*Supercharged JavaScript Graphics*

*with HTML5 Canvas and jQuery*

# JavaScript 高效图形编程

[英] *Raffaele Cecco* 著

徐鹏飞 译

O'REILLY®

人民邮电出版社

POSTS & TELECOM PRESS

# JavaScript高效图形编程

随着HTML5的出现和Web浏览器对JavaScript支持的日益改进，JavaScript已经成为创建高性能Web图形的首选工具。本书介绍了如何使用JavaScript、jQuery、DHTML和HTML5的Canvas元素来为台式机和移动设备创建富Web应用程序。

通过本书的示例，有经验的Web开发人员可以学习创建游戏、DHTML特效、业务仪表板和其他应用的方法。除了丰富的示例外，本书的另一个特点是通俗易懂、循序渐进，每个主题都为下一个要讲解的主题提供了基础。

- JavaScript性能优化；
- 结合jQuery和传统的DHTML来创建图形动画；
- 学习使用jQuery UI和Ext JS库的高级UI技术；
- 用碰撞检测、对象处理、JavaScript滚动技术构建游戏；
- 掌握HTML5 Canvas，如绘制、填充、位图和动画等；
- 用jQuery Mobile和PhoneGap创建手机应用；
- 用Google的数据可视化工具创建交互式仪表板。

Raffaele Cecco是欧洲视频游戏产业的资深软件开发人员。他曾在伦敦King of the Jungle软件工作室任技术总监，其客户包括美国孩之宝玩具公司（Hasbro）和英国维珍（Virgin）集团。Raffaele现在是一名Web开发人员，并喜欢在他的站点（www.professorcloud.com）上进行各种尝试。

"Raffaele Cecco是绝对的行家里手。通过这本书，我深入理解了广受欢迎的Web游戏和交互应用是如何创建的。"

——Shelley Powers，JavaScript Cookbook和Painting the Web的作者

O'REILLY®
oreilly.com.cn

O'Reilly Media, Inc.授权人民邮电出版社出版

此简体中文版仅限于中国大陆（不包含中国香港、澳门特别行政区和中国台湾地区）销售发行
This Authorized Edition for sale only in the territory of People's Republic of China (excluding Hong Kong, Macao and Taiwan)

分类建议：计算机/程序设计/网络编程

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-27881-4

ISBN 978-7-115-27881-4

定价：45.00 元

O'REILLY®

# JavaScript 高效图形编程

[英] Raffaele Cecco 著

徐鹏飞 译

人 民 邮 电 出 版 社

北 京

**图书在版编目（CIP）数据**

JavaScript高效图形编程 / （英）切克（Cecco, R.）著 ; 徐鹏飞译. -- 北京 : 人民邮电出版社, 2012.5
ISBN 978-7-115-27881-4

Ⅰ. ①J… Ⅱ. ①切… ②徐… Ⅲ. ①JAVA语言－程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2012)第051662号

## 版权声明



**JavaScript 高效图形编程**

◆ 著 [英] Raffaele Cecco
译 徐鹏飞
责任编辑 傅道坤

◆ 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：787×1000 1/16
印张：15.5
字数：317 千字 2012 年 5 月第 1 版
印数：1－3 000 册 2012 年 5 月北京第 1 次印刷

著作权合同登记号 图字：01-2012-1199 号
ISBN 978-7-115-27881-4

定价：45.00 元

**读者服务热线：(010)67132705 印装质量热线：(010)67129223**
**反盗版热线：(010)67171154**

# 内 容 提 要

本书是一本具有很强实操性的 JavaScript 图书，全书共分 10 章，涵盖的主要内容有：JavaScript 的面向对象机制、JavaScript 性能优化、jQuery 和 ExtJS 库、高级 UI 设计、Web 游戏开发、面向移动设备的开发、图形编程知识等。

本书适合有一定 Web 开发经验和 JavaScript 基础的开发人员学习。

# O'Reilly Media，Inc.介绍

O'Reilly Media 通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自 1978 年开始，O'Reilly 一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O'Reilly 的发展充满了对创新的倡导、创造和发扬光大。

O'Reilly 为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了 Make 杂志，从而成为 DIY 革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O'Reilly 的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O'Reilly 现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版，在线服务或者面授课程，每一项 O'Reilly 的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

## 业界评论

“O'Reilly Radar 博客有口皆碑。”

——Wired

“O'Reilly 凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——Business 2.0

“O'Reilly Conference 是聚集关键思想领袖的绝对典范。”

——CRN

“一本 O'Reilly 的书就代表一个有用、有前途、需要学习的主题。”

——Irish Times

“Tim 是位特立独行的商人，他不光放眼于最长远、最广阔的视野并且切实地按照 Yogi Berra 的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去 Tim 似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——Linux Journal

# 作者简介

**Raffaele Cecco** 是欧洲视频游戏产业的资深程序员。他曾在伦敦 King of the Jungle 软件工作室任技术总监，其客户包括美国孩之宝玩具（Hasbro）公司和英国维珍（Virgin）集团。他使用过各种 Web 开发技术，并开发过零售电子商务系统。

# 封面介绍

本书封面的动物是一只蛮羊，或称鬃羊、巴贝里绵羊。

蛮羊是一类相对较大的羊科动物。它源于北非，现在还分布在西班牙东南部、美国西南部以及墨西哥部分区域。这些居住在沙漠之中的食草动物也被称为鬣羊。

蛮羊栖息在炎热荒凉的岩石和沙土地带，身体所需水分多来自于吃的各种植物。它们硕大的弯弯的角储有丰富的血液供应，使它可以在炎热干燥的沙漠中降温。除了特殊的角外，蛮羊从它的喉部到前胸再到前肢都长着柔软的长毛，其毛色为黄褐色。

蛮羊和其他沙漠中的动物一样，在白天寻找遮阳处，在较凉爽的黎明和傍晚最为活跃。它们擅长攀登和跳跃，可以在极端陡峭的斜坡上攀登跳跃，这使得它们极难被捕猎。由于它们的生活区域内少有高大植物藏身，因此它们主要依靠其表皮颜色骗过捕食者。在北非，它们曾经的捕食者有狞猫、狮子、北非豹，而如今人类才是它们的主要威胁。

尽管不易被捕猎，但在非洲，人类的捕猎还是让蛮羊的数量急剧减少。20 世纪 50 年代，蛮羊被引入美国西南部，曾使其数量有短暂回升。蛮羊的数量目前在 5000 到 10000 之间，而且由于捕猎和丧失栖息地，预计在未来 15 年内数量将下降 10%。因此，蛮羊已被列入国际自然和自然资源保护联合会的濒危物种名单。

封面图片来自于 Riverside Natural History。

# 前言

作为资深的视频游戏开发人员，我已经习惯于和高性能的编程语言和硬件打交道，因此刚开始我并没有对 JavaScript 进行图形编程有太高的期望。不过后来发现，实际上 JavaScript 是一个优秀和高效的编程语言，而且随着更好的浏览器支持、本身的性能提升，以及新的工具库加入，JavaScript 还在不断变好。JavaScript 结合了 HTML5 Canvas 等特性，给 Web 开发人员提供了真正可以不用 Adobe Flash 等插件的方案。而 WebGL 等特性则为使用 JavaScript 和浏览器进行图形编程描绘了非常美好的未来。

这本书的目标读者需要具备一定的 JavaScript 知识，并且想要学习真正的 Web 图形编程，而不仅仅依赖于 jQuery 这样的库做一些动画特效。本书中涵盖了下面这些内容：

- 如何重用和优化代码，包括继承技术和性能优化经验；
- 用普通的 DOM 操作（DHTML）来构建图形化应用；
- 使用更高级的画布元素；
- 创建视频游戏；
- 创建图形和动画所需的数学；
- 使用谷歌可视化 API 和画图工具来呈现你的数据；
- 如何有效使用 jQuery 并开发面向图形的 jQuery 插件；
- 使用 jQuery Mobile 创建适合移动设备的图形 Web 应用；
- 使用 PhoneGap 将你的 Web 应用转换为本地 Android 应用。

本书将带你了解不同的图形编程技术，你可以进一步探索自己感兴趣的领域。

多做尝试，你会获得很多乐趣！

## 目标读者

打算阅读本书的读者应该对网站和 Web 应用开发，特别是 JavaScript，具有一定知识和使用经验。

为了方便开发和表达，本书的许多示例代码都使用了 jQuery。一般来说，本书中用到的所有外部库及其文件都可以从谷歌等可靠的内容分发网络获取。

另外，本书用到了一些基础的数学知识，包括向量和三角函数。

## 本书组织结构

本书节奏较快，读者从第 1 章中就可以看到第一个图形编程的示例。

剩余章节涉及多个图形相关的技术，这些技术可以给你的 Web 应用增加视觉冲击力和交互性。

讨论交互式图形的书无法避开视频游戏。本书中将开发一个完整的视频游戏应用，并讨论相关的子图和滚动等技术。

本书每章的内容可以总结如下。

*第 1 章，代码重用和优化*

本章讨论 JavaScript 面向对象编程技术，以及图形应用中涉及的代码优化（包括 jQuery 优化）。本章甚至将介绍如何使用鲜为人知的 JavaScript 位操作符进行性能优化。

*第 2 章，DHTML 基础*

本章展示了如何使用普通的 DOM 操作（DHTML）创建图形应用。我们将在本章开发一个适用于游戏和其他场景的子图系统，并将其以 jQuery 插件的形式封装。

*第 3 章，滚动*

本章首先讨论了 CSS 滚动技术，包括视差特效。然后本章将介绍基于 JavaScript 的滚动技术，以及基于块的视差卷轴特效。我们还将介绍一个强大的地图编辑器，用于创建基于块的地图。

*第 4 章，高级 UI*

本章覆盖了 jQuery UI 和 Ext JS 两个 UI 库。我们将探讨两个库的不同工作方式和

各自适合的应用类型。另外，我们还将构建一个三维旋转木马的示例。

*第 5 章，JavaScript 游戏介绍*

本章演示了如何用开放 Web 技术，而不是 Flash 插件来构建有趣的 Web 游戏。我们将通过开发一个怀旧的视频游戏来说明我们讨论的技术。

*第 6 章，HTML5 画布*

本章通过许多示例来深入介绍 Canvas 元素，包括如何使用 Canvas 和 WebSockets 创建一个图形化的聊天应用。其中涉及的画布主题包括：绘制、描边、填充、渐变、递归绘制、位图和动画。

*第 7 章，游戏和模拟中的向量*

本章介绍图形应用和游戏中广为使用的二维向量。代码示例包括大炮和火箭的模拟。

*第 8 章，谷歌可视化*

本章使用谷歌图表工具来对多种数据进行可视化，从基本的饼图到仪表图。本章不仅介绍了静态的可视化图形，而且覆盖了交互式的可视化图表，以及必要的数据格式化技术。

*第 9 章，使用 jQuery Mobile 为移动设备开发*

本章描述了 jQuery Mobile，一个基于 jQuery 的、面向移动设备的开发框架。jQuery Mobile 可以将普通的 HTML 页面转化为交互式和动画式的手机体验。本章中的主要例子是一个使用 jQuery UI、面向移动设备的图形化滑动解谜游戏。

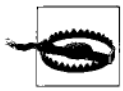
*第 10 章，用 PhoneGap 创建 Android 应用*

本章介绍如何使用 PhoneGap 将 Web 应用转换为手机的本地应用。本章解释了如何安装和配置 PhoneGap 来创建本地 Android 应用。在此之后，我们将把第 9 章的滑动解谜游戏转换为可以部署到移动设备上的本地应用。

## 本书的惯例

**提示**

这个图标用来强调一个提示、建议或一般说明。



**警告**
这个图标用来说明一个警告或注意事项。

本书中提到一些有用的网站和页面，通常除了页面 URL 外，还会提供页面名称。因此你可以选择直接输入 URL 或者通过搜索引擎搜索页面名称，找到相关页面，可以在地址比较复杂，或页面地址被改变时使用后者。

## 代码示例的使用

本书包含许多代码片段、示例和一些完整充实的应用。有时手动输入代码很麻烦，因此推荐从本书的代码库中复制代码。本书的许多代码中穿插了普通文本，直接从代码库复制代码可以避免你去拼接不同位置的代码。

在本书的 HTML 页面示例中，大部分使用 HTML5 文档类型：

```
<!DOCTYPE html>
```

为方便起见，示例中的所有 CSS 样式都被直接嵌入 HTML 页面。在实际 Web 应用开发中，还是推荐使用外部文件保存 CSS 样式。本书的示例代码可以在 http://www.professorcloud. com/supercharged 中找到。

## 目标浏览器

本书绝大部分示例代码都可以在较新的浏览器上工作，比如：

- Firefox 3.6x+
- Safari 4.0x+
- Opera 10.x+
- Chrome 5.x+
- Internet Explorer 8+

有些例子甚至可以在 IE6 和 IE7 上工作。

这些例子在 Windows XP、Windows Vista 和 Windows 7 上进行了完整测试，在 iOS 上进行了部分测试。理论上，这些例子也应能在上述浏览器的 Linux 版本上工作。

画布（Canvas）标签的使用则限于支持画布的浏览器，对 IE 来说，只有 IE9 可以（无需额外插件或库）直接支持。

有少量的例子需要特殊的环境，比如手机开发环境（PhoneGap）、服务器语言（PHP）或特殊浏览器。

如果是这种情况，书中会提到相关环境的设置和配置。

## Safari®在线图书

Safari 在线图书是一个按需订阅的数字图书馆。它有不少于 7500 本技术和创意相关的书籍和视频供你参考和搜索。

通过订阅，你可以在线阅读任何页面或视频，甚至可以从手机或移动设备上在线阅读。

你可以在书籍出版前访问到它们，并给作者发送反馈。其他功能还包括：复制和赋值代码、组织收藏夹、下载和标记章节、做笔记、打印等。

O'Reilly Media 已经将本书英文版上传到 Safari 在线图书服务。在 http://my.safaribooksonline.com 上免费注册，你就可以访问本书所有章节以及类似主题的书籍。

## 联系方式

如果你想就本书发表评论或有任何疑问，敬请联系出版社：

美国：

O'Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

中国：

北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室（100035）

奥莱利技术咨询（北京）有限公司

我们还为本书建立了一个网页，其中包含了勘误表、示例和其他额外的信息。你可以通过如下网址访问该网页：

http://www.oreilly.com/catalog/9781449393632

关于本书的技术性问题或建议，请发邮件到：

bookquestions@oreilly.com

欢迎登录我们的网站（http://www.oreilly.com），查看更多我们的书籍、课程、会议和最新动态等信息。

Facebook: http://facebook.com/oreilly

Twitter: http://twitter.com/oreillymedia

YouTube: http://www.youtube.com/oreillymedia

## 致谢

以作者一己之力出版一本书几乎是一件不可能的事情，在此我想特别感谢为本书做出贡献的人们。

- 感谢 Simon St.Laurent 为本书付出的热心、鼓励和帮助。
- 感谢所有的评审专家，特别是 Shelley Powers 为本书提供了大量的真知灼见。
- 感谢我的文字编辑 Rachel Monaghan，以及其他为本书完成提供帮助的伙伴。
- 感谢无私的开发者社区，自由地分享他们的工作和知识，来帮助推动 Web 的发展。
- 感谢我的妻子 Rebecca 和女儿 Sofa，能够容忍我随时随地带着笔记本准备写作。

# 目录

# 第 1 章

# 代码重用和优化

JavaScript 受到了许多不公平的评价。许多人说 JavaScript 在面向对象编程上存在局限，甚至有人认为 JavaScript 不能归为面向对象编程（OOP）语言。尽管 JavaScript 和 C++、Java 有许多相似之处，但它没有等价 Class 的声明，也没有显而易见的方式去实现流行的 OOP 技术，如继承（代码复用）和封装。JavaScript 的类型非常松散，也没有编译器，因此在运行出错前只能提供很少的错误或警告。JavaScript 是把双刃剑，一方面给了程序员很大的自由，另一方面也给程序员带来一些陷阱。

JavaScript 中充满了对传统编程“过失”的忽略，传统的程序员可能对此颇为郁闷。比如，在 JavaScript 中全局函数和变量是默认行为，而忘记加分号是完全可接受的。对 JavaScript 的工作方式缺乏了解，往往导致程序员无比郁闷。如果首先了解一些基础事实，将有助于你编写 JavaScript 应用：

- JavaScript 不是一个基于类的语言；
- 写好代码，并不一定需要基于类的面向对象编程语言。

有些编程人员尝试用 JavaScript 写 C++风格的代码。尽管在某种程度上可以达到目标，但最终结果往往让人感觉不自然。

没有任何编程语言是完美的，人们有理由争论某个编程语言或 OOP 本身的优越性是否仅仅是皇帝的新衣。根据我的个人经验，用 C++、Java 或 PHP 编写的软件生成的 bug 和问题，并不比用 JavaScript 编写的软件生成的少。我认为 JavaScript 的灵活性和表达力，可以使你更快地进行项目开发。

幸运的是，大部分 JavaScript 的缺点都不是无药可医。解决之道并不是一味模仿其

他语言，而是扬长避短：利用 Javascript 的灵活性，而小心避开难处理的部分。基于类的其他语言容易引起笨拙的类层次和臃肿的代码，JavaScript 则提供了同样有效但更轻量级的继承模式。

JavaScript 可以有许多种方法来实现继承。下面的代码使用原型继承来创建一个 Pet 对象，和一个继承它的 Cat 对象。JavaScript 教程中常常能见到这种“经典”的继承模式。

```
// Define a Pet object. Pass it a name and number of legs.
var Pet = function (name, legs) {
    this.name = name; // Save the name and legs values.
    this.legs = legs;
};

// Create a method that shows the Pet's name and number of legs.
Pet.prototype.getDetails = function () {
    return this.name + ' has ' + this.legs + ' legs';
};

// Define a Cat object, inheriting from Pet.
var Cat = function (name) {
    Pet.call(this, name, 4); // Call the parent object's constructor.
};

// This line performs the inheritance from Pet.
Cat.prototype = new Pet();

// Augment Cat with an action method.
Cat.prototype.action = function () {
    return 'Catch a bird';
};

// Create an instance of Cat in petCat.
var petCat = new Cat('Felix');

var details = petCat.getDetails();    // 'Felix has 4 legs'.
var action = petCat.action();         // 'Catch a bird'.
petCat.name = 'Sylvester';            // Change petCat's name.
petCat.legs = 7;                      // Change petCat's number of legs!!!
details = petCat.getDetails();        // 'Sylvester has 7 legs'.
```

上述代码可以工作，但不是特别优雅。如果你熟悉其他 OOP 语言比如 C++或 Java，new 声明是好理解的。但关键字 prototype 显得很啰嗦，并且没有隐私；注意外部代码将 petCat 的 legs 属性改成了一个不合理的值：7。这种继承方法没有提供对外部继承的保护，在涉及多个程序员的复杂项目中这个缺点也许会影响很大。

另一个选项无需使用 prototype 或 new，而是利用 JavaScript 的“函数继承（functional inheritance）”特性来吸收和增强对象实例（object instances）：

```
    // Define a pet object. Pass it a name and number of legs.
    var pet = function (name, legs) {
        // Create an object literal (that). Include a name property for public use
        // and a getDetails() function. Legs will remain private.
        // Any local variables defined here or passed to pet as arguments will remain
        // private, but still be accessible from functions defined below.
        var that = {
        name: name,
        getDetails: function () {
            // Due to JavaScript's scoping rules, the legs variable
            // will be available in here (a closure) despite being
            // inaccessible from outside the pet object.
            return that.name + ' has ' + legs + ' legs';
        }
    };
    return that;
};

// Define a cat object, inheriting from pet.
var cat = function (name) {
    var that = pet(name, 4); // Inherit from pet.
    // Augment cat with an action method.
    that.action = function () {
        return 'Catch a bird';
    };
    return that;
};

// Create an instance of cat in petCat2.
var petCat2 = cat('Felix');

details = petCat2.getDetails();   // 'Felix has 4 legs'.
action = petCat2.action();        // 'Catch a bird'.
petCat2.name = 'Sylvester';       // We can change the name.
petCat2.legs = 7;                 // But not the number of legs!
details = petCat2.getDetails();   // 'Sylvester has 4 legs'.
```

这里没有可笑的 prototype，而且所有东西都封装得很漂亮。最重要的是：legs 变量是私有的。如果尝试从 cat 外部修改不存在的公共 legs 属性，仅导致创建一个没有用过的 legs 属性。真正的 legs 值安全地保存在 pet 的 getDetails()方法创建的闭包（Closure）内部。闭包在函数执行结束后，保持了函数的局部变量。在这个例子中这个函数指的是 pet()。

事实上，用 JavaScript 实现继承并没有所谓“正确”的方法。但我个人认为函数继承方式非常自然。你和你的应用也许倾向其他方法。通过搜索“JavaScript Inheritance”你可以找到许多在线资源。

**提示**

使用原型继承的好处之一是内存效率；不管它被继承多少次，对象的原型属性和方法只被保存一次。

函数继承则相反：每个新的实例都会创建重复的属性和方法。如果你要创建许多（如上千个）大对象的实例，内存消耗可能会成为一个问题。不过这个问题很容易解决：可以将较大的属性或方法保存在一个对象中，并将其作为参数传给构建函数。这样所有实例就可以共同使用一个对象资源，而不是创建自己的版本。

## 1.1 快速运行

“快节奏的 JavaScript 图形编程”的概念也许听起来很矛盾（oxymoron）。

老实说，尽管 JavaScript 和 Wed 浏览器的组合不太可能创作出最尖端的游戏软件，但还是有很大空间来创建漂亮、快节奏和图形丰富的应用，包括游戏。可用的工具虽然不是最快的，但却免费、灵活、而且容易上手。

作为一种解释性语言，JavaScript 不能像 C++等语言那样从编译优化中获益。尽管现代浏览器已经大大提升了 JavaScript 效率，还是有很多空间来提高 JavaScript 应用的执行效率。这需要程序员去决定使用什么算法，优化哪段代码，如何以高效的方式操作 DOM。目前还没有一个健壮的优化工具可以为你完成这件事。

除非代码实在太烂，否则一般情况下简单的 JavaScript 应用（如仅处理少量鼠标单击或进行零散 AJAX 调用）是没必要进行代码优化的。而本书中涉及的应用则需要高效的代码才能保证用户体验——好的动画不应该慢或抖动。

本章剩余部分将不讨论如何提高从服务器加载页面的速度，而是考虑服务器资源加载之后的代码执行。具体来说，它讨论适用于 JavaScript 图形编程的优化技术。

## 1.2 优化什么，何时优化？

和优化技术同等重要的是：知道什么时候不优化。过早优化会带来晦涩的代码和 bug，优化很少执行的代码区域也没有必要。以帕莱托法则（即 80-20 法则）来看，20%的代码将占用 80%的 CPU 周期。程序员应该集中于优化这 20%、

10%或 5%，而忽略其他部分。这样 bug 会更少，大部分代码都保持了可读性，也保证了你的头脑清醒。

你可以用 Firebug 等性能测试工具，来了解哪些函数花费了绝大部分执行时间；然后检查这些函数并决定要优化的代码段。Firebug 性能测试器依赖火狐（Firefox）浏览器，有些浏览器有自己的性能测试器。而老版本的浏览器则不一定有类似的工具。

图 1-1 是 Firebug 性能测试器的界面。在 Console 菜单，选择 Profile 来开始性能测试，然后再选择 Profile 来停止测试。然后 Firebug 会显示所有在开始和结束点之间被调用的 JavaScript 函数分析。所显示的信息如下所示。


**图 1-1　运行中的 Firebug 性能测试器**

*Function*

被调用的函数名

*Percent*

在函数中所花费的时间和总时间的比例

*Call*

函数被调用的次数

*Own time*

在函数中所花费的时间（不包括对其他函数的调用）

*Time*

在函数中所花费的时间（包括对其他函数的调用）

*Average*

Own time 的平均值

*Min*

函数的最快执行时间

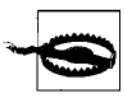
*Max*

函数的最慢执行时间

*File*

函数所在的 JavaScript 文件

如果你能自己创建适合所有浏览器的性能测试集，可以提高开发效率，并在没有测试工具的浏览器上使用。它只是将相同的测试页面载入到每一个浏览器，然后阅读测试结果。它也可以用来迅速检查函数内的细微优化。“自定义代码性能测试”一节将讨论如何创建自己的性能测试集。



**警告**

类似 Firebug 的调试器会给时间数值带来不小的误差。在执行性能测试之前，要确保关闭调试器。

“优化”是一个很宽泛的词，程序员可以从不同方面，以不同方式来对一个 Web 应用进行优化。

算法

> 应用程序是使用最有效的方法来处理数据的吗？代码优化没法修正一个差劲的算法。实际上，找对算法和 DOM 操作的高效一样，是保证应用快速运行最重要的因素之一。

有时，如果应用要求不高，一个慢但容易实现的算法就足够了。但如果性能是个问题，你也许需要检查研究一下当前所使用的算法。

本书不会讨论常见的搜索和排序等具体算法，因为读者可以在相关的计算机书籍和网络资源中找到许多关于它们的讨论。即使是游戏中涉及的 3D 图形学、物理和碰撞检测等这些更专业的问题和算法，也有很多书籍可以参考。

*JavaScript*

> 仔细检查调用得非常频繁的代码，在应用的某些关键区域中，对频繁执行部分的一个小小优化都会有不错的收益。

*DOM* 和 *jQuery*

> DOM 加 jQuery 是操纵 Web 页面非常方便的一种方式。但如果你没有注意到一些简单规则的话，也会成为性能重灾区。DOM 搜索和操作比较慢，应该尽量避免。

## 1.3 自定义代码性能测试

浏览器并不是运行准确代码性能测试的完美环境。短时间的定时器不够准确、事件的要求、零散的垃圾回收和系统上运行的其他进程都会导致结果偏差。一般可以这样来测试 JavaScript 代码的性能。

```
var startTime = new Date().getTime();
// Run some test code here.
var timeElapsed = new Date().getTime() - startTime;
```

这种方法虽然理论上可行，但由于前面提到的原因，现实中它不能给出准确的结果，尤其是当被测试代码只有几毫秒执行时间的情况下。

更好的方法是让被测试代码循环运行较长的时间（比如 1 秒），然后用在那段时间内完成的循环次数来评价性能。如果你要计算均值（mean）和中值（median）等统计信息，可以重复测试几次。

为保证测试运行较长时间，使用这个代码：

```
// Credit: based on code by John Resig.

var startTime = new Date().getTime();
for (var iters = 0; timeElapsed < 1000; iters++) {
   // Run some test code here.
   timeElapsed = new Date().getTime() - startTime;
}
// iters = number of iterations achieved in 1000 milliseconds.
```

无论系统性能如何，这些测试都会运行相同的时间。更快的系统会完成更多的循环次数。在实践中采用这种方法能得到较为一致的结果。

你可以运行 5 次这样的性能测试，每次 1 秒，循环次数的中值可作为最终的衡量标准。

# 1.4 优化 JavaScript

严格来说，任何用于 JavaScript 的优化也适用于其他语言。到了 CPU 层，道理都是一样的：尽量少做工作。在 JavaScript 中，CPU 层的工作和程序员距离太远，以至于很难确定到底 CPU 层进行了哪些工作。使用一些前人证实过可行的方法，一般来说对你的代码是有益处的，尽管只有通过实验测试才能明确证明。

## 1.4.1 查找表

高开销的计算可以预先进行，并将值存在一个查找表（lookup table）中，使用时给出简单的整型下标（index）就可以取出查找表中的值。只要查找表访问的代价比从头计算的代价低，你的应用程序就能因此获得更好的性能。比如，JavaScript 的三角函数就可以利用查找表加速。在这节中，将用一个查找表取代 Math.Sin()函数，并用它来构建一个图形动画的应用。

Math.sin()函数接受一个参数：角度（以弧度为单位），并返回一个−1～1 之间的值。角度参数的有效范围是 0～2π（约 6.283 18）弧度。这个范围对索引一个查找表没什么帮助，因为只有 6 个可能的整数。与其这样，不如完全不用弧度，而是让查找表接受 0～4 095 的整数索引。这个粒度对大多数应用来说足够了，但你可以通过给参数 steps 设置更大的值来得到更精密的查找表。

```
var fastSin = function (steps) {
    var table = [],
        ang = 0,
        angStep = (Math.PI * 2) / steps;
    do {
        table.push(Math.sin(ang));
        ang += angStep;
    } while (ang < Math.PI * 2);
    return table;
};
```

fastSin()函数将 2π 弧度分为参数中定义的步数，并将每一步得到的结果保存在数组中。

图 1-2 为 Math.sin()和查找表的性能测试结果的比较。

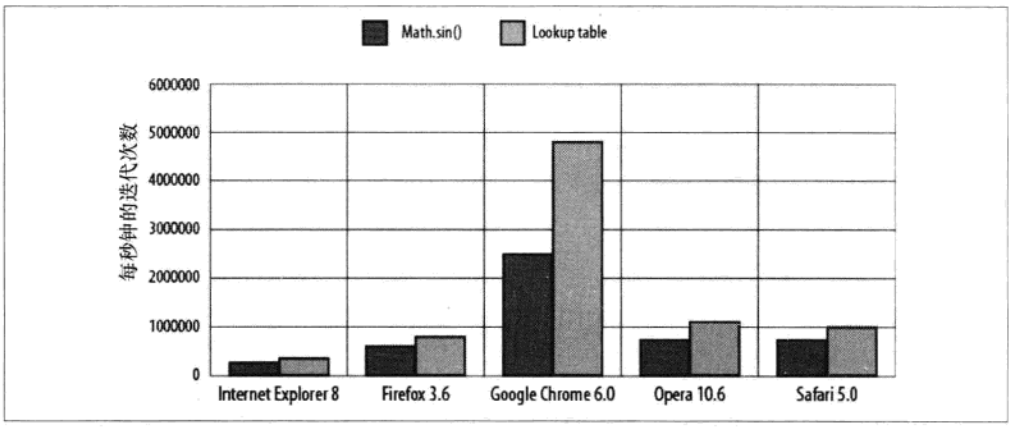



图 1-2　Math.sin()和查找表的性能测试对比。数值越大，性能越好

大多数浏览器上的性能提高大约有 20%，而 Google Chrome 上的提高幅度更大。如果查找表里面的值是由比 Math.sin()更复杂的函数计算出来的，查找表方法的性能优势将更明显；因为不论计算值的时间多长，查找表的访问时间保持不变。

下面的应用使用 fastSin()查找表来创建一个动画，其显示结果如图 1-3 所示。

```
<!DOCTYPE html>
<html>

    <head>
        <title>
            Fast Sine Demonstration
        </title>
        <script type="text/javascript"
```

```
            src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js">
        </script>
        <style type="text/css">
            #draw-target {
                width:480px; height:320px;
        background-color:#000; position:relative;
    }
</style>
<script type="text/javascript">
    $(document).ready(function() {
        (function() {
            var fastSin = function(steps) {
                var table = [],
                    ang = 0,
                    angStep = (Math.PI * 2) / steps;
                do {
                    table.push(Math.sin(ang));
                    ang += angStep;
                } while (ang < Math.PI * 2);
                return table;
            };
```

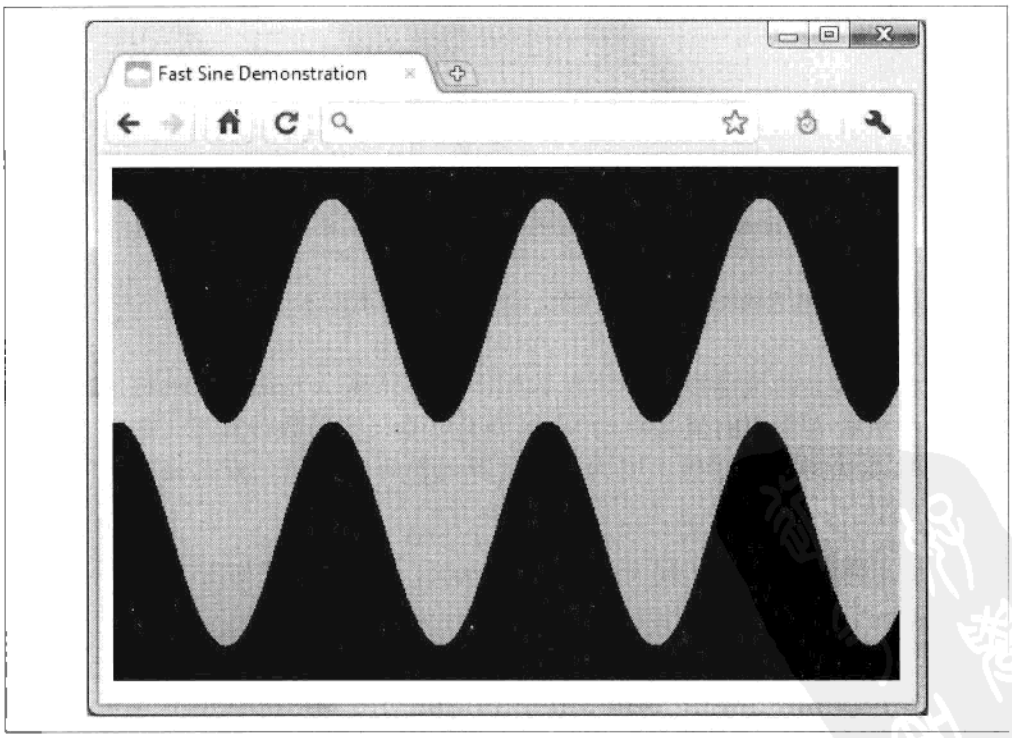



图 1-3　在一个动画应用中使用的 sine 查找表

下面的代码调用 fastSin()函数创建一个 sine 的查找表，保存在变量 sinTable[]中。

```
var sinTable = fastSin(4096),
    $drawTarget = $('#draw-target'),
    divs = '',
    i, bars, x = 0;
```

下面的 drawGraph()函数通过更新许多 1 像素宽的 div 的高度和位置，画出一个正弦波。表 1-1 列出了相关参数。

```
var drawGraph = function(ang, freq, height) {
    var height2 = height * 2;
    for (var i = 0; i < 480; i++) {
        bars[i].style.top =
        160 - height + sinTable[(ang + (i * freq)) & 4095]
            * height + 'px';
        bars[i].style.height = height2 + 'px';
    }
};
```

表 1-1　　传递给 drawGraph()的参数

| 参　数 | 描　述 |
|---|---|
| ang | 正弦波的开始角度 |
| freq | 正弦波的频率，定义了波的“紧密度” |
| height | 正弦波的幅度，也影响画线的宽度 |

下面的循环创建 480 个 1 像素宽的 div 元素。这些 div 被添加到$drawTarget 中。drawGraph()函数通过 bars[]数组来引用这些 div。

```
for (i = 0; i < 480; i++) {
    divs +=
        '<div style = "position:absolute;width:1px;height:40px;'
        + 'background-color:#0d0; top:0px; left: '
        + i + 'px;"></div>';
}
$drawTarget.append(divs);
bars = $drawTarget.children();
```

setInterval()函数以连续变化的参数，重复调用 drawGraph()，创造出动画效果：

```
    setInterval(function() {
        drawGraph(x * 50, 32 - (sinTable[(x * 20) & 4095] * 16),
            50 - (sinTable[(x * 10) & 4095] * 20));
        x++;
    }, 20);
})();

});
```

```
        </script>
    </head>

    <body>
        <div id="draw-target">
        </div>
    </body>

</html>
```

## 1.4.2 位操作、整数和二进制数

在 JavaScript 中，所有数都以浮点数形式表示。和 C++和 Java 等语言不同，JavaScript 语言中无法显示声明 int 和 float 类型。这个惊人的遗漏是由于 JavaScript 早期只是面向 Web 设计者和业余爱好者的简单语言。虽然 JavaScript 的单个数值类型帮程序员避免了许多数值类型错误，但毕竟整数更快，CPU 更容易处理，在许多情况下是其他语言的首选数值类型。

**提示**

ECMAScript规范中定义JavaScript的数值表示为“双精度64位的IEEE 754格式，即IEEE二进制浮点数算术标准”。其表示范围很广，大约从大数（$\pm 1.797\,693\,134\,862\,315\,7 \times 10^{308}$）到小数（$\pm 5\times 10^{-324}$）。不过需要注意的是：浮点数是有误差的，比如 alert(0.1+0.2)会显示0.300 000 000 000 000 04，而不是0.3。

不过，仔细阅读 ECMAScript 标准会发现 JavaScript 有几个内部操作可以处理整数：

ToInteger

转为整数

ToInt32

转为有符号 32 位整数

ToUint32

转为无符号 32 位整数

ToUint16

转为无符号 16 位整数

你不能直接使用这些操作，而是在执行位操作时被自动调用，使得数字被预

先转为合适的整型。虽然这些操作看起来和 Web 编程不相关，但实际上它们可用于优化。

**警告**

位操作将数字转为 32 位整数，数字范围为 -2 147 483 648 ~ 2 147 483 647。超过这个范围的数字也会被调整到这个范围。

### 1．二进制数的快速回顾

曾几何时，程序员经常要与二进制数打交道。使用彼时计算机所需的底层编程要求对二进制和十六进制有很好的理解。如今，二进制数很少被用在 Web 编程，但在硬件驱动和网络等领域仍有一席之地。

每个人都熟悉十进制数系统。在表 1-2 的第一行，从右到左每列所表示的权重从小到大是 10 的幂。将第二行的数字和对应的权重乘起来，并将所有乘积相加，就得到了最终数字为：

$$(3\times1\,000)+(9\times1)=3\,009$$

**表 1-2　　十进制数系统**

| 10 000 | 1 000 | 100 | 10 | 1 |
|---|---|---|---|---|
| 0 | 3 | 0 | 0 | 9 |

二进制数系统也是类似的，不同的是每列的权重为 2 的幂，而不是 10 的幂。第二行中的数字只能是 0 或 1，也称比特或位（bit）。二进制数简单的开关特性使其非常适合在数字电路中模拟。表 1-3 显示了十进制数 69 的二进制表示：

$$(1\times64)+(1\times4)+(1\times1)=69$$

**表 1-3　　十进制数 69 的 8 位二进制数表示**

| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |

二进制数如何取反？一般采用一个叫做补码的系统：

1．将二进制数中的每位取反，因此 01000101 变为 10111010。

2．加 1，因此 10111010 变为 10111011（-69）。

最左边的比特叫做符号位，0 代表正，1 代表负。使用同样的步骤，我们可以从-69回到+ 69。

### 2．JavaScript 的位操作

JavaScript 的位操作在整数的二进制数字（或位）上进行。

**位与（x&y）**：对操作数进行二进制与的操作，如果两个操作数的某一位都为 1，将对应的结果位设为 1。因此 0x0007&0x0003 的结果为 0x003。此操作可用于检查一个对象是否有一组属性或标记。表 1-4 显示了一个宠物对象的标记。一个小型、年老、棕色的狗可以用 64 + 16 + 8 + 2 = 90 来标记。

表 1-4　　一个宠物对象的二进制标记

| 大型 | 小型 | 年轻 | 年老 | 棕色 | 白色 | 狗 | 猫 |
|---|---|---|---|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

搜索一个有特定标记的宠物，只需要和搜索值进行位与操作。下面的代码搜索大型、年轻和白色的宠物（猫狗都可以）：

```
var searchFlags = 128 + 32 + 4;
var pets = []; // This is an array full of pet objects.
var numPets = pets.length;
for (var i = 0; i < numPets; i++) {
    if (searchFlags & pets[i].flags === searchFlags) {
        /* Found a Match! Do something. */
    }
}
```

整型有 32 位来表示不同的标记，而相比之下其他方法，如分开表示标记或其他类型的条件测试，要慢许多。比如：

```
var search = ['big','young','white'};
var pets = []; // This is an array full of pet objects.
var numPets = pets.length;
for (var i = 0; i < numPets; i++) {
    // The following inner loop makes things much slower.
    for(var c=0;c<search.length;c++) {
        // Check if the property exists in the pet object.
        if ( pets[i][search[c]] == undefined) break;
    }
    if( c == search.length ) {
        /* Found a Match! Do something. */
    }
}
```

&运算符也可达到类似取余运算符（%）的效果，也就是返回除法后的余数。下面的代码将保证变量 value 总是在 0 到 7 之间：

```
value &= 7; // Equivalent to value % 8;
```

不过这种等价性只有在&后面的值是 2 的幂−1（1，3，6，15，31，...）时才成立。

**位或（x|y）**：对操作数进行二进制或的操作，如果两个操作数的某一位至少有一个为 1，将对应的结果位设为 1。因此 0x0007|0x0003 的结果为 0x0007。

**位异或（x^y）**：对操作数进行二进制异或的操作，如果两个操作的某一位只有一个为 1，将对应的结果位设为 1。因此 0x0000^0x0001 的结果是 0x0001，而 0x0001^0x0001 的结果是 0x0000。这可以用于方便地切换变量：

```
toggle ^= 1;
```

每次执行 toggle^=1;，toggle 值将在 1 和 0 值之间转换（假设原来的值是 1 或 0）。下面是等价的 if-else 代码：

```
if (toggle) {
    toggle = 0;
}else {
    toggle = 1;
}
```

或者：

```
toggle = toggle ? 0:1;
```

**位非（~x）**：对所有位进行取反。例如 11100111 将变为 00011000。如果操作数是有符号整数（最左位为符号位），则~操作符等于取负减 1（前面提过补码中取负对应各位取反加 1）。

**位左移（x<<numBits）**：对 x 的二进制向左移 numBits 位。所有位向左移，最左的位丢失，0 填补最右的位。这等价于无符号整数的乘法 x*2^numBits。例如：

```
y = 5 << 1; // y = 10; Equivalent to Math.floor(5 * (2^1)).
y = 5 << 2; // y = 20; Equivalent to Math.floor(5 * (2^2)).
y = 5 << 3; // y = 40; Equivalent to Math.floor(5 * (2^3)).
```

测试显示左移位运算和对应的乘法运算符（*）相比没有性能提升。

**算术位右移（x>>numBits）**：对 x 的二进制向右移 numBits 位。除了（最左）符号位，所有位向右移，最右位丢失。这相当于有符号整数除法 x/2^numBits。例如：

```
y = 10 >> 1; // y = 5; Equivalent to Math.floor(5 / (2^1)).
y = 10 >> 2; // y = 2; Equivalent to Math.floor(5 / (2^2)).
y = 10 >> 3; // y = 1; Equivalent to Math.floor(5 / (2^3)).
```

测试显示右移位运算和对应的除法运算符（/）相比没有性能提升。

下面的代码看起来毫无用处：

```
x = y >> 0;
```

但是它使得 JavaScript 调用其内部的整数转换函数，剔除数字的小数部分。这实际上是一个快速的 Math.floor()函数。图 1-4 显示其在 IE8、Google Chrome 和 Safari 5.0 中都有速度提升。

**逻辑位右移（x>>>y）**：很少用到，类似>>操作符，但符号位不保留而填补为 0。对正数来说，这和>>操作符没两样；对负数来说，逻辑位右移的结果将成正数。例如：

```
y = 10 >>> 1; // y = 5;
y = -10 >>> 2; // y = 1073741821;
y = -10 >>> 3; // y = 536870910;
```

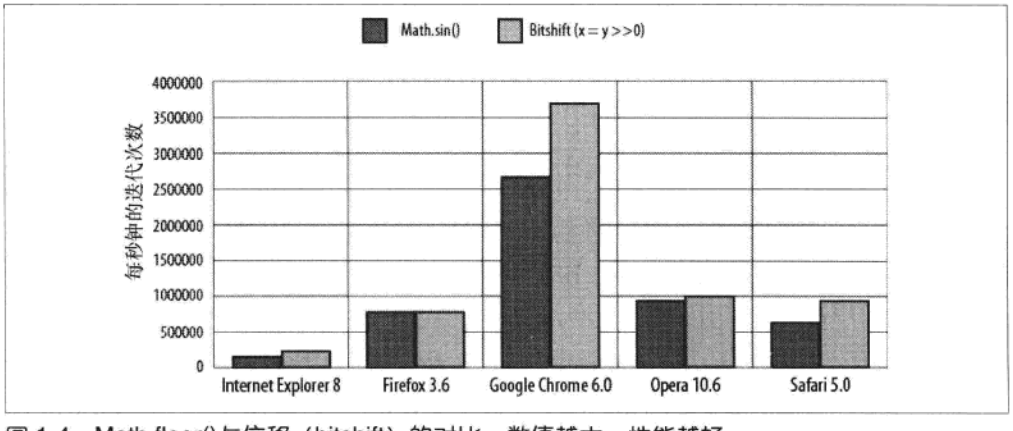



图 1-4　Math.floor()与位移（bitshift）的对比。数值越大，性能越好

### 3．循环展开：麻烦的真相

任何编程语言中的循环都会增加额外的开销。循环通常需要维护一个计数器和/或检查结束条件，这两者都花费时间。

移除循环开销将提供一些性能提升。一个典型的 JavaScript 循环如下：

```
for (var i = 0; i < 8; i++) {
    /*** do something here  **/
}
```

如果替换成下面的代码，你可以完全去除循环开销：

```
/*** do something here  ***/
/*** do something here  ***/
/*** do something here  ***/
/*** do something here  ***/
/*** do something here  ***/
/*** do something here  ***/
/*** do something here  ***/
/*** do something here  ***/
```

不过，对只有 8 次迭代的循环，性能的提升不大。假设循环体是一个简单语句（如 x++），循环展开可能会快 300%，但只是在毫秒级的；3 毫秒比 1 毫秒不会有很大的差别。如果循环体花费时间较长，那可能是 0.100 003 秒和 0.100 001 秒的差别，也不太值得去优化。

有两个因素决定了循环展开是否会带来可观的好处：

- 循环迭代的次数。事实上，需要许多（比如上千）个迭代才能带来明显的区别。
- 循环体开销和循环开销的比例。如果前者比后者的比例越大，性能提升越少。这是因为更多的时间是花费在循环体，而不是循环开销中。

要完全展开成千上百的迭代并不现实。现实的解决方案是使用达夫设备经典算法的变种，部分展开循环。比如，1 000 个迭代的循环可以分成 125 个展开 8 次的迭代：

```
// Credit: from Jeff Greenberg's site via an anonymous donor.
var testVal = 0;
var n = iterations % 8
while (n--)
{
    testVal++;
}

n = parseInt(iterations / 8);
while (n--)
{
    testVal++;
    testVal++;
    testVal++;
    testVal++;
    testVal++;
    testVal++;
    testVal++;
    testVal++;
  }
}
```

第一个 while 循环处理了不能被 8 整除的部分迭代。比如 1 004 次迭代需要 1 个 4 次（1 004%8）普通迭代的循环，然后跟着 125 个（parseInt(1 004/8)）展开的 8 次

迭代。下面是一个稍稍改进的版本：

```
var testVal = 0;
var n = iterations >> 3; // Same as: parseInt(iterations / 8).
while(n--){
    testVal++;
    testVal++;
    testVal++;
    testVal++;
    testVal++;
    testVal++;
    testVal++;
    testVal++;
}
n = iterations - testVal; // testVal has kept a tally, so do the remainder here.
while(n--) {
    testVal++;
}
```

**提示**

达夫设备指的是由 Tom Duff 在 1983 年开发的一种循环展开的 C 语言优化技术。循环展开是汇编语言中常用的技术，细小的优化就可以在内存复制等领域发挥作用。具有优化功能的编译器也可能进行自动的循环展开。

对一个循环体很少的 10 000 次循环的迭代，这会得到很大的性能提升。图 1-5 显示了结果。那我们应该像这样优化所有循环吗？不。这个测试是不现实的：循环体只有一个局部变量自增的操作是比较少见的。

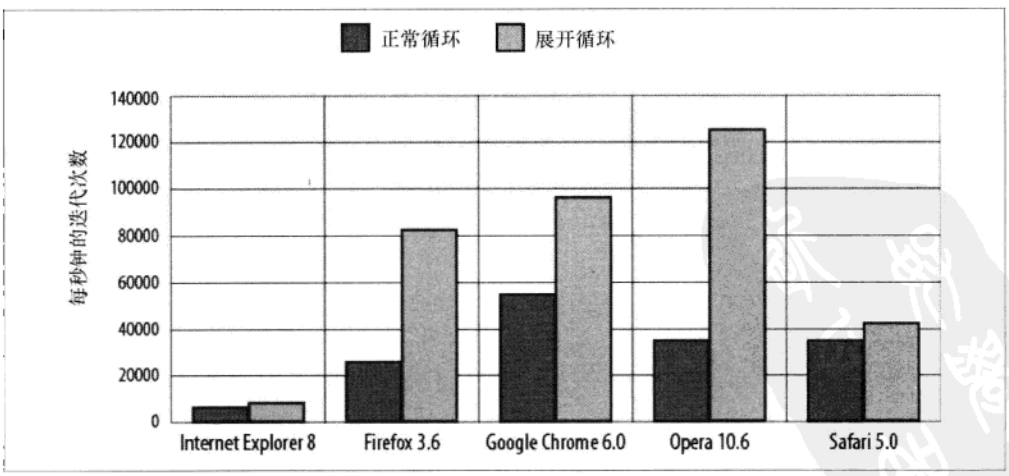



**图 1-5　展开一个循环体很少的 10 000 次循环。结果不错，但是不要激动。数值越大，性能越好**

一个更好的测试是迭代一个数组并用数组内容调用函数。下面是一个更接近现实的应用：

```
// Initialize 10000 items.
var items = [];
for (var i = 0; i < 10000; i++) {
    items.push(Math.random());
}
// A function to do some useful work.
var processItem = function (x) {
    return Math.sin(x) * 10;
};

// The slow way.
var slowFunc = function () {
    var len = items.length;
    for (var i = 0; i < len; i++) {
        processItem(items[i]);
    }
};

// The 'fast' way.
var fastFunc = function () {
    var idx = 0;
    var i = items.length >> 3;
    while (i--) {
        processItem(items[idx++]);
        processItem(items[idx++]);
        processItem(items[idx++]);
        processItem(items[idx++]);
        processItem(items[idx++]);
        processItem(items[idx++]);
        processItem(items[idx++]);
        processItem(items[idx++]);
    }
    i = items.length - idx;
    while (i--) {
        processItem(items[idx++]);
    }
};
```

图 1-6 显示了性能的提升。注意，一个更现实的循环体使得循环展开的作用大大降低。这就好像点了一个 4 000 卡路里的超级汉堡套餐，而希望低糖汽水可以帮助减肥。对于 10 000 次的迭代来说，图中的结果让人失望。

通过实验我们发现 JavaScript 循环实际上很高效，你需要在实际应用的背景下去测试循环展开这种优化技术，来实际测试它们的好处。

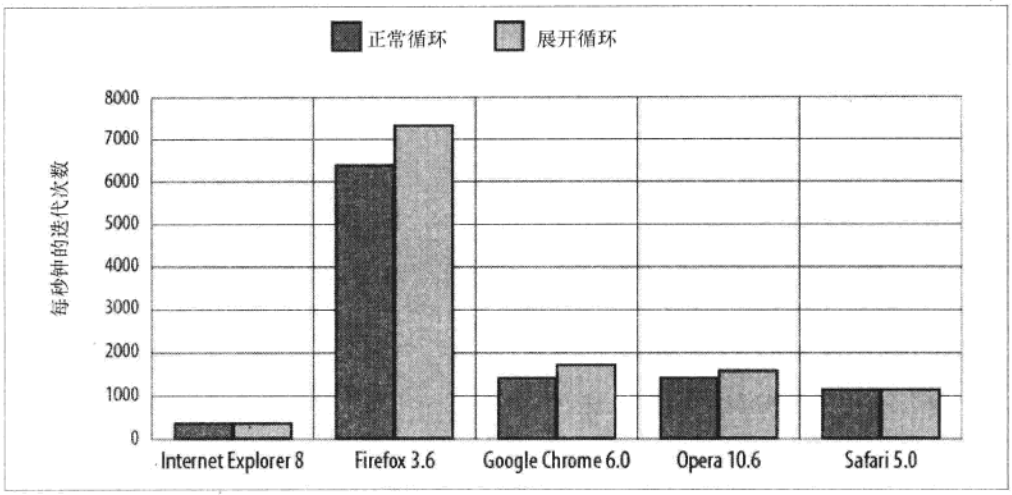



图 1-6　展开一个循环体很少的 10 000 次循环。结果令人失望。数值越大越好

# 1.5　优化 jQuery 和 DOM 交互

jQuery 是一个被广泛使用的 JavaScript 库，它能用简洁方便灵活的方式来访问和操作 DOM 元素，还可以减轻跨浏览器问题，使得你可以集中于核心应用开发而不是浏览器兼容问题。jQuery 以选择器引擎为核心，是你能以熟悉的 CSS 样式的选择器语言来找到 DOM 元素。例如，下面的代码返回一个 jQuery 对象（一种数组）包含所有具有“big” CSS 类的图像元素：

```
$images = jQuery('img.big');
```

或 jQuery 简写方式：

```
$images = $('img.big');
```

$images 变量只是一个普通变量，前面的$仅仅是提醒它引用一个 jQuery 对象。

有一点需要特别注意的是：一个看似简单无害的 jQuery 语句会在幕后做很多工作。如果只偶尔访问少量的元素，还没有太大关系。不过，如果要连续访问许多元素，比如在一个动画感很强的页面，就会严重地影响性能。

## 1.5.1　优化 CSS 格式变化

用 DHTML 创建 JavaScript 图形的一个基本操作就是快速操作 DOM 元素的 CSS 样式属性。在 jQuery 中，你可以这么做：

```
$('#element1').css('color','#f00');
```

这个语句会找到 id 是 element1 的元素，并修改其 CSS 颜色样式为红色。

这个语句分解开来做了这么些事：

- 调用 jQuery 并让它在 DOM 中搜索一个 id 为 element1 的元素。除了搜索本身之外，它还涉及进行正则表达式测试来决定需要搜索的类型。
- 返回找到的元素列表，一个特殊的 jQuery 数组对象。
- 调用 jQuery 的 css()函数。这会进行不同的检查，如决定是读或写，是否传入一个字符串参数、对象或者更多，最后更新元素样式本身。

连续进行这类的工作将会很慢，不管 jQuery 如何高效：

```
$('#element1').css('color','#f00');    // Make red.
$('#element1').css('color','#0f0');    // Make green.
$('#element1').css('color','#00f');    // Make blue.
$('#element1').css('left,'100px');     // Move a bit.
```

因为这里每行进行一次 id 为 element1 元素的搜索，这样很没有效率。

一个更快的方法是指定 jQuery 应该搜索的范围。jQuery 默认情况下要从 document 根或 DOM 层次的最上层开始搜索。而在许多情况下，这是没有必要的。如果你指定一个范围，会减少 jQeury 的搜索工作，更快地返回结果。

下面的例子搜索所有具有 alien CSS 类的元素，从环境参数（container）内的 DOM 元素中开始搜索：

```
$aliens = $('.alien', container); // Search within a specific DOM element.
```

环境参数的类型是灵活的，可以是另一个 jQuery 对象或 CSS 选择器：

```
// Start search within the elements of the jQuery object, $container.
$aliens = $('.alien', $container);

// Look for an element with id of 'container' and start the search there.
$aliens = $('.alien', '#container');
```

当然要确保搜索环境不比搜索元素本身慢。如果可能的话，应尽量直接引用 DOM 元素。

理想情况下，一旦元素被找到，你不应该再重新搜索它们。我们可以将搜索结果存起来：

```
var $elem = $('#element1');    // Cache the search results.
$elem.css('color','#f00');     // Make red.
$elem.css('color','#0f0');     // Make green.
$elem.css('color','#00f');     // Make blue.
$elme.css('left,'100px');      // Move a bit.
```

不过上面的代码中 jQuery 的 css()函数调用还是做了额外的工作，我们可以直接引用到 DOM 元素的实际式样对象中：

```
// Get the first element ([0]) from the jQuery search results and store
// a reference to the style object of that element in elemStyle.

var elemStyle = $('#element1')[0].style;

// It is now quicker to manipulate the CSS styles of the element.
// jQuery is not being used at all here:

elemStyle.color = '#f00';       // Make red.
elemStyle.color = '#0f0';       // Make green.
elemStyle.color = '#00f';       // Make blue.
elemStyle.left = '100px';       // Move a bit.
```

图 1-7 显示了前面 3 种方式的性能结果。在页面更复杂、CSS 选择器更慢的情况，这种差别会更加明显。

直接操作元素的属性本身比使用 jQuery 更快。比如，jQuery.html()方法要比直接使用一个元素的 innerHTML 对象要慢许多。

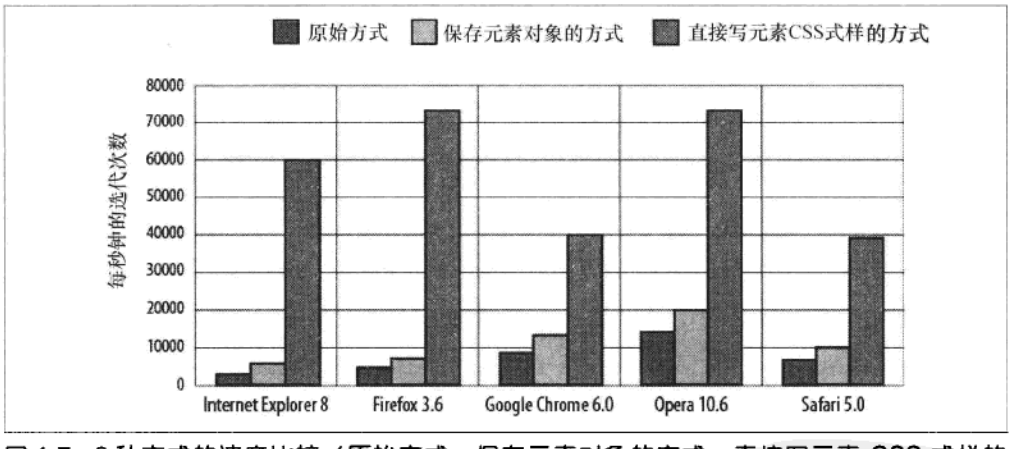



**图 1-7　3 种方式的速度比较（原始方式、保存元素对象的方式、直接写元素 CSS 式样的方式）**

图 1-7 的结果意味着我们应该完全不使用 jQuery 吗？不是，jQuery 是不容错过的，在某些环境下慢些是可以理解和接受的。但在速度很关键的代码区内应注意 jQuery 的使用方式。这通常只是所有代码中的一小部分。你的大部分应用程序可以（也应该）使用 jQuery 来快速方便地进行开发。

## 1.5.2　优化 DOM 插入

如果你的应用需要加入大量的元素到 DOM 中，可能会影响性能。DOM 是一个复

杂的数据结构，应尽量少去修改。这在动态 Web 页面中当然是不太可能的，因此你需要一个高效的方式来插入元素。

你可以通过 jQuery 来插入一个元素到 DOM 中：

```
$('#element1').append('<p>element to insert</p>');
```

这对几个元素来说是足够了，但当你需要插入成百上千个元素时，单个插入这些元素会太慢。

更好的方式是将所有要插入的元素构建为一个大的字符串，然后一次插入。对每个元素，这防止了 jQuery 调用和进行各种内部测试的开销。

```
var    elements = '';

// First build up a string containing all the elements.
for (var i = 0; i < 1000; i++) {
    elements += '<p>This is element ' + i + '</p>';
}

// They can now be inserted all at once.
$('#element1').append(elements);
```

## 1.6 其他资源

如果你想扩展自己的 JavaScript 知识，可以阅读下面两本经典书籍：

- JavaScript: The Definitive Guide by David Flanagan (O'Reilly; http://oreilly.com/catalog/ 9780596101992)
- JavaScript: The Good Parts by Douglas Crockford (O'Reilly; http://oreilly.com/catalog/ 9780596517748)

# 第 2 章

# DHTML 基础

在 HTML5 Canvas、SVG 和 Flash 等现代浏览器技术的背景下，DHTML 今天看起来有点过时。不过，就像龟兔赛跑中的龟，当更令人激动的方法不能保证可用的情况下，DHTML 总是那个更可靠的方案。

实际上，很多时候你只需要 DHTML 就够了；使用其他方法往往是因为开发者“想要”而不是“需要”。休闲游戏、图像缩放和许多其他特效都不需要借助其他“强力工具”就能完美实现。jQuery 这样的库还能使其操作起来更简单。熟练的 DOM 操作技术加上一点点想法就能保证 DHTML 图形的快速和流畅。

在本章，我们将用 vanilla JavaScript 和 DHTML 开发一个快速 sprite 系统。出于兼容性考虑，我们会避免使用语言的最新特性，而集中于核心 JavaScript 的有效使用。

## 2.1 创建 DHTML sprite

在计算机图形学中，sprite 是可以用软件控制移动的二维比特图对象。在三维多边形图形学之前，视频游戏几乎无一例外的使用 sprite 来生成可移动的角色。如今，移动设备上的休闲游戏和其他的用户界面效果等，引起了 sprite 图形的复兴。你可以用 DHTML 来模拟 sprite 功能。下面章节中，我们将创建一个用于不同应用的 DHTMLSprite 对象。尽管创建 sprite 效果有更新、更快的方法，如 HTML5 Canvas 元素，但普通的 DHTML 可以提供不错的浏览器兼容性，在许多情况下作为 Adobe Flash 的替代方案是完全可行的。

**提示**

本章中的 sprite 和 CSS sprite 是有区别的。CSS sprite 是一个流行的 Web 设计技术，指的是仅通过改变 HTML 元素的 CSS 背景位置，使得元素显示一个大背景图像的一小部分，一般用于实现动画效果。在计算机图形学术语中，这叫做动态纹理坐标。本章提到的 sprite，还是取其原意：一个移动的图形对象。同时我们也将用到 CSS sprite 技术来改变其图像。

DHTMLSprite 应该足够灵活以用在不同应用中，并提供下列功能：

- 用一个简单的函数调用和图像索引（index）来改变其图像（动画）。
- 在内部管理自身的 DOM 元素。
- 不改变 DOM 的情况下隐藏和显示自己。
- 移除其 DOM 元素并进行必要的清理。

## 2.1.1 图像动画

没有动画的 sprite 很没劲，因此我们需要一个简洁的方法来改变 sprite 中所用的图像。尽管 img 元素似乎是一个很明显的选择，但它需要对每个动画帧载入不同的图像文件。有一个更好的办法可以使用少量的图像文件，而处理多个 sprite 图像。

CSS 的 background-position（背景位置）属性使得 HTML 元素（如一个 div）可以显示图像的一小部分。因此一个大图像可以作为许多小 sprite 图像的容器。要使用这些 sprite 图像，我们必须定义 background-position 属性在 div 内的水平和垂直位移，以及宽和高。但这种动画方式并不直接，而需要技巧。最好是通过简单的索引就能引用到 sprite 图像。比如在图 2-1 中，组成一个齿轮动画的 5 幅图像可以用索引 0、1、2、3 和 4 表示。而第一个正方形用索引 5 表示，依此类推。

我们需要将索引转化为容器图像内的像素位移。一种方法是手动创建一个表格来记录 sprite 图像索引和对应的像素位移。尽管这个方法很有效，但手动输入和更新这些位移将很枯燥。更好的方法是通过计算得到这些位移。

将索引转换为水平和垂直像素位移只需要很简单的算术。在图 2-1 中，容器图像是 256 像素宽，每个 sprite 图像（底层除外）是 64 像素的正方形。像素位移可以用 JavaScript 这样计算：

```
// This code is unoptimized, but illustrates the calculations required.
var vertOffset = -Math.floor (index * 64 / 256) * 64; // 64 is the sprite height.
var horizOffset = -(index * 64 % 256);                // 64 is the sprite width.
```

注意计算出的值是负数。想象 div 元素是在对准第一个齿轮图像（索引为 0）、宽与高各 64 像素的正方形。为了显示索引为 1 的下一张图像，容器图像必须向左移 64 像素（负水平位移）。如果要显示索引为 4 的最后一个齿轮图像，容器图像必须向上移 64 像素（负垂直位移）。

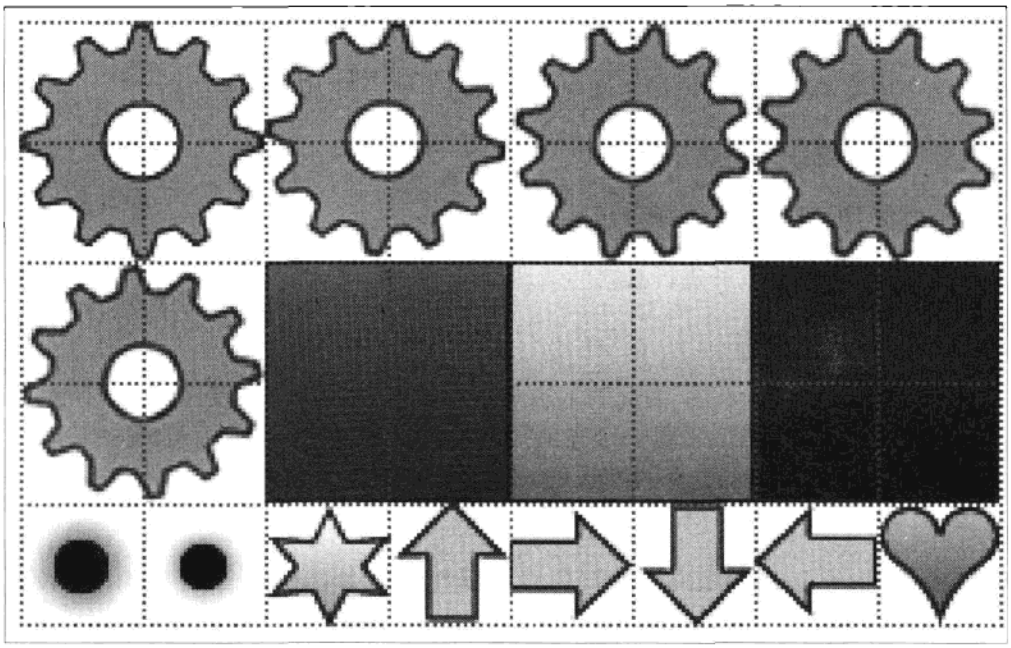


图 2-1　嵌入到一个容器图像中的动画图像，每个虚线格子为 32 像素宽、高的正方形

如何处理不同大小的 sprite 呢？在图 2-1 中，在容器图像底部有一些更小的 32 像素宽、高的 sprite 图像。

决定像素位移的计算和之前一样，不同的是 sprite 大小改为 32 像素：

```
// This code is unoptimized, but illustrates the calculations required.
var vertOffset = -Math.floor (index * 32 / 256) * 32; // 32 is the sprite height.
var horizOffset = -(index * 32 % 256);                // 32 is the sprite width.
```

考虑到现在的 sprite 大小是 32 像素，图 2-1 中第一个 32 像素的 sprite 图像（底行第一个小黑圈）的索引为 32。只要 sprite 图像的边缘坐标是它们大小的倍数，就可以使用索引计算的方式。

**提示**

图 2-1 中的容器图像是一个 32 位 PNG 文件，支持百万颜色和一个用于透明度的 alpha 通道。不过，32 位 PNG 不适用于 IE6，因为透明区域会变成不透明的灰色。一个解决方案是将图像存为 8 位的调色板 PNG。这可以在 IE6 中正确显示，不过半透明区域会完全消失并显示粗糙的边缘。

## 2.1.2 封装和画图抽象

将所有 DOM 操作细节，封装在 DHTMLSprite 中，隐藏在使用它的应用之外，会使代码更简单更易维护；应用可以集中于应用逻辑而不是画图细节。由于应用逻辑和画图细节的分离，将应用转为另一个画图方法如 HTML5 Canvas 元素或 SVG 变得更简单，甚至可以使应用程序根据浏览器能力选择合适的画图方法。

## 2.1.3 最小化 DOM 插入和删除

重复的增删和销毁 DOM 元素对性能会有不利的影响。为了降低性能影响，可以初始化一个隐藏的 sprite 列表。当需要 sprite 时，你可以将其从列表中取出并使其可见，而不是真的在 DOM 中插入新的东西。当 sprite 不再需要时，你可以将其隐藏并放回列表中。在 DHTMLSprite 中提供一个 show 和 hide 方法将使应用实现这项技术。

如果要永久地移除一个 DHTMLSprite，应移去其 DOM 元素并进行相关的其他清理工作。

## 2.1.4 sprite 代码

与其将若干单独的参数传给 sprite，不如将所有设置参数放入叫做 params 的对象传入。除了避免参数次序的麻烦之外，还使从 DHTMLSprite 继承的其他对象，可以将它们自己的设置参数加入 params 中。任何使用 params 的对象都可以忽略跟它不相关的参数。表 2-1 显示了 params 对象中的参数。

```
var DHTMLSprite = function (params) {
```

表 2-1 DHTMLSprite 对象参数

| 参　数 | 描　述 |
| --- | --- |
| images | 图像文件的路径 |

续表

| 参　数 | 描　述 |
| --- | --- |
| ImagesWidth | 图像文件的像素宽度 |
| width | sprite 的像素宽度 |
| height | sprite 的像素高度 |
| $drawTarget | sprite 将要附加于的父元素 |

下面，我们将 params 属性复制为局部变量。通过局部变量访问参数比通过 params 对象的属性要快。如此定义的局部变量是私有的，只能从 DHTMLSprite 内的方法访问。

```
var width = params.width,
    height = params.height,
    imagesWidth = params.imagesWidth,
```

接下来，我们在 params.$drawTaget 指定的 DOM 元素后加上一个 sprite div 元素。$element 保存了一个对 sprite div 的引用。变量和属性名前的$符号用做提醒它们指向 jQuery 对象。elemStyle 直接引用了 sprite div 的 style 属性，用于快速更新其 CSS 属性。

```
$element = params.$drawTarget.append('<div/>').find(':last'),
elemStyle = $element[0].style,
// Store a local reference to the Math.floor function for faster access.
mathFloor = Math.floor;
```

现在我们要给 sprite div 元素设置初始 CSS 属性。因为我们只进行一次初始化，因此可以使用方便的 jQuery css()函数，尽管这也许不是改变属性最快的方式。

```
$element.css({
    position: 'absolute',
    width: width,
    height: height,
    backgroundImage: 'url(' + params.images + ')'
});
```

下面我们要在 that 中创建并保存一个 DHTMLSprite 对象。它包含了所有的 sprite 方法，注意 that 的方法可以访问前面定义的局部变量。这个 that 对象创建了一个闭包，它能永久访问前面 DHTMLSprite 函数里定义的变量。

```
var that = {
```

draw 方法更新 sprite div 元素的位置：

```
draw: function (x, y) {
    elemStyle.left = x + 'px';
    elemStyle.top = y + 'px';
},
```

changeImage()方法改变显示的 sprite 图像。将索引转为像素位移的方法和前面描述的一样，但有些小的优化：

- 局部变量 mathFloor()指向 Math.floor()函数，我们通过前者来调用后者。
- index 变量只乘一次。

```
changeImage: function (index) {
    index *= width;
    var vOffset = -mathFloor(index / imagesWidth) * height;
    var hOffset = -index % imagesWidth;
    elemStyle.backgroundPosition = hOffset + 'px ' + vOffset + 'px';
},
```

然后，我们定义隐藏、显示和移除 sprite div 元素的方法：

```
        show: function () {
            elemStyle.display = 'block';
        },
        hide: function () {
            elemStyle.display = 'none';
        },
        destroy: function () {
            $element.remove();
        }
    };
    // Return the instance of DHTMLSprite.
    return that;
};
```

## 2.1.5 一个简单的 sprite 应用程序

下面是一个基本的 HTML 页面，它初始化并显示了两个 sprite。

```
<!DOCTYPE html>
<html>
    <head>
        <title>
            Sprite Demonstration
        </title>
        <script type="text/javascript"
            src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js">
        </script>
        <style type="text/css">
            #draw-target {
                width:480px;
                height:320px;
                background-color: #ccf;
                position:relative;
```

```
        }
    </style>
    <script type="text/javascript">
        var DHTMLSprite = function(params) {
            /*** DHTMLSprite code removed for conciseness ***/
        };

        $(document).ready(function() {
```

为了创建 sprite，我们需要一个包含初始化参数的对象：

```
var params = {
    images: '/images/cogs.png',
    imagesWidth: 256,
    width: 64,
    height: 64,
    $drawTarget: $('#draw-target')
};
```

下面创建两个 sprite。因为两个 sprite 大小相等并使用同一个 DOM 画图区域，所以不需要改变任何参数。第一个 sprite 使用默认索引值 0，而第二个 sprite 的图像索引值为 5。

```
var sprite1 = DHTMLSprite(params),
    sprite2 = DHTMLSprite(params);
sprite2.changeImage(5);
```

最后画出这两个 sprite。图 2-2 显示了输出结果。

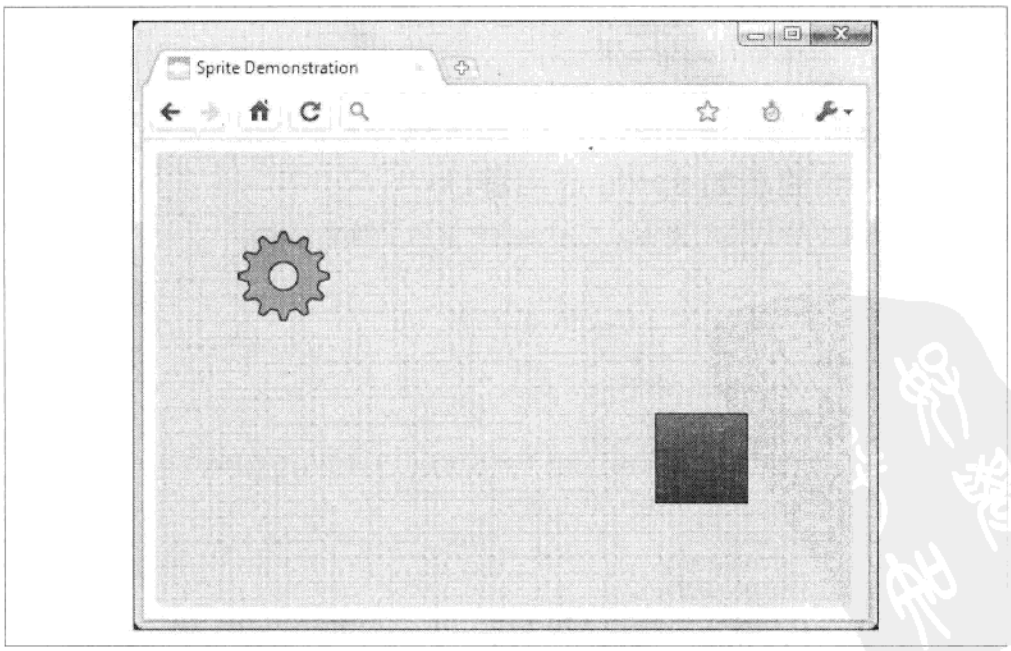



图 2.2　画出来的两个 sprite

```
                sprite1.draw(64, 64);
                sprite2.draw(352, 192);
            });
        </script>
    </head>
    <body>
        <div id="draw-target">
        </div>
    </body>
</html>
```

这个应用中没有移动也没有动画，让我们在下一个例子中“动”起来。

## 2.1.6 一个更动态的 sprite 应用程序

下面的应用展示了 sprite 的存在价值：动画和移动。之前我们画了两个 sprite，而没有控制它们移动。这个例子中我们定义一个新对象：bouncySprite，一个会反弹的 DHTMLSprite。实现方法之一是在 bouncySprite 中创建一个 DHTMLSprite，并将其作为单独的实例控制。更简洁的方法是让 bouncySprite 继承所有 DHTMLSprite 的能力，并添加自己额外的能力。在 JavaScript 中这种继承和增强很简单：

```
var bouncySprite = function (params) {
```

为了提高速度，我们用局部变量保存设置参数。这里的 params 对象也包含 DHTMLSprite 的参数，但这些和 bouncySprite 无关。表 2-2 显示了传入的参数。

```
var x = params.x,
        y = params.y,
        xDir = params.xDir,
        yDir = params.yDir,
        maxX = params.maxX,
        maxY = params.maxY,
```

表 2-2 bouncySprite 对象参数

| 参　数 | 描　述 |
| --- | --- |
| x | 像素 x 位置 |
| y | 像素 y 位置 |
| xDir | x 移动方向 |
| yDir | y 移动方向 |
| maxX | 最大 x 位置 |
| maxY | 最大 y 位置 |

animIndex 保存了当前动画图像索引：

```
animIndex = 0,
```

我们在 that 中创建和引用一个 DHTMLSprite。params 对象包含了其设置参数。

```
that = DHTMLSprite(params);
```

接着给 that 引用的 DHTMLSprite 实例加一个 moveAndDraw 方法，实际上就是创建一个 bouncySprite 实例：

```
that.moveAndDraw = function () {
```

通过增加 xDir 和 yDir 变量来移动 sprite 的 x 和 y 位置：

```
x += xDir;
y += yDir;
```

下面的代码根据 xDir 方向对 animIndex 变量进行增或减，接着用取余操作（%）将其维持在-4 到+4 之间。如果 animIndex 是负的，纠正到对应的正索引。

```
animIndex += xDir > 0 ? 1 : -1;
animIndex %= 5;
animIndex += animIndex < 0 ? 5 : 0;
```

接着检查 bouncySprite 是否超过了 maxX 和 maxY 定义的范围。如果超过，对移动的方向取负，使 bouncySprite 弹回。

```
if ((xDir < 0 && x < 0) || (xDir > 0 && x >= maxX)) {
    xDir = -xDir;
}
if ((yDir < 0 && y < 0) || (yDir > 0 && y >= maxY)) {
    yDir = -yDir;
}
```

更新 bouncySprite 动画索引，并将其画到新位置：

```
    that.changeImage(animIndex);
    that.draw(x, y);
};
```

返回在 that 中引用的 bouncySprite 实例，供应用程序使用：

```
    return that;
};
```

定义了 bouncySprite 对象后，我们可以初始一些对象，并在 setInterval()或 settTimeout()控制下调用它们的 moveAndDraw()方法。更好的方法是创建一个对象可以初始化和处理任意数量的 bouncySprite。这个对象可以叫做 bouncyBoss。bouncyBoss 可以传入两个参数，如表 2-3 所示。

```
var bouncyBoss = function (numBouncy, $drawTarget) {
```

表 2-3　　bouncyBoss 对象参数

| 参　数 | 描　述 |
| --- | --- |
| numBouncy | 要初始化的 bouncySprite 个数 |
| $drawTarget | bouncySprite 要添加到的目标父元素 |

创建指定数目的 bouncySprite，并放入 bouncys 数组中。每个 bouncySprite 给一个随机起始位置和移动方向（xDir 和 yDir），并根据$drawTarget 的宽和高计算最大范围。

```
var bouncys = [];
for (var i = 0; i < numBouncy; i++) {
    bouncys.push(bouncySprite({
        images: '/images/cogs.png',
        imagesWidth: 256,
        width: 64,
        height: 64,
        $drawTarget: $drawTarget,
        x: Math.random() * ($drawTarget.width() - 64),
        y: Math.random() * ($drawTarget.height() - 64),
        xDir: Math.random() * 4 - 2,
        yDir: Math.random() * 4 - 2,
        maxX: $drawTarget.width() - 64,
        maxY: $drawTarget.height() - 64
    }));
}
```

现在我们定义 moveAll 方法，它调用了 bouncys 数组中每个 bouncySprite 的 moveAndDraw 方法。每次移动，它创建一个 setTimeOut 来调用自己，实现连续的循环。

```
    var moveAll = function () {
        var len = bouncys.length;
        for (var i = 0; i < len; i++) {
            bouncys[i].moveAndDraw();
        }
        setTimeout(moveAll, 10);
    }
    // Call the moveAll() function to start.
    moveAll();
};
```

下面是使用新 bouncyBoss 对象的页面布局：

```
<!DOCTYPE html>
<html>
    <head>
        <title>
            Sprite Demonstration
        </title>
        <style type="text/css">
            #draw-target {
                width:480px;
                height:320px;
```

```
                background-color:#ccf;
                position:relative;
            }
            </style>
            <script type="text/javascript"
                src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.js">
            </script>
            <script type="text/javascript">
                var DHTMLSprite = function(params) {
                    /*** DHTMLSprite code removed for conciseness ***/
                };
                var bouncySprite = function(params) {
                    /*** bouncySprite code removed for conciseness ***/
                };
    var bouncyBoss = function(numBouncy, $drawTarget) {
        /*** bouncyBoss code removed for conciseness ***/
    };
    $(document).ready(function() {
```

一个 bouncyBoss 创建了 50 个 bouncySprite 对象，并连续调用它们的 moveAndDraw 方法。图 2-3 显示了输出结果。

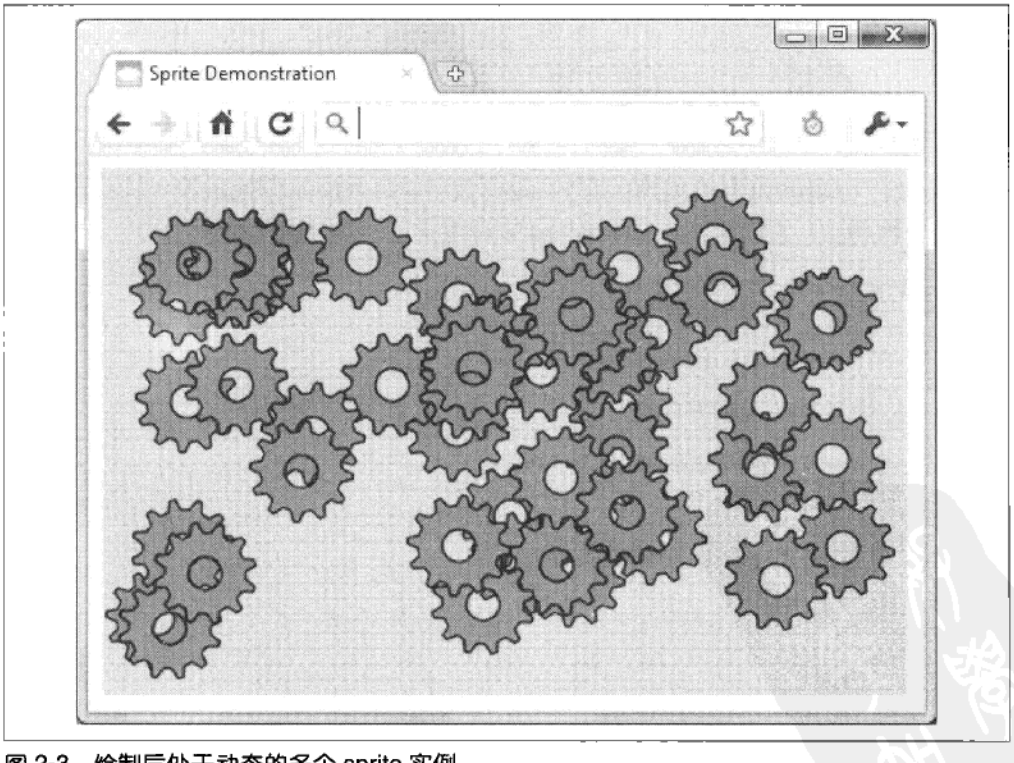



图 2-3　绘制后处于动态的多个 sprite 实例

```
            bouncyBoss(50, $('#draw-target'));
        });
    </script>
</head>
<body>
    <div id="draw-target">
    </div>
</body>
</html>
```

## 2.2 转为一个 jQuery 插件

将 bouncySprite 转为一个 jQuery 插件，可以利用 jQuery 通过 CSS 选择器搜索并返回 DOM 元素列表的能力。这个插件可以搜索任何元素并用 bouncyBoss 给它附上多个 bouncySprite 实例，并可以改变附加的实例个数和背景颜色。

将 bouncySprite 转为一个灵活的 jQuery 插件，实际没有想象得那么难。因为 DHTMLSprite、bouncySprite 和 bouncyBoss 对象是以模块化方式开发的，可以很顺利地转为 jQuery 插件结构。

下面这个单独的分号看起来奇怪，但它可以避免前面的代码遗漏分号可能导致的问题。通常这不会是个问题，因为 JavaScript 通过换行符将插件代码识别为新语句。不过，如果代码和插件是压缩的，空格和换行符也许会被删除。而插件也许会因为它和前面代码之间缺乏换行符而失效。

```
;    // Initial solitary semicolon.
```

下面我们定义一个匿名函数。这将所有插件代码包成一个自给自足的语境。$是传入的参数，这里指的是全局 jQuery 对象本身（见插件最后一行）。现在，在插件中我们可以使用$()代替 jQuery()来调用 jQuery。传 jQuery 对象似乎没必要，因为它已经在全局定义了。不过，它可以避免由于外部代码（比如其他 JavaScript 库）重新定义$变量而导致插件不能使用$()调用 jQuery。

```
(function ($) {
```

为增强 jQuery 的能力，我们将对插件的引用保存在 jQuery 的 fn 属性中。如果另一个插件定义了同样的名字，可能会出现冲突。为避免这种情况，你应该起个有想象力的名字。比如，“zoom”很可能冲突，而“cloudZoom”则不太会冲突。

```
$.fn.bouncyPlugin = function (option) {
```

此处我们插入 DHTMLSprite、bouncySprite 和 bouncyBoss 的代码，插入时没必要做任何修改。因为它们以局部变量保存，所以是对插件私有的。

```
var DHTMLSprite = function (params) {
    /*** DHTMLSprite code removed for conciseness ***/
};
var bouncySprite = function (params) {
    /*** bouncySprite code removed for conciseness ***/
};
var bouncyBoss = function (numBouncy, $drawTarget) {
/*** bouncyBoss code removed for conciseness ***/
};
```

这个插件使用 option 对象的属性作为选项。用这种方式给插件传入选项比较灵活，因为它可以传入所有参数、一些参数或者空参数。jQuery 的 extend 函数融合了 option 属性和$.fn.bouncyPlugin.defaults 对象定义的默认 option 属性。option 属性具有优先权，它们不存在时才使用默认属性。因为默认选项是公共的，一个应用可以通过在$.fn.bouncyPlugin.defaults 中创建新的 defaults 对象来改变默认选项。

```
option = $.extend({}, $.fn.bouncyPlugin.defaults, option);
```

这个插件在找到的 DOM 元素列表上进行迭代，对每个元素执行一个匿名函数。在这个函数里，this 指的是 DOM 元素列表中的当前元素。函数从 this 中创建一个 jQuery 对象，并存在$drawTarget 中。option 中定义的背景颜色被应用到$drawTarget 中，option 中指定的 numBouncy 则传入一个新的 bouncyBoss 实例。

```
        return this.each(function () {
            var $drawTarget = $(this);
            $drawTarget.css('background-color', option.bgColor);
            bouncyBoss(option.numBouncy, $drawTarget);
        });
    };
    $.fn.bouncyPlugin.defaults = {
        bgColor: '#f00',
        numBouncy: 10
    };
})(jQuery);
```

下面是一个包含此插件的 HTML 页面。显示结果如图 2-4 所示。

```
<!DOCTYPE html>
<html>
   <head>
      <title>
          Sprite Demonstration
      </title>
      <style type="text/css">
          .draw-target {
              width:320px;
              height:256px;
              position:relative;
              float:left;
              margin:5px;
```

```
        }
    </style>
    <script type="text/javascript"
        src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.js">
    </script>
    <script type="text/javascript">
```

在此处插入插件：

```
;(function($) {
    $.fn.bouncyPlugin = function(option) {
        /*** bouncyPlugin code removed for conciseness ***/
    };
})(jQuery);
```

当页面准备好后，在指定元素上调用此插件，此例指的是任何有 draw-target CSS 类的元素。

```
        $(document).ready(function() {
            $('.draw-target').bouncyPlugin({
                numBouncy: 20,
                bgColor: '#8ff'
            });
        });
    </script>
</head>
<body>
```

接着，我们定义 4 个 draw-target 类的 div 元素：

```
    <div class="draw-target">
    </div>
    <div class="draw-target">
    </div>
    <div class="draw-target">
    </div>
    <div class="draw-target">
    </div>
  </body>
</html>
```

关于 jQuery 的更多细节，请阅读 jQuery 社区专家撰写的 jQuery Cookbook（O'Reilly; http://oreilly.com/catalog/9780596159788）。

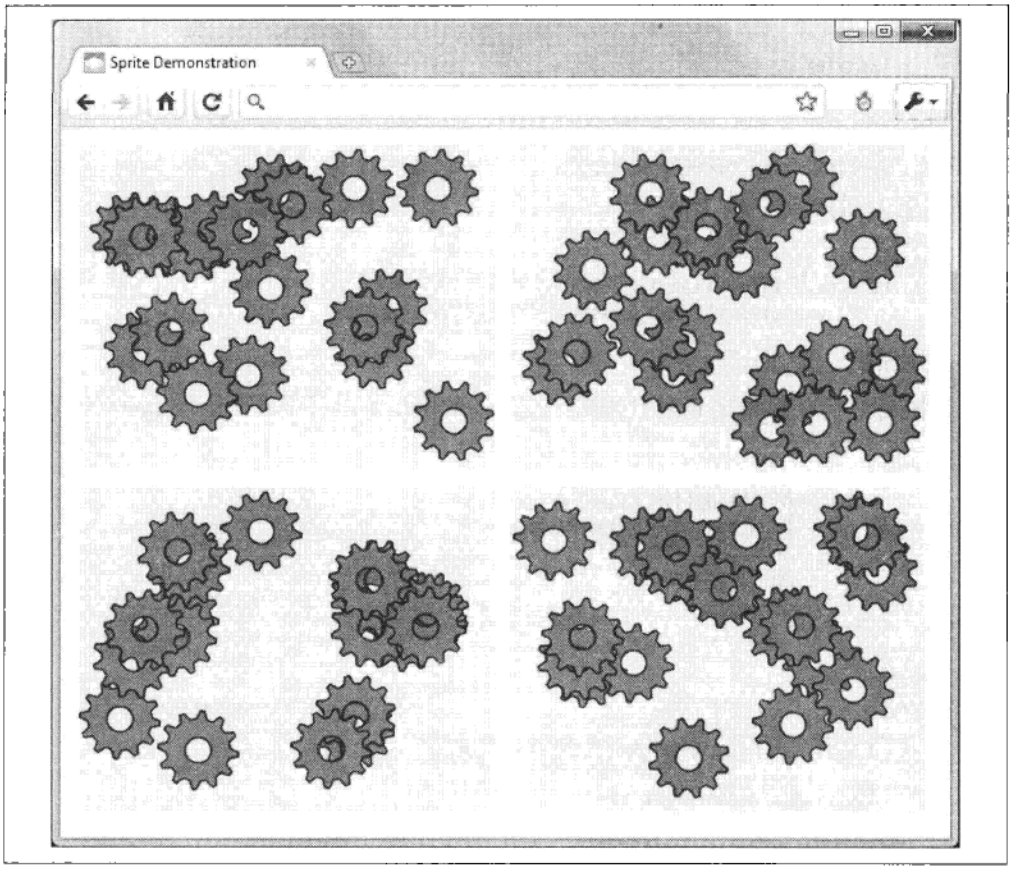



图 2-4　通过一个 jQuery 插件生成的多个 bouncyBoss 对象实例

## 2.3　定时器、速度和帧速率

本节将讨论如何在 JavaScript 中控制图形的更新速度以保证用户体验。我们想要图形有平滑流畅的移动，不要太快也不要太慢。用户计算机的性能会影响图形更新的速度。下面我们将讨论减少不同机器上速度差异的解决方案。

### 2.3.1　使用 setInterval 和 setTimeout

JavaScript 的 setInterval()和 setTimeout()函数使你可以定期调用 JavaScript 代码。需要定期更新图形的应用，比如电脑游戏，几乎都离不开这些函数。

你可以将回调函数传给 setInterval()来重复调用此函数：

```
// This is a callback function.
var bigFunction = function() {
    // Do something...
    // This code needs to be called regularly.
    // It takes 20ms to execute
};

// setInterval will attempt to call bigFunction() every 50 milliseconds.
setInterval(bigFunction, 50);
```

注意执行 bigFunction()花费 20 毫秒。如果循环间隔比这个值小呢?

```
setInterval(bigFunction, 15);
```

看起来 20 毫秒的 bigFunction()会在第一个回调函数返回之前被重新调用。实际上，新的回调将排在队列中直到前面的回调函数结束。

如果延时更短点呢?

```
setInterval(bigFunction, 5);
```

可以预计每执行一个回调函数，若干个回调函数在排队。事实上，通常的行为是只有一个排队的 bigFunciton()会激活。排队的回调函数会在第一个回调函数结束后立即执行吗？有可能，但不一定。其他时间和浏览器中运行的代码可能使得 setInterval()回调函数被延时或丢弃。回调函数甚至有可能连续发生（比规定的间隔短），如果 JavaScript 发现一个时间窗口，可以清除队列。

这里想要说明的是：不能保证回调函数以指定的间隔执行。

setTimeout()在指定的延时后调用一个函数，和 setInterval()类似，但可预见性更强。

```
setTimeout(bigFunction,50);
```

这会在 50 毫秒后，调用一次 bigFunction()。和 setInterval()一样，这个延时仅仅是一个参考。

你可以用 setTimeout()连续调用一个函数，其行为将比 setInterval()更可预见：

```
// This is a callback function.
var bigFunction = function() {
    // Do something...
    // This code needs to be called regularly.
    // It takes 20ms to execute
    setTimeout(bigFunction, 10);
};
```

每当 bigFunction()结束，它设置另一个以自己作为回调函数的 setTimeout()。

在这个例子中，尽管设置的 timeout 值比 bigFunction()执行时间要短，回调函数只会在 bigFunction()结束后再执行。实际上，执行的频率和下面使用 setInterval()的代

码类似：

```
setInterval(bigFunction, 20+10);
```

## 2.3.2 定时器精度

Windows 下的浏览器只有粗粒度的定时器。例如 Windows XP 的底层操作系统定时器提供 15 毫秒精度。这意味着 Date()、setInterval()和 setTimeout()等 JavaScript 函数不能提供可靠的 15 毫秒以下的定时。Google Chrome 是例外之一，它将 Windows 切换到一个准确的定时器模式并提供 1 毫秒的精度。

**提示**

阅读下面的在线文章，可以深入了解 JavaScript 定时器这一主题：

- http://ejohn.org/blog/how-javascript-timers-work/
- http://ejohn.org/blog/javascript-in-chrome/

这里的要点是一个应用程序不应该依赖低于 15 毫秒（约 1/64 秒）的定时器。这个问题严重吗？一般情况下不严重。浏览器中不太可能或不应该运行对时间这么敏感的应用程序。动画也许会比预计的慢一点或快一点，游戏等应用程序中帧率也不是绝对的稳定。如果在一段时间内细致检查这些不精确的累加效果，也许可以看到一定的误差。不过，在通常情况下，比如玩游戏或看菜单特效时，这些误差是察觉不到的。

不过在使用 Date()进行代码性能分析时要小心。下面的例子中，如果执行的代码太快结束的话，将得到不准确的结果：

```
var startTime = new Date().getTime();
/*** Execute some code here that takes less than 15 milliseconds ***/
var endTime = new Date().getTime();
var elapsedTime = endTime - startTime;
```

一个更好的解决方案是在较长的时间段（如 1 秒）内循环执行代码，然后用期间完成的迭代次数来衡量执行速度。

## 2.3.3 保持速度一致

前面的 sprite 实现，具体来说是移动 sprite 的代码，存在一个问题——不同的浏览器下动画和移动的速度（即帧率）不一样。比如 2.8GHz 的 PC、Opera 或 Google Chrome 等浏览器可以在移动 100 个 Sprite 时轻松达到 50FPS（每秒帧数），Firfox 也许能有 30FPS，而 IE8 也许只有 25FPS。如果考虑不同的硬件，帧率的差异会更大。

这对装饰性的动画和特效不是大问题，但游戏等应用程序需要一致的移动速度来保证可玩性。

为在不同的软硬件环境中保持速度一致，必须在 sprite 移动和动画涉及的计算中考虑帧率的不同。具体来说，一个以 30FPS 每帧移动两个像素的 sprite，和一个以 60FPS 每帧移动一个像素的 sprite 看起来速度一样。这两者之间的主要视觉区别是 30FPS sprite 的移动不如 60FPS sprite 的移动流畅。不过至少看起来他们是以同样的速度在屏幕上移动。

为此，必须计算一个时间系数，并在移动和动画代码中使用。表 2-4 显示了一个例子。

表 2-4　　时间系数

| 目标 FPS | 实际 FPS | 时 间 系 数 |
|---|---|---|
| 60 | 30 | 2 |
| 60 | 15 | 4 |
| 30 | 40 | 0.75 |
| 50 | 50 | 1 |

很明显，时间系数=目标 FPS/实际 FPS。

为计算实际 FPS，可以用 JavaScript 的 Date 对象记录当前时间（即开始时间，单位为毫秒位）。在执行所有应用逻辑后再记录时间（结束时间）。下面是代码：

```
actualFPS = 1000 / (endTime - startTime);
```

如果 CPU 负荷过重，帧率会很慢。以 6FPS、每帧 10 像素在屏幕上移动的 sprite 看起来不平稳，肯定不适合游戏。表 2-5 列出了帧率和流畅度的对应关系。

表 2-5　　FPS 和流畅度

| FPS | 流　畅　度 |
|---|---|
| <15 | 相当不平稳 |
| 15～20 | 勉强可接受 |
| 20～30 | 基本流畅 |
| 30～40 | 流畅 |
| 40 以上 | 非常流畅 |

这不是说 10FPS 的低帧率毫无用处。对俄罗斯方块这样的游戏而言，这种帧率也许就可以接受了。

现在我们创建一个 timeInfo 对象，它将提供保持应用速度一致所需的所有功能。它接受一个 goalFPS 参数，即我们想要达到的目标 FPS。如果达不到，函数将调整移动速度使其至少看起来达到了 goalFPS。timeInfo 对象中还提供了其他时间相关的信息。

下面的函数返回一个对象，其中包含 getInfo()方法。getInfo()方法返回一个对象，其属性如表 2-6 所示。

```
var timeInfo = function (goalFPS) {
   var oldTime, paused = true,
       interCount = 0,
       totalFPS = 0;
       totalCoeff = 0;
   return {
       getInfo: function () {
```

表 2-6　　timeInfo.getInfo()返回的对象属性

| 属　性 | 描　述 |
| --- | --- |
| elapsed | 从上次 getInfo()调用开始的毫秒数 |
| coeff | 在移动和动画计算中所用的参数 |
| FPS | 从上次 getInfo()起所达到的 FPS |
| averageFPS | 从第一次 getInfo()起所达到的平均 FPS |
| averageCoeff | 平均参数 |

paused 变量表明这是在应用程序开始或暂停后，getInfo()第一次被调用。它保证在经过一个很长的暂停之后，getInfo()传回的值是良性的，并且不会返回一个非常大的值。

```
    if (paused === true) {
        paused = false;
        oldTime = +new Date();
        return {
            elapsed: 0,
            coeff: 0,
            FPS: 0,
            averageFPS: 0,
            averageCoeff: 0
        };
    }
```

我们通过从上一次 getInfo()中记录的 oldTime，减去新时间，得到经过时间(elapsed time)。然后用经过时间来计算帧率。+new Date()语句等价于 new Date().getTime();：

```
    var newTime = +new Date();  // get time in milliseconds
    var elapsed = newTime - oldTime;
    oldTime = newTime;
    var FPS = 1000 / elapsed;
    iterCount++;
    totalFPS += FPS;
    var coeff = goalFPS / FPS;
    totalCoeff += coeff;
```

然后返回一些有用的信息属性，如表 2-6 所示。

```
        return {
            elapsed: elapsed,
            coeff: goalFPS / FPS,
            FPS: FPS,
            averageFPS: totalFPS / iterCount,
            averageCoeff: totalCoeff / interCount
        };
    },
```

接着我们定义 pause()方法，在暂停应用程序时都应调用此方法。

```
        pause: function () {
            paused = true;
        }
    };
};
```

现在，我们可以在原始的 bouncySprite 和 bouncyBoss 代码中使用 timeInfo 对象了：

```
var bouncySprite = function (params) {
    var x = params.x,
        y = params.y,
        xDir = params.xDir,
        yDir = params.yDir,
        maxX = params.maxX,
        maxY = params.maxY,
        animIndex = 0,
```

```
        that = DHTMLSprite(params);
    that.moveAndDraw = function (tCoeff) {

        x += xDir * tCoeff;
        y += yDir * tCoeff;
        animIndex += xDir > 0 ? 1 * tCoeff : -1 * tCoeff;
        var animIndex2 = (animIndex % 5) >> 0;
        animIndex2 += animIndex2 < 0 ? 5 : 0;

        if ((xDir < 0 && x < 0) || (xDir > 0 && x >= maxX)) {
            xDir = -xDir;
        }
        if ((yDir < 0 && y < 0) || (yDir > 0 && y >= maxY)) {
            yDir = -yDir;
        }
        that.changeImage(animIndex2);
        that.draw(x, y);
    };
    return that;
};
```

moveAndDraw 方法现在接受时间系数作为参数。计算和原来相似，但使用了时间系数。changeImage()函数的参数应该是整数，但因为 animIndex 受时间系数影响，也许不是整数。为此，我们复制一个整数版的 animIndex 为 animIndx2，并传入 changeImage():

```
var bouncyBoss = function (numBouncy, $drawTarget) {
    var bouncys = [],
        timer = timeInfo(40);
    for (var i = 0; i < numBouncy; i++) {
        bouncys.push(bouncySprite({
            images: '/images/cogs.png',
            imagesWidth: 256,
            width: 64,
            height: 64,
            $drawTarget: $drawTarget,
            x: Math.random() * ($drawTarget.width() - 64),
            y: Math.random() * ($drawTarget.height() - 64),
            xDir: Math.random() * 4 - 2,
            yDir: Math.random() * 4 - 2,
            maxX: $drawTarget.width() - 64,
            maxY: $drawTarget.height() - 64
        }));
    }
    var moveAll = function () {
        var timeData = timer.getInfo();
        var len = bouncys.length;
        for (var i = 0; i < len; i++) {
            bouncys[i].moveAndDraw(timeData.coeff);
        }
        setTimeout(moveAll, 10);
    }
    moveAll();
};
```

bouncyBoss 对象现在要创建一个目标 FPS 为 40 的 timeInfo 实例（存在 timer 变量中）。moveAll()在每个迭代调用 timeInfo.getInfo()得到时间系数，并将其传给每个 bouncySprite 实例的 moveAndDraw()方法。注意只需要一个 timeInfo 实例即可，因为每个 bouncySprite 实例可以使用同一个系数。

## 2.4 IE6 背景图像缓存

即使对完全正当的跨浏览器代码，IE6 也不能完全处理好。具体来说，IE6 在缓存背景图像上有问题。当多次访问同一个背景图像时，IE6 从服务器重新获取图像，而不是从本地缓存读取。在用背景图像实现动画的情况下，这显然会极大地影响性能。如果你认为还是有必要兼容 IE6，可以采用这种变通方案：

```
// IE6 background image caching fix.
// Include this JavaScript at the top of your page.
try {
    document.execCommand("BackgroundImageCache", false, true);
} catch(e) {}
```

# 第 3 章

# 滚动

在浏览器中上下或左右滚动页面是很常见的动作。当内容太多而无法在浏览器窗口或特定的浏览器元素中完全查看时，你可以通过滚动在内容上移动视点。本章我们将从纯 CSS 实现到 JavaScript 实现，讨论更图形化和更具创意的滚动方式。

作为一本 JavaScript 图形编程书籍，为什么本书要介绍 CSS 滚动呢？原因之一是我们需要了解纯 CSS 实现的局限性。此外，仅用 CSS 可以加入不错的效果，这些技术是你值得了解的。

## 3.1 纯 CSS 滚动特效

CSS 提供了对滚动内容的一些基本控制，利用它们你可以创建不错的效果。在 CSS 禅意花园网站上的 Retro Theater 中（如图 3-1 所示），电影院屏幕上的网站主要内容，被电影院建筑图像的 div 元素包围。当用户移动浏览器的垂直滚动条时，影院屏幕上的网站主要内容看上去就像垂直移动的片尾字幕一样。

这个特效是如何创建的？以包含座椅的影院底部为例，其 CSS 如下：

```
#extraDiv3
{
    position:fixed !important;
    position:absolute;
    bottom:0;
    left:0;
    width:100%;
    height:30% !important;
    height:110px;
```

```
    min-height:110px;
    max-height:318px;
    background:url('bas.png') no-repeat 50% 0%;
    z-index:4;
}
```


图 3-1　Eric Rogé 在 Retro Theater 中用简单但有效的 CSS，创建了一个不错的滚动字幕特效（http://www.csszengarden.com/?cssfile=202/202.css）

这个特效的关键在 position:fixed 这条 CSS 规则，它保证了 div 元素对窗口的相对位置不变。此处的 div 被固定在窗口的底部。还有一些规则确保元素留在一个高度范围内。

图 3-2 显示了一个更精致的纯 CSS 滚动。在 Silverback 网站，你会注意到 3 层树枝和叶子在页面的顶端。当你调整浏览器窗口大小，这些层将以不同速度移动从而创造出一种三维效果，前景移动最快，越远的层移动越慢。这种效果叫做视差卷轴（parallax scrolling），常用在二维视频游戏和卡通动画中。此处视口是浏览器窗口的整个宽度。

例 3-1 中的代码使用了类似的技术，创建了图 3-3 显示的视察特效。这个特效使用 3 张 256 色 PNG 格式的图像（如图 3-4 所示）。注意最前面的绿草上使用了模糊效果来进一步增强深度感。

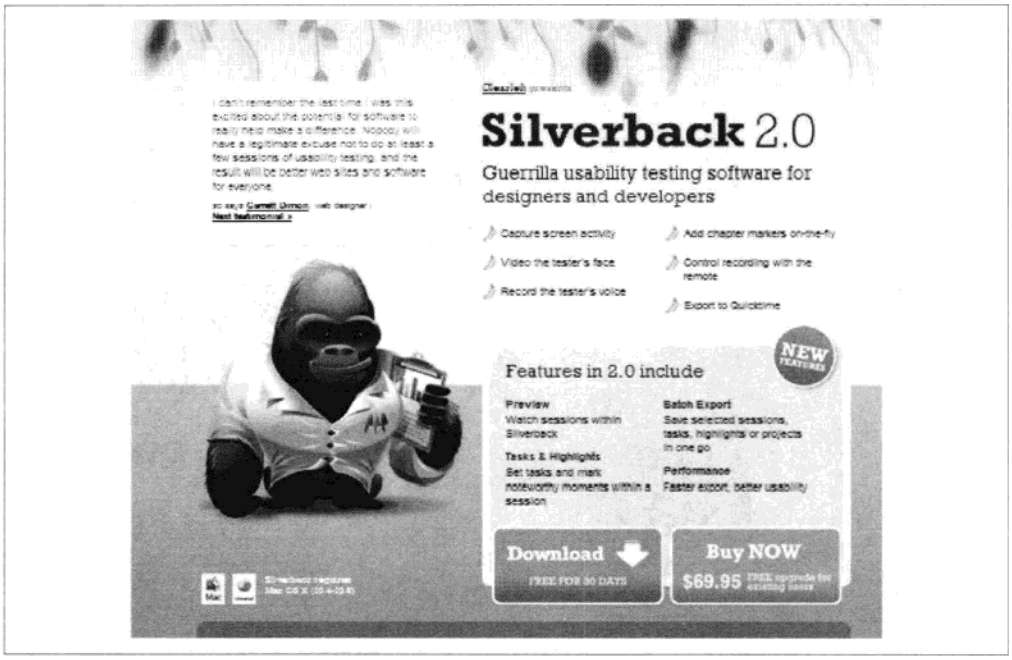



图 3-2　页面顶端的树枝和叶子具有视差卷轴特效（http://silverbackapp.com/）

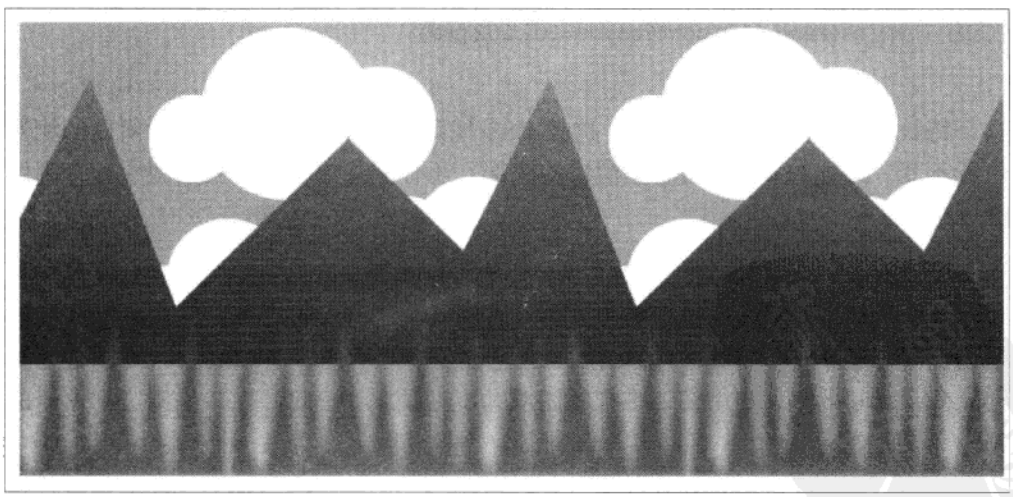


图 3-3　3 层的视差 CSS 特效

**提示**

图 3-4 中使用的图片是带 alpha 通道的 8 位（256 色）PNG 图片。这些图片比 32 位图像占更少内存，而且能在不支持透明 PNG 图像的浏览器，如 IE6 上显示（尽管缺少了 alpha 像素）。

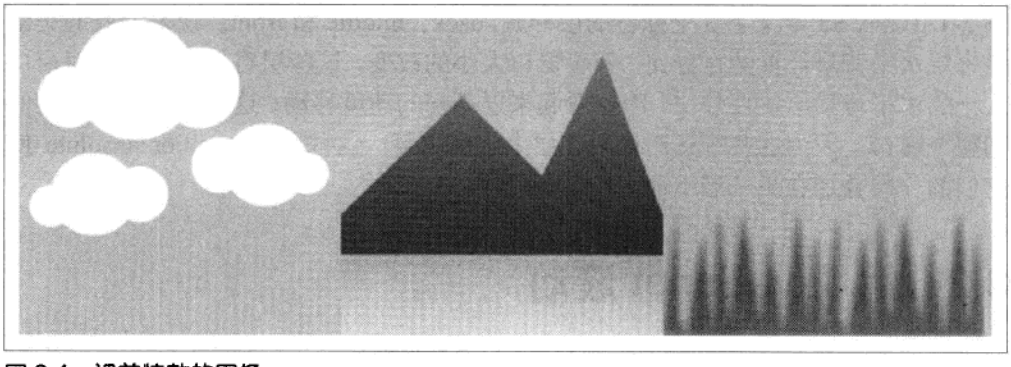


图 3-4　视差特效的图像

例 3-1　CSS 视差卷轴

```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>CSS Parallax</title>
    <style type="text/css">
        body {
            padding:0px;
            margin:0px;
        }

        .layer {
            position:absolute;
            width:100%;
            height:256px;
        }
        #back {
            background: #3BB9FF url(back1.png) 20% 0px;

        }
        #middle{
            background: transparent url(back2.png) 30% 0px ;

        }
        #front{
            background: transparent url(back3.png) 40% 0px;
```

```
        }
    </style>
</head>
<body>
    <div id = "back" class = "layer"></div>
    <div id = "middle" class = "layer"></div>
    <div id = "front" class = "layer"></div>
</body>
</html>
```

例 3-1 中的 CSS 定义了 3 个独特的层样式: back、middle 和 front。视差特效的关键是每层水平背景位置的百分比。随着窗口大小的改变，这些层将保持它们相对窗口大小的水平背景位置百分比，因此看起来以不同的速度移动。这些层的宽度将延伸到整个窗口，因为这些背景图像要在层上水平重复。这些层用 position:absolute 使它们的一层叠加在另一层上面。

## 3.2 用 JavaScript 滚动

尽管前面描述的 CSS 滚动特效很不错，但 CSS 实现有缺乏控制的问题：视差特效只有在窗口大小改变时才出现，不能保证用户都能看到这个特效。而采用 JavaScript，就没有特效发生方式和时间的限制了。本节中，我们将学习两类 JavaScript 滚动技术：背景图像滚动和更复杂的基于块的图像滚动。

### 3.2.1 背景图像滚动

下面我们重新创建了图 3-3 的 CSS 特效，但这次鼠标移动控制了滚动的方向和速度。

当鼠标移动到页面左侧或右侧，就会朝这一方向加速滚动；当鼠标在页面中间时，则放慢滚动速度；当鼠标离开页面时，则完全停止滚动。

例 3-1 CSS 滚动代码中的背景图像位置是以浏览器窗口大小的百分比设置。例 3-2 中的背景图像位置则以像素位置操纵。

**提示**

为方便阅读，此例的代码中使用了 jQuery。

例 3-2 简单的 JavaScript 滚动

```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

```
    <title>CSS Parallax</title>
    <style type="text/css">
        body {
            padding:0px;
            margin:0px;
        }
    .layer {
        position:absolute;
        height:256px;
        width:100%;
    }

    #back {
        background: #3BB9FF url(back1.png);

    }
    #middle{
        background: transparent url(back2.png);

    }
    #front{
        background: transparent url(back3.png);

    }
</style>
<script type="text/javascript"
    src="http://ajax.googleapis.com/ajax/libs/jquery/1.5.0/jquery.min.js">
</script>

<script type="text/javascript">
    $(function () {
        var speed = 0,
            $back = $('#back'),     // Initial speed.
            $middle = $('#middle'), // Cache layers as jQuery objects.
            $front = $('#front'),
            xPos = 0,               // Initial x position of background images.
            $win = $(window);       // Cache jQuery reference to window.

        // Respond to mousemove events.
        $(document).mousemove(function (e) {
            var halfWidth = $win.width()/2;
            // Calculate speed based on mouse position.
            // 0 (center of screen) to 1 at edges.
            speed = e.pageX - halfWidth;
            speed /= halfWidth;
        });

        // Kill speed on mouseout.
        $(document).mouseout(function (e) {
            speed = 0;
        });

        // Every 30ms, update each layer's background image position.
        // The two front layers use a scaled-up x position to
        // create the parallax effect.
```

```
        setInterval(function () {

            // Update the background position variable.
            xPos += speed;
                // Apply it to the layers' background image positions,
                // scaled up for the front two layers so they move quicker
                // than the farthest layer.
                $back.css({
                    backgroundPosition: xPos + 'px 0px'
                });
                $middle.css({
                    backgroundPosition: (xPos * 2) + 'px 0px'
                });
                $front.css({
                    backgroundPosition: (xPos * 3) + 'px 0px'
                });


            }, 30);

        });
    </script>

</head>
<body>
    <div id = "back" class = "layer"></div>
    <div id = "middle" class = "layer"></div>
    <div id = "front" class = "layer"></div>
</body>
</html>
```

下面是例 3-2 中 JavaScript 滚动代码的基本工作原理：

- 当 mousemove事件发生时，根据鼠标位置计算速度。
- 当 mouseout 事件发生时，将速度设为 0。
- 每 30 毫秒将计算得到的速度值加到 x 坐标的位置变量(xPos)，将缩放后的 xPos 应用到每层水平背景图像坐标。每层 x 位置的缩放比率分别是 1、2 和 3。

## 3.2.2 基于块的图像滚动

上述滚动例子的一个缺点是图像会重复地平铺在浏览器窗口中。当滚动背景图像时，很快就发现它们缺少变化。解决方法是使用较大的图片，使得必须滚动较多才会出现重复图像。但这样会出现另一个问题，如果内容面积较大，图像大小将很快失去控制。假设需要的内容区域长宽各 2 048 像素（大约是上网本屏幕的两倍）。那每一层就需要 400 万像素的图像。如果我们需要长、宽各 10 万像素的 3 个层呢？很明显，当你需要较大的区域来滚动时，基于大图像的方法是不实际的。

一个更有效的方式是使用基于块的图像滚动。内容区域上将重复使用一组大小一致（如长、宽各 64 像素）的图像块（image tile）。图像区域实际上变成了一个由图像块组成的均匀网格，或“地图（map）”。借助合适的排列和绘制，这些块组成的地图会给人带来一张大图片的感觉。每个块通过一个索引来引用，地图的定义是这些索引组成的数组。因此如果使用长、宽 64 像素的图像块，一个长、宽各 2 048 像素的地图需要存储 32×32（即 1 024）个块索引。这种通过重复小元素来创建大整体的思想，和文本文件的机制很相似：一个文本文件以字符索引（如 ASCII 码）保存，而不是字符的位图。

有一种直接的方法可以实现这种技术，它是创建一个 div 元素（作为手柄）并附上 image 元素，每个 image 元素代表地图中的一个块。通过在一个更小的视点元素中移动手柄元素，达到滚动效果。这种方法行的通，但有如下问题：

- 你必须在 DOM 中插入大量块元素（image 元素）。一个大的地图也许需要几千个块。由于浏览器要处理所有的 image 元素（即使不同时可见），因此庞大的 DOM 也就意味着内存消耗变高和性能变差。
- 每个独立的 image 元素块都需要从网络载入自己的位图。当独立的块数比较少时，这不是问题；但对上百个块而言，这会导致页面载入变慢。

第二个问题容易解决。如同第 2 章中介绍的 DHTML Sprite，我们可以使用 div 元素而不是 image 元素。div 可以引用一个大的位图（或块集合）中的一小部分作为它们的背景图像。这可以降低从网络上载入的图块数，而通过改变 CSS 背景位置属性来改变块内的图像。

对第一个问题，我们可以采用下面描述的 snapping 技术来极大地减少 DOM 中块元素的数目。

### 1．Snapping...

我们可以将需要的块数降低为在视点中用到的块数。图 3-5 显示了 640×384 像素的视点窗口。背后的网格代表了地图中可见的部分，由 64 像素长、宽的块组成。视点中最多可以显示 11×7，共 77 个块。注意虽然地图很大，但这个值和地图的大小无关。下面的公式说明了是怎么计算得到 11 和 7 的（axisSize 和 tileWidth 以像素为单位）：

```
numTilesAxis = Math.ceil((axisSize + tileWidth) / tileWidth);
```

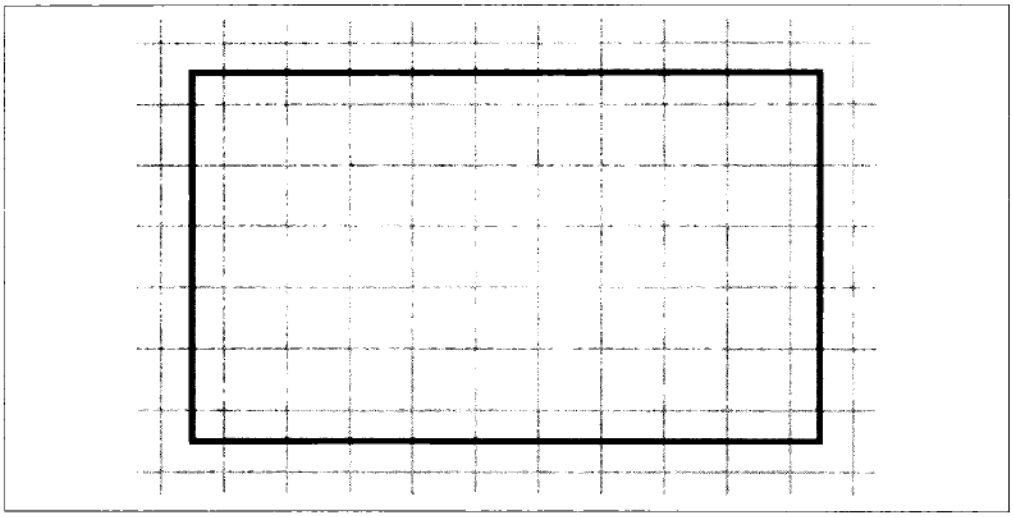


图 3-5　640 × 384 像素的视点内最多能显示 77 个 64 像素长宽的块

我们需要一个方法，使得创建和操作的块数等于视点中最多可显示的块数，而和地图的大小无关。

想象向右滚动地图，77 个块将向左移动。如果最左边的块被移动到了视点之外呢？由于我们没有创建地图右边延伸部分的图像块，视点的右边就没有元素可以显示了，因此将显示空白区域。这不是我们想要的效果。解决办法是将所有块重新 snap 到右边，使滚动的最大量不超过一个块。这个方法可以同样应用到其他滚动方向上。

**提示**

什么是地图滚动位置？想象整个地图是一个大虚拟位图。地图滚动位置指的是视点的左上角在虚拟位图中的像素位置。

如何计算 snap 位置呢？用当前的地图滚动位置除以块宽度并对余数取负：

```
snapPos =  -(scrollPosition % tileWidth);
```

如果块宽度是 64 像素而水平滚动位置每次增加 8 像素（从 0 开始），snap 位置将如下重复：

0, −8, −16, −24, −32, −40, −48, −56, 0, −8, −16, −24, −32, −40, −48, −56,等等

包含所有块的句柄元素的 left 或 top 位置（取决于是垂直或水平滚动），将使用计算

得到的 snap 位置。从左到右或从上到下的移动都可采用同样的计算。snap 位置必须分别针对水平坐标轴和垂直坐标轴计算。

你也许已经意识到仅将所有块 snap 到右边，将重复显示地图的同一部分，因此出现跳动效果。所以还需根据滚动位置改变块的位图。为选择正确的位图，你必须从地图中提取出合适的块索引（别忘了地图只是块索引的数组）。假设 yPos、xPos、mapWidth 都以块为单位，可以如下计算块索引：

```
index = map[(yPos * mapWidth) + xPos];
```

### 2. Wrapping

我们可以再加入一个很有用的特性：无限 wrap 一个地图。没有 wrapping，当地图在视点内按一个方向滚动，最终会看到地图的边缘。因为在这个方向没有可以显示的，所以空白会出现；换句话说，它到达了地图的尽头。

用 wrapping 技术后，在显示右边的边缘块后，我们将接着显示左边的边缘块。好比在遍历数组时，从 A[n−1]跳到 A[0]，这使得滚动可以无限继续下去，如同一个无限大的地图。如果这两个边缘块之间能自然地连上，那看起来就会非常自然而感觉不到边缘的存在。

地图 wrapping 技术特别适用于无限大的背景特效，如蓝天白云、璀璨星辰、连绵山脉等。

### 3. 提高速度

尽管我们已经用 snapping 方法极大减少了块的数目，一个 640 × 384 像素的视点内的 3 层视差卷轴还是会达到 3 × 77，即 231 块。我们可以通过增大块的大小来进一步降低块的个数。如果其中一层只是有少许云朵的天空，我们可以使用 128 像素的块，这样就可以将块数从 77 降低到 24。

为保证帧率，滚动时对这些块的操作应该降低到最小：尽可能预先计算，将滚动时需要进行的运算降为最小。优化滚动代码时需要注意：

- 少进行函数调用，尤其是 jQuery，因为 jQuery 函数会在幕后做很多事情。
- 仅进行简单循环和算术。
- 将每个块的样式属性的引用存在数组中，方便快速改变诸如背景位置等属性。

- 将每个块索引的背景位置作为字符串存在数组中：'0px 0px'、'0px 64px'，等等。你可以将这些字符串一次直接存入背景位置属性，而不是分别更新 left 和 top 属性。
- 因为你只在设置视点期间一次加入可见的块，因此在滚动时浏览器不会进行页面内容回流（reflow）或其他耗费时间的动作。

**提示**

浏览器回流指当页面流变动时，重新计算所有 DOM 元素的位置和大小。绝对或固定位置的元素是在页面流之外的，操作它们不会引起回流。

### 4. 块滚动代码

块滚动代码可以分为两个主要部分：

- 初始化和预计算；
- 绘制。

一个视点内的滚动由 tileScroller 实例处理。设置参数将作为一个对象传入，其属性如表 3-1 所示。

表 3-1　　传入 tileScroller 的参数

| 属　性 | 描　述 |
|---|---|
| $viewport | DOM 中的视点元素 |
| tileWidth | 块的宽度（像素为单位） |
| tileHeight | 块的高度（像素为单位） |
| wrapX | 是否水平 wrap 地图 |
| wrapY | 是否垂直 wrap 地图 |
| mapWidth | 地图的宽度（块为单位） |
| mapHeight | 地图的高度（块为单位） |
| image | 块集合图像的 URL，包含所有块图像 |
| imageWidth | 块集合图像的宽度（像素为单位） |
| imageHeight | 块集合图像的高度（像素为单位） |
| map | 块索引数组 |

图 3-6 显示了例 3-3 中滚动代码的效果。

例 3-3　3 层基于块的滚动

```
// One instance of tileScroller is required for each viewport.
var tileScroller = function (params) {

    var that = {},
        $viewport = params.$viewport,
        // Calculate maximum number of tiles that can be displayed in viewport.
        tilesAcross = Math.ceil(($viewport.innerWidth()
            + params.tileWidth) / params.tileWidth),
        tilesDown = Math.ceil(($viewport.innerHeight()
            + params.tileHeight) / params.tileHeight),

        // Create a handle element that all tiles will be attached to.
        // If this element is moved, so all the attached tiles will move.
        html = '<div class="handle" style="position:absolute;">',
        left = 0,    // General counters.
        top = 0,
        tiles = [],    // Stores a reference to each tile's style property.
        tileBackPos = [], // Stores the background position offset for each tile.

        mapWidthPixels = params.mapWidth * params.tileWidth,
        mapHeightPixels = params.mapHeight * params.tileHeight,
        handle, i; // General counter.

    // Attach all the tiles to the handle. This is done by creating
    // a big DOM string containing all the tiles and attaching it
    // in one jQuery call. This is faster than attaching each one individually.
    for (top = 0; top < tilesDown; top++) {
        for (left = 0; left < tilesAcross; left++) {
            html += '<div class="tile" style="position:absolute;' +
            'background-image:url(\'' + params.image + '\');' +
            'width:' + params.tileWidth + 'px;' +
            'height:' + params.tileHeight + 'px;' +
            'background-position: 0px 0px;' +
            'left:' + (left * params.tileWidth) + 'px;' +
            'top:' + (top * params.tileHeight) + 'px;' + '"/>';
        }
    }
    html += '</div>';
    // Put the whole lot in the viewport.
    $viewport.html(html);

    // Get a reference to the handle DOM element.
    handle = $('.handle', $viewport)[0];

    // For each tile in the viewport, store a reference to its
    // css style attribute for speed.
    // This will be updated with the tile's visibility status
    // when scrolling later on.
    for (i = 0; i < tilesAcross * tilesDown; i++) {
        tiles.push($('.tile', $viewport)[i].style);
    }
```

```
// For each tile image in the large bitmap, calculate and store the
// the pixel offsets to be used for the tiles' background image.
// This is quicker than calculating when updating later.
tileBackPos.push('0px 0px'); // Tile zero - special 'hidden' tile.
for (top = 0; top < params.imageHeight; top += params.tileHeight) {
    for (left = 0; left < params.imageWidth; left += params.tileWidth) {
        tileBackPos.push(-left + 'px ' + -top + 'px');
    }
}

// Useful public variables.
that.mapWidthPixels = mapWidthPixels;
that.mapHeightPixels = mapHeightPixels;


// The 'draw' function.
that.draw = function (scrollX, scrollY) {
    // If wrapping, transform start positions to valid positive
    // positions within the dimensions of the map.
    // This makes the wrapping code simpler later on.
    var wrapX = params.wrapX,
        wrapY = params.wrapY;
    if (wrapX) {
        scrollX = (scrollX % mapWidthPixels);
        if (scrollX < 0) {
            scrollX += mapWidthPixels;
        }
    }
    if (wrapY) {
        scrollY = (scrollY % mapHeightPixels);
        if (scrollY < 0) {
            scrollY += mapHeightPixels;
        }
    }

    var xoff = -(scrollX % params.tileWidth),
        yoff = -(scrollY % params.tileHeight);
    // >> 0 alternative to math.floor. Number changes from a float to an int.
    handle.style.left = (xoff >> 0) + 'px';
    handle.style.top = (yoff >> 0) + 'px';

    // Convert pixel scroll positions to tile units.
    scrollX = (scrollX / params.tileWidth) >> 0;
    scrollY = (scrollY / params.tileHeight) >> 0;


    var map = params.map,
        sx, sy = scrollY,       // Copies of scrollX & Y positions (tile units).
        countAcross, countDown, // Loop counts for drawing tiles.
        mapWidth = params.mapWidth,     // Copy of map width (tile units).
        mapHeight = params.mapHeight,   // Copy of map height (tile units).
        i,              // General counter.
        tileInView = 0, // Start with top-left tile in viewport.

        tileIndex, // Tile index number taken from map.
```

```
        mapRow;
    // Main drawing loop.
        for (countDown = tilesDown; countDown; countDown--) {
            // Wrap vertically?
            if (wrapY) {
                if (sy >= mapHeight) {
                    sy -= mapHeight;
                }
            } else
            // Otherwise, clip vertically (just make the whole row blank).
            if (sy < 0 || sy >= mapHeight) {
                for (i = tilesW; i; i--) {
                    tiles[tileInView++].visibility = 'hidden';
                }
                sy++;
                continue;
            }
            // Draw a row.
            sx = scrollX,
            mapRow = sy * mapWidth;
            for (countAcross = tilesAcross; countAcross; countAcross--) {
                // Wrap horizontally?
                if (wrapX) {
                    if (sx >= mapWidth) {
                        sx -= mapWidth;
                    }
                } else
                // Or clipping horizontally?
                if (sx < 0 || sx >= mapWidth) {
                    tiles[tileInView++].visibility = 'hidden';
                    sx++;
                    continue;
                }
                // Get tile index no.
                tileIndex = map[mapRows + sx];
                sx++;
                // If tile index nonzero, then 'draw' it;
                if (tileIndex) {
                    tiles[tileInView].visibility = 'visible';
                    tiles[tileInView++].backgroundPosition = tileBackPos[tileIndex];
                }
                // otherwise, hide it.
                else {
                    tiles[tileInView++].visibility = 'hidden';
                }
            }
            sy++;
        }
    };
    return that;
};
```

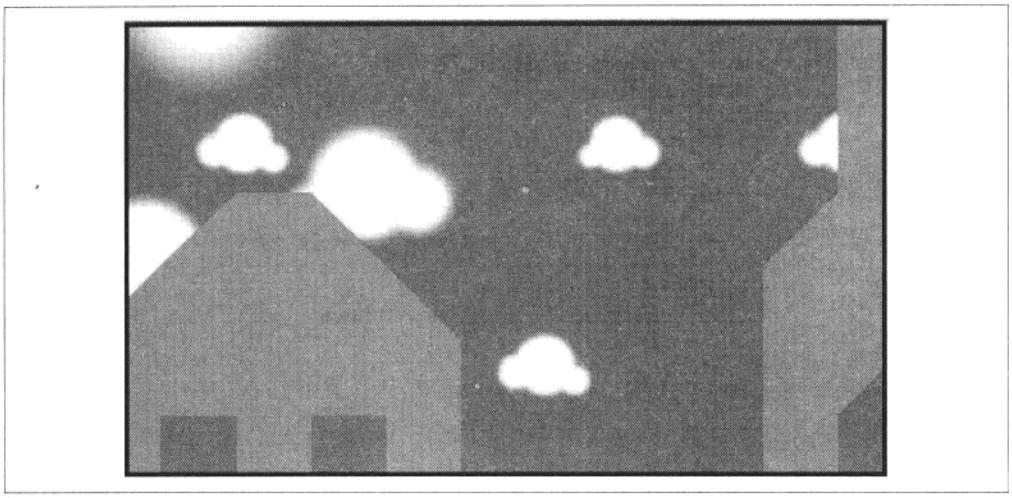


图 3-6　基于块滚动的效果

## 5．用 Tiled 创建块地图

手动输入块的索引费时、费力而且容易出错。幸运的是，有一些块地图编辑器可以提高块地图设计的效率。由 Thorbjørn Lindeijer 等人贡献的开源跨平台编辑器 Tiled（http://www.mapeditor.org/；如图 3-7 所示）可以说是其中最好的。它可以在 Windows、Mac 和 Linux 等操作系统上运行。

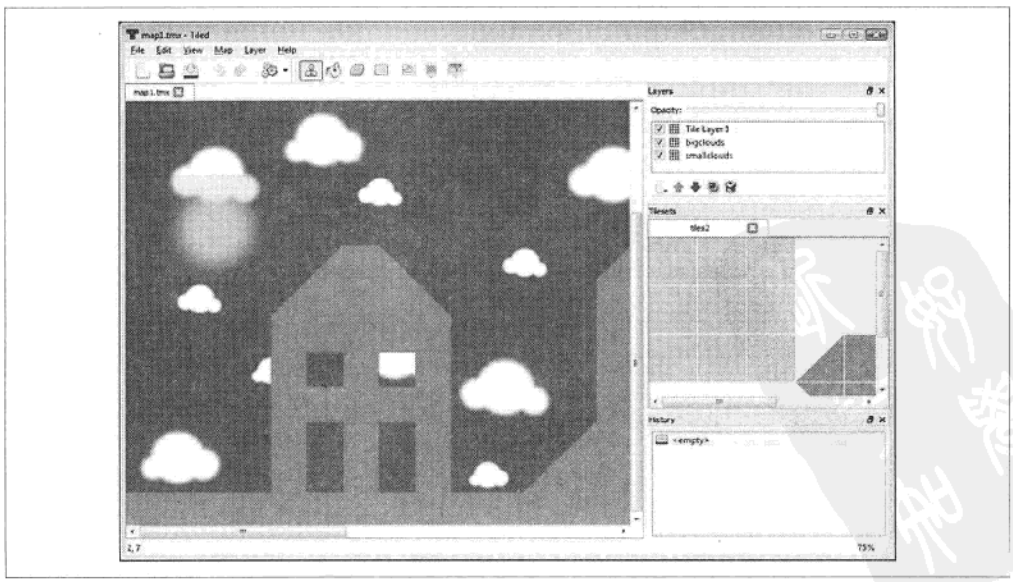



图 3-7　Tiled 编辑器

Tiled 维基提供了用 Tiled 创建地图的介绍（http://sourceforge.net/apps/mediawiki/tiled/index.php?title=Creating_a_simple_map_with_Tiled）。

Tiled 可以使你创建多层地图。在下面的例子中，图 3-6 中的时差卷轴特效包括了 3 个层：

- 小云朵；
- 大云朵；
- 前景。

**提示**

尽管下面的例子中只使用一个块集合图像（tileset image），但 Tiled 可以支持多个块集合图像的管理。这个功能不仅可以用来更好地组织图像，而且可以将颜色接近的块放在一起，并将其存在 8 位（256 色）PNG 而不是 32 位 PNG 中。由于块之间颜色接近，颜色数减少的影响很小，因此可以在不太影响效果的情况下减少带宽。

Tiles 可以创建长方形网格地图（如图 3-7 所示）或斜视角地图（isometric map，如图 3-8 所示）。选择 New 创建新的地图，弹出图 3-9 中的对话框。在对话框中设置地图类型、地图宽和高（以块为单位）、块的宽和高（以像素为单位）。

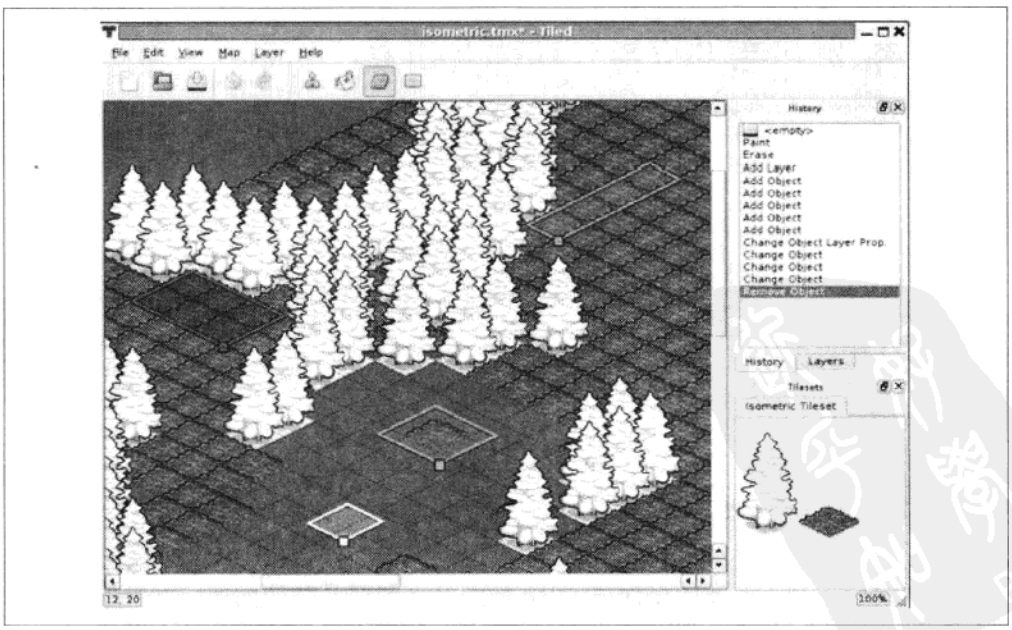



图 3-8　Tiled 中的斜视角地图

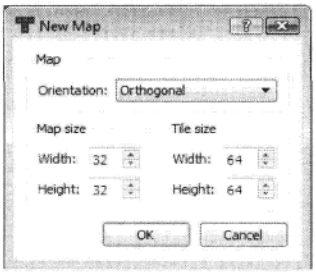



图 3-9　地图设置对话框

单击 OK，Tiled 将自动创建一个叫做“Tile Layer 1”的地图层。在 Layers panel 中双击名字可以对层进行重命名。加入更多层可以选择 Layer→Add Tile Layer。

现在这些层还毫无用处，因为没有块集合图像可用。选择 Map→New Tile Set，将弹出图 3-10 中的对话框。给这个块集合选择一个名字并在浏览本地目录找到一个块集合图像。如果在块集合图像中没有使用特殊颜色来代表透明区域的话，不要选中“Use transparent color”复选框。

当块集合图像载入后，你可以从块集合图像中选择不同的块并将它们作为笔刷在地图主要区域放置块。当多个块组成地图中的一个元素时，你可以在块集合上点击和拖拽，以选择具有多个块的矩形图案作为笔刷，加快地图构建。比如一个建筑物由 8 个块组成，你可以在块集合中一下选择所有 8 个块。

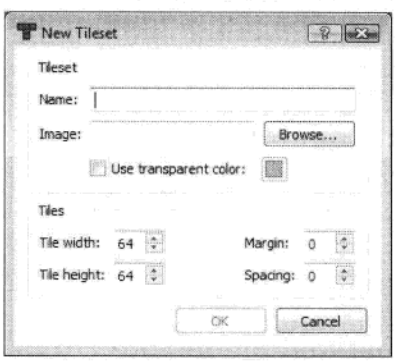



图 3-10　块集合设置对话框

创建好地图后，剩下要做的是保存 Tiled 地图文件并将其和块图像一起复制到服务器。下节将讲解如何在代码中解析和使用 Tiled 地图文件，完成块滚动效果。

### 6. Tiled 文件格式

尽管 Tiled 文件后缀是.tmx，但文件格式是 XML（见例 3-4）。XML 格式文件有一些优势：

- 可读性强，便于查看数据；
- 可以用文本编辑器编辑；
- 绝大部分编程语言和库都有 XML 解析工具。

例 3-4　Tiled 文件格式

```
<?xml version="1.0" encoding="UTF-8" ?>
<map version="1.0" orientation="orthogonal" width="32" height="16" tilewidth="64"
tileheight="64">
    <tileset firstgid="1" name="tiles2" tilewidth="64" tileheight="64">
        <image source="tiles2.png" width="512" height="320" />
    </tileset>
    <layer name="smallclouds" width="32" height="16">
        <data encoding="csv">
            0,1,2,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,0,0,0,0,0,0,0,7,8,0, ...
            <!-- Rest of data deliberately omitted -->
        </data>
    </layer>
    <layer name="bigclouds" width="32" height="16">
        <data encoding="csv">
            0,0,0,0,0,0,0,0,4,5,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, ...
            <!-- Rest of data deliberately omitted -->
        </data>
    </layer>
    <layer name="foreground" width="32" height="16">
        <data encoding="csv">
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,20,22,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, ...
            <!-- Rest of data deliberately omitted -->
        </data>
    </layer>
</map>
```

Tiled 可以将地图数据（XML 文件中的<data>标签）存为下列格式：

- Base64 编码
- Base64 gzip（压缩）
- Base64 zlib（压缩）
- CSV（逗号分割）

jQuery 可以毫不费力地解析 XML 数据，如例 3-5 的 loadMap()函数中使用的 jQuery

ajax()命令。

对 JavaScript 来说，使用 CSV 是最简单的，因为你可以用 split()函数直接将数据值放入 JavaScript 数组。使用其他的格式则需要加入解压缩和 base64 解码的代码。当地图非常大时压缩是值得的——CSV 数据不是特别紧凑，但对这些例子来说足够了。

为将地图文件存为 CSV，在 Tiled 中设置：

Edit→Preference→Saving and Loading→Store the tile layer data as→CSV

在 JavaScript 中解析 XML 文件需要注意的一点是，原始数据是字符串，而不是数值。这意味着从 XML 中取出两个数值并相加，将得到下面的结果：

```
var val1 = '64', val2 = '64';// String values as stored in XML file.
var total = val1 + val2;     // = string '6464', not number 128.
```

换句话说，我们进行的是字符串连接，而不是数值相加。

为保证 XML 中取出的值是作为数值，而不是字符串使用，在它们前面加上加号（+）：

```
var val1 = '64', val2 = '64';// Values as stored in XML file.
var total = +val1 + +val2;   // = number 128 as desired.
```

这是下面代码的简写版本：

```
var val1 = '64', val2 = '64';// Values as stored in XML file.
var total = parseInt(val1) + parseInt(val2);   // = number 128 as desired.
```

例 3-5 的 loadMap()函数用 ajax()载入和解析了一个多层的 Tiled 地图文件。在处理了 Tiled 文件中的参数之后，它初始化了 tileScroller 对象并在 DOM 中创建了所需的视点。一旦所有图层和视点设置好之后，执行一个回调函数。这个回调函数一般包含每个视点中滚动控制的代码。

例 3-5　通过 ajax()载入一个 Tiled 地图

```
var loadMap = function(xmlFile,$viewports,callback) {
    var tileScrollers = []; // Array of tileScroller instances for each viewport.
    $.ajax({
        type: "GET",
        url: xmlFile,
        dataType: "xml",
        // Success function called when map has loaded.
        success: function(xml) {
            // Get references to image and map information.
            var $imageInfo = $(xml).find('image'),
                $mapInfo = $(xml).find('map'),
                i;
            // For each layer, create a tileScroller object.
```

```
            $(xml).find('layer').each(function() {
                // Setup parameters to pass to tileScroller.
                // The + operator before some values is to ensure
                // they are treated as numerics instead of strings.
                var params = {
                    tileWidth: +$mapInfo.attr('tilewidth'),
                    tileHeight:+$mapInfo.attr('tileheight'),
                    wrapX:true,
                    wrapY:true,
                    mapWidth:+$mapInfo.attr('width'),
                    mapHeight:+$mapInfo.attr('height'),
                    image:$imageInfo.attr('source'),
                    imageWidth: +$imageInfo.attr('width'),
                    imageHeight: +$imageInfo.attr('height')
                },
                    // Get the actual map data as an array of strings.
                    mapText = $(this).find('data').text().split(','),
                    // Create a viewport.
                    $viewport = $('<div>');
                    $viewport.attr({
                        'id':$(this).attr('name')
                    }).css({
                        'width':'100%',
                        'height':'100%',
                        'position':'absolute',
                        'overflow':'hidden'
                    });
                // Attach viewport to viewports wrapper.
                $viewports.append($viewport);
                // Store viewport in parameters.
                params.$viewport = $viewport;
                // Create a map array and store in parameters.
                params.map = [];
                // Convert previous text array map into numeric array.
                for(i=0;i<mapText.length;i++) {
                    params.map.push(+mapText[i]);
                }
                // Create a tileScroller and save reference.
                tileScrollers.push( tileScroller(params) );
            });
            // Call callback when map loaded, passing array
            // of tileScrollers as parameter.
            callback(tileScrollers);
        }
    });
};
```

### 7. 块滚动的页面布局

块滚动的实际 HTML 页面（见例 3-6）包含对 loadMap()的调用，它初始化了所有视点，并在鼠标移动时控制不同层以不同速度滚动（视差特效）。最后用一个 30 毫秒的 setInterval 调用完成这个效果。注意页面代码顶部的 viewpoints div 的 CSS，

定义了视点的大小；对应页面代码底部的 viewpoints div 元素。loadMap()中创建的所有视点都将插入这个 div。

例 3-6　Tile 滚动页面代码

```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>JavaScript Tile Map Scrolling</title>
    <style type="text/css">
        body {
            padding:0px;
            margin:0px;
        }
        #viewports {
            position:absolute;
            border:4px solid #000;
            background-color:#3090C7;
            width:640px;
            height:384px;
        }

    </style>
    <script type="text/javascript"
        src="http://ajax.googleapis.com/ajax/libs/jquery/1.5.0/jquery.min.js">
    </script>

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


            var tileScroller = function (params) {
                /*** CODE REMOVED FOR CONCISENESS ***/
            };

            var loadMap = function(xmlFile,$viewports,callback) {
                /*** CODE REMOVED FOR CONCISENESS ***/
            };

        // Call the loadMap function. The callback passed
        // is a function that scrolls each viewport at different speeds according
        // to mouse movement.
        loadMap("map1.tmx", $('#viewports'), function (tileScrollers) {

            var ts1 = tileScrollers[0],  // Get the three tileScrollers.
                ts2 = tileScrollers[1],
                ts3 = tileScrollers[2],
                scrollX = 0,             // Current scroll position.
                scrollY = 0,
                xSpeed = 0,              // Current scroll speed.
                ySpeed = 0,
                // Width and height of viewports.
                viewWidth = $('#viewports').innerWidth(),
                viewHeight = $('#viewports').innerHeight();
```

```
            // As mouse is moved around viewports,
            // calculate a speed to scroll by.
            $('#viewports').mousemove(function (ev) {
                xSpeed = ev.clientX - (viewWidth / 2);
                xSpeed /= (viewWidth / 2);
                xSpeed *= 10;
                ySpeed = ev.clientY - (viewHeight / 2);
                ySpeed /= (viewHeight / 2);
                ySpeed *= 10;
            });
            // Every 30 milliseconds, update the scroll positions
            // for the three tileScrollers.
            setInterval(function () {
                // Each tileScroller is given a different scroll position
                // for a parralax effect.
                ts1.draw(scrollX / 3, scrollY / 3);
                ts2.draw(scrollX / 2, scrollY / 2);
                ts3.draw(scrollX, scrollY);
                // Update scroll position.
                scrollX += xSpeed;
                scrollY += ySpeed;
                // Stop scrolling at edges of map.
                // This code can be removed to test the wrapping.
                if (scrollX < 0) {
                    scrollX = 0;
                }
                if (scrollX > ts3.mapWidthPixels - viewWidth) {
                    scrollX = ts3.mapWidthPixels - viewWidth;
                }
                if (scrollY < 0) {
                    scrollY = 0;
                }
                if (scrollY > ts3.mapHeightPixels - viewHeight) {
                    scrollY = ts3.mapHeightPixels - viewHeight;
                }
            }, 30);
        });

    });
    </script>

</head>
<body>
    <!-- This div will contain the three viewports -->
    <div id="viewports"></div>
</body>
</html>
```

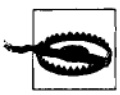
# 第 4 章

# 高级 UI

图形编程并不仅仅是生成漂亮的图片，通过有吸引力和有趣的交互元素，你可以使用户和页面更有效地交互。在本章中，我们将探索如何超越 HTML 表单元素的局限性，使用库或者自定义的元素提高用户体验。

## 4.1 HTML5 表单

HTML5 引入了一些功能更强的新表单，给 Web 设计人员减轻了表单验证和组件绘制等负担。理论上，这些特性无需额外客户端编程，就能提供更丰富的浏览体验。



**警告**

在客户端验证固然方便，但也为伪装表单发送非法数据到服务器提供了便利。服务器端还是应该验证所有表单输入，来避免处理恶意或垃圾数据。

新 HTML5 输入包括下列类型：

- email
- tel
- url
- number
- range
- search

- color
- date
- week
- month
- time
- datetime
- datetime-local

实现这些新输入类型和实现传统输入类型（如 hidden、text 或 password）一样：

```
<input type='date'>
```

HTML5 在表单元素的跨浏览器和丰富性上有了不小进步，但仍有一些局限性：

- 浏览器支持还不够好，比如有些不支持的元素被替换为普通的<input>标签；
- 不同浏览器中的外观和行为不一致，不利于网站在不同浏览器下的体验一致性。

图 4-1 显示了在 Opera、Chrome 和 Firefox 下的 HTML5 date 输入元素。Opera 显示了一个完整的日历，Chrome 只有上下按钮来调整日期，而 Firefox 仅显示了一个普通输入。


图 4-1　Opera、Chrome 和 Firefox 下的 HTML5 date 输入元素（从上至下）

Opera 看来做了最好的尝试，但最终效果不是那么令人振奋，而 Chrome 的效果看起来是经过深思熟虑的。这种不一致性令人沮丧，因此在 HTML5 输入在不同浏览器上取得一致效果之前，我们必须依赖 JavaScript 达到我们要的效果。这也不是件坏事，因为 JavaScript 能提供了更大的创造空间、更多的可能性。

下节将介绍流行的 JavaScript UI 库，你可以在此基础上创建功能极强、体验不弱于传统桌面应用的 Web 应用。尽管不可能将用户数据存于远程服务器，一些新浏览器如 Chrome 提供了本地数据库存储功能。本地存储的支持尚未标准化，你可以关注此领域的最新发展。

## 4.2 使用 JavaScript UI 库

由于当前 HTML5 的新输入元素也许还不够稳定和精细，我们可以使用 JavaScript 来提供迷人且兼容的效果。有两种实现途径：使用已有的 JavaScript UI 库，或从头创建界面组件。

本节中我们将简单介绍两个最流行的 JavaScript UI 库：jQuery UI 和 Ext JS。有人认为这两个库是竞争关系，但仔细研究你会发现它们在预期用途上有很大差别。比如，如果你要开发一个电子商务 Web 应用，你会发现更轻量级的 jQuery UI 适合面向客户的前端，而 Ext JS 适合复杂的、面向管理员的后台。两者之间的一个重要区别是下载文件（包括例子和文档）的大小：jQuery UI 大约 1MB，而 Ext JS 有 13MB，这反映了两者不同的发展方向。

### 4.2.1 使用 jQuery UI 增强 Web 界面

jQuery UI（http://www.jqueryui.com）在 jQuery 之上提供了额外的 UI 元素。jQuery 的用户应该认真考虑 jQuery UI，因为 jQuery 本身的代码已经被页面载入了。

图 4-2 显示了在一个主题下的不同 jQuery UI 元素，共有 24 个主题可供选择。这个例子中用的是 Start 主题。仅需少许改动，所有这些元素就可以在几乎所有浏览器上正确且一致地显示。

jQuery UI 当前支持下列 UI 元素：

- Accordion
- Autocomplete

- Button
- Datepicker
- Dialog
- Progressbar

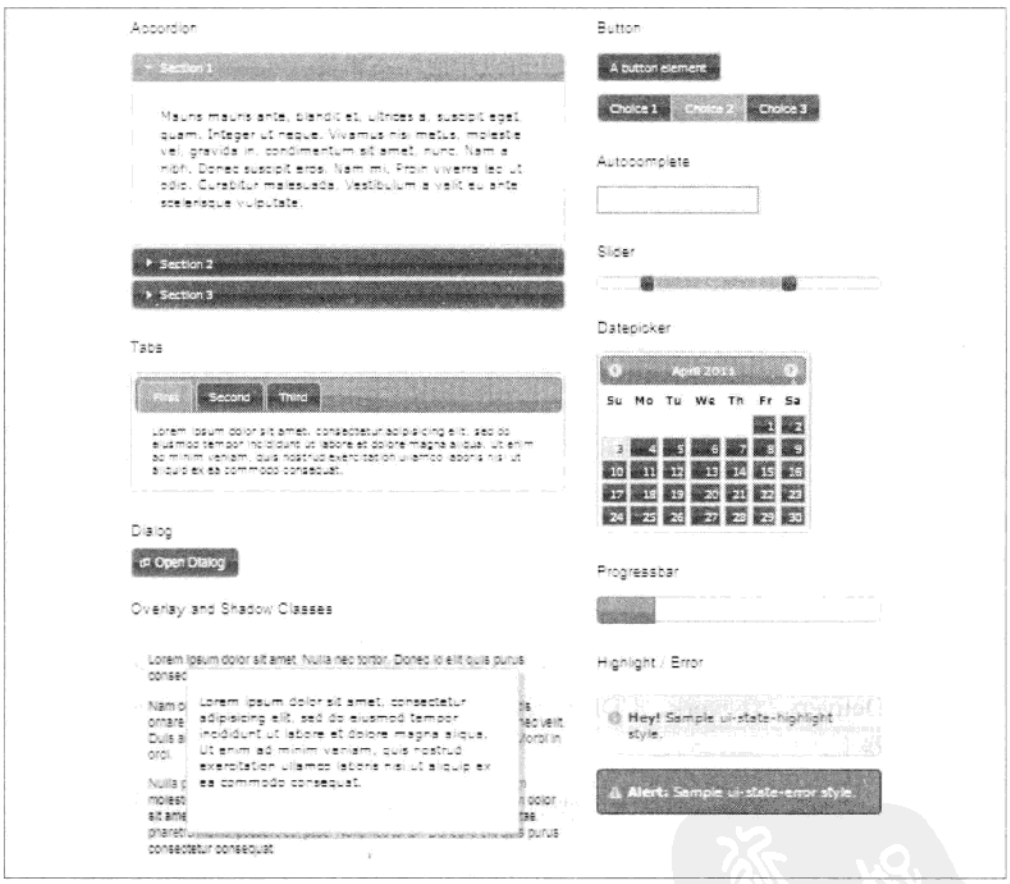



图 4-2　jQuery UI 元素

- Slider
- Tabs

虽然数量不多，但这些组件都很漂亮和稳定，还有一些组件正在创作中。这个库的特点是：易用、相对轻便、适合大多数表单和页面布局任务。这个库更多的是提供增强的网站体验，而不是提供重量级的应用体验。

除了UI组件，jQuery UI还提供一些有用的底层交互，你可以将其应用到任意DOM元素中：

Draggable

用鼠标移动元素。

Droppable

当将一个元素拖入另一个元素时，可以生成一个事件。

Resizable

通过拖动边缘和角点改变元素大小。

Selectable

点击选中一个或多个元素。

Sortable

通过拖拽元素来给它们重新排序。

你可以用这些来创建自己的组件。

### 1. 载入和使用 jQuery UI

jQuery UI的安装和使用很简单，只需在页面顶端包含少量JavaScript和CSS文件。所有相关文件（jQuery、jQuery UI 和 CSS 主题）都可以从谷歌内容分发网络（Content Delivery Network，CDN）载入，当然你也可以把这些文件放在自己的Web服务器上。

例4-1显示了如何设置一个基本的带日期选择组件的jQuery UI页面。图4-3显示了结果。

例4-1 基本jQuery UI设置

```
<!DOCTYPE html>
<html>
<head>
    <title>jQuery UI</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">

    <!--    jQuery UI font sizes are relative to document's,
            so set a base size here. -->
    <style type="text/css">
```

```
        body {
            font-size: 12px;
            font-family: sans-serif
        }
    </style>

    <!-- Load the jQuery UI style sheet. -->
    <link rel="stylesheet" href="
        http://ajax.googleapis.com/ajax/libs/jqueryui/1.8.11/themes/start/jquery-ui.css"
        type="text/css" media="all" />

    <!-- Load jQuery. -->
    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.5.1/jquery.min.js"
        type="text/javascript"></script>

    <!-- Load jQuery UI. -->
    <script src="http://ajax.googleapis.com/ajax/libs/jqueryui/1.8.11/jquery-ui.min.js"
        type="text/javascript"></script>


    <script>
        // On DOM loaded, initialize a date picker widget on the input element
        // with id of 'datepicker'.
        $(function() {
            $("#datepicker").datepicker();
        });
    </script>

</head>
<body>
    <!-- The following input element will be turned into a date picker. -->
    <p>Enter Date: <input type="text" id="datepicker"></p>
</body>
</html>
```


图 4-3　jQuery UI 日期选择

## 2．jQuery UI 主题

如果默认主题不适合需求的话，可以很轻松地换成其他 jQuery UI 主题。在载入 jQuery UI CSS 样式文件的那一行，将/start/部分修改为其他的主题名称。例如：

...ajax/libs/jqueryui/1.8.11/themes/ui-lightness/jquery-ui.css

或

...ajax/libs/jqueryui/1.8.11/themes/le-frog/jquery-ui.css

主题名称中如果有空格（如 UI Lightness），用连字号（-）代替并将名称转为小写：ui-lightness。

访问 http://jqueryui.com/themeroller/，查看 24 个标准主题的完整列表。

如上所述，除了直接链接到谷歌 CDN 上的主题文件之外，你也可以下载保存在自己的服务器上。

除标准主题外，你还可以使用上述网页上的 ThemeRoller 应用（如图 4-4 所示）来修改已有主题或从头创建新主题，并下载和使用。注意 jQuery UI 字体大小是相对页面基本字体大小而言的，因此最好为页面设置一个默认字体大小，否则字体会太大。


图 4-4　jQuery UI ThemeRoller

### 3．用 ExtJS 创建重量级 UI

与 jQuery UI 相反，Ext JS 提供了一个全面的重量级 UI 系统。在它更严格的应用框架

下提供了看似无尽的一系列 UI 功能。Ext JS 使得 Web 开发和客户端 GUI 开发几乎一样。它适合复杂的后台管理界面（如电子商务管理）或精细的前台 Web 应用。反过来说，如果你只需要少量额外的组件，使用 Ext JS 有点大才小用。这种情况，你应该考虑更轻量级的 jQuery UI。

Ext JS 的更多信息请见 Sencha 网页：http://www.sencha.com。

因为 Ext JS 几乎没什么不能做的，所以完全不必列出 Ext JS 的所有功能。Sencha 网站的一些例子远远超出了基本组件，包括了整个 Web 桌面、复杂数据网格、论坛浏览器等应用。它有布局管理器来分离和组织 UI 页面内容，还有绑定不同组件和远程数据源的工具。Ext JS 还有一些意想不到的特性，包括 Google Maps 和图表（如图 4-5 和图 4-6 所示）。

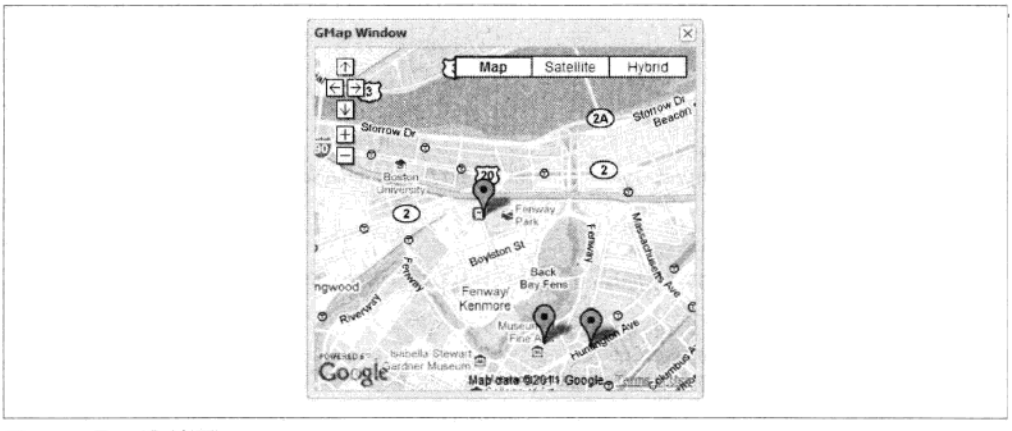



图 4-5　Ext JS 地图

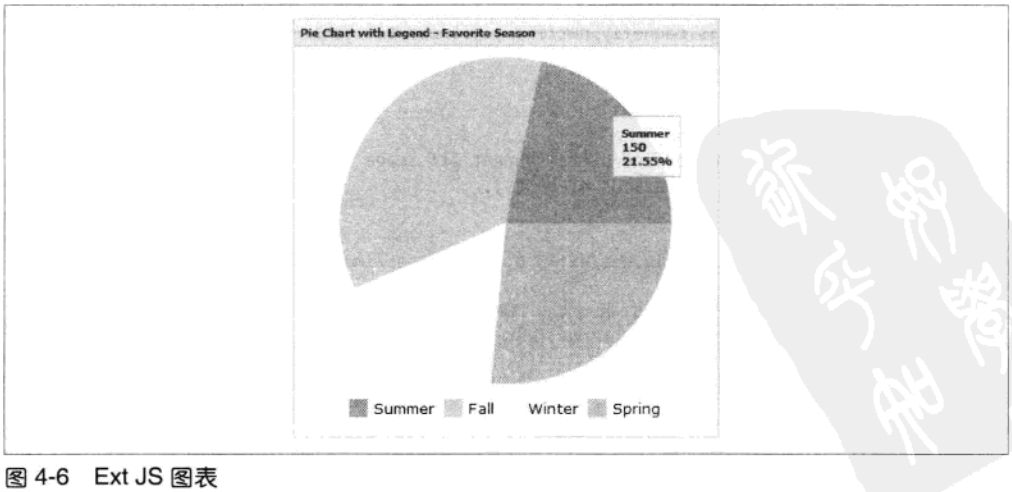



图 4-6　Ext JS 图表

### 4．载入和使用 Ext JS

和 jQuery UI 一样，载入 Ext JS 的资源很简单，CDN 中有相应的 CSS 和 JavaScript 文件。Cachefly 网络提供了这些文件，当然你也可以在自己的服务器上提供相应文件。

尽管 Ext JS 可以像 jQuery 一样直接操控 DOM 元素，但 Ext JS 的标准做法更倾向于创建对象并在页面上呈现它们。在许多方面，使用 Ext JS 就更像是传统、非 DOM 式的应用开发。这种做法在大型项目中也许能增加可读性。最终，你更喜欢 Ext JS 或 jQuery 更多取决于个人品味。

图 4-2 创建了一个窗口对象（不要和 DOM 窗口混淆）并给它附加一个日期组件、一个空白（spacer）对象和一个滑块组件（如图 4-7 所示）。

例 4-2　基本 Ext JS 设置

```
<!DOCTYPE html>
<html>
<head>
    <title>Ext JS</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">

    <!-- Load the Ext JS CSS. -->
    <link rel="stylesheet" type="text/css"
        href="http://extjs.cachefly.net/ext-3.3.1/resources/css/ext-all.css" />

    <!-- Load the Ext JS base JavaScript. -->
    <script type="text/javascript"
        src="http://extjs.cachefly.net/ext-3.3.1/adapter/ext/ext-base.js">
    </script>

    <!-- Load the rest of Ext JS.  -->
    <script type="text/javascript"
        src="http://extjs.cachefly.net/ext-3.3.1/ext-all.js">
    </script>

    <script type="text/javascript">

    // Tell Ext JS where to find a transparent gif image
    // (used for rendering various elements).

    Ext.BLANK_IMAGE_URL =
        'http://extjs.cachefly.net/ext-3.0.0/resources/images/default/s.gif';

    // Ext JS onReady is called when the DOM has loaded,
    // similar to jQuery's $(function(){}).
    Ext.onReady(
        function(){

            // Create a DateField object.
            var dateField = new Ext.form.DateField({
```

```
                fieldLabel: 'Date Widget',
                emptyText:'Enter date...',
                format:'Y-m-d',
                width: 128
            }),

            // Create a Slider object.
            slider = new Ext.Slider({
                width: 280,
                minValue: 0,
                maxValue: 100,
                plugins: new Ext.slider.Tip()
            }),

            // Create a Spacer object.
            space = new Ext.Spacer({
                height:64
            }),

            // Create a Window object to attach all of the above.
            win = new Ext.Window({
                title: 'Ext JS Demo',
                bodyStyle:'padding: 10px',
                width:320,
                height:280,
                items:[dateField, space, slider],
                layout:'form'
            });

            // Show the window.
            win.show();
        }
    );
    </script>
</head>
    <body>
    </body>
</html>
```

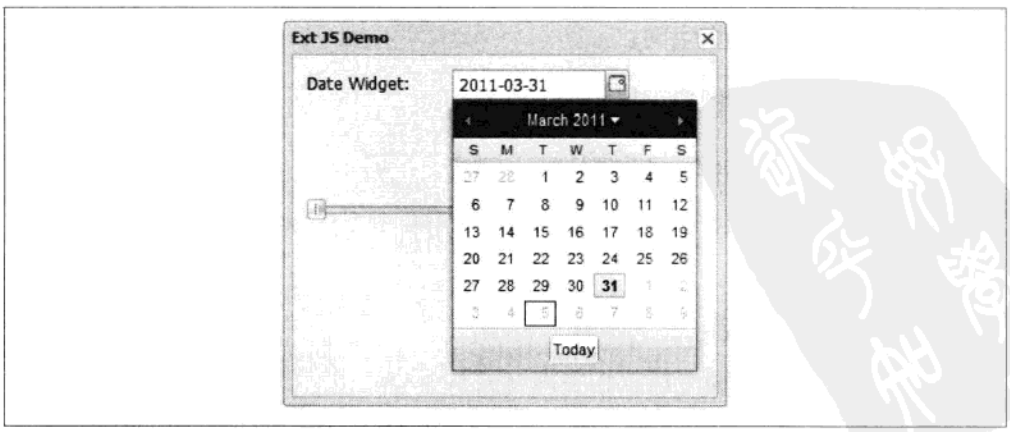



图 4-7　Ext JS 窗口对象、日期选择器和滑块

## 4.3 从头创建 UI 元素

使用已有的 UI 库对许多应用已完全足够，但有时还需要一个完全定制的组件。jQuery 这样的框架使这种定制简单了很多，你可以完全自由地调节元素的外观和行为，而不用担心是否能融入 UI 框架。

你可以采用本书前面介绍的一些技术来创建动态组建，如：

- 对自由浮动的组件元素采用绝对定位（position: absolute）；
- 在动画中使用定时器（setInterval()、setTimeout()）；
- 通过背景图像位置操作来显示一幅大图中的一小部分。

在本书第 9 章你将看到 jQuery 本身有一些动画机制，但自己定制动画代码可以给你创建更多特效的灵活性。下面的小节描述了如何用定制动画创建一个 3D 旋转木马（carousel）组件，可以在椭圆路径上缩放和移动元素。

### 4.3.1 创建 3D 旋转木马

本节中，我们将使用 jQuery 开发一个旋转木马组件。它将一组页面上的普通 HTML 图像（如图 4-8 所示）转化为一个具有 3D 缩放效果的旋转木马组件（如图 4-9 所示）。

为什么要创建旋转木马呢?

- 它在视觉上吸引人；
- 它使得图像占用更少空间；
- 它使得不同数目的图像占用的空间不变。

#### 1．旋转木马规格（specification）

当开发一个旋转木马这样的交互元素时，我们需要考虑目标浏览器的多样性和元素的页面环境。比如：

- 如果 JavaScript 被关闭了会怎样?
- 如果当前使用的是纯文本的屏幕阅读器会怎样?
- 在 IE6 等老浏览器上会怎样?

图 4-8　普通图像

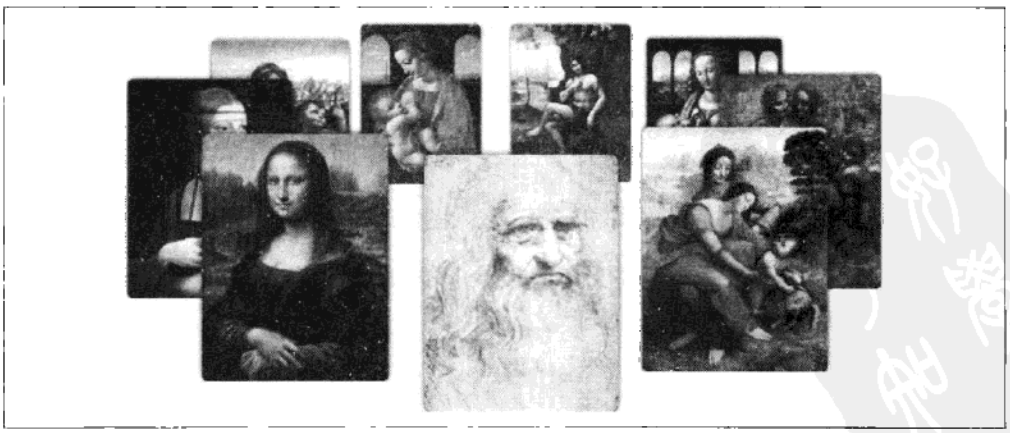


图 4-9　3D 旋转木马

如果旋转木马不能初始化，应该给用户呈现普通图像（或 alt 标签中的文本）。如果浏览器支持，旋转木马插件负责将这些普通图像转化为更有趣的东西。旋转木马初始化失败时，图像完全消失是不能接受的。另外，页面的 HTML 应该不需要为使用此插件而违反 WC3 验证或语义。

旋转木马还应在 IE6/7 这些老浏览器上工作，尽管这些不安全的浏览器的市场占有率在下降，还是有不少人在使用它们。根据微软鼓励停止使用 IE6 的 IE6 倒计时网站(http://www.theie6countdown.com)，到 2011 年 4 月还有 11.4%的网民使用 IE6。

**提示**

*尽管旋转木马可以在 IE6 下工作，但是例子中的 PNG 图像不能正确绘制。一个简单的处理办法是将 PNG 图像改为 JPEG 图像，因为 JPEG 图像可以在所有浏览器上正确显示。*

页面中旋转木马的个数不应受到限制。这意味着我们开发组件时需要将代码封装好，使得可以实例化无限次。以 jQuery 插件的方式来实现旋转木马可以使得初始化多个旋转木马非常简单。我们只需要将图像包装在 jQuery 可以识别的元素中，然后对它们进行插件调用。比如，下面的代码对所有带 carousel3d CSS 类的元素初始化一个旋转木马：

```
$('.carousel3d').Carousel();
```

下面这些规范可以提高旋转木马的观感：

- 所有图像应该保持它们的属性和附加上的基于事件的功能；
- 图像周围的链接不应该被旋转木马影响；
- 旋转木马的外观应该比较灵活，如维度和图像的缩放；
- 旋转木马应能自动给不同数目的元素平均分配空间；
- 为避免 DOM 改变引起的颤动和颠簸，载入图像时旋转木马元素应能干净地淡入；
- 旋转木马应在鼠标移动到其元素上时停止转动，鼠标移开时重新开始转动。这可以使选择元素更容易。

### 2. 载入旋转木马图像

为正确地初始化旋转木马，我们必须知道图像的高和宽来进行位置和缩放的计算。

理想情况下，我们在载入图像前就知道它们的大小。实际中不一定如此，我们可能需要载入图像后才能读到高度和宽度属性。

检测图像什么时候被载入是件挺麻烦的事，并不是给图像加个 load 事件那么简单，因为图像的 load 事件在不同浏览器上是不一致的。有的浏览器在载入图像时根本不触发 load 事件，有的则是从网络载入时触发而从浏览器缓冲载入时不触发。一个保险的做法是监听窗口 load 事件，因为当这个事件触发时，意味着所有的页面资源（包括所有图片）都被载入了。这种方法的缺点是必须等待整个页面都被载入，用户才能开始和内容交互。

尽管重新载入 DOM 中已有的 image 元素看起来浪费时间，但实际上这个代价非常小，因为如果这些已经载入的图像是从浏览器缓冲中重新载入的。

下面的 loadImage()函数将进行图像载入初始化和检测，并考虑了浏览器的多样性。它初始化图像的载入，并当从网络或缓冲获取到图像时执行一个回调函数。这个函数可以处理 DOM 中已有的图像元素，或用 new Image()创建的图像元素。loadImage()的参数是：一个图像元素、图像的源 URL 和一个回调函数。

```
// Function to execute a callback when an image has been loaded,
// either from the network or from the browser cache.

var loadImage = function ($image, src, callback) {

    // Bind the load event BEFORE setting the src.
    $image.bind("load", function (evt) {

        // Image has loaded, so unbind event and call callback.
        $image.unbind("load");
        callback($image);

    }).each(function () {
        // For Gecko-based browsers, check the complete property,
        // and trigger the event manually if image loaded.
        if ($image[0].complete) {
            $image.trigger("load");
        }
    });
    // For Webkit browsers, the following line ensures load event fires if
    // image src is the same as last image src. This is done by setting
    // the src to an empty string initially.
    if ($.browser.webkit) {
        $image.attr('src', '');
    }
    $image.attr('src', src);
};
```

注意在设置图像源前绑定事件。避免在设置好事件处理函数之前，图像已经很快地从缓冲中载入。

### 3. 旋转木马元素对象

旋转木马由若干围绕中心点旋转的元素组成，随着距离增加而缩小形成三维效果。每个元素由 createItem()函数创建，作为一个单独的对象实例对待。此函数对单个旋转木马元素进行如下操作：

- 它通过 loadImage()触发了初始的图像载入（图像可能已经在浏览器缓冲中）；
- 一旦载入图像，呈现淡入效果，将宽度和高度保存，以备 update()函数中计算缩放使用；
- update()函数根据元素的旋转角度，更新其位置、缩放和 z 深度。

```
// Create a single carousel item.
var createItem = function ($image, angle, options) {
    var loaded = false, // Flag to indicate image has loaded.
        orgWidth,       // Original, unscaled width of image.
        orgHeight,      // Original, unscaled height of image.
        $originDiv,     // Image is attached to this div.
    // A range used in the scale calculation to ensure
    // the frontmost item has a scale of 1,
    // and the farthest item has a scale as defined
    // in options.minScale.
    sizeRange = (1 - options.minScale) * 0.5,

    // An object to store the public update function.
    that;

// Make image invisible and
// set its positioning to absolute.
$image.css({
    opacity: 0,
    position: 'absolute'
});
// Create a div element ($originDiv). The image
// will be attached to it.
$originDiv = $image.wrap('<div style="position:absolute;">').parent();

that = {
    update: function (ang) {
        var sinVal, scale, x, y;

        // Rotate the item.
        ang += angle;

        // Calculate scale.
        sinVal = Math.sin(ang);
        scale = ((sinVal + 1) * sizeRange) + options.minScale;

        // Calculate position and zIndex of origin div.
        x = ((Math.cos(ang) * options.radiusX) * scale) + options.width / 2;
        y = ((sinVal * options.radiusY) * scale) + options.height / 2;
```

```
            $originDiv.css({
                left: (x >> 0) + 'px',
                top: (y >> 0) + 'px',
                zIndex: (scale * 100) >> 0
            });
            // If image has loaded, update its dimensions according to
            // the calculated scale.
            // Position it relative to the origin div, so the
            // origin div is in the center.
            if (loaded) {
                $image.css({
                    width: (orgWidth * scale) + 'px',
                    height: (orgHeight * scale) + 'px',
                    top: ((-orgHeight * scale) / 2) + 'px',
                    left: ((-orgWidth * scale) / 2) + 'px'
                });
            }
        }
    };

    // Load the image and set the callback function.
        loadImage($image, $image.attr('src'), function ($image) {
            loaded = true;
            // Save the image width and height for the scaling calculations.
            orgWidth = $image.width();
            orgHeight = $image.height();
            // Make the item fade-in.
            $image.animate({
                opacity: 1
            }, 1000);

        });
        return that;
    };
```

传给 createItem()函数的图像元素从 DOM 而来。除了一些 CSS 改动和附属到一个“手柄”div 元素中，这个图像元素保留了其附属的事件和锚元素。

### 4．旋转木马对象

旋转木马对象是旋转木马的“大脑”，进行各种初始化和处理工作：

- 迭代一个所有封装元素的所有图像子元素，对每个图像初始化一个旋转木马元素，并在 items[]数组中保存对它们的引用。
- 监听从旋转木马元素起泡（bubble up）的 mouseover 和 mouseout 事件。当检测到图像上的 mouseover 事件时，旋转木马暂停；当检测到 mouseout 事件，旋转木马在短暂延迟后重新启动。这个延迟是为了避免当用户移动到元素间的空隙时短暂启动后又停止。

最后我们创建一个 setInterval()循环来更新旋转木马的旋转值，并通过调用 update()

函数将值传给每个旋转木马元素。旋转木马每 30 毫秒进行此操作（或根据 options 中的 frameRate 属性）。默认的 30 毫秒保证了流畅的动画。值越大，流畅度越差，但占有越少的 CPU 资源。这对有若干旋转木马的页面可能更合适。

```
// Create a carousel.
var createCarousel = function ($wrap, options) {
    var items = [],
        rot = 0,
        pause = false,
        unpauseTimeout = 0,
        // Now calculate the amount to rotate per frameRate tick.
        rotAmount = ( Math.PI * 2) * (options.frameRate/options.rotRate),
        $images = $('img', $wrap),
        // Calculate the angular spacing between items.
        spacing = (Math.PI / $images.length) * 2,
        // This is the angle of the first item at
        // the front of the carousel.
        angle = Math.PI / 2,
        i;

    // Create a function that is called when the mouse moves over
    // or out of an item.
    $wrap.bind('mouseover mouseout', function (evt) {
        // Has the event been triggered on an image? Return if not.
        if (!$(evt.target).is('img')) {
            return;
        }

        // If mouseover, then pause the carousel.
        if (evt.type === 'mouseover') {
            // Stop the unpause timeout if it's running.
            clearTimeout(unpauseTimeout);
            // Indicate carousel is paused.
            pause = true;
        } else {
            // If mouseout, restart carousel, but after a small
            // delay to avoid jerking movements as the mouse moves
            // between items.
            unpauseTimeout = setTimeout(function () {
                pause = false;
            }, 200);
        }

    });

    // This loop runs through the list of images and creates
    // a carousel item for each one.
    for (i = 0; i < $images.length; i++) {
        var image = $images[i];
        var item = createItem($(image), angle, options);
        items.push(item);
        angle += spacing;
    }

    // The setInterval will rotate all items in the carousel
```

```
    // every 30ms, unless the carousel is paused.
    setInterval(function () {
        if (!pause) {
            rot += rotAmount;
        }
        for (i = 0; i < items.length; i++) {
            items[i].update(rot);
        }
    }, options.frameRate);
};
```

### 5. jQuery 插件部分

我们用一个标准的 jQuery 插件函数来初始化旋转木马。这使得我们可以用习惯的选择器方式来初始化旋转木马。我们可以定义这样一个 HTML 布局：

```
<div class="carousel" ><!-- This is the wrapping element -->
    <img src="pic1.png" alt="Pic 1"/>
    <img src="pic2.png" alt="Pic 2"/>
    <img src="pic3.png" alt="Pic 3"/>
    <img src="pic4.png" alt="Pic 4"/>
    <img src="pic5.png" alt="Pic 5"/>
</div>

<div class="carousel" ><!-- This is the wrapping element -->
    <img src="pic1.png" alt="Pic 1"/>
    <img src="pic2.png" alt="Pic 2"/>
    <img src="pic3.png" alt="Pic 3"/>
</div>
```

注意我们使用一个封装 div 来包含旋转木马的真正元素。此例中，我们使用 carousel CSS 类来标记封装元素，你也可以使用其他的选择器。你可以给单独的图像元素加入锚元素或给其绑定事件。当图像被转化为旋转木马元素后，它们还仍然保留。

初始化例子中的两个旋转木马，可以使用标准的 jQuery 插件调用：

```
$('.carousel').Carousel();
```

或加入选项：

```
$('.carousel').Carousel({option1:value1, option2:value2...});
```

下面是插件代码：

```
// This is the jQuery plug-in part. It iterates through
// the list of DOM elements that wrap groups of images.
// These groups of images are turned into carousels.
$.fn.Carousel = function(options) {
    this.each( function() {
        // User options are merged with default options.
        options = $.extend({}, $.fn.Carousel.defaults, options);
        // Each wrapping element is given relative positioning
```

```
        // (so the absolute positioning of the carousel items works),
        // and the width and height are set as specified in the options.
        $(this).css({
            position:'relative',
            width: options.width+'px',
            height: options.height +'px'
        });
        createCarousel($(this),options);
    });
};
```

下面的代码定义了一组默认选项，在初始化旋转木马时可以覆盖这些选项。

```
// These are the default options.
$.fn.Carousel.defaults = {
    radiusX:230,    // Horizontal radius.
    radiusY:80,     // Vertical radius.
    width:512,      // Width of wrapping element.
    height:300,     // Height of wrapping element.
    frameRate: 30, // Frame rate in milliseconds.
    rotRate: 5000, // Time it takes for carousel to make one complete rotation.
    minScale:0.60  // This is the smallest scale applied to the farthest item.
};
```

## 6. 旋转木马页面

下面的页面（见例 4-3）定义了一个有 9 个元素的旋转木马，其中一个元素是链接（达芬奇自画像），一个元素绑定了点击事件（蒙娜丽莎）。

例 4-3　旋转木马页面

```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Carousel</title>
    <style type="text/css">
        img { border:none;}
    </style>
    <script
        src="http://ajax.googleapis.com/ajax/libs/jquery/1.5.1/jquery.min.js">
    </script>

    <script type="text/javascript">

    // Start of jQuery carousel plug-in.
    (function($) {

        // Function to execute a callback when an image has been loaded,
        // either from the network or from the browser cache.
        var loadImage = function ($image, src, callback) {
            /*** CODE REMOVED FOR CONCISENESS ***/
        };

        // Create a single carousel item.
```

```
        var createItem = function ($image, angle, options) {
            /*** CODE REMOVED FOR CONCISENESS ***/
        };
        // Create a carousel.
        var createCarousel = function ($wrap, options) {
             /*** CODE REMOVED FOR CONCISENESS ***/
        };

        // This is the jQuery plug-in part. It iterates through
        // the list of DOM elements that wrap groups of images.
        // These groups of images are turned into carousels.
        $.fn.Carousel = function(options) {
              /*** CODE REMOVED FOR CONCISENESS ***/
        };

        // These are the default options.
        $.fn.Carousel.defaults = {
            /*** CODE REMOVED FOR CONCISENESS ***/
        };
    })(jQuery);
    // End of jQuery carousel plug-in.

    $(function(){
        // Create a carousel on all wrapping elements
        // with a class of .carousel.
        $('.carousel').Carousel({
            width:512, height:300,  // Set wrapping element size.
            radiusX:220,radiusY:70, // Set carousel radii.
            minScale:0.6            // Set min scale of rearmost item.

        });

        // Bind a click event to one of the pictures (Mona Lisa)
        // to show events are preserved after images become
        // carousel items.
        $('#pic2').bind('click', function() {
            alert('Pic 2 clicked!');
        });
    });
    </script>

</head>
<body>

    <div class="carousel" ><!-- This is the wrapping element -->
        <a href="http://en.wikipedia.org/wiki/Self-portrait_(Leonardo_da_Vinci)"
            target="_blank">
            <img src="pic1.png" alt="Pic 1"/>
        </a>
        <img id="pic2" src="pic2.png" alt="Pic 2"/>
        <img src="pic3.png" alt="Pic 3"/>
        <img src="pic4.png" alt="Pic 4"/>
        <img src="pic5.png" alt="Pic 5"/>
        <img src="pic6.png" alt="Pic 6"/>
        <img src="pic7.png" alt="Pic 7"/>
        <img src="pic8.png" alt="Pic 8"/>
        <img src="pic9.png" alt="Pic 9"/>
```

```
        </div>

    </body>
    </html>
```

请尝试修改代码来包含更多的旋转木马，给一些其他图像加入更多点击功能，或创建链接。

Web 应用和桌面应用在观感上的差距在缩小。现在有许多工具，使一个现代 Web 应用完全可以看起来比桌面版本更好。实际上随着浏览器、JavaScript 性能和库的不断进步，基于云的 Web 应用在许多情况下完全可以作为传统桌面应用的替代品。而采用 Web 应用，用户不需要任何客户端安装和更新负担就能保持他们的软件处于最新状态。

# 第 5 章

# JavaScript 游戏介绍

2010 年 5 月 22 日，谷歌为了纪念吃豆人（Pac-Man）游戏诞生 30 周年，在首页特别推出了该款游戏的在线版本（如图 5-1 所示）。许多人开始认为它是用 HTML5 实现的，但仔细查看才发现除了声音之外，它完全是用普通 DHTML 实现的。在本章中，我们将继续怀旧主题，并开发我们自己的 DHTML 游戏：向经典游戏太空入侵者（Space Invader）致敬的轨道攻击（Orbit Assault，如图 5-2 所示）。

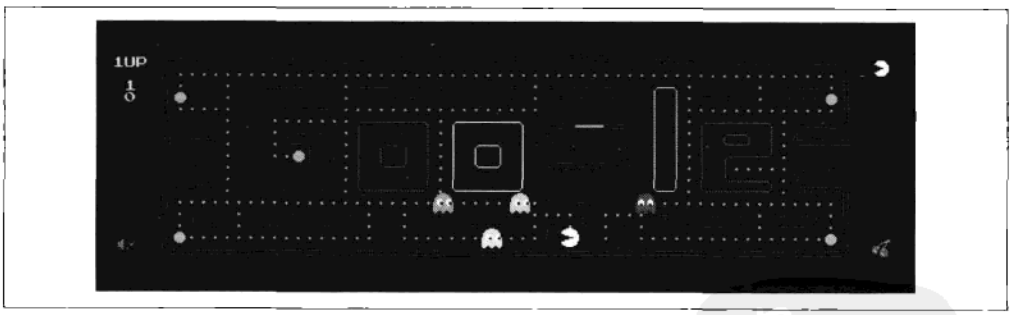



图 5-1　谷歌主页上的吃豆人

从头开始开发一个完整的游戏听起来任重道远，但确实是介绍游戏开发各方面最好的办法。

但为什么要限制在 DHTML 呢？为什么不直接使用更强大的技术，比如 HTML5 Canvas？这就像是高原训练：如果我们可以仅仅用 DHTML 就写出不错的东西，就意味着我们已经准备好了用 Canvas 开发出更好的东西。

太空入侵者游戏 1978 年由日本 Taito 公司发布，设计师为西角友宏。他不仅设计和

实现了游戏，而且还实现了它所运行的硬件平台。这个经典游戏被认为是游戏产业的一个标志，至今仍具有趣味性。

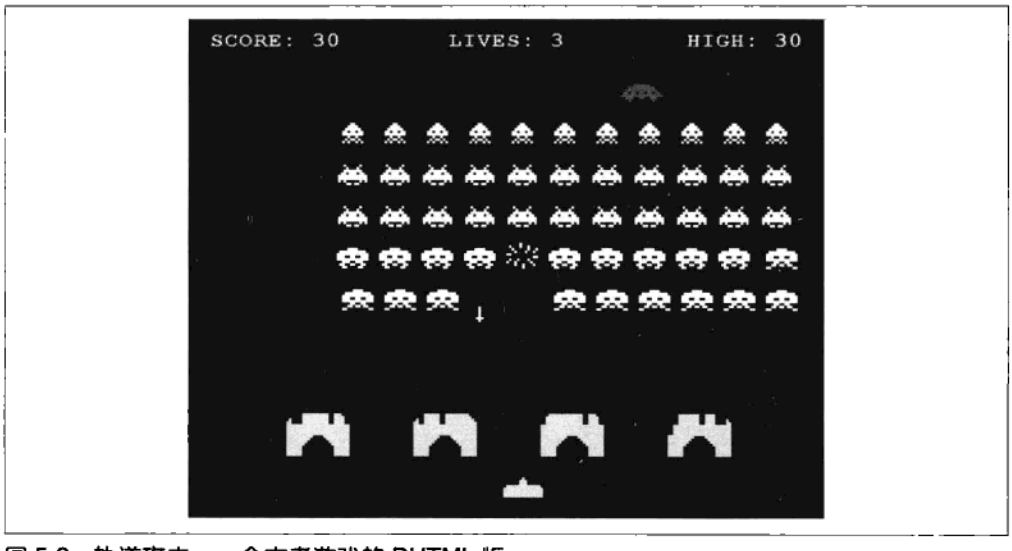



图 5-2　轨道攻击，一个古老游戏的 DHTML 版

为了让网络用户一样喜欢我们的 DHTML 版轨道攻击游戏，我们提出了下列需求：

- 应该能在不同硬件平台的流行浏览器下正常运行；
- 显而易见，我们应该在不同硬件和浏览器下保持速度一致；
- 我们应该保留原太空入侵者游戏最好的特点，包括像护甲这样较复杂的元素。

我们已经完成了部分工作：第 2 章中的 DHTMLSprite 和 timeInfo 对象。本章将在它们的基础上继续开发。

## 5.1　游戏对象概述

“轨道攻击”使用 6 个关键的游戏对象。下面将讨论这些对象的行为特点以及它们如何交互。图 5-3 显示了这些对象使用的 sprite 图像。

外星人侵者

游戏中最让人印象深刻的特色也许就是外星人在屏幕上移动的催眠舞蹈。这些外星人排列在 5 行 11 列的格子中，在屏幕上横向移动，如果碰到了游戏区域的

边缘则稍微下降一点并改变横向移动的方向。只要有一个外星人到达游戏区域的最底部，游戏结束。

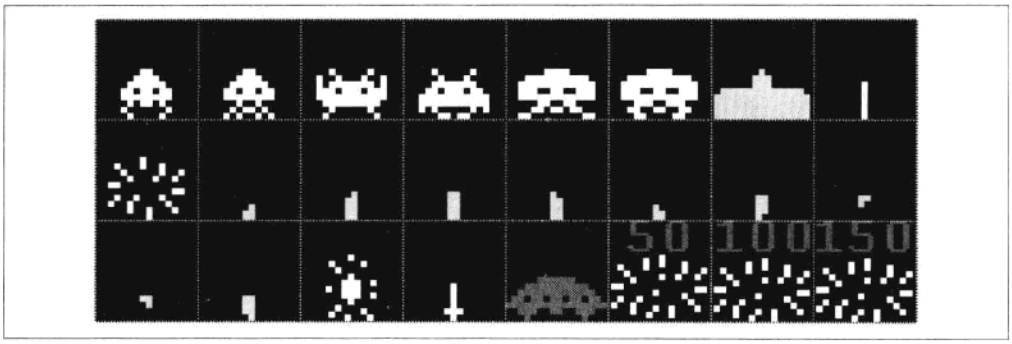



**图 5-3　轨道攻击使用 32 像素的正方形 sprite，被放在一个位图中**

玩家从他的坦克中发射激光来消灭外星人，同时还得躲开每列最底层外星人投放的炸弹。最上层的外星人最小，命中后奖励 40 分；下面两行的外星人较宽，命中后奖励 20 分；最下面两行的外星人最大，命中后奖励 10 分。

每次玩家消灭了 55 个外星人后，下一波攻击的初始位置要比上一次更低些。有个诀窍是先消灭两边的外星人，因为这会延长外星人碰到游戏区域边缘的时间。

原版本太空入侵者时代的硬件太慢，不能同时移动 55 个外星人，因此外星人在每个游戏周期（1/60 秒）轮流移动。这是为什么外星人移动时有闪烁的效果，它也产生了一个巧妙的游戏机制：外星人越少，他们移动的速度越快。这仅仅是每个外星人的运动变得更加频繁，其他外星人被渐渐淘汰的时候所产生的副作用，最后一个孤立的外星人将运动得非常迅速，因为它根本不需要等待。

太空入侵者实现最常见的错误是让所有的外星人组成一个固定的组同时运动，它需要根据留下来的外星人的个数添加额外的代码，给外星人加速，而且这样还会丢失掉原始游戏中最重要的方面之一。

外星炸弹

在每列中位置最低的外星人能够向玩家扔炸弹。在某一时间通常只能看见一个外星炸弹。从玩家的坦克发射的激光可以摧毁外星炸弹。外星炸弹会将玩家的护甲逐渐裂成碎片。

护甲

玩家的坦克有 4 个护甲，它们逐渐被外星炸弹和坦克激光摧毁。护甲是一把双刃剑，它们既能够给玩家坦克提供掩护，也会阻挡玩家的坦克激光。一个小技巧是，在护甲上爆破一个洞，用来通过玩家的坦克激光。

在原始游戏中，护甲被摧毁为小且不规则的像素块，使用 DHTML 模仿这种效果会非常棘手。在我们的游戏版本中，每个盾分割成 48 个独立的元素，这样可以在合理的 CPU 利用率下提供真实的摧毁效果。

玩家的坦克

坦克在玩家的控制下水平移动，外星炸弹一次就能摧毁它。它可以对外星人发射激光，而它的运动范围被限制在左护甲与右护甲之间。游戏开始时玩家有 4 辆坦克，每增加 5 000 分可额外获得一辆坦克。

玩家的激光

坦克射出的垂直激光，可以伤害护甲、摧毁外星人和拦截外星炸弹。一次只能使用一个激光，使这个游戏更加具有挑战性。一个直达场地顶端的错误射击，将导致你在这段时间内不能发射另一个激光。

神秘飞碟

在随机的时间间隔里，飞碟出现在外星人上方，并横穿整个游戏区域，如果激光成功攻击到飞碟，玩家将随机获得加分 50、100 或 150。

## 5.2 游戏代码

本节审查整个游戏的代码，解构所有的游戏元素并全面详细地解释它们。

### 5.2.1 游戏变量

下面的代码定义了各种游戏变量；为了清晰和方便起见，不变的常量用大写形式表示。$drawTarget 指的是游戏区域，被定义成了一个页面内的 div 元素。

```
var PLAYER = 1,
    LASER = 2,
    ALIEN = 4,
    ALIEN_BOMB = 8,
    SHIELD = 16,
    SAUCER = 32,
    TOP_OF_SCREEN = 64,
```

```
    TANK_Y = 352 - 16,
    SHIELD_Y = TANK_Y - 56,
    SCREEN_WIDTH = 480,
    SCREEN_HEIGHT = 384,
    ALIEN_COLUMNS = 11,
    ALIEN_ROWS = 5,
    SYS_process,
    SYS_collisionManager,
    SYS_timeInfo,
    SYS_spriteParams = {
        width: 32,
        height: 32,
        imagesWidth: 256,
        images: '/images/invaders.png',
        $drawTarget: $('#draw-target')
    };
```

## 5.2.2 读取键盘输入

jQuery 使 JavaScript 读取键盘输入更容易。通过监听绑定到页面（文档）的 keydown 和 keyup 事件，然后在触发键盘事件后读取事件对象的属性，我们就能够判断哪个键被按下或者释放。轨道攻击需要监听 3 个键：左、右，以及发射键。

```
var keys = function () {
```

keyMap{}对象将事件的键值映射到我们感兴趣的游戏按钮上。下面的代码中，键 Z 是向左键，键 X 是向右键，键 M 是发射键。你也可以根据表格 5-1 中列出的键值使用其他键。

```
var keyMap = {
    '90': 'left',
    '88': 'right',
    '77': 'fire'
},
```

表 5-1　　JavaScript 的键盘键值

| 键 | 值 | 键 | 值 | 键 | 值 |
|---|---|---|---|---|---|
| Backspace | 8 | Tab | 9 | Enter | 13 |
| Shift | 16 | Ctrl | 17 | Old | 18 |
| Pause/Break | 19 | Caps Lock | 20 | Escape | 27 |
| Page Up | 33 | Page Down | 34 | End | 35 |
| Home | 36 | Left arrow | 37 | Up arrow | 38 |
| Right arrow | 39 | Down arrow | 40 | lnsert | 45 |
| Delete | 46 | 0 | 48 | 1 | 49 |
| 2 | 50 | 3 | 51 | 4 | 52 |
| 5 | 53 | 6 | 54 | 7 | 55 |

续表

| 键 | 值 | 键 | 值 | 键 | 值 |
|---|---|---|---|---|---|
| 8 | 56 | 9 | 57 | a | 65 |
| b | 66 | c | 67 | d | 68 |
| e | 69 | f | 70 | g | 71 |
| h | 72 | i | 73 | j | 74 |
| k | 75 | l | 76 | m | 77 |
| n | 78 | o | 79 | p | 80 |
| q | 81 | r | 82 | s | 83 |
| t | 84 | u | 85 | v | 86 |
| w | 87 | x | 88 | y | 89 |
| z | 90 | Left window | 91 | Right window | 92 |
| Select | 93 | Numeric pad 0 | 96 | Numeric pad 1 | 97 |
| Numeric pad 2 | 98 | Numeric pad 3 | 99 | Numeric pad 4 | 100 |
| Numeric pad 5 | 101 | Numeric pad 6 | 102 | Numeric pad 7 | 103 |
| Numeric pad 8 | 104 | Numeric pad 9 | 105 | Multiply | 106 |
| Add | 107 | Subtract | 109 | Decimal point | 110 |
| Divide | 111 | F1 | 112 | F2 | 113 |
| F3 | 114 | F4 | 115 | F5 | 116 |
| F6 | 117 | F7 | 118 | F8 | 119 |
| F9 | 120 | F10 | 121 | F11 | 122 |
| F12 | 123 | Num Lock | 144 | Scroll Lock | 145 |
| Semicolon | 186 | Equals sign | 187 | Comma | 188 |
| Dash | 189 | Period | 190 | Forward slash | 191 |
| Grave accent | 192 | Open bracket | 219 | Backslash | 220 |
| Close bracket | 221 | Single quote | 222 | | |

kInfo{}对象包含 3 个游戏按钮的状态，1 表示按下，0 表示释放。任何时间你都可以通过返回的 kInfo{}对象（在游戏变量 SYS_keys 中引用）检查游戏按钮的状态：

```
kInfo = {
    'left': 0,
    'right': 0,
    'fire': 0
},
key;
```

页面（文档）绑定了 keydown 和 keyup 事件。当键盘事件被触发时，我们检查

按下的键是否在 keyMap{}里。如果是，在 kInfo{}里设置合适的游戏按钮状态。return false 语句是为了防止定义在 keyMap 对象中的按键事件冒泡到浏览器，而做出滚动页面（如果游戏使用了光标键）或到页底（如果游戏使用了空格键）等恼人的事情。

```
$(document).bind('keydown keyup', function (event) {
    key = '' + event.which;
    if (keyMap[key] !== undefined) {
        kInfo[keyMap[key]] = event.type === 'keydown' ? 1 : 0;
        return false;
    }
});
```

kInfo{}对象被返回，并将在游戏变量 SYS_keys 对象中引用。

```
    return kInfo;
}();
```

## 5.2.3 移动所有物体

游戏里的移动物体尽管有不同的性质，但都有一个共同的特点：它们都需要在每个游戏循环中执行特定的行为：

- 执行逻辑，比如检查它们是否被击中；
- 更新它们的视觉位置和碰撞位置；
- 如果适当的话，改变它们现有的形象。

鉴于它们需求的相近性，我们给移动物体添加一个 move()方法，并且将它们添加到一个共同的“处理”列表中。移动所有的游戏对象就是简单的列表遍历，并对每个对象都执行 move()方法。

删除对象更容易：一个对象可以设置自己的删除标志，再次遍历列表时它就被删除。我们创建一个 processor 对象处理所有这些功能。游戏变量 SYS_processor 引用了一个 processor 对象：

```
var processor = function () {
```

我们维护两个列表，processList[]里面包含需要移动的对象，而 addItems[]放置当 processList[]正在被遍历的时候创建的新对象：

```
var processList = [],
    addedItems = [];
```

add()方法增加新对象到处理列表，它们的 move()方法将被 process()方法调用：

```
return {
    add: function (process) {
        addedItems.push(process);
    },
```

process()方法遍历当前清单 processList[]，移动没有删除标记的对象，并将它们放入 newProcessList[]中，标记了删除标记的对象将会在下一次遍历中丢失。最后，我们用 newProcessList[]加 addedItems[]创建一个新的 processList[]，并且重新设定 addedItems[]，使它能接受新对象。

我们注意到，在遍历过程中，没有新的项目从 processList[]添加或删除，这使得处理遍历循环要简单得多：

```
        process: function () {
            var newProcessList = [],
            len = processList.length;
            for (var i = 0; i < len; i++) {
                if (!processList[i].removed) {
                    processList[i].move();
                    newProcessList.push(processList[i]);
                }
            }
            processList = newProcessList.concat(addedItems);
            addedItems = [];
        }
    };
};
```

## 5.2.4 一个简单的动画

这个通用的动画效果对象适用于产生像爆炸一样的效果。imageList 参数是动画用的图像编号数组，尽管在轨道攻击游戏中，动画是由单幅的图像组成的。timeout 参数是指动画持续的时间，以毫秒为单位。

```
var animEffect = function (x, y, imageList, timeout) {
    var imageIndex = 0,
        that = DHTMLSprite(SYS_spriteParams);
```

我们定义持续时间，在此时间后移除特效：

```
setTimeout(function(){
    that.removed = true;
    that.destroy();
}, timeout);
```

move()函数更新图像编号，当它达到图像列表末尾时，它会循环从头播放：

```
that.move = function () {
    that.changeImage(imageList[imageIndex]);
    imageIndex++;
    if (imageIndex === imageList.length) {
        imageIndex = 0;
    }
    that.draw(x, y);
};
```

动画效果把它自己加入了处理列表：

```
        SYS_process.add(that);
    };
```

## 5.2.5 碰撞检测

轨道攻击这个游戏仅仅只需要矩形重叠测试来判断两个物体是否相接触，但还是有大量的游戏对象组合可以相互碰撞：

- 外星人和激光；
- 飞碟和激光；
- 护甲和激光；
- 炸弹和坦克；
- 炸弹和护甲。

为每个组合编写具体的碰撞检测功能是一个繁琐的解决方案。更好的选择是开发一个能适用于所有组合，甚至可以用在其他类型游戏中的一般性碰撞系统。

另一个值得关注的是每个周期执行的碰撞测试次数。一个优化点是确保碰撞是单向的：如果已经对 A 测试了 B，就没有必要再对 B 测试 A。如果我们维护两组二进制标志，colliderFlag 和 collideeFlags，游戏对象可以迅速确定是否需要测试对方。表 5-2 说明了我们如何设置 3 个物体的碰撞标志。在这个例子中，激光将对飞碟和护甲测试，因为它们在 colliderFlag 中有 collideeFlags 值。飞碟和护甲将不会测试对方，因为它们的 collideeFlags 中是 0。

表 5-2　　碰撞标志

| | 激光 | 飞碟 | 护甲 |
|---|---|---|---|
| ColliderFlag | 1 | 2 | 4 |
| CollideeFlags | 2+4 | 0 | 0 |

一个快速检查标志的方式就是执行二进制与操作：

```
doCheck = objectA.collideeFlag & objectB.colliderFlag;
```

非零的结果意味着应该测试。

即便有这样的改善，仍然有大量的测试需要执行：

- 坦克的激光应该对以下的游戏对象进行检查：

  ——各有 48 个元素的 4 个护甲；

——外星人炸弹；

——飞碟；

——55 个外星人。

总共等于 249 个对象。

- 外星人炸弹应该对以下元素进行检查：

——各有 48 种元素的 4 个护甲；

——坦克。

总共等于 193 个对象。

每个周期都执行 442 次碰撞检测有点多，如果有更多的激光、炸弹和外星人，这种情况将会变得糟糕，也许会导致每个周期需要执行成千上万的测试。

我们可以通过消除对不可能碰撞对象的多余检查，来进一步减少测试的次数。一个简洁的实现方式就是创建一个网格，其中每个方格都包含了占领它的对象清单。一个对象只需要对在同一网格或相邻网格（共 9 格）内的对象执行碰撞检查。只要一个网格可以容下最大的对象，就能使用这个技术。在轨道攻击中，网格大小为 32 像素。图 5-4 显示了我们如何通过这种方式，来消除对明显无法碰撞的对象的检查。只有左边的 5 个外星人有机会接触到坦克，它们将会被检查，而右边的 3 个外星人将被忽略。

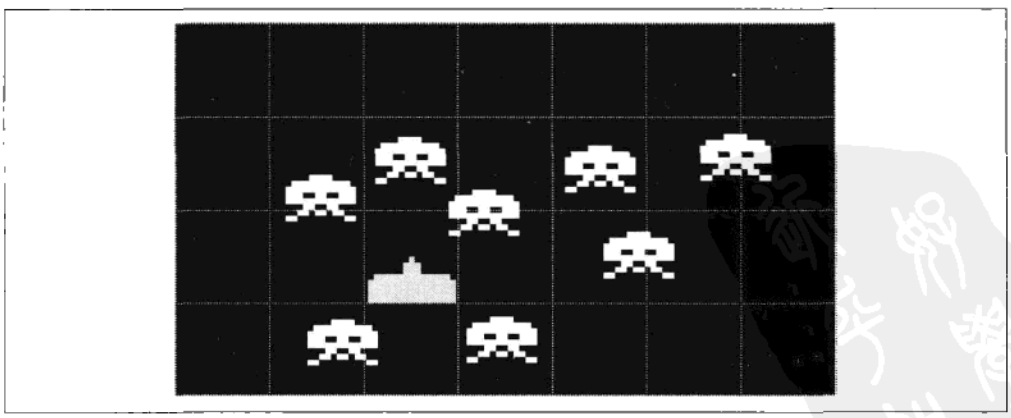


**图 5-4　游戏对象被分割成网格来执行碰撞检测**

这种简单碰撞测试中采用对象分区的做法被称为宽阶段（broad-phase）碰撞检

测，它仍然是现代电视游戏中保持速度的一个关键因素。你可以采用更复杂的方法，如数据树（优化对象的搜索和排序），但是消除冗余测试的目标仍然是相同的。通常一个现代三维游戏会在使用更多复杂的几何测试前先执行宽阶段碰撞检测。

轨道攻击使用一个碰撞管理对象，它返回碰撞对象。游戏对象使用这些碰撞对象给它们自己增加碰撞能力。我们在 SYS_collisionManager 中引用一个游戏级（game-wide）碰撞管理器：

```
var collisionManager = function () {
```

接下来，我们声明变量，包括网格本身和 listIndex，这是用来作为放置在网格中每个碰撞对象的唯一标志符。checkList 中保留着那些需要测试的碰撞对象名单，checkListIndex 是 checkList 中的碰撞对象的唯一标志符。gridWidth 和 gridHeight 定义整个网格的大小（以方格为单位），每个方格代表一个 32 像素的正方形的面积。

```
    var listIndex = 0,
        grid = [],
        checkListIndex = 0,
        checkList = {},
        gridWidth = 15,
        gridHeight = 12;
```

初始化时将网格内每个方格设为一个空对象。这些方格对象将保持每个方格中的碰撞对象列表，每个碰撞对象作为方格对象的一个属性。我们通过 listIndex 变量来命名这些属性，为什么使用对象而不用数组呢？因为对象比数组更容易删除某一个属性（我们的碰撞对象），而不影响剩余属性的索引。当碰撞对象在网格上移动的时候，我们需要不断在方格对象上添加或者删除碰撞对象：

```
    for (var i = 0; i < gridWidth * gridHeight; i++) {
        grid.push({});
    }
```

getGridList() 函数接受 x 和 y 像素坐标，返回坐标对应的方格对象。如果坐标在边界以外，它返回 undefined：

```
    var getGridList = function (x, y) {
        var idx = (Math.floor(y / 32) * gridWidth) + Math.floor(x / 32);
        if (grid[idx] === undefined) {
            return;
        }
        return grid[idx];
    };
```

newCollider()函数由需要和其他对象发生碰撞的游戏对象调用。它接受 colliderFlag 和 collideeFlags 来决定需要对哪些游戏对象进行碰撞检测。其他的参数包括：游戏

对象的宽度和高度（以像素为单位），碰撞发生时候的回调。

在这里，我们计算出的游戏对象的半宽和半高，这将在我们以后的碰撞检测函数中用到：

```
return {
    newCollider: function(colliderFlag, collideeFlags, width, height, callback){
        var list, indexStr = '' + listIndex++,
            checkIndex;
        var colliderObj = {
            halfWidth: width / 2,
            halfHeight: height / 2,
            centerX: 0,
            centerY: 0,
            colliderFlag: colliderFlag,
            collideeFlags: collideeFlags,
```

update()函数允许一个游戏对象更新其碰撞对象的位置。我们计算出游戏对象的中心点，并存储在 centerX 和 centerY 属性中。碰撞对象将从旧位置的方格列表中删除，并放置在一个新位置的方格列表中：

```
update: function (x, y) {
    colliderObj.centerX = x + 16;
    colliderObj.centerY = y + 32 - colliderObj.halfHeight;
    if (list) {
        delete list[indexStr];
    }
    list = getGridList(colliderObj.centerX, colliderObj.centerY);
    if (list) {
        list[indexStr] = colliderObj;
    }
},
```

remove()函数将碰撞对象从 collisionManager 网格中移除：

```
remove: function () {
    if (collideeFlags) {
        delete checkList[checkIndex];
    }
    if (list) { // list could be undefined if item was off-screen
        delete list[indexStr];
    }
},
```

接下来，callBack()方法调用 newCollider()参数中指定的回调函数：

```
callback: function () {
    callback();
},
```

checkCollisions()函数遍历方格中的碰撞对象，检查是否发生碰撞。首先确保碰撞物体没有对自己进行测试，然后检查碰撞标志。之后才能够执行矩形测试来判定两个对象是否接触。它检查两个对象的中心点在 x 轴和 y 轴的距离，如果距离大于它们

半高度或半宽度的总和，那么它们就没有接触：

```
        checkCollisions: function (offsetX, offsetY) {
            var list = getGridList(colliderObj.centerX + offsetX,
                                   colliderObj.centerY + offsetY);
            if (!list) {
                return;
            }
            var idx, collideeObj;
            for (idx in list) {
                if (list.hasOwnProperty(idx) &&
                    idx !== indexStr &&
                    (colliderObj.collideeFlags & list[idx].colliderFlag)) {
                    collideeObj = list[idx];
                    if(Math.abs(colliderObj.centerX - collideeObj.centerX) >
                        (colliderObj.halfWidth + collideeObj.halfWidth)) {
                        continue;
                    }
                    if(Math.abs(colliderObj.centerY - collideeObj.centerY) >
                        (colliderObj.halfHeight + collideeObj.halfHeight)) {
                        continue;
                    }
                    collideeObj.callback(colliderObj.colliderFlag);
                    callback(collideeObj.colliderFlag);
                    return true;
                }
            }
            return false;
        }
    };
```

如果这个碰撞对象有一个 collideeFlags 值非零，它将被添加到 checkList 中执行碰撞测试，最后返回碰撞对象。

```
        if (collideeFlags) {
            checkIndex = '' + checkListIndex++;
            checkList[checkIndex] = colliderObj;
        }
        return colliderObj;
    },
```

checkCollisions()是 collisionManager 的主要碰撞函数，它通过调用碰撞对象自己的 checkCollisions()函数测试所有相关的碰撞对象（由碰撞标志定义）：

```
    checkCollisions: function () {
        var idx, colliderObj;
        for (idx in checkList) {
                    if (checkList.hasOwnProperty(idx)) {
                        colliderObj = checkList[idx];
                        for (var y = -32; y <= 32; y += 32) {
                            for (var x = -32; x <= 32; x += 32) {
                                if (colliderObj.checkCollisions(x, y)) {
                                    break;
                                }
                            }
```

```
                    }
                }
            }
        }
    };
};
```

## 5.2.6 外星人

外星人可能是游戏中最复杂的元素。本节将覆盖其行为的各个方面，如炸弹投掷逻辑、移动和反应速度。

### 1. 外星人炸弹

外星人炸弹随机地从每一列任意一个最低的外星人上降落。炸弹的起始位置（x,y）和一个回调函数（当炸弹被移去时执行）将作为参数传递进来。最后我们设置炸弹的图像序号。

```
var alienBomb = function (x, y, removedCallback) {
    var that = DHTMLSprite(SYS_spriteParams),
        collider;
    that.changeImage(19);
```

当任何对象与炸弹相碰的时候，remove()函数将被调用，它创造一个动画效果（一个小规模的爆炸），同时移除炸弹本身，并调用参数中指定的回调函数 removed Callback()：

```
that.remove = function () {
    animEffect(x, y + 8, [18], 250, null);
    that.destroy();
    collider.remove();
    that.removed = true;
    removedCallback();
};
```

炸弹被添加到碰撞系统。请注意炸弹的碰撞检测只针对护甲，不针对玩家；而玩家的坦克则执行对外星炸弹的检查。

```
collider = SYS_collisionManager.newCollider(ALIEN_BOMB, SHIELD,
    6, 12, that.remove);
```

move()函数只是在更新碰撞对象时简单地将炸弹往下移动，并且检查是否其位置已通过坦克的垂直水平。

```
that.move = function () {
    y += 3.5 * SYS_timeInfo.coeff;
    that.draw(x, y);
    collider.update(x, y);
    if (y >= TANK_Y) {
        that.remove();
    }
};
```

外星炸弹将它自己添加到处理列表中：

```
        SYS_process.add(that);
    };
```

## 2. 外星入侵者

每个外星人侵者都非常愚蠢，它无非就是保持一个绘图的子图（sprite），并且接受高层的 aliensManager 对象传达行进命令而移动。

外星人侵者接受 x 和 y 像素坐标和图像编号。其他参数包括一个分值和命中回调函数。canFire 属性用来决定一个外星人侵者是否可以投弹，它的初始值设为 false：

```
var alien = function (x, y, frame, points, hitCallback) {
    var animFlag = 0,
        that = DHTMLSprite(SYS_spriteParams),
        collider, collisionWidth = 16;
    that.canFire = false;
```

当外星人侵者被击中时，remove()将被调用。如果外星人侵者被护甲击中，remove()函数将立即返回；如果外星人侵者被玩家的坦克激光击中，它将创建一个爆炸的动画效果，并设置其自身的 remove 属性。最后调用 hitCallback()：

```
that.remove = function (colliderFlag) {
    if (colliderFlag & SHIELD) {
        return;
    }
    animEffect(x, y, [8], 250, null);
    that.destroy();
    collider.remove();
    that.removed = true;
    hitCallback(points);
};
```

外星人侵者的碰撞宽度将调整与所用的图像帧的尺寸相匹配：

```
if (frame === 2) {
    collisionWidth = 22;
}
else if (frame === 4) {
    collisionWidth = 25;
}
```

在这里，我们创建了一个碰撞对象，并且执行初始更新来设置碰撞对象的位置：

```
collider = SYS_collisionManager.newCollider(ALIEN, 0, collisionWidth, 16,
    that.remove);
collider.update(x, y);
```

move()函数接受两个确定运动方向的参数（dx 和 dy）。在 move()函数，将播放子图的动画，并更新 x 和 y 位置：

```
that.move = function (dx, dy) {
    that.changeImage(frame + animFlag);
    animFlag ^= 1;
    x += dx;
    y += dy;
```

接下来，我们对外星人侵者的垂直位置进行测试，看它是否在护甲之上或者在护甲附近。如果在，那么一个新的碰撞对象将取代旧的，但是这次它是针对护甲测试碰撞，使得碰撞时外星人侵者能够摧毁它们。在执行垂直位置测试之前，我们要保证只有临近护甲的外星人侵者才会执行检查，从而最大限度地减少工作量。

```
    if (!collider.collideeFlags && y >= SHIELD_Y - 16) {
        collider.remove();
        collider = SYS_collisionManager.newCollider(ALIEN, SHIELD,
        collisionWidth, 16, that.remove);
    }
```

更新碰撞对象和子图的位置，检测是否超过了游戏区的水平范围。如果超过，返回true；反之，返回 false：

```
    collider.update(x, y);
    that.draw(x, y);
    if ((dx > 0 && x >= SCREEN_WIDTH - 32 - 16) || (dx < 0 && x <= 16)) {
        return true;
    }
    return false;
};
```

getXY()方法返回外星人侵者的 x 和 y 像素位置：

```
that.getXY = function () {
    return {
        x: x,
        y: y
    };
};
```

返回外星对象的实例：

```
    return that;
};
```

### 3. 外星人管理器

aliensManager 对象比外星人本身有趣的多，它编排外星人以经典的方式移动并且决定哪个外星人将要扔出炸弹。

AliensManager 被传入两个参数：给游戏控制对象发送信息的回调函数，和第一排外星人的 y 坐标。我们设置了各种变量（包括外星人列表），并定义一个命中函数(hitFunc())，每当外星人被激光击中调用此函数：

```
var aliensManager = function (gameCallback, startY) {
    var aliensList = [],
        aliensFireList = [],
        paused = false,
        moveIndex,
        dx = 4,
        dy = 0,
        images = [0, 2, 2, 4, 4],
        changeDir = false,
        waitFire = false,
        scores = [40, 20, 20, 10, 10],
        that,
        hitFunc = function (points) {
            if (!paused) {
                that.pauseAliens(150);
            }
            gameCallback({
                message: 'alienKilled',
                score: points
            });
        };
```

在这里，我们初始化所有的外星人，并且设定它们的图像编号、得分、命中回调函数。我们为初始最低行的外星人设定 canFire 属性为 true。最后我们设置第一移动的外星人为右下角的外星人：

```
for (var y = 0; y < ALIEN_ROWS; y++) {
    for (var x = 0; x < ALIEN_COLUMNS; x++) {
        var anAlien = alien((x * 32) + 16, (y * 32) + startY,
            images[y], scores[y], hitFunc);
        aliensList.push(anAlien);
        if (y == ALIEN_ROWS - 1) {
            aliensList[aliensList.length - 1].canFire = true;
        }
    }
}
moveIndex = aliensList.length - 1;
```

接下来，我们创建一个 aliensManager()的实例：

```
that = {
```

pause()方法允许一整群的外星人在一段设定的时间内保持静态，当一个玩家或者一个外星人被击中时，它将被调用：

```
pauseAliens: function (pauseTime) {
    paused = true;
    setTimeout(function () {
        paused = false;
    }, pauseTime);
},
```

move()函数执行主要的外星人控制逻辑，在每个游戏循环中被调用。这个函数在一个周期内只能移动一个外星人（那个被 moveIndex 引导的外星人）。如果外星人被

暂停，它将会立刻返回。如果没有外星人留下，aliensManger 将会被标记删除，并且将有一条消息发到主游戏，表示所有的外星人都被清除掉了：

```
move: function () {
    if (paused) {
        return;
    }
    if (!aliensList.length) {
        that.removed = true;
        gameCallback({
            message: 'allAliensKilled'
        });
        return;
    }
```

如果当前的外星人已被标记为删除，我们就搜索同一列最低的外星人，将最低的外星人的 canFire 属性设置为 true，这样它就能投弹了。最后，标记为删除的外星人将从外星人列表中移除，moveIndex 也会调整到指向下一个有效的外星人：

```
var anAlien = aliensList[moveIndex];
if (anAlien.removed) {
    for (var i = aliensList.length - 1; i >= 0; i--) {
        if (aliensList[i].getXY().x === anAlien.getXY().x &&
            i !== moveIndex) {
            if (i < moveIndex) {
                aliensList[i].canFire = true;
            }
            break;
        }
    }
    aliensList.splice(moveIndex, 1);
    moveIndex--;
    if (moveIndex === -1) {
        moveIndex = aliensList.length - 1;
    }
    return;
}
```

如果当前外星人的 canFire 属性为 true，它将被添加到一个可能投弹的外星人清单（aliensFireList）中，之后这些外星人将会被随机地选择一个出来投弹：

```
if (anAlien.canFire) {
    aliensFireList.push(anAlien);
}
```

如果这些外星人在沿直线下降，那就不该有水平移动。移动当前的外星人，如果返回值为 true，说明到达了水平边界，我们将设置一个标志（changeDir），表明所有的外星人都将下降得更低，并且改变水平移动的方向：

```
var dx2 = dy ? 0 : dx;
if (anAlien.move(dx2, dy)) {
    changeDir = true;
}
```

如果当前外星人已经达到了和玩家的坦克一样的垂直高度，那就意味着游戏结束：

```
if (anAlien.getXY().y >= TANK_Y) {
    gameCallback({
        message: 'aliensAtBottom'
    });
    return;
}
```

减少当前的 moveIndex 来指向下一个外星人。如果所有的外星人都被移动了，那将发生这些事件：重新设定 moveIndex 的值；如果需要改变水平移动方向（changeDir == true），水平运动方向（dx）将发生翻转，而外星人的下一个运动方向将是垂直向下（dy）；如果当前没有活跃的外星炸弹（waitFire == false），将从射击名单中随机挑选出一个外星人投弹：

```
moveIndex--;
if (moveIndex === -1) {
    moveIndex = aliensList.length - 1;
    dy = 0;
    var coeff = SYS_timeInfo.averageCoeff;
    dx = 4 * (dx < 0 ? -coeff : coeff);
    if (changeDir === true) {
        dx = -dx;
        changeDir = false;
        dy = 16;
    }
    if (!waitFire) {
        var fireAlien = aliensFireList[Math.floor(Math.random() *
            (aliensFireList.length))];
        var xy = fireAlien.getXY();
        alienBomb(xy.x, xy.y, function () {
            waitFire = false;
                });
                aliensFireList = [];
                waitFire = true;
            }
        }
    }
};
```

这里 alienManager 的对象实例将会被添加到处理列表中，并将实例返回到 that 里：

```
    SYS_process.add(that);
    return that;
};
```

## 5.2.7 玩家

这部分涵盖了相对简单的玩家坦克以及激光的行为。

### 1. 坦克

坦克对象被传入一个回调函数，回调函数通知主游戏对象坦克已被击中。在坦克对象中我们声明了一些变量，创建了一个 DHTMLSprite 实例（that），设定图像编号，在坦克的起始位置绘制坦克：

```
var tank = function (gameCallback) {
    var x = ((SCREEN_WIDTH / 2) - 160),
        canFire = true,
        collider,
        waitFireRelease = true,
        that = DHTMLSprite(SYS_spriteParams);
    that.changeImage(6);
    that.draw(x, TANK_Y);
```

move()函数将首先检查左、右方向键，设置合适的水平移动量。我们根据帧率来调整移动幅度，使它在不同的硬件和浏览器组合中的速度保持一致：

```
that.move = function () {
    var dx = keys.left ? -2 : 0;
    dx = keys.right ? 2 : dx;
    x += dx * SYS_timeInfo.coeff;
```

接下来，我们将更新后的坦克水平位置限制在游戏区域内：

```
if (dx > 0 && x >= (SCREEN_WIDTH / 2) + 168) {
    x = (SCREEN_WIDTH / 2) + 168;
}
if (dx < 0 && x <= (SCREEN_WIDTH / 2) - 200) {
    x = (SCREEN_WIDTH / 2) - 200;
}
```

将坦克画在一个新的位置，并更新碰撞对象：

```
that.draw(x, TANK_Y);
collider.update(x, TANK_Y);
```

如果坦克能发射，检查发射键的状态。为防止玩家一直按住发射键来发射激光，我们还要确保玩家是在松开之后再按的发射键：

```
if (canFire) {
    if (keys.fire) {
        if (!waitFireRelease) {
```

如果所有条件已经具备，创建一束坦克激光。传入回调函数，使得在此激光被移除后，坦克可以重新发射激光：

```
        laser(x, TANK_Y+8, function(){canFire = true;} );
        canFire = false;
        waitFireRelease = true;
    }
```

如果玩家没有按住发射键，waitFireRelease 标志将被清除，以确保下次按下发射键

后能发射激光：

```
            } else {
                waitFireRelease = false;
            }
        }
    }; // End of move() method.
```

当坦克被击中时，Move()函数将被调用。它将移除碰撞对象，移除子图，设置坦克的 removed 标记，初始化碰撞的动画效果，并通知主游戏对象坦克已被击中：

```
    that.hit = function () {
        collider.remove();
        that.destroy();
        that.removed = true;
        animEffect(x, TANK_Y, [8], 250, null);
        gameCallback({
            message: 'playerKilled'
        });
    };
```

现在我们设定碰撞对象，坦克的属性将被添加到进程列表中：

```
        collider = SYS_collisionManager.newCollider(PLAYER, ALIEN_BOMB,
            30, 12, that.hit);
        SYS_process.add(that);
    };
```

## 2．激光

激光对象将被给予一个初始位置和当激光被移除调用的回调函数。在激光对象中我们先创建一个 DHTMLSprite 实例：

```
    var laser = function (x, y, callback) {
        var that = DHTMLSprite(SYS_spriteParams);
```

当激光与其他物体发生碰撞时，removed()函数被调用。如果激光与屏幕的顶端、护甲或外星人炸弹相撞，将会生成一个动画效果（一个较小的爆炸子图）。然后激光被移除，在一个小的延迟后（进一步限制玩家发射的速度）调用回调函数：

```
    that.remove = function (collideeFlags) {
        if (collideeFlags & (TOP_OF_SCREEN + SHIELD + ALIEN_BOMB)) {
            animEffect(x, y, [18], 250, null);
        }
        that.destroy();
        collider.remove();
        that.removed = true;
        setTimeout(callback, 200);
    };
```

下面我们创建一个碰撞对象实例，引用 remove()值作为回调函数。然后设置激光图像：

```
    var collider = SYS_collisionManager.newCollider(LASER, ALIEN + ALIEN_BOMB +
        SHIELD + SAUCER, 2, 10, that.remove);
    that.changeImage(7);
```

当更新碰撞对象时，move()函数简单地将激光螺栓往上移动。如果垂直距离 y 达到了游戏区域的顶端，那么 remove 函数就将被调用：

```
    that.move = function () {
        y -= 7 * SYS_timeInfo.coeff;
        that.draw(x, y);
        collider.update(x, y);
        if (y <= -8) {
            that.remove(TOP_OF_SCREEN);
        }
    };
```

激光实例（that）被添加到处理列表中：

```
        SYS_process.add(that);
    };
```

## 5.2.8 护甲

护甲在激光或外星人炸弹攻击下渐渐地被摧毁。如果外星人的高度足够低，它们也能摧毁护甲。每个护甲可以看作是 40 个“砖块”的封装对象：

护甲的位置（x，y）以参数的形式传递：

```
    var shield = function (x, y) {
```

下面定义 shieldBrick 对象，它接受一个位置（x，y）和一个图像编号作为参数。我们使用图像参数初始化 DHTMLSprite：

```
    var shieldBrick = function (x, y, image) {
        var that = DHTMLSprite(SYS_spriteParams),
            collider,
```

如果有任何东西攻击到了护甲砖，hit()函数将被调用：

```
    hit = function () {
        that.destroy();
        collider.remove();
    };
```

我们初始化一个碰撞对象，用先前定义的 hit()函数作为回调函数：

```
        collider = SYS_collisionManager.newCollider(SHIELD, 0, 4, 8, hit);
        that.removed = false;
        that.changeImage(image);
        that.draw(x, y);
        collider.update(x, y);
    },
```

brickLayout[]数组定义了创建一个护甲需要的 shieldBrick 排列和图像编号：

```
    brickLayout = [
        1, 2, 3, 3, 3, 3, 3, 3, 3, 3, 4, 5,
        3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
        3, 3, 3, 6, 7, 0, 0, 8, 9, 3, 3, 3,
        3, 3, 3, 0, 0, 0, 0, 0, 0, 3, 3, 3];
```

遍历 brickLayout[]数组，以 12×4 为基准初始化 shieldBrick。brickLayout[]数组中的 0 表明此位置不需要初始化砖块。相对于传入护甲对象的 x 和 y 参数计算砖块位置。

```
        for (var i = 0; i < brickLayout.length; i++) {
            if (brickLayout[i]) {
                shieldBrick(x + ((i % 12) * 4), y + (Math.floor(i / 12) * 8),
                brickLayout[i] + 8);
            }
        }
    };
```

## 5.2.9 神秘飞碟

神秘飞碟的参数是一个对主游戏对象的回调函数。在飞碟对象中，将计算一个随机的运动方向（dx），并设置合适的开始位置：

```
    var saucer = function (gameCallback) {
        var dx = (Math.floor(Math.random() * 2) * 2) - 1,
            x = 0;
        dx *= 1.25;
        if (dx < 0) {
            x = SCREEN_WIDTH - 32;
        }
```

我们创建了一个 DHTMLSprite 实例，并设置适当的图像编号：

```
    var that = DHTMLSprite(SYS_spriteParams);
    that.changeImage(20);
```

当飞碟到达游戏区的对面时，remove()函数将被调用：

```
    var remove = function () {
        that.destroy();
        collider.remove();
        that.removed = true;
    };
```

当玩家的激光击中飞碟的时候，碰撞系统调用飞碟的命中函数。这个命中函数将返回一个消息（以及飞碟的位置）给主游戏，通知它这个飞碟已经被击中了：

```
    var hit = function () {
        remove();
        gameCallback({
            message: 'saucerHit',
            x: x,
            y: 32
        });
    };
```

我们创建一个以 hit()函数作为回调函数的碰撞对象：

```
var collider = SYS_collisionManager.newCollider(SAUCER, 0, 32, 14, hit);
```

move()函数将飞碟沿 dx 的方向移动，更新其碰撞对象，检查是否达到游戏区域边界：

```
that.move = function () {
    that.draw(x, 32);
    collider.update(x, 32);
    x += dx;
    if (x < 0 || x > SCREEN_WIDTH - 32) {
        remove();
    }
};
```

将飞碟加入到处理列表中：

```
    SYS_process.add(that);
};
```

## 5.2.10 游戏

所有的游戏对象和游戏逻辑都一起捆绑在一个高层的游戏对象里。它执行各种关键任务，比如为所有游戏对象调用 move()函数（通过 process 对象），进行碰撞测试（通过 collisionManager）。通过响应来自游戏对象的信息，它控制了活动的流程——检查何时所有的外星人被击中，何时玩家被击中，何时游戏结束：

```
var game = function () {
```

这里我们声明各种变量，包括标题画面上显示的文字：

```
var time,
    aliens,
    gameState = 'titleScreen',
    aliensStartY,
    lives,
    score = 0,
    highScore = 0,
    extraLifeScore = 0,
    saucerTimeout = 0,
    newTankTimeout,
    newWaveTimeout,
    gameOverFlag = false,
    startText =
        '<div class="message">' +
        '<p>ORBIT ASSAULT</p>' +
        '<p>Press FIRE to Start</p>' +
        '<p>Z - LEFT</p>' +
        '<p>X - RIGHT</p>' +
        '<p>M - FIRE</p>' +
        '<p>EXTRA TANK EVERY 5000 POINTS</p>' +
        '</div>',
```

initShields()函数用来创建 4 个均匀分布的护甲：

```
initShields = function () {
    for (var x = 0; x < 4; x++) {
        shield((SCREEN_WIDTH / 2) - 192 + 12 + (x * 96), SHIELD_Y);
    }
},
```

updateScores()函数首先检查是否应该获得一个额外的坦克，每获得 5 000 分就应获得一个。如果得分超过了最高分就更新最高分。最后在游戏区域写出更新得分、最高分以及剩余坦克数的文本：

```
updateScores = function () {
    if (score - extraLifeScore >= 5000) {
        extraLifeScore += 5000;
        lives++;
    }
    if (!$('#score').length) {
        $("#draw-target").append('<div id="score"></div>' +
            '<div id="lives"></div><div id="highScore"></div>');
    }
    if (score > highScore) {
        highScore = score;
    }
    $('#score').text('SCORE: ' + score);
    $('#highScore').text('HIGH: ' + highScore);
    $('#lives').text('LIVES: ' + lives);
},
```

newSaucer()函数在 5 到 20 秒之间的随机时间间隔内初始化一个新的神秘飞碟：

```
newSaucer = function () {
    clearTimeout(saucerTimeout);
    saucerTimeout = setTimeout(function () {
        saucer(gameCallback);
        newSaucer();
    }, (Math.random() * 5000) + 15000);
},
```

init()函数清除游戏区域内的任何对象，并且初始化一个游戏对象处理器（SYS_process）、一个碰撞管理器（SYS_collisionManager）、一个外星人管理器（aliens）。它安排 2 秒之后初始化玩家的坦克，启动神秘飞碟的随机定时器，并更新得分、最高分以及坦克数的文本显示：

```
init = function () {
    $("#draw-target").children().remove();
    SYS_process = processor();
    SYS_collisionManager = collisionManager();
    aliens = aliensManager(gameCallback, aliensStartY);
    setTimeout(function () {
        tank(gameCallback);
    }, 2000);
    initShields();
```

```
        newSaucer();
        updateScores();
    },
```

gameOver()函数清空所有计时器，以防止再出现坦克、外星人，或者飞碟。最后在通常的标题画面上显示 Game Over 的信息：

```
    gameOver =  function() {
        gameOverFlag = true;
        clearTimeout(newTankTimeout);
        clearTimeout(newWaveTimeout);
        clearTimeout(saucerTimeout);
        setTimeout(function () {
            $("#draw-target").children().remove();
            $("#draw-target").append('<div class="message">' +
                '<p>*** GAME OVER ***</p></div>' + startText);
            gameState = 'titleScreen';
        }, 2000);
    },
```

gameCallBack()函数响应从游戏对象发送的消息。一个 switch-case 根据收到的消息来进行合适的动作。如果游戏已经结束，函数则返回：

```
    gameCallback = function (messageObj) {
        if (gameOverFlag) {
            return;
        }
        switch (messageObj.message) {
```

当所有的外星人都被击中，更新分数：

```
    case 'alienKilled':
        score += messageObj.score;
        updateScores();
        break;
```

当玩家击中神秘的飞碟，他将被赠与一个随机的得分：50、100 或者 150 分。 对于此分数呈现合适的动画效果：

```
    case 'saucerHit':
        var pts = Math.floor((Math.random() * 3) + 1);
        score += pts * 50;
        updateScores();
        animEffect(messageObj.x, messageObj.y, [pts + 20], 500, null);
        break;
```

如果玩家的坦克被击中，外星人被暂停并且坦克个数减一。如果没有剩余坦克，游戏结束；否则，安排一个新的坦克在 2 秒钟内再现：

```
    case 'playerKilled':
        aliens.pauseAliens(2500);
        lives--;
        updateScores();
        if (!lives) {
           gameOver();
        } else {
```

```
            newTankTimeout = setTimeout(function () {
                tank(gameCallback);
            }, 2000);
        }
        break;
```

当所有的外星人都被击中，下一波的起始位置将比前一波的起始位置低 32 像素。安排在 2 秒钟内启动新一波：

```
case 'allAliensKilled':
    if (aliensStartY < 160) {
    aliensStartY += 32;
}
newWaveTimeout = setTimeout(function () {
    init();
}, 2000);
break;
```

如果有外星人到达游戏区的底部，那么游戏结束：

```
        case 'aliensAtBottom':
           gameOver();
           break;
        }
    },
```

gameLoop()函数每 15 毫秒被调用一次，它有两种运行状态："playing"或"titleScreen"。在 playing 状态下，要处理游戏对象并检测碰撞；titleScreen 状态在一个循环中，当发射键被按下的时候，准备开始游戏。如果需要的话，很容易添加更多的动画或者其他效果到 titleScreen 状态中：

```
gameLoop = function () {
    switch (gameState) {
    case 'playing':
        SYS_timeInfo = time.getInfo();
        SYS_process.process();
        SYS_collisionManager.checkCollisions();
        break;

    case 'titleScreen':
```

如果玩家在 titleScreen 状态下按发射键，积分、坦克个数以及外星人的初始位置将会被重新设置；游戏的状态将被设为"playing"，并且初始化一个新的游戏：

```
        if (keys.fire) {
            gameOverFlag = false;
            time = timeInfo(60);
            keys.fire = 0;
            lives = 3;
            score = 0;
            extraLifeScore = 0;
            aliensStartY = 64;
            gameState = 'playing';
            init();
        }
```

```
    }
    setTimeout(gameLoop, 15);
}();
```

显示标题画面开头的文本，主循环开始：

```
    $("#draw-target").append(startText);
    gameLoop();
}();
```

## 5.2.11 所有代码

轨道攻击 HTML 页面（见例 5-1），包含了 JavaScript 源代码、少量的 CSS，以及游戏区域元素（绘制目标）。

例 5-1 轨道攻击页面代码

```
<!DOCTYPE html>
<html>
<head>
<title>Orbit Assualt</title>
<script type="text/javascript"
    src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js">
</script>
<style type="text/css">

#draw-target {
    width:480px;
    height:384px;
    background-color:#000;
    position:relative;
    color:#FFF;
    font-size:16px;
    font-family:"Courier New", Courier, monospace;
    font-weight:bold;
    letter-spacing:1px;
}
.message {
    margin-left: auto;
    margin-right: auto;
    padding-top:32px;
    text-align:center;
}
#score {
    position:absolute;
    top:8px;
    left:16px;
}
#highScore {
    position:absolute;
    top:8px;
    right:16px;
}
#lives {
    margin-left: auto;
```

```
    margin-right: auto;
    padding-top:8px;
    text-align:center;
}
</style>
<script type="text/javascript">
    $(document).ready(function() {

        // For IE6
        try {
    document.execCommand("BackgroundImageCache", false, true);
} catch(err) {};

var PLAYER = 1,
    LASER = 2,
    ALIEN = 4,
    ALIEN_BOMB = 8,
    /*** CODE REMOVED FOR CONCISENESS ***/
    };

var processor = function () {
    /*** CODE REMOVED FOR CONCISENESS ***/
};

var collisionManager = function () {
    /*** CODE REMOVED FOR CONCISENESS ***/
};

var DHTMLSprite = function (params) {
    /*** CODE REMOVED FOR CONCISENESS ***/
};

var timeInfo = function (goalFPS) {
    /*** CODE REMOVED FOR CONCISENESS ***/
};

var keys = function () {
    /*** CODE REMOVED FOR CONCISENESS ***/
}();

var animEffect = function (x, y, imageList, timeout) {
    /*** CODE REMOVED FOR CONCISENESS ***/
};

var alien = function (x, y, frame, points, hitCallback) {
    /*** CODE REMOVED FOR CONCISENESS ***/
};
  // aliens
var aliensManager = function (gameCallback, startY) {
    /*** CODE REMOVED FOR CONCISENESS ***/
};

var laser = function (x, y, callback) {
    /*** CODE REMOVED FOR CONCISENESS ***/
};
```

```
var alienBomb = function (x, y, removedCallback) {
    /*** CODE REMOVED FOR CONCISENESS ***/
};

var tank = function (gameCallback) {
    /*** CODE REMOVED FOR CONCISENESS ***/
};

var shield = function (x, y) {
            /*** CODE REMOVED FOR CONCISENESS ***/
        };

        var saucer = function (gameCallback) {
            /*** CODE REMOVED FOR CONCISENESS ***/
        };

        var game = function () {
            /*** CODE REMOVED FOR CONCISENESS ***/
        }();

    });
</script>
</head>
<body>
    <div id="draw-target"> </div>
</body>
</html>
```

# 第 6 章

# HTML5 画布

HTML5 最吸引人的一个特性就是画布（Canvas）元素。画布的形式是在页面上类似于 div 的一个矩形区域，使你可以用 JavaScript 绘制复杂的图形。它最初是由苹果为在 Mac 操作系统上用 Safari 浏览器渲染用户界面组件和其他图形而开发的。苹果公司将画布相关的专利以 WWW 联盟（W3C）的免版税许可条款发布。也就是说，苹果将为在 W3C HTML 推荐范围内的画布提供免版税的许可。

本章介绍了画布的基本知识并用它来实现多种实际应用。除了阅读本章之外，你不妨考虑下面的书籍进一步增加这方面的知识：

- *Canvas Pocket Reference* by David Flanagan (O'Reilly; http://oreilly.com/catalog/0636920016045)
- *HTML5 Canvas* by Steve Fulton and Jeff Fulton (O'Reilly; http://oreilly.com/catalog/0636920013327)

画布是一个底层、立即模式的应用程序编程接口（API）：

底层

> 画布提供了快速但相当基本的功能集。例如，矩形是仅有的原生形状。不过你可以通过 JavaScript 编程来增强其功能。

立即模式

> 当画布绘图指令被调用时，就立即被执行。这和 SVG 在绘图之前使用中间数据结构保存层级图形对象是不同的，画布没有这样的中间数据结构。这意味着，

可以加入无限层的绘图操作而不影响性能——这对诸如像素图艺术包或其他精细的“层次”特效是特别完美的。

下面的画布示例会显示一个蓝色矩形：

```
<!DOCTYPE html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script type="text/javascript"
        src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js">
    </script>
    <script>
        $(document).ready(function() {
            var a_canvas = $("#a_canvas")[0],
                ctx = a_canvas.getContext("2d");
            ctx.fillStyle = "rgb(0,0,255)";
            ctx.fillRect(50, 25, 150, 100);
        });
    </script>
</head>
<body>
    <canvas id="a_canvas">
    </canvas>
</body>
</html>
```

（是否使用 jQuery 取决于个人的喜好）

画布的底层特性使其整洁和简单，而其速度非常适合于动态图形应用。任何熟悉位图图形编程的人使用画布时都会有宾至如归的感觉。

## 6.1 画布的支持

大部分流行的浏览器都支持画布元素，包括 Firefox、Chrome、Opera 和 Safari。2010 年 7 月 1 日，微软通过 IE9 开发博客宣布其最新的浏览器会支持画布。事实上，该公司甚至为画布支持提供了硬件加速。这个相对低调的声明掩盖了它的重要性：IE 仍然持有大部分的浏览器市场份额，它对画布提供的支持对使用画布的开发人员是个好消息。然而 IE9 仅适用于 Windows Vista 和 Windows7，它不支持 Windows XP。所有的 Windows 用户都能享受画布，还需要一段时间。

## 6.2 位图、矢量图，或两者兼而有之？

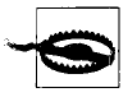
画布为不同的应用，设计了一个小而精的矢量图和位图命令集。两者之间的区别是什么？

矢量图

矢量图形由直线和曲线的数学表示定义。你可以填充矢量形状或/和描绘其轮廓。矢量图形的关键优势在于它们可以缩放到任意大小而不损失质量：边缘和细节依旧锋利。矢量图最适合单色或渐变区域面积较大、细节密度较小的图像。最典型的有：图表、图形、旗帜、线路图和卡通风格的图像。因为其数学特性，JavaScript 操作矢量图特别方便。

位图

位图图像（如无所不在的 JPEG 格式）是不同颜色的像素组成的网格。它们不能很好地进行缩放，当放大时将看到明显的方块，而缩小时将损失信息。这是因为单个像素不是被放大了就是被丢失了。有些画布实现可以通过使用模糊滤波来降低这种不良效应。位图最适用于有着大量细节的摄影风格的图像。



**警告**

无论如何生成图像，画布最终可视的输出结果始终是位图。如果你要利用矢量缩放的优势，你需要使用矢量图命令在新的尺度下重绘图像。仅仅使用浏览器交互或 CSS 来放大画布，其效果和放大位图图像一样：会有块状/模糊效应。

## 6.3 画布限制

使用画布时有一些限制，其中一些和其底层特点有关：

- 缺乏视觉元素的数据结构，意味着你必须在 JavaScript 中创建自己的对象，来更新非静态图形元素的位置和其他属性。
- 同样，你不能给画布中绘制的元素添加事件（如鼠标点击），因为它们并不是有形的实体，而只是瞬态的绘图操作。你必须通过编程来实现这样的功能。
- 充分利用画布必须有良好的 JavaScript 知识。

## 6.4 画布与 SVG 的对比

有些人最初对苹果创建另一个浏览器图形标准持保留意见，他们可能认为 SVG 已经足够了。从表面上看 SVG 和画布提供类似的图形能力，但它们有一个根本的区

别：SVG 是一个高层的、基于 XML 的标记语言，可以通过创建 XML 元素属性来定义图像；而画布则提供了可以直接从 JavaScript 访问的绘图 API。

你可以使用任何文本编辑器手动创建 SVG XML，或将它从 Adobe Illustrator 或 Inkscape 等绘图软件中导出。以下的 SVG 例子显示了一个蓝色矩形：

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE svg PUBLIC
    "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="100%" height="100%" version="1.1" xmlns="http://www.w3.org/2000/svg">
    <rect id='a_rectangle' width="300" height="100" style="fill:rgb(0,0,255)" />
</svg>
```

要通过 JavaScript 操纵这个矩形，你需要访问 a_rectangle 元素并适当调整其属性。听起来很熟悉吗？就像 HTML 中那样，我们通过一个类似 DOM 的结构去定义视觉效果。试想假如我们需要 1000 个矩形？没错，我们必须插入 1000 矩形元素到 XML 中。这种方式对需要更多编程的动态图形来说，不是特别有效或直观。

不过，SVG 不用 JavaScript 就能带给你绘制和动画功能，而且有大量设计工具供你编辑 SVG 图形。由于现在 IE9 中也提供了基本的支持，当需要矢量图时，SVG 是一个不错的解决方案。维基百科等网站就广泛使用 SVG 作为插图。

## 6.5 画布与 Adobe Flash 的对比

大多数网民都熟悉 Adobe Flash。大量的在线广告内容、视频和游戏都使用了 Flash。事实上，有很多网站是完全用 Flash 创建的。这是一个可以追溯到 1996 年的成熟插件，现在几乎所有系统上都安装 Flash。不过 Flash 也有自身的问题，HTML5（包括画布）的发展，可能预示着互联网富内容创建会有翻天覆地的变化。

- Flash 是一个由 Adobe 公司所拥有的专有格式。播放 Flash 内容不收取费用，但开发 Flash 内容则需要购买相应的制作软件。用 Flash 这样一个封闭系统来创建 Web 内容，和 Web 的自由开放化进程是不协调的。
- Flash 在桌面机上根深蒂固。而在苹果公司的流行移动设备上，如 iPod、iPhone 和 iPad，浏览器是不支持 Flash 的。苹果公司在 2010 年 9 月做了少许妥协，允许软件以 Flash 开发，并封装为原生应用。
- 尽管一些移动设备可以用 Flash Lite，而且 Android 2.2 设备支持 Flash 10.1，但移动用户对由 Flash 的依赖较小。因为他们可以很容易下载原生应用和游戏。

- YouTube、Facebook 和哥伦比亚广播公司等热门网站，现在开始以 HTML5 兼容的格式（H.264）提供它们的视频内容。

HTML5 和 Flash 的辩论，会引起很多争议。希望 Flash 长存的资深 Flash 开发人员自然会质疑 HTML5 能否取代 Flash，而开放 Web 的支持者会说 HTML5 使 Flash 显得多余。

现实中，Flash 很快消失是不太可能的。它太根深蒂固了，而 HTML5 在各个方面进展缓慢。然而，考虑到跨浏览器支持、熟悉和免费的开发工具，只有最乐观的 Flash 开发人员会忽略 HTML5。不过，随着 JavaScript 性能的提升和工具库（如 jQuery）的发展，以及考虑到画布等因素，已没有什么理由去以 100%-Flash 的方式去构建网站了，估计用不了多久这种网站就会停止出现。

## 6.6 画布导出器

画布完全由 JavaScript 控制，因此了解 JavaScript 是充分利用画布的前提。基于标记语言的方式无法来访问画布功能。不过倒是出现了画布导出器和转换器，它们可以生成绘制画布需要的 JavaScript 代码。这对 JavaScript 能力较弱的设计师来说是个极好的消息，对程序员来说也是如此，因为通过输入画布命令的方式手动创建精致的矢量艺术是枯燥且易出错的。

Adobe Flash CS5+（http://www.adobe.com/products/flash.html）

> Adobe 的 Flash CS5+有一个画布导出器可以将部分 Flash 导出为 JavaScript 画布源代码。这对想兼顾 Flash 和画布的开发者很有用。然而，由于该解决方案需要购买 Flash 创作工具，它对仅仅想开发画布的人来说可能不太划算。

Canvg（http://code.google.com/p/canvg/）

> Canvg（如图 6-1 所示）是一个用画布绘制 SVG 数据的 JavaScript 库。不幸的是，其画布 JavaScript 语句没有以任何形式保存下来，因此你必须始终包含 Canvg 库来绘制 SVG。

SVG-to-Canvas（http://www.professorcloud.com/svg-to-canvas/）

> 这个在线工具将静态的 SVG 转换为 JavaScript 的画布函数。它使用了一个修改版的 Canvg 库。

AI-Canvas（http://visitmix.com/labs/ai2canvas/）

这个复杂的 Adobe Illustrator 插件（如图 6-2 所示）可以转换静态图片和动画。如果插件遇到不能转换的图像元素，它会把这些元素转换成位图。所有这些图像元素都被转化为可以进一步修改的画布 JavaScript 函数。

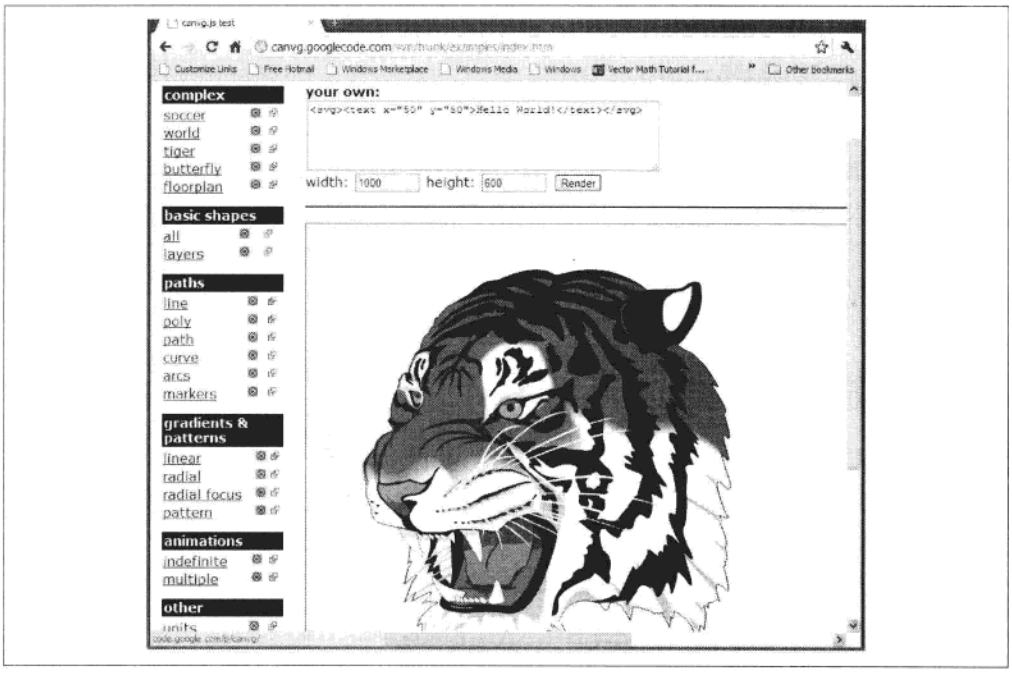



图 6-1　Canvg

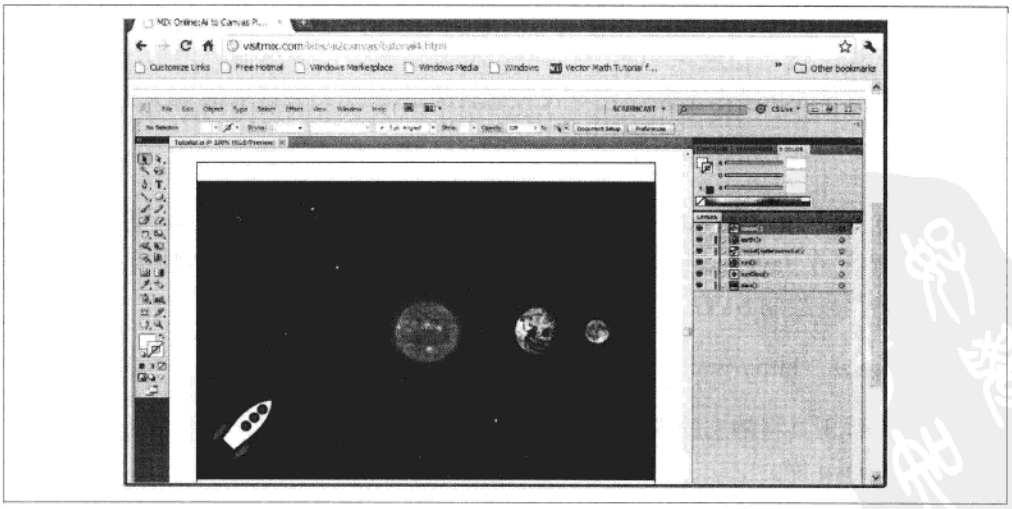


图 6-2　AI-Canvas 可以处理动画

## 6.7 画布绘制基础

下面的九节将讨论基本的画布绘图命令。

### 6.7.1 画布元素

在网页中插入画布元素和插入任何其他 HTML 元素没有什么不同。

```
<canvas id = 'mycanvas' width = 512 height = 384>
    Fallback content
</canvas>
```

如果你不指定任何宽度或高度属性，默认大小为 300×150 像素。可以但不推荐通过 CSS（例如，宽度：50%）改变画布大小。输出有可能是被扭曲或被缩放的，这取决于浏览器的实现。但是，你可以用 CSS 设置边框、边距和背景颜色，虽然这绝不会影响绘制到画布内容本身。坐标系统默认左上角为原点（0，0），因此绘制在坐标（10，15）的图案将定位在从左往右第 10 像素，从上往下第 15 个像素。

如果浏览器不支持画布，将显示开始和结束之间的替代内容（fallback content）。理想的情况下，替代内容应该是画布所显示数据常规的文本或 HTML 表示。例如，画布中可能显示饼图，替代内容会显示一个普通表格。有的情况下，替代内容根本无法取代画布；游戏和绘图应用程序没有对应的文本或 HTML 表示。在这种情况下，替代内容应显示一个有用的信息，向用户解释：画布不可用，浏览器应升级。

单独放到页面的画布没有给我们任何的功能，它必须由 JavaScript 控制才能做些有用的事。你很少会看到没有 id 属性的画布，因为 JavaScript 代码通常用 id 属性来识别画布。通常情况下，JavaScript 将这样得到画布的“句柄”变量：

```
var canvas = document.getElementById('mycanvas');
// Or, using jQuery:
var canvas = $('#mycanvas')[0];
```

### 6.7.2 绘图环境

我们必须从画布获得一个“绘图环境”后才可以使用绘图命令：

```
var canvas = document.getElementById('mycanvas');
var ctx = canvas.getContext('2d');
```

虽然不是正式的推荐，不过在画布例子代码中你会经常看到用 ctx 来代表绘图环境。

**提示**

画布还提供了一个 3D 绘图环境，使你可以访问目前处于试验阶段的 WebGL 接口。WebGL 基于 OpenGL ES 2.0 的标准(OpenGL 的削减版本)，并通过 JavaScript 提供 3D 图形处理能力。在大多数浏览器的开发版本中都支持。OpenGL 实际是一个的底层函数集，你仍然需要做大量的工作来创建一个 3D 应用程序。

在 Web 社区曾经有人怀疑 JavaScript 是否能在较复杂的 3D 场景中管理对象的层次；不管这些对象是不是由 WebGL 绘制，管理一个 3D 应用或游戏需要进行大量的计算。不过随着 JavaScript 性能的不断改善，大家对 JavaScript 越来越有信心，而且出现各种更高层的 3D 库，可以简化 3D 应用开发。所有这些库都是建立在 WebGL 之上的:

- O3D（原本是一个插件，但现在是一个 JavaScript 库）
- GLGE
- C3DL
- SpiderGL
- SceneJS
- Processing.js

## 6.7.3 绘制矩形

画布内置的绘图形状非常有限，实际上只有矩形而已：

```
// Draw a 100 by 150 pixel filled rectangle at coordinate (10,10).
ctx.fillRect(10,10,100,150);

// Draw a 100 by 150 pixel outlined rectangle at coordinate (10,10).
ctx.strokeRect(10,10,100,150);

// Clear a 100 by 150 pixel rectangle at coordinate (10,10).
ctx.clearRect(10,10,100,150);
```

不过这个限制不算大问题，因为我们可以用直线和曲线组合定义的路径来创建所有其他形状。

## 6.7.4 绘制直线和曲线的路径

路径定义可以填充和/或使用大纲描边的形状。画布包括以下功能执行路径绘制：

| 函　数 | 描　述 |
| --- | --- |
| beginPath() | 开始新路径 |
| moveTo() | 设置路径的起始位置 |

续表

| 函　数 | 描　述 |
| --- | --- |
| LineTo() | 定义从当前位置开始的直线 |
| arc() | 定义弧 |
| acrTo() | 定义从当前位置开始的弧 |
| quadraticCurveTo() | 定义从当前位置开始的二次曲线 |
| bezierCurveTo() | 定义从当前位置开始的贝塞尔曲线 |
| closePath() | 结束路径 |
| stroke() | 勾画路径 |

需要注意的是，“to”命令（lineTo()、bezierCurveTo()等）的结束位置也定义了下一个“to”命令的开始位置。你可以将“to”命令想象成用笔在纸上连续地画线（不离开纸面）。moveTo()命令则使你将笔离开纸面，并从其他地方重新开始画。

下面的示例使用线在左上角绘制一个填充三角形和描边三角形（如图 6-3 所示），假设画布尺寸为 500×500 像素：

```
// Draw a filled triangle, top-left.
ctx.beginPath();
ctx.moveTo(20,20);
ctx.lineTo(470,20);
ctx.lineTo(20,470);
ctx.fill();
    // Draw a stroked triangle, bottom-right.
ctx.beginPath();
ctx.moveTo(480,30);
ctx.lineTo(480,480);
ctx.lineTo(30,480);
ctx.closePath();
ctx.stroke();
```

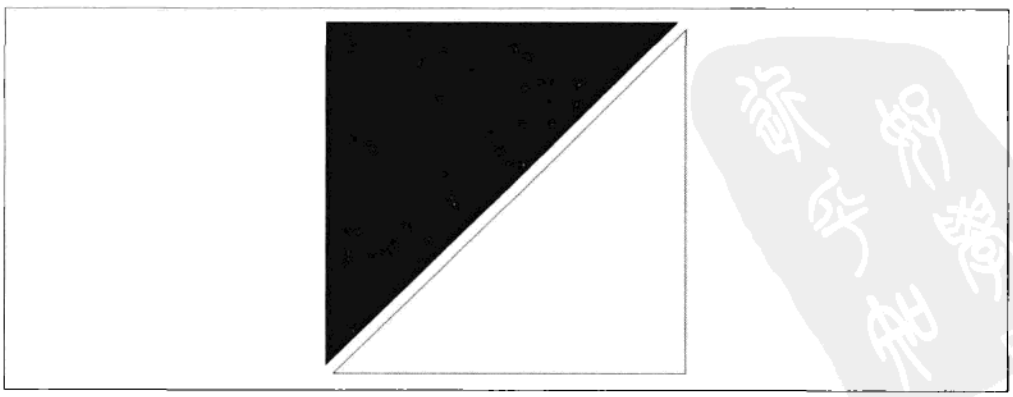


图 6-3　填充和描边三角形

注意你不需要为填充三角形执行 closePath()命令，因为 fill()自动关闭路径。

**提示**

画布允许你指定分数像素位置。你可能觉得这很奇怪，因为像素是不能分割的单元元素。实际上画布是使用了抗锯齿技术给人以分数像素位置存在的假象。这可以使视觉上边缘更干净，移动更平滑，尤其当移动速度较慢时。

你可以使用 arc()命令来绘制圆，或圆的部分：

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

参数如下：

x,y

圆心位置。

radius

像素半径。

startAngle，endAngle

绘图将“横扫”这两个角度之间。角度以弧度定义，2π 弧度（约 6.283）相当于 360°。

antiClockwise

绘制弧线的方向。

下面是弧度转换所需的计算：

```
var radians = degrees * Math.PI / 180;
var degrees = radians * 180 / Math.PI;
```

下面的代码绘制了两排圆，每个圆的开始角度为 0 弧度，endAngle 逐渐增加。上一行以顺时针绘制，下一行以逆时针绘制（如图 6-4 所示）。

```
var endAngle = 0.0;
for (var x = 50; x < 500; x += 100) {
    ctx.beginPath();
    ctx.moveTo(x, 190);
    endAngle += (2 * Math.PI) / 5;
    ctx.arc(x, 190, 50, 0, endAngle, false);
    ctx.fill();
}
endAngle = 0.0;
for (x = 50; x < 500; x += 100) {
```

```
        ctx.beginPath();
        ctx.moveTo(x, 310);
        endAngle += (2 * Math.PI) / 5;
        ctx.arc(x, 310, 50, 0, endAngle, true);
        ctx.fill();
    }
```

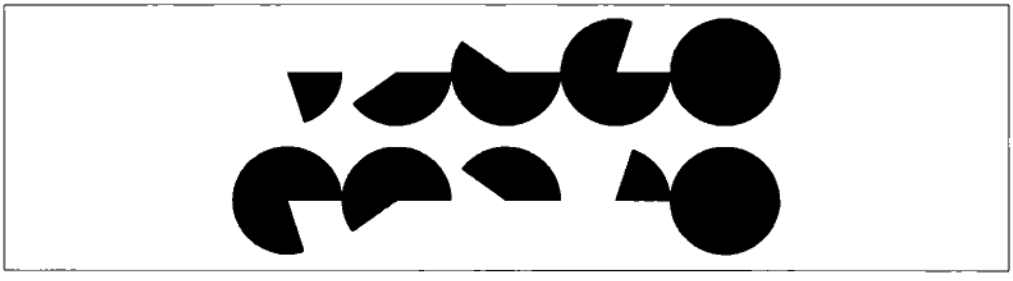


图 6-4　结束角度递增的弧，第一排是顺时针，第二排是逆时针

**提示**

如果没有使用 moveto()来设置开始位置，将从上个弧的结束位置开始画新的弧。

arcTo()命令和 arc()命令类似，但是以不同的方式指定曲线：

```
arcTo(x1,y1, x2,y2, radius);
```

曲线由两条直线定义，第一条直线是从当前位置到第一个点（x1，y1），第二条直线是从点（x1，y1）到点（x2，y2）。这样定义一条曲线是为了方便创建直线之间的圆角。曲线将占据两条直线相交的角。

下面的函数绘制（w，h）大小的圆角矩形。圆角的半径由参数 cr 定义。

```
var drawRoundedRect = function (ctx, x, y, w, h, cr) {
    ctx.beginPath();
    ctx.moveTo(x + w / 2, y);              // Start in the middle of the top edge.
    ctx.arcTo(x + w, y, x + w, y + h, cr); // Top edge and upper-right corner.
    ctx.arcTo(x + w, y + h, x, y + h, cr); // Right edge and lower-right corner.
    ctx.arcTo(x, y + h, x, y, cr);         // Bottom edge and lower-left corner
    ctx.arcTo(x, y, x + w, y, cr);         // Left edge and upper-left corner.
    ctx.closePath();
    ctx.stroke();
};
```

图 6-5 显示了使用不同圆角半径（从 0 开始，依次增加 2π 弧度）调用此函数的结果。

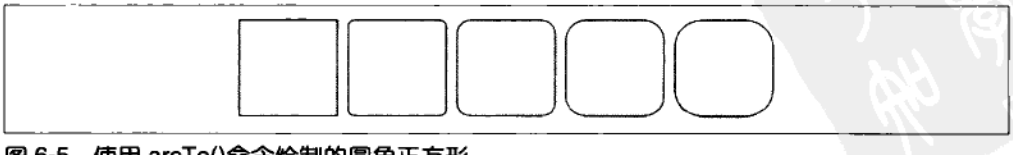


图 6-5　使用 arcTo()命令绘制的圆角正方形

以下的页面代码显示了如何在循环内调用 drawRoundedRect()给出图 6-5 所示的输出：

```
<!DOCTYPE html>
<html>

    <head>
        <title>
            Canvas Rounded Rectangles
        </title>
        <script type="text/javascript">
            window.onload = function() {
                var canvas = document.getElementById('mycanvas');
                var ctx = canvas.getContext('2d');

                var drawRoundedRect = function(ctx, x, y, w, h, cr) {
                        ctx.beginPath();
                        ctx.moveTo(x + w / 2, y);
                        ctx.arcTo(x + w, y, x + w, y + h, cr);
                        ctx.arcTo(x + w, y + h, x, y + h, cr);
                        ctx.arcTo(x, y + h, x, y, cr);
                        ctx.arcTo(x, y, x + w, y, cr);
                        ctx.closePath();
                        ctx.stroke();
                    };

                var cr = 0;
                for (x = 0; x < 500; x += 100) {
                    drawRoundedRect(ctx, x + 5,
                        ctx.canvas.height / 2 - 45, 90, 90, cr);
                    cr += Math.PI * 2;
                }

            };
        </script>
        <style type="text/css">
            #mycanvas {border:1px solid;}
        </style>
    </head>

    <body>
        <canvas id="mycanvas" width=5 00, height=5 00>
        </canvas>
    </body>

</html>
```

quadtraticCurveTo()和 bezierCurveTo()命令使我们绘制一个或两个控制点的曲线。控制点可以使曲线弯曲，从而得到 arc()和 arcTo()命令不能绘制的非对称曲线。这种类型的曲线经常可以在 Photoshop、Freehand 和 Inkscape 等矢量绘图工具中见到。在 JavaScript 中使用这些曲线可能比较棘手，因为我们不能直接看到控制点的位置和它们对曲线的效果。

以下页面代码分别在画布顶部和底部显示了二次曲线和贝塞尔曲线（如图 6-6 所示）。它还显示了可以用鼠标拖动的控制点，使用了 jQuery UI 的“可拖动”功能来移动控制点。请注意控制点实际上是普通的 div 元素，而不是画布路径。以这种方式组合画布和普通 DOM 元素不仅完全合法，而且非常有用：

```
<!DOCTYPE html>
<html>

    <head>
        <script type="text/javascript"
            src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js">
        </script>
        <script type="text/javascript"
            src="http://ajax.googleapis.com/ajax/libs/jqueryui/1.8.0/jquery-ui.min.js">
        </script>
        <script type="text/javascript">
            $(function() {
                var canvas = document.getElementById('mycanvas');
                var ctx = canvas.getContext('2d');
            $('.dragger').draggable({
                cursor: 'crosshair'
            });
            // Trapping the 'mousedown' event and returning false
            // prevents the text select caret from appearing.
            $('.dragger').bind("mousedown", function() {
                return false;
            });
            $('.dragger').bind("drag", function() {

                ctx.clearRect(0, 0, canvas.width, canvas.height);
                var canvasX = $(canvas).position().left,
                    canvasY = $(canvas).position().top,
                    cpx1, cpy1, cpx2, cpy2, $dragr = $('#dragger1');
                // The control point positions are made relative to the canvas,
                // although this calculation is not strictly necessary for
                // this demonstration, as the canvas is in the top-left of the
                // page.
                cpx1 = $dragr.position().left - canvasX;
                cpy1 = $dragr.position().top - canvasY;

                // Draw the quadratic curve (one control point).
                ctx.beginPath();
                ctx.moveTo(50, 150);
                ctx.quadraticCurveTo(cpx1, cpy1, 450, 150);
                ctx.closePath();
                ctx.stroke();

                // Get the position of the other two control points.
                $dragr = $('#dragger2');
                cpx1 = $dragr.position().left - canvasX;
                cpy1 = $dragr.position().top - canvasY;
                $dragr = $('#dragger3');
                cpx2 = $dragr.position().left - canvasX;
                cpy2 = $dragr.position().top - canvasY;
```

```
                // Draw the Bezier curve (two control points).
                ctx.beginPath();
                ctx.moveTo(50, 350);
                ctx.bezierCurveTo(cpx1, cpy1, cpx2, cpy2, 450, 350);
                ctx.closePath();
                ctx.stroke();
            });

            // Trigger an initial drag event so the curves are drawn.
            $('.dragger').trigger("drag");

        });
    </script>
    <style type="text/css">
        .dragger {width:10px; height:10px;z-index:1}
        #mycanvas {border:1px solid;position:absolute;top:0px;}
    </style>
</head>
    <body style="position:relative;">
        <div class="dragger" id="dragger1" style="background-color:#f00;">
        </div>
        <div class="dragger" id="dragger2" style="background-color:#0f0;">
        </div>
        <div class="dragger" id="dragger3" style="background-color:#00f;">
        </div>
        <canvas id="mycanvas" width=500, height=500>
        </canvas>
    </body>
</html>
```

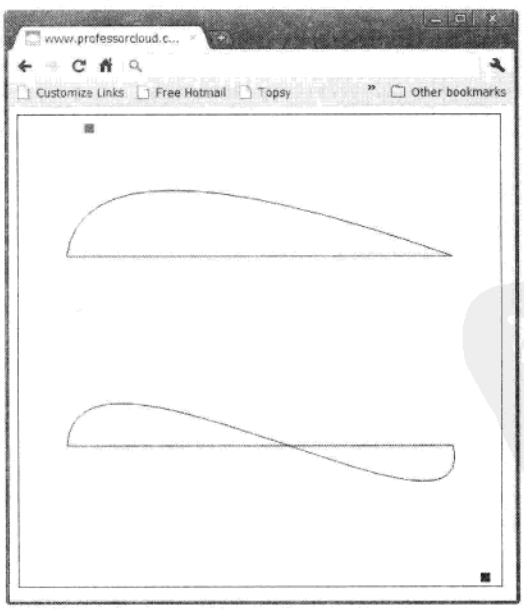



图 6-6　一个控制点的二次曲线（顶部），两个控制点的贝塞尔曲线（底部）

## 6.7.5 绘制位图图像

我们可以用 drawImage()命令绘制位图图像。这个命令可以有 3、5 或 9 个参数。在所有情况下，第一个参数指定图像源以提供绘制的像素数据。图像源可以是用 image()函数载入的图像、普通的<img>标签、甚至是另一个画布或<video>标签。这种指定图像源的灵活性为你提供了巨大的创造潜力。例如，图 6-7 中为制造“爆炸”效果使用<video>标签作为图像源，而图 6-8 用一个大图的随机部分创建自然的星云动画。

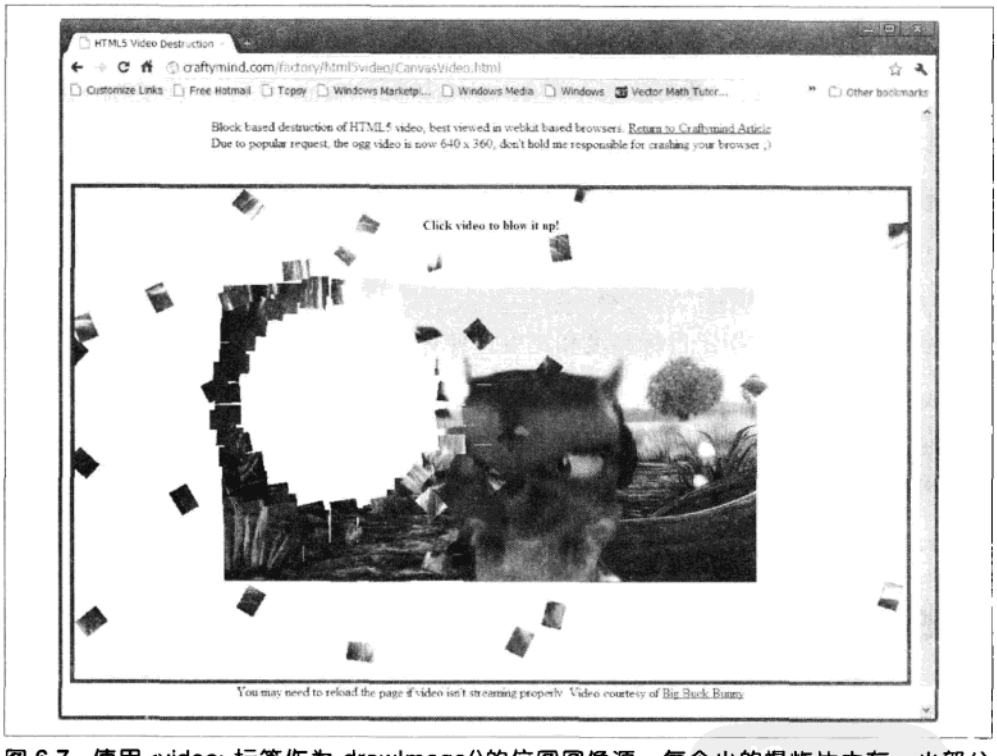
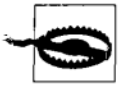



**图 6-7　使用<video>标签作为 drawImage()的位图图像源；每个小的爆炸片内有一小部分视频**



**警告**

如果使用 drawImage()时遇到性能问题，确保图像源是另一个画布标签可能是有益的。这防止了某些浏览器上的图像转换开销。例如，图 6-7 中的视频“爆炸”效果将视频图像复制到画布元素，再使用 drawImage()分为小片。

图 6-8　叠加和放大大图中的随机部分来创建一个自然的星云动画特效

3 个参数版本的 drawImage()最容易使用，它只简单将图像源复制到画布的（x，y）坐标。位图的宽度和高度由源位图本身决定：

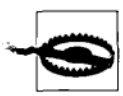
```
drawImage(source, x, y);
```

5 个参数版本允许你指定目标高度和宽度，使你能够缩放图像到所需的大小：

```
drawImage(source, x, y, width, height);
```

9 个参数版本允许你复制图像源的一部分，其中参数 2～5 指定源图像中的源矩形块，参数 6～9 指定内绘制在画布上的目标矩形：

```
drawImage(source, sx, sy, swidth, sheight, x, y, width, height);
```



**警告**

如果你使用 drawImage()分数像素位置，有些浏览器（特别是 Firefox 和 Opera）可能遭受严重的性能损失和其他奇怪的故障。为了避免这些问题，确保将位置四舍五入为整数形式：

```
Math.floor(x)
```

或

```
(x>>0)
```

## 6.7.6　颜色、描边和填充

在前面的例子中，我们使用了 stroke()命令来创建一个默认黑色的 1 像素宽的路径轮廓。你可以使用 lineWidth 和 strokeStyle 属性更改轮廓的风格，并用 fillStype 属

性指定内部填充颜色。下面是一个加入这些属性的圆角矩形代码（如图 6-9 所示）：

```
var drawRoundedRect = function (ctx, x, y, w, h, cr) {
    ctx.beginPath();
    ctx.moveTo(x + w / 2, y);              // Start in the middle of the top edge.
    ctx.arcTo(x + w, y, x + w, y + h, cr); // Top edge and upper-right corner.
    ctx.arcTo(x + w, y + h, x, y + h, cr); // Right edge and lower-right corner.
    ctx.arcTo(x, y + h, x, y, cr);         // Bottom edge and lower-left corner
    ctx.arcTo(x, y, x + w, y, cr);         // Left edge and upper-left corner.
    ctx.closePath();
    ctx.strokeStyle = '#f00';              // Set stroke color to bright red.
    ctx.lineWidth = 4;                     // Set line width to 4 pixels.
    ctx.stroke();
    ctx.fillStyle = '#0f0';                // Here we specify a green fill.
    ctx.fill();
};
```

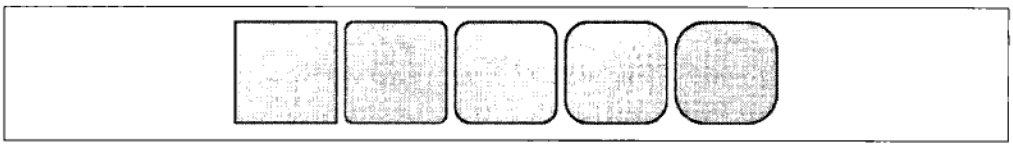
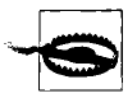


图 6-9　设置为 lineWidth=4，strokeStyle = '#f00'及 fillStyle = '#0f0'

请注意描边比指定的 4 个像素薄。这是因为描边以路径为中心，而内部的两个项目被绿色填充隐藏了。增加线条的宽度才能获得所期望的结果。

你还可以同 alpha 值指定颜色的透明度。alpha 值的范围从 0（完全透明）到 1（完全不透明）。除了为当前描边或填充命令设定本地 alpha 值外，你还可以使用 globalAlpha 属性给所有描边和填充设置 alpha 值；本地 alpha 值将被乘以 globalAlpha 属性。

此外，你还可以用 globalApha 属性给位图设置透明度。位图中所有像素的 alpha 值将被乘以 globalAlpha 属性。PNG 图像包含了一个 alpha 通道实现透明效果，图像中 alpha 为 0.5 的像素，用 globalAlpha 为 0.5 绘制将实际得到 alpha 为 0.25。



**警告**

绘制 alpha 值小于 1 的元素将涉及浏览器额外的工作，因为浏览器必须进行额外的计算来显示每个像素的最终颜色。无论画布实现是否使用硬件加速都是如此。当设计你的应用程序时，考虑是否绝对需要使用 alpha 值，尤其是当绘制速度很重要时。

如果指定（或通过 globalAlpha 计算得到）alpha 值为 0（完全透明），浏览器可能仍然会尝试绘制。这涉及不必要的工作，可能带来性能问题。尽量避免画 alpha 值为 0 的元素。

我们用 CSS3 定义画布中的颜色。下面这些声明语句中的任意一条都可以设定填充颜色为红色：

```
ctx.fillStyle = 'red';                    // HTML4 color name.
ctx.fillStyle = '#f00';                   // Hexadecimal RGB.
ctx.fillStyle = '#ff0000';                // Hexadecimal RRGGBB.
ctx.fillStyle = 'rgb(255, 0, 0);          // Decimal integers (0-255)
ctx.fillStyle = 'rgba(255, 0, 0, 0.5);    // Decimal integers with 0.5 alpha.
ctx.fillStyle = 'rgb(100%, 0%, 0%)';      // Percentages.
ctx.fillStyle = 'rgb(100%, 0%, 0%, 0.5)'; // Percentages with alpha.
ctx.fillStyle = 'hsl(0, 100%, 100%)';     // Hue, saturation, luminance (HSL).
ctx.fillStyle = 'hsl(0, 100%, 100%, 0.5)'; // HSL with alpha.
```

除了纯色的填充和描边外，你可以使用 createLinearGradient()或 createRadialGradient() 命令指定颜色渐变。

**提示**

用 createLinearGradient()创建渐变，需要一些设置：

1. 使用 createLinearGradient()创建一个 CanvasGradient 对象。传入的 4 个参数定义了将绘制渐变颜色的线。
2. 沿着这条线添加颜色点，其中 0 表示线的开始，1 表示线的末尾。定义渐变你必须至少设置两个颜色点。
3. 使用 CanvasGradient 对象作为填充或描边的样式。

我们使用 CanvasGradient addColorStop()命令添加颜色点。此命令接受 0 和 1 之间的值，其中 0 表示渐变的开始，1 代表结束。下面的代码定义了一个从黑色到白色再到红色的渐变：

```
cg.addColorStop(0, 'black');
cg.addColorStop(0.5, 'white');
cg.addColorStop(1, 'red');
```

下面的函数产生一个渐变的天空和草地效果（如图 6-10 所示）：

```
var drawSkyAndGrass = function (ctx){
    // The gradient line is defined from the top to the bottom of the canvas.
    var cg = ctx.createLinearGradient(0, 0, 0, ctx.canvas.height);
    // Start off with sky blue at the top.
    cg.addColorStop(0, '#00BFFF');
    // Fade to white in the middle.
    cg.addColorStop(0.5, 'white');
    // Green for the top of the grass.
    cg.addColorStop(0.5, '#55dd00');
    // Fade to white at the bottom.
    cg.addColorStop(1, 'white');
    // Use the CanvasGradient object as the fill style.
    ctx.fillStyle = cg;
    // Finally, fill a rectangle the same size as the canvas.
    ctx.fillRect(0, 0, ctx.canvas.width, ctx.canvas.height);
};
```

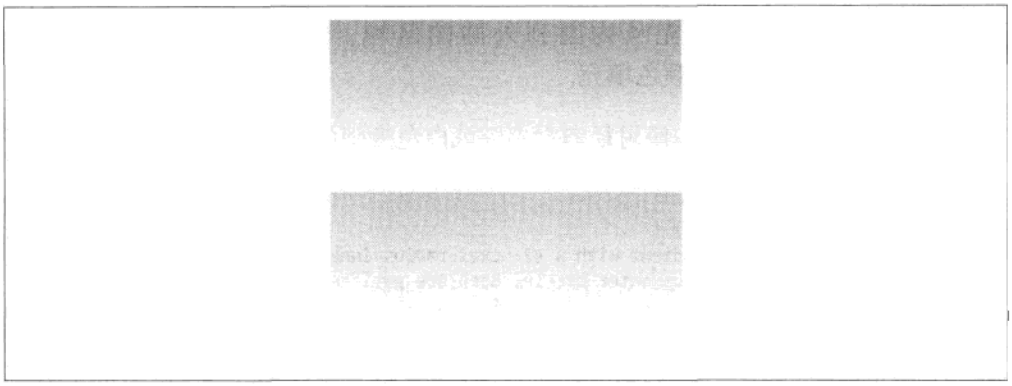


图 6-10　用 createLinearGradient()创建的渐变天空和草地效果

我们调用这个函数时要传入一个画布环境。

如果绘制的矩形和画布的大小不同，会发生什么？我们将前面的函数最后一行替换为下面一行，使得图 6-11 显示了 1/4 个画布大小的矩形。

```
ctx.fillRect(0, 0, ctx.canvas.width/2, ctx.canvas.height/2);
```

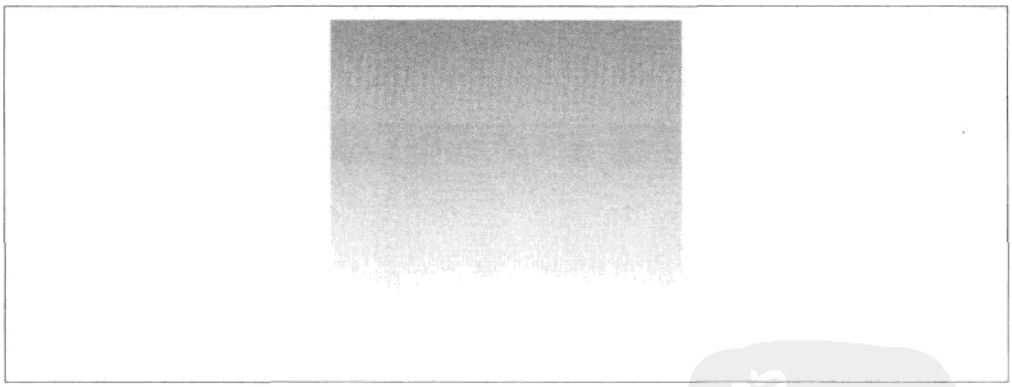


图 6-11　与图 6-10 相同的渐变，但是使用的是更小的矩形运行绘制

请注意绘制矩形的行为像在 CanvasGradient 对象定义的渐变上开一个“窗口”。

**createRadialGradient()**命令则可以创建一个跨越两个圆的径向渐变。该命令接受指定圆心和半径的两个圆：

```
ctx.createRadialGradient(circle1x, circle1y, circle1Radius,
                         circle2x, circle2y, circle2Radius);
```

通常情况下，两个圆心在相同位置，而且第一个圆在第二个圆之内。内圆的所有区

域都用 addColorStop()定义的第一种颜色填充；这种颜色将渐变到由 addColorStop()定义的最终颜色，并填充从内圆到外圆的区域。而外圆外的区域也是由addColorStop()定义的最终颜色填充。

下面的函数创建一个太阳，使用径向渐变，纯白色变淡黄色透明。在天空渐变和草地渐变上放上这个太阳，即可得到一个阳光明媚的效果（如图 6-12 所示）：

```
var drawSun = function(ctx) {
    // Create a radial gradient with a 32-pixel-radius inner circle
    // and a 64-pixel-radius outer circle. Both are positioned at (64,64).
    var radGrad = ctx.createRadialGradient(64, 64, 32, 64, 64, 64);
    // The inner circle is white and opaque.
    radGrad.addColorStop(0,'white');
    // The outer circle is yellow and fully transparent,
    // thus making the sun fade from solid white to transparent yellow.
    radGrad.addColorStop(1,'rgba(255,255,0,0)');
    ctx.fillStyle = radGrad;
    // Fill a 128-pixel-wide rectangle with the sun gradient.
    ctx.fillRect(0, 0, 128, 128);
};
```

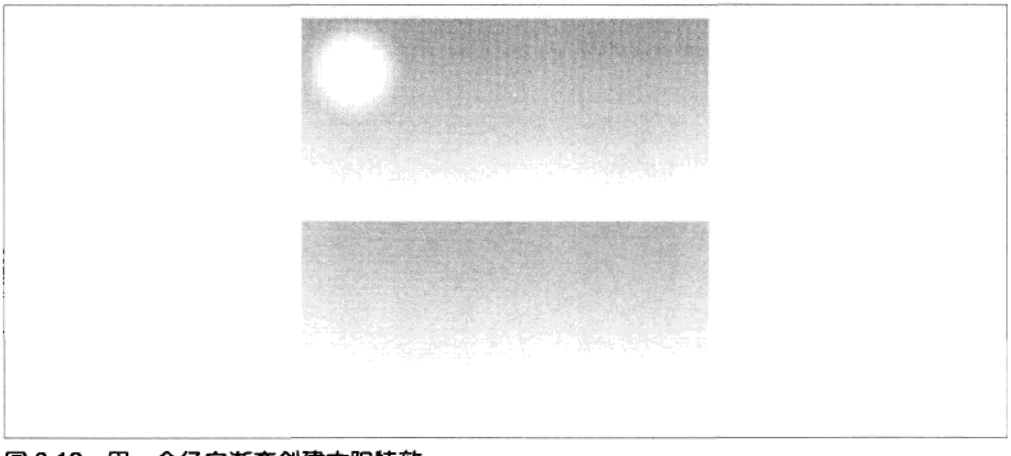


图 6-12　用一个径向渐变创建太阳特效

## 6.8　使用画布创建动画

使用 JavaScript（或 jQuery 等 JavaScript 库）时，你可能习惯操作页面元素的位置、大小、图像或色彩，并看着它神奇地直接忘记其旧属性，而更新其新属性。按这个逻辑，如果我们不断增加一个元素的 x 和 y 位置，可以创建将此元素移到页面右下角的动画效果。但如果我们在画布上以这种方式移动方块的话，结果可能让我们很意外（如图 6-13 所示）：

```
<!DOCTYPE html>
<head>
    <title>
        Naive implementation of animation in Canvas.
    </title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script type="text/javascript"
        src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js">
    </script>
    <script>
        $(document).ready(function() {
            var a_canvas = $("#a_canvas")[0];
            var ctx = a_canvas.getContext("2d");
            for (var p = 0; p < 450; p++) {
                ctx.fillStyle = "rgb(0,0,255)";
                // Draw a rectangle
                ctx.fillRect(p, p, 50, 50);
            }
        });
    </script>
</head>
<body>
    <canvas id="a_canvas" width=500 height=5 00>
    </canvas>
</body>

</html>
```

**图 6-13　简单地在画布上移动方块得到的结果**

请记住画布是一个低级别和立即模式系统：每次循环在屏幕上绘制的另一个矩形，都会叠加于上次迭代的矩形之上。这样的结果是一个大涂鸦，而不是一个动画。为创建在页面上移动的方块动画，我们需要稍稍多做点工作：

1．存储方块的初始（x，y）位置；

2．清除画布；

3．更新方块的（x，y）位置；

4．在新的位置绘制方块；

5．等待一小会儿；

6．循环回到第 2 步。

基本上所有的位图动画系统都在幕后做类似上述循环的操作。在某些情况下第 2 步是可选的。例如，如果背景完全被实色、渐变或位图图像填补，那就没有必要清除它。第 5 步是必要的，这使用户有机会看到动画，并让浏览器有时间去做其他事情，否则该浏览器将立即被冻结。通常情况下大约 20～50 毫秒的延迟比较合适。以下页面会呈现我们想要的效果：

```
<!DOCTYPE html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script type="text/javascript"
        src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js">
    </script>
    <script>
        $(function() {
            var a_canvas = $("#a_canvas")[0],
                ctx = a_canvas.getContext("2d"),
                p = 0;
            // We use setInterval() to create a delay between iterations.
            setInterval(function() {
                // Clear canvas.
                ctx.clearRect(0, 0, a_canvas.width, a_canvas.height);
                // Change position, and restart at top left if position reaches 451.
                if (p++ > 450) {
                    p = 0
                };
                // Draw a rectangle.
                ctx.fillStyle = "rgb(0,0,255)";
                ctx.fillRect(p, p, 50, 50);
            }, 30);

        });
    </script>
</head>
<body>
    <canvas id="a_canvas" width=500 height=500>
    </canvas>
</body></html>
```

## 6.9 画布和递归绘图

立即模式绘图的好处之一是不需要创建和操作什么中间数据结构。在立即模式绘图中，你可以立即忘记刚刚执行的绘图命令，或者叠加多个绘图命令。这对在画布中使用高密度、递归的绘图函数（如不规则碎片形）特别有用。递归函数指的是调用

自己的函数。通过将函数的上次结果返回给函数本身，我们创建了软件的反馈环路。下面的例子递归调用自己 10 次：

```
var rescurse(value1, value2) {
    value1--;
    value2++;
    if (value1 <= 0) return;
    recurse(value1, value2);
};
recurse(10,0);
```

这个例子虽然很简单，但它演示了递归函数的两个重要方面：

- 在递归函数内修改值，然后将这个值反馈回去。
- 我们需要一个条件语句来跳出死循环。

下面我们将尝试用更有趣的东西，具体说是用一点三角函数和一点随机元素，来代替简单的递归递减。图 6-14 显示了递归调用简单的画布画线命令得出的树。递归图形函数的一个显著特点是看起来比较自然，我们可以看到分支的末梢非常精细。这是由于前面提过的分数像素级抗锯齿技术。

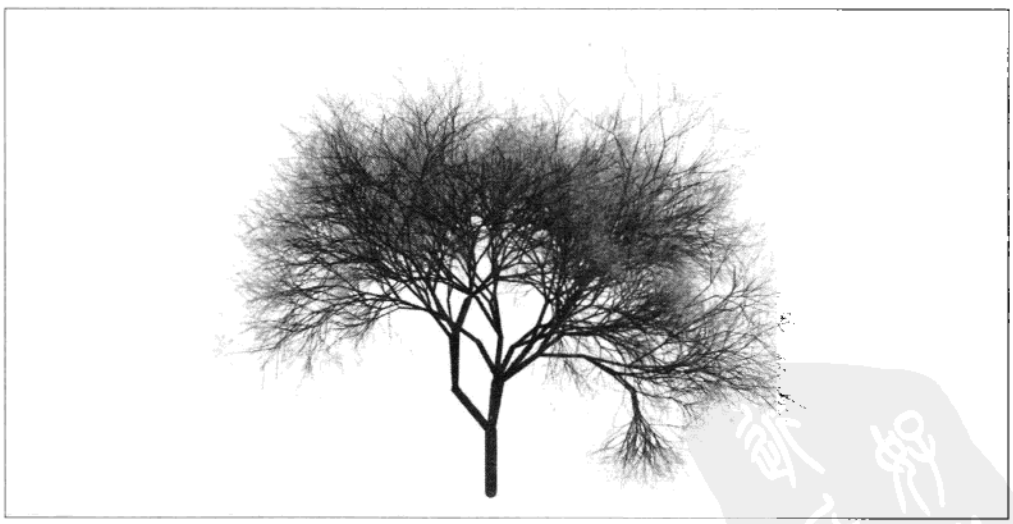


**图 6-14　在画布上递归绘制的树**

尽管外观复杂、细节丰富，实现代码却非常简单：

```
var drawTree = function (ctx, startX, startY, length, angle, depth, branchWidth) {
    var rand = Math.random,
        newLength, newAngle, newDepth, maxBranch = 3,
        endX, endY, maxAngle = 2 * Math.PI / 4,
```

```
        subBranches, lenShrink;
    // Draw a branch, leaning either to the left or right (depending on angle).
    // First branch (the trunk) is drawn straight up (angle = 1.571 radians)
    ctx.beginPath();
    ctx.moveTo(startX, startY);
    endX = startX + length * Math.cos(angle);
    endY = startY + length * Math.sin(angle);
    ctx.lineCap = 'round';
    ctx.lineWidth = branchWidth;
    ctx.lineTo(endX, endY);
// If we are near the end branches, make them green to look like leaves.
if (depth <= 2) {
    ctx.strokeStyle = 'rgb(0,' + (((rand() * 64) + 128) >> 0) + ',0)';
}
// Otherwise, choose a random brownish color.
else {
    ctx.strokeStyle = 'rgb(' + (((rand() * 64) + 64) >> 0) + ',50,25)';
}
ctx.stroke();

// Reduce the branch recursion level.
newDepth = depth - 1;
// If the recursion level has reached zero, then the branch grows no more.
if (!newDepth) {
    return;
}
// Make current branch split into a random number of new branches (max 3).
// Add in some random lengths, widths, and angles for a more natural look.
subBranches = (rand() * (maxBranch - 1)) + 1;
// Reduce the width of the new branches.
branchWidth *= 0.7;
// Recursively call drawTree for the new branches with new values.
for (var i = 0; i < subBranches; i++) {
        newAngle = angle + rand() * maxAngle - maxAngle * 0.5;
        newLength = length * (0.7 + rand() * 0.3);
        drawTree(ctx, endX, endY, newLength, newAngle, newDepth, branchWidth);
    }
};
```

## 6.9.1 画布树的页面布局

修改 drawTree()的初始值，你会发现初始值的微小变化可以给出非常不同的结果。不推荐使用远超过 12 的 depth 值（倒数第二个参数），除非你非常有耐心！

```
<!DOCTYPE html>
<html>
    <head>
        <title>
            Recursive Canvas Tree
        </title>
        <script type="text/javascript"
            src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js">
        </script>
        <script type="text/javascript">

            /*** drawTree() function goes here ***/
```

```
            $(document).ready(function() {
                var canvas = document.getElementById('mycanvas');
                var ctx = canvas.getContext('2d');
                drawTree(ctx, 320, 470, 60, -Math.PI / 2, 12, 12);
            });
        </script>
    </head>
    <body>
        <canvas id="mycanvas" width=640, height=4 80></canvas>
        </div>
    </body>
</html>
```

/*这里是 drawTree()函数的代码*/

# 6.10 用画布 sprite 取代 DHTML sprite

在第 2 章中，我们开发了 DHTML sprite 动画系统，并用它创建了各种图形演示。在第 5 章中，我们用这个系统制作了一个 DHTML 视频游戏。我们尽量将绘制 sprites 的细节“隐藏”在 DHTMLSprite 对象中，使得应用程序可以很容易实现和使用一个不同的 sprite 系统。现在我们将在演示中使用一个新的 CanvasSprite 对象，它利用了性能更强的画布元素。

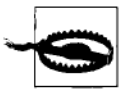
## 6.10.1 新 CanvasSprite 对象

CanvasSprite 是 DHTMLSprite 对象的一个直接替换。除了加了一个画布环境参数（ctx）外，params 对象的所有参数都和以前一样被传入：

```
var CanvasSprite = function (params) {
    // The canvas drawing context is passed in the params object.
    var ctx = params.ctx,
        width = params.width,
        height = params.height,
        imagesWidth = params.imagesWidth,
        vOffset = 0,
        hOffset = 0,
        hide = false,
        // An Image object is created, and this will be used as the source
        // for the canvas drawImage function below.
        img = new Image();
    img.src = params.images;

    return {
        draw: function (x, y) {
            if (hide) {
                return;
            }
            // The canvas drawImage function allows us to extract individual
            // sprite images from a larger composite image.
            ctx.drawImage(img, hOffset, vOffset, width, height,
```

```
                x >> 0, y >> 0, width, height);
        },
        changeImage: function (index) {
            index *= width;
            vOffset = Math.floor(index / imagesWidth) * height;
            hOffset = index % imagesWidth;
        },
        show: function () {
            hide = false;
        },
        hide: function () {
            hide = true;
        },
        destroy: function () {
            return;
        }
    };
};
```



**警告**

注意我们使用移位运算符（x>>0，y>>0）确保渲染位置为整数。Firefox 和 Opera 浏览器在分数像素位置绘制时性能会有很大影响。这对普通绘图影响不大，但对高速图形应用，将非常影响性能。

## 6.10.2 其他的代码更改

下面的代码中粗体标注的是让 CanvasSprite 工作所需做的其他修改。你可以参考第 2 章中 DHTMLSprite 的代码进行比较。

```
var bouncySprite = function(params) {
    // Other code as before goes here...
    // We are now referencing CanvasSprite instead of DHTMLSprite.
    // that = DHTMLSprite(params);
    that = CanvasSprite(params);
    // Other code as before goes here...
};

var bouncyBoss = function (numBouncy, $drawTarget, ctx) {
    var bouncys = [];

    for (var i = 0; i < numBouncy; i++) {
        bouncys.push(bouncySprite({
            // Other code as before goes here...
            maxY: $drawTarget.height() - 64,
            ctx: ctx    // Pass a Canvas context as one of the parameters to bouncy
                        // sprite.
        }));
    }
    var moveAll = function () {
        // The moveAll() function now clears the Canvas before drawing
        // all the sprites.
        ctx.clearRect(0, 0, ctx.canvas.width, ctx.canvas.height);
```

```
        // Other code as before goes here...
    };
    moveAll();
};

$(document).ready(function () {
    // Pass a Canvas context to bouncyBoss.
    var canvas = $('#draw-target')[0];
    bouncyBoss(80, $('#draw-target'), canvas.getContext("2d"));
});
```

## 6.11 一个图形使用画布的 WebSockets 聊天应用

在下面的例子中我们将看到一个更实用的画布应用程序：一个伪 3D 聊天应用程序（如图 6-15 所示）。这个例子将演示如何将画布和其他 HTML5 特性如 WebSockets 结合。

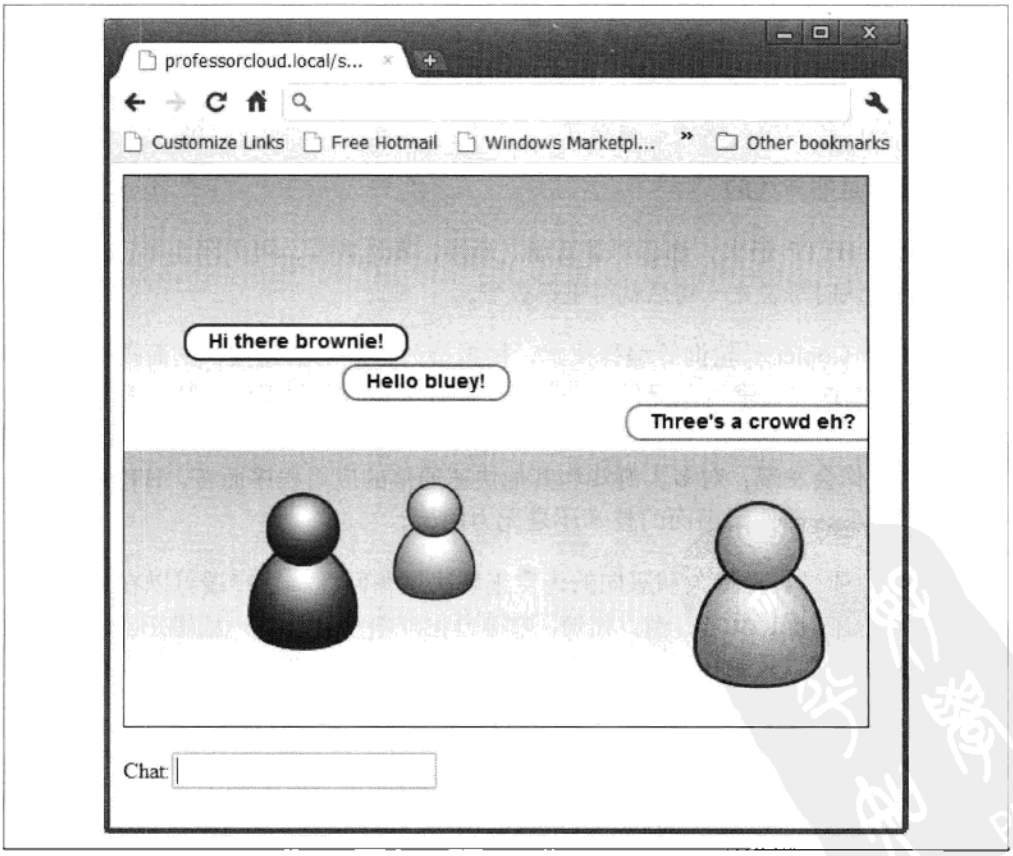



图 6-15 一个使用 HTML 画布和 WebSockets 的伪 3D 图形聊天应用

### 6.11.1 WebSockets 优势

除画布外，另一个同样令人振奋（但可能比较不知名）的 HTML5 元素是 WebSockets。虽然这本书是关于图形的，但还是值得讨论一下为什么 WebSockets 对现代 Web 应用程序意义重大，以及如何将它们与画布集成。

通常服务器和客户端浏览器之间是用 HTTP 协议传递数据的，但 HTTP 有一定的局限性（和新的 WebSockets 相比），使它不适合高速、双向的网络通信。

HTTP 是单行线

> HTTP 的模式是：客户端 Web 浏览器向服务器请求数据，服务器满足其要求。服务器不能在没有被请求的情况下“推送”信息到客户端。

它的开销很大

> HTTP 数据携带了大量的头信息。请求一个字节的数据可能会导致发送数百个字节的、额外的、“看不见的”头信息也被发送。头信息通常包含被传送数据的性质，如内容类型、缓存、编码等。

它的连接都是非持久的

> 为每个 HTTP 请求，都必须重新建立连接，发送数据，再关闭连接。这好比是电话交谈时每说完一句话就得重新拨号。

你可以使用 Comet/长轮询等编程技术，模拟持久、双向的连接，从而提高 HTTP 的性能。虽然这些技术可以提供一些改善，但有可能服务器无法支持这些技术所需要的超高 HTTP 连接数；如 Apache 这样的服务器在处理此类连接时就不是特别有效率。最终你会发现：对多人游戏和其他快速通信的应用程序而言，HTTP 的网络传输效率太低，即使用前面的技术还是无力回天。

WebSockets 通过真正持久和双向的连接来解决这些问题。客户端可以在任何时候给服务器发送数据，反之亦然。此外，数据开销非常小，因为一旦建立了连接就不需要传递头数据。只是在数据前后各加了一个 0 字节（0x00）和 0xFF 表示结束和终止。

### 6.11.2 WebSockets 支持和安全

目前有 Firefox 4+、Google Chrome 4+、Opera 10.70+和 Safari5 支持 WebSockets。不过由于 WebSockets 通信相关的安全问题，使得 Firefox 和 Opera 的开发者关闭了默认的

WebSockets 功能，而其他浏览器厂商可能也会采取同样的行动。你可以在下面的链接找到更多细节：http://www.ietf.org/mail-archive/web/hybi/current/msg04744.html。

在解决安全问题之前，默认 WebSockets 功能看来还处于不太稳定的状态。然而，我们可以现在就试验 WebSockets 协议，为未来做好准备。

在 Firefox 4 和 Opera 11 上的 WebSockets

幸运的是，Firefox 和 Opera 用户可以在开发中打开 WebSockets 功能。

Firefox4：

1．在浏览器的地址栏输入 about:config。

2．查找和更改 network.websocket.override-security-block 标志。

Opera：

1．在浏览器的地址栏输入 opera:config。

2．在首选项编辑器，打开“用户首选项”部分，并设置 Enable WebSockets。

## 6.11.3 聊天应用程序

我们的聊天应用程序大致由 4 个主要部分组成：

- 在 Web 服务器上运行的套接字服务器；
- 可以左右移动和“聊天”的客户化身（avatar）；
- 聊天文字；
- 文本输入区。

当用户连接到聊天页面时，将自动为他创建一个颜色随机的化身。然后用户可以在页面上点击来移动化身，也可以在聊天框中输入文字。化身的移动和聊天文字将反映给所有其他的连接用户。也就是说，每个用户看到的都是相同的页面，但只能控制自己的化身。

### 1．套接字服务器

套接字服务器需要处理连接，并在连接的客户端之间传输信息。它必须运行在所有客户端可以连接到的服务器上。

套接字服务器的编程语言可以是任何流行的服务器端语言，如 PHP、Java 或 Python。JavaScript 程序员可以关注 node.js，它是一个基于服务器的 JavaScript 实现和使其适合高效网络编程的一些相关库。

这个例子里我选择了 PHP，因为几乎所有基于 Linux 的托管主机上都已安装了 PHP。

**提示**

套接字服务器（server.php）实际上由两部分组成：一个通用的套接字处理类（WebSocketServer），可用于各种应用；一个聊天应用程序特定的回调函数（process()），这是为我们的聊天应用程序定制的。请从本书的代码中阅读 server.php 的源代码。

本书不可能展开讨论 PHP。PHP 语言比较简单易学，网上有大量的学习资源。如果你打算仔细看看套接字服务器代码，请注意下面几个 PHP 语法：

- 变量前面有一个美元符号（不要和 jQuery 中的$混淆）。
- 字符串连接符号不是加号（+），而是句号（.）。这点比较特殊。

套接字服务器至少要执行以下操作：

- 接受新连接，并保持连接的客户端列表；
- 接收来自客户端的数据更新（聊天的文字和位置）；
- 传输数据更新到所有连接的客户端；
- 当连接断开时（如浏览器关闭），从列表中删除客户。

### 2. 在本地安装一个网站托管环境

除非你有专门或虚拟主机的 root 权限，否则是不可能让套接字服务器工作的。正确配置的共享网站托管环境都会有一个防火墙，防止使用任何的通信端口。幸运的是，你可以安装一个本地环境来运行服务器端代码。

安装一个网站托管环境曾经是痛苦而漫长的过程。幸好“Apache Friends”XAMPP 软件将所需模块（Apache 和 PHP）合并到一个下载文件中，你可以在几分钟之内将其安装到 Windows、Mac 或 Linux 系统上。你可以去 http://www.apachefriends.org/en/xampp.html 下载 XAMPP 软件。

图 6-16 显示了 XAMPP 的控制面板。注意 PHP 是以透明方式运行的，在控制面

板中不显示。

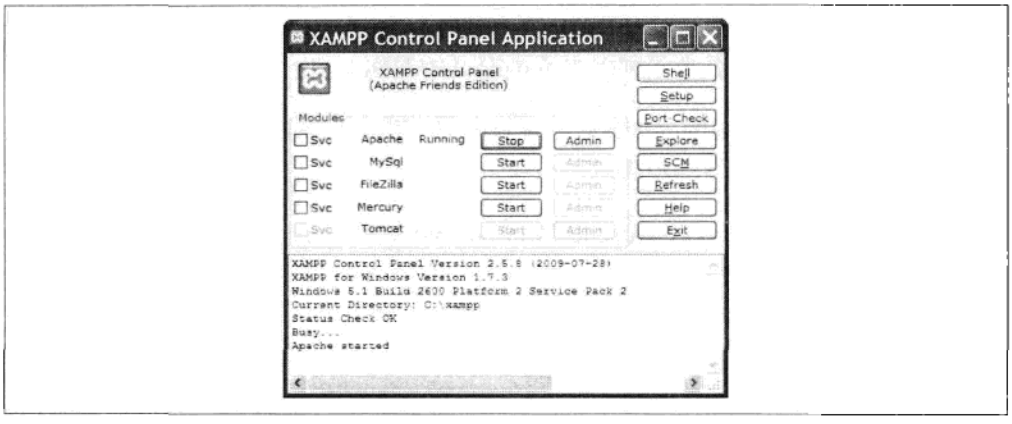



**图 6-16　你可以用 XAMPP 在本地轻松安装一个完整的网站托管环境**

要通过 XAMPP 启动套接字服务器，单击 XAMPP 控制面板上的 Shell 按钮。这将呈现一个命令行，使你可以运行套接字服务器（如图 6-17 所示）。输入 php *path-to-socket-server*\server.php 来运行套接字服务器，然后按 Enter 键。

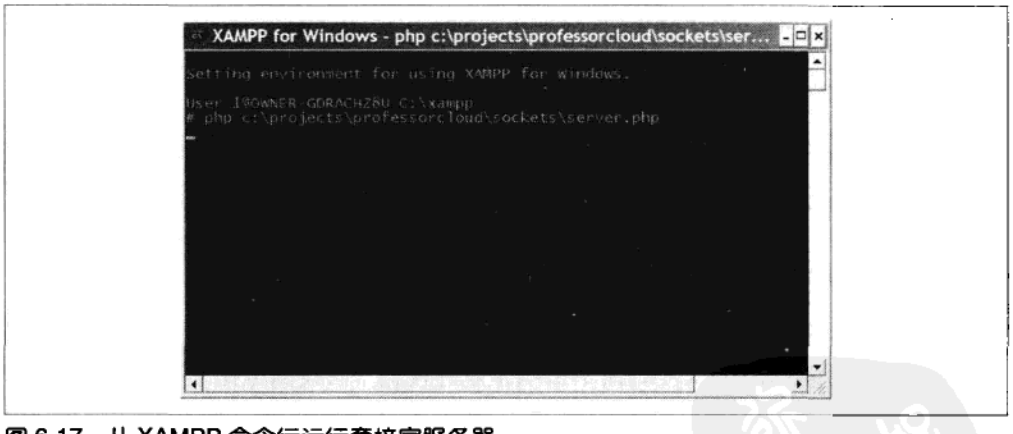



**图 6-17　从 XAMPP 命令行运行套接字服务器**

你需要将 server.php 的路径改为这个文件在你本地机器上的位置。

套接字服务器现在正在等待聊天应用 JavaScript 端的连接。

最后，我们需要在浏览器中运行实际的聊天网页（JavaScript）。最简单的方法是在浏览器中指定页面的文件系统位置，如 file:///C:/professorcloud.com/book/canvas/canvas-websockets-chat.htm。

当我们使用 file:///协议时，我们从浏览器直接访问 HTML 文件，并没有用到 Apache。

你还可以打开多个浏览器来测试多个聊天网页的实例，自己和自己聊天。

测试聊天应用程序更精细的一个方式是通过网络（Web 服务器）来运行聊天服务器、提供聊天页面。这里涉及的步骤有：

（1）在 Web 服务器上，修改 Apache 的 httpd.conf 文件中的一个虚拟主机条目，将 Web 服务器的 IP 地址映射到 Web 服务器上聊天应用程序的位置。定义的第一个虚拟主机是 Web 服务器 IP 地址的默认主机。

（2）通过 XAMPP 控制面板启动 Apache。

网络上其他计算机的用户可以通过在合适的浏览器上输入 Web 服务器的 IP 地址，连接到聊天应用程序。

下面是关于如何在不同操作系统上设置 XAMPP 的文档：

- Windows：http://www.apachefriends.org/en/xampp-windows.html
- Mac：http://www.apachefriends.org/en/xampp-macosx.html
- Linux：http://www.apachefriends.org/en/xampp-linux.html

### 3. 相机

相机对象决定了聊天区域的透视图。它包含了 3 个应用函数：

setFOVandYPos()

> 传入相机的视野（FOV, Field of View）的角度和垂直位置。从视野计算相机距离使你可以改变画布大小，而不影响呈现的视图（假设高宽比不变）。我们使用 125 度的视野和−128 的相机 y 位置。

worldToScreen()

> 从传入的世界坐标计算画布屏幕上的坐标。它给化身计算尺度使得它们在远处显得比较小，有效地模拟透视效果。

screenToWorld()

> 这个函数和 worldToScreen()相反，从画布上的位置得到对应的世界坐标。

screenToWorld()将鼠标单击的位置转化为用户化身的新世界坐标位置。我们使用 toFixed()方法防止返回值过于精细。例如，188.42620390960207 将被 188.426 替代，因为前者需要更长的网络传输时间且没有太多必要。

```
var camera = function () {
    var camDist, camY;
    return {
        setFOVandYPos: function (angle, y) {
            camY = y;
            angle *= (Math.PI / 180);
            camDist = (ctx.canvas.width * 0.5) / Math.tan(angle * 0.5);
        },
        worldToScreen: function (x, y, z) {
            return {
                sx: (camDist * x) / z,
                sy: (camDist * (y - camY)) / z,
                scale: (camDist / z)
            };
        },
        screenToWorld: function (sx, sy) {
            sx -= ctx.canvas.width / 2;
            sy -= ctx.canvas.height / 2;
            var wz = (-camY / sy) * camDist;
            return {
                wx: (sx / camDist * wz).toFixed(3),
                wy: (sy / camDist * wz).toFixed(3),
                wz: wz.toFixed(3)
            };
        }
    };
}();
```

## 4. 化身

化身是客户端的图形表示。它们各自以随机颜色出现，以区别于其他化身。我们通过单击鼠标将其移动要新位置上。它们由分别代表头和身体的两个矢量形状组成。我们对化身用径向渐变填充以增加其深度效果，使用深色描边来将其和背景分离。

```
var avatar = function (color) {
    var that = {},
        destX = 0,
        destZ = 0,
        x = 0,
        z = 0,
        textX, avatarHW = 40.5,
        avatarH = 106,
        outlineColor = color.substr(1),
        gradient1, gradient2;
    outlineColor = (parseInt(outlineColor, 16) & 0xfefefe) >> 1;
    outlineColor = '#' + outlineColor.toString(16);

    gradient1 = ctx.createRadialGradient(37.7, 55.6, 0.0, 37.7, 55.6, 46.1);
    gradient1.addColorStop(0.00, "#fff");
```

```
gradient1.addColorStop(1.00, color);
gradient2 = ctx.createRadialGradient(37.6, 15.3, 0.0, 37.6, 15.3, 31.1);
gradient2.addColorStop(0.00, "#fff");
gradient2.addColorStop(1.00, color);

that.remove = false;

that.setDest = function (dstX, dstZ) {
    destX = dstX;
    destZ = dstZ;
};
that.getZ = function () {
    return z;
};
that.getTextX = function () {
    return textX;
};
that.move = function (coeff) {

    var vx = destX - x,
        vz = destZ - z,
        dist = Math.sqrt(vx * vx + vz * vz),
        p, x1, y1;

    // Normalize (make unit length) the vector from old pos to new pos.
    if (dist) {
        vx /= dist;
        vz /= dist;
    }
    // Apply the vector capped to a maximum of 4 units.
    if (dist > 4) {
        dist = 4;
    }
    x += vx * (dist * coeff);
    z += vz * (dist * coeff);
    p = camera.worldToScreen(x - avatarHW, -avatarH, z);
    textX = p.sx + (avatarHW * p.scale) + (ctx.canvas.width / 2);

    // Draw the body.
    ctx.save();
    ctx.translate(p.sx + (ctx.canvas.width / 2), p.sy + (ctx.canvas.height / 2));
    ctx.scale(p.scale, p.scale);
    ctx.beginPath();
    ctx.moveTo(73.1, 83.6);
    ctx.bezierCurveTo(71.7, 102.1, 52.2, 105.2, 37.4, 105.2);
    ctx.bezierCurveTo(22.5, 105.2, 3.0, 102.1, 1.6, 83.6);
    ctx.bezierCurveTo(0.1, 62.7, 14.0, 35.3, 37.4, 35.3);
    ctx.bezierCurveTo(60.8, 35.3, 74.7, 62.7, 73.1, 83.6);
    ctx.closePath();
    ctx.fillStyle = gradient1;
    ctx.fill();
    ctx.lineWidth = 2.0;
    ctx.lineJoin = "miter";
    ctx.miterLimit = 4.0;
    ctx.strokeStyle = outlineColor;
```

```
        ctx.stroke();
        // Draw the head.
        ctx.beginPath();
        ctx.moveTo(61.2, 25.3);
        ctx.bezierCurveTo(61.2, 38.4, 50.5, 49.1, 37.4, 49.1);
        ctx.bezierCurveTo(24.2, 49.1, 13.6, 38.4, 13.6, 25.3);
        ctx.bezierCurveTo(13.6, 12.1, 24.2, 1.5, 37.4, 1.5);
        ctx.bezierCurveTo(50.5, 1.5, 61.2, 12.1, 61.2, 25.3);
        ctx.closePath();
        ctx.fillStyle = gradient2;
        ctx.fill();
        ctx.strokeStyle = outlineColor;
        ctx.stroke();
        ctx.restore();
    };
    return that;
};
```

### 5. 聊天文字

聊天的文字将在“发言的”化身上方出现，并随着用户输入更多文字逐渐向屏幕上方移动。为了使文字更清晰并加入聊天泡泡的效果，我们用白色填充的圆角矩形环绕文字，并用化身的颜色给这个矩形描边。

当化身聊天时，textScroller 对象管理和绘制生成的文字。addText()方法将新的文字添加到列表的开始，同时删除 5 句之前的文字。这创建了垂直的滚动效果，文字向画布上方移动时最顶层的文字将丢失。这个方法接受化身的水平位置作为文字的中心位置，以及化身的颜色。

drawText()方法遍历文字列表，并绘制每一行文字。为使文字更加突出，我们在文字周围显示白色填充的圆角矩形，并用化身颜色对其描边。我们使用画布的 measureText()方法来计算文本和圆角矩形的宽度。

```
var textScroller = function () {
    var textList = [];
    return {
        addText: function (text, x, color) {
            if (textList.length > 5) {
                textList.splice(0, 1);
            }
            textList.push({
                text: text,
                x: x,
                color: color
            });
        },
        drawText: function () {
            var y = (ctx.canvas.height / 2) - 16,
                tx, w, x1, y1, w1, i;
            ctx.font = "bold 14px sans-serif";
```

```
            ctx.fillStyle = '#000';
            for (i = textList.length - 1; i > -1; i--) {
                tx = textList[i];
                w = ctx.measureText(tx.text).width / 2;
                ctx.beginPath();
                y1 = y - 17;
                x1 = tx.x - 2; // Same as stroke width.
                w1 = w + 16;
                // Begin in middle of top.
                ctx.moveTo(x1, y1);
                // Top and upper-right corner.
                ctx.arcTo(x1 + w1, y1, x1 + w1, y1 + 24, 10);
                 // Right and lower-right corner.
                ctx.arcTo(x1 + w1, y1 + 24, x1 - w1 - 10, y1 + 24, 10);
                 // Bottom and lower-left corner.
                ctx.arcTo(x1 - w1, y1 + 24, x1 - w1, y1, 10);
                // Left and upper-left corner.
                ctx.arcTo(x1 - w1, y1, x1 + w1, y1, 10);
                ctx.closePath();
                ctx.fillStyle = 'white';
                ctx.fill();
                ctx.lineWidth = 2;
                ctx.strokeStyle = tx.color;
                ctx.stroke();
                ctx.fillStyle = 'black';
                ctx.fillText(tx.text, x1 - w, y);
                y -= 28;
            }
        }

    };
}();
```

## 6. 背景

drawBackground 对象绘制一个渐变的蓝天和绿地。蓝天和绿地都被渐变为白色以给人一种三维的深度感。

```
var drawBackground = function () {
    var linGrad = ctx.createLinearGradient(0, 0, 0, ctx.canvas.height);
    linGrad.addColorStop(0, '#00BFFF');
    linGrad.addColorStop(0.5, 'white');
    linGrad.addColorStop(0.5, '#55dd00');
    linGrad.addColorStop(1, 'white');
    return function () {
        ctx.fillStyle = linGrad;
        ctx.fillRect(0, 0, ctx.canvas.width, ctx.canvas.height);
    };
}();
```

## 7. 初始化

initAndGo()函数执行各种设置工作，如建立事件处理程序、连接到服务器。最后它执行移动和绘制头像与文字的循环：

```
var initAndGo = function () {
    // Set the field of view and camera vertical position.
    camera.setFOVandYPos(125, -128);
    // Socket server is running on the local machine
on port 8999.
    var host = "ws://127.0.0.1:8999",
        socket, avatarList = [];
    // The send function transmits an arbitrary number of arguments to the
    // server.
    var send = function () {
        var data = '';
        for (var i = 0; i < arguments.length; i++) {
            data += arguments[i] + ',';
        }
        socket.send(data);
    };
    try {
        socket = new WebSocket(host);
        // When the socket connects, it creates a new avatar in a random color.
        // It also sets the border color around the text input area.
        socket.onopen = function (msg) {
            // Random color for avatar.
            var rColor = Math.round(0xffffff * Math.random());
            rColor = ('#0' + rColor.toString(16)).
                replace(/^#0([0-9a-f]{6})$/i, '#$1');
            send('CONNECT', rColor, 250);
            $('#text-input').css({
                border: "2px solid " + rColor,
                color: rColor
            });
        };
        socket.onmessage = function (msg) {
            if (msg.data) {
                var textData = msg.data,
                    data;
                // Parse the returned socket data into a JavaScript object
                // via JSON.
                textData = textData.replace(/[\x00-\x1f]/, '');
                data = $.parseJSON(textData);
                for (var userId in data) {
                    if (avatarList[userId] === undefined) {
                        // Initialize a new avatar if the the userId doesn't
                        // yet exist in the avatarList[].
                        avatarList[userId] = avatar(data[userId].colr);
                    }
                    if (data[userId].pos !== undefined) {
                        // Update avatar's destination x and z positions.
                        var pos = data[userId].pos.split(',');
                        avatarList[userId].setDest(pos[0], pos[1]);
                }
                if (data[userId].chattext !== undefined) {
                    // Add chat text if present in data.
                    textScroller.addText(unescape(data[userId].chattext),
                        avatarList[userId].getTextX(), data[userId].colr);
                }
                if (data[userId].disconnect) {
```

```
                    // Flag avatar for removal if the server says so.
                    avatarList[userId].remove = true;
                }
            }
        }
    };
} catch (ex) {
    alert('Socket error: ' + ex);
}
// Stop text input losing focus when clicking on canvas.
$('#the-canvas').bind('mousedown', this, function (event) {
    return false;
});
// Get clicks on canvas and convert to world coordinates.
// Send these coordinates back to the server.
$(ctx.canvas).bind('click', function (evt) {
    var canvas, bb, mx, my, p;
    canvas = ctx.canvas;
    // Get canvas size and position
    bb = canvas.getBoundingClientRect();
    // Convert mouse event coordinates to canvas coordinates
    mx = (evt.clientX - bb.left) * (canvas.width / bb.width);
    my = (evt.clientY - bb.top) * (canvas.height / bb.height);
    // Stop avatars going too far back.
    if (my < canvas.height / 2 + 32) {
        return;
    }
    p = camera.screenToWorld(mx, my);
    send('UPDATE', p.wx, p.wz);
});
// Get key presses and send chat text to server if return key is pressed.
// The text is escaped to ensure correct transmission.
$(window).bind('keypress', function (evt) {
    if (evt.which == 13) {
        send('CHATTEXT', escape($('#text-input').val()));
        $('#text-input').val('');
    }
});
var oldTime = new Date().getTime();

// The main loop is executed via setInterval at 20-millisecond intervals.
setInterval(function () {
    var newTime = new Date().getTime(),
        elapsed = newTime - oldTime,
        i = 0,
        avatarListNew = [],
        sortList = [],
        // Work out a coefficient of movement based on elapsed time
            // to ensure consistent speed across different browsers and hardware.
            coeff = elapsed / 20;
        oldTime = newTime;

        // Draw the background. There is no need to erase
        // the canvas first, as the background completely fills it.
        drawBackground();

        // Place non-removed avatars into sortlist ready for drawing.
```

```
        // Also place them in avatarListNew.
        for (var av in avatarList) {
            if (!avatarList[av].remove) {
                sortList[i++] = avatarListNew[av] = avatarList[av];
            }
        }

        // Sort the list into z-order.
        sortList.sort(function (a, b) {
            return b.getZ() - a.getZ();
        });

        // Move the avatars.
        for (i = 0; i < sortList.length; i++) {
            sortList[i].move(coeff);
        }

        // avatarListNew now becomes our current avatar list.
        // It does not contain removed avatars.
        avatarList = avatarListNew;

        // Finally, draw all the chat text.
        textScroller.drawText();
    }, 20);
}();
```

## 8. 页面代码

下面是聊天应用程序的 HTML 页面布局，被保存在一个名为 canvas-webSockets-chat.htm 的文件中：

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <script type="text/javascript"
            src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js">
        </script>
        <script type="text/javascript">                jQuery( function($) {
            var ctx = $('#the-canvas')[0].getContext('2d');

            var camera = function () {
                /*** CODE REMOVED FOR CONCISENESS ***/
            }();
            var textScroller = function () {
                /*** CODE REMOVED FOR CONCISENESS ***/
            }();
            var avatar = function (color) {
                /*** CODE REMOVED FOR CONCISENESS ***/
            };
            var drawBackground = function () {
                /*** CODE REMOVED FOR CONCISENESS ***/
            }();
            var initAndGo = function () {
                /*** CODE REMOVED FOR CONCISENESS ***/
            }();
```

```
        });
        </script>
        <style type="text/css">
            body {font-family: sans-serif}
            #text-input {font-size:16px;}
            #the-canvas {border:1px solid;}
        </style>
    </head>

    <body>
        <canvas id="the-canvas" width='512' height='384'>
        </canvas>
        <p>
            <label for='text-input'>
                Chat:
            </label>
            <input id='text-input' />
        </p>
    </body>
</html>
```

# 第 7 章

# 游戏和模拟中的向量

绝大多数程序员会认为编程远比数学有趣得多，但是在某些情况下，掌握一点数学知识可以帮助你解决很大的问题。其中，向量知识是一个“超值的”的数学主题。将其和其他一些数学成分结合，你就会拥有一个可用于各种应用程序的向量工具包。为给大家参考，本章会对所有公式提供其相应的 JavaScript 代码。熟悉相关数学知识当然会对阅读本章很有益处，但这并不是必须的。

向量通常被描述为一个既有长度又有方向的量。可这到底是什么意思？一些简单的例子可以很好地说明向量的概念。

非向量

- 2 米
- 12 英寸
- 1 千米

向量

- 北方 2 米
- 距右侧 12 英寸
- 东北方向 1 千米

为什么向量是有用的？因为它使得所有的移动和空间行为都可以更容易被理解和在代码中实现。向量可以相加，缩放，旋转，指向某物体。它为像物理模拟这样更复杂的编程主题提供了基础。最重要的是，当你掌握了向量之后，你会

发现它们非常有趣。

前面描述的现实生活中的距离和方向是我们熟悉的，也是讲得通的；但是作为一个 JavaScript 程序员，我们对与程序有关的大小和方向更感兴趣——而不是简单的米和英寸。

我们应该使用什么测量单位？事实上，实际的测量单位是没有关系的：只要我们将所有的测量都使用统一的单位，最终我们就可以将向量转化成屏幕上的像素点，准备绘制。

在现实世界的例子中，向量的方向有指南针方向和“向右”等来指定。这些标准并不适用于 JavaScript，所以我们必须用其他的方法表示向量的方向。一个方向和长度（即向量）在二维空间（例如你的电脑屏幕）中可以用横坐标 x 和纵坐标 y 来表示。图 7-1 的同一个网格中表示了 4 个不同的向量及其 x 和 y 分量。

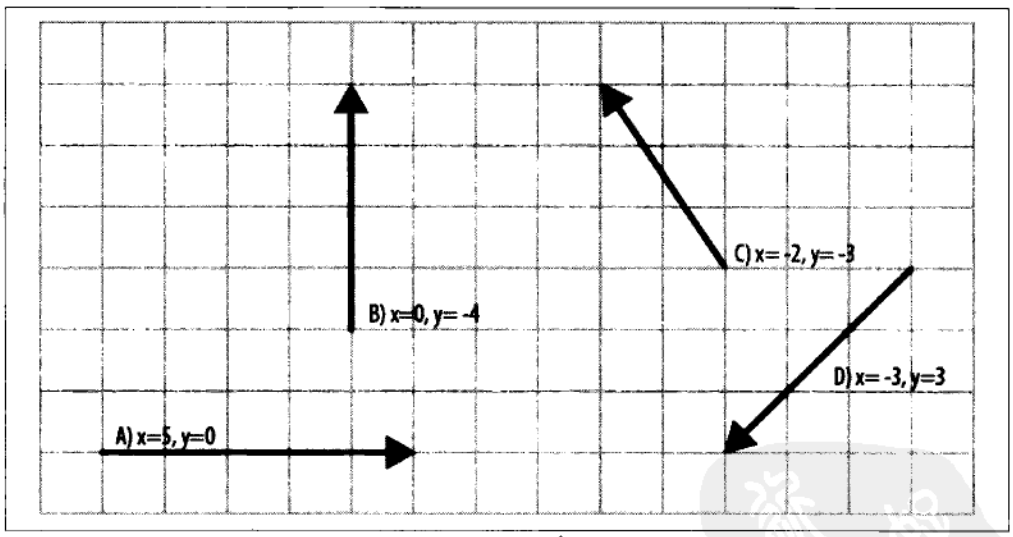



图 7-1　4 个不同的向量及其 x 和 y 分量

在本章中，我们仍然使用前面提到的 CSS/位图的屏幕坐标系统，即左上角为原点、x 轴沿右侧增加、y 轴沿下方增加（亦称做笛卡儿坐标系）。在这个例子中，向量代表方向和长度，而不是位置。它们在网格上的位置可以是任意的。不管怎样，如何用 x 和 y 分量表示位置，取决于如何在应用程序中使用向量。

在图 7-1 中，方向由 x 和 y 分量指定，但向量的长度是如何表示的？如果向量和某一个坐标轴恰好是同一个方向，那么向量的长度就是沿着这个坐标轴的长度。例如，显而易见的，向量 A 的长度是 5，向量 B 的长度是 4。但是，怎么计算那些不和任何坐标轴平行的向量长度呢？比如向量 C 和向量 D 的长度。在这种情况下，向量的长度就不是那么明显的了，x 和 y 都不能简单地用来表示向量的长度。

幸运地，我们可以根据 x、y 坐标利用勾股定理来计算出向量的长度。如下是勾股定理的定义：

对于直角三角形，斜边的平方等于两条直角边的平方和（如图 7-2 所示）。

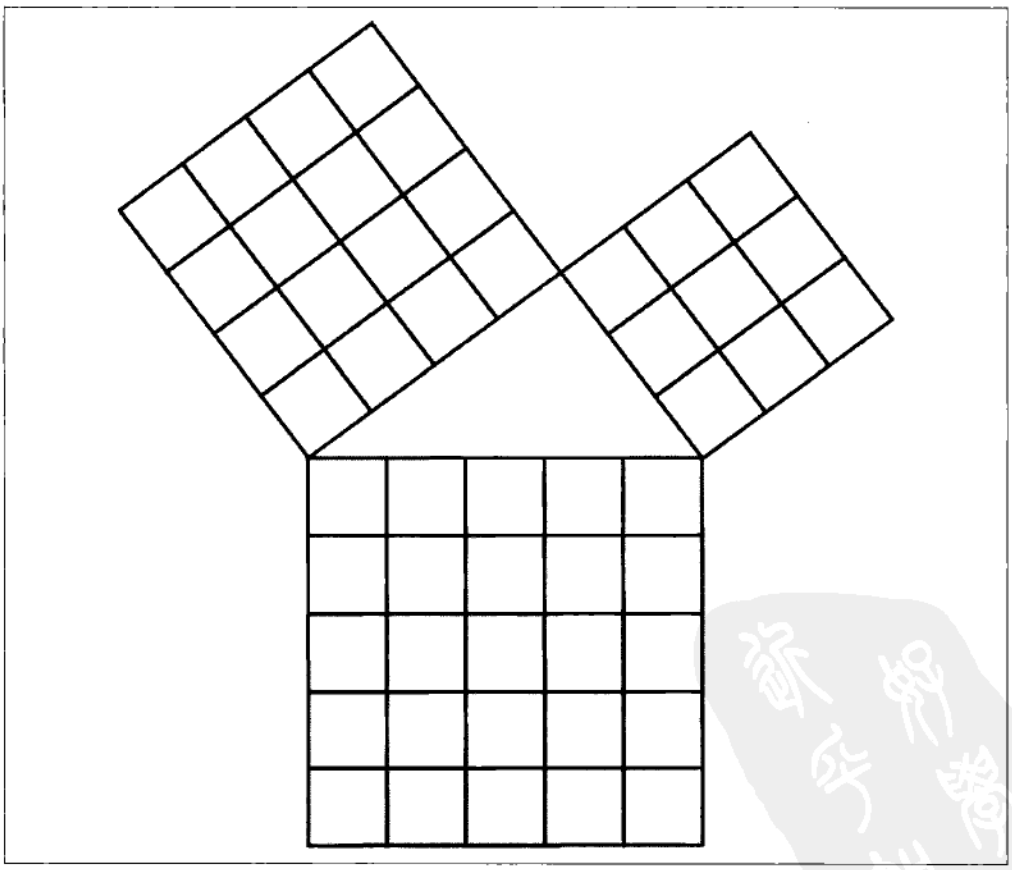


图 7-2　勾股定理

斜边就是直角三角形的最长边（在图 7-2 中，就是三角形底部的那条边，正对着顶

部的直角)。无论三角形的斜边在哪里，该定理都适用。这些和我们的向量有什么关系呢？假设图 7-2 中三角形的两条短边分别是向量的 x、y 坐标。向量的长度的平方等于三角形的斜边的平方：

$$\text{length}^2 = x^2 + y^2$$

或者在 JavaScript 中是：

```
lengthSquared = (x*x + y*y);
```

向量的长度的平方是有用的，但是我们需要的是向量的实际长度。我们可以通过长度的平方的平方根来计算出向量的实际长度。

$$\text{length} = \sqrt{(x^2 + y^2)}$$

或者在 JavaScript 中是：

```
length = Math.sqrt(x*x + y*y);
```

图 7-3 表示了一个横坐标为 x = −3，y = 3 的向量。把这些量带入勾股定理，我们计算出长度大约是 4.24。

```
length = Math.sqrt(-3*-3 + 3*3);    // length = 4.24.
```

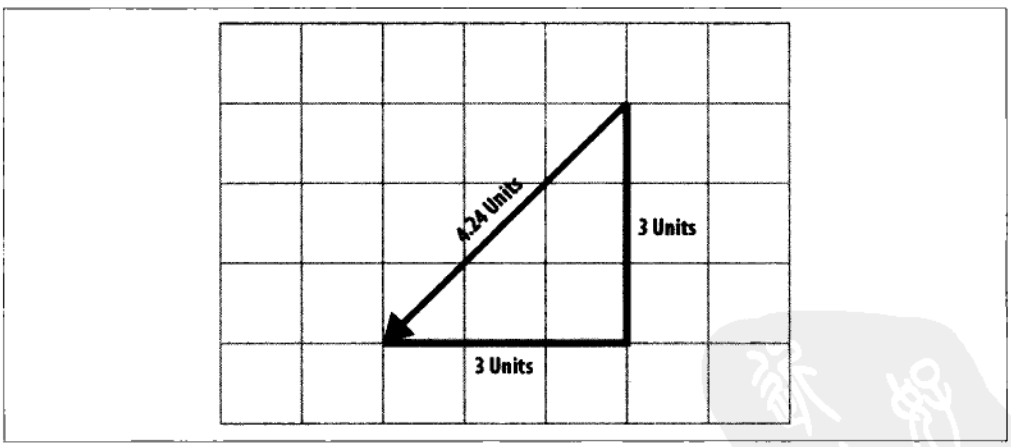



图 7-3　勾股定理可以根据一个向量的 x，y 坐标计算出它的长度

# 7.1　向量运算

我们可以把一些有用的运算应用于向量，在下面的部分列出了一些简单的向量运算以及一些可能的应用程序。

### 7.1.1 加法和减法

你可以通过加减向量的 x，y 坐标来实现向量的相加或相减。这就和常规的计算是一样的，所以，一个向量和自己相加可以使它的长度扩大两倍，一个向量减掉它自己会得到零向量。一些例子包括：

- 给飞行的皮球加上一个重力向量，让它真实地下落
- 把两个碰撞物体的向量加在一起，计算逼真的碰撞反馈
- 给宇宙飞船增加一个火箭推力的向量，让宇宙飞船移动

### 7.1.2 缩放

按照一定比例缩放向量的 x、y 坐标，你就可以缩小或放大向量的长度。一些实例包括：

- 反复以略小于 1 的向量缩放移动向量，让对应物体缓慢地停下来
- 将大炮的方向向量扩大，作为一个发射炮弹的初始向量

### 7.1.3 标准化

有时需要将一个向量变为单位长度，或者说将向量长度变为一个单位。这个过程叫做标准化，单位长度的向量叫做单位向量。当我们感兴趣的是向量的方向而不是长度时，我们经常会这样做。单位向量可以表示：

- 定向喷射的方向
- 斜坡的倾斜方向
- 大炮的发射方向

一旦有了单位向量，我们可以将其缩放，使其代表喷射推力或者炮弹的初始速度。

### 7.1.4 旋转

以任意的角度来旋转一个向量是非常有用的，因为这可以让你指向任意一个你想要的方向。实例包括：

- 使一个对象始终指向另一个
- 更改一个虚拟的喷射引擎的推力方向

- 根据发射器的方向改变投射体的初始发射方向

在 JavaScript 数学函数中，角是以弧度（radian）为单位，而不是我们熟悉的角度（degree）指定的。1 弧度是弧长和半径相等的弧（如图 7-4 所示）。圆的周长可以用公式 $2\pi r$ 来计算，其中 r 代表圆的半径。因此，圆的弧度是 $2\pi$（大约是 6.282）。

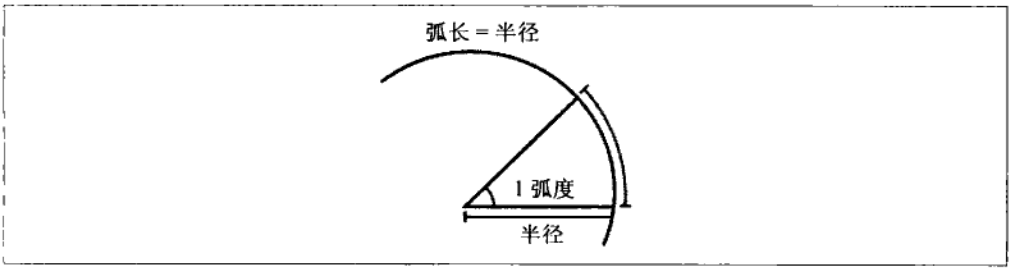



图 7-4　1 弧度

弧度看起来不是特别的直观，但是利用 JavaScript 中的这些函数很容易将弧度和角度进行转化：

```
// Degrees to radians.
degToRad =  function(deg) {
    return deg * (Math.PI/180);
};

// Radians to degrees.
radToDeg = function(rad) {
     return rad * (180/Math.PI);
};
```

弧度和角度的另外一个不同点就是 0 弧度实际上是沿着横坐标轴指向右侧，而 0 角度通常被认为是沿着纵坐标轴指向上方。

## 7.1.5　向量的点乘

点乘给出两个向量之间的角度的余弦，或者说点乘告诉我们两个向量的方向相近程度。点乘的结果在−1 到 1 之间变化（假设是单位向量）。这是点乘结果含义的一些例子：

- 向量的方向相同：点积 = 1
- 向量的夹角是 45°：点积 = 0.5
- 向量的夹角是 90°：点积 = 0

- 向量的夹角是 180°：点积 = −1

点乘可以让我们知道两个物体面向对方的程度。比如，在游戏中，我们可以根据点乘判断两个角色是否可以“看到”对方，或者某形状的某一边是否指向某个方向。

## 7.2 创建一个 JavaScript 向量对象

为了在 JavaScript 中充分利用向量，我们可以把前面描述的函数总结成一个可以反复利用的向量对象，这样在程序中使用向量就会非常方便。需要时我们可以加入更多与向量相关的功能。

向量的 x 和 y 分量在此对象中被称做 vx 和 vy。这使得我们以后在处理一些含有其他 x、y 变量的代码时，能够清除地区分开：

```
var vector2d = function (x, y) {

    var vec = {
        // x and y components of vector stored in vx,vy.
        vx: x,
        vy: y,

        // scale() method allows us to scale the vector
        // either up or down.
        scale: function (scale) {
            vec.vx *= scale;
            vec.vy *= scale;
        },

        // add() method adds a vector.
        add: function (vec2) {
            vec.vx += vec2.vx;
            vec.vy += vec2.vy;
        },

        // sub() method subtracts a vector.
        sub: function (vec2) {
            vec.vx -= vec2.vx;
            vec.vy -= vec2.vy;
        },

        // negate() method points the vector in the opposite direction.
        negate: function () {
            vec.vx = -vec.vx;
            vec.vy = -vec.vy;
        },

        // length() method returns the length of the vector using Pythagoras.
        length: function () {
```

```
            return Math.sqrt(vec.vx * vec.vx + vec.vy * vec.vy);
        },

        // A faster length calculation that returns the length squared.
        // Useful if all you want to know is that one vector is longer than another.
        lengthSquared: function () {
            return vec.vx * vec.vx + vec.vy * vec.vy;
        },

        // normalize() method turns the vector into a unit length vector
        // pointing in the same direction.
        normalize: function () {
            var len = Math.sqrt(vec.vx * vec.vx + vec.vy * vec.vy);
            if (len) {
                vec.vx /= len;
                vec.vy /= len;
            }
            // As we have already calculated the length, it might as well be
            // returned, as it may be useful.
            return len;
        },

        // Rotates the vector by an angle specified in radians.
        rotate: function (angle) {
            var vx = vec.vx,
                vy = vec.vy,
                cosVal = Math.cos(angle),
                sinVal = Math.sin(angle);
            vec.vx = vx * cosVal - vy * sinVal;
            vec.vy = vx * sinVal + vy * cosVal;
        },

        // toString() is a utility function for displaying the vector as text,
        // a useful debugging aid.
        toString: function () {
            return '(' + vec.vx.toFixed(3) + ',' + vec.vy.toFixed(3) + ')';
        }
    };
    return vec;
};
```

## 7.3 使用向量的大炮模拟

我们现在已经定义了向量对象，可以利用它创建一个简单的大炮模拟（如图 7-5 所示）。首先，我应该限定一下“模拟”一词的概念：我们的目标不是尝试绝对真实地复制大炮，而是要创建一个足够现实的模拟，用于应用程序如游戏。即使游戏中最高级的物理模拟也得在一定程度上去掉一些现实。例如，游戏中的人物角色不模拟直立行走的物理过程，游戏中的飞机也不模拟保持悬浮的物理过程。

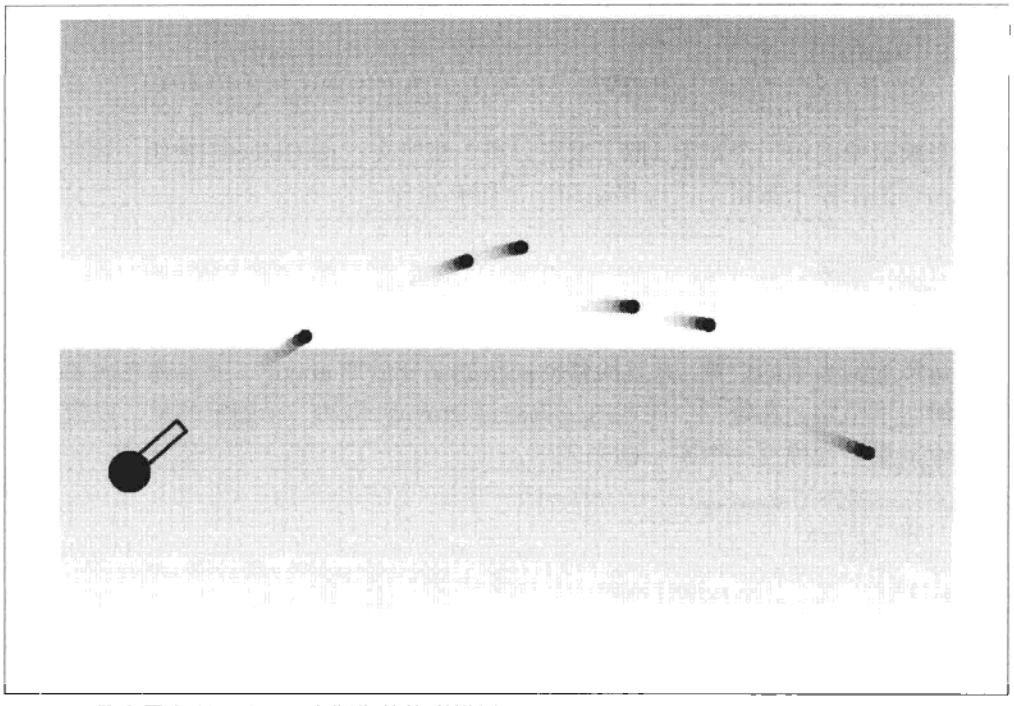


图 7-5　用向量和 HTML5 画布制作的炮弹模拟

**提示**

严格地说，为了达到精确的模拟效果，你应该在计算中考虑每一帧画面的经过时间。不过在这个演示中，我们假设了一个 30 毫秒的帧速率。实际上，某些浏览器上的计时器反正不是特别准确，所以缺乏时间的计算是没有多大损失的。

这个模拟使用 HTML5 的画布来绘制图形，你也加入浏览器中其他的渲染方法（SVG、CSS3 等）。为了使代码集中于对向量的使用和相关计算，这里仅使用了基本的图形。

大炮模拟中使用向量来：

- 表示大炮的瞄准方向；
- 表示炮弹的运动（最初来源于大炮的瞄准方向）。

## 7.3.1　模拟范围的变量

在主模拟函数的顶部我们定义少量的模拟范围的变量。尽管这些变量在模拟的所有

函数中都可用，但是它们被封装在主模拟函数中而并不是全局变量。

```
var gameObjects = [],                    // An array of game objects.
    canvas = document.getElementById('canvas'), // A reference to the Canvas.
    ctx = canvas.getContext('2d');       // A reference to the drawing context.
```

我们将模拟中的每一个对象（除了背景以外）都加入 gameObjects[]数组。然后模拟的主循环遍历整个数组来实现移动和绘制所有对象。

## 7.3.2 炮弹

我们初始化炮弹时需要传入初始 x 和 y 位置以及移动向量。在每一个周期中，我们将向量添加到当前的位置，并且给移动向量的 y 分量加 gravity，使炮弹在向前运动的同时向下坠落。在每一个周期内，我们给 gravity 增加一个固定的量，以模拟重力加速效果。炮弹用一个实心圆来表示。

```
var cannonBall = function (x, y, vector) {
    var gravity = 0,
        that = {
            x: x,                        // Initial x position.
            y: y,                        // Initial y position.
            removeMe: false,             // A flag to indicate removal.

            // move() method updates position with velocity,
            // and checks for cannonball hitting the ground.
            move: function () {
                vector.vy += gravity;    // Add gravity to vertical velocity.
                gravity += 0.1;          // Increase gravity.
                that.x += vector.vx;     // Add velocity vector to position.
                that.y += vector.vy;

                // When cannonball gets too low, flag it for removal.
                if (that.y > canvas.height - 150) {
                    that.removeMe = true;
                }
            },
            // draw() method draws a filled circle, centered on the position
            // of the ball.
            draw: function () {
                ctx.beginPath();
                ctx.arc(that.x, that.y, 5, 0, Math.PI * 2, true);
                ctx.fill();
                ctx.closePath();
            }
        };
    return that;
};
```

## 7.3.3 大炮

大炮用安装在车轮上的一个简单的矩形炮管来表示，它的轴始终对准鼠标指针。要

计算对鼠标指针的角度，我们使用 Math.atan2(y,x)函数。Math.atan2(y,x)函数返回水平轴和与点(x,y)之间的弧度。假设水平轴通过大炮的支点，输入应该是鼠标指针位置相对于大炮的支点位置：

```
angle = Math.atan2(mouseY - cannonY, mouseX - cannonX);
```

当你点击鼠标的时候，大炮会发射一颗炮弹。炮弹会有一个初始位置（大炮的支点）和一个移动向量。我们通过鼠标指针的位置相对于大炮的位置来计算移动的向量。

```
vector = vector2d(mouseX - cannonX, mouseY - cannonY);
```

尽管这个向量的方向是正确的，但是它的长度是炮弹到鼠标指针的距离。这是没有用处的，因为这个距离是变化的：不能被简单地放大或者缩小一个固定的量。解决的办法是将其标准化为单位向量，然后缩放到想要的长度。

```
vec.normalize();   // Make it unit length.
vec.scale(25);     // Scale it up to 25 units.
```

下面是一个完整的大炮对象：

```
var cannon = function (x, y) {
    var mx = 0,
        my = 0,
        angle = 0,
        that = {
            x: x,
            y: y,
            angle: 0,
            removeMe: false,

            // move() method does nothing more than angle the cannon
            // toward the mouse pointer.
            move: function () {
                // Calculate angle to mouse pointer.
                angle = Math.atan2(my - that.y, mx - that.x);
            },

            draw: function () {
                ctx.save();
                ctx.lineWidth = 2;
                //  Origin will be bottom-center of barrel.
                ctx.translate(that.x, that.y);

                // Apply the rotation previously calculated in the
                // move() method.
                ctx.rotate(angle);
                // Draw a rectangular 'barrel'.
                ctx.strokeRect(0, -5, 50, 10);

                // Draw 'wheel' at bottom of cannon.
                ctx.moveTo(0, 0);
                ctx.beginPath();
                ctx.arc(0, 0, 15, 0, Math.PI * 2, true);
```

```
                ctx.fill();
                ctx.closePath();
                ctx.restore();
            }
        };

    // When mouse is clicked, fire a cannonball.
    canvas.onmousedown = function (event) {
        // Create a vector from cannon postion in direction of mouse.
        var vec = vector2d(mx - that.x, my - that.y);
        vec.normalize(); // Make it unit length.
        vec.scale(25);   // Scale it up to 25 units per frame.
        // Create a new cannonball, and add it to the gameObjects list.
        gameObjects.push(cannonBall(that.x, that.y, vec));
    };

    // Keep a note of the mouse position over the canvas.
    canvas.onmousemove = function (event) {
        var bb = canvas.getBoundingClientRect();
        mx = (event.clientX - bb.left);
        my = (event.clientY - bb.top);
    };

    return that;
};
```

## 7.3.4 背景

眼光锐利的人很可能已经注意到在图 7-5 中，炮弹在空中飞的时候有尾巴。我们能获得这个效果是因为在蓝天绿地背景上有趣地使用了画布的 globalAlpha 属性。通常情况下，当我们用画布处理动画时，我们每一帧都需要重绘整个画面来擦除前一帧的图像。如果我们不这样做，画布上会留下之前的痕迹，所有移动图像都会有拖尾。通过指定背景的 alpha 值，我们只是在一定程度上清除了前一帧。由于这些半透明背景是一层层叠加的，它们最终完全清除了前一帧图像。可以把背景想象成描图纸：一张或者两张描图纸看起来是透明的，但是如果我们一直增加描图纸的数量，一堆描图纸看起来就不会是透明的了。最终效果是任何移动的物体会留下一条看起来像是运动模糊的尾巴。背景的 alpha 值越小，尾巴消失所用的时间就越长。

```
// Draws a blue sky and grass, with the horizon in the middle of the canvas.
// Drawn as semitransparent to give the illusion of blurring on moving objects.
var drawSkyAndGrass = function (){
    ctx.save();
    // Set transparency.
    ctx.globalAlpha = 0.4;
    // Create a CanvasGradient object in linGrad.
    // The gradient line is defined from the top to the bottom of the canvas.
    var linGrad = ctx.createLinearGradient(0, 0, 0, canvas.height);
    // Start off with sky blue at the top.
```

```
        linGrad.addColorStop(0, '#00BFFF');
        // Fade to white in the middle.
        linGrad.addColorStop(0.5, 'white');
        // Green for the top of the grass.
        linGrad.addColorStop(0.5, '#55dd00');
        // Fade to white at the bottom.
        linGrad.addColorStop(1, 'white');
        // Use the CanvasGradient object as the fill style.
        ctx.fillStyle = linGrad;
        // Finally, fill a rectangle the same size as the canvas.
        ctx.fillRect(0, 0, canvas.width, canvas.height);
        ctx.restore();
    };
```

## 7.3.5 主循环

我们将主循环封装在 setInterval()的一个匿名函数中。主循环每 30 毫秒被处理一次，它调用了模拟对象中的 move()函数和 draw()函数，同时还创建一个没有设置 removeMe 标志的对象列表。所有设置 removeMe 标志的物体将不再被包括在新列表中，因此就从模拟对象中消失了。炮弹到地平线以下后将用此方法被移除。

## 7.3.6 页面布局

这里是大炮模拟的最终页面布局，注意，为了避免重复，我删除了一些函数代码。只需要用本章前面提到的完整的适当函数来代替就可以了。

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript" >
    window.onload = function() {
        var gameObjects = [],
            canvas = document.getElementById('canvas'),
            ctx = canvas.getContext('2d');

        var vector2d = function (x, y) {
            /*** Code Removed For Conciseness ****/
        };

        var cannonBall = function (x, y, vector) {
            /*** Code Removed For Conciseness ****/
        };

        var cannon = function (x, y) {
            /*** Code Removed For Conciseness ****/
        };

        var drawSkyAndGrass = function (){
            /*** Code Removed For Conciseness ****/
```

```
        };

        // Add an initial cannon to the game objects list.
        gameObjects.push(cannon(50,canvas.height-150));

        // This is the main loop that moves and draws everything.

        setInterval( function() {
            drawSkyAndGrass();

            // Here, we loop through all the object in the gameObjects[]
            // Array. As each object is found, it is drawn, moved, and then
            // added to the gameObjectsFresh[] array,UNLESS it has its removeMe flag
            // set. gameObjectsFresh[] is then copied into gameObjects[] ready for
            // the next frame. gameObjects[] will now not contain any removed
            // objects, and they will disappear, as nothing references them anymore.
            gameObjectsFresh = [];
            for(var i=0;i<gameObjects.length;i++) {
                gameObjects[i].move();
                gameObjects[i].draw();
                if ( gameObjects[i].removeMe === false) {
                    gameObjectsFresh.push(gameObjects[i]);
                }
            }
            gameObjects = gameObjectsFresh;

        },30);
    };
</script>

</head>
    <body>
        <canvas id = "canvas" width = "640" height = "480" style="border:1px solid">
            No HTML5 Canvas detected!
        </canvas>
    </body>
</html>
```

## 7.4 火箭模拟

下面的火箭模拟（如图 7-6 所示）是一个更详尽的向量使用示范。这个模拟主要包括一个可操纵的火箭和五颜六色的障碍物。火箭会时时刻刻转向鼠标指针，你可以通过按下鼠标左键来启动发动机推力。这个是模拟是没有重力和摩擦的，所以它要求的是灵活地使用鼠标来控制火箭的驾驶方向。我们之前在大炮模拟中定义的向量对象在这个火箭模拟中也可以使用。

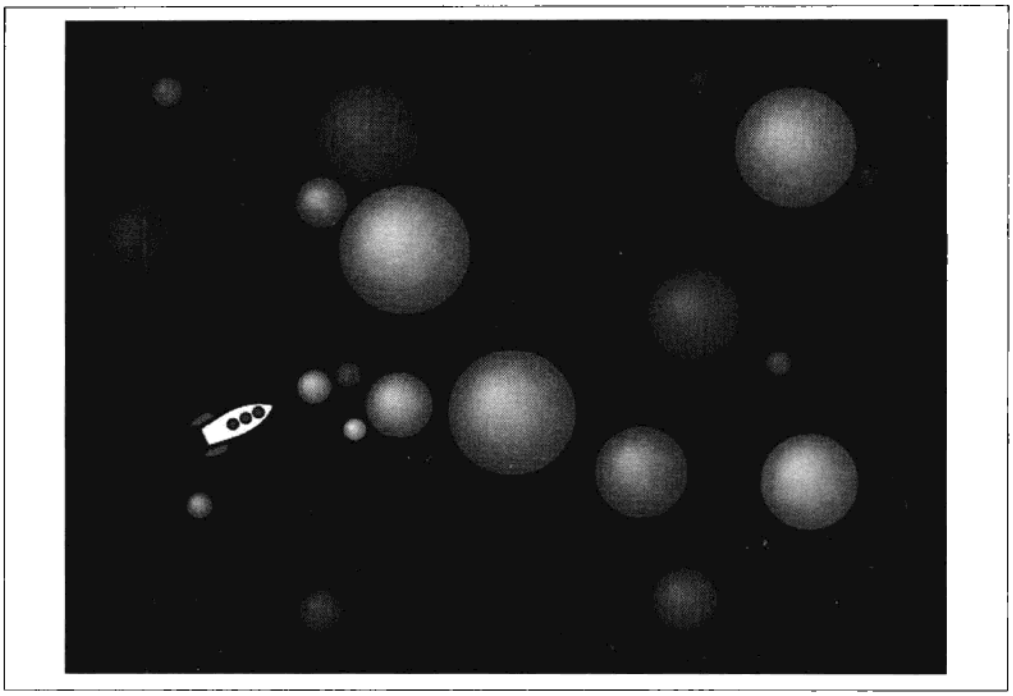


图 7-6　模拟运动中的火箭

## 7.4.1　游戏对象

在火箭模拟中我们使用函数继承从 gameObject 中来创建火箭和障碍物，gameObject 提供了一些公共的方法和属性。

```
var gameObject = function (x, y, radius, mass) {
    var that = {
        x: x,
        y: y,
        vel: vector2d(0, 0),
        radius: radius,
        mass: mass,
        removeMe: false,

        move: function () {
            that.x += that.vel.vx;
            that.y += that.vel.vy;
            if (that.vel.vx < 0 && that.x < -50) {
                that.x += canvas.width + 100;
            } else if (that.vel.vx > 0 && that.x > canvas.width + 50) {
                that.x -= canvas.width + 100;
            }
            if (that.vel.vy < 0 && that.y < -50) {
                that.y += canvas.height + 100;
            } else if (that.vel.vy > 0 && that.y > canvas.height + 50) {
```

```
                that.y -= canvas.height + 100;
            }
        },

        draw: function () {
            return;
        }
    };
    return that;
};
```

从本质上讲，该对象代表一个有半径和质量的球。我们初始化 gameObject 时需要传入其 x、y 位置，半径和质量。我们使用速度向量 vel 保存 gameObject 的当前方向和移动速度，其初始化值为 0（静止）。

move()方法给 gameObject 目前的 x、y 位置增加了速度向量 vel 来移动火箭。move()方法中其他的测试和计算则保证 gameObject 在画布内部。例如，一个刚刚从画布右侧飘走的物体将在画布左侧出现。这使得整个模拟更加流畅，避免物体不断地在画布边缘反弹的现象出现。

gameObject 的 draw()方法其实是一个虚拟函数。火箭和障碍物会用它们自己的 draw()方法实现来覆盖这个函数，并真正绘制一些东西。

## 7.4.2 障碍物对象

障碍物对象将继承 gameObject 的所有方法和属性，但多了一个合适的 draw()方法并执行一些额外的设置。我们初始化一个 gameObject，并给第四个参数（质量）设为和半径相同的值，使得障碍物的质量和大小成比例。创建 gameObject 后，我们创建一个随机颜色 randColor1。然后我们将 randColor2 设为和 randColor1 同样的颜色，但亮度为 randColor1 的一半：>>1 表达式是位右移，和除以 2 等价。字符串 slice()函数的调用是为保证颜色总是以完整的 6 个十六进制数字表示的（#123456）。

注意我们没有定义 move()方法，因为从 gameObject 中继承来的方法已经有了我们需要的功能。

draw()方法用传入的半径绘制一个圆，并且使用 randColor1 和 randColor2 的径向渐变填充这个圆。randColor1 是这两个颜色中较浅的一个，用做实心球形的高亮颜色。高亮的位置稍稍偏向圆的左上方，高亮的半径是障碍物半径的 1/8。最后我们应用 3 个像素的黑色描边。

```
var obstacle = function (x, y, radius) {
    var that = gameObject(x, y, radius, radius),
        randColor1 = Math.floor(Math.random()*0xffffff),
```

```
        randColor2 = ((randColor1 & 0xfefefe)>>1).toString(16);
    randColor1 = randColor1.toString(16);
    randColor1 = '#000000'.slice(0,7-randColor1.length) + randColor1;
    randColor2 = '#000000'.slice(0,7-randColor2.length) + randColor2;

    that.draw = function () {
        ctx.beginPath();
        var radgrad = ctx.createRadialGradient(that.x, that.y, radius,
            (that.x - (radius / 4)), (that.y - (radius / 4)), (radius / 8)   );
        radgrad.addColorStop(0, randColor2);
        radgrad.addColorStop(1, randColor1);
        ctx.fillStyle = radgrad;
        ctx.arc(that.x, that.y, that.radius, 0, Math.PI * 2, true);
        ctx.fill();
        ctx.strokeStyle = '#000';
        ctx.lineWidth = 3;
        ctx.stroke();
        ctx.closePath();
    };
    return that;
};
```

## 7.4.3 火箭物体

draw()方法占据了火箭物体的大部分代码。事实上，draw()方法使用的是从 Adobe Illustrator 的 AI→Canvas 插件输出的代码，这个代码可能比人工编写的代码长。但是它非常节省时间，所以在这个例子中我们使用的是未优化的输出代码。

move()函数给火箭的速度增加了一个推力向量，并且限制在 5 个单位以内，以防止火箭嗖嗖跑得太快。

我们使用 3 个鼠标事件来控制火箭：

onmousedown

这将在鼠标指针的方向创建一个推力向量。我们通过在火箭和鼠标指针建立一个向量来计算推力向量，将其标准化到单位长度，然后缩放到适当的长度。

onmouseup

这将推力向量设为 0，阻止推力再被加到火箭的速度上去。

onmouseover

这记录了鼠标在画布上移动的位置，并将其保存在 mx 和 my 中。鼠标指针的角度也保存下来，以便以正确的角度绘制出火箭，使其面对着鼠标指针。

```
var rocket = function (x, y) {
    // mx and my store the mouse position over the canvas.
```

```
var mx = 0,
    my = 0,
    // Initial angle and thrust vector are zero.
    angle = 0,
    thrust = vector2d(0, 0),
    // gameObject is initialized with radius of 15 and mass of 15.
    that = gameObject(x, y, 15, 15),
    // Keep a reference to the parent (gameObject) move() method,
    // so it can be called in the overridden move() method later on.
    move = that.move;

// Method to draw a rocket.
// Output generated by AI->Canvas plug-in for Adobe Illustrator.
that.draw = function () {
    ctx.save();
    ctx.translate(that.x, that.y);
    ctx.rotate(angle);
    ctx.scale(0.5, 0.5);
    ctx.beginPath();
    ctx.moveTo(-49.5, -16.0);
    ctx.lineTo(-48.9, 16.5);
    ctx.bezierCurveTo(-10.0, 19.9, 32.4, 31.4, 68.3, -1.6);
    ctx.bezierCurveTo(31.3, -33.5, -10.9, -21.8, -49.5, -16.0);
    ctx.closePath();
    ctx.fillStyle = "rgb(255, 255, 0)";
    ctx.fill();
    ctx.lineWidth = 6.0;
    ctx.lineJoin = "round";
    ctx.stroke();

    ctx.beginPath();
    ctx.moveTo(40.1, 5.6);
    ctx.bezierCurveTo(36.1, 5.7, 32.8, 2.5, 32.7, -1.4);
    ctx.bezierCurveTo(32.7, -5.3, 35.8, -8.6, 39.8, -8.7);
    ctx.bezierCurveTo(39.8, -8.7, 39.8, -8.7, 39.8, -8.7);
    ctx.bezierCurveTo(43.8, -8.7, 47.1, -5.6, 47.2, -1.6);
    ctx.bezierCurveTo(47.2, 2.3, 44.1, 5.6, 40.1, 5.6);
    ctx.bezierCurveTo(40.1, 5.6, 40.1, 5.6, 40.1, 5.6);
    ctx.closePath();
    ctx.fillStyle = "rgb(0, 127, 127)";
    ctx.fill();
    ctx.lineWidth = 3.6;
    ctx.stroke();

    ctx.beginPath();
    ctx.moveTo(19.7, 5.9);
    ctx.bezierCurveTo(15.7, 6.0, 12.4, 2.9, 12.4, -1.1);
    ctx.bezierCurveTo(12.3, -5.0, 15.5, -8.3, 19.5, -8.3);
    ctx.bezierCurveTo(19.5, -8.3, 19.5, -8.3, 19.5, -8.3);
    ctx.bezierCurveTo(23.5, -8.4, 26.7, -5.3, 26.8, -1.3);
    ctx.bezierCurveTo(26.9, 2.6, 23.7, 5.9, 19.7, 5.9);
    ctx.bezierCurveTo(19.7, 5.9, 19.7, 5.9, 19.7, 5.9);
    ctx.closePath();
    ctx.fill();
    ctx.stroke();
    ctx.beginPath();
```

```
    ctx.moveTo(-1.0, 6.3);
    ctx.bezierCurveTo(-4.9, 6.3, -8.2, 3.2, -8.3, -0.7);
    ctx.bezierCurveTo(-8.4, -4.7, -5.2, -7.9, -1.2, -8.0);
    ctx.bezierCurveTo(-1.2, -8.0, -1.2, -8.0, -1.2, -8.0);
    ctx.bezierCurveTo(2.8, -8.1, 6.1, -4.9, 6.2, -1.0);
    ctx.bezierCurveTo(6.2, 3.0, 3.0, 6.2, -0.9, 6.3);
    ctx.bezierCurveTo(-1.0, 6.3, -1.0, 6.3, -1.0, 6.3);
    ctx.closePath();
    ctx.fill();
    ctx.stroke();

    ctx.beginPath();
    ctx.moveTo(-49.5, -16.0);
    ctx.lineTo(-68.3, -25.1);
    ctx.bezierCurveTo(-56.3, -31.0, -39.9, -37.8, -29.5, -35.3);
    ctx.bezierCurveTo(-22.7, -33.7, -14.5, -21.6, -14.5, -21.6);
    ctx.lineTo(-49.5, -16.0);
    ctx.closePath();
    ctx.fillStyle = "rgb(255, 0, 0)";
    ctx.fill();
    ctx.lineWidth = 6.0;
    ctx.stroke();

    ctx.beginPath();
    ctx.moveTo(-47.9, 16.4);
    ctx.lineTo(-66.4, 26.2);
    ctx.bezierCurveTo(-54.3, 31.7, -37.7, 38.0, -27.4, 35.2);
    ctx.bezierCurveTo(-20.6, 33.3, -12.8, 21.0, -12.8, 21.0);
    ctx.lineTo(-47.9, 16.4);
    ctx.closePath();
    ctx.fill();
    ctx.stroke();
    ctx.restore();
};
that.move = function () {
    var speed;
    // Calculate angle to mouse pointer.
    angle = Math.atan2(my - that.y, mx - that.x);
    // Add thrust to current velocity.
    that.vel.add(thrust);
    speed = that.vel.length();
    // If speed is > 5, then scale velocity back down.
    if (length > 5) {
        that.vel.normalize();
        that.vel.scale(5);
    }
    move();
};
// When mouse is held down, thrust.
canvas.onmousedown = function (event) {
    // Create a vector from rocket postion in direction of mouse.
    thrust = vector2d(mx - that.x, my - that.y);
    thrust.normalize(); // Make it unit length.
        thrust.scale(0.1); // Scale it down.
    };
    // When mouse is released, cancel thrust.
```

```
    canvas.onmouseup = function (event) {
        thrust = vector2d(0, 0);
    };

    // Keep a note of the mouse position over the canvas.
    canvas.onmousemove = function (event) {
        var bb = canvas.getBoundingClientRect();
        mx = (event.clientX - bb.left);
        my = (event.clientY - bb.top);
    };

    return that;
};
```

## 7.4.4 背景

drawBackground()函数用一个深蓝到紫色的渐变填充画布作为太空背景。因为填充了和画布一样大小的背景，我们不需要每帧清除画布。

```
// Draws a spacey-looking background - a dark blue gradient fading to dark purple
// in the middle.
var drawBackground = function (){
    ctx.save();
    // Create a CanvasGradient object in linGrad.
    // The gradient line is defined from the top to the bottom of the canvas.
    var linGrad = ctx.createLinearGradient(0, 0, 0, canvas.height);
    // Start off with dark blue at the top.
    linGrad.addColorStop(0, '#000044');
    // Fade to purple in the middle.
    linGrad.addColorStop(0.5, '#220022');
    // Fade to dark blue at the bottom.
    linGrad.addColorStop(1, '#000044');
    // Use the CanvasGradient object as the fill style.
    ctx.fillStyle = linGrad;
    // Finally, fill a rectangle the same size as the canvas.
    ctx.fillRect(0, 0, canvas.width, canvas.height);
    ctx.restore();
};
```

## 7.4.5 碰撞检测和反馈

碰撞函数首先检查是否有两个游戏对象是重叠的，如果有，则彼此弹开。初始的重叠测试检查游戏对象的位置和半径定义的圆是否碰撞：当两个圆的圆心间距离小于其半径之和时，两圆相交。在图 7-7 中，因为 D1 小于 R1+R2，因此两个圆碰撞。D2 代表了两圆相交的程度。碰撞之后我们把圆移动 D2，以确保它们将不再相交，并重新达到两圆相切的视觉效果。

collideAll()函数使用一个嵌套循环检查每一个物体和其他物体之间的位置关系。这里一个小而重要的优化就是让内层循环从外层循环的当前位置+1 开始。这确保了碰撞检测只是单向的——只有物体 1 对物体 2 检测，而没有物体 2 对物体 1 检测——

这样做可以使所需的测试次数减少一半。

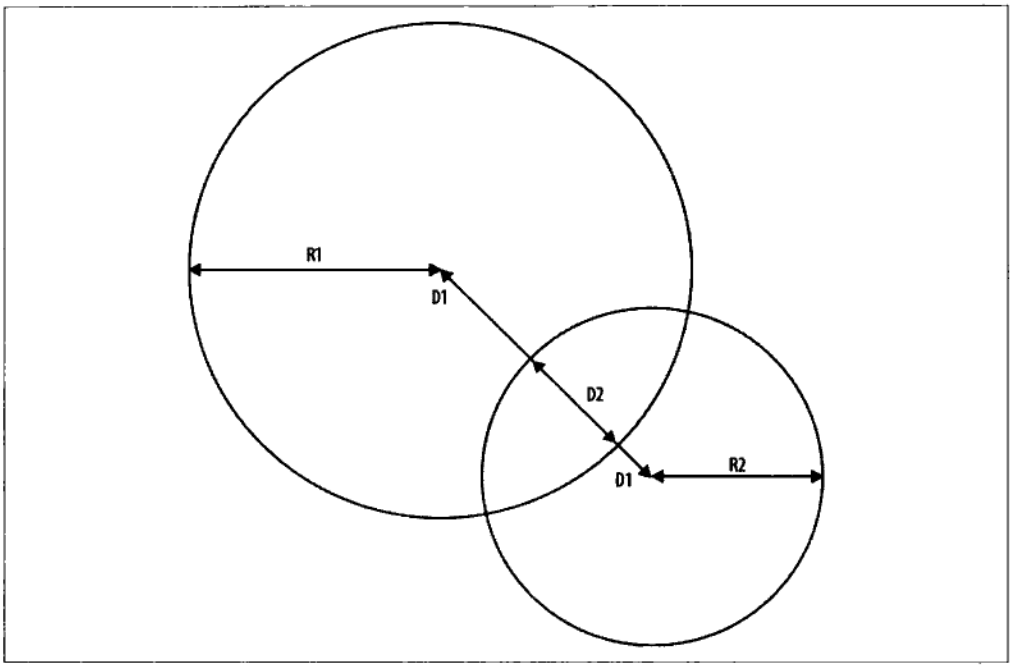



图 7-7　当两圆心之间的距离小于两圆的半径之和时，两圆碰撞

如果游戏物体重叠，要做到以下几点：

1．像前面所描述的，以物体重叠的部分将物体分开。

2．让游戏物体相撞后反弹，计算出新的速度向量：

```
var bounce = function(ball1,ball2) {
    var colnAngle = Math.atan2(ball1.y - ball2.y, ball1.x - ball2.x),
        length1 = ball1.vel.length(),
        length2 = ball2.vel.length(),
        dirAngle1 = Math.atan2(ball1.vel.vy, ball1.vel.vx),
        dirAngle2 = Math.atan2(ball2.vel.vy, ball2.vel.vx),
        newVX1 = length1 * Math.cos(dirAngle1-colnAngle),
        newVX2 = length2 * Math.cos(dirAngle2-colnAngle);
    ball1.vel.vy = length1 * Math.sin(dirAngle1-colnAngle);
    ball2.vel.vy = length2 * Math.sin(dirAngle2-colnAngle);
    ball1.vel.vx =((ball1.mass-ball2.mass)*newVX1 +
        (2*ball2.mass)*newVX2) /
        (ball1.mass+ball2.mass);
    ball2.vel.vx =((ball2.mass-ball1.mass)*newVX2 +
        (2*ball1.mass)*newVX1) /
        (ball1.mass+ball2.mass);
    ball1.vel.rotate(colnAngle);
    ball2.vel.rotate(colnAngle);
```

```
};

var collideAll = function () {
    var vec = vector2d(0, 0),
        dist, gameObj1, gameObj2, c, i;
    // Check each object against every other object.
    for (var c = 0; c < gameObjects.length; c++) {
        gameObj1 = gameObjects[c];
        // The inner loop starts at one past the outer loop.
        // This ensures efficient one-way testing:
        // A against B, but not B against A.
        for (i = c + 1; i < gameObjects.length; i++) {
            gameObj2 = gameObjects[i];
            // Get the distance between the two objects.
            vec.vx = gameObj2.x - gameObj1.x;
            vec.vy = gameObj2.y - gameObj1.y;
            dist = vec.length();
            // If distance < sum of the two radii, then we
            // have a collision.
            if (dist < gameObj1.radius + gameObj2.radius) {
                // Move objects apart so they are no longer intersecting,
                // but flush against each other.
                vec.normalize();
                vec.scale(gameObj1.radius + gameObj2.radius - dist);
                vec.negate();
                gameObj1.x += vec.vx;
                gameObj1.y += vec.vy;
                // Finally, bounce the two colliding objects.
                bounce(gameObj1, gameObj2);
            }
        }
    }
};
```

bounce()函数使用三角函数和弹性碰撞计算求出两个碰撞物体的反弹幅度和方向。这个函数通过旋转运动向量使我们可以进行一维的弹性碰撞计算。计算得到的向量再被旋转回去，得到物体的反弹方向。这仅仅是计算反弹向量的一种方法，还有其他可用的方法。如果你对这其中涉及的数学感兴趣，请在谷歌中搜索 billiard physics 或者 pool physics。

## 7.4.6 页面代码

下面是页面布局代码。为避免重复，我删除了一部分函数。

```
<!DOCTYPE html>
<html>
<head>

<script type="text/javascript" >
    window.onload = function() {
        var gameObjects = [],
            canvas = document.getElementById('canvas'),
            ctx = canvas.getContext('2d');
```

```
// Vector object.
var vector2d = function (x, y) {
    /*** CODE REMOVED FOR CONCISENESS ***/
};


var gameObject = function (x, y, radius, mass) {
        /*** CODE REMOVED FOR CONCISENESS ***/
};

var obstacle = function (x, y, radius) {
    /*** CODE REMOVED FOR CONCISENESS ***/
};

var bounce = function(ball1,ball2) {
    /*** CODE REMOVED FOR CONCISENESS ***/
};

var rocket = function (x, y) {
    /*** CODE REMOVED FOR CONCISENESS ***/
};

var collideAll = function () {
    /*** CODE REMOVED FOR CONCISENESS ***/
};

// Draws a spacey-looking background,
// a dark blue gradient fading to dark purple
// in the middle.
var drawBackground = function (){
    /*** CODE REMOVED FOR CONCISENESS ***/
};

// Add rocket to the game objects list.
gameObjects.push(rocket(50,canvas.height-150));
// Create a bunch of obstacles.
for(var i=0;i<20;i++) {
    var radius = ((Math.random()*4)+1)*10;
    var x = Math.random() * (canvas.width-(radius*2)) +radius;
    var y = Math.random() * (canvas.height-(radius*2))+radius;
    gameObjects.push(obstacle(x,y,radius));
}


// This is the main loop that moves and draws everything.
setInterval( function() {
    var gameObjectsFresh = [];
    drawBackground();
    // Here, we loop through all the object in the gameObjects[]
    // array. As each object is found, it is drawn, moved, and then
    // added to the gameObjectsFresh[] array, UNLESS it has its removeMe flag
    // set. gameObjectsFresh[] is then copied into gameObjects[], ready for
    // the next frame. gameObjects[] will now not contain any removed
    // objects, and they will disappear as nothing references them anymore.
    for(var i=0;i<gameObjects.length;i++) {
        gameObjects[i].move();
```

```
                gameObjects[i].draw();
                if ( gameObjects[i].removeMe === false) {
                    gameObjects[i].push(gameObjects[i]);
                }
            }
            collideAll();
            gameObjects = gameObjectsFresh;
        },30);
    };

</script>

</head>
    <body>
        <canvas id = "canvas" width ="640" height = "480" style="border:1px solid">
            No HTML5 Canvas detected!
        </canvas>
    </body>
</html>
```

## 7.4.7 可能的改进方案

- 给运动的物体添加摩擦，使它们速度减慢并停下来。提示：在游戏物体的 move() 函数中，用一个不小于 1 的数来缩小速度向量。
- 火箭的 draw()方法是绘制了许多形状和轮廓的大代码块。为了提高性能，可以在隐藏的画布元素中绘制一个火箭，然后在画布的 drawImage()函数中把这个隐藏的元素作为一个位图源使用。这将大幅度地调高效率。提示：在火箭中创建一个 drawOnce()方法来在隐藏的画布中绘制火箭。将 draw()方法改为使用 drawImage()。
- 考虑让 drawBackground()函数做更多的事情。尝试加入一些星星或者其他细节。
- collide()函数使用速度较慢的 length()来计算两个物体之间的距离。修改此代码来使用更快的 lengthSquared()函数。提示：你需要比较返回值和两个半径平方的和。
- 开发一套新的火箭控制系统，使用键盘或者不同的鼠标动作，比如拖动。

# 第 8 章

# 谷歌可视化

谷歌图表工具 API（Application Programming Interface）是一个丰富且不断完善的数据可视化工具集合，它能使你的数据更具视觉冲击力。如果你还在使用老式沉闷的饼图和柱状图，应该继续读下去：谷歌图表工具（如图 8-1 所示）中充满交互性、动画和纯粹的乐趣。实际上，谷歌图表工具中除了图表，还有其他工具和特性：

- 地图
- 动态图标
- 拨号盘和仪表盘（o-meter）风格的显示
- 公式
- QR 码（即二维条码、物理世界的超链接）
- 大量的第三方可视化工具
- 创建自定义可视化工具的能力

本章没有详细介绍所有的 API，而是介绍了一些基本要领，使你能更好地使用其官方文档（http://code.google.com/apis/charttools/index.html）进一步学习。我们还将开发一些有用的函数和例子，来帮助你更好地使用谷歌图表工具。

谷歌图表工具可以分成两个不同的部分：

图像图表（即图表 API）

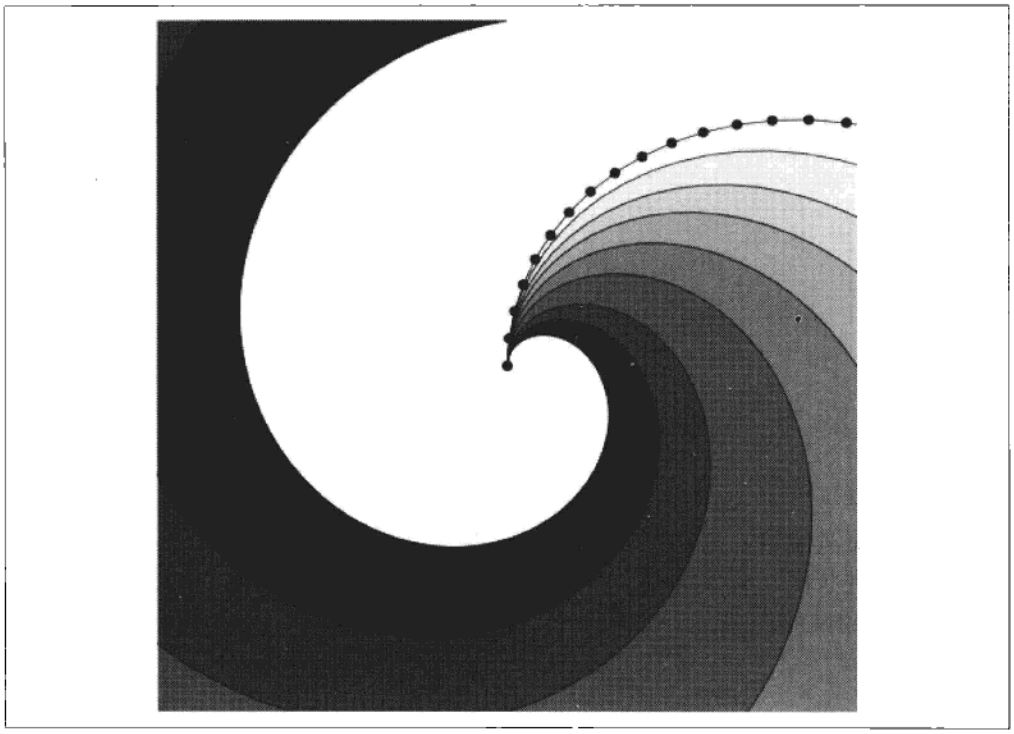


图 8-1　你的会计师不会欣赏这样的图表

图像图表由传给谷歌图表服务器的一个特殊格式的 URL 创建。服务器返回一个可以包含在网页中的静态图像。通常情况下，这个 URL 当作<img>标记的 src 属性值。图像图表易于使用，无需外部库，几乎不用编程就可以工作。但创建 URL 不太直观，需要些技巧。可使用 JavaScript 编程来增加图像图表的实用性和简便性。图 8-2 是图像图表的示例。

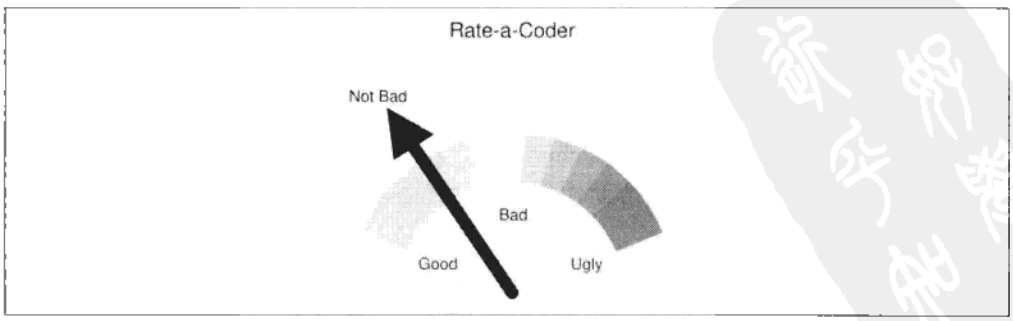



图 8-2　谷歌图表工具可以突出多种类型的数据。这是一个谷歌仪表盘图像图表

互动式图表（即谷歌可视化 API）

互动式图表使用 JavaScript API（作为一个外部库加载），在浏览器中呈现各种动态图表和图形。使用交互式图表时需要一些编程技巧，最大的挑战是从繁多的选项中做出选择。图 8-3 显示了一个交互式图表的例子。

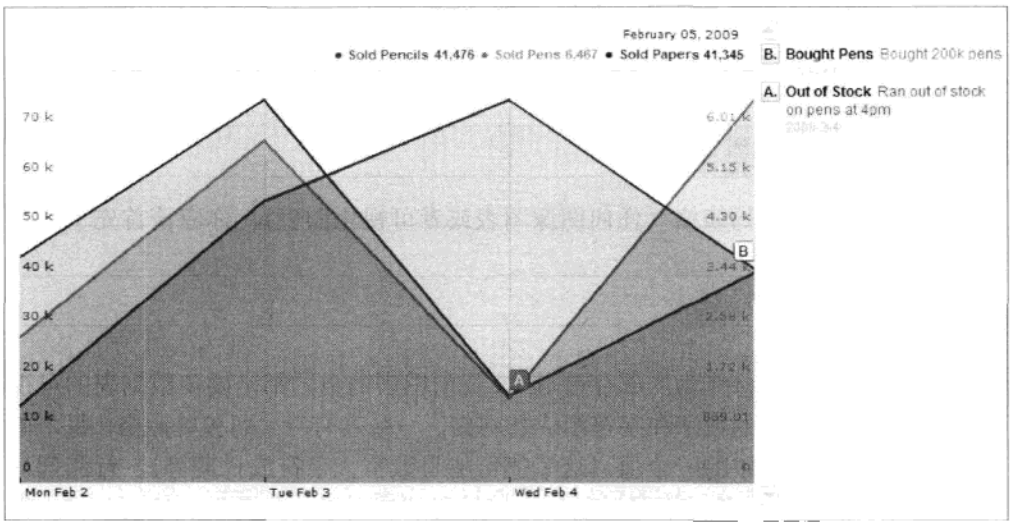



**图 8-3　一个可以用鼠标缩放和拖曳的交互式图表**

本章首先探讨图像图表，然后探讨交互式图表。

## 8.1　限制

谷歌图表工具，尤其是图像图表，有一定局限性：

- 图像图表最大为 300 000 像素，其中最长边不超过 1 000 像素。现实中这对大部分常用图表已经足够了。
- 图像图表 URL 的最大值是 2KB（GET）和 16KB（POST）。使用 AJAX 连接到谷歌图表服务器的交互式图表 URL 没有类似限制。
- 图表请求的最大数量为每天 25 万。如果你的需要超过此数额，则需要联系谷歌。如果你使用的是不变的图片图表，一个解决方法是，将谷歌生成的图像保存在自己的 Web 服务器使用，避免不断从谷歌请求同样的图像。

**提示**

POST 和 GET 都是将数据发送到 Web 服务器的方式。它们之间有什么区别呢？GET 数据通常用于简单的请求，比如浏览器地址栏的 URL 或<img>标签的 src URL（或图像图表 URL）。它通常是在浏览器地址栏或在网页的源代码里可见的。POST 通常用于发送更重要的数据给服务器处理和保存。典型的 POST 例子包括提交表单内容，如信用卡信息或电子邮件。POST 数据内容在正常情况下是不可见的。

## 8.2 相关术语表

无论你使用哪种类型的表格（任何图像图表或者可视化图表），你应该首先了解一些基础元素：

数据表

图表的数据以表格作为内部存储形式，任何图表的目的都是要使数据表的可视性尽量高于基本的数字和字符组成的网格。一张表有行、列和单元格，每个单元格包含这个表中的一个值（这个值可能是数字、字符或日期等）。行和列从 0 开始计数，一个单元格可以通过它的行和列被引用。在表 8-1 中，(0，2) 的单元格含有 75 的值，在 (0，0) 单元格的值是 Monday，在 (2，2) 处的单元格含有数值 35。图 8-4 是用柱形图来表示数据表内容。我在代码示例中刻意使用一些特别的值使其更容易分辨。

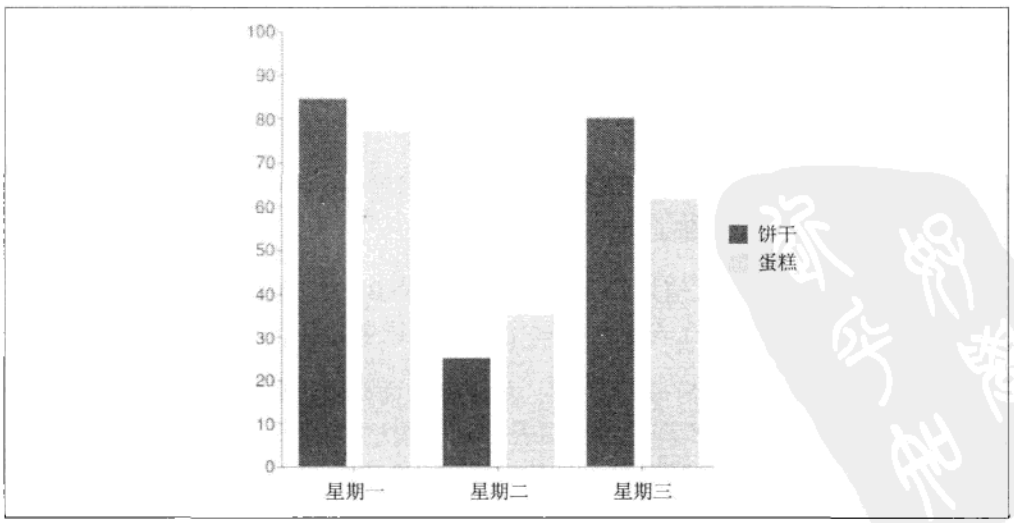



图 8-4 由柱形图表示的两组数据

数据序列

一个数据序列代表数据表中的一组相关数据，任何表格都有一个或多个数据序列。在表 8-1 中，每一列代表一个数据序列：日期、饼干销售额、蛋糕销售额。

表 8-1　3 天内面包店销售情况的示例图表，每一列都代表一个数据序列

| 日　　期 | 饼干销售额 | 蛋糕销售额 |
|---|---|---|
| 星期一 | 90 | 75 |
| 星期二 | 40 | 65 |
| 星期三 | 60 | 35 |

轴标签

轴标签是沿轴方向延伸的文本或数字标签。图 8-4 中的图表具有沿横轴的文本标签和沿纵轴的数字标签。你可以通过指定范围和步长自动生成数字标签。数据系列可能被用来生成轴标签，请注意日期数据系列是怎么生成横轴标签的。

说明

说明是用来描述图表中的数据序列的。图 8-4 中有两种颜色标记的说明（饼干和蛋糕），它们描述出表 8-1 中的相应序列。

## 8.3　图像图表

图像图表可以使没有 JavaScript 编程知识的非程序员用户创造令人印象深刻的图表图像。如果你知道一点 HTML，就可以使用图像图表。相比于交互式图表 API，它无需任何特殊的 JavaScript 库，因为图表是使用普通的 URL 从谷歌图表服务器请求到的。图 8-5 显示了以下 HTML 页面的效果：

```
<html>
    <body>
        <img src = 'https://chart.googleapis.com/chart?
            cht=p3&chd=t:60,40&chs=500x250&chl=Hello|World'/>
    </body>
</html>
```

从这么简短的代码得到这样的效果已经很不错了。但是创建这样的 URL 是一个复杂的过程。值得庆幸的是，如果你所用的数据集是静态的（事先已知的不变值），你可以使用谷歌提供的“图表向导（Chart Wizard）”，使创建图像图表的 URL 的

过程更简便（如图 8-6 所示）。“图表向导”可以在 http://code.google.com/apis/chart/docs/chart_wizard.html 找到。

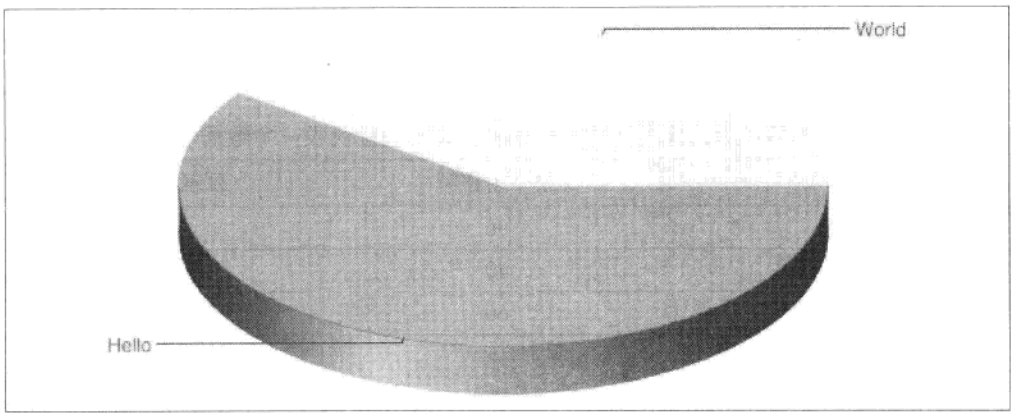



图 8-5　图像图表只需要一个简单的 URL 请求就可以产生理想的结果

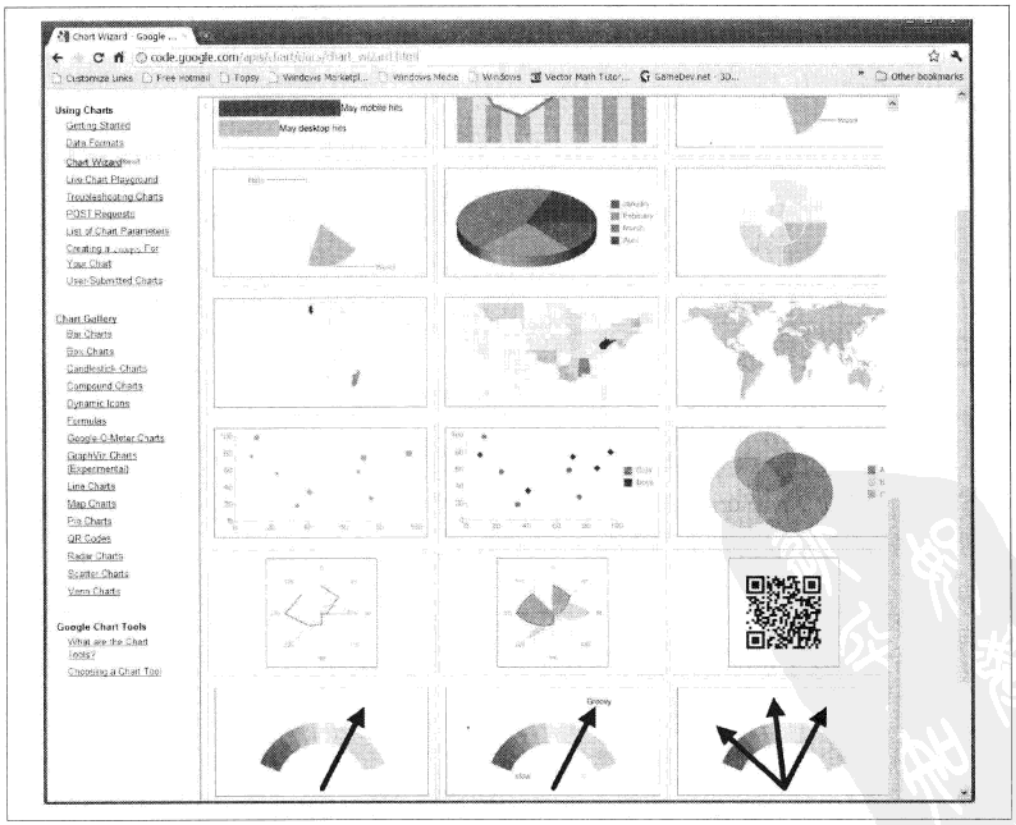



图 8-6　想要图表吗？使用谷歌图表向导成为图表达人吧！

**提示**

在谷歌图表向导中，一个数据序列被称为一个数据集。

虽然图表向导对处理静态数据序列非常好用，但处理动态数据，例如用 JavaScript 从服务器中所读到的未知值时，就没这么好用了。如果你要用带有动态数据的图像表，你需要了解 JavaScript 并理解 URL 的结构。

图像图表的一个缺点是传给谷歌服务器的 URL 比较令人费解。让我们更仔细地观察一下生成图 8-5 中饼图的 URL 格式。

*https://chart.googleapis.com/chart?*

*cht=p3&*

*chs=500x250&*

*chd=t:60,40&*

*chl=Hello|World*

这段代码的细解如下：

*http://chart.googleapis.com/chart?*

所有的图表要求都被发送到这个地址。

*Cht=p3&*

图表的格式（此例中为一个 3D 的饼图）。

*chs=500x250&*

图表的像素，宽 × 高。

*hd=t:60,40&*

饼图中每一片（一个数据系列）数据，以在 0～100 的浮点数值的文本格式给出。还有其他方法来给出 URL 中的数据，将在本章后面讨论。

*chl=Hello|World*

饼图中每一片的标签，这些部分以管道字符作为分隔符给出。

**警告**

在网址中使用连字符（&）会导致 XHTML 编码网页的问题。如果在 XHTML 页面中遇到 WC3 验证或其他问题，可以用&amp 来替换原有符号。

请通过在线文档研究不同图表样式的细节，并设置它们需要的 URL 参数。

### 8.3.1 数据格式及图表分辨率

你可以用以下 4 种方式中的任意一种为图像图表指定数据：

基本文本

如饼图例子中所用，允许范围是 0～100 的浮点数。

自定义范围文本格式

适合没有范围限制的正或负的浮点数字。

简单编码格式

一种紧凑的格式，由单个字符表示 0～61 的整数值。

扩展编码格式

一种紧凑的格式，由两个字符代表 0～4 095 的整数值。

为什么我们需要这些不同的格式呢？因为 URL 限定的大小为 2KB（假定 URL 是以 GET 方式发送的），用两种编码格式可以让图表数据看起来更加紧凑，从而可以让一些更复杂的图表存储在 2KB 的限制空间中。两种非编码的格式则更容易创建、读取，但所占用空间较大。也可以使用 POST 方法通过 HTML 表单提交 16KB 数据。但这种方法需要额外的编程，来提交数据给服务器并显示回复数据。

每种数据格式允许值的范围决定了使用这种格式的图表的分辨率。对于大多数的应用程序，扩张编码格式（4 096 个不同值）是可以提供足够分辨率的。至于简单编码格式，62 个不同的值，也足以应付较小的图表。较大的图表在这个分辨率下仍然可以工作，但你可能会注意到因为粒度有限引起的不准确。

如下 JavaScript 计算可以将任意数据值缩放到可用的分辨率下：

```
scaledValue = resolution * dataValue / maxDataValue;
```

maxDataValue 是指图表中所有未缩放数据值都不超过的上限。我们可以将这个值简单地设定为数据集中的最大值，但对于某些类型的图表，设定一个更大的 maxDataValue 能使图表更美观。比如你使用一个最大值为 185 的数据集，以 200 为 maxDataValue，可以确保垂直轴比垂直条形图的最高一栏更高些。

### 1．基本文本格式

基本文本格式允许设定的范围是浮点数 0～100。小于 0 的值将丢失，大于 100 的值将被缩短。

基本文本格式的语法如下：

```
chd=t:val_1_1,val_1_2,val_1_3|val_2_1,val_2_2,val_2_3|...
```

请注意使用管道字符来分隔数据序列。基本文本格式使用和创建起来很容易，但这种格式占用的空间大，需要注意 2KB 的限制。

### 2．自定义范围文本格式

自定义范围文本格式和基本文本格式相似，但它使用了一个额外的 chds 参数来描述每一个数据序列的范围。其指定的数值范围没有限制，这个范围以最小值-最大值对来表示。

自定义缩放文本格式的语法如下：

```
chd=t:val_1_1,val_1_2,val_1_3|val_2_1,val_2_2,val_2_3|...
chds=<series_1_min>,<series_1_max>,<series_2_min>,<series_2_max>,...
```

如果最小值-最大值对数比数据系列数少，则所有没有被分配范围的数据序列将使用最后一对最小值-最大值作为范围。在很多情况下，图表中的所有数据系列将使用同一个范围，这时我们只需要指定一对最小值-最大值。

### 3．简单编码格式

简单编码格式允许设置 0～61 的整数值。虽然我们可以调整数据来适应这个范围，但有限的粒度（间隔）意味着在较大图表中可能显示不准确。因此这种紧凑的格式更适用于小图表。

简单编码格式的语法如下：

```
chd:s<series_1>,<series_2>,<series_n>,...
```

数据序列的值由单字符来表示，如下：

- A～Z，其中 A = 0, B = 1, C = 2, ... Z = 25

- a～z，其中 a = 26, b = 27, c = 28, ... z = 51
- 0～9，其中 0 = 52, 1 = 53, ... 9 = 61
- 下划线（_）表示空值

基本文本格式的数据：

```
chd=t:1,19,27,53,61,-1|12,39,57,45,51,27
```

用简单编码格式表示为

```
chd=s:BTb19_,Mn5tzb
```

请注意值之间没有分隔符，且用逗号分隔数据序列。下面的函数将一个 JavaScript 的数值数组转换成简单字符串编码：

```
var simpleEncode = function (valueArray, maxValue) {

    var simpleEncoding =
       'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789',
        chartData = '';
    for (var i = 0; i < valueArray.length; i++) {
        var currentValue = valueArray[i];
        if (!isNaN(currentValue) && currentValue >= 0) {
            // Calculate the character for the value, ensuring value is scaled to
            // fit within maxval.
            chartData += simpleEncoding.charAt(
            Math.round((simpleEncoding.length - 1) * currentValue / maxValue));
        } else {
            // Invalid values will be ignored.
            chartData += '_';
        }
    }
    return chartData;
};
```

我们将在下一节“使用动态数据”中看到这个函数的具体使用案例。

### 4．扩展编码格式

扩展编码格式与简单的编码格式相似，但使用两个字符来表示一个值，可表示的数值范围为 0～4 095。下表给出了一个可能值的简单列表：

| | |
|---|---|
| **AA = 0, AB = 1, ...AZ = 25** | **90 = 3956, 91 = 3957, ... 99 = 3965** |
| **Aa = 26, Ab = 27, ... Az = 51** | **9- = 3966, 9. = 3967** |
| **A0 = 52, A1 = 53, ... A9 = 61** | **-A = 3968, -B = 3969, ... -Z = 3993** |
| **A- = 62, A. = 63** | **-a = 3994, -b = 3995, ... -z = 4019** |

| | |
|---|---|
| **BA = 64, BB = 65, ... BZ = 89** | **-0 = 4020, -1 = 4021, ... -9 = 4029** |
| **Ba = 90, Bb = 91, ... Bz = 115** | **-- = 4030, -. = 4031** |
| **B0 = 116, B1 = 117, ... B9 = 125** | **.A = 4032, .B = 4033, ... .Z = 4057** |
| **B- = 126, B. = 127** | **.a = 4058, .b = 4059, ... .z = 4083** |
| **9A = 3904, 9B = 3905, ... 9Z = 3929** | **.0 = 4084, .1 = 4085, ... .9 = 4093** |
| **9a = 3930, 9b = 3931, ... 9z = 3955** | **.- = 4094, .. = 4095** |

扩展编码格式的语法如下：

```
chd:e<series_1>,<series_2>,<series_n>,...
```

基本文本格式的数据：

```
chd=t:90,1000,2700,3500|3968,-1,1100,250
```

用扩展编码格式表示为

```
chd=e:BaPoqM2s,-A__RMD6
```

请注意值之间没有分隔符，且逗号分隔每个数据序列。下面的函数将一个 JavaScript 的数字数组转换成扩展字符串编码：

```
var extendedEncode = function (valueArray, maxVal) {
    var extendedEncoding =
        'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-.',
        extendedEncodingLen = extendedEncoding.length,
        exLenSquared = extendedEncodingLen * extendedEncodingLen,
        chartData = '';
    for (var i = 0, len = valueArray.length; i < len; i++) {
        var numericVal = valueArray[i];
        // Scale the value to fit within maxVal.
        var scaledVal = Math.floor(exLenSquared * numericVal / maxVal);
        if (scaledVal > exLenSquared - 1) {
            chartData += "..";
        } else if (scaledVal < 0) {
            // Negative values will be ignored.
            chartData += '__';
        } else {
            // Calculate first and second characters and add them to the output.
            var quotient = Math.floor(scaledVal / extendedEncodingLen);
            var remainder = scaledVal - extendedEncodingLen * quotient;
            chartData += extendedEncoding.charAt(quotient) +
                extendedEncoding.charAt(remainder);
        }
    }
    return chartData;
};
```

我们将在下一节“使用动态数据”的一个例子中探讨如何使用此函数。

## 8.3.2 使用动态数据

要想在图像图表使用动态数据，你必须从数据中自动生成图表请求 URL。你可以用以下两种方式来实现：

- 在浏览器使用 JavaScript
- 在服务器上使用如 PHP 的语言（这里不做说明）

下例演示了如何用 JavaScript 创建和使用一个图像图表的 URL。它给面包店销售图表创建了一些随机数据，我们当然也可以从服务器中读取数据。前面定义的帮助函数 extendedEncode() 可以将数据转成正确的格式。这里我们也可以使用 simpleEncode()，但要记住降低的分辨率。每次刷新页面时，都会生成一个新的随机图表：

```
<html>

    <head>
        <script type="text/javascript">
            var extendedEncode = function(valueArray, maxVal) {
                // CODE REMOVED FOR CONCISENESS
            };

            // Fill two arrays representing the two data sets with random values.
            var dataSet1 = [];
            var dataSet2 = [];
            var maxVal = 100;
            for (var i = 0; i < 3; i++) {
                dataSet1.push(Math.random() * maxVal);
                dataSet2.push(Math.random() * maxVal);
            }
            // Create the URL using the random data sets.
            window.onload = function() {
                var URL =  'https://chart.googleapis.com/chart?' +
                           'cht=bvg& +
                           'chd=e:' +
                           extendedEncode(dataSet1, maxVal) + ',' +
                           extendedEncode(dataSet2, maxVal) +
                           '&chs=500x300' +
                           '&chxt=x,y' +
                           '&chco=4D89F9,C6D9FD' +
                           '&chdl=Cookies|Cakes' +
                           '&chbh=30,10,20' +
                           '&chl=Monday|Tuesday|Wednesday';

                // Locate the image element in the DOM and set its src attribute.
                var image = document.getElementById('chart');
                image.setAttribute('src', URL);
```

```
        }
    </script>
</head>

<body>
    <!-- The image source will be changed on each page refresh. -->
    <img id="chart">
    </div>
</body>

</html>
```

在下一个例子中，我们将创造一个以 1 秒钟为间隔的随机谷歌仪表（O-Meter）图表。随机值的范围为从 0～100，100 减去此随机数得到的值是箭头标签所用的值。请尝试修改代码，来显示两个或更多的箭头（提示：谷歌仪表图表为数据集中的每个值绘制一个箭头）。

```
<html>

    <head>
        <script type="text/javascript">
            setInterval(function() {
                // Create a random value between 0-100
                var value = Math.floor(Math.random() * 100);
                // Create the URL with random value for the data
                // and 100 - value specified as the label for the arrow.
                var URL = 'https://chart.googleapis.com/chart?' +
                          'cht=gom&' +            // Specify Google-O-Meter chart.
                          'chtt=Rate-a-Coder&' +  // The chart title.
                          'chts=000000,18&' +     // Title size and color.
                          'chs=500x250&' +        // Chart size.
                          // Show both x-axis (arrow) and y-axis (values) labels.
                          'chxt=x,y&' +
                           // Label for arrow (data set 0),
                           // and labels for values (data set 1).
                          'chxl=0:|' + (100 - value) + '|1:|Good|Bad|Ugly&' +
                          // Color and size of label text.
                          'chxs=0,000000,14,0,t|1,000000,14,0,t&' +
                          // Color range of chart, red-yellow-green.
                          'chco=00FF00,FFFF00,FF0000&' +
                          // Finally, set the actual value for the arrow.
                          'chd=t:' + value;

                // Locate the image element in the DOM, and set its src attribute.
                var image = document.getElementById('chart');
                image.setAttribute('src', URL);
            }, 1000);
        </script>
    </head>

    <body>
        <!-- The image source will be changed once a second. -->
        <img id="chart">
        </div>
```

```
    </body>
</html>
```

以下代码并不创建什么图表。它根据输入的文本，创建快速响应（QR）代码（如图 8-7 所示）。


图 8-7　QR 码可以存储大约 4 KB 的信息

**提示**

QR 码是一类二维条码，随着配备相机和条形码阅读器的移动设备的出现变得流行。QR 码可以存储各种信息（最多可到 4 296 个字符），如网站的网址、联系方式和地理位置，提供了一个将信息输入手机的快捷方式，替代耗时的打字和与其他设备的交互方式。例如，手机应用程序的网站往往显示一个 QR 码，你可以用手机扫描来完全自动化软件的安装过程。QR 码也可以打印在名片上，使得接受名片的人能够通过扫描名片将详细联系方式输入手机。

```
<html>
    <head>

        <script type="text/javascript">

            window.onload = function() {
```

```
            // Generate a new barcode when the submit button is clicked.
            document.getElementById('submit').onclick = function() {

                // Get the text from the input.
                var text = document.getElementById('text-input').value;
                // Create the URL for QR bar codes.
                var URL = "https://chart.googleapis.com/chart?" +
                    "chs=256x256&" +                // Size.
                    "cht=qr&" +                     // Chart type.
                    "chl=" + escape(text) + '&' +   // The text.
                    "choe=UTF-8&";                  // Encoding.

                // Locate the image element in the DOM, set its src attribute.
                document.getElementById('chart').setAttribute('src', URL);
            }
        }
    </script>

</head>

<body>
    <!-- The image source will be changed when submit is pressed -->
    <img id="chart" width = "256" height="256"/>
    <hr/>
    <input id="text-input" type="text" size="48"
        style="font-size:18px" value = "Enter text:"/>
    <input id = "submit" type="button" value = "Create Barcode!"/>
    <hr/>
    </div>
</body>

</html>
```

### 8.3.3 总结

前面概述的图像图表显示了如何用一个格式正确的 URL 创建基本图表，以及如何使用 JavaScript 给动态数据创建图表。图表 API 为你提供了几乎无限的选择和图表组合，谷歌给每天少于 25 万次请求的用户提供“无条件”的 API 使用，用户有大量的实验空间。

下一步，我们来看看交互式图表（又名谷歌可视化 API），这是一个更面向程序员的谷歌图表服务接口。

## 8.4 交互式图表

与图像图表这种只能显示常规图像的图表不同，交互式图表由动态图形组成，这些图形由 DHTML、Flash、Canvas、SVG 和 VML 等各种浏览器功能绘制。典型的交互方式包括滚动、缩放、排序、工具提示和悬停特效。

采用的画图方法对开发者大多是透明的，因为一个好的可视化将根据目标浏览器使用合适的渲染方法。这可以节省很多时间，使开发人员可以将精力集中在图表的功能和美学上，而不是画图的细枝末节。

互动式图表需要使用外部 API，和其他的外部库一样，它应该放在网页的<head>部分：

```
<script type="text/javascript" src="https://www.google.com/jsapi"></script>
```

以下是使用可视化 API 绘制图表需要的步骤：

1．加载常规的谷歌 AJAX API。

2．请求适当的可视化 API。

3．当可视化 API 加载完毕，准备数据，最后将图表画进页面上的一个元素中。

以下代码绘制了此章使用过的面包店销售数据：

```
<html>

    <head>
        <!-- Load the general Google AJAX API -->
        <script type="text/javascript" src="https://www.google.com/jsapi">
        </script>
        <script type="text/javascript">
            // Load the visualization API, using the 'corechart' package within it.
            google.load("visualization", "1", {
                packages: ["corechart"]
            });
            // Define a function to draw the chart.
            var drawChart = function() {
                // Create a data table (initially empty).
                var data = new google.visualization.DataTable();
                // Define the columns in the table.
                data.addColumn('string', 'Day');
                data.addColumn('number', 'Cookies');
                data.addColumn('number', 'Cakes');
                // Specify the number of rows in the table.
                data.addRows(3);
                // Now add the data into each cell of the table.

                // Row 0
                data.setValue(0, 0, 'Monday');
                data.setValue(0, 1, 90);
                data.setValue(0, 2, 75);
                // Row 1
                data.setValue(1, 0, 'Tuesday');
                data.setValue(1, 1, 40);
                data.setValue(1, 2, 65);
                // Row 2
                data.setValue(2, 0, 'Wednesday');
```

```
                data.setValue(2, 1, 60);
                data.setValue(2, 2, 35);

                // Find an element in the page to draw the chart into.
                chartElement = document.getElementById('chart');
                // Create a chart object.
                var chart = new google.visualization.ColumnChart(chartElement);
                // Draw it!
                chart.draw(data, {
                    width: 500,
                    height: 300,
                    title: 'Bakery Sales',
                    vAxis: {
                        minValue: 0,
                        maxValue: 100
                    }
                });
            }

            // Wait for the API loaded event to happen, then draw the chart.
            google.setOnLoadCallback(drawChart);
        </script>
    </head>

    <body>
        <!-- This is the element into which the chart will be drawn. -->
        <div id="chart">
        </div>
    </body>

</html>
```

乍一看，输出看起来与我们先前创建的图像图表并没有什么特别不同。但是，如果你将鼠标悬停在图表元素上，就会看到一些默认的交互，如改变颜色和确切数值（如图 8-8 所示）。

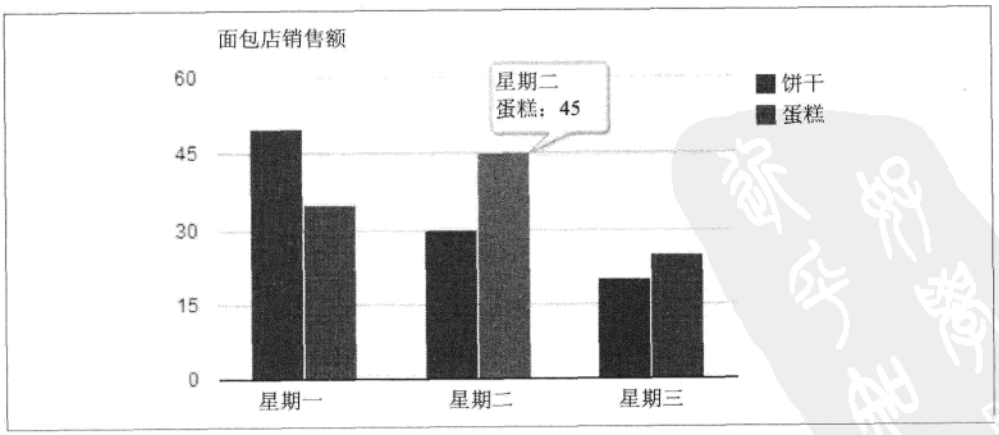



**图 8-8　一个简单柱形图上的默认互动性，包括提示信息和悬停时改变颜色**

如果你同时有 Internet Explorer 和 Firefox 浏览器，可以研究图表生成的源代码；这段源代码作为一个<iframe>被嵌入指定的图表元素。在 Internet Explorer 中图表是以 VML 创建，而在 Firefox 中以 SVG 创建，两者完全不同。

**提示**

要检验一个 HTML 页面的动态生成的部分，你不能使用浏览器默认的“查看源代码”功能。这样做只会显示从服务器下载下来的原页面，是没有图表的。你将需要使用浏览器的开发者工具，如在 Firefox 中的 Firebug，或在 Internet Explorer（从第 8 版起）里按 F12 键。但是请注意，IE 浏览器有时不会正确显示开发工具窗口。

前面的代码示例演示了使用可视化 API 的一些关键要素：

创建一个可视化的 API 数据表（var data = new google.visualization.Data Table()）

这将创建表中的数据结构，以待填入数值。

添加列到表（data.addColumn(type，label)）

这增加了一列（一个数据序列）到数据表。此列后面的所有值都应该是 type 类型。每一列的 label 的显示方式取决于所用的可视化类型。例如，柱形图用第 0 列的标签作为 x 轴标签；而饼图用第 0 列的标签命名饼图扇面（slice）。不同的可视化需要定义不同数目的列。例如，饼图和仪表需要定义两个列（如图 8-9 所示）：

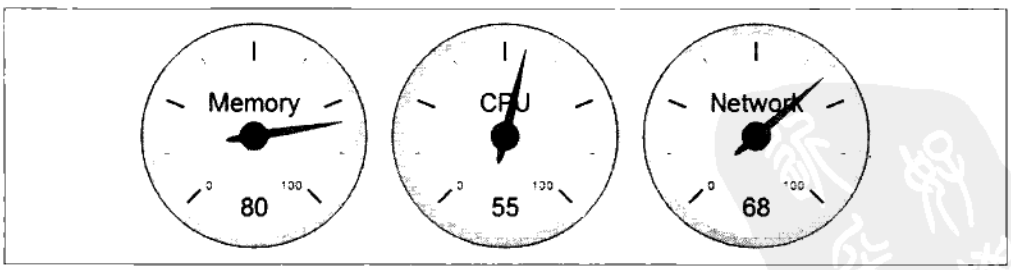



图 8-9　用仪表来显示 3 个数据值

- 一列文本用来做饼图的名称或仪表的名称
- 一列数值来确定饼图扇面的大小或仪表指针所指向的值的大小

使用不同的可视化时，要查看在线文档，以确定列的数量及类型的详细信息。

在表中添加行（data.addRows(numRows)）

这将在表中添加 numRows 个空行。或者，它可以接受一个数据数组来直接填充到行中。在前面的代码示例中，可以像这样填入数据：

```
data.addRows( [['Monday', 90, 75],
               ['Tuesday', 40, 65],
               ['Wednesday', 60, 35]]);
```

设定单元格的数值（data.setValue(row，column，value)）

这是给表中一个特定的单元格设定数值。如果值不是列定义的类型，将会产生错误。

**提示**

将数据添加到数据表还有其他方法，包括使用高效的对象实体方法，对非常大的表格这种方法更快。

可视化 API 也可以从可视化数据源来获取数据。一个数据源提供一个网址，使用 GET 请求可以从其获得正确格式的数据。通常，这一数据来自数据库或是文件。你用谷歌可视化 API Wire 协议获得数据源，而服务器软件也需要使用此协议。谷歌为不同服务器语言使用此协议提供开发库，包括谷歌可视化 API 查询语言的解析器。

有关添加数据以及使用数据源的信息请查询在线说明书。

用以下代码创建一个表（var chart = new google.visualization.ColumnChart (char tElement)）

以上代码将创建一个特定类型的图表对象（例子中是柱形的），等待被绘制。chartElement 为页面元素（通常是一个 ），用于设定绘制该图表之处。

用以下代码绘制一个表（chart.draw(data, options)）

当图表对象已被创建时，以上代码将实际绘制出设定好的图表元素。options 参数是用于设定包括双方共同的（如宽度和高度）以及可视化特定的选项的一个对象描述。认真阅读在线说明书中关于可视化的部分，以便找到可供选择的方案。

## 8.4.1 交互式图表事件

你可以通过使用事件来增加更多的交互性。每一个可视化都可能会触发自己的事件，而 JavaScript 可监听这些事件并做出相应处理。不同可视化所触发事件的信息请查询在线文档。表 8-2 列出了之前所用的柱形图表的可用事件。

表 8-2　　柱形图表可视化的可用事件

| 事件名称 | 描　述 | 返回值 |
| --- | --- | --- |
| error | 绘制图表发生错误时触发 | ID、message |
| onmouse over | 鼠标移动到一栏上时触发 | row、column |
| onmouse out | 鼠标移出一栏上时触发 | row、column |
| ready | 当图表已准备好交互时触发。你可以在该事件没有发生时和图表进行交互，但其行为是没有保障的 | 无 |
| select | 当一个柱或一个注解被点击时触发。当柱被点击时，将设置行值和列值。它们可以用来从数据表中选择正确的值。当注解被点击时，只有列被设置 | 无 |

### 1. 获取事件信息

使用可视化事件的一个小难点在于：有些事件会将事件信息直接传给事件监听器代码，而有些则需要调用可视化对象本身的一个方法。比如，select 事件不给事件监听器传任何信息，但调用可视化对象的 getSelection()方法可以得到那个图表元素被选中了。

在柱/列状图的情况下，我们可以监听这两种类型的事件：

```
// onmouseover events pass values back to listeners.
var eventListener = function(e) {
    // Display row and column of item clicked.
    alert(e.row + ',' + e.column);
};
google.visualization.events.addListener(chart, 'onmouseover', eventListener);

// select events do not pass values directly back to listeners.
// The visualizations getSelection() method must be called to get useful data.
var eventListener = function() {
    var sel = chart.getSelection();
    // getSelection() passes back an array of selected items. Here we just display
    // details of the first one.
```

```
    // Display row and column data of selected item.
    alert(sel[0].row + ',' + sel[0].column;
};
google.visualization.events.addListener(chart, 'select', eventListener);
```

下面的代码显示与之前一样的柱形图，但现在拥有鼠标悬停、鼠标移出、选择 3 个事件监听器。注意 getSelection()方法返回一个数组。一些可视化图表（例如表格可视化）中可能同时有多个元素被选中。但在柱形图中只能选择一个元素。

```
<html>
    <head>
        <!-- Load the general Google AJAX API -->
        <script type="text/javascript" src="https://www.google.com/jsapi">
        </script>
        <script type="text/javascript">
            // Load the visualization API, using the 'corechart' package within it.
            google.load("visualization", "1", {
                packages: ["corechart"]
            });
            var chart;
            // Define a function to draw the chart.
var drawChart = function() {
    // Create a data table (initially empty).
    var data = new google.visualization.DataTable();
    // Define the columns in the table.
    data.addColumn('string', 'Day');
    data.addColumn('number', 'Cookies');
    data.addColumn('number', 'Cakes');
    // Specify the number of rows in the table.
    data.addRows(3);
    // Now add the data into each cell of the table.

    // Row 0
    data.setValue(0, 0, 'Monday');
    data.setValue(0, 1, 90);
    data.setValue(0, 2, 75);
    // Row 1
    data.setValue(1, 0, 'Tuesday');
    data.setValue(1, 1, 40);
    data.setValue(1, 2, 65);
    // Row 2
    data.setValue(2, 0, 'Wednesday');
    data.setValue(2, 1, 60);
    data.setValue(2, 2, 35);

    // Find an element in the page to draw the chart into.
    chartElement = document.getElementById('chart');
    // Create a chart object.
    chart = new google.visualization.ColumnChart(chartElement);
    // Draw it!
    chart.draw(data, {
        width: 500,
        height: 300,
        title: 'Bakery Sales',
```

```
    vAxis: {
        minValue: 0,
        maxValue: 100
    }
});

// Add an event listener for onmouseover.
// It sets the hover-text paragraph on the page to show
// the row and column.
google.visualization.events.addListener(chart, 'onmouseover',
function(event){
    document.getElementById('hover-text').innerHTML =
        event.row + ' ' + event.column;
});

// Add an event listener for onmouseover.
// It clears the hover-text paragraph.
google.visualization.events.addListener(chart, 'onmouseout',
function(event){
    document.getElementById('hover-text').innerHTML = "";
});
            // This event listener shows various details about
            // the cell/column being clicked.
            // Columns are selected by clicking the legends.
            google.visualization.events.addListener(chart, 'select',
            function(){
                var selectData = chart.getSelection(),
                    message = '',row,column;
                for(var i=0; i<selectData.length; i++) {

                    var info = selectData[i];
                    row = info.row;
                    column = info.column;
                    // If both row and column are set,
                    // then a specific cell is selected.
                    if (row !== undefined && column !== undefined) {
                        message += 'cell[' + row + ',' + column + ']=' +
                        data.getValue(row, column) + ', ';
                    }
                    // Otherwise, just show the row...
                    else if (row !== undefined ) {
                       message += 'row=' + row + ', ';
                    }
                    // or column.
                    else if (column !== undefined ) {
                       message += 'column=' + column + ', ';
                    }
                }
                alert (message);
             });
        }
        // Wait for the API loaded event to happen, then draw the chart.
        google.setOnLoadCallback(drawChart);
    </script>
</head>
```

```
    <body>
        <!-- This is the element into which the chart will be drawn. -->
        <div id="chart"></div>
        <!-- This paragraph text will change with the hover events. -->
        <p id="hover-text"></p>
    </body>
</html>
```

# 第 9 章

# 使用 jQuery Mobile 为移动设备开发

带有上网功能的移动设备为程序员和设计师开辟了众多的开发方案。有这么多可用的移动平台，为每个手机的操作系统开发本机应用程序是不实际的。下面是一个不完整的手机操作系统清单：

- iOS
- Symbian
- Android
- BlackBerry OS
- Windows Mobile
- webOS

这些操作系统都有自己的开发环境和编程语言。例如苹果公司的 iOS 使用的是 Cocoa 开发环境和 Objective-C 编程语言，而 Android 是基于 Linux 环境使用 Java 开发。不幸的是，这些小小的移动设备，对应的是复杂的底层软件。我们即使忽略学习另一种编程语言的代价，仍面对着庞大而复杂的操作系统。

为了获得移动设备的最佳性能以及最好地利用它们的硬件设施，理想的情况下我们应使用本机的操作系统和平台编程语言进行开发。但当绝对性能不是最重要的，而开发时间有限又需要支持多平台时，就可以考虑另一种方案。使用普通的 Web 开

发工具的 JavaScript、HTML 和 CSS，能以较小的代价开发出实现大部分本机应用程序的外观和感觉。但重要的是要了解这种开发方法的局限性：JavaScript 并不是最快的编程语言，在低端移动设备的环境下，它可能会更加缓慢。即使像 Canvas 这样的新功能，对最高端以外的其他移动设备也可能存在性能问题。不用自身环境来开发一个快速移动的移动街机游戏当然是困难的。但因为不断有更新、更强大的设备发布，我们可以期待移动 JavaScript 性能会不断提高。

在本章中，我们将集中于开发一个简单的适用于移动设备的游戏程序，TilePic。它将使用新的 jQuery Mobile 库来提供更加接近本地应用的体验。

## 9.1 jQuery Mobile

随着 jQuery 发展成为最受欢迎的 JavaScript 库，使用它开发移动程序已是大势所趋。jQuery Mobile 是在 jQuery 框架上搭建的，它为所有流行的移动设备提供了统一的用户界面。压缩后的大小为 12KB，带宽要求适度。在编写本书时，该库已更新至 1.0 Alpha 3 版本。其支持平台名单就像一个手机操作系统世界里的名人录：

Apple iOS (3.1～4.2)

在 iPhone、iPod Touch 以及 iPad 产品上测试

Android (1.6～2.3)

所有设备在 HTC Incredible、Motorola Droid、谷歌 G1 以及 Nook Color 上测试

BlackBerry 6

在 Torch 以及 Style 上测试

Palm webOS (1.4)

在 Pre 和 Pixi 上测试

Opera Mobile (10.1)

Android

Opera Mini (5.02)

iOS 以及 Android

Firefox Mobile (beta)

Android

即将发布的 beta 版本（在编写本书时），计划支持的平台还有 BlackBerry 5、Nokia/Symbian 以及 Windows Phone 7。

Web 开发人员需要支持多种浏览器，如 Internet Explorer、Firefox 以及 Safari。在移动设备上，由于众多的平台和设备自己的浏览器，事情更加混乱。jQuery Mobile 的目标是在每个平台的自身浏览器上获得完全支持（CSS 与 JavaScript）或接近完全的支持。在其他支持较差的浏览器中会自动降低到旧版本的 HTML 和 CSS。

jQuery 使页面元素的查询和操纵更容易，从而帮助你实现 Web 应用程序的预期功能。你必须从零开始创造额外的用户界面元素，或者通过第三方插件或像 jQuery UI 这样的扩展库。而 jQuery Mobile 有更高层的途径：它将具有语义的简洁 HTML 转化为针对手机优化的丰富浏览体验。实际上，它是基于 jQuery 建立的一个移动 UI 库，大量使用 HTML5 data-属性来改变页面元素的行为和外观。比如，下面简单的代码创建了图 9-1 中的按钮：

```
<a href="#" data-role="button" data-icon="delete">Delete</a>
```


图 9-1　jQuery Mobile 按钮

HTML5 data-属性允许你在 DOM 元素上附加任意数据，通常在 jQuery 中这样使用：

```
value = $('#myelement).attr('data-mydata');   // value = contents of data-mydata.
```

在 DOM 中，可以像这样利用数据-属性指定元素：

```
<div id='myelement' data-mydata = '99' ></div>
```

data-属性越来越流行，这也增加了命名空间冲突的可能性，即相同的 data-属性名被用于不同目的的情况（比如，在你自己的代码中和在外部函数库中）。一个简单的解决方法是当使用 data-属性时包含一个唯一识别符。比如，data-myuniqueid-icon 就不会与 jQuery Mobile 中 data-icon 的使用发生冲突。

只有当你用<!DOCTYPE html>将文档类型正确设置为 HTML5 时，才会用 WC3 验证器验证HTML5 data-属性。除了面向移动设备的用户界面元素之外，jQuery Mobile 还提供了下列移动功能：

- 移动设备式的页面转换

- 轻拍、滑动和朝向事件
- 辅助功能特性
- 适应设备朝向的布局
- 主题框架
- Ajax 页面载入和历史管理

## 9.2 TilePic：移动友好的网络应用程序

使用 jQuery Mobile，我们将创建一个移动友好的应用程序——一种称为 TilePic 的游戏（如图 9-2 所示）。TilePic 是带一些额外选项和特性的一个图片滑动拼图游戏。它是能在普通移动设备上运行的图形化 Web 应用的一个好例子。如果仅面向高端设备，我们可以更有野心些，但这里我们要做的是创造可以在更多设备上运行的应用程序。

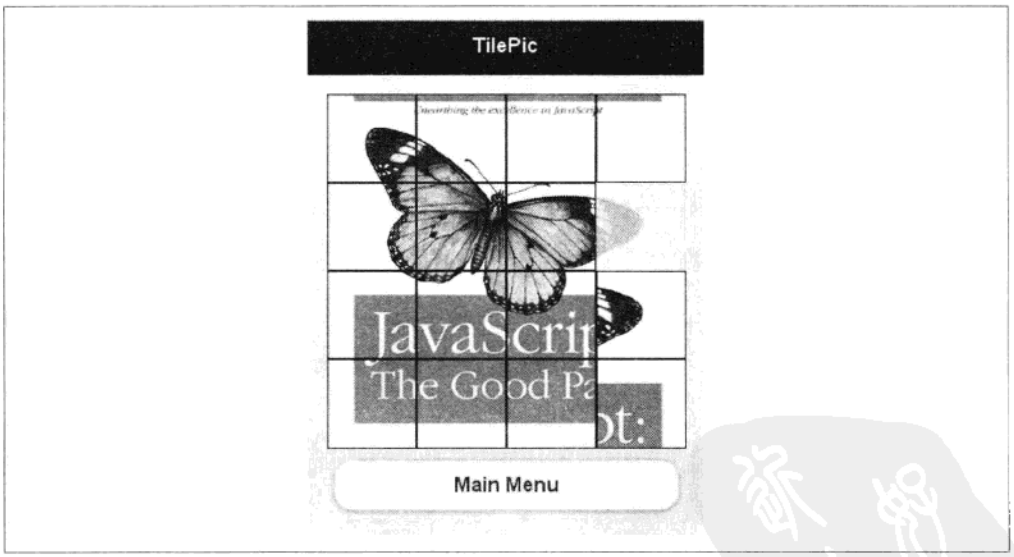



图 9-2 TilePic，这是一个简单的移动友好型的滑动益智游戏

### 9.2.1 TilePic 游戏概述

TIlePic 的流程如下：

1．给用户呈现主菜单页面（如图 9-3 所示）。

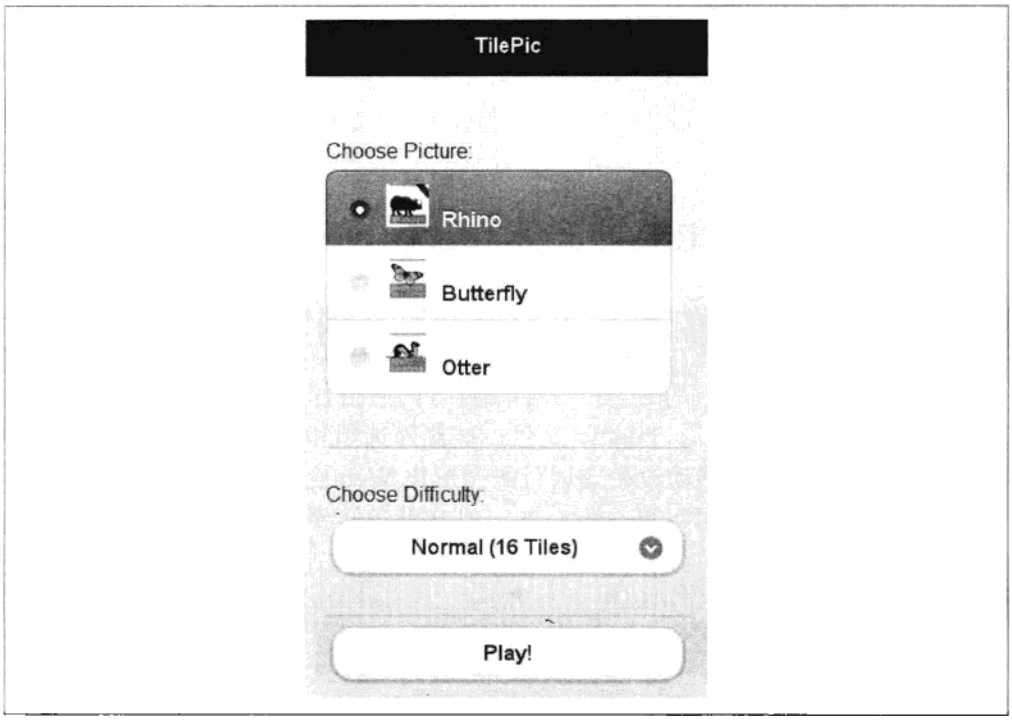



图 9-3　TilePic 主菜单

2．用户选择 3 个图象中的任意一个。

3．用户选择将图像分割成碎片的数量：9、16 或 25 小块。

4．用户点击“开始”键开始游戏，用户选择的图片被分割成相应数量的小块，并散乱地分布在屏幕上。为降低游戏难度，完整的图像以模糊水印的形式在这些块下面呈现。

5．用户通过移动小碎片，试图拼接出完整的图像。用户可以在任何时候返回主菜单来选择另一个图像或难度等级。

**提示**

这个应用程序将在合适的时候自动移动一整行小块到正确位置。这一特点使游戏更轻松，用户只需要点击一行中的最后一块，而不需要再移动每一个小拼块来把一整行移动到正确位置。

当所有的小拼块都被摆放到正确位置时，游戏会恭喜用户并显示完整的没有分割线的图像。用户可以选择主菜单按钮回到原始屏幕（如图 9-4 所示）。

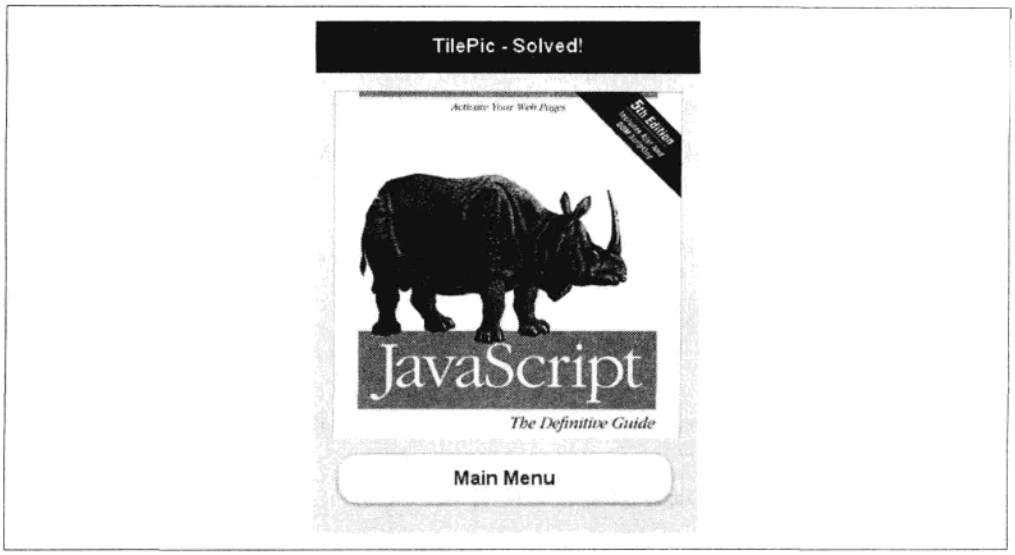



图 9-4　拼图完成！

## 9.2.2　TilePic 游戏代码

整个 TilePic 应用程序被包裹在一个匿名函数中，以保证全部变量和函数的封装。这就确保了应用程序中的内容不会在全局范围内出现，也减小了与外部库和代码发生冲突的机会（参看“TilePic 页面布局”一节）。

### 1．应用级变量

我们定义了若干应用级变量：

```
var tileSize,    // Tile size in pixels.
    numTiles,    // Number of tiles, e.g. 4 = 4 by 4 grid.
    tilesArray, // An array of tile objects.
    emptyGx,     // X position of empty tile space.
    emptyGy,     // Y position of empty tile space.
    imageUrl;    // Url of image to tile.
```

### 2．拼块对象

tileObj 对象封装了每个小拼块所需的所有数据和功能。它包括了对小拼块（$element）对应 DOM 的一个引用，和当前块的网格位置坐标（gx 和 gy）。小拼块的原始正确位置（solvedGx 与 solvedGy）已经被预先保存起来，我们通过比较小拼块的当前位置和原始位置来判断它是否被摆放正确。我们使用 move()方法移动小拼块（带或不带动画）到网格中的新位置。我们用 jQuery animate()方法来加上动

画效果，它接受小拼块的新坐标作为块元素的目标 left 和 top 属性。

checkSolved()方法执行一个简单的比较操作，来判断小拼块的当前位置是否是其原始位置，是则显示“已解决”。我们使用 jQuery data()方法在块 DOM 元素中保存对 tileObj 的引用。这使得我们在响应其 DOM 元素的事件时，可以方便地访问块对象 tileObj。

```
// tileObj represents a single tile in the puzzle.
// gx and gy are the grid position of the tile.
var tileObj = function (gx, gy) {
    // solvedGx and solvedGy are the grid coordinates
    // of the tile in its 'solved' position.
    var solvedGx = gx,
        solvedGy = gy,
        // Left and top represent the equivalent css pixel positions.
        left = gx * tileSize,
        top = gy * tileSize,
        $tile = $("<div class='tile'></div>"),

        that = {
            $element: $tile,
            gx: gx,
            gy: gy,

            // The move() method makes a tile move to a new grid position.
            // The use of animation is optional.
            move: function (ngx, ngy, animate) {
                that.gx = ngx;
                that.gy = ngy;
                tilesArray[ngy][ngx] = that;
                if (animate) {
                    $tile.animate({
                        left: ngx * tileSize,
                        top: ngy * tileSize
                    }, 250);
                } else {
                    $tile.css({
                        left: ngx * tileSize,
                        top: ngy * tileSize
                    });
                }
            },
            // The checkSolved() method returns true if the tile

        // is in the correct 'solved' position.
        checkSolved: function () {
            if (that.gx !== solvedGx || that.gy !== solvedGy) {
                return false;
            }
            return true;
        }
    };
// Set up the tile element's css properties.
$tile.css({
```

```
            left: gx * tileSize + 'px',
            top: gy * tileSize + 'px',
            width: tileSize - 2 + 'px',
            height: tileSize - 2 + 'px',
            backgroundPosition: -left + 'px ' + -top + 'px',
            backgroundImage: 'url(' + imageUrl + ')'
        });
        // Store a reference to the tileObj instance
        // in the jQuery DOM tile element.
        $tile.data('tileObj', that);
        // Return a reference to the tile object.
        return that;
    };
```

### 3．判断拼图是否完成

CheckSolved()函数遍历所有的小拼块，调用它们每一个的 checkSolved()方法。如果有小拼块未被摆放到正确位置，则整个拼图游戏仍未完成。每当用户移动一个小拼块时，此函数都会被调用。

```
// The checkSolved() function iterates through all the tile objects
// and checks if all the tiles in the puzzle are solved.
var checkSolved = function () {
    var gy, gx;
    for (gy = 0; gy < numTiles; gy++) {
        for (gx = 0; gx < numTiles; gx++) {
            if (!tilesArray[gy][gx].checkSolved()) {
                return false;
            }
        }
    }
    return true;
};
```

### 4．小拼块的移动

当用户点击小拼块时，应用程序需要决定几个因素，包括：被点击的小拼块与空白拼块的距离，以及小拼块应该向什么方向运动。可能出现的情况如下：

- 被点击的拼块就在空白拼块的上、下、左或者右边，在此情况下，小拼块应该被移动到空白拼块。
- 被点击的拼块不直接和空白拼块相邻，但在同一行或列。在这种情况下，这一行或列从被点击拼块到空白拼块的所有拼块都向空白拼块一起移动。
- 前两条件都不满足，在这种情况下被点击拼块不能移动。

稍加思索，就可以拿出一个适用于所有情况的解决方案，下面以图 9-5 为例说明如何处理不同类型的拼块移动。

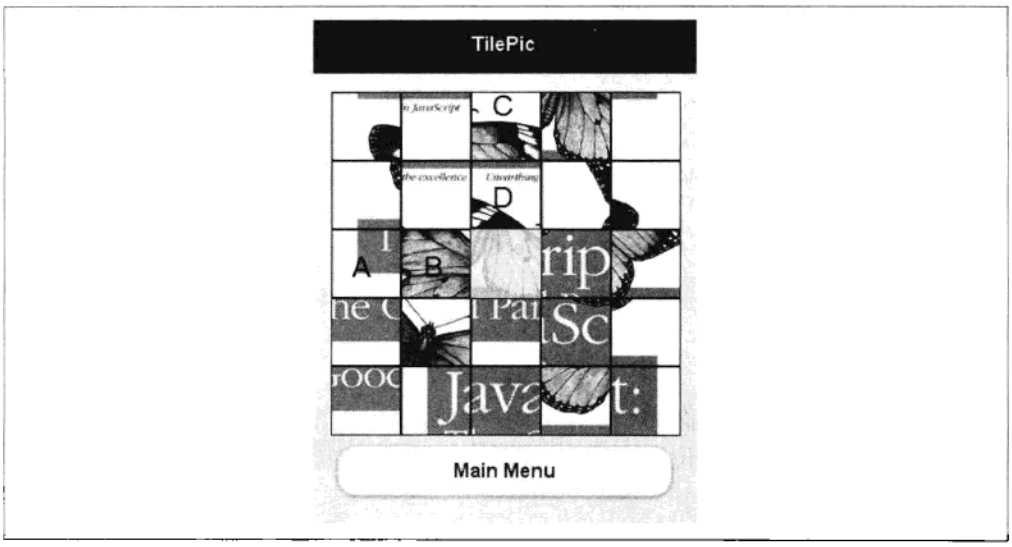



图 9-5　拼块移动

对中间一排，假设用户点击拼块 A，我们将:

1．确定被点击拼块和空白拼块在同一行。

2．确定被点击拼块到空白拼块的方向（dir=1）。

3．设置起始位置（x）为空白拼块的位置减去 dir(x=1)。

4．得到此位置的拼块（拼块 B），并移动 dir。

5．重复第 4 步（向左移动）直到位置（x）等于被点击位置减去 dir。在这个例子中，下一个拼块为 A。

6．最后设置空白拼块的位置和被点击的位置相同。

对拼块 C 和 D 的垂直运动，在概念上是相同的，只是它的移动方向发生在垂直轴上。

```
// When a tile is clicked on, the moveTiles() function will
// move one or more tiles into the empty space. This can be done
// with or without animation.
var moveTiles = function (tile, animate) {
    var clickPos, x, y, dir, t;
    // If empty space is on same vertical level as clicked tile,
    // move tile(s) horizontally.
    if (tile.gy === emptyGy) {
        clickPos = tile.gx;
```

```
            dir = tile.gx < emptyGx ? 1 : -1;
            for (x = emptyGx - dir; x !== clickPos - dir; x -= dir) {
                t = tilesArray[tile.gy][x];
                t.move(x + dir, tile.gy, animate);
            }
            // Update position of empty tile.
            emptyGx = clickPos;
        }
        // If empty space is on same horizontal level as clicked tile,
        // move tile(s) vertically.
        if (tile.gx === emptyGx) {
            clickPos = tile.gy;
            dir = tile.gy < emptyGy ? 1 : -1;
            for (y = emptyGy - dir; y !== clickPos - dir; y -= dir) {
                t = tilesArray[y][tile.gx];
                t.move(tile.gx, y + dir, animate);
            }
            // Update position of empty tile.
            emptyGy = clickPos;
        }
    };
```

### 5. 打乱拼块

shuffle()函数在空白拼块的同一列或行随机选择一个拼块，然后对其调用moveTiles()函数。使用模运算符（%）确保所选择的小拼块不是空白块，而是在网格范围内有效的拼块。

```
    // Randomly shuffles the tiles, ensuring that the puzzle
    // is solvable. moveTiles() is called with no animation.
    var shuffle = function () {
        var randIndex = Math.floor(Math.random() * (numTiles - 1));
        if (Math.floor(Math.random() * 2)) {
            moveTiles(tilesArray[emptyGx][(emptyGy + 1 + randIndex) % numTiles], false);
        } else {
            moveTiles(tilesArray[(emptyGx + 1 + randIndex) % numTiles][emptyGy], false);
        }
    };
```

shuffle()函数只执行一次拼块的随机运动，要真正打乱拼块一定要多次调用shuffle()。

### 6. 游戏设置代码

setup()函数在每次游戏赛前，执行各种清理和设置操作，包括：

- 从图片框中删除上次游戏的拼块
- 在图片框中创建水印提示图像

- 创建新的拼块（当留出右下角的位置）
- 将空白拼块放到右下角
- 打乱新的拼块

```
// Initial setup. Clears picture frame of old tiles,
// creates new tiles, and shuffles them.
var setup = function () {
    var x, y, i;
    imageUrl = $("input[name='pic-choice']:checked").val();
    // Create a subtle watermark 'guide' image to make the puzzle
    // a little easier.
    $('#pic-guide').css({
        opacity: 0.2,
        backgroundImage: 'url(' + imageUrl + ')'
    });
    // Prepare the completed 'solved' image.
    $('#well-done-image').attr("src", imageUrl);
    // Remove all old tiles.
    $('.tile', $('#pic-frame')).remove();
    // Create new tiles.
    numTiles = $('#difficulty').val();
    tileSize = Math.ceil(280 / numTiles);
    emptyGx = emptyGy = numTiles - 1;
    tilesArray = [];
    for (y = 0; y < numTiles; y++) {
        tilesArray[y] = [];
        for (x = 0; x < numTiles; x++) {
            if (x === numTiles - 1 && y === numTiles - 1) {
                break;
            }
            var tile = tileObj(x, y);
            tilesArray[y][x] = tile;
            $('#pic-frame').append(tile.$element);
        }
    }
    // Shuffle the new tiles randomly.
    for (i = 0; i < 100; i++) {
        shuffle();
    }
};
```

## 7. TilePic 事件

在页面加载时，bindEvents()函数被调用来给页面元素绑定合适的事件。它将“tap”事件绑定到图片框，这比给每个拼块绑定“tap”事件更高效。

当用户点击一个拼块元素，该事件将向上冒泡到图片框元素。这时我们可以通过 jQuery 的 data()方法访问 tileObj 元素，然后以合适的方式调用 moveTiles()函数，最后调用 checkSolved()看是否已经完全拼好。若是，将页面重定向到显示“Well Done”

消息的页面。

bindEvents()函数还给播放按钮链接绑定点击事件，来调用 setup()函数启动一个新的游戏。

```
var bindEvents = function () {
    // Trap 'tap' events on the picture frame.
    $('#pic-frame').bind('tap',function(evt) {
        var $targ = $(evt.target);
        // Has a tile been tapped?
        if (!$targ.hasClass('tile')) return;
        // If a tile has been tapped, then move the appropriate tile(s).
        moveTiles($targ.data('tileObj'),true);
        // Check if the puzzle is solved.
        if (checkSolved()) {
            $.mobile.changePage("#well-done","pop");
        }
    });

    $('#play-button').bind('click',setup);
};
```

## 8. TilePic 页面布局

```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>TilePic - A jQuery Mobile Game</title>
    <script src="http://code.jquery.com/jquery-1.5.min.js"></script>
        <script type="text/javascript">
        $(function() {

            var tileSize,   // Tile size in pixels.
                numTiles,   // Number of tiles, e.g. 4 = 4 by 4 grid.
                tilesArray, // An array of tile objects.
                emptyGx,    // X position of empty tile space.
                emptyGy,    // Y position of empty tile space.
                imageUrl;   // Url of image to tile.

            // tileObj represents a single tile in the puzzle.
            // gx and gy are the grid position of the tile.
            var tileObj = function (gx, gy) {
               /*** CODE REMOVE FOR CONCISENESS ***/

        };

        // The checkSolved() function iterates through all the tile objects
        // and checks if all the tiles in the puzzle are solved.
        var checkSolved = function () {
            /*** CODE REMOVE FOR CONCISENESS ***/
        };

        // When a tile is clicked on, the moveTiles() function will
        // move one or more tiles into the empty space. This can be done
        // with or without animation.
```

```
        var moveTiles = function (tile, animate) {
            /*** CODE REMOVE FOR CONCISENESS ***/
        };

        // Randomly shuffles the tiles, ensuring that the puzzle
        // is solvable. moveTiles() is called with no animation.
        var shuffle = function () {
            /*** CODE REMOVE FOR CONCISENESS ***/
        };

        // Initial setup. Clears picture frame of old tiles,
        // creates new tiles, and shuffles them.
        var setup = function () {
            /*** CODE REMOVE FOR CONCISENESS ***/
        };

        var bindEvents = function () {
            /*** CODE REMOVE FOR CONCISENESS ***/
        };

        bindEvents();
        setup();

    });
</script>

<link rel="stylesheet"
   href="http://code.jquery.com/mobile/1.0a3/jquery.mobile-1.0a3.min.css" />
<script
      src="http://code.jquery.com/mobile/1.0a3/jquery.mobile-1.0a3.min.js">
    </script>

<style type="text/css">
    label img {
        margin-right:10px;
    }

    #pic-frame {
        width:280px;
        height:280px;
        position:relative;
        left:0px;
        top:0px;
    }
        #pic-guide {
            position:absolute;
            backround-repeat:no-repeat;
            width:100%;
            height:100%;
        }

        .tile {
            border:1px solid;
            position:absolute;
        }

        #well-done {
            position:relative;
```

```
        }

    </style>
</head>
<body>

    <!-- Menu page -->
    <div id="menu" data-role="page">
        <div data-role="header" data-backbtn="false">
            <h1>
                TilePic
            </h1>
        </div>
        <div data-role="content">
            <div id="pic-choice" data-role="fieldcontain">
                <fieldset data-role="controlgroup">
                    <legend>
                        Choose Picture:
                    </legend>

                    <input type="radio" name="pic-choice" id="pic-choice-1"
                         value="rhino.jpg"    checked="checked" />
                    <label for="pic-choice-1">
                        <img width="32" src="rhino.jpg" />
                        Rhino
                    </label>

                    <input type="radio" name="pic-choice" id="pic-choice-2"
                        value="butterfly.jpg" />
                    <label for="pic-choice-2">
                        <img width="32" src="butterfly.jpg" />
                        Butterfly
                    </label>

                    <input type="radio" name="pic-choice" id="pic-choice-3"
                        value="otter.jpg" />
                    <label for="pic-choice-3">
                        <img width="32" src="otter.jpg" />
                        Otter
                    </label>
                </fieldset>
            </div>
            <div data-role="fieldcontain">
                <label for="difficulty" " class="select ">Choose Difficulty:</label>
                <select name="difficulty "" id="difficulty">
                    <option value="3">
                        Easy (9 Tiles)
                    </option>
                    <option value="4" selected="1">
                        Normal (16 Tiles)
                    </option>
                    <option value="5">
                        Hard (25 Tiles)
                    </option>
                    </select>
            </div>
```

```
            <a id="play-button" href="#game" data-role="button">Play!</a>
        </div>
    </div>

    <!-- Game page -->
    <div id="game" data-role="page" data-backbtn="false">
        <div data-role="header" data-backbtn="false">
            <h1>
                TilePic
            </h1>
        </div>
        <div data-role="content">
            <div id="pic-frame">
                <div id="pic-guide">
                </div>
            </div>
            <a href="#menu" data-role="button">Main Menu</a>
        </div>
    </div>

    <!-- Well done popup page -->
    <div id="well-done" data-role="page">
        <div data-role="header" data-backbtn="false">
            <h1>
                TilePic - Solved!
            </h1>
        </div>
        <div data-role="content">
            <img id="well-done-image" width="280" height="280" />
            <a href="#menu" data-role="button">Main Menu</a>
        </div>
    </div>

</body>
</html>
```

## 9.3 PhoneGap

PhoneGap 是一个多平台的本地库包，它可以将普通 Web 应用嵌入一个本地的外壳（wrapper）应用。这使你可以将 Web 应用以本地应用的形式在多个移动平台上发布和销售。不过尽管 PhoneGap 的确很有用，但它也不是你一点击就会把网络应用程序转换为畅销的本地应用程序的解决方案。你使用 PhoneGap 的时候需要注意以下几点：

PhoneGap 不会提高 Web 应用程序的性能。一个性能较低的网络应用程序由 PhoneGap 转换成本地应用程序后，性能依然会较低。

在安装有关的 PhoneGap 函数库时，你也需要安装相应的平台 SDK。在某些情况下，这不是一件微不足道的工作，可能会给你带来一些麻烦。

你仍然需要通过相应平台的批准和支付一些费用，才能以 PhoneGap 包装的方式发布 Web 应用程序。

下一章将为你演示如何用 PhoneGap 将 TilePic 程序转换成本地应用程序。

# 第 10 章

# 用 PhoneGap 创建 Android 应用

在前面的章节，我们用 jQuery Mobile 开发了一款适合移动设备的 Web 应用。在本章，我们将要用 PhoneGap 把这一款 Web 应用转换成 Android 软件。把一个 JavaScript 应用软件转换成 Android 应用程序似乎不可思议：本地 Android 程序通常是用 Java 编写的，Java 不同于 JavaScript，它们之间也不能相互转换。那 PhoneGap 是怎么做到的呢？实际上，PhoneGap 的神奇功能来源于 Android 系统本身。Android 系统提供了一个叫做 WebView 的机制，可以使本地应用程序显示普通的 Web 内容，包括在 Web 内容中执行 JavaScript 的能力。

WebView 一个令人兴奋的特性就是它还能支持本地 Android 应用程序和网络内容之间的交互。这对开发者非常有用，Android 开发者可以在应用程序中显示 Web 内容或者与其交互，Web 开发者则可以借此利用摄像头和感应器等 Android 设备。本质上，PhoneGap 就是一个使用 WebView 的 Android 应用程序，它还提供一个 JavasScript 库来访问 Android 设备的一些设施。

其他版本的 PhoneGap 也以类似的方式工作。比如，iPhone 版本的 PhoneGap 使用 iOS 系统的 UIWebView 设施。不管底层的实现细节如何，PhoneGap JavaScript 库为这些设备的特性提供了一个一致性的接口。

本章解释如何用 Eclipse 开发环境在 Windows 系统上安装 Android 版的 PhoneGap。在写这本书的时候，PhoneGap 已经有了适用于苹果 iOS、BlackBerry、Plam webOS、Windows 7 Mobile 和 Symbian 的版本。所有版本都需要你安装合适的开发环境。

# 10.1 安装 PhoneGap

网站开发者习惯于在 HTML 页面中加一个简单的 script 标签来安装 JavaScript 库。安装 PhoneGap 和相关的应用与文件可比这费劲得多。一些必需的元素或许已经安装在你的系统，但本章假定所有必需的元素都得安装。这里是相关的步骤：

1．安装一个 Java 开发工具包（JDK）。这不同于通常安装在系统上的 Java 运行环境（JRE）。JRE 允许 Java 程序在一个系统运作，而 JDK 包含了所有实际上开发 Java 应用程序的资源（并且也包含 JRE）。请记住，PhoneGap 是一个真正的 Android Java 应用程序，因此必须在你的系统上安装 Java 开发环境。

2．给 Windows 安装 Android 软件开发工具包。这能让你安装所有想要的 Android 平台版本、工具等，包括一个虚拟设备管理器和一个 Android 设备模拟器。Android 设备模拟器使你无须有设备硬件就可以测试应用程序。

3．安装 Eclipse 集成开发环境（IDE）。对于 Java 来说，这是一个首选的 IDE，它包括一个代码编辑器、调试器、工程设施和许多其他工具，可以提高你开发 Java 应用程序的效率。

4．为 Eclipse 安装 Android 开发工具（ADT）插件程序。这利用了 Eclipse 的 Java 开发工具并将其变为一个完整的 Android IDE。

5．安装 PhoneGap 本身。这也是这些步骤中最小的下载。

## 10.1.1 安装 Java 开发工具包（JDK）

你可以在 Oracle 网站上找到最新的 Java JDK，网址是：http://www.oracle.com/technetwork/java/javase/downloads/index.htm。这个网站的网址将来有可能会改动，但是只要在谷歌上搜索“JDK 下载”就可以找到它。

点击最左边的 Java 链接，就可以下载 JDK（如图 10-1 所示）。选择适合的平台（比如 Windows），并同意本条款的协议。下载的文件名应该是一个链接（比如 jdk-6u24-windows-i586.exe），文件大约有 75MB（其中应该有 15MB 是 JRE；这样可以帮你检查你正在下载的是不是正确的文件）。一旦下载完成，只要双击就可以安装了。

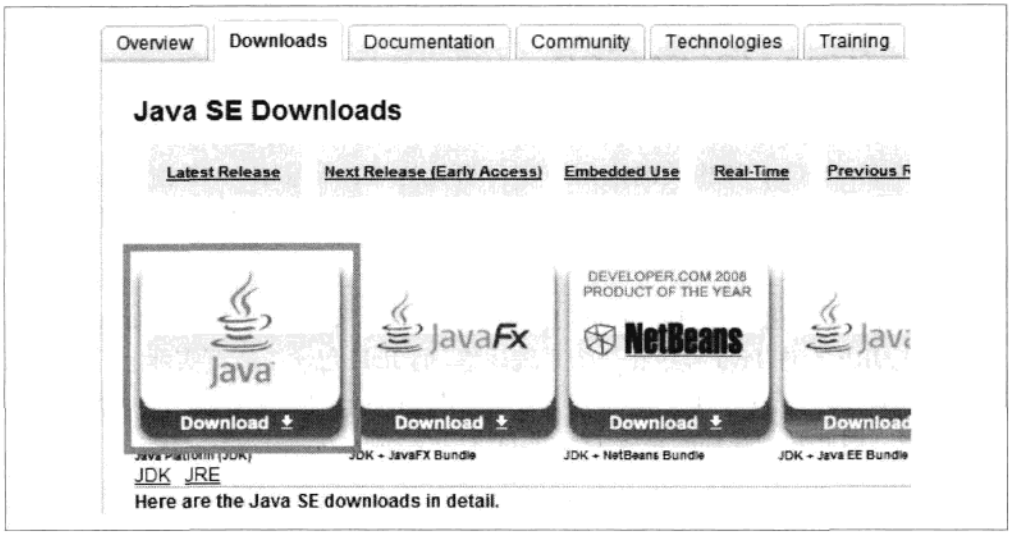



图 10-1 下载 Java JDK

## 10.1.2 安装 Android 软件开发工具包（SDK）

访问 http://developer.android.com/sdk/index.html，就可以下载 Android 软件开发工具包（Android SDK）。

对于 Windows 系统，推荐下载的文件都是 Windows 安装版本（写着 installer_ r10-windows.exe 字样的）。这将为 Android 开发安装基础软件，也叫做 AndroidSDK 管理器（ASM）。ASM 有一组自己的工具，来下载和管理 Android 平台及其他组件的更新（如图 10-2 所示）。在 ASM 上下载整套 Android 平台和组件是需要一段时间的，但是你只需要做一次，以后无论什么时候只要它正常工作，下载就可以更新。

除了常规的 Windows 应用程序窗口，ASM 还开了一个命令行窗口（也就是 DOS 对话框）。没必要惊慌，这是很正常的。ASM 实际上是一套 Android 命令行工具的 Windows 包装。实际上，所有的 Android 开发工作都可以用简单的文本编辑器和 Android 命令行工具来完成，但是 Eclipse 环境及其编辑器会使这些工作变得更容易。

如果当安装 ASM 时突然弹出一条消息说没有找到 Java JDK（虽然它肯定已经安装），就在那个对话框中点击“上一步”按钮，再试一次。这是 ASM 安装器的一个 Bug，它有可能发出一条“没有找到”的错误信息。

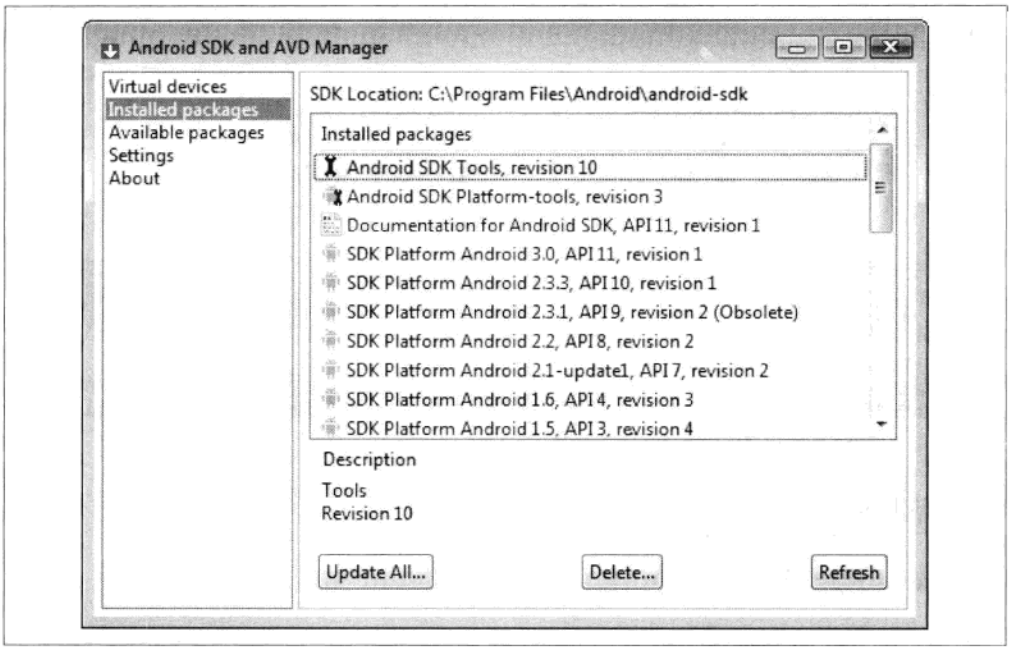



图 10-2　Windows Android SDK 管理器

## 10.1.3　安装 Eclipse

Eclipse 是一种可支持许多编程语言的很流行的 IDE 集成开发环境，也是 Android 开发所推荐的选择。加上 Eclipse Android 插件的支持，Eclipse 能够提供 Android 项目所需要的所有工具。

Eclipse 为不同编程语言和平台准备了几种版本。我们对 Android 推荐的版本感兴趣，即 Eclipse Classic（如图 10-3 所示），它是从 Java 开发版本，可以从 http://www.eclipse.org/downloads/下载下来。

作为 Java 应用程序，Eclipse 不会产生任何的 Windows 注册表项、程序组或者快捷图标。它通过其安装目录下的 eclipse.exe 文件运行。如果需要的话，你可以自行创建这个执行文件的快捷图标。

如果你热衷于全面的应用开发或者希望增强 PhoneGap 的其他功能，有大量和 Eclipse 开发相关的资源。下面这个链接就是一个很好的起点：http://www.eclipse.org/resources/。


图 10-3　Eclipse Classic

## 10.1.4　安装 Android 开发工具

如果没有插件，Eclipse 什么也做不了，几乎每一种编程语言都有一个 Eclipse 插件。插件将特定的语言功能添加到了 Eclipse，例如：语法突出显示、类浏览、调试功能等。你当然也能用常规的 Java 版 Eclipse 去做 Android 开发，但是 ADT 插件能提供最有效的 Android 开发环境。你可以在 http://developer.android.com/sdk/eclipse-adt.html 获得关于安装 ADT Eclipse 插件的信息。

ADT 是从 Eclipse 安装的，安装 ADT 步骤如下：

1．启动 Eclipse，然后选择 Help→Install New Software。

2．单击“Add”按钮

3．在出现的添加库对话框（Add Repository）中，输入 ADT Plugin 作为名称，在下面的 URL 地址中输入 https://dl-ssl.google.com/android/eclipse/，然后单击“OK”，

接下来在“Work with”项中，应该显示“ADT Plugin-https://dl-ssl.google.com/android/eclipse/”。
如果你在下载 ADT 插件时遇见问题，尝试在网址中用 http://而不是 https://。

4．选择开发工具附近的复选框并单击“下一步”，Eclipse 将会检查远程 ADT 文件，并且显示它将执行的下载内容，继续单击“下一步”，接受许可协议中的条款，然后单击“Finish”。下载将开始安装。如果出现“未签名的文件”警告，只需单击“OK”继续。

5．最后，重启 Eclipse，你已经为 Android 开发做好准备了。

到这里，所有剩下的工作就是安装 PhoneGap 本身了。

### 10.1.5 安装 PhoneGap

PhoneGap 不是实际安装的，它仅仅包含了我们需要在项目中加入的几个文件。你可以在 http://www.phonegap.com/start#android 网站上下载 Android PhoneGap。

点击 PhoneGap.zip 文件的“下载”按钮，这个文件大小大概为 4.5MB，将此文件解压缩到一个目录下，为加入项目做好准备。

## 10.2 在 Eclipse 中创建一个 PhoneGap 项目

现在一切准备就绪，是为准备测试 Android 模拟器或真正的 Android，来在 Eclipse 中创建 PhoneGap 程序的时候了。

1．启动 Eclipse，选择“文件→新建→项目”。

2．从新建项目对话框中选择 Android 项目。

3．在新的 Android 项目对话框中，输入如图 10-4 所示的详细信息。默认位置将与你的 Eclipse 工作区相同，但如果需要的话，你可以改变它的默认位置。选择一个与你的虚拟 Android 设备或真实的 Android 设备中相兼容的 Android 版本。包的名字遵循通常的 Java 包命名规则，一般来说是保证唯一的反向域名，这样就不会与其他 java 包冲突。

4．转到项目的根目录，并创建两个新的目录：/libs 和 /assets/www（请注意/assets 目录已经存在）。

5．从解压的 PhoneGap 目录下的 Android 目录中，复制 phonegap.js 文件到/assets/

www 下。

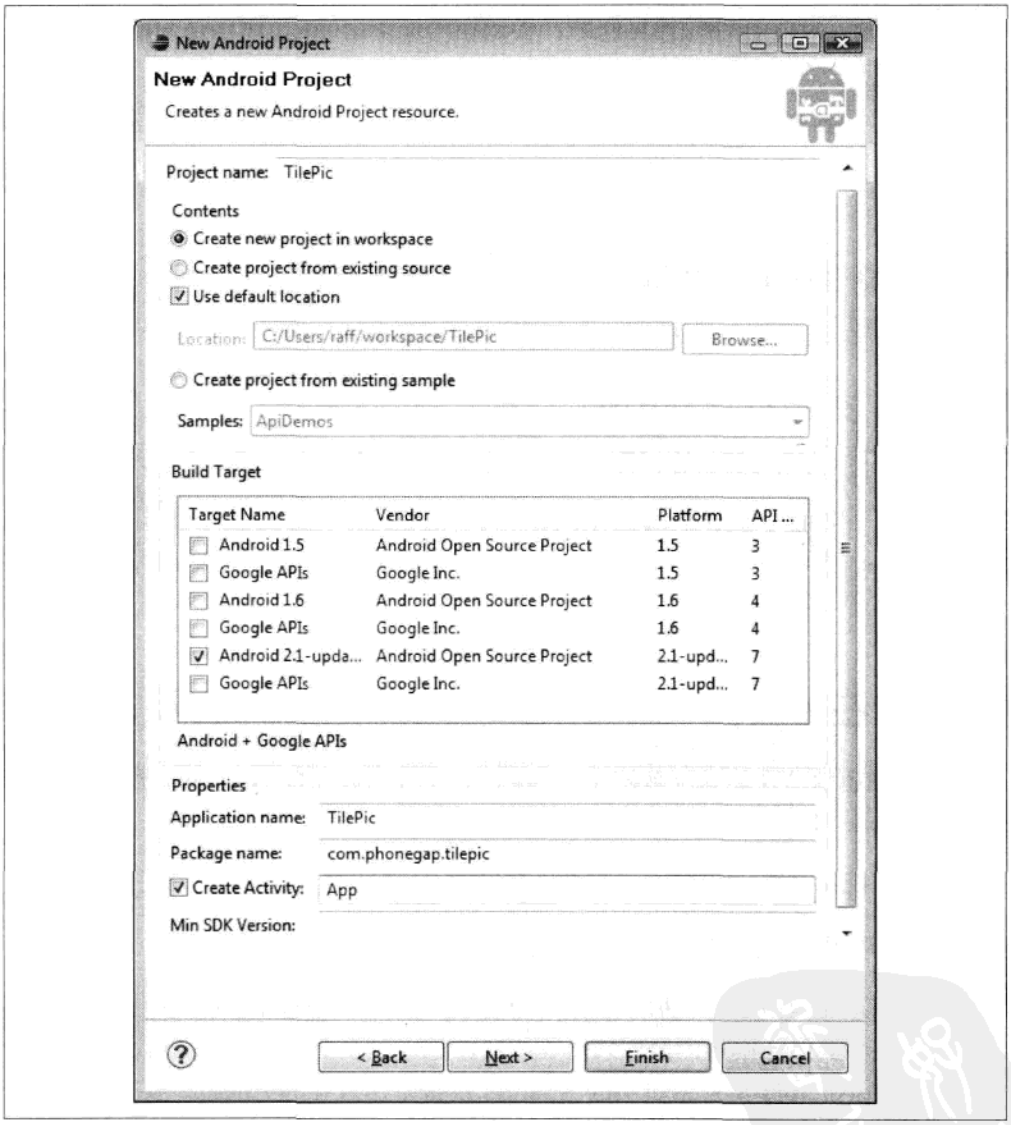



**图 10-4 新建 Android 项目对话框**

6．从解压的 PhoneGap 目录下的 Android 目录中，复制 phonegap.jar 到/libs。

当在你创建好两个目录以及复制完 phonegap.jar 和 phonegap.js 之后，在 Package Explorer 中的 Eclipse 项目结构应该如图 10-5 所示。

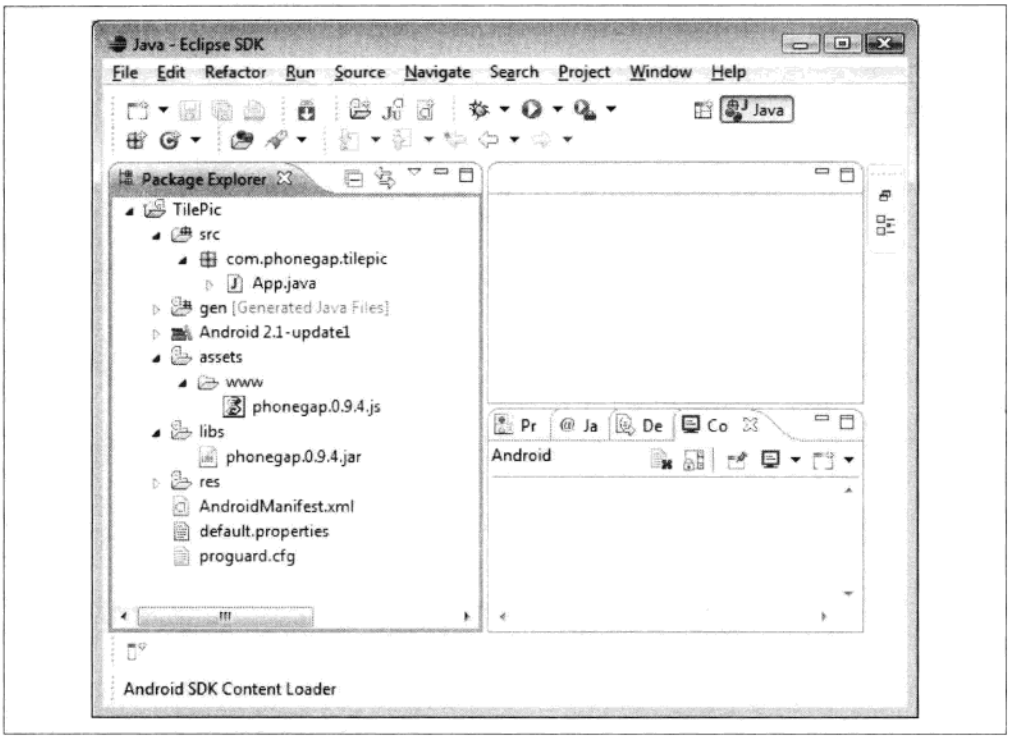



图 10-5　在 Eclipse 中的项目布局

## 10.2.1　更改 App.java 文件

双击 App.java 文件，用下面的代码替换其内容：

```
package com.phonegap.tilepic;
import android.app.Activity;
import android.os.Bundle;
import com.phonegap.*;

public class App extends DroidGap {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.loadUrl("file:///android_asset/www/index.html");
    }
}
```

这时 Eclipse 可能会出现几个错误。这是因为目前 Eclipse 不能识别/libs 目录下的 PhoneGap Java 库。为了解决这个问题，请右键单击在 Package Explorer 中的/ libs 文件夹，然后选择“生成路径→配置构建路径”。

当出现“TilePic 属性”的对话框时，单击 Libraries 选项卡，然后单击添加 JARs 按

钮。此后将出现一个 JAR 文件选择对话框，允许你选择 PhoneGap 的 JAR 文件（如图 10-6 所示）。


图 10-6　选择列入 PhoneGap 库

## 10.2.2　改变 AndroidManifest.xml 文件

接下来，用下面的代码替换 AndroidManifest.xml 文件的内容：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.phonegap.helloworld" android:versionCode="1"
    android:versionName="1.0">
    <supports-screens android:largeScreens="true"
        android:normalScreens="true" android:smallScreens="true"
        android:resizeable="true" android:anyDensity="true" />
    <uses-permission
      android:name="android.permission.CAMERA" />
    <uses-permission
      android:name="android.permission.VIBRATE" />
    <uses-permission
      android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission
      android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission
```

```
    android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS" />
  <uses-permission
    android:name="android.permission.READ_PHONE_STATE" />
  <uses-permission
    android:name="android.permission.INTERNET" />
  <uses-permission
    android:name="android.permission.RECEIVE_SMS" />
  <uses-permission
    android:name="android.permission.RECORD_AUDIO" />
  <uses-permission
    android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
  <uses-permission
    android:name="android.permission.READ_CONTACTS" />
  <uses-permission
    android:name="android.permission.WRITE_CONTACTS" />
  <uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
  <uses-permission
    android:name="android.permission.ACCESS_NETWORK_STATE" />

  <application android:icon="@drawable/icon"
      android:label="@string/app_name">
      <activity android:name=".App" android:label="@string/app_name"
          android:configChanges="orientation|keyboardHidden">
          <intent-filter>
              <action android:name="android.intent.action.MAIN" />
              <category android:name="android.intent.category.LAUNCHER" />
          </intent-filter>
      </activity>
  </application>
</manifest>
```

## 10.2.3 创建和测试一个简单的 Web 应用程序

现在，我们可以创建一个简单的 Web 应用程序，并用模拟器或者是一个真实的 Android 设备来进行测试。

右键单击 Package Explorer 中的 assets/www 文件夹，选择“新建→文件夹”，创建一个名为 index.html 的文件。右键单击 Package Explorer 中的 index.html，然后选择“打开→文本编辑器”。将下面的代码加到文件里：

```
<!DOCTYPE HTML>
<html>
<head>
    <title>TilePic Test</title>
    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
</head>
<body>
    <h1>TilePic Test</h1>
</body>
</html>
```

单击 Package Explorer 中的顶级 TilePic 文件夹，然后右键单击并选择“Run As→

Android Application”。Eclipse 将在 Emulator 或真实的 Android 设备（如果已经连接）上运行应用程序。应用程序将显示文本“TilePic 测试”。干得好，你刚刚创建了第一个 PhoneGap 的 Android 应用程序！

## 10.2.4 测试 TilePic 应用程序

要运行本机应用程序 TilePic 益智游戏，将所有 TilePic 的网页复制到 assets/www 文件夹。较早创建的 index.html 测试文件将被 TilePic index.html 文件覆盖。assets/ www 应包含的文件和文件夹如图 10-7 所示。

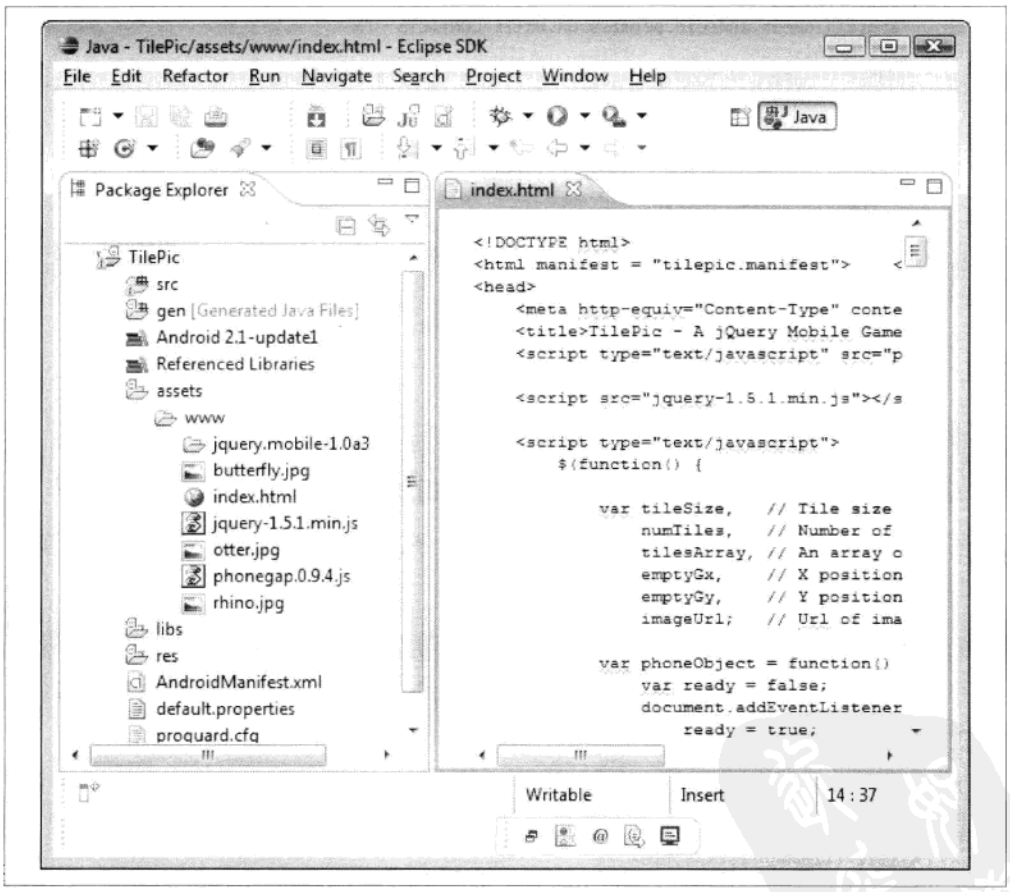



图 10-7 完整的 TilePic 项目和文件目录

在 Package Explorer 中选择顶层的 TilePic 文件夹，并右键单击后选择“Run As→ Android Application”就可以运行和测试 TilePic 了。