图形。如果你觉得这有些单调，也许会对三维图形感兴趣。Canvas中的三维图形是建立在一种称作WebGL的技术上的。虽然并不是所有的浏览器都支持WebGL（在撰写本书时，仅有Chrome、Webkit开发版本以及Firefox支持WebGL），但它的表现仍然非常出色。

如果你希望尝试这种技术，我强烈建议你访问由Doob创建的three.js库（https://github.com/mrdoob/three.js）。three.js库能够让你用尽可能少的代码来创建一些绚丽的3D图形（如图10-5所示），这大大降低了用户学习WebGL的门槛（众所周知，这种技术对用户的专业要求非常高）。

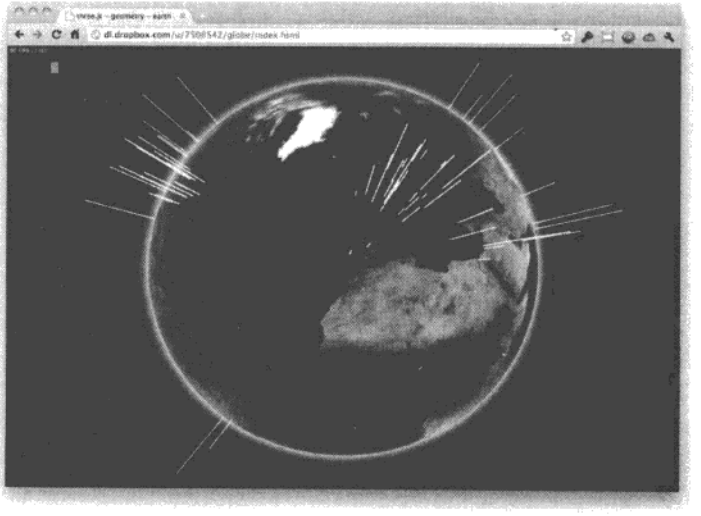


图10-5　Doob使用three.js创建的3D地球

如果我说在Canvas中创建3D图形并不复杂，那我一定是在撒谎，但我想这丝毫不会影响大家的兴致。即便你不打算使用这种技术，我还是要稍微介绍一下。

10.6 与外围设备交互

与连接的外围设备（例如网络摄像头和麦克风）进行交互是浏览器还没有涉及的一个主要领域。这是Flash特别擅长的，这也让我们看到了不使用插件方式就能够与外围设备交互的希望。

实际上有一群人的专职工作就是研究浏览器与连接设备之间进行通信的规范。这个组织就是“设备与API策略工作组”（Device and APIs Policy Working Group），或简称为DAP。我认为他们正在研究的Media Capture API是最有趣的。

通过Media Capture API，JavaScript开发人员可以使用网络摄像头来访问图像，然后通过其他方式操纵或使用这些图像。例如，可以构建人脸检测（face detection）程序，一组简单的过滤器程序（类似于Mac的Photo Booth），甚至是通过类似于WebSocket规范的标准来创建视频流。总之，这种可能是永无止境的。

你一定要访问DAP工作组的官方网站（网址为：http://www.w3.org/2009/dap/），了解他们的最新工作动态将是一件非常有趣的事情。

Rainbow 项目

迄今为止，DAP还没有展示太多的成果。但在撰写本书时，Mozilla浏览器已经开始进行试验了，尝试通过JavaScript来访问网络摄像头和麦克风。这个试验称作Rainbow（https://mozillalabs.com/rainbow/，如图10-6所示），这也让我们看到了该技术的发展前景。目前，你可以访问网络摄像头和麦克风，并把内容录制为一种开放格式的视频文件。可以通过JavaScript来访问得到的文件，因此可以操纵文件，甚至可以把文件上传到服务器上。真的很棒！

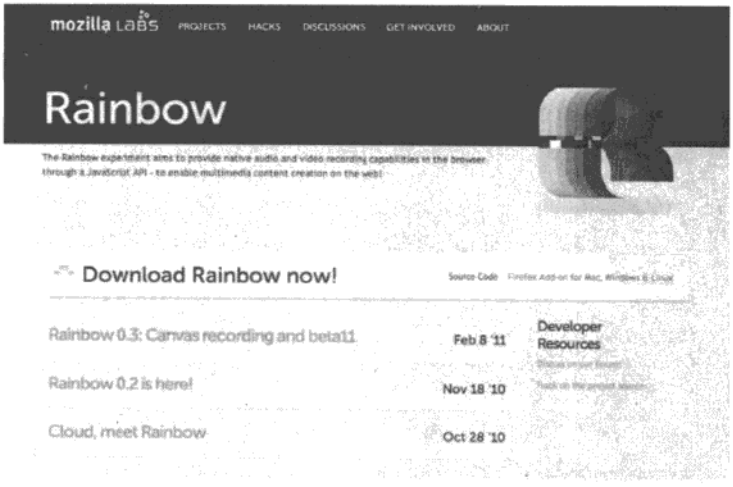


图10-6 Rainbow项目网站

Rainbow最出色、最有趣的特性之一是它能够录制Canvas的内容。这样，你就可以录制动画或游戏片段，并将它保存为视频文件。它从根本上改变了只能使用应用程序来录制整个桌面内容的现状。

10.7　用WebSocket技术构建多人游戏

我们在本书中创建的两个游戏都是单人游戏。但是，通过添加一些简单的多人逻辑，我们就可以轻松地创建多人游戏，例如，在最后一个游戏中使用一组不同的按键让第二个玩家移动另一只火箭。但是，这仍然要求两个玩家在同一台计算机上操作游戏，并且使用同一个键盘。但通过WebSocket技术，你就可以在网络上（例如因特网）创建真正意义上的多人游戏。

其实WebSocket技术与游戏并没有真正的联系，正如第1章结尾所述，它们只是一种实时进行双向通信的方法。WebSocket技术本身并没有什么迷人之处，但是当你把它与其他技术（例如Canvas）组合起来，就会感受到它的无穷魅力。

通过WebSocket，你可以双向发送数据，这就为创建游戏拓展了空间。例如，可以首先通过WebSocket向服务器发送玩家的位置坐标（x, y），然后再使用WebSocket向连接到游戏中的所有其他玩家发送该位置坐标。下面展示了一个最基本的多人游戏示例（http://rawkets.com/），Rawkets是我创建的第一个多人游戏，在游戏中我组合了Canvas和WebSocket两种技术（如图10-7所示）。

图10-7　组合Canvas和WebSocket创建的多人游戏Rawkets

与本章的其他内容相比，WebSocket也许有些难度，但这不应该成为不学它的理由。我建议你积极探索WebSocket的强大功能，并尝试创建自己的多人游戏。这些内容也许可以用一整本书的篇幅来描述，但我希望通过本书的介绍，你能对WebSocket产生浓厚的兴趣，并在将来积极进行尝试。

10.8 灵感

在结束本书之前，我还想向你展示一些出色的HTML5游戏，这些游戏都是在本书撰写的过程中别人编写的，我觉得它们能给你带来一些灵感。希望你喜欢它们！

10.8.1 Sketch Out 游戏

这是一个神奇的体验式游戏，它以太空为背景，你置身于一场与其他行星进行战斗的奇妙场景中。你的任务是在自己所在的行星周围绘制防御线来保护自己，避免遭受其他行星的撞击。游戏的音乐和场景都非常出色（如图10-8所示）。关于该游戏的详细信息请访问以下网址：http://sketch-out.appspot.com/。

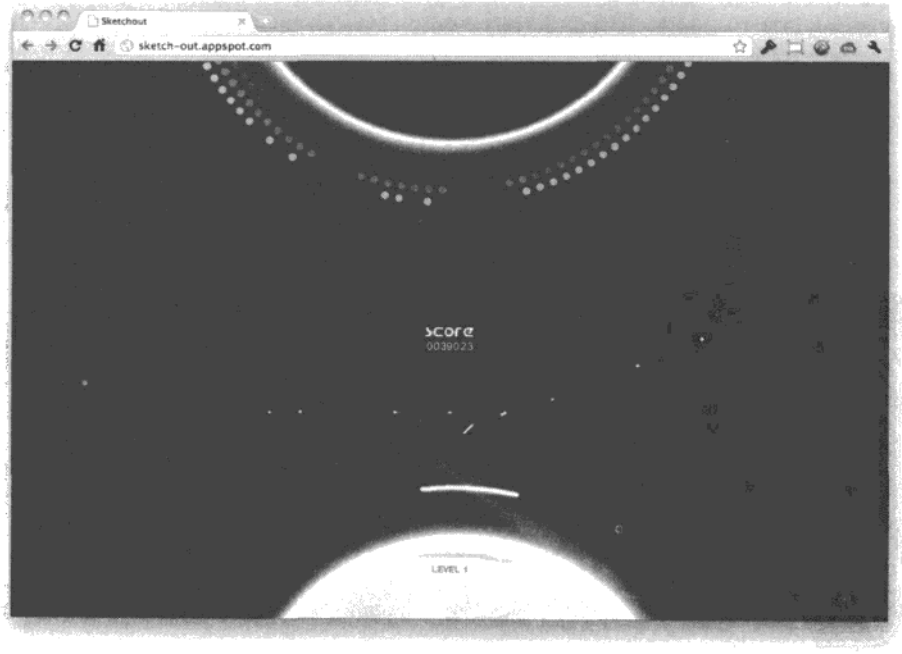


图10-8 由Fi创建的Sketch Out游戏

10.8.2　Z-Type 游戏

这是一个非常有趣的游戏，它建立在单词的基础上。的确如此，整个游戏都是围绕输入屏幕上显示的单词而展开的，每一个正确输入的字母都会击中你的对手。太棒了！关于该游戏的详细信息请访问以下网址：http://www.phoboslab.org/ztype/（如图10-9所示）。

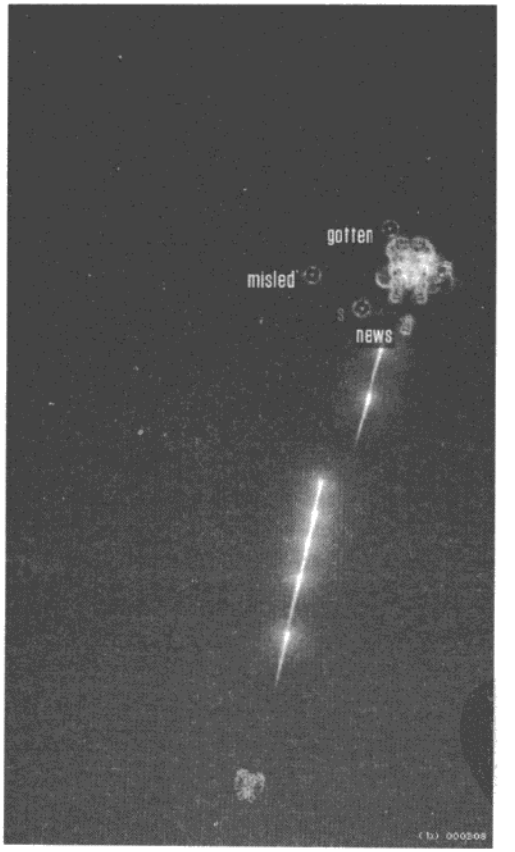


图10-9　由Dominic Szablewski创建的Z-Type游戏（图片由Mozilla实验室提供）

10.8.3　Sinuous 游戏

这是一个非常精彩的游戏，它与上一章创建的躲避小行星游戏在功能上非常相似。实际上Sinuous游戏更高级一些，要想创建它，你需要花费一些时间和精力，此外必须掌握该游戏使用的技术。关于该游戏的详细信息请访问以下网址：http://sinuousgame.com/（如图10-10所示）。

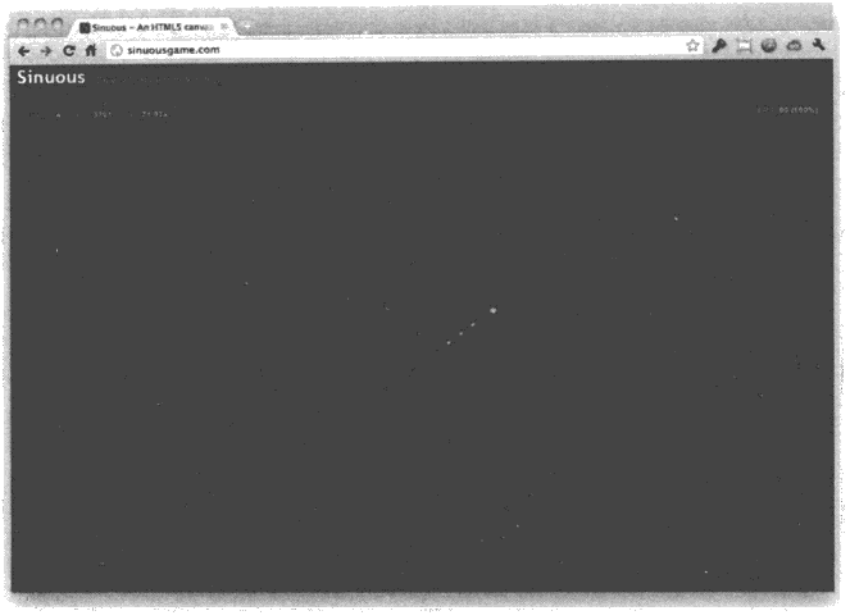


图10-10 由Hakim El Hattab创建的Sinuous游戏

10.9 小结和结束语

本章探讨了一些与Canvas相关的领域，并展望了Canvas在动画和游戏开发方面的前景。另外，我们还把Canvas与一些技术做了比较，希望通过这些比较，你能在项目中更加合理地选择合适的工具。

某些主题也许在本书前面有所提及，但我觉得把这些主题组合起来更能体现Canvas的美好未来。总而言之，Canvas的前景是非常光明的，在开发过程中你应该能体会到这种令人兴奋的感受。那么，就从现在开始学习使用Canvas吧，也许在不久的将来你就能开发出非常优秀的作品！

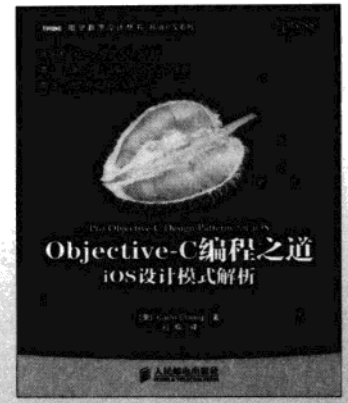
但事实上，Canvas的发展才刚刚开始。像硬件加速和WebGL等技术仍然需要跨平台的支持，因此它们将直接影响Canvas的应用前景，同时也将决定Canvas的用处。

总而言之，Canvas是一个非常优秀的开发工具，尤其是通过组合使用WebSocket和其他HTML5技术（例如视频和音频）之后，它的功能将更加强大。虽然Canvas的发展刚刚开始，但我希望本书能够激发你尝试使用Canvas的兴趣。

我期望着听到你成功开发出作品的消息。还等什么呢？赶快用Canvas开发吧！

TURING 图灵教育

图灵最新重点图书



书　名：Objective-C 编程之道：iOS 设计模式解析
书　号：978-7-115-26586-9
作　者：[美] Carlo Chung
译　者：刘威
定　价：59.00 元

- 解析 iOS 设计模式的开山之作
- 优化 Objective-C 编程实践的必修宝典
- 由此迈入移动开发高手行列

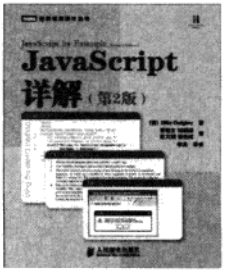
本书是基于 iOS 的软件开发指南。书中应用 GoF 的经典设计模式，介绍了如何在代码中应用创建型模式、结构型模式和行为模式，如何设计模式以巩固应用程序，并通过设计模式实例介绍 MVC 在 Cocoa Touch 框架中的工作方式。

本书适用于那些已经具备 Objective-C 基础、想利用设计模式来提高软件开发效率的中高级 iOS 开发人员。

布道之道：引领团队拥抱技术创新
书号：978-7-115-26727-6
定价：29.00 元

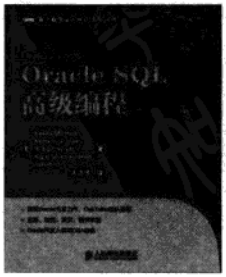
HTML5 和 CSS3 实例教程
书号：978-7-115-26724-5
定价：39.00 元

精通 Android 3
书号：978-7-115-26602-6
定价：128.00 元



JavaScript 详解（第 2 版）
书号：978-7-115-26291-2
定价：95.00 元

JavaScript 修炼之道
书号：978-7-115-26556-2
定价：29.00 元



Oracle SQL 高级编程
书号：978-7-115-26614-9
定价：89.00 元