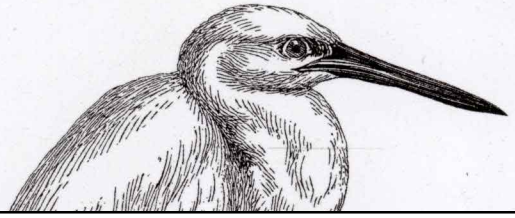
*JavaScript Cookbook*





# JavaScript 经典实例


O'REILLY®

中国电力出版社



*Shelley Powers* 著

李强 译

# JavaScript经典实例

当你在JavaScript中遇到问题的时候，没有必要再去做一些重复无谓的劳动。因为本书各节中的完整代码解决了常见的编程问题，并且给出了在任何浏览器中构建Web应用程序的技术。只需要将这些代码示例复制并粘贴到你自己的项目中就行了，可以快速完成工作，并且在此过程中学习JavaScript的很多知识。

你还将学习如何利用ECMAScript 5和HTML5中的最新功能，包括新的跨域挂件通信技术、HTML5的video和audio元素，以及绘制画布。书中一些章节介绍了如何将这些技术与JavaScript一起使用，构建高品质的应用程序界面。

- 创建交互式Web和桌面应用程序。
- 使用JavaScript对象，例如String、Array、Number和Math。
- 使用JavaScript和Scalable Vector Graphics (SVG)，以及canvas元素。
- 以不同的方式存储数据，从简单的到复杂的。
- 用新的HTML5 audio和video元素编程。
- 用Web Worker实现并发编程。
- 使用和创建jQuery插件。
- 使用ARIA和JavaScript创建完全可访问性的富Internet应用程序。

“你在寻找JavaScript解决方案甚至灵感以期走出编码迷宫，让整个事情变得容易很多？Shelley Powers通过将所有的这些优秀解决方案和技巧收集到本书中，从而帮了全世界的Web设计师一个大忙。”

——Christopher Schmitt,
CSS Cookbook作者

Shelley Powers已经从事Web技术工作和写作达15年之久，从JavaScript初次开发到现在最新的图形和设计工具发布。她最近在O'Reilly出版的图书包括语义网、Ajax、JavaScript和Web图形。

O'REILLY®
oreilly.com.cn

www.oreilly.com
O'Reilly Media, Inc.授权中国电力出版社出版



ISBN 978-7-5123-2058-1
9 787512 320581 >
定价：78.00元

# JavaScript经典实例

*Shelley Powers* 著
李强 译

O'REILLY®

*Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo*

O'Reilly Media, Inc.授权中国电力出版社出版

中国电力出版社

**图书在版编目（CIP）数据**

JavaScript经典实例/（美）鲍尔斯编著；李强译. －北京：中国电力出版社，2011.8
书名原文：JavaScript Cookbook
ISBN 978-7-5123-2058-1

I.①J… II.①鲍… ②李… III.①JAVA语言－程序设计 IV.①TP312
中国版本图书馆CIP数据核字（2011）第172070号

北京市版权局著作权合同登记
图字：01-2010-7925号



书　　名/ JavaScript经典实例
书　　号/ ISBN 978-7-5123-2058-1
责任编辑/ 刘炽
封面设计/ Karen Montgomery，张健
出版发行/ 中国电力出版社（http://www.cepp.sgcc.com.cn）
地　　址/ 北京市东城区北京站西街19号（邮政编码100005）
经　　销/ 全国新华书店
印　　刷/ 航远印刷有限公司
开　　本/ 787毫米×980毫米　16开本　33.125印张　625千字
版　　次/ 2012年3月第一版　2012年3月第一次印刷
印　　数/ 0001－3000册
定　　价/ 78.00元（册）

# 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

# 目录

# 前言

我在15年前编写了自己的第一本JavaScript图书，并且那时候必须匆忙地寻找足够的材料来充实一本图书。对于本书，我从数百种用法中选择要包含什么内容。毕竟，这些年看着JavaScript发展，我仍然对于JavaScript的用法可以变得如此深远而感到惊讶。在我看来，没有比它更加有用的编程语言或开发工具了。这是在HTML中唯一的如此广泛地应用的技术。

本书针对那些已经接触过JavaScript，并且想要尝试新技术的人，或者想要加强对JavaScript基础知识和高级功能的掌握的人。通过本书，我将介绍：

- 如何使用`String`、`Array`、`Number`和`Math`这样的JavaScript对象。
- 创建可重用的对象。
- 在文档对象模型（Document Object Model，DOM）中查询和创建新的元素。
- 使用新的Selectors API以更高效且有目标地查询。
- 将JavaScript与HTML 5新技术一起使用，例如，新的媒体元素video和audio。
- 创建可交互的应用程序。
- 管理Web页面空间。
- 以不同的方式存储数据，从简单的到复杂的。
- 使用JavaScript和Scalable Vector Graphics (SVG)以及canvas元素。
- 使用一些有趣的数据结构，例如微格式和RDFa。
- 将库打包以供其他人使用，以及在你的应用程序中使用其他的库。

- 通过Accessible Rich Internet Applications (ARIA)来确保你的应用程序的可访问性。
- 在典型桌面浏览器以外的环境中工作，例如，创建移动手机Web应用程序，用新的行为扩展Photoshop。
- 使用并创建jQuery插件。
- 开发Ajax应用程序。
- 使用浏览器的调试器来调试应用程序。
- 使用新的HTML 5拖放功能。
- 使用新的HTML 5跨文档技术来通信。
- 使用Web Workers实现并发编程。
- 使用File API直接在客户端口JavaScript中访问一个桌面文件。

本书不是关于现今的JavaScript用法的百科全书，当然还没有一本书能够覆盖所有这些内容。但是，希望你读完本书后，对于能使用JavaScript做的事情能够有更广的认识。

你一定会感到津津有味！

# 目标读者、所需背景和阅读方法

本书的读者应该对Web开发和JavaScript的用法有所了解。此外，各个小节的形式，意味着我们会关注特定的任务，而不是提供一个全面的一般性介绍。我不会介绍JavaScript主题的每一个方面，如String。相反，我们会关注与该主题相关的、较为常见的任务或实战。

书中会有很多代码，一些是代码段，另一些是完整的应用程序。各个小节也是相互引用的，因此，如果在一个小节中我所介绍的特定话题，在另一个小节中也有所涉及，我会在该小节的“参见”部分包含这些信息。为了给你提供帮助，我还为所有的小节创建了示例代码，以便你可以下载并立即使用。

## 目标浏览器

整个本书中，我会提到目标浏览器。本书中主要的示例代码，都针对最常使用的浏览器的新版本而设计，并且进行了测试。这些浏览器是：

- 在Mac和Windows上运行的Firefox 3.6x。
- 在Mac和Windows上运行的Safari 4.0x。

- 在Mac和Windows上运行的Opera 10.x。
- 在Windows上运行的Chrome 5.x。
- 在Windows上运行的Internet Explorer 8。

我没有使用Linux的机器来测试这些环境，但是，运气好的话，大多数小节在Linux环境下使用Firefox也能工作。我也没有一个System 7来测试IE9的预览版，但是，大多数应用程序应该能够工作，包括那些使用了SVG（IE9新添加的功能）的应用程序。

一些小节需要特定的环境，例如，一个移动设备或模拟器，或者浏览器的beta版（或alpha版）发布。对于那些在目标浏览器中无法工作的示例，或者需要特定环境或浏览器的示例，我做出了说明。此外，我还介绍了只在某种浏览器的alpha/beta版中实现了的新技术和API。同样，我也对浏览器支持给出了说明。

很多示例在IE6中无效。在编写本书之前，我决定不对IE6提供支持，包括任何的解决方案代码也是如此。很多主流的站点现在都不再支持太早或不太安全的浏览器，包括Amazon、Google、YouTube和Facebook。此外，IE6所需的解决方案如此知名，并且在互联网上有很好的记载，这使我觉得不必再在本书中提及。

在IE8中有效的大多数示例，也会在IE7中有效，只有一部分例外。IE7不支持常见属性上的`getAttribute/setAttribute`，例如`style`、`id`和`class`，并且根本不支持`hasAttribute`。此外，IE7不支持CSS选择器，也不支持Selectors API方法，例如`document.querySelectorAll`（将在第11章中介绍）。

IE7无效的地方，要么我在可以下载的示例代码中提供特定于IE7的解决方案，或者我在该小节中注明不支持，或者二者都做到。

## 示例代码体例

全书中有很多代码片段和功能完备的示例。大多数都是基于HTML的，但是，也有一些基于XHTML，即HTML的XML序列化。此外，大多数示例都是基于HTML 5的，尽管我也已经使用了几个其他的HTML版本，特别是在SVG的示例中：

HTML5

```
<!DOCTYPE html>
```

XHTML5

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
```

XHTML+SVG

```
<!DOCTYPE html PUBLIC
   "-//W3C//DTD XHTML 1.1 plus MathML 2.0 plus SVG 1.1//EN"
   "http://www.w3.org/2002/04/xhtml-math-svg/xhtml-math-svg.dtd">
```

还有几个基于HTML的示例的不同版本。如果该示例是X/HTML5，你不需要对`style`或`script`元素使用`type`属性。此外，在很多XHTML示例页面中，我用一个CDATA区段把代码包围起来，如下所示：

```
<script>
//<![CDATA[
Preface | xv
...
//--><!]]>
</script>
```

对XHTML中的脚本块使用一个CDATA区段的原因是，像尖括号(< >)和&这样的字符，在JavaScript中是有意义的，但是，它们作为标记也是有意义的。当一个XML解析器看到这些字符的时候，它想要将其解释为标记。为了预防这一点，CDATA区段告诉解析器，忽略该区段。

我试图在同一页面中保持所有的样式设置和脚本，从而简化示例。然而，在现实世界中，你需要尽可能地将样式表和脚本分别放到不同的文件中。这么做是为了保持HTML文件的整洁，同时，使得样式和脚本都易于修改。

## 阅读方法

在本书中，我们首先从基本的JavaScript功能开始介绍，因为这仍然是应用程序开发必备的基础。我还使用一些小节，介绍了较新的功能，包括使用canvas元素，尝试新的跨域挂件通信技术（PostMessage），使用新的File API和Web Workers，将代码与流行的jQuery库整合，甚至使用新的HTML video和audio元素（其中有很多乐趣）。我还介绍了JavaScript的一些新用法，例如，在移动设备和离线桌面应用程序中的使用，以及访问页面中的微格式和RDFa这样的元数据。

## 组织方式

本书介绍的是相对深奥的话题，主要覆盖了在过去几年里，我所看到的这一领域有趣的和发展的地方。还包含了对新的ECMAScript 5和HTML5创新的介绍。

然而，本书确实包含了两个相对通用的部分：第一个部分关注已有的JavaScript功能和对象，第二个部分更多地关注JavaScript在环境中的应用，例如浏览器。如果你是一个JavaScript新手，建议你先学习完前10章中的所有小节，然后再学习本书后面的章节。

以下是各章内容的简单介绍。

*第1章 使用JavaScript字符串*

介绍了一些较为常见的字符串任务，例如，连接字符串、删除空白、根据符号分解字符串，以及在字符串中查找子字符串。

*第2章 使用正则表达式*

介绍了正则表达式的使用，以及使用JavaScript RegExp对象。各个小节包括基本操作，例如，如何交换单词，用指定的实体替换HTML标签，验证一个社会安全号码（以及其他模式对象），以及全局地替换值。

*第3章 日期、事件和定时器*

介绍了如何访问日期和事件，以及如何格式化日期字符串、记录流逝的事件、获取未来的日期，以及使用新的和旧的ISO 8610 JavaScript功能。本章还介绍了JavaScript定时器，以及使用定时器和函数闭包。

*第4章 使用Number和Math*

包括基本的数字功能，例如，保持一个递增的计数器，十六进制数和十进制数之间的转换，产生随机颜色，把表格中的字符串转换为数字，以及弧度和角度之间的转换（在使用canvas和SVG的时候很重要）。

*第5章 使用数组和循环*

数组是本章的主要内容，介绍了如何使用数组来创建FIFO队列和LIFO栈，以及如何排序一个数组，使用多维数组，遍历数组，使用新的ECMAScript 5数组功能来创建过滤的数组，以及验证数组内容。本章还介绍了关联数组，以及遍历数组的各种方式。

*第6章 使用JavaScript函数构建可重用性*

JavaScript函数是这一语言的核心和灵魂，并且本章关注如何创建函数、向一个函数传入值和从函数传出值，创建递归函数，以及构建一个动态函数。本章还包括如何使用Memorization和Currying，来提高应用程序效率和性能，以及如何匿名函数以包括全局。

*第7章 处理事件*

介绍了基本的事件处理任务，包括捕获事件、取消事件、访问Event对象，以及使用鼠标和键盘事件。本章还介绍了新的HTML拖放功能，以及使用Safari的Orientation Events（用于移动开发）。

*第8章 浏览器模块*

本章深入到所有浏览器以及很多其他用户代理的基本工作部件，包括创建新的窗口、更改样式表、修改图像、给Web页面添加面包屑路径，将一个动态页面加入

书签，以及在Ajax应用程序中保留返回按钮。本章还介绍了保留动态状态的新的HTML 5 History功能。

*第9章 表单元素和验证*

本章接着第2章继续介绍正则表达式，但是，重点关注表单元素和验证。本章还介绍了如何打开或关闭表单元素，隐藏或显示元素，如何修改一个选项列表，以及取消一次表单提交。

*第10章 调试和错误处理*

没有人喜欢错误，但所有人都需要处理它。本章关注应用程序中的错误处理，以及如何使用本书的目标浏览器中的不同调试工具。

*第11章 访问页面元素*

本章介绍了可以访问一个或多个文档元素的方法。介绍了访问某种类型的所有元素、访问一个特定元素，或者使用新的Selectors API以及类似CSS的语法来查找元素，还包括对命名空间指定及其何时适用的讨论。

*第12章 创建和删除元素和属性*

本章包括向Web文档创建和添加元素的方式，包括添加文本、段落、使用表格元素，移动和删除文档元素。本章还介绍了如何添加和访问元素属性，以及命名空间指定及其何时适用。

*第13章 使用Web页面空间*

Web页面是我们在其上创建的一个画布，本章介绍了如何确定Web页面的区域、页面元素的大小、它们的位置，以及如何隐藏和显示页面部分。像伸缩/手风琴以及页面重叠这样的流行的行为，以及标签页页面，都作了介绍；还有如何创建一个可折叠的边栏和一个基于悬停的弹出消息框。

*第14章 使用JavaScript、CSS和ARIA创建交互性和可访问性效果*

长时间以来，我们的动态Web页面效果对于一个显著的Web群体都是沉默的，就是那些使用屏幕阅读器的人们。本章介绍了新的Web Accessibility Initiative Accessible Rich Internet Applications (WAIARIA)属性和角色，并且介绍了它们如何让一个Web页面变得对所有访问者都活跃起来，而不只是对那些能看到的人来说是鲜活的。本章还介绍了其他非常常见的交互性效果，包括提供一个颜色闪亮来表示一次事件，使用弹出消息框，创建Live Regions，以及在验证表单的时候提供可访问性效果。

*第15章 创建富媒体和交互应用程序*

我并不具备艺术家的气质，但是，我知道如何让JavaScript使用canvas元素和SVG。在本章中，我提供了使用这些媒体的基本的步骤，以及新的WebGL 3D环境，还有新的HTML 5 video和audio元素。

*第16章 JavaScript对象*

可能这是本书中最重要的章节之一，它介绍了创建JavaScript对象的基本支持，包括如何保持数据成员私有、添加Getters/Setters、使用新的ECMAScript 5对象保护功能，链化对象方法，以及使用新的Prototype.bind。

*第17章 JavaScript库*

本书所有内容都关注创建你自己的JavaScript对象和应用程序。本章介绍了jQuery，这是最流行的JavaScript框架库之一。它介绍了常见的库任务，例如，如何把代码打包到库中，如何测试库，如何构建一个jQuery插件，以及如何将你的库与其他库（如jQuery）一起使用。

*第18章 通信*

本章主要关注Ajax任务，包括准备要发送的数据、创建一个XMLHttpRequest对象、检查错误以及处理结果。还包括了如何对一个持续更新的查询使用定时器，如何创建一个动态图像弹出效果，如何针对跨域请求使用JSON-P。本章介绍了post-Message功能，用于一个远端寄存的挂件和你的应用程序之间的通信。

*第19章 使用结构化数据*

任务包括如何处理从一个Ajax调用返回的一个XML文档，使用新的JSON对象来解析JSON，或者字符串化一个JavaScript对象。本章还介绍了如何使用页面中的微格式或RDFa。

*第20章 持久化*

本章介绍了如何创建和使用一个HTTP cookie，当然，还有如何使用页面URL来存储数据，以及如何使用HTML 5中引入的新的sessionStorage和localStorage持久化技术，并且还介绍了客户端SQL数据库。

*第21章 JavaScript创新用法*

本章简单地介绍了如今可以使用JavaScript的所有各种方式，而这些与传统的Web页面开发无关。包括讨论创建移动和桌面挂件、移动设备应用程序开发，为浏览器开发插件和扩展，以及如何将JavaScript与如此众多的应用程序一起使用，包括OpenOffice（我编写本书使用的软件）和Photoshop。我还讨论了桌面应用程序开发，包括对离线应用程序的支持，并且给出了Web Workers API和File API的示例。

# 版式说明

本书中使用如下字体惯例：

*斜体字（Italic）*

表示新的术语、URL、电子邮件地址、文件名、文件扩展名等。

等宽字体（Constant width）

表示各种计算机代码，包括命令、数组、元素、语句、选项、选择、变量、属性、键、函数、类型、类、命名空间、方法、模块、特性、参数、值、对象、事件、事件处理程序、XML标签、HTML标签、宏、文件内容以及命令的输出。

等宽粗体（**Constant width bold**）

显示需用户输入的命令行或其他文本。

*等宽斜体（Constant width italic）*

用来显示需要用户提供的值或环境确定的值来代替的文本。

---

**注意**：表示一个提示、建议或一般性提示。

---

---

**警告**：这个图标表示警告或小心。

---

本书提到的Web站点和页面能够帮助你找到有用的在线信息。通常，页面的地址(URL)和名称（标题、题目）都会提到。一些地址相对复杂，但是，你可以使用你喜欢的搜索引擎来查找页面的名称，名称通常放在引号之间，从而很容易地找到页面。如果不通过地址找到页面，这些信息也是有帮助的。它可能移到了别的地方，但是，名称可能没有变化。

# 使用代码示例

本书的目的就是为了帮助读者尽快掌握相关领域的知识。通常，可以在程序或文档中使用本书中的代码。如果涉及的代码量不是很多，一般不需要得到我们的许可。例如，读者在自己编写的程序中用到了本书的几段代码就不需要得到许可。但是，销售或分发包含O'Reilly图书示例代码的光盘，则必须得到许可。在解答他人的问题时以本书内容为例，或者引用本书的示例代码不需要得到许可。然而，要在产品文档中大量使用本书的示例代码，则必须得到许可。

虽然我们不会要求，但如果读者在引用本书代码时注明出处，我们将不胜感激。在注明出处时，通常应该包含书名、作者、出版社和ISBN。例如，“JavaScript Cookbook, by Shelley Powers. Copyright 2010 Shelley Powers, 9780596806132.”。

如果你认为自己对本书代码的使用超出了合理的或上述默认许可的范围，随时可以通过*permissions@oreilly.com*与我们联系。

# 如何联系我们

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

美国：

O'Reilly 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

本书的网页上列出了勘误、示例和其他相关信息，可以从以下页面进行访问：

*http://www.oreilly.com/catalog/9780596806132*

如果想要发表关于本书的评论或询问技术问题，请发送邮件到：

*bookquestions@oreilly.com*

关于图书、会议、资源中心和O'Reilly网络的其他信息，请查看我们的站点：

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

# 致谢

我要感谢Simon St.Laurent，他是一位资深编辑，也是我的老朋友，感谢他在这本书以及我之前的图书中提供的所有帮助和鼓励。

我还要感谢那些花时间和精力来评阅本书，帮助我把工作做得更好的人们：Elaine Nelson, Zachary Kessin、Chris Wells和Sergey Ilinsky。多谢Gez Lemon，他提供了富有思想的评论，并且帮助我完成关于ARIA的章节，还有Brad Neuberg对于第15章中的SVGWeb所提供的帮助。

我要感谢我的责任编辑，以及其他产品团队：Colleen Toporek、Adam Zaremba、Rob Romano、Kiel Van Horn和Seth Maislin。

还要感谢那些帮助创建HTML 5和ECMAScript 5等规范、API、库和浏览器的人们，还有那些致力于使得今天的JavaScript开发如同15年前一样有趣的所有其他的人。

第1章

# 使用JavaScript字符串

## 1.0 简介

JavaScript字符串是JavaScript最重要的部分，可能比任何其他的数据类型都更多地用到。尽管你可能从Web页面表单获取数字值，但是，这个值仍然是作为字符串获取的，然后，我们必须将其转换为数字值。在通过Ajax调用服务器端应用程序的时候，以及形成每个JavaScript对象的基本的序列化格式的时候，字符串也用作参数。所有JavaScript对象共享的方法之一是`toString`，它返回一个字符串，其中包含了该对象的序列化格式。

### 字符串基本数据类型

JavaScript字符串可以是一个基本数据类型或者一个对象。作为基本数据类型，它与JavaScript的其他4种基本数据类型并列：数字、布尔（真或假）、`null`（无值）和`undefined`（未知）。此外，作为基本数据类型，字符串也是JavaScript直接量：这是一个集合，包含数字（浮点数或整数），以及数组、对象和正则表达式、数字和布尔值的直接量格式。

---

**注意**：在整本书中，我们将会见到各种JavaScript对象的直接量格式。

---

字符串具有0个或多个字符。字符串是由引号括起来的。使用单引号括起来：

```
'This is a string'
```

或者用双引号括起来：

```
"This is a string"
```

使用哪种类型的引号，并没有严格规定。如果你要在文本中包含单引号，则字符串更可能要使用双引号的形式：

```
"This isn't a number."
```

如果你混合使用引号类型，以一个单引号开始，并且以一个双引号结束，那么，将会产生一个应用程序错误：

```
var badString = 'This is a bad string"; // 糟糕，出错了
```

本书中将会交叉使用两种引号类型。

## 字符串对象

字符串对象叫做String，这很合适，并且，与所有其他的JavaScript对象一样，它拥有预先构建到对象类型中的一组属性。

可以使用JavaScript的new运算符来实例化一个String对象，从而创建一个新的对象实例：

```
var city = new String("St. Louis");
```

一旦实例化了，可用的字符串属性中的任何一个，都可以通过字符串进行访问，例如，在如下的代码中，使用String对象方法toLowerCase将字符串变为小写：

```
var lcCity = city.toLowerCase(); // 新字符串现在是st. louis
```

如果你没有使用new来访问String构造函数，将会创建一个字符串直接量，而不是一个String对象。

```
var city = String("St. Louis");
```

如果需要在一个直接量上访问String对象方法，也可以做到。具体的情况是，JavaScript引擎创建了一个String对象，用它包含了字符串直接量，执行方法调用，然后，丢弃掉String对象。

与使用字符串直接量相比，使用String的时候，要依赖于具体环境。除非你计划使用String对象属性，否则应该尽可能地使用字符串直接量。然而，如果你要使用String方法，那么，将字符串创建为对象。

## 参见

Mozilla有一个不错的页面，讨论了JavaScript直接量和不同类型的概念。可以通过*https://developer.mozilla.org/en/Core_JavaScript_1.5_Guide/Literals*访问该页面。

# 1.1 连接两个或多个字符串

## 问题

想要把两个或多个字符串合并为一个。

## 解决方案

使用相加运算符(+)来连接字符串：

```
var string1 = "This is a ";
var string2 = "test";

var string3 = string1 + string2; // 使用"This is a test"创建一个新字符串
```

## 讨论

相加运算符(+)通常用来把两个数字加到一起：

```
var newValue = 1 + 3; // 结果是4
```

然而，在JavaScript中，相加运算符重载了，这意味着，它可以用于多种数据类型，包括字符串。当对字符串使用的时候，结果是连接字符串，即将表达式中后面的字符串附加到字符串结果的末尾。

可以将两个字符串相加：

```
var string3 = string1 + string2;
```

或者，可以将多个字符串相加：

```
var string1 = "This";
var string2 = "is";
var string3 = "a";
var string4 = "test";
var stringResult = string1 + " " + string2 + " " +
string3 + " " + string4; //结果是"This is a test"
```

连接字符串还有缩写形式，这就是JavaScript的缩写赋值运算符(+=)。如下的代码片段使用了这个运算符：

```
var oldValue = "apples";
oldValue += " and oranges"; // 字符串现在是"apples and oranges"
```

等于：

```
var oldValue = "apples";
oldValue = oldValue + " and oranges";
```

缩写赋值运算符用于字符串的时候，会将运算符右侧的字符串连接到左侧的字符串的末尾。

有一个内建的String方法，可以连接多个字符串，这就是concat。它接受一个或多个字符串参数，其中的每一个都连接到字符串对象的末尾：

```
var nwStrng = "".concat("This ","is ","a ","string"); // 返回"This is a string"
```

concat方法可能是从多个值产生一个字符串的一种较简单的方法，例如，从几个表单字段来生成一个字符串。然而，相加运算符是更加常用的方法。

# 1.2 连接字符串和另一种数据类型

## 问题

想要把一个字符串和另一种数据类型（例如，一个数字）连接起来。

## 解决方案

使用与连接字符串的时候完全相同的运算符，例如相加(+)和缩写赋值(+=)：

```
var numValue = 23.45;
var total = "And the total is " + numValue; // 字符串是"And the total is 23.45"
```

## 讨论

当把一个字符串和其他数据类型相加的时候，过程有所不同。在其他数据类型的情况下，例如布尔或数字，JavaScript引擎首先将其他数据类型的值转换为一个字符串，然后，执行连接：

```
//把一个布尔值加到一个字符串
var boolValue = true;
var strngValue = "The value is " + boolValue; // 结果是"The value is true"
// 把一个数字加到字符串
var numValue = 3.0;
strngValue = "The value is " + numValue; //结果是"The value is 3"
```

如果要连接一个String对象和一个字符串直接量，也会发生自动数据转换。如果你不能确定所要操作的字符串是对象还是直接量，但还是想要创建一个连接的字符串，那么，自动转换是必需的一种功能：

```
var strObject = new String("The value is ");
var strngLiteral = "a string";
var strngValue = strObject + strngLiteral; // 结果是 "The value is a string"
```

最终的字符串是一个字符串直接量，而不是一个String对象。

# 1.3 条件比较字符串

## 问题

想要比较两个字符串看看它们是否相同。

## 解决方案

在一个条件测试中，使用相等运算符(==)。

```
var strName = prompt("What's your name?", "");

if (strName == "Shelley") {
   alert("Your name is Shelley! Good for you!");
} else {
   alert("Your name isn't Shelley. Bummer.");
}
```

## 讨论

使用相等运算符（==）可以比较两个字符串。当在一个条件语句中使用的时候，如果测试结果为true（字符串是相等的），会运行一个代码段：

```
if (strName == "Shelley") {
   alert("Your name is Shelley! Good for you!");
}
```

如果字符串不相等，条件语句块后面的第一条语句将会执行。如果使用了一个if...else条件语句，else关键字后面的代码块将会执行：

```
if (strName == "Shelley") {
   alert("Your name is Shelley! Good for you!");
} else {
   alert("Your name isn't Shelley. Bummer.");
}
```

有一些因素会影响到字符串比较的成功。例如，字符串有大小写，并且，可能是由大写字母、小写字母，或者二者的混合形式组成的。除非大小写不能改变，在进行比较之前，你很可能想使用内建的String方法toLowerCase和toUpperCase，把字符串转换为全部小写或全部大写，如下面的代码所示：

```
var strName = prompt("What's your name?", "");

if (strName.toUpperCase () == "SHELLEY") {
   alert("Your name is Shelley! Good for you!");
} else {
   alert("Your name isn't Shelley. Bummer.");
}
```

注意，toUpperCase方法（和toLowerCase方法）不接受任何参数。

在1.2节中，我介绍了在把一个数字值、布尔值或String对象连接成一个字符串的时候，会自动进行数据类型转换。在使用相等运算符的时候，如果一个值不是字符串，也会发生同样的数据类型转换。在下面的代码中，数字10.00转换为字符串“10”，然后用于比较：

```
var numVal = 10.00;
if (numVal == "10") alert ("The value is ten"); succeeds
```

然而，可能有的时候，你不想让自动数据转化发生，例如，想要在比较的值拥有不同的数据类型时让比较失败。如果一个值是字符串直接量，而另一个值是一个String对象，当两个变量具有不同的数据类型的时候，我们想让比较失败。在这种情况下，你需要使用一个不同的相等性运算符，即严格相等运算符（===）：

```
var strObject = new String("Shelley");

var strLiteral = "Shelley";

if (strObject == strLiteral) // 这个比较是成功的

...

if (strObject === strLiteral) // 由于具有不同的数据类型，比较失败
```

如果要比较的两个变量具有不同的数据类型，尽管它们的基本字符串值是相同的，比较也会失败。

有时候，你可能想要专门测试两个字符串是不相等的，而不是相等的。那么，所要使用的运算符就是不相等运算符（!=）和严格不相等运算符（!==）。它们的工作方式与前面刚刚讨论的两种运算符相同，但是，当字符串不相等的时候返回true：

```
var strnOne = "one";
var strnTwo = "two";
if (strnOne != strnTwo) //为真，因为它们不是相同的字符串值
```

如果两个字符串的值不相同，或者两个运算数（运算符两边的值）的数据类型不同，严格不相等运算符返回true：

```
var strObject = new String("Shelley");
var strLiteral = "Shelley";
if (strObject !== strLiteral) //成功，因为两个运算数的数据类型不同
```

如果你对于发现两个字符串如何不相同感兴趣，可以使用其他的比较运算符，见表1-1。

表1-1：比较运算符

| 运算符 | 说明 | 示例 |
|---|---|---|
| 相等 == | 如果运算数是相同的，为真；否则，为假 | var sVal = "this";<br>if (sVal == "this") // true |
| 严格相等 === | 如果运算数相同并且具有相同的类型，为真；否则，为假 | var sVal = "this";<br>var sVal2 = new String("this");<br>if (sVal === sVal2) // not true |
| 不相等 != | 如果运算数不相同，为真；否则，为假 | var sVal = "this";<br>if (sVal == "that") // true |
| 严格不相等 !== | 如果运算数不相等，或者具有不同的数据类型，为真；否则，为假 | var sVal = "this";<br>var sVal2 = new String("this");<br>if (sVal !== sVal2) // true |
| 大于 > | 如果第一个运算数的值大于第二个运算数，为真 | var sOne = "cat";<br>var sTwo = "dog";<br>if (sOne > sTwo) // false |
| 大于或等于 >= | 如果第一个运算数的值大于或等于第二个运算数，为真 | var sOne = "Cat";<br>var sTwo = "cat";<br>if (sOne >= sTwo) // false |
| 小于 < | 如果第二个运算数的值大于第一个运算数，为真 | var sOne = "cat";<br>var sTwo = "Cat";<br>if (sOne < sTwo) // false |
| 小于或等于 <= | 如果第二个运算数的值大于或等于第一个运算数，为真 | var sOne = new String("cat");<br>var sTwo = "cat";<br>if (sOne <= sTwo) // equal, true |

比较运算符对于数字以数字的方式工作，对于字符串以词法的方式工作。例如，值“dog”比值“cat”大，因为“dog”中的字母“d”在字母表中排在“cat”中的字母“c”的后面：

```
var sOne = "cat";
var sTwo = "dog"
if (sOne > sTwo)//为假，因为"cat"在词法上比"dog"小
```

如果两个字符串直接量只是大小写不同，大写字符在词法上比小写字母小：

```
var sOne = "Cat";
var sTwo = "cat";
if (sOne >= sTwo) //为假，因为'c'在词法上比'C'大
```

没有严格大于或严格小于运算符，因此，如果运算数的数据类型不同的话，它们没有意义：

```
var sOne = new String("cat");
var sTwo = "cat";
if (sOne <= sTwo) //都相等，因此为真，因为数据类型不匹配
```

在结束这一节之前，还有另一种方法可以用来比较字符串，但是，这是一个较少用到的方法。它基于String方法localeCompare。localeCompare方法接受一个参数，这是一个字符串，它和该方法所绑定的字符串的值进行比较。该方法返回一个数字值，如果两个字符串相同的话，这个值为0；如果字符串参数从词法上比原始字符串大的话，该值为−1；否则的话，该值为1。

```
var fruit1 = "apple";
var fruit2 = "grape";
var i = fruit1.localeCompare(fruit2); // 返回-1
```

大多数时候，你可能将使用比较运算符而不是localeCompare方法，但是，多了解一种方法总是有好处的。

## 参见

要了解关于字符串转换为数值的更多内容，请参见4.5节。

# 1.4 在字符串中查找子字符串

## 问题

想要知道一个子字符串（特定的一串字符）是否存在于一个字符串中。

## 解决方案

使用String对象内建的indexOf方法来查找子字符串的位置，如果它存在的话：

```
var testValue = "This is the Cookbook's test string";
var subsValue = "Cookbook";

var iValue = testValue.indexOf(subsValue); //返回值为 12，子字符串的索引

if (iValue != -1) // 成功，因为子字符串存在
```

## 讨论

String indexOf方法返回一个数字，表示子字符串的第一个字符的索引或位置，0表示字符串的第一个字符的索引位置。

要测试子字符串是否存在，可以将返回值和-1进行比较，如果没有找到子字符串的话就会返回-1：

```
if (iValue != -1) // 如果找到子字符串的话，结果为真
```

indexOf接受两个参数：子字符串以及一个可选的第二个参数，后者是开始搜索的位置的一个索引值。

```
var tstString = "This apple is my apple";
var iValue = tstString.indexOf("apple", 10); // 返回17，这是第二个子字符串的索引
```

indexOf方法从左向右工作，但是，有时候你可能想要在字符串中从右向左搜索，来查找一个子字符串的索引。还有另一个String方法，即lastIndexOf，它返回在一个字符串中一个子字符串的最后出现的索引位置：

```
var txtString = "This apple is my apple";
var iValue = tstString.lastIndexOf("apple");    // 返回17，
                                                // 子字符串最后一次出现的索引
```

与indexOf一样，lastIndexOf也接受一个可选的第二个参数，它是开始搜索的位置的一个索引值，只不过是从右边开始计数的位置：

```
"This apple is my apple".lastIndexOf("apple"); // 返回值17
"This apple is my apple".lastIndexOf("apple",12); //返回值5
"This apple is my apple".lastIndexOf("apple", 3); //返回值-1，没找到
```

注意，lastIndexOf的返回值根据开始位置不同而有所变化，即从字符串的右边开始计数的位置。

---

**注意：** 很少见到在引用的文本上直接调用一个String方法，但是，在JavaScript中，这与在一个字符串直接量或一个字符串变量上调用该方法没有任何区别。

---

## 参见

String方法search和正则表达式一起使用，以在一个字符串中找到一个特定模式，我们将在第2.3节中介绍它。String方法replace可以使用一个正则表达式来替代所找到的一个子字符串，我们将在第2.4节中介绍它。

# 1.5 从一个字符串提取子字符串

## 问题

有一个字符串是由几个句子组成的，其中的一个句子拥有一个项目列表。该列表以一个冒号开始（:），以一个句点结束（.）。你想要提取这个列表。

## 解决方案

使用indexOf String方法来找到冒号，然后再次使用它找到冒号后面的第一个句点。有了这两个位置，使用String substring方法提取字符串。

```
var sentence = "This is one sentence. This is a sentence with a list of items:
cherries, oranges, apples, bananas.";

var start = sentence.indexOf(":");
var end = sentence.indexOf(".", start+1);

var list = sentence.substring(start+1, end);
```

## 讨论

这个列表有一个起始的冒号字符和一个结束的点号分隔开。在第一个搜索中使用indexOf方法来查找冒号的时候，不带有第二个参数；再次使用该方法的时候，冒号的位置（加上1）用来修改查找点号的起始位置：

```
var end = sentence.indexOf(".",start+1);
```

如果没有修改对结束的点号的搜索，可能最终得到了第一个句子的点号的位置，而不是第二个句子的点号的位置。

一旦拥有了列表的开始位置和结束位置，使用substring方法，传入了表示字符串的开始位置和结束位置的两个索引值：

```
var list = sentence.substring(start+1, end);
```

list中的结果字符串是：

```
cherries, oranges, apples, bananas
```

然后，我们可以使用一个方法，例如String.split，把列表分割为其单个的值：

```
var fruits = list.split(",") ; //值的数组
```

还有另一个字符串提取方法——substr，但是，它是基于子字符串开始的索引位置的，然后，传入子字符串的长度作为第二个参数。在现实的应用程序中，我们不知道句子的长度。

## 参见

参见第1.7节了解使用String.split方法的更多信息。

# 1.6 检查一个存在的、非空的字符串

## 问题

想要检查一个已经定义了的变量，是一个字符串，并且它不为空。

## 解决方案

使用typeof运算符、通用的valueOf方法（这都是JavaScript对象共享的）以及String length属性来创建一个条件测试，以确保一个变量是存在的，是一个字符串，并且不为空：

```
//如果变量存在，是一个字符串，并且其长度大于0，结果为真
if(((typeof unknownVariable != "undefined") &&
     (typeof unknownVariable.valueOf() == "string")) &&
    (unknownVariable.length > 0)) {
   ...
}
```

## 讨论

可能String最重要的内建属性是length。你可以使用length来了解字符串有多长，并且测试字符串变量是否是一个空字符串（长度为0）：

```
if (strFromFormElement.length == 0) //测试是否是空字符串
```

然而，当你使用字符串的时候并且不确定它们是否存在的时候，你是不能检查其长度的，因为如果还没有设置该变量的话，你将会得到一个未定义的JavaScript错误。必须把长度检测和另一项存在性测试组合起来，并且，这会让我们了解typeof运算符。

JavaScript typeof运算符返回一个变量的类型。可能的返回值如下所示：

- 如果变量是一个数字，返回"number"。
- 如果变量是一个字符串，返回"string"。
- 如果变量是一个布尔类型，返回"boolean"。
- 如果变量是一个函数，返回"function"。
- 如果变量是null、一个数组，或者其他的JavaScript对象，返回"object"。
- 如果变量未定义，返回"undefined"。

最后一个值现在对我们就有意义，因为，一个没有定义的变量拥有undefined数据类型。

当数据类型测试和字符串长度测试通过一个逻辑AND(&&)运算符组合起来的时候，整个语句成功的唯一的机会，是该变量已经定义，并且它包含一个长度大于0的字符串：

```
//如果变量存在，并且长度大于0，就成功
if ((typeof unknownVariable !== "undefined") && (unknownVariable.length > )) {
  ...
}
```

如果第一个测试失败，即变量没有定义，那么，第二个测试不会继续，因为整个语句会失败。这就预防了访问一个未定义的变量的属性的错误。

上面的代码段中的条件语句是有效的，但是，如果变量定义了，但不是一个字符串，将会发生什么情况呢？例如，如果该变量是一个数字？好了，在这种情况下，条件将会失败，因为对于一个数字来说，length属性是未定义的。然而，如果该值是一个String对象的话，将会怎么样呢？

如果你不确定变量的类型是什么，也可以在测试长度之前，明确地测试"string"数据类型：

```
//如果字符串的长度大于0的话，成功
if ((typeof unknownVariable == "string") && (unknownVariable.length > 0)) {
...
}
```

如果这个测试成功，你确切地知道拥有什么：一个字符串，其长度大于0。然而，如果这个变量是一个String对象，而不是一个直接量，typeof将返回一个"object"数据类型而不是"string"。这就是为什么这一解决方案要加入另一个JavaScript对象方法：valueOf。

valueOf方法对于所有的JavaScript对象都可用，并且不管对象是什么，都返回其基本

值：对于Number、String和布尔类型，也就是它们的原始值；对于函数，是函数文本；依次类推……因此，如果该变量是一个String对象，valueOf返回一个字符串直接量。如果该变量已经是一个字符串直接量，对其应用valueOf方法会临时性地将它封装成一个String对象，这意味着，valueOf仍然将返回一个字符串直接量。

然后，我们的条件测试最终加入了一个测试，来看看该变量是否已经设置，并且，如果设置了，使用valueOf检测它是否是一个String对象或直接量，最终，该字符串的长度是否大于0：

```
//如果变量存在，是一个字符串，并且长度大于0，结果为真
if(((typeof unknownVariable != "undefined") &&
     (typeof unknownVariable.valueOf() == "string")) &&
    (unknownVariable.length > 0)) {
   ...
}
```

看上去似乎是很多工作，但是，通常你的应用程序在测试一个值的时候不一定如此广泛。一般情况下你只需要测试一个变量是否已经设置，或者获知一个字符串的长度，从而确保它不是一个空字符串。

# 1.7 将一个关键字字符串分解为单独的关键字

## 问题

有一个带有关键字的字符串，关键字由逗号隔开。你要将该字符串分解为单个关键字的一个数组，然后使用一个关键字标签来打印出关键字。

## 解决方案

使用String split方法根据逗号来分割字符串。遍历数组，打印出这些单个的值。示例1-1给出了一个展示这一方法的完整Web页面。关键字由Web页面通过一个提示窗口读取，然后处理关键字并打印到该Web页面上。

示例1-1：展示使用String split来获取关键字列表

```
<!DOCTYPE html>
<head>
<title>Example 1-1</title>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8" >
<script type="text/javascript">

window.onload = function() {

  // 获取关键字列表
   var keywordList = prompt("Enter keywords, separated by commas","");
```

```
  // use split to create array of keywords
  var arrayList = keywordList.split(",");

  // 构建最终HTML
  var resultString = "";
  for (var i = 0; i < arrayList.length; i++) {
    resultString+="keyword: " + arrayList[i] + "<br />";
  }

  //打印到页面
  var blk = document.getElementById("result");
  blk.innerHTML = resultString;
}

</script>
</head>
<body>
<div id="result">
</div>
</body>
</html>
```

## 讨论

String split方法接受两个参数：一个是必需的参数，带有表示split方法使用的分隔符的字符；第二个参数是可选的，是表示进行分割的次数的一个数字。在示例1-1中，分隔符是逗号（,），没有提供第二个参数。使用第二个参数的示例如下：

```
var strList = "keyword1,keyword2,keyword3,keyword4";
```

如下的split方法调用将产生带有两个条目的一个数组：

```
var arrayList = strList.split(",",2); // 生成两元素的一个数组
```

不指定第二个参数将会在每次找到分隔符的时候都分割：

```
var arrayList = strList.split(","); // 生成四元素的一个数组
```

这里有split的一种有趣的用法，如果你想要根据每个字符来分割一个字符串，指定空字符串(''')或(""")作为分隔符：

```
var arrayList = strList.split("");
```

也可以使用一个正则表达式作为参数来进行分割，尽管这可能有点复杂。例如，要找到与1.5节的解决方案示例代码所返回的相同的句子列表，可以使用几个正则表达式：

```
var sentence = "This is one sentence. This is a sentence with a list of items:
cherries, oranges, apples, bananas.";
var val = sentence.split(/:/);
```

```
alert(val[1].split(/\./)[0]);
```

这个正则表达式先查找一个冒号，然后，将它用于第一次分割。第二次分割在第一次分割的基础上使用一个正则表达式来查找点号。该列表现在位于结果的第一个数组元素中了。

这里需要点技术，而且熟练掌握有点难，但是，在没有其他方法可用的时候，将正则表达式和split一起使用是一种方便的选择。

## 参见

参见第5.3节关于从一个数组创建一个字符串的讨论。参见第11.1节关于使用文档对象访问一个页面元素的讨论，以及第12.1节关于使用innerHTML属性的介绍。第2章详细介绍了正则表达式。第2.6节介绍了使用捕获圆括号和一个正则表达式来获取与本节解决方案中同样的结果。

# 1.8 插入特殊字符

## 问题

想要向字符串中插入一个特殊字符，例如一个换行。

## 解决方案

在字符串中使用转义序列之一。例如，要向添加到页面的一段文本中添加一个版权符号（见图1-1），使用转义序列\u00A9：

```
var resultString = "<p>This page \u00A9 Shelley Powers </p>";

// 打印到页面
 var blk = document.getElementById("result");
 blk.innerHTML = resultString;
```

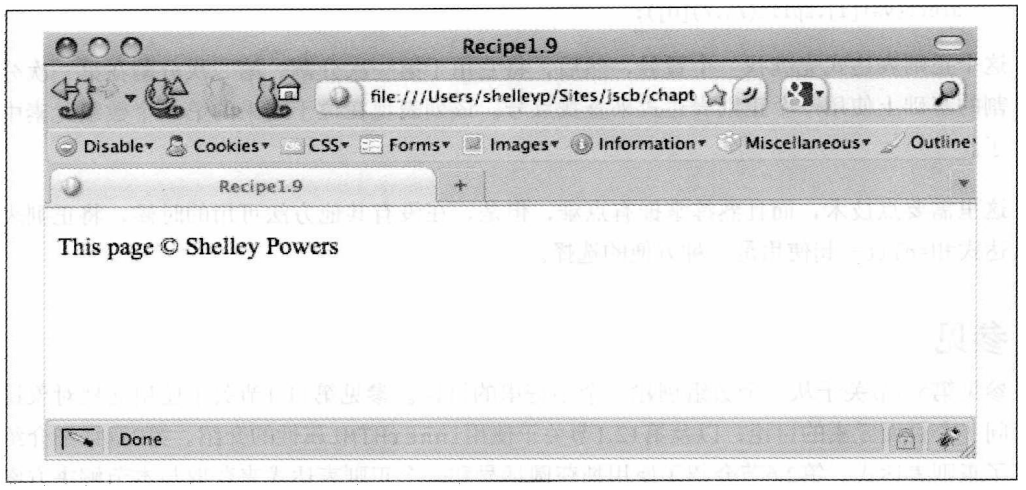



**图1-1：展示使用转义序列创建版权符号的页面**

## 讨论

JavaScript中的转义序列都以一个反斜杠开始(\)。这个字符告诉处理字符串的应用程序，后续的部分是需要特殊处理的一个字符序列。

表1-2列出了其他的转义序列。

**表1-2：转义序列**

| 序列 | 字符 |
|---|---|
| \' | 单引号 |
| \" | 双引号 |
| \\ | 反斜杠 |
| \b | 退格 |
| \f | 换页符 |
| \n | 换行 |
| \r | 回车 |
| \t | 水平制表符 |
| \ddd | 八进制序列（3位:ddd） |
| \xdd | 十六进制序列（2位:dd） |
| \udddd | Unicode序列（4位hex数：dddd） |

表1-2中的最后三个转义序列是模式，它们提供不同的数字值，将导致不同的转移序列。解决方案中的版权符号，就是Unicode序列的一个例子。

表1-2中列出的所有转义序列也可以用一个Unicode序列来表示。例如，水平制表符（\t）也可以表示为Unicode转义序列\u0009。当然，如果用户代理忽略特殊字符，就像浏览器对水平制表符所做的那样，这种用法没有什么实际意义。

转义序列最重要的用法之一，是在双引号或单引号分隔开的字符串之中，包含双引号或单引号：

```
var newString = 'You can\'t use single quotes in a string surrounded by single
quotes';
```

# 1.9 处理textarea的单个行

## 问题

你想要能够处理textarea框中的单个行。

## 解决方案

使用String split方法，结合换行转移序列（\n），把textarea的内容分割到其单独的行中：

```
var txtBox = document.getElementById("inputbox");
var lines = txtBox.value.split("\n");

// 将最后一行打印到页面
var blk = document.getElementById("result");
blk.innerHTML = lines[lines.length-1];
```

## 讨论

转义序列的用途远不止是构建字符串，它们还用于模式匹配运算中。在这个解决方案中，通过查找换行字符的转义序列（\n），把textarea分割到其单个的行中。

这也是将可能使用转义序列编码的文本（例如，换行字符）转换为正确格式的HTML的一种方法。例如，要修改示例以严格按照输入来输出textarea，但是作为HTML输出，使用如下代码：

```
// 获取textarea字符串，并根据换行符分割
var txtBo x = document.getElementById("test");
var lines = txtBox.value.split("\n");

//生成文本的HTML版本
var resultString = "<p>";
for (var i = 0; i < lines.length; i++) {
```

```
    resultString += lines[i] + "<br />";
}

resultString += "</p>";

// 打印到页面
var blk = document.getElementById("result");
blk.innerHTML = resultString;
```

这段代码将所有的换行转换成HTML br元素。当添加回页面的时候，文本按照它们在textarea中的样子显示，包括换行和所有内容。这是用来响应评论的一种常用技术，因为评论在很多博客中是即时预览的时候输入的。

## 参见

替换字符串中的字符的另一种方法是，将一个正则表达式和String replace方法一起使用，第2.5节中介绍了这一点。

# 1.10 去除字符串末尾的空白

## 问题

你想要去除从表单元素获取的一个字符串周围的空白。

## 解决方案

使用新的ECMAScript 5 String trim方法：

```
var txtBox = document.getElementById("test");
var lines = txtBox.value.split("\n");
var resultString = "";

for (var i = 0; i < lines.length; i++) {
    var strng = lines[i].trim();
    resultString += strng + "-";
}
alert(resultString);
```

## 讨论

在ECMAScript 5发布之前，必须使用正则表达式和String replace方法来去除掉一个字符串周围不需要的空白。现在，要修整一个字符串只需要简单地调用trim。

---

**注意：** Firefox已经支持一个trim方法，但是，在ECMAScript 5之前，其用法没有标准化。

---

大多数（如果不是所有的）浏览器最终都将支持trim。在本书的目标浏览器中，只有IE 8不支持它。下面给出了一个可供使用的替代方法，它不仅允许使用trim，而且如果trim不存在的话，它也是一种备用方案。

首先，在你需要使用trim功能的时候，必须测试看看trim是否作为String对象的一个属性存在。如果它不存在，你需要使用String prototype给该对象添加一个定制的trim：

```
if (typeof String.trim == "undefined") {
   String.prototype.trim = function() {
      return this.replace(/(^\s*)|(\s*$)/g, "");
   }
}
```

一旦执行这段代码，当你对任何字符串调用trim的时候，它将返回两端的空白都去除掉的一个字符串（只要应用程序在页面的范围之内）。这一功能的实现，与该方法是否已经由浏览器内建，或是否通过你的hack添加无关。

```
var strng = lines[1].trim();
```

---

**注意**：大多数JavaScript框架库，例如第17章介绍的jQuery，都已经添加了trim方法。

---

ECMAScript 5中其他新的、相关的方法是trimLeft和trimRight，前者会去除掉字符串左边的空白，后者会去除掉字符串右边的空白。

## 参见

第2章将介绍正则表达式的用法。JavaScript对象prototype属性的用法将在第16.3节中介绍。

# 1.11 左补充或右补充一个字符串

## 问题

你需要创建一个字符串，使用给定的字符来补充其左边或右边。

## 解决方案

测试字符串的长度，然后，产生由重复的、给定的字符所构成的一个补充字符串，要么将其连接到最初字符串后面（如果从右边补充的话），要么将其附加到字符串的开始处

（如果从左边补充的话）。如下代码，是使用连续的指定字符（ ，即空格）来左补充已有的字符串：

```
<!DOCTYPE html>
<head>
<title>Recipe 1.12</title>
</head>
<body>
<div id="result"></div>
<script>

  var prefLineLength = 20;
  var oldStr = "This is a string";

  var diff = prefLineLength - oldStr.length;
  var filler = ' ';

  for (var i = 0; i < diff; i++) {
       oldStr=filler + oldStr;
  }

  document.getElementById("result").innerHTML=oldStr;

</script>

</body>
```

## 讨论

你不想补充将要保存到数据库中的字符串，因为，你想要保持数据库中的数据尽可能地小而且高效。但是，在Web页面中显示它之前，你可能需要补充该值。

用于补充字符串的字符，根据其用途而定。通常，我们将使用空格。然而，如果该值是要插入到忽略多个空格的一个Web页面中，内容需要与XHTML兼容，你要么必须使用指定的实体（ ），要么使用其等价的数字（ ）。或者，你可以只使用CSS来格式化文本位置。例如，要右对齐文本，创建如下的CSS规则：

```
.rightformatted
{
   text-align: right:
}
```

然后，当你要向页面添加字符串的时候，应用该规则。你可以把CSS规则用做一个带有innerHTML属性的类名：

```
var div = document.getElementById("item");
item.innerHTML="<p>" + strValue + "</p>";
```

或者，可以使用文档对象模型（Document Object Model，DOM）层级2的功能：

```
var num = 123.55;
var item = document.getElementById("item");

//创建一个文本节点和段落元素
var txt = document.createTextNode(num);
var p = document.createElement("p");

// 把文本节点附加到段落
p.appendChild(txt);
p.setAttribute("class","rightformatted");

//把段落附加到文档元素
item.appendChild(p);
```

## 参见

参见第11章和第12章，了解关于使用DOM访问、创建和删除Web页面元素及元素属性的更多内容。

# 第2章

# 使用正则表达式

## 2.0 简介

正则表达式是可以用来查找与给定模式匹配的文本的搜索模式。例如，在上一章中，我们在一个较长的字符串中查找子字符串Cookbook：

```
var testValue = "This is the Cookbook's test string";
var subsValue = "Cookbook";

var iValue = testValue(subsValue); //返回值12，即子字符串的索引
```

这段代码有效，因为我们要查找一个严格的匹配。但是，如果想要一个更加通用的搜索，该怎么办呢？例如，我想要在“Joe's Cooking Book”或“JavaScript Cookbook”这样的字符串中搜索单词Cook和Book。

当要查找匹配一个模式的字符串，而不是一个具体的字符串的时候，我们需要使用正则表达式。我们也可以尝试用String函数来做到这点，但是，最终实际上使用正则表达式要更为简单，尽管语法和格式要略微奇怪一点，并且不一定“用户友好”。

最近，我在查看从一个字符串提取RGB的代码，以便将颜色转换为十六进制的格式。我试图只是使用String.split函数，并且根据逗号来分割，但是，随后我必须去除掉圆括号和不相关的空白。另一个需要考虑的问题是，如何可以确保值是十进制的格式呢？

我们可能找到的是：

```
rgb (100%, 0, 0)
```

而不是：

```
rgb (255, 0, 0)
```

还有另一个问题：一些浏览器将颜色（例如，一个背景颜色）返回为一个RGB值，而不是一个十六进制值。在构建一个一致的转换程序的时候，你需要两种形式都能处理。

最后，一组正则表达式使得我们能够解决这些问题。起初，这些看上去好像是鸡毛蒜皮的问题，但是，最终，它们变得要复杂很多。在流行的jQuery UI库的一个例子中，正则表达式用来匹配颜色值。这是一项复杂的任务，因为该颜色值可能呈现多种不同的格式，正如程序的这一部分所示：

```
// 查找#a0b1c2
if (result = /#([a-fA-F0-9]{2})([a-fA-F0-9]{2})([a-fA-F0-9]{2})/.exec(color))
    return [parseInt(result[1],16), parseInt(result[2],16), parseInt(result[3],16)];

// 查找#fff
if (result = /#([a-fA-F0-9])([a-fA-F0-9])([a-fA-F0-9])/.exec(color))
    return [parseInt(result[1]+result[1],16), parseInt(result[2]+result[2],16),
parseInt(result[3]+result[3],16)];

// 在Safari 3中查找rgba(0, 0, 0, 0) == transparent
if (result = /rgba\(0, 0, 0, 0\)/.exec(color))
    return colors['transparent'];

// 否则，我们很可能处理一个命名的颜色
    return colors[$.trim(color).toLowerCase()];
```

尽管这个正则表达式似乎有些复杂，它们实际上只不过是描述模式的一种方式。在JavaScript中，正则表达式通过RegExp对象来管理。

## RegExp直接量

与我们在第1章中学习的String一样，RegExp也可以是一个直接量和一个对象。要创建一个RegExp直接量，使用如下的语法：

```
var re = /regular expression/;
```

正则表达式模式包含在开始的斜杠和结束的斜杠之间。注意，这个模式不是一个字符串，你不想使用单引号或双引号括起模式，除非引号本身也是要匹配的模式的一部分。

正则表达式是由字符组成的，要么只是字符，要么是与特殊字符的组合，后者提供更加复杂的匹配。例如，下面是匹配包含单词Shelley和单词Powers的一个字符串的模式的正则表达式，这两个单词依次出现，之间有一个或多个空白字符隔开：

```
var re = /Shelley\s+Powers/;
```

这个示例中的特殊字符是反斜杠字符（\），它有两个目的：要么与一个常规字符一起使用，表明该字符是一个特殊字符；要么与一个特殊字符一起使用，例如，加号（+），表明该字符应该当做直接量对待。在这个例子中，反斜杠与“s”一起使用，将该字母转换为一个特殊字符，表示空白字符，例如一个空格、制表符、换行或换页。\s特殊字符后面跟着一个加号，即\s+，其中+是一个标志，表示匹配之前的字符（在这个例子中，是一个空白字符）一次或多次。这个正则表达式将对如下形式有效：

```
Shelley Powers
```

它也会对下面的形式有效：

```
Shelley    Powers
```

但它对下面的形式无效：

```
ShelleyPowers
```

Shelley和Powers之间有多少空白是无关紧要的，因为使用了\s+。然而，使用加号则至少需要一个空白字符。

表2-1给出了JavaScript应用程序中最常用的特殊字符。

**表2-1：正则表达式特殊字符**

| 字符 | 匹配 | 示例 |
|---|---|---|
| ^ | 匹配输入的开始 | /^This/ 匹配“This is...” |
| $ | 匹配输入的结束 | /end$/ 匹配“This is the end” |
| * | 匹配0次或多次 | /se*/匹配“seeee”，也匹配“se” |
| ? | 匹配0次或一次 | /ap?/ 匹配“apple”和“and” |
| + | 匹配一次或多次 | /ap+/匹配“apple”但不匹配“and” |
| {n} | 严格匹配*n*次 | /ap{2}/匹配“apple”，但不匹配“apie” |
| {n,} | 匹配*n*次或多于*n*次 | /ap{2,}/匹配“apple”和“appple”中的所有p，但是不匹配“apie” |
| {n,m} | 匹配至少*n*次，至多*m*次 | /ap{2,4}/匹配“apppppple”中的4个“p” |
| . | 除了换行以外的任何字符 | /a.e/匹配“ape”和“axe” |
| [...] | 方括号中的任何字符 | /a[px]e/匹配“ape”和“axe”，但不匹配“ale” |
| [^...] | 方括号中字符以外的任何字符 | /a[^px]/ “ale”，但是不匹配“axe”或“ape” |
| \b | 匹配边界上的单词 | /\bno/匹配“nono”中的第一个“no” |
| \B | 匹配非边界上的单词 | /\Bno/匹配“nono”中的第二个“no” |

表2-1：正则表达式特殊字符（续）

| 字符 | 匹配 | 示例 |
| --- | --- | --- |
| \d | 从0到9的数字 | /\d{3}/匹配“Now in 123”中的“123” |
| \D | 任何非数字的字符 | /\D{2,4}/匹配“Now in 123”中的“Now” |
| \w | 匹配单词字符（字母、数字、下划线） | /\w/匹配javascript中的“j” |
| \W | 匹配任何非单词的字符（非字母、数字或下划线） | \W/匹配“100%”中的“%” |
| \n | 匹配一个换行 | |
| \s | 一个单个的空白字符 | |
| \S | 一个单个的非空白字符 | |
| \t | 一个制表符 | |
| (x) | 捕获圆括号 | 记住匹配的字符 |

## 作为对象的RegExp

RegExp是一个JavaScript对象，也是一个直接量，因此，它也可以使用一个构造函数来创建，如下所示：

```
var re = new RegExp("Shelley\s+Powers");
```

何时使用这两种形式呢？ RegExp直接量在脚本运行时才编译，因此，当你知道表达式不会修改的时候，使用一个RegExp直接量。一个编译过的版本会更高效。当要修改表达式或构建表达式或提供给运行时的时候，就使用构造函数。

与其他的JavaScript对象一样，RegExp也有一些属性和方法，本章中将会介绍其中最常见的几个。

---

**注意：** 正则表达式很强大，但是使用的时候需要一些技巧。本章更多的是在介绍如何在JavaScript中使用正则表达式，而不是一般性地介绍正则表达式。如果你想要学习有关正则表达式的更多内容，我推荐Jan Goyvaerts和Steven Levithan编写的一本好书《Regular Expressions Cookbook》(O'Reilly)。

---

## 参见

本小节中出现的jQuery函数是一个jQuery内部函数融入了一个定制jQuery插件的变体。第17章将详细介绍jQuery，而第17.7节将介绍一个jQuery插件。

# 2.1 测试一个子字符串是否存在

## 问题

想要测试一个字符串中是否包含另一个字符串。

## 解决方案

使用一个JavaScript正则表达式来定义一个搜索模式，然后，根据要搜索的字符串来应用该模式，使用RegExp test方法。在如下的代码中，我想要匹配带有*Cook*和*Book*这两个单词的任何字符串，两个单词按照前后顺序出现：

```
var cookbookString = new Array();

cookbookString[0] = "Joe's Cooking Book";
cookbookString[1] = "Sam's Cookbook";
cookbookString[2] = "JavaScript CookBook";
cookbookString[3] = "JavaScript BookCook";

//搜索模式
var pattern = /Cook.*Book/;
for (var i = 0; i < cookbookString.length; i++)
  alert(cookbookString[i] + " " + pattern.test(cookbookString[i]));
```

第一个字符串和第三个字符串有一个正确的匹配，而第二个字符串和第四个字符串则没有。

## 讨论

RegExp test方法接受两个参数：要测试的字符串和一个可选的修饰符。它对该字符串应用正则表达式，如果有一个匹配的话，返回true；如果没有匹配的话，返回false。

在这个例子中，模式是单词*Cook*出现在字符串中的某处，并且单词*Book*出现在字符串中*Cook*之后的某处。两个单词之间可以有任意数目的字符，包括没有任何字符，正如模式中用两个正则表达式字符所指定的那样：小数点（.）和星号（*）。

正则表达式中的小数点是一个特殊字符，它匹配换行符以外的任何字符。在示例的模式中，小数点后面跟着星号，它表示匹配之前的字符0次或多次。组合到一起，它们产生一个模式，匹配任何字符0次或多次，除了换行符之外。

在这个示例中，第一个和第三个字符串匹配，因为它们都满足了*Cook*和*Book*之间可以有任何内容的要求。第四个字符串不匹配，因为在字符串中，*Book*出现在了*Cook*的前面。第二个字符串也不匹配，由于*book*的第一个字母是小写的而不是大写的，并且该匹配模式是区分大小写的。

# 2.2 测试不区分大小写的子字符串匹配

## 问题

你想要测试一个字符串是否包含于另一个字符串之中，但是你并不在意两个字符串中的字符的大小写。

## 解决方案

在创建正则表达式的时候，使用忽略大小写标志(i)：

```
var cookbookString = new Array();

cookbookString[0] = "Joe's Cooking Book";
cookbookString[1] = "Sam's Cookbook";
cookbookString[2] = "JavaScript CookBook";
cookbookString[3] = "JavaScript cookbook";

//搜索模式
var pattern = /Cook.*Book/i;
for (var i = 0; i < cookbookString.length; i++) {
  alert(cookbookString[i] + " " + pattern.test(cookbookString[i],i));
}
```

所有4个字符串都匹配该模式。

## 讨论

该解决方案使用了一个正则表达式标志(i)，来修改对模式匹配的限制。在这个例子中，该标志去除了模式匹配必须按照大小写匹配的限制。使用这一标志，*book*的值和*Book*的值都将是匹配的。

还有几个正则表达式标志，参见表2-2。它们可以与RegExp直接量一起使用：

```
var pattern = /Cook.*Book/i; // 'i'是忽略标志
```

也可以在创建一个RegExp对象的时候，通过可选的第二个参数来使用它们：

```
var pattern = new RegExp("Cook.*Book","i");
```

表2-2：正则表达式标志

| 标志 | 含义 |
| --- | --- |
| g | 全局匹配：在整个字符串中匹配，而不是在第一次匹配之后停止 |
| i | 忽略大小写 |
| m | 对多行字符串中的每一行，应用行首和行末的特殊字符（分别是^和$） |

# 2.3 验证社会安全号码

## 问题

你需要验证一个文本字符串是否是一个有效的美国社会安全号码（在美国，这是收税者用来找到我们的身份标识）。

## 解决方案

使用String match方法和一个正则表达式来验证一个字符串是一个社会安全号码：

```
var ssn = document.getElementById("pattern").value;
var pattern = /^\d{3}-\d{2}-\d{4}$/;
if (ssn.match(pattern))
  alert("OK");
    else
      alert("Not OK");
```

## 讨论

美国社会安全号码是9位数字的一个组合，通常依次是3个数字、2个数字和4个数字，之间可能有连字符或者没有。

社会安全号码中的数字可以用数字特殊字符(\d)来匹配。要查找一组数字，可以使用花括号把期待的数字的数目括起来。在这个例子中，前3个数字可以用\d{3}来匹配。

第二组数字可以使用同样的规则来确定。由于数字序列之间只有一个连字符，可以不用任何特殊的字符来给定它。然而，如果字符串也可能有一个不带连字符的社会安全号码，你希望将该正则表达式模式修改为如下所示：

```
var pattern = /^\d{3}-?\d{2}-?\d{4}$/;
```

问号特殊字符（?）匹配其前面的字符0次或者仅1次，在这个例子中，这个字符就是连字符（-）。通过这一修改，如下的字符串将匹配：

```
444-55-3333
```

如下的也会匹配：

```
555335555
```

但是，下面的字符串不会匹配，它的连字符太多了：

```
555---60--4444
```

另一个特点是，检查字符串是否包含社会安全号码（并且只有社会安全号码）。输入开始特殊字符（^）用来表示社会安全号码从字符串的开始处开始，行末特殊字符（$）用来表示该行以社会安全号码的末尾为结束。

由于只是对验证字符串是一个有效格式的社会安全号码感兴趣，我们使用了String对象的match方法。我们还可以使用RegExp test方法，但是，两种方法差不多，都可以接受。

还有其他的方法可以验证社会安全号码，但是较为复杂，是基于社会安全号码可以用空格代替连字符的原理。这就是为什么大多数Web站点要求按照3个不同的输入字段来提供一个社会安全号码，以避免出现各种变体。正则表达式不应该用来替代良好的表单设计。

此外，没有方法真正地验证给定的数字就是一个真实的社会安全号码，除非你拥有关于这个人的更多信息，并且有一个存储所有社会安全号码的数据库。使用正则表达式所能够做的，只是验证数字的格式。

## 参见

有一个站点，它提供了较为复杂的社会安全号码正则表达式，此外还有很多其他有趣的正则表达式“秘诀”，这就是Regular Expression Library。

# 2.4 找到并突出显示一个模式的所有实例

## 问题

想要在一个字符串中找到一个模式的所有实例。

## 解决方案

在一个循环中，使用RegExp exec方法和全局标志（g），来找到一个模式的所有实例，例如，任何以*t*开头并且以*e*结尾、中间有任意多个字符的单词或其他文本：

```
var searchString = "Now is the time and this is the time and that is the time";
var pattern = /t\w*e/g;
var matchArray;

var str = "";
while((matchArray = pattern.exec(searchString)) != null) {
  str+="at "+ matchArray.index + "we found" + matchArray[0] + "<br />";
}
document.getElementById("results").innerHTML=str;
```