Learning JQuery

前言

Learning jQuery 的作者 blog

http://learningjquery.com/

本书覆盖的内容

本书的第一部分会介绍 JQuery，帮助你理解小题大做是怎么回事。

第一章包含的内容有，

下载和安装 jQuery 库，也会教你写第一个脚本。

本书的第二部分会一步步带你学习 jQuery 库的每一个主要的方面。

第二章你会了解到怎样

得到你想要的。jQuery 中选择器表达式让你找到页面上所有的元素，

你将会使用选择器表达

式来样式化页面上不同的元素，有时候可以不是纯 css。

在第三章里，你会学习如何触发事件，浏览器发事件时，你将会使用

jQuery 的事件处理机

制处理行为。你 也能够在 jQuery 的秘密 sauce 中获得内在的消息：

在 页面完成加载前不经意

地附加事件。

在第四章里，你会学习到如何添加功能到你的动作中。将会介绍

jQuery 的动画技术，如很

方便地隐藏、显示和移动页面元素。

在第五章中，你会学习到如何使用命令来改变你的页面。这章会教你如何改变在飞的 html
文档的结构。
在第六章里，你会学习到如何让你的站点时髦与兼容。阅读本章后，你也能够不用刷新页面
就可以访问服务器端的功能。
本书的第三部分会采用不同的方法，在这里你会通过几个实例来学习，汇集你上一章学过的
知识，建立健全的 jQuery 程序解决常见的问题。
在第八章中，你会掌握客户端验证的微妙之处，设计一个自适应的表单布局，并实现交互式
的客户端－服务器端的表单特性，例如自动完成功能。
在第九章中，你 将通过展示它们的一小部分就可以增强页面元素的美感与可用性，你会使信
息本身和用户控制来让信息飞进飞出。
在第十章中，你会学到 jQuery 的可观的扩展能力，你会研究三个突出的 jQuery 插件并使用
它们，然后着手从头开发你自己的插件。
附录 A 提供一个些信息站点，包括广泛的主题如 jQuery、javascript 和 web 开发。

附录 B 推荐一些有用的第三方软件，并 可在你的开发环境中编辑、调 试 jQuery 代码的工具。

附录 C 讨论一个 javascript 语言中常见的难题，你来依赖强大的闭包，而不是害怕它的副作用。

目录

今天的万维网是一个动态的环境，他们的用户为网站的风格和功能设置了一个高标准。为了

创建有趣的交互式的网站，开发者正都转向如 jQuery 这样的 javascript 库，使普通的任务自

动化和简化复杂的任务。jQuery 库是流行的选择的一个原因是它能够协助大范围的任务。

因为 jQuery 有很多不同的函数，它可像有挑战性的知道从哪开始。然而，这个库的设计是

一致(coherence)与对称(symmetry)的。它的概念的大部分都借鉴于 HTML 与 Cascading

Style Sheets (CSS) 的架构。国 为很多 web 开发者使用这些技术比较 javascript 有更多的经

验。库的设计让只有少量编程经验的设计师可以快速入门。事实上，在开始的这章我们会只

用三行代码写一个功能的 jQuery 程序。另一方面，有经验的程序员在概念的一致性得到帮
助，我们将会在后面的更高级的章节中看到。
但我们用一个实例来说明库的操作之前，我们应该在开始的位置讨论一下我们为什么需要
它。

jQuery 能做什么

jQuery 库为共同的 web 脚本提供了一种通用的抽象层，并 且它几乎在每种脚本环境都是有
用的。它的可扩展性意味着我们无法在一本书里涵盖所有可能的用途与功能，它以插件的形
式持续地通过开发加入新的功能。这核心的特性，虽然满足以下的需求：
获取页面的部分内容，不使用 JavaScript 库，必须写很多行代码来遍历 DOM 树，并定位
一一个 HTML 文档的指定部分。jQuery 提供了一个强大而有效的选择机制来返回被检查或
者被操作的文档。
修改页面的外观，CSS 提供了一个影响文档渲染的强大方法，当 web 浏览器不支持同样的
标准时，它却是不尽人意的。jQuery 能弥补这个差距，提供了跨所有浏览器的同样的标准

的支持。另外，即使页面被渲染后，jQuery 仍可改变文档一部分中的类或者独立的样式属
性。
修改页面的内容，不仅限于外观的改变，jQuery 还可以用很少的按键就可修改文档的内容
本身。文本可改变，图像可插入或替换，列表可重新排序或者整个 HTML 结构可被重写和扩

展，完成这些只需一套非常易用的 API 函数。

在页面中响应用户的交互，当它们发生时，如果我们不能控制，即使是最周密最强大的行为

也是没有用的。jQuery 库提供了一个优雅的方法来截取多种事件，例如用户单击链接，我

们不需要将事件句柄混杂到 HTML 代码中。同时，事件句柄 API 删除浏览器不一致性，

往往会让 web 开发者感到很烦恼。

给页面加上动画，为 了有效地执行交互行为，设 计师必须提供可视的反馈给用户，j Query 库

提供了一组效果来推进它，效果如褐色，清空来，也可定制一套新的工具。

无刷新返回服务器端的信息，这个代码模式已经以 Asynchronous JavaScript and XML

（AJAX）著称了，并协助 web 开发者制作可响应的功能丰富的网站。

简化共同的 JavaScript 任务，除了 jQuery 指定的文档的所有特性外，这个库还提供了改

进基本的 JavaScript 结构，如秩代和数组操作。

为什么 jQuery 工作那么好

随着近期动态 HTML 兴趣的复兴，JavaScript 框架快速增加。有些

很专业，侧重于一个或
者两个以上的任务。其它的则企图列出一切的行为与动画，这些服务都归结于预先包装。为
了保持其广泛的特性而不失简洁，jQuery 使用了一些策略。
CSS 的杠杆知识，以 CSS 选择器定位页面元素机制为基础，jQuery 继承了表现文档结构
的简洁(terse)而易读(legible)的方法。由于做专业的 web 开发的必备知识是 CSS 语法，
jQuery 为想在页面加上行为的设计师提供了一个入口。
支持扩展，为了避免特征变化，jQuery 提交了专项用途的插件。创建一个新的插件的方法
是很简单的和有明确记录的，并且已经带动发展了各种发明和有用的模块。甚至在基本的
jQuery 下载中的大部分特性都是通过插入式结构内部实现的，可如预期地被删除，产生一
个更小的库。
去除浏览的错误，web 开发的一个不幸的现实是每个浏览器都有一套自己的偏离发布的标
准，任何网络应用的一个重要的部分可在每个平台上处理不同的特性。但是不断变化的浏览
器现状使得一些高级特性不能可编写出完美的浏览器中立的基本代码，jQuery 加入了一个

抽象层来规范这些共同的任务，并减少代码的大小，尽量简化它。

总是以集合工作，当我们通知 jQuery，查找所有带类 collapsible 的元素并隐藏它，不必循环

每个返回的元素。相反，如 .hide()方法被设计工作在对象集上而不是单独元素。这种技术

被称为隐含秩代（implicit iteration），意味着很多循环结构变得不必要了。

允许在一行有多个动作，为了避免过度使用临时的变量或浪费的重复，jQuery 在它的大多

数方法中使用了一个称为链式的编程模式，这 意味着在对象的很多操作的结果都这个对象的

本身，为下一个动作做准备以应用它。

这些战略一直保持 jQuery 包很轻巧，压缩后只有 20KB 左右，同时，也提供了保持使用这
个库的自定义代码的简单性的技术。

这个优雅的库部分由设计而来，部 分是由进化过程中由于项目涌现的社区的推动。jQ uery 的
用户不但聚在一起讨论插件的开发，而且也改进了核心库。附录 A 详细说明了许多对
jQuery 开发者有用的社区资源。

尽管所有的努力需要设计出如此灵活和强有力的系统，但 最终的产品对所有人都是自由使用
的。这 个开源项目是在 GNU Public License(appropriate for inclusion in many other opensource
projects)和 MIT License(to facilitate use of jQuery within proprietary software)下的双重许可的。

我们的第一个 jQuery 文档

既然我们已经涵盖了对我们有用的 jQuery 的所有特性，我们就可测试如何实践这个库了。

下载 jQuery

官方 jQuery 网站（http://jquery.com/）总是有最新的代码资源和与库相关的信息，为了开始

实践，我们需要一个 jQuery 的拷贝，可以从网站首页直接下载，一些 jQuery 版本可能在
任何时刻都是有用的，最适合我们的将会是最新的没有压缩的版本。不需要安装它，为了使
用 jQuery，我们只需要放它到我们网站的一个公共位置，既然 JavaScript 是一种解释型语
言，那么不用担心会有编译和创建阶段。无论何时我们需要的话，jQuery 都是有效的，我
们简单地从 HTML 文档中参考这个文件的位置就可以了。

建立 html 文档

很多 jQuery 用法的实例都有三块：H TML 文档本身，C SS 文件用来样式化它和 JavaScript
文件来执行它。我们的第一个实例中，我们用一本书的摘录的一个页面，页面将有很多类应
用到它不同的部分。

```
<?xml version="1.0" encoding="UTF8"
?>
<!DOCTYPE html PUBLIC "//
W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1transitional.
dtd">
<html  xmlns="http://www.w3.org/1999/xhtml"  xml:lang="en"
```

```
lang="en">

<head>

<meta httpequiv="

ContentType"

content="text/html.

charset=utf8"/>

<title>Through the LookingGlass</

title>

<link rel="stylesheet" href="alice.css" type="text/css"

media="screen" />

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

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

</head>

<body>

<div id="container">
```

```
<h1>Through the LookingGlass</
h1>
<div class="author">by Lewis Carroll</div>
<div class="chapter" id="chapter1">
<h2 class="chaptertitle">
1. LookingGlass
House</h2>
<p>There was a book lying near Alice on the table, and while
she sat watching the White King (for she was still a
little anxious about him, and had the ink all ready to
throw over him, in case he fainted again), she turned over
the leaves, to find some part that she could read, <span
class="spoken">"&mdash.for it's all in some language I
don't know,"</span> she said to herself.</p>
<p>It was like this.</p>
<div class="poem">
<h3 class="poemtitle">
YKCOWREBBAJ</h3>
<div class="poemstanza">
<div>sevot yhtils eht dna ,gillirb sawT'</div>
```

.ebaw eht ni elbmig dna eryg diD

,sevogorob eht erew ysmim llA

.ebargtuo shtar emom eht dnA

She puzzled over this for some time, but at last a bright thought struck her. "Why, it's a Lookingglass book, of course! And if I hold it up to a glass, the words will all go the right way again."

This was the poem that Alice read.

<h3 class="poemtitle">
JABBERWOCKY</h3>

'Twas brillig, and the slithy toves

Did gyre and gimble in the wabe.

All mimsy were the borogoves,

And the mome raths outgrabe.

</body>

</html>

注意：在服务器上文件的实际的层没有问题。一个文件参考到另一个文件只需要调整匹配我们选择的

组织。本书的很多例子里，我们用相对路径来参考文件（../images/foo.png），不用绝对路径

（/images/foo.png）。这允许代码可本地运行，而不必在 web 服务器上。

Immediately following the normal HTML preamble, the stylesheet is loaded. For this example,

we'll use a Spartan(斯巴达的) one.

马上跟着常规的 HTML 导言，样式表已被加载。

```
body {
font：62.5% Arial, Verdana, sansserif.
}
h1 {
fontsize：
2.5em.
marginbottom：
0.
}
h2 {
fontsize：
1.3em.
marginbottom：
.5em.
}
h3 {
```

```
fontsize：
1.1em。
marginbottom：
0。
}
.poem {
margin：0 2em。
}
.emphasized {
fontstyle：
italic。
border：1px solid #888。
padding：0.5em。
}
```

样式表被参考后，JavaScript 文件被包含。jQuery 库的脚本标签放到我们自定义的脚本标签之前是很重要的，否则，我们的代码尝试参考 jQuery 的时候，jQuery 框架就不起作用。

注意：本书剩下的部分，只有与 HTML 和 CSS 相关的部分才会打印。完整的文件可以从本书的伙伴网站 http://book.learningjquery.com 或者本书出版的网站

http://www.packtpub.com/support 中获得。

现在我们有一个像下面的页面：

我们用 jQuery 来给 peom text 应用一个新的样式。

这个例子被设计只是来显示 jQuery 的一个简单的应用。实际情况，我们会用 纯粹的 CSS

来样式化。

书写 jQuery 代码

我们自己的代码将会是在第二步，通常是空的 JavaScript 代码，我们从 HTML 用<script

src="alice.js" type="text/javascript"></script>来包含它。这个实例中，我们只需要三行代码：

```
$(document).ready(function() {
$('.poemstanza').
addClass('emphasized').
}).
```

查找 Poem 文本

jQuery 基本的操作是选择文档的一部分，这个工作使用 $() 结构来完成。通常，它把一个

可包含任何 CSS 选择器表达式的字符串作为参数。既然这样，我们希望查找包含

poemstanza

类的文档的所有部分，所以这个选择器是很简单的，但通过本书的课

程我们会涵盖更为

顶尖的选择。在第 2 章我们会透过定位一个文档的部分的不同方法来继续学习。

$() 函数其实是一个 jQuery 的一个工厂，它是一个基本的创建块，从现在开始我们会使用

它来工作。jQuery 对象封装了 0 个以上的 DOM 元素，并允许我们用很多不用的方法来与

它们交互。既然这样，我们希望修改页面的这些部分的外观，并且我们通过应用到 poem 文

本的类来完成。

加入新类

.addClass() 方法是不说自明的，它应用一个 CSS 类到我们选择的页面部分。它唯一的参数

是加入的类的名称。这个方法和它类似的方法，.removeClass()，当我们研究不同而有用的

选择器表达式时允许我们在运行中观察 jQuery。现在我们的实例简单地加入定义了斜体和

边框样式的 emphasized 类。

注意，不必循环给所有的 poem 类加上这个类，我们讨论过，jQuery 在如 .addClass()这样
的方法里使用隐含循环，所以调用单独的函数是修改了文档所有选择的部分。

执行代码

$() 和 .addClass() 一起使用足够完成改变 poem 文本外观的目的。然而，如果这行代码被
单独地插入到文档的头部，将会看不到任何效果。一般地，只要在浏览器中遇到 JavaScript，
那么 JavaScript 代码就会运行，同时头部信息（header）被处理，（no 变成相反了，这里根
据我们的理解翻译）HTML 被样式后显示。我们要延迟代码的执行直到 DOM 有效后。

JavaScript 代码运行的传统控制机制是在事件句柄里调用代码。对用户驱动的事件许多句柄
都是有效的，例如鼠标单击和按下键盘。如果没有 jQuery 可以使用，我们就需要依赖
onLoad 句柄，页面（与它里面的所有图像）被渲染后激发。为了从 onload 触发我们的代
码，我们把代码放进一个函数里面：

```
function emphasizePoemStanzas() {
$('.poemstanza').
addClass('emphasized').
}
```

然而，我们修改 HTML <body> 标签，附加一个函数给这个事件来参考它。

```
<body onload="emphasizePoemStanzas()."></body>
```

This causes our code to run after the page is completely loaded.

这个页面完全被加载后才我们的代码才会被执行。

可是这个方法有一些缺点，我们修改 HTML 本身来改变这个行为。

这个紧密的结构和函数

混乱了代码，可 以在许多不同的页面重复调用这个相同的函数，或者在其它事件如在页面上

每个元素的实例进行鼠标单击。在 两个不同的地方加入一个新的行为会需要进行循环，增加

了产生错误的机会，并且使得设计师和程序员并行工作复杂化了。

为了避免这个缺陷，jQuery 允许我们确定函数调用的时间，DOM 被加载后才调用。使用

$(document).ready() 结构，不需要等图像加载。用我们上面定义的函数，我们可以这样写：

```
$(document).ready(emphasizePoemStanzas).
```

这个技术并没有要求任何的 HTML 修改。反 而，这 个行为从

JavaScript 文件里完全地附加。

在第 3 章，我们会学习如何响应其它类型的用户动作，从 HTML 结构中脱离他们的效果。

可是这个化身仍然有点浪费，因为这个被定义的函数 emphasizePoemStanzas() 被马上使用并

只用一次。这意味着我们在这个全局的函数命名空间中使用过一个标识了，我 们不得不记住

不要再次使用它。JavaScript 像其它的编程语言一样，有一个围绕无效的方法称为匿名函数

（有时也被称为 lambda 函数）。我们回到最初展现的代码：

```
$(document).ready(function() {
$('.poemstanza').
addClass('emphasized').
}).
```

用 function 这个关键词而不是用一个函数名，我们详细地定义了一个我们需要的函数，不

是像之前那样。这就去除了混乱并带回来三行 JavaScript 代码。这个用法在 jQuery 代码中

是非常方便的，因此很多方法把一个函数作为参数用并且这些函数是几乎不被重复的。s

当这种语法被用来在其它函数主体里定义一个匿名函数，一个闭包可被创建。这 是一个高级

且强大的概念，但应该被理解为当扩展嵌套函数定义时，在内存使用上会有一个意想不到的

结果和分流。

成品

既然我们的 JavaScript 在适当的位置，这个页面看起来像这样：

现在这节诗（poem）被设为斜体并被包在盒里，是由 JavaScript 代码插入 emphasized 类实

现的。

小结

现在我们已经知道了为什么一个开发者会选择用一个 JavaScript 框架而不从头开始写所有

的代码，甚至是为最基本的任务。我们也看到 jQuery 作为一个框架，其里面的一些方法的

优秀之处，并 且我们为什么抛弃其它而选择它。通 常我们也知道

jQuery 使得任务更加简单。

在这一章里，我们已经学习了在我们的 web 页面上，如何使 jQuery 对 JavaSript 代码有

效，使用 $() 工厂函数来定位已经加上类的页面的一部分，调用 .addClass() 来对页面的部

分加上另外的样式，调用 $(document).ready() 来引发页面加载上执行代码。

我们已经用这个简单的例子来说明 jQuery 如何工作，但在实际情况不是很有用。在下一章

里，我们会详述代码，据此研究 jQuery 优秀的选择器语言，为这个技术寻找实际的用法。

第二章 选择器－得到你想要的

jQuery 利用 CSS 和 XPath 选择器使我们在文档对象模型(DOM)中快速、容易地访问元素
或者元素组。在本章里，我们会研究一些 CSS 和 XPath 选择器和 jQuery 的自定义选择
器。我们也会看看 DOM 遍历方法，它提供了更灵活的访求来得到我们想要的。

文档对象模型

jQuery 最强有力的方面是它能很容易进行 DOM 遍历。文档对象模型是以家族树排列。
HTML，像其它的标记语言一样，使用这个模型来描述页面上的东西的关系。当我们参考这
些关系，如父节点、子节点等等。一个简单的实例能帮助我们理解对应一个档的家族树的隐
喻。

```
<html>
<head>
<title>the title</title>
</head>
<body>
```

This is a paragraph.

This is another paragraph.

This is yet another paragraph.

</body>

</html>

这时在，< html> 是一个所有其它元素的祖先，换 句话说，所 有其它元素都是 <html>的子孙。

<head> 和 <body> 元素是 <html> 的孩子。因此，除了是 <head> 和 <body> 的祖先外，

<html> 还是他们的父亲。 元素是 的孩子（后代），也是 <body> 和 <html>的

后代，也是其它元素的兄弟。

我们开始之前要注意的很重要的一点是从我们不同的选择器和方法中得到的元素其实就是

jQuery 对象。jQuery 对象很容易与我们在页面找到的元素一起工作。我们也很容易绑定一

个事件到对象，并加入一些效果，也可将多个修改和效果连接在一起。

$() 工厂函数

无论我们想在 jQuery 中使用哪种选择器类型（CSS，XPath 或者自定义），我们总会以美

元符号和圆括（$()）号开始。

在第一章提到，$() 函数不需要做 for 循环来访问一组元素，放在圆括号里面的东西都会自动

的进行循环并作为 jQuery 对象存储。我们可把任何东西放到 $() 函数的圆括号里面。包含

一些更变通的实例：

A tag name：$('p') gets all paragraphs in the document.

An ID：$('#someid')

gets the single element in the document that has the corresponding someid

ID.

A class：$('.someclass')

gets all elements in the document that have a class of someclass.

注意：使 jQuery 与其它 JavaScript 库很好地结合。

在 jQuery 里，$ 是 jQuery 的缩写。因为 $() 函数在 JavaScript 库里是非常普遍的，在一

个给定的页面中，如果我超过一个库被使用，冲突很可能发生。我们可以通过在我们自定义

的 jQuery 代码中使用 jQuery 来替换 $ 来解决这个冲突。另一种解决方法是可在第 10 章

中被找到。

既然这样，我们已经涵盖了基础，那么我们准备开始研究一些更强大的选择器的用法。

CSS 选择器

jQuery 支持大部分的选择器，包括在 World Wide Web Consortium 网站上列出的

（http://www.w3.org/Style/CSS/#specs） CSS 规范 1 到 3。这个支持允许开发者增强他们的

网站，而不必担心有些浏览器（特别是 IE6 或者更低）可能不理解高级选择器，只有这些

浏览器支持 JavaScript 就可以了。

注意：可靠的 jQuery 开发者对他们的代码应该总是有逐步增强和功能降级的概念，即使页

面在 JavaScript 不可用时与 JavaScript 可用时不是一样漂亮，我们也会让页面正确地渲染。

我们会通过本书继续研究这些概念。

为我开始学习 jQuery 与 CSS 选择器一起工作，我们使用了出现在许多网站的一个结构，

就是导航，嵌套无序的列表（List）。

```
<ul id="selectedplays">
<li>Comedies
<ul>
<li><a href="http://www.mysite.com/asyoulikeit/">
As You Like It</a></li>
<li>All's Well That Ends Well</li>
<li>A Midsummer Night's Dream</li>
<li>Twelfth Night</li>
</ul>
</li>
<li>Tragedies
<ul>
<li><a href="hamlet.pdf">Hamlet</a></li>
```

Macbeth

Romeo and Juliet

Histories

Henry IV (email)

Part I

Part II

Henry V

Richard II

注意第一个 <ul> 有一个 selectedplays

ID，但 <li> 标签没有类，也没有任何的样式，这

个列表看似像：

这个嵌套的列表如我们期望的那样显示，一 套竖式排列的子弹项目

并且根据他们的等级进行

缩进。

样式化列表项

让我们假设我们想最高级的项目，仅仅是最高级的项目，被水平排列，

我们可在样式表中定

义一个 horizontal 类来开始：

.horizontal {

```
float：left。
liststyle：
none。
margin：10px。
}
```

Horizontal 类将跟着它的元素向左浮动，删除列表项的子弹样式，并将它的所有面的边界都
加入 10 个像素。

为了说明 jQuery 的选择器的用法我们动态地添加它到列表的最高级项，只有 Comedies、
Tragedies 和 Histories，而不是将 horizontal 类直接附加到我们的 HTML 中。

```
$(document).ready(function() {
$('#selectedplays
> li').addClass('horizontal')。
}).
```

第 13 页

第 1 章讨论过，我们使用 $(document).ready() 包住我们的 jQuery 代码，DOM 加载完毕后

就可以使它所有东西都可用。

第二行使用子结合器（>）来添加 horizontal 类到所有最高级的项。

$（）函数里的选择器意思

是 find each list item (li) that is a child (>) of an element with an ID of selectedplays

(#selectedplays) (

保留原意，不做翻译了)。

现在这个类被应用了，我们的嵌套列表看起来像这样：

可以用很多种不同的方法样式化所有其它不在最高级的项目。既然我们已经应用 horizontal

类到最高级的项目，为了指定那些没有 horizontal 类的所有列表项，我们可使用 negation

pseudoclass

（保留原意，否定伪类）方法获得所有子级的项目。

```
$(document).ready(function() {
$('#selectedplays
> li').addClass('horizontal').
$('#selectedplays
```

li:not(.horizontal)').addClass('sublevel')

.

}).

这时我们得到每个列表项：

1、 是否 ID 为 selectedplays

的元素的子孙

2、 本身没有类 horizontal

当我们添加 sublevel

类到这些项目中，他 们得到灰黄色的背景，在 样式表中定义：.s

ublevel

{backgroundcolor：

#ffc.}。现在这个嵌套的列表看起来像这样：

XPath 选择器

XML 路径语言（XPath）是一种在 XML 文档中指定不同元素或者它们的值的语言，它与

CSS 在 HTML 文档中指定元素的方法相类似。jQ uery 库支持支持一套基本的 XPath 选择

器，如果我们想的话，我们可以让它与 CSS 选择器一起工作。使用 jQuery，不管文档的

类型如何，我们都可使用 XPath 和 CSS 选择器。

当谈到属性选择器时，jQuery 使用 XPath 指定属性的约定，属性通过在方括号里用 @ 符

号作为前缀指定，而不是用 CSS 的方法，它缺乏灵活性。例如，选择所有带有 title 属性

的链接，我们会这样写：

```
$('a[@title]')
```

XPath 语法允许方括号不使用 @ 的另一个用法来指定一个不包含其它元素的元素。例如，

我们可用下面的选择器表达式来得到所有包含一个 ol 元素的 div 元素：

```
$('div[ol]')
```

样式化链接

属性选择器接收类正则表达式的语法来指定一个字符串的开始(^)与结束($)。它们也可用

asterisk(*) 来指示一个字符串的任意位置。

举例来说，我 们想显示带不同文本颜色的不同种链接，我 们首页在我们的样式表中定义样式：

```
a {
color: #00f. /* make plain links blue */
a.mailto {
color: #f00. /* make email links red */
}
```

```
a.pdflink {
color: #090. /* make PDF links green */
}
a.mysite {
textdecoration:
none. /* remove internal link underline */
borderbottom:
1px dotted #00f.
}
```

然后，我们用加入三个类（mailto、pdflink 和 mysite），并用 jQuery 将它们加到适合的链
接。

为了得到所有的 email 链接，我 们创建一个选择器来查找所有的 anchor 元素（a），选 择器
用以 mailto 开头（^="mailto:"）的 href 属性（[@href]），如下：

```
$(document).ready(function() {
$('a[@href^="mailto:"]').addClass('mailto').
}).
```

为了得到所有连接到 PDF 文件的链接，我们使用美元符号($)不是用脱字符号(^)，为了得
到所有以 .pdf 结尾的 href 属性的链接，代码如下：

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

```
$('a[@href^="mailto:"]').addClass('mailto').
$('a[@href$=".pdf"]').addClass('pdflink').
}).
```

最后，为了得到内部链接，例如，在 mysite.com 连接到其它页面，我们用星号：

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

```
$('a[@href^="mailto:"]').addClass('mailto').
$('a[@href$=".pdf"]').addClass('pdflink').
$('a[@href*="mysite.com"]').addClass('mysite').
}).
```

这里，mysite.com 在 href 的值出可出现在任何地方。如果我们也想在 mysite.com 里包含

链接到任何子域名，那么这是特别重要的。

应用到三种链接的三个类，我们应该下面的样式应用：

用虚线下划线的蓝色文本：

```
<a href="http://www.mysite.com/asyoulikeit/">As You Like It</a>
Green text: <a href="hamlet.pdf">Hamlet</a>
Red text: <a href="mailto:henryiv@king.co.uk">email</a>
```

以下是一个样式后的链接的截图：

自定义选择器

对于各种的 CSS 和 XPath 选择器，jQuery 加入了它自定义的选择器，大部分自定义的选

择器允许我们在一个队列外选择某些元素。这个语法是与 CSS 伪类语法，即选择器以冒号

(:)开头。例如，如果我们想从一组匹配选择带有 horizontal 类的 div

中选择第二项，我

们像这样写：

$('div.horizontal:eq(1)')

注意 eq(1) 获得是第二项,因为 JavaScript 的数组编号方式是基于

零的，指的是从零开始

算起。相反，CSS 选择器如$('div:nthchild(

1)')获得任何 div 的父亲的第一个孩子。

样式化交替的行

在 jQuery 库中有两个很有用的自定义选择器是 :odd 和 :even，让

我们看看如何使用这些

选择器为 table 制作基本的间条，给定以下的 table：

<table>

<tr>

<td>As You Like It</td>

<td>Comedy</td>

</tr>

<tr>

<td>All's Well that Ends Well</td>

<td>Comedy</td>

</tr>

<tr>

<td>Hamlet</td>

<td>Tragedy</td>

</tr>

<tr>

<td>Macbeth</td>

<td>Tragedy</td>

</tr>

<tr>

<td>Romeo and Juliet</td>

<td>Tragedy</td>

</tr>

<tr>

<td>Henry IV, Part I</td>

<td>History</td>

</tr>

<tr>

<td>Henry V</td>

<td>History</td>

</tr>

</table>

现在我们可加入两个类到样式表，一个是给奇数行的，一个是给偶数行的。

.odd {

backgroundcolor：

#ffc. /* pale yellow for odd rows */

}

.even {

backgroundcolor：

#cef. /* pale blue for even rows */

}

最后，我们写我们的 jQuery 代码，附加这些类到表格的行（<tr> 标签）：

$(document).ready(function() {

$('tr:odd').addClass('odd').

$('tr:even').addClass('even').

}).

简单的一些代码让表格看起来像这样：

第 17 页

第一眼看见，行的颜色可能是相反显示了。然而，只是因为 :eq()、:odd() 和 :even() 选择

器使用 JavaScript 本地的基零编辑方式。

注意，在一个页面中如果有超过一个表格的话，或许这不是我们想看到的结果。例如，既然

在这个表格的最后一行有一个灰蓝的背景，那 么在下一个表格的第一行将会有一个灰黄的背

景。在第 7 章我们会研究如何避免这类问题。

谈到最后一个自定义的选择吕在，让我们以某些原因假设，我们想高亮任何相关 Herry 做

的表格单元。我们要做的所有事件是加入一个类到样式表，使文本变粗体和颜色变红色

（.highlight {fontweight:

bold. color: #f00.}），并加入一行使用用 :contains() 选择器 jQuery 代码。

```
$(document).ready(function() {
$('tr:odd').addClass('odd').
$('tr:even').addClass('even').
$('td:contains("Henry")').addClass('highlight').
}).
```

所以，现在我们能够看到我们有趣的有间条的表格，并且与 Henry 相关的特显了：

诚然，有一些方法可以不用 jQuery 也能达到高亮的效果，或者任何客户端编程。然而，当

内容不是动态产生的并且我们没有访问任一 HTML 或者服务器端代码，jQuery 结合 CSS

对做这种样式是一个很好的选择。

DOM 遍历方法

到目前为止，我们研究的 jQuery 选择器允许我们遍历 DOM 树并过虑结果，然后得到一组

元素。如果这是得到元素的唯一方法，我们的选择会受到限制（尽管，老实说，在它们自己

上的选择器表达式在它们自己的方式是很强大的，特别当与正规的 DOM 脚本比较时）。

在获得一个父亲或者祖先元素时，有很多场合是基本的。并且那是 jQuery 遍历方法来操作

的地方。我们设计的这些方法，很容易遍历 DOM 树。

在选择器表达式中，一些方法几乎是相同的。例如，我们来用添加类的那一行，

$('tr:odd').addClass('odd')., 可用.filter 方法来重写如下：

$('tr').filter(':odd').addClass('odd').

在极大程度上，然而，获得元素的这两种方法是互补的，让我们再看一看表格间条的例子，

为了了解用这些方法什么是可能的。

首先，这个表格可用一个标题列，所以我们会加入另一个包含两个 <th> 的 <tr> 元素，而

不是 <td> 元素：

[…]

<tr>

<th>Title</th>

<th>Category</th>

</tr>

[…]

注意：从语义上更清楚上看，我们可以用<thead></thead>来包装标题行，并用<tbody></tbody>来包装剩下的行，但为了这个实例的目的，我们不加入额外的标记。

We'll style that heading row differently from the rest, giving it a bold yellow background color

instead of the pale blue that it would get with the code the way we left it.

我们会样式化标题行区别于剩下的行，给它一个黄色的背景色来代替灰蓝色背景。

第二，我们的客户只是浏览网站并喜欢间条，也想在 category 单元的 Henry 行里而不是在

标题单元里显示红色文本。

样式化标题行

不同地给标题行加上样式的任务可通过指定 <th> 标签和获得它们的父亲来完成。其 它行可

被选择通过结合 CSS、XPath 自定义选择器来过滤元素来设置样式

如下：

```
$(document).ready(function() {
$('th').parent().addClass('tableheading')
.
$('tr:not([th]):even').addClass('even').
$('tr:not([th]):odd').addClass('odd').
$('td:contains("Henry")').addClass('highlight').
}).
```

以标题行，即使括号中没有任何东西我们都能得到一个普通的父亲，parent()，因为我们知道它是一个 <tr> 并有且只一个。尽管我们希望这个 <tr> 加上两次 tableheading 类，因为在它里面有两个 <th> 元素，如果类已经存在，jQuery 应该自动避免加上一类名到元素上。

对于主体行，我们应用 :odd 或 :even 过滤后，我们开始排除有 <th> 作为子孙的 <tr> 。

要注意的是选择器的顺序是很重要的。如果我们用了，我们的表格看起来会很不同，例如，

$('tr:odd:not([th])') 而不是 $('tr:not([th]):odd')。

样式化 Category 单元

为了样式化这个单元后面的每一个包含 Henry 的单元，我 们可用已经可好的了选择器开始，

并简单地加入 next() 方法：

```
$(document).ready(function() {
$('the').parent().addClass('tableheading')
.
$('tr:not([th]):even').addClass('even').
$('tr:not([th]):odd').addClass('odd').
$('td:contains("Henry")').next().addClass('highlight').
}).
```

根据加上 tableheading

类和 highlight 类，现在应用样式到 category 列的单元格，这个表格看起来应该像这样：

.next() 方法得到的是下一下兄弟元素。如果有很多列我们会做什

么？如果有一个 Year

Published 列，例如，当它的行在 Title 列中包含 Henry 时，我们也想在那列中的文本高

亮显示。换句话说，对每一在它里面的包含 Henry 的单元格的行，在那行里，我们想得到

所有单元格。我们可用很多使用选择器表达式和 jQuery 方法结合的方法达到。

1.得到包含 Henry 的单元格,然后它的兄弟(不只是下一个的兄弟)。加入这个类：

```
$('td:contains("Henry")').siblings().addClass('highlight').
```

2．得到包含 Henry 的单元格，得到它的父亲，然后查找所有在它里面大于 0 的单元格 (0

是第一个单元格)，加入这个类：

```
$('td:contains("Henry")').parent().find('td:gt(0)') .addClass('highlight').
```

3．得到包含 Henry 的单元格，得到它的父亲，查找所有在它里面，然后过滤那些除了包

含 Henry 的，加入这个类：

```
$('td:contains("Henry")').parent().find('td').not(':contains("Henry")') ).addClass('highlight').
```

4．得到包含 Henry 的单元格，得到它的父亲，查找在它的孩子里面的第二个单元格，然

后加入这个类，取消上一个 .find() ，在孩子里查找第三个单元格，并加入这个类：

```
$('td:contains("Henry")').parent().find('td:eq(1)').addClass( '
highlight').end().find('td:eq(2)').addCl
```

ass('highlight').

所有这些选择都会产生相同的结果：

只是要清楚，不 是所有结合选择器表达式和方法的方法都被推荐的。

事 实上，第 四个方法是。

然而，他们应该说明了 jQuery 的 DOM 遍历选择是难以置信的灵活。

链接

所有这四个选择也说明了 jQuery 的链接能力。用 jQuery 为得到多套元素和用它们来做多

件事是完全可能的，并且所有事件都一行代码上实现。

```
$('td:contains("Henry")') //get every cell containing "Henry"
.parent() //get its parent
.find('td:eq(1)') //find inside the parent the 2nd cell
.addClass(highlight') //add the "highlight" class to that cell
.end() //revert back to the parent of the cell containing "Henry"
.find('td:eq(2)') //find inside the parent the 3rd cell
.addClass('highlight'). //add the "highlight" class to that cell
```

链接很像我们一口气说一整段话，它能迅速地完成工作，但它可能难以为别人理解，拆散它

成多行并加入清晰的注释，在长远来看，可节省更多的时间。

访问 DOM 元素

每一个选择器表达和大部分 jQuery 方法都返回一个 jQuery 对象，它几乎是总是我们想要

的，因为有了它，我们可以实现隐含式循环和链式功能。

有时我们的代码中，可能会直接访问 DOM 元素，jQuery 提供了.get()方法，要访问第一个元

素可以用.get(0)，有多个元素的话，可以用.get(index)来选择。如果我们想知道 id 为

#myelement

的元素的 tag name 的话，可以用：

```
var myTag = $('#myelement').
get(0).tagName.
```

为了更方便，jQuery 提供了.get()的速记，用 $('#myelement')[0]来代替

```
$('#myelement').
get(0).tagName
```

，方括号语法更像 DOM 中数组。

小结

本章中我们已经覆盖了这些技术，我们现在能够使用样式表的选择器来样式嵌套的 list 的高

水平与子水平项目，使用 Xpath attribute 选择器来对不同类型的超链接应用不同的样式，用

自定义的 jQuery 选择器:odd 与:even 来为 table 加入相间的条纹，使用 chaining jQuery 方法

在某个表格单元中高亮文本。

现在，我们已经使用$(document).ready()事件来加入 class 来匹配元素。在下一章中，我们将

会研究在用户新加入的各种事件的响应里加入 class 的方法。

Chapter 7 Table Manipulation

第七章 表格操作

Let 'em wear gaudy colorsOr avoid display —Devo, "Wiggly World"

In the first six chapters, we explored the jQuery library in a series of tutorials that focused on each

jQuery component and used examples as a way to see those components in action. In Chapters 7 through

9 we invert the process; we'll begin with the examples and see how we can use jQuery methods to

achieve them.

在前六章中，我们通过一系列和 jQuery 组件的有关的教程来研究了 jQuery 库，并运用了一些例子

弄清楚了这些组件是如何在实际运用中起作用的。在第 7 章到第 9 章中，我们将反过来学会如何使

用 jQuery 的方法来实现这些例子。

Here we will use an online bookstore as our model website, but the techniques we cook up can be

applied to a wide variety of other sites as well, from weblogs to portfolios, from market-facing business

sites to corporate intranets. Chapters 7 and 8 focus on two common elements of most sites—tables and

forms—while Chapter 9 examines a couple of ways to visually enhance sets of information using
animated shufflers and rotators.
这里我们会使用在线书店作为网站的例子，但是我们所使用的技术也使用于其他类型的网站，比
如博客、证券网站、市场营销网站和企业内部网站等等。第 7 章和第 8 章集中介绍了网站中的两大
基本元素——表格和表单，而第 9 章介绍了使用拖拽和旋转来增强网页上信息的视觉表现。
In this chapter, we will use jQuery to apply techniques for increasing the readability, usability, and visual
appeal of tables, though we are not dealing with tables used for layout and design. In fact, as the web
standards movement has become more pervasive in the last few years, table-based layout has
increasingly been abandoned in favor of CSSbased designs. Although tables were often employed as a
somewhat necessary stopgap measure in the 1990s to create multi-column and other complex layouts,
they were never intended to be used in that way, whereas CSS is a technology expressly created for
presentation.

在这一章中，我们会只用 jQuery 来实现增强表格的可读性、可用性和外观性的技术，但是我们并
不是把表格用来做排版和设计。实际上，由于过去几年来 Web 标准的不断发展和深入，基于表格
的布局已经渐渐退出了历史舞台，取而代之的是 CSS。尽管在 20 世纪 90 年代表格经常作为创建多
列数据和其他复杂布局的一种权宜的方法，但是并没有规定一定要那样去使用表格，反而 CSS 才
是为了页面布局而产生的一门技术。

But this is not the place for an extended discussion on the proper role of tables. Suffice it to say that in
this chapter we will explore ways to display and interact with tables used as semantically marked up
containers of tabular data. For a closer look at applying semantic, accessible HTML to tables, a good
place to start is Roger Johansson's blog entry, Bring on the Tables at http://www.456bereastreet.com/
archive/200410/bring_on_the_tables/.

但是现在不是要讨论表格应该扮演什么样的角色。在这一章中我们要研究怎样把表格作为表列数
据的标记容器来显示和交互。如果要对表格的 HTML 相关知识有进一步认识，请访问 Roger

Johansson 的博客 Bring on the Tables，地址是 http://www.456bereastreet.com/archive/200410/bring_on_the_tables/.

Some of the techniques we apply to tables in this chapter can be found in plugins such as Christian Bach's Table Sorter. For more information, visit the jQuery Plugin Repository at http://jquery.com/Plugins.

在这一章中我们用在表格上的一些技术可以在一些插件中找到，比如 Christian Bach 的 Table Sorter。如果需要更详细的信息，请访问 http://jquery.com/Plugins 上的 jQuery Plugin Repository。

Sorting

排序

One of the most common tasks performed with tabular data is sorting. In a large table, being able to rearrange the information that we're looking for is invaluable. Unfortunately, this helpful operation is one of the trickiest to put into action. We can achieve the goal of sorting in two ways, namely Server-Side Sorting and JavaScript Sorting.

表列数据中最常用到的功能就是排序。在一个庞大的表格中，对我们要寻找的数据进行排序是没有意义的。不过，排序却是实践中最有技巧性的一个操作。我们可以用两种方法来实现排序，一种是服务器端排序，另一种是 JavaScript 排序。

Server-Side Sorting

服务器端排序

A common solution for data sorting is to perform it on the server side. Data in tables often comes from a database, which means that the code that pulls it out of the database can request it in a given sort order (using, for example, the SQL language's ORDER BY clause). If we have server-side code at our disposal, it is straightforward to begin with a reasonable default sort

order。

普遍用在数据排序上的一种方法，就是把排序放在服务器端执行。表格中的数据常常来自与数据

库，这意味着把数据从数据库内提取出来的代码可以把数据按照某个顺序进行排序（比如使用

SQL 语言的 ORDER BY 子句）。如果我们使用了服务器端代码，得到的数据就会有一个默认的顺

序。

Sorting is most useful when the user can determine the sort order. A common idiom is to make the

headers of sortable columns into links. These links can go to the current page, but with a query string

appended indicating the column to sort by:

```
<table id="my-data">
<tr>
<th class="name"><a href="index.php?sort=name">Name</a></th>
<th class="date"><a href="index.php?sort=date">Date</a></th>
</tr>
...
</table>
```

在用户能够决定排序顺序的时候，排序是非常有用的。普遍的做法是给能够排序的列的页眉加上
链接。这些链接指向当前页面，但是带有排序字段作为必要的参数。

```
<table id="my-data">
<tr>
<th class="name"><a href="index.php?sort=name">Name</a></th>
<th class="date"><a href="index.php?sort=date">Date</a></th>
</tr>
...
</table>
```

The server can react to the query string parameter by returning the database contents in a different order.
服务器会根据查询字符串的参数返回对应排序的数据库内容。

Preventing Page Refreshes
阻止页面刷新

This setup is simple, but requires a page refresh for each sort operation. As we have seen, jQuery allows
us to eliminate such page refreshes by using AJAX methods. If we have the column headers set up as
links as before, we can add jQuery code to change those links into

AJAX requests:

```
$(document).ready(function() {
$('#my-data .name a').click(function() {
$('#my-data').load('index.php?sort=name&type=ajax');
return false;
});
```

```
$('#my-data .date a').click(function() {
$('#my-data').load('index.php?sort=date&type=ajax');
return false;
});
});
```

上面的方法很简单，但是每次排序都会刷新页面。不过我们已经知道，jQuery 允许我们使用
AJAX 来不进行页面的刷新。如果我们已经把页眉设置成了排序链接，就可以使用 jQuery 代码把
这些链接做成 AJAX 的形式。

```
$(document).ready(function() {
$('#my-data .name a').click(function() {
$('#my-data').load('index.php?sort=name&type=ajax');
return false;
});
$('#my-data .date a').click(function() {
$('#my-data').load('index.php?sort=date&type=ajax');
return false;
});
});
```

Now when the anchors are clicked, jQuery sends an AJAX request

to the server for the same page. We
add an additional parameter to the query string so that the server can determine that an AJAX request is
being made. The server code can be written to send back only the table itself, and not the surrounding
page, when this parameter is present. This way we can take the response and insert it in place of the
table.

现在点击那些链接时，jQuery 就会发送 AJAX 请求到服务器上。我们给查询字符串增加了一个参
数，这样服务器就能够判断是否有 AJAX 请求。当设置了这个参数的时候，服务器的代码可以只
更新表格自身，而不会刷新整个页面。这样我们就能得到服务器返回的数据，把表格内原有的数
据更新。

This is an example of progressive enhancement. The page works perfectly well without any JavaScript
at all, as the links for server-side sorting are still present. When JavaScript is present, however, the
AJAX hijacks the page request and allows the sort to occur without a full page load.

这是一个循序渐进的例子。在没有使用 JavaScript 的时候，页面的

服务器端排序功能的表现很

好。引入了 JavaScript 后，AJAX 拦截了页面的请求，没有刷新整个

页面就完成了排序的操作。

JavaScript Sorting

JavaScript 排序

There are times, though, when we either don't want to wait for server responses when sorting, or don't

have a server-side scripting language available to us. A viable alternative in this case is to perform the

sorting entirely on the browser using JavaScript client-side scripting.

有些时候，我们既不想在排序的时候服务器端有响应，也不想执行任何服务器端的代码。这种情

况下可靠的方法就是使用 JavaScript 客户端程序在浏览器内进行排序。

For example, suppose we have a table listing books, along with their authors, release dates, and prices:

```
<table class="sortable">
<thead>
<tr>
<th></th>
<th>Title</th>
<th>Author(s)</th>
```

```
<th>Publish Date</th>

<th>Price</th>

</tr>

</thead>
```

```
<tbody>
<tr>
<td>
<img src="../covers/small/1847192386.png" width="49"
height="61" alt="Building Websites with
</td>
<td>Building Websites with Joomla! 1.5 Beta 1</td>
<td>Hagen Graf</td>
<td>Feb 2007</td>
<td>$40.49</td>
</tr>
<tr>
<td><img src="../covers/small/1904811620.png" width="49"
height="61" alt="Learning Mambo: A Step-by-Step
Tutorial to Building Your Website" /></td>
<td>Learning Mambo: A Step-by-Step Tutorial to Building Your
Website</td>
<td>Douglas Paterson</td>
<td>Dec 2006</td>
<td>$40.49</td>
</tr>
```

...

</tbody>

</table>

比如，我们有一个表格列出了一些书的内容，还有他们的作者、出版日期和价格。

<table class="sortable">

<thead>

<tr>

<th></th>

<th>Title</th>

<th>Author(s)</th>

<th>Publish Date</th>

<th>Price</th>

</tr>

</thead>

<tbody>

<tr>

<td>

<img src="../covers/small/1847192386.png" width="49"

height="61" alt="Building Websites with

</td>

<td>Building Websites with Joomla! 1.5 Beta 1</td>

<td>Hagen Graf</td>

<td>Feb 2007</td>

<td>$40.49</td>

</tr>

<tr>

<td><img src="../covers/small/1904811620.png" width="49"

height="61" alt="Learning Mambo: A Step-by-Step

Tutorial to Building Your Website" /></td>

<td>Learning Mambo: A Step-by-Step Tutorial to Building Your

Website</td>

<td>Douglas Paterson</td>

<td>Dec 2006</td>

<td>$40.49</td>

</tr>

...

</tbody>

</table>

We'd like to turn the table headers into buttons that sort by their respective columns. Let us look into

ways of doing this.

我们要把表格的页眉部分变成排序的按钮，下面让我们来看看实现的方法。

Row Grouping Tags

行分组标签

Note our use of the <thead> and <tbody> tags to segment the data into row groupings. Many HTML

authors omit these implied tags, but they can prove useful in supplying us with more convenient CSS

selectors to use. For example, suppose we wish to apply typical even/odd row striping to this table, but

only to the body of the table:

```
$(document).ready(function() {
$('table.sortable tbody tr:odd').addClass('odd');
$('table.sortable tbody tr:even').addClass('even');
```

});

请注意<thead>和<tbody>的使用把数据进行了行分组。很多人在写 HTML 的时候忽略了这些标

签，但是它们能够提供我们更加便捷的 CSS 选择符来使用。例如，我们要让一个表格内的奇数、

偶数行的背景色不同，之需要修改表格体的部分。

```
$(document).ready(function() {
$('table.sortable tbody tr:odd').addClass('odd');
$('table.sortable tbody tr:even').addClass('even');
});
```

This will add alternating colors to the table, but leave the header untouched:

这样就让表格的行与行之间的背景色有所不同，但是并没有修改表格的页眉。

Basic Alphabetical Sorting

基本的字母顺序排序

Now let's perform a sort on the Title column of the table. We'll need a class on the table header cell so

that we can select it properly:

```
<thead>
<tr>
<th></th>
<th class="sort-alpha">Title</th>
<th>Author(s)</th>
<th>Publish&amp;nbsp;Date</th>
<th>Price</th>
</tr>
</thead>
```

现在我们给表格内的 Title 列加上排序功能。我们要给页眉的 Title 单元格加上一个 class 属性，这样

就方便我们找到它。

```
<thead>
<tr>
<th></th>
<th class="sort-alpha">Title</th>
```

<th>Author(s)</th>

<th>Publish Date</th>

<th>Price</th>

</tr>

</thead>

To perform the actual sort, we can use JavaScript's built in .sort() method. It does an inplace sort on an

array, and can take a function as an argument. This function compares two items in the array and should

return a positive or negative number depending on the result. Our initial sort routine looks like this:

```
$(document).ready(function() {
$('table.sortable').each(function() {
var $table = $(this);
$('th', $table).each(function(column) {
if ($(this).is('.sort-alpha')) {
$(this).addClass('clickable').hover(function() {
$(this).addClass('hover');
```

```
}, function() {
$(this).removeClass('hover');
}).click(function() {
var rows = $table.find('tbody > tr').get();
rows.sort(function(a, b) {
var keyA =
$(a).children('td').eq(column).text() .toUpperCase();
var keyB =
$(b).children('td').eq(column).text() .toUpperCase();
if (keyA < keyB) return -1;
if (keyA > keyB) return 1;
return 0;
});
$.each(rows, function(index, row) {
$table.children('tbody').append(row);
});
});
}
});
});
});
```

为了实现排序，我们可以使用 JavaScript 内建的.sort()方法。它对一个数组执行操作，并可以使用
函数作为参数。这个函数比较数组中的两个元素，并根据结果会返回一个正数或是负数。我们最
初的排序方法如下：

```
$(document).ready(function() {
$('table.sortable').each(function() {
var $table = $(this);
$('th', $table).each(function(column) {
if ($(this).is('.sort-alpha')) {
$(this).addClass('clickable').hover(function() {
$(this).addClass('hover');
}, function() {
$(this).removeClass('hover');
}).click(function() {
var rows = $table.find('tbody > tr').get();
rows.sort(function(a, b) {
var keyA =
$(a).children('td').eq(column).text() .toUpperCase();
var keyB =
$(b).children('td').eq(column).text() .toUpperCase();
if (keyA < keyB) return -1;
```

```
if (keyA > keyB) return 1;
return 0;
});
$.each(rows, function(index, row) {
$table.children('tbody').append(row);
});
});
}
});
});
});
```

The first thing to note is our use of the .each() method to make iteration explicit. Even though we could bind a click handler to all headers with the sort-alpha class just by calling $('table.sortable th.sortalpha').click(), this wouldn't allow us to easily capture a crucial bit of information—the column index of the clicked header. Because .each() passes the iteration index into its callback function, we can use it to find the relevant cell in each row of the data later.

首先要注意的是我们使用了.each()方法做了迭代循环。我们可以使用$('table.sortable th.sort-

alpha').click()给所有使用了 sort-alpha class 的页眉绑定 click 事件，但是我们却不能轻易地得到那最
重要的信息——被点击的页眉的索引。因为.each()把迭代的索引传给它的回调函数，我们可以用
它来找到每一行对应的单元格。

Once we have found the header cell, we retrieve an array of all of the data rows. This is a great example
of how .get() is useful in transforming a jQuery object into an array of DOM nodes; even though jQuery
objects act like arrays in many respects, they don't have any of the native array methods available, such
as .sort().

一旦我们找到了页眉的单元格，我们就重新得到了一个所有数据行所组成的数组。这就是.get()
在把 jQuery 对象转换成 DOM 节点的数组所起到的作用的一个例子，尽管在许多方面 jQuery 对象可
以像数组一样工作，但是 jQuery 对象并没有数组的原生方法可以使用，比如.sort()。

With .sort() at our disposal, the rest is fairly straightforward. The rows are sorted by comparing the
textual contexts of the relevant table cell. We know which cell to look

at because we captured the

column index in the enclosing .each() call. We convert the text to uppercase because string comparisons

in JavaScript are case-sensitive and we wish our sort to be case-insensitive. Finally, with the array

sorted, we loop through the rows and reinsert them into the table. Since .append() does not clone nodes,

this moves them rather than copying them. Our table is now sorted.

使用了.sort()之后，剩下的工作就直截了当了。所有数据行通过比较相关的单元格内的文本内容

进行排序。我们知道是哪个单元格作为排序条件，因为我们在.each()的调用中得到了列的索引。

由于 JavaScript 是大小写敏感的语言所以我们在比较时把文本就转换成了大写，而且我们希望我

们的排序是大小写不敏感的。最终，数据行的数组排序完成之后，通过循环重新在表格上一行一

行地显示出来。由于.append()方法不会复制数组的 DOM 节点，执行之后数组内的节点就会被移

除。这样一来，表格就完成了排序。

This is an example of progressive enhancement's counterpart, graceful degradation. Unlike with the

AJAX solution discussed earlier, we cannot make the sort work without

JavaScript, as we are assuming
the server has no scripting language available to it in this case. The JavaScript is required for the sort to
work, so by adding the "clickable" class only through code, we make sure not to indicate with the
interface that sorting is even possible unless the script can run. The page degrades into one that is still
functional, albeit without sorting available.

这是另外一个循序渐进的例子——功能退化。和前面谈到的 AJAX 解决方式不同，没有 JavaScript
的情况下我们没有办法实现排序，因为在这种情况服务器端没有执行任何语句。要实现排序就必
须使用 JavaScript，所以通过在代码中添加 clickable 的 class 属性，只有在脚本能够运行的情况下，
我们才能确定从界面上确定排序是否有效。否则页面就成了只有显示功能而没有排序功能的东
西。

We have moved the actual rows around, hence our alternating row colors are now out of whack:

我们把数据行进行了排序，但是确把原有的交错行的背景色给弄乱了。

We need to reapply the row colors after the sort is performed. We can do this by pulling the coloring

code out into a function that we call when needed:

```
$(document).ready(function() {
var alternateRowColors = function($table) {
$('tbody tr:odd', $table).removeClass('even').addClass('odd');
$('tbody tr:even', $table).removeClass('odd').addClass('even');
};
$('table.sortable').each(function() {
var $table = $(this);
alternateRowColors($table);
$('th', $table).each(function(column) {
if ($(this).is('.sort-alpha')) {
$(this).addClass('clickable').hover(function() {
$(this).addClass('hover');
}, function() {
```

```
$(this).removeClass('hover');
}).click(function() {
var rows = $table.find('tbody > tr').get();
rows.sort(function(a, b) {
var keyA =
$(a).children('td').eq(column).text() .toUpperCase();
var keyB =
$(b).children('td').eq(column).text() .toUpperCase();
if (keyA < keyB) return -1;
if (keyA > keyB) return 1;
return 0;
});
$.each(rows, function(index, row) {
$table.children('tbody').append(row);
});
alternateRowColors($table);
});
}
});
});
});
```

在排序完成之后，我们必须重新设置每行的背景颜色。我们可以把设置颜色的代码写在函数里，

在需要的时候调用。

```
$(document).ready(function() {
var alternateRowColors = function($table) {
$('tbody tr:odd', $table).removeClass('even').addClass('odd');
$('tbody tr:even', $table).removeClass('odd').addClass('even');
};
$('table.sortable').each(function() {
var $table = $(this);
alternateRowColors($table);
$('th', $table).each(function(column) {
if ($(this).is('.sort-alpha')) {
$(this).addClass('clickable').hover(function() {
$(this).addClass('hover');
}, function() {
$(this).removeClass('hover');
}).click(function() {
var rows = $table.find('tbody > tr').get();
rows.sort(function(a, b) {
var keyA =
$(a).children('td').eq(column).text() .toUpperCase();
```

```
var keyB =
$(b).children('td').eq(column).text() .toUpperCase();
if (keyA < keyB) return -1;
if (keyA > keyB) return 1;
return 0;
});
$.each(rows, function(index, row) {
$table.children('tbody').append(row);
});
alternateRowColors($table);
});
}
});
});
});
```

This corrects the row coloring after the fact, fixing our issue:

刚才的代码修正了每行的背景颜色，解决了刚才的问题。

The Power of Plug-ins

插件的力量

The alternateRowColors() function that we wrote is a perfect candidate to become a jQuery plug-in. In fact, any operation that we wish to apply to a set of DOM elements can easily be expressed as a plug-in.

In this case, we only need to modify our existing function a little bit:

```
jQuery.fn.alternateRowColors = function() {
$('tbody tr:odd', this).removeClass('even').addClass('odd');
$('tbody tr:even', this).removeClass('odd').addClass('even');
return this;
};
```

上面我们所写的 alternateRowColors()函数 jQuery 插件的一个非常完美的候选程序。实际上，任何
运用于 DOM 元素上的操作都可以写成 jQuery 插件的形式。这样一来，我们只需要稍微修改一下
原来的程序。

```
jQuery.fn.alternateRowColors = function() {
$('tbody tr:odd', this).removeClass('even').addClass('odd');
$('tbody tr:even', this).removeClass('odd').addClass('even');
return this;
};
```

We have made three important changes to the function.

It is defined as a new property of jQuery.fn rather than as a standalone function. This registers the

function as a plug-in method.

We use the keyword this as a replacement for our $table parameter.

Within a plug-in method, this

refers to the jQuery object that is being acted upon.

Finally, we return this at the end of the function. The return value makes our new method chainable.

这个函数上我们作了三处重要的修改：

相比于原来的独立的函数，我们重新把这个函数定义成 jQuery.fn 的一个属性。这样做就把原

函数注册成为一个插件函数。

我们使用了 this 关键字来替换原来的$table 参数。在 jQuery 插件方法里，这种做法会直接引用

当前正在操作的 jQuery 对象。

最后，在函数结尾，我们返回了 this。这个返回值使得我们的新方法成为了链式方法。

More information on writing jQuery plug-ins can be found in Chapter 10. There we will discuss making a plug-in ready for public consumption, as opposed to the small example here that is only to be used by our own code.

在第 10 章中可以找到有关撰写 jQuery 插件的更多的内容。在那个部分我们会探讨怎样把一个插件做成公用程序，而不像现在这样只是适用于我们自己的代码。

With our new plug-in defined, we can call $table.alternateRowColors(), which is a more natural jQuery syntax, intead of alternateRowColors($table).

定义了插件之后，我们可以通过调用$table.alternateRowColors()这样一个更加有着 jQuery 语法风格的方法，来替换原来的 alternateRowColors($table)。

Performance Concerns

关注性能

Our code works, but is quite slow. The culprit is the comparator function, which is performing a fair amount of work. This comparator will be called many times during the course of a sort, which means

that every extra moment it spends on processing will be magnified.

现在我们的代码可以运行，但是效率很低。罪魁祸首就是比较函数，它的开销太大了。在排序的

过程中，比较函数被调用了很多次，这就是说排序过程中比较函数每次进行处理都占用了额外的

资源。

The actual sort algorithm used by JavaScript is not defined by the standard. It may be a simple sort like a

bubble sort (worst case of $\Theta(n2)$ in computational complexity terms) or a more sophisticated approach

like quick sort (which is $\Theta(n \log n)$ on average). In either case doubling the number of items increases

the number of times the comparator function is called by more than double.

JavaScript 中所使用的实际排序算法并没有在标准中定义。它可能是一个像冒泡排序一样的简单

算法（算法复杂性$\Theta(n2)$）；或者是像快速排序那样复杂的算法（算法复杂性$\Theta(n \log n)$）。不管

哪种情况，增加一倍的数据量，在排序过程中比较函数的调用次数增加远远不止一倍。

The remedy for our slow comparator is to pre-compute the keys for the comparison. We begin with the

slow sort function:

```
rows.sort(function(a, b) {

keyA = $(a).children('td').eq(column).text().toUpperCase();

keyB = $(b).children('td').eq(column).text().toUpperCase();

if (keyA < keyB) return -1;

if (keyA > keyB) return 1;
```

```
return 0;
});
$.each(rows, function(index, row) {
$table.children('tbody').append(row);
});
```

为了修正比较函数执行效率的问题，我们的办法是在比较之前事先处理数组的排序的关键字。先从原来的函数入手。

```
rows.sort(function(a, b) {
keyA = $(a).children('td').eq(column).text().toUpperCase();
keyB = $(b).children('td').eq(column).text().toUpperCase();
if (keyA < keyB) return -1;
if (keyA > keyB) return 1;
return 0;
});
$.each(rows, function(index, row) {
$table.children('tbody').append(row);
});
```

We can pull out the key computation and do that in a separate loop:

```
$.each(rows, function(index, row) {
row.sortKey =
```

```
$(row).children('td').eq(column).text().toUpperCase();
});
rows.sort(function(a, b) {
if (a.sortKey < b.sortKey) return -1;
if (a.sortKey > b.sortKey) return 1;
return 0;
});
$.each(rows, function(index, row) {
$table.children('tbody').append(row);
row.sortKey = null;
});
```

我们可以把对关键字的处理放在一个单独的循环中。

```
$.each(rows, function(index, row) {
row.sortKey                                                    =
$(row).children('td').eq(column).text().toUpperCase();
});
rows.sort(function(a, b) {
if (a.sortKey < b.sortKey) return -1;
if (a.sortKey > b.sortKey) return 1;
return 0;
});
$.each(rows, function(index, row) {
```

```
$table.children('tbody').append(row);
row.sortKey = null;
});
```

In the new loop, we are doing all of the expensive work and storing the result in a new property. This kind of property, attached to a DOM element but not a normal DOM attribute, is called an expando. This is a convenient place to store the key since we need one per table row element. Now we can examine this attribute within the comparator function, and our sort is markedly faster.

在这个新的循环中，我们处理了所有繁重的任务并把结果保存在一个新的属性中。这个新属性被称为扩展属性，它被绑定到 DOM 元素而不是普通的 DOM 属性。这是一个非常方便的用于保存关键字的东西，表格的每一个行元素都需要用到。现在我们可以在比较函数中用上这个属性，新的排序明显快多了。

在完成了比较之后我们把扩展属性设置成了 null。在这种情况下并不是非要这么做不可，但是这
是一个编程的好习惯，因为如果不去管扩展属性的话，可能会造成内存溢出。如果要了解更多的
内容，请参阅附录 C。

Finessing the Sort Keys

优化排序关键字

Now we want to apply the same kind of sorting behavior to the Author(s) column of our table. By
adding the sort-alpha class to its table header cell, the Author(s) column can be sorted with our existing
code. But ideally authors should be sorted by last name, not first. Since some books have multiple
authors, and some authors have middle names or initials listed, we need outside guidance to determine
what part of the text to use as our sort key. We can supply this guidance by wrapping the relevant part of
the cell in a tag:

```
<tr>
<td>
<img src="../covers/small/1847192386.png" width="49"
```

```
height="61"
alt="Building Websites with Joomla! 1.5 Beta 1" /></td>
<td>Building Websites with Joomla! 1.5 Beta 1</td>
<td>Hagen <span class="sort-key">Graf</span></td>
<td>Feb 2007</td>
<td>$40.49</td>
</tr>
<tr>
<td>
<img src="../covers/small/1904811620.png" width="49"
height="61"
alt="Learning Mambo: A Step-by-Step Tutorial to Building
Your Website" /></td>
<td>
Learning Mambo: A Step-by-Step Tutorial to Building Your Website
</td>
<td>Douglas <span class="sort-key">Paterson</span></td>
<td>Dec 2006</td>
<td>$40.49</td>
</tr>
<tr>
<td>
```

<img src="../covers/small/1904811299.png" width="49" height="61"
alt="Moodle E-Learning Course Development" /></td>
<td>Moodle E-Learning Course Development</td>
<td>William Rice</td>
<td>May 2006</td>
<td>$35.99</td>
</tr>

现在我们需要给表格中的 Author(s)也加上同样的排序功能。给 Author(s)的页眉单元格加上 sortalpha
的 class 属性之后，在我们现有代码的作用下，Author(s)也具有了排序功能。但是理想情况下，
作者的名字应该是根据姓而不是名来进行排序的。而且有些书有多个作者，有些作者有中间名或
是名字的字母缩写，我们需要根据生活常规来决定哪些问题来作为排序关键字。根据常规，我们
用标签把单元格内用于排序的部分独立出来。

```
<tr>
<td>
<img src="../covers/small/1847192386.png" width="49"
height="61"
alt="Building Websites with Joomla! 1.5 Beta 1" /></td>
<td>Building Websites with Joomla! 1.5 Beta 1</td>
<td>Hagen <span class="sort-key">Graf</span></td>
<td>Feb 2007</td>
<td>$40.49</td>
</tr>
<tr>
<td>
<img src="../covers/small/1904811620.png" width="49"
height="61"
alt="Learning Mambo: A Step-by-Step Tutorial to Building
Your Website" /></td>
<td>
Learning Mambo: A Step-by-Step Tutorial to Building Your Website
</td>
<td>Douglas <span class="sort-key">Paterson</span></td>
<td>Dec 2006</td>
```

```
<td>$40.49</td>
</tr>
<tr>
<td>
<img src="../covers/small/1904811299.png" width="49"
height="61"
alt="Moodle E-Learning Course Development" /></td>
<td>Moodle E-Learning Course Development</td>
<td>William <span class="sort-key">Rice</span></td>
<td>May 2006</td>
<td>$35.99</td>
</tr>
```

Now we have to modify our sorting code to take this tag into account, without disturbing the existing behavior for the Title column, which is working well. By prepending the marked sort key to the key we have previously calculated, we can sort first on the last name if it is called out, but on the whole string as a fallback:

```
$.each(rows, function(index, row) {
var $cell = $(row).children('td').eq(column);
row.sortKey = $cell.find('.sort-key').text().toUpperCase() + '
```

' + $cell.text().toUpperCase();

});

现在，考虑到标签的引入，我们需要修改排序的代码，而且不修改现在的 Title 的排序功能。事先

考虑到把标记的排序关键字作为之前已经处理过的关键字，我们在调用 Author(s)排序的时候，

把姓作为排序条件，然后在返回结果中，需要将全部的名字都显示出来。

$.each(rows, function(index, row) {

var $cell = $(row).children('td').eq(column);

row.sortKey = $cell.find('.sort-key').text().toUpperCase() + '

' + $cell.text().toUpperCase();

});

Sorting by the Author(s) column now uses the last name:

现在 Author(s)这一列是根据姓的顺序来进行排序的。

If two last names are identical, the sort uses the entire string as a tiebreaker for positioning.

如果有两条数据中的姓是一样的，那么排序的时候就会根据整个姓名来确定顺序。

Sorting Other Types of Data

给其他类型的数据排序

Our sort routine should be able to handle not just the Title and Author columns, but the Publish Dates

and Price as well. Since we streamlined our comparator function, it can handle all kinds of data, but the

computed keys will need to be adjusted for other data types. For example, in the case of prices we need

to strip off the leading $ character and parse the rest, then compare them:

```
var key = parseFloat($cell.text().replace(/^[^\d.]*/, ''));
row.sortKey = isNaN(key) ? 0 : key;
```

我们的排序不应该只处理 Title 和 Author 这两列，还应该处理 Publish Dates 和 Price。由于我们改

进了比较函数，它可以处理所有类型的数据，只是处理时的关键字需要根据其他的数据类型进行

修改。例如，在根据价格进行排序的时候，我们要把前面的$去掉，分析剩下的数据并进行比
较。

```
var key = parseFloat($cell.text().replace(/^[^\d.]*/, ''));
row.sortKey = isNaN(key) ? 0 : key;
```

The result of parseFloat() needs to be checked, because if no number can be extracted from the text,

NaN is returned, which can wreak havoc on .sort(). For the date cells, we can use the JavaScript Date

object:

```
row.sortKey = Date.parse('1 ' + $cell.text());
```

函数 parseFloat()的结果需要检验一下，因为从文本中没有取出任何数字的话，会返回 isNaN 这个

在排序中造成严重问题的数据。对于日期的单元格，我们可以使用 JavaScript 的 Date 对象。

```
row.sortKey = Date.parse('1 ' + $cell.text());
```

The dates in this table contain a month and year only; Date.parse() requires a fullyspecified date, so we

prepend the string with 1. This provides a day to complement the month and year, and the combination is

then converted into a timestamp, which can be sorted using our normal comparator.

表格中的日期只包含了月份和年份；而 Date.parse()要求参数是一个完整格式的日期，所以我们给

日期前面加了个 1。这就给月份和年份添加了一个天数的数据，而这个日期会被转换成一个时间

戳的格式，然后能够用于我们的比较函数进行排序。

We can apportion these expressions across separate functions, and call the appropriate one based on the

class applied to the table header：

```
$.fn.alternateRowColors = function() {
$('tbody tr:odd', this).removeClass('even').addClass('odd');
$('tbody tr:even', this).removeClass('odd').addClass('even');
return this;
};
$(document).ready(function() {
var alternateRowColors = function($table) {
$('tbody tr:odd', $table).removeClass('even').addClass('odd');
$('tbody tr:even', $table).removeClass('odd').addClass('even');
};
$('table.sortable').each(function() {
var $table = $(this);
$table.alternateRowColors($table);
$('th', $table).each(function(column) {
var findSortKey;
if ($(this).is('.sort-alpha')) {
```

```
findSortKey = function($cell) {
return $cell.find('.sort-key').text().toUpperCase() + ' ' +
$cell.text().toUpperCase();
};
}
else if ($(this).is('.sort-numeric')) {
findSortKey = function($cell) {
var key = parseFloat($cell.text().replace(/^[^\d.]*/, ''));
return isNaN(key) ? 0 : key;
};
}
else if ($(this).is('.sort-date')) {
findSortKey = function($cell) {
return Date.parse('1 ' + $cell.text());
};
}
if (findSortKey) {
$(this).addClass('clickable').hover(function() {
$(this).addClass('hover');
```

```
}, function() {
$(this).removeClass('hover');
}).click(function() {
var rows = $table.find('tbody > tr').get();
$.each(rows, function(index, row) {
row.sortKey = findSortKey($(row).children('td').eq(column));
});
rows.sort(function(a, b) {
if (a.sortKey < b.sortKey) return -1;
if (a.sortKey > b.sortKey) return 1;
return 0;
});
$.each(rows, function(index, row) {
$table.children('tbody').append(row);
row.sortKey = null;
});
$table.alternateRowColors($table);
});
}
});
});
```

```
});
```

我们可以用独立的函数来执行这些表达式，然后通过页眉中的 class **属性来调用不同的函数。**

```
$.fn.alternateRowColors = function() {
$('tbody tr:odd', this).removeClass('even').addClass('odd');
$('tbody tr:even', this).removeClass('odd').addClass('even');
return this;
};
$(document).ready(function() {
var alternateRowColors = function($table) {
$('tbody tr:odd', $table).removeClass('even').addClass('odd');
$('tbody tr:even', $table).removeClass('odd').addClass('even');
};
$('table.sortable').each(function() {
var $table = $(this);
$table.alternateRowColors($table);
$('th', $table).each(function(column) {
var findSortKey;
if ($(this).is('.sort-alpha')) {
findSortKey = function($cell) {
return $cell.find('.sort-key').text().toUpperCase() + ' ' +
$cell.text().toUpperCase();
```

```
};
}
else if ($(this).is('.sort-numeric')) {
findSortKey = function($cell) {
var key = parseFloat($cell.text().replace(/^[^\d.]*/, ''));
return isNaN(key) ? 0 : key;
};
}
else if ($(this).is('.sort-date')) {
findSortKey = function($cell) {
return Date.parse('1 ' + $cell.text());
};
}
```

```
if (findSortKey) {
$(this).addClass('clickable').hover(function() {
$(this).addClass('hover');
}, function() {
$(this).removeClass('hover');
}).click(function() {
var rows = $table.find('tbody > tr').get();
$.each(rows, function(index, row) {
row.sortKey = findSortKey($(row).children('td').eq(column));
});
rows.sort(function(a, b) {
if (a.sortKey < b.sortKey) return -1;
if (a.sortKey > b.sortKey) return 1;
return 0;
});
$.each(rows, function(index, row) {
$table.children('tbody').append(row);
row.sortKey = null;
});
$table.alternateRowColors($table);
});
```

```
}
});
});
});
```

The findSortKey variable doubles as the function to calculate the key and a flag to indicate whether the
column header is marked with a class making it sortable. We can now sort on date or price:

变量 findSortKey 扮演了两种角色，一种是处理排序关键字的函数，一种是页眉单元格能否进行排
序的标记。现在我们可以根据日期和价格来进行排序。

Column Highlighting

列的高亮

It can be a nice user interface enhancement to visually remind the user of what has been done in the past.

By highlighting the column that was most recently used for sorting, we can focus the user's attention on

the part of the table that is most likely to be relevant. Fortunately, since we've already determined how to

select the table cells in the column, applying a class to those cells is simple:

```
$table.find('td').removeClass('sorted')
. filter(':nth-child(' + (column + 1) + ')').addClass('sorted');
```

在页面上显示出什么内容发生了改变是一个非常好的用户界面的作法。将最近一次进行排序的列

进行高亮显示，我们能够将用户的注意力放到表格中和这个列相关的部分。很幸运的，因为我们

已经确定如何选出表格中列的单元格，给这些单元格加上一个 class 属性是非常容易的事情。

```
$table.find('td').removeClass('sorted')
. filter(':nth-child(' + (column + 1) + ')').addClass('sorted');
```

Note that we have to add one to the column index we found earlier,

since the :nth-child() selector is onebased

rather than zero-based. With this code in place, we get a highlighted

column after any sort

operation:

请注意，我们给之前找到的列索引进行了加 1 操作，因为：nth-child()
是从 1 开始计数而不是从 0 开始
计数的。加入这段代码之后，在每次排序操作之后，对应的那一列就
会变成高亮显示。

Alternating Sort Directions

降序和升序的切换

Our final sorting enhancement is to allow for both ascending and descending sort orders. When the user
clicks on a column that is already sorted, we want to reverse the current sort order.

我们最后对排序的改进是加入能够升序和降序的排序。当用户点击一个已经排序的列的时候，我
们要把当前的排序顺序给反转过来。

To reverse a sort, all we have to do is to invert the values returned by our comparator. We can do this
with a simple variable：

```
if (a.sortKey < b.sortKey) return -newDirection;
if (a.sortKey > b.sortKey) return newDirection;
```

为了让排序能够反转顺序，我们要做的就是对比较函数的返回值进行取反。可以用一个简单的变
量来实现这个功能。

```
if (a.sortKey < b.sortKey) return -newDirection;
if (a.sortKey > b.sortKey) return newDirection;
```

If newDirection equals 1, then the sort will be the same as before. If it equals -1, the sort will be reversed. We can use classes to keep track of the current sort order of a column:

```
$.fn.alternateRowColors = function() {
$('tbody tr:odd', this).removeClass('even').addClass('odd');
$('tbody tr:even', this).removeClass('odd').addClass('even');
return this;
};
$(document).ready(function() {
var alternateRowColors = function($table) {
$('tbody tr:odd', $table).removeClass('even').addClass('odd');
$('tbody tr:even', $table).removeClass('odd').addClass('even');
};
$('table.sortable').each(function() {
var $table = $(this);
$table.alternateRowColors($table);
$('th', $table).each(function(column) {
var findSortKey;
```

```
if ($(this).is('.sort-alpha')) {
findSortKey = function($cell) {
return $cell.find('.sort-key').text().toUpperCase()
+ ' ' +
$cell.text().toUpperCase();
};
}
else if ($(this).is('.sort-numeric')) {
findSortKey = function($cell) {
var key = parseFloat($cell.text().replace(/^[^\d.]*/, ''));
return isNaN(key) ? 0 : key;
};
}
else if ($(this).is('.sort-date')) {
findSortKey = function($cell) {
return Date.parse('1 ' + $cell.text());
};
}
if (findSortKey) {
$(this).addClass('clickable').hover(function() {
$(this).addClass('hover');
}, function() {
```

```
$(this).removeClass('hover');
}).click(function() {
var newDirection = 1;
if ($(this).is('.sorted-asc')) {
newDirection = -1;
}
var rows = $table.find('tbody > tr').get();
$.each(rows, function(index, row) {
row.sortKey = findSortKey($(row).children('td').eq(column));
});
rows.sort(function(a, b) {
if (a.sortKey < b.sortKey) return -newDirection;
if (a.sortKey > b.sortKey) return newDirection;
return 0;
```

```
});
$.each(rows, function(index, row) {
$table.children('tbody').append(row);
row.sortKey = null;
});
$table.find('th').removeClass('sorted-asc').removeClass('sorte
d-desc');
var $sortHead = $table.find('th').filter(':nth-child(' + (column +
1) + ')');
if (newDirection == 1) {
$sortHead.addClass('sorted-asc');
} else {
$sortHead.addClass('sorted-desc');
}
$table.find('td').removeClass('sorted').filter(':nth-child('     +
(column + 1) +
')') .addClass('sorted');
$table.alternateRowColors($table);
});
}
});
```

```
});
});
```

如果 newDirection 等于 1，排序顺序和以前一样，如果等于-1，那么排序顺序就会反转。我们给当前列使用 class 属性来追踪不同的排序顺序。

```
$.fn.alternateRowColors = function() {
$('tbody tr:odd', this).removeClass('even').addClass('odd');
$('tbody tr:even', this).removeClass('odd').addClass('even');
return this;
};
$(document).ready(function() {
var alternateRowColors = function($table) {
$('tbody tr:odd', $table).removeClass('even').addClass('odd');
$('tbody tr:even', $table).removeClass('odd').addClass('even');
};
$('table.sortable').each(function() {
var $table = $(this);
$table.alternateRowColors($table);
$('th', $table).each(function(column) {
var findSortKey;
if ($(this).is('.sort-alpha')) {
findSortKey = function($cell) {
```

```
return $cell.find('.sort-key').text().toUpperCase()
+ ' ' +
$cell.text().toUpperCase();
};
}
else if ($(this).is('.sort-numeric')) {
findSortKey = function($cell) {
var key = parseFloat($cell.text().replace(/^[^\d.]*/, ''));
return isNaN(key) ? 0 : key;
};
}
else if ($(this).is('.sort-date')) {
findSortKey = function($cell) {
return Date.parse('1 ' + $cell.text());
};
}
if (findSortKey) {
```

```
$(this).addClass('clickable').hover(function() {
$(this).addClass('hover');
}, function() {
$(this).removeClass('hover');
}).click(function() {
var newDirection = 1;
if ($(this).is('.sorted-asc')) {
newDirection = -1;
}
var rows = $table.find('tbody > tr').get();
$.each(rows, function(index, row) {
row.sortKey = findSortKey($(row).children('td').eq(column));
});
rows.sort(function(a, b) {
if (a.sortKey < b.sortKey) return -newDirection;
if (a.sortKey > b.sortKey) return newDirection;
return 0;
});
$.each(rows, function(index, row) {
$table.children('tbody').append(row);
row.sortKey = null;
```

```
});
$table.find('th').removeClass('sorted-asc').removeClass('sorte
d-desc');
var $sortHead = $table.find('th').filter(':nth-child(' + (column +
1) + ')');
if (newDirection == 1) {
$sortHead.addClass('sorted-asc');
} else {
$sortHead.addClass('sorted-desc');
}
$table.find('td').removeClass('sorted').filter(':nth-child('     +
(column + 1) +
')') .addClass('sorted');
$table.alternateRowColors($table);
});
}
});
});
});
```

As a side benefit, since we use classes to store the sort direction we can style the columns headers to indicate the current order as well:

另一个方面的好处就是，因为我们使用了 class 属性来保存排序顺序，我们就可以用列的页眉的样

式来表示当前的排序顺序。

Pagination

分页

Sorting is a great way to wade through a large amount of data to find information. We can also help the

user focus on a portion of a large data set by paginating the data. Pagination can be done in two ways—

Server-Side Pagination and JavaScript Pagination.

排序是在一大堆数据中寻找数据的好办法。我们也可以通过给数据进行分页来让用户只关注于所

有数据的一部分。分页可以用两种方式来实现——服务器端分页和 JavaScript 分页。

Server-Side Pagination

服务器端的分页

Much like sorting, pagination is often performed on the server. If the data to be displayed is stored in a

database, it is easy to pull out one chunk of information at a time using MySQL's LIMIT clause,

ROWNUM in Oracle, or equivalent methods in other database engines.

就好像排序一样，分页经常在服务器端执行。如果要显示的数据被保存在数据库内，使用 SQL 语
句是很容易一次性找出所有数据中的那一部分，比如使用 MySQL 的 LIMIT 子句，Oracle 的
ROWNUM，以及其他的数据库中相对应的方法。

As with our initial sorting example, pagination can be triggered by sending information to the server in a
query string, such as index.php?page=52. And again as before, we can perform this task either with a full
page load or by using AJAX to pull in just one chunk of the table. This strategy is browser-independent,
and can handle large data sets very well.

跟我们最开始的排序的例子一样，也可以通过向服务器端发送一个查询字符串来进行分页的操
作，比如 index.php?page=52。同样，我们也可以使用整个页面刷新或者是 AJAX 部分刷新的方式
来进行分页。这个方法是和浏览器无关的，并且能够非常好地处理大量的数据。

Sorting and Paging Go Together

排序和分页同时进行

Data that is long enough to benefit from sorting is likely long enough

to be a candidate for paging. It is
not unusual to wish to combine these two techniques for data presentation. Since they both affect the set
of data that is present on a page, though, it is important to consider their interactions while implementing
them.

需要进行排序的数据，往往也需要进行分页。把这两种技术融合在一起用于数据的显示是经常见
到的事情。因为这两种技术都会对页面上显示的数据产生影响，所以，在实现排序和分页的同时
也需要考虑它们两者的分工协作。

Both sorting and pagination can be accomplished either on the server or in the web browser. However,
we must keep the strategies for the two tasks in sync; otherwise, we can end up with confusing behavior.
Suppose, for example, that both sorting and paging is done on the server:

排序和分页都可以在服务器端或者是页面上完成。然而，我们必须让这两者同步，否则，就会出
现令人感到非常迷惑的结果。假设，当排序和分页都在服务器上完成时：

When the table is re-sorted by number, a different set of rows is present on Page 1 of the table. If paging
is done by the server and sorting by the browser, the entire data set is not available for the sorting
routine, making the results incorrect:

当表格按照数字进行排序的时候，表格的第 1 页上就是一组不同的数据了。如果分页由服务器端
完成而排序由浏览器完成，整个数据的集合对于排序来说就不是正确的数据，得到的结果会有
误。

Only the data already present on the page can be displayed. To prevent this from being a problem, we
must either perform both tasks on the server, or both in the browser.

只有已经在页面上存在的数据才能显示出来。为了避免发生问题，我们必须把排序和分页都放在
服务器端执行或者都在浏览器内执行。

JavaScript Pagination

JavaScript 分页

So, let's examine how we would add JavaScript pagination to the table we have already made sortable in
the browser. First, we'll focus on displaying a particular page of data,

disregarding user interaction for

now:

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

$('table.paginated').each(function() {

var currentPage = 0;

var numPerPage = 10;

var $table = $(this);

$table.find('tbody tr').show()
```

```
.lt(currentPage * numPerPage)
.hide().
end()
.gt((currentPage + 1) * numPerPage - 1)
.hide()
.end();
});
});
```

This code displays the first page—ten rows of data.

现在让我们看看如何给浏览器中已经能够排序的表格加上 JavaScript 分页。首先，我们要完成在页面上显示所有数据中的某个特定的页面，暂时不用顾及到用的操作：

```
$(document).ready(function() {
$('table.paginated').each(function() {
var currentPage = 0;
var numPerPage = 10;
var $table = $(this);
$table.find('tbody tr').show()
.lt(currentPage * numPerPage)
.hide()
```

```
.end()
.gt((currentPage + 1) * numPerPage - 1)
.hide()
.end();
});
});
```

这段代码会显示第 1 页的 10 行数据。

Once again we rely on the presence of a <tbody> element to separate data from headers; we don't want to have the headers or footers disappear when moving on to the second page. For selecting the rows containing data, we show all the rows first, then select the rows before and after the current page, hiding them. The method chaining supported by jQuery makes another appearance here when we filter the set of matched rows twice, using .end() in between to pop the current filter off the stack and start afresh with a new filter.

我们再一次使用<tbody>标签来把数据和页眉分隔开，但是在访问第二页的时候，我们并不想让页眉和页脚显示出来。为了选出包含数据的行，我们要先显示所有的行，然后找出在当前页之前

和之后的所有行，隐藏它们。当我们两次匹配的行对匹配的行进行过滤，使用.end()将现在的过
滤器从堆栈中弹出，并初始化一个新的过滤器的时候，jQuery 的这些方法就会让页面上显示出我
们所需要的内容。

The most error-prone task in writing this code is formulating the expressions to use in the filters. To use
the .lt() and .gt() methods, we need to find the indices of the rows at the beginning and end of the current
page. For the beginning row, we just multiply the current page number by the number of rows on each
page. Multiplying the number of rows by one more than the current page number gives us the beginning
row of the next page; to find the last row of the current page, we must subtract one from this.

在这里最容易犯的错误就是准确的写出在过滤器中的表达式。要使用.lt()和.gt()方法，我们就要
找到当前页的第一个和最后一个行索引。对于第一个行索引，我们用当前页的页数乘上每页显示
的行数就可以得到。同样的方法可以计算下一页的第一个行索引，减去 1 之后就得到了当前页的
最后一个行索引。

Displaying the Pager

显示分页链接

To add user interaction to the mix, we need to place the pager itself next to the table. We could do this by simply inserting links for the pages in the HTML markup, but this would violate the progressive enhancement principle we've been espousing. Instead, we should add the links using JavaScript, so that

users without scripting available are not misled by links that cannot work.

为了能够让用户进行交互操作，我们需要给表格加上分页链接。我们可以给页面用 HTML 来加上

链接，但是这个可能会和我们一直提倡的循序渐进规则有冲突。取而代之的做法是使用

JavaScript 来添加链接，这样当脚本无效的时候，用户们就不会被那些链接给误导了。

To display the links, we need to calculate the number of pages and create a corresponding number of

DOM elements:

```
var numRows = $table.find('tbody tr').length;
var numPages = Math.ceil(numRows / numPerPage);
var $pager = $('<div class="pager"></div>');
for (var page = 0; page < numPages; page++) {
$('<span class="page-number">' + (page + 1) + '</
span>') .appendTo($pager).addClass('clickable');
}
$pager.insertBefore($table);
```

为了显示出链接，我们需要计算出有多少页并创建出相关的 DOM 元

素。

```
var numRows = $table.find('tbody tr').length;
var numPages = Math.ceil(numRows / numPerPage);
var $pager = $('<div class="pager"></div>');
for (var page = 0; page < numPages; page++) {
$('<span class="page-number">' +
(page + 1) + '</
span>') .appendTo($pager).addClass('clickable');
}
$pager.insertBefore($table);
```

The number of pages can be found by dividing the number of data rows by the number of items we wish
to display on each page. If the division does not yield an integer, we must round the result up using
Math.ceil() to ensure that the final partial page will be accessible. Then, with this number in hand, we
create buttons for each page and position the new pager before the table:

分页数可以通过用数据的总数除以每页的显示数量得到。如果结果不是一个整数的话，就需要使
用 Math.ceil()进行向上取整。得到页数之后，就可以给每一页建立分页按钮并把整个分页链接放

在表格之前。

Enabling the Pager Buttons

使用分页按钮

To make these new buttons actually work, we need to update the currentPage variable and then run our pagination routine. At first blush, it seems we should be able to do this by setting currentPage to page, which is the current value of the iterator that creates the buttons:

```
$(document).ready(function() {
$('table.paginated').each(function() {
var currentPage = 0;
var numPerPage = 10;
var $table = $(this);
var repaginate = function() {
$table.find('tbody tr').show()
.lt(currentPage * numPerPage)
.hide()
.end()
.gt((currentPage + 1) * numPerPage - 1)
.hide()
.end();
};
```

```
var numRows = $table.find('tbody tr').length;
var numPages = Math.ceil(numRows / numPerPage);
var $pager = $('<div class="pager"></div>');
for (var page = 0; page < numPages; page++) {
$('<span class="page-number">' + (page + 1) +
'</span>') .click(function() {
currentPage = page;
repaginate();
}) .appendTo($pager).addClass('clickable');
}
$pager.insertBefore($table);
repaginate();
});
});
```

为了能够让这些新的按钮实际工作起来，我们需要更新变量 currentPage 并执行分页程序。最初看来，我们只需要把 currentPage 设置到页面上就能达到这个目的了，这样就能根据当前值来创建分页按钮。

```
$(document).ready(function() {
$('table.paginated').each(function() {
var currentPage = 0;
var numPerPage = 10;
var $table = $(this);
var repaginate = function() {
$table.find('tbody tr').show()
.lt(currentPage * numPerPage)
.hide()
.end()
.gt((currentPage + 1) * numPerPage - 1)
.hide()
.end();
};
var numRows = $table.find('tbody tr').length;
var numPages = Math.ceil(numRows / numPerPage);
```

```
var $pager = $('<div class="pager"></div>');
for (var page = 0; page < numPages; page++) {
$('<span class="page-number">' + (page + 1) +
'</span>') .click(function() {
currentPage = page;
repaginate();
}) .appendTo($pager).addClass('clickable');
}
$pager.insertBefore($table);
repaginate();
});
});
```

This mostly works. The new repaginate() function is called when the page loads and when any button is

clicked. All of the buttons take us to a page with no rows on it, though:

这种做法挺有效的。新引入的 repaginate()函数在页面刷新或者是任何分页按钮被点击的时候会被

调用。不过任何分页按钮链接到的页面都是没有任何数据的。

The problem is that in defining our click handler, we have created a closure. The click handler refers to

the page variable, which is defined outside the function. When the

variable changes the next time

through the loop, this affects the click handlers that we have already set up for the earlier buttons. The

net effect is that, for a pager with 7 pages, each button directs us to page 8 (the final value of page).

More information on how closures work can be found in Appendix C, JavaScript Closures.

出现这个问题的原因是在定义 click 事件的时候，我们创建了闭包。Click 事件调用了 page 变量，这

个变量是在循环外部定义的。当变量通过循环发生改变时，就会对之前我们已经设置好的按钮的

click 事件产生影响。实际的结果就是，比如一个有 7 个页面的分页链接，每一个按钮都是指向第 8

页的链接（page 最后的值）。闭包的原理可以在附录 C JavaScript 闭包中找到。

To correct this problem, we'll take advantage of one of the more advanced features of jQuery's event

binding methods. We can add a set of data to the handler when we bind it that will still be available when

the handler is eventually called. With this capability in our bag of tricks, we can write:

```
$('<span class="page-number">' + (page + 1) + '</span>')
.bind('click', {'newPage': page}, function(event) {
currentPage = event.data['newPage'];
repaginate();
}) .appendTo($pager).addClass('clickable');
```

为了修正这个问题，我们会利用到 jQuery 的事件绑定方法的更加突出的特性。当我们进行事件绑

定的时候可以给事件处理器加上一些数据，当事件被执行的时候这些数据就能起到作用了。我们

可以像下面这样写：

```
$('<span class="page-number">' + (page + 1) + '</span>')
.bind('click', {'newPage': page}, function(event) {
currentPage = event.data['newPage'];
repaginate();
}) .appendTo($pager).addClass('clickable');
```

The new page number is passed into the handler by way of the event's data property. In this way the page
number escapes the closure, and is frozen in time at the value it contained when the handler was bound.

Now our pager buttons can correctly take us to different pages：

通过绑定 event 的 data 属性，新的 page 的值会被传递到事件处理器。这样一来，page 的值就从闭包
中脱离出来了，而且在事件处理器被绑定的时候它的当前值会被锁定。这样一来现在分页链接的
按钮就能够链接到正确的页面了。

Marking the Current Page

标记当前页

Our pager can be made more user-friendly by highlighting the current page number. We just need to

update the classes on the buttons every time one is clicked:

```
var $pager = $('<div class="pager"></div>');
for (var page = 0; page < numPages; page++) {
$('<span class="page-number">' + (page + 1) + '</span>')
.bind('click', {'newPage': page}, function(event) {
currentPage = event.data['newPage'];
repaginate();
$(this).addClass('active').siblings().removeClass('active');
}) .appendTo($pager).addClass('clickable');
}
$pager.find('span.page-number:first').addClass('active');
$pager.insertBefore($table);
```

现在的分页链接能够通过当前页的高亮来做到更加人性化。我们只需要在每次点击的时候更新按

钮的 class 属性。

```
var $pager = $('<div class="pager"></div>');
for (var page = 0; page < numPages; page++) {
```

```
$('<span class="page-number">' + (page + 1) + '</span>')
.bind('click', {'newPage': page}, function(event) {
currentPage = event.data['newPage'];
repaginate();
$(this).addClass('active').siblings().removeClass('active');
}) .appendTo($pager).addClass('clickable');
}
$pager.find('span.page-number:first').addClass('active');
$pager.insertBefore($table);
```

Now we have an indicator of the current status of the pager:

现在分页链接的当前状态就有了一个明确的显示。

Paging with Sorting

使用排序的分页

We began this discussion by noting that sorting and paging controls needed to be aware of one another to

avoid confusing results. Now that we have a working pager, we need to make sort operations respect the

current page selection.

之前提及到排序和分页需要交互的处理以避免出现错误的结果。现在我们有一个页面，需要做到

在不影响分页的情况下做到正确排序。

Doing this is as simple as calling our repaginate() function whenever a sort is performed. The scope of

the function, though, makes this problematic. We can't reach repaginate() from our sorting routine
because it is contained inside a different $(document).ready() handler. We could just consolidate the two
pieces of code, but instead let's be a bit sneakier. We can decouple the behaviors, so that a sort calls the
repaginate behavior if it exists, but ignores it otherwise. To accomplish this, we'll use a handler for a
custom event.
最简单的做法就是在任何要进行排序的时候调用 repaginate()函数。不过 repaginate()这个函数的作
用域会让这个做法有点问题。在排序的程序中我们不能访问 repaginate()函数，因为它是在另外一
个 different $(document).ready()事件中的。我们仅仅能够合并这两段代码，不过倒是可以取巧。我
们可以把这些操作独立出来，这样如果分页时页码重编的操作存在，排序就会调用它，不存在的
时候就忽略。为了达到这个目的，我们需要给一个自定义事件使用使用事件处理器。
In our earlier event handling discussion, we limited ourselves to event names that were triggered by the

web browser, such as click and mouseup. The .bind() and .trigger()
methods are not limited to these
events, though; we can use any string as an event name. In this case,
we can define a new event called
repaginate as a stand-in for the function we've been calling:

```
$table.bind('repaginate', function() {
$table.find('tbody tr').show()
.lt(currentPage * numPerPage)
.hide()
.end()
.gt((currentPage + 1) * numPerPage - 1)
.hide()
.end();
});
```

在之前有关事件处理的讨论中，我们只限于浏览器触发的事件，比如
click 和 mouseup。可
是.bind()和.trigger()方法并不局限于这些事件，我们可以使用任何
字符串作为事件的名字。这种
情况下，我们就能够定义一个名字叫做 repaginate 的事件作为我们调
用的函数的替代品。

```
$table.bind('repaginate', function() {
$table.find('tbody tr').show()
```

```
.lt(currentPage * numPerPage)
.hide()
.end()
.gt((currentPage + 1) * numPerPage - 1)
.hide()
.end();
});
```

Now in places where we were calling repaginate(), we can call:

```
$table.trigger('repaginate');
```

We can issue this call in our sort code as well. It will do nothing if the table does not have a pager, so we
can mix and match the two capabilities as desired.

现在在我们调用 repaginate()的地方，可以改成：

```
$table.trigger('repaginate');
```

我们也可以在排序的代码中使用这个方法。当表格没有分页链接的时候它就什么都不做，这样我
们就如期把排序和分页整合到了一起。

The Finished Code

最终代码

The completed sorting and paging code in its entirety follows:

```
$.fn.alternateRowColors = function() {
$('tbody tr:odd', this).removeClass('even').addClass('odd');
```

```
$('tbody tr:even', this).removeClass('odd').addClass('even');
return this;
};
```

```
$(document).ready(function() {
var alternateRowColors = function($table) {
$('tbody tr:odd', $table).removeClass('even').addClass('odd');
$('tbody tr:even', $table).removeClass('odd').addClass('even');
};
$('table.sortable').each(function() {
var $table = $(this);
$table.alternateRowColors($table);
$table.find('th').each(function(column) {
var findSortKey;
if ($(this).is('.sort-alpha')) {
findSortKey = function($cell) {
return $cell.find('.sort-key').text().toUpperCase()
+ ' ' +
$cell.text().toUpperCase();
};
}
else if ($(this).is('.sort-numeric')) {
findSortKey = function($cell) {
var key = parseFloat($cell.text().replace(/^[^\d.]*/, ''));
return isNaN(key) ? 0 : key;
```

```
};
}
else if ($(this).is('.sort-date')) {
findSortKey = function($cell) {
return Date.parse('1 ' + $cell.text());
};
}
if (findSortKey) {
$(this).addClass('clickable').hover(function() {
$(this).addClass('hover');
}, function() {
$(this).removeClass('hover');
}).click(function() {
var newDirection = 1;
if ($(this).is('.sorted-asc')) {
newDirection = -1;
}
rows = $table.find('tbody > tr').get();
$.each(rows, function(index, row) {
row.sortKey = findSortKey($(row).children('td').eq(column));
});
rows.sort(function(a, b) {
```

```
if (a.sortKey < b.sortKey) return -newDirection;
if (a.sortKey > b.sortKey) return newDirection;
return 0;
});
$.each(rows, function(index, row) {
$table.children('tbody').append(row);
row.sortKey = null;
});
$table.find('th').removeClass('sortedasc')  .removeClass('sorted
-desc');
var $sortHead = $table.find('th').filter(':nth-child(' + (column +
1) + ')');
if (newDirection == 1) {
$sortHead.addClass('sorted-asc');
} else {
```

```
$sortHead.addClass('sorted-desc');
}
$table.find('td').removeClass('sorted')
.filter(':nth-child(' + (column + 1) + ')') .addClass('sorted');
$table.alternateRowColors($table);
$table.trigger('repaginate');
});
}
});
});
});
$(document).ready(function() {
$('table.paginated').each(function() {
var currentPage = 0;
var numPerPage = 10;
var $table = $(this);
$table.bind('repaginate', function() {
$table.find('tbody tr').show()
.lt(currentPage * numPerPage)
.hide()
.end()
```

```
.gt((currentPage + 1) * numPerPage - 1)
.hide()
.end();
});
var numRows = $table.find('tbody tr').length;
var numPages = Math.ceil(numRows / numPerPage);
var $pager = $('<div class="pager"></div>');
for (var page = 0; page < numPages; page++) {
$('<span class="page-number">' + (page + 1) + '</span>')
.bind('click', {'newPage': page}, function(event) {
currentPage = event.data['newPage'];
$table.trigger('repaginate');
$(this).addClass('active').siblings().removeClass('active');
})
.appendTo($pager).addClass('clickable');
}
$pager.find('span.page-number:first').addClass('active');
$pager.insertBefore($table);
$table.trigger('repaginate');
});
});
```

最终完整的排序和分页的代码如下面所示：

```
$.fn.alternateRowColors = function() {
$('tbody tr:odd', this).removeClass('even').addClass('odd');
$('tbody tr:even', this).removeClass('odd').addClass('even');
return this;
};
$(document).ready(function() {
var alternateRowColors = function($table) {
$('tbody tr:odd', $table).removeClass('even').addClass('odd');
$('tbody tr:even', $table).removeClass('odd').addClass('even');
};
$('table.sortable').each(function() {
var $table = $(this);
```

```
$table.alternateRowColors($table);
$table.find('th').each(function(column) {
var findSortKey;
if ($(this).is('.sort-alpha')) {
findSortKey = function($cell) {
return $cell.find('.sort-key').text().toUpperCase()
+ ' ' +
$cell.text().toUpperCase();
};
}
else if ($(this).is('.sort-numeric')) {
findSortKey = function($cell) {
var key = parseFloat($cell.text().replace(/^[^\d.]*/, ''));
return isNaN(key) ? 0 : key;
};
}
else if ($(this).is('.sort-date')) {
findSortKey = function($cell) {
return Date.parse('1 ' + $cell.text());
};
}
```

```
if (findSortKey) {
$(this).addClass('clickable').hover(function() {
$(this).addClass('hover');
}, function() {
$(this).removeClass('hover');
}).click(function() {
var newDirection = 1;
if ($(this).is('.sorted-asc')) {
newDirection = -1;
}
rows = $table.find('tbody > tr').get();
$.each(rows, function(index, row) {
row.sortKey = findSortKey($(row).children('td').eq(column));
});
rows.sort(function(a, b) {
if (a.sortKey < b.sortKey) return -newDirection;
if (a.sortKey > b.sortKey) return newDirection;
return 0;
});
$.each(rows, function(index, row) {
$table.children('tbody').append(row);
row.sortKey = null;
```

```
});
$table.find('th').removeClass('sortedasc')  .removeClass('sorted
-desc');
var $sortHead = $table.find('th').filter(':nth-child(' + (column +
1) + ')');
if (newDirection == 1) {
$sortHead.addClass('sorted-asc');
} else {
$sortHead.addClass('sorted-desc');
}
$table.find('td').removeClass('sorted')
.filter(':nth-child(' + (column + 1) + ')') .addClass('sorted');
$table.alternateRowColors($table);
$table.trigger('repaginate');
});
```

```
}
});
});
});
$(document).ready(function() {
$('table.paginated').each(function() {
var currentPage = 0;
var numPerPage = 10;
var $table = $(this);
$table.bind('repaginate', function() {
$table.find('tbody tr').show()
.lt(currentPage * numPerPage)
.hide()
.end()
.gt((currentPage + 1) * numPerPage - 1)
.hide()
.end();
});
var numRows = $table.find('tbody tr').length;
var numPages = Math.ceil(numRows / numPerPage);
var $pager = $('<div class="pager"></div>');
```

```
for (var page = 0; page < numPages; page++) {
$('<span class="page-number">' + (page + 1) + '</span>')
.bind('click', {'newPage': page}, function(event) {
currentPage = event.data['newPage'];
$table.trigger('repaginate');
$(this).addClass('active').siblings().removeClass('active');
})
.appendTo($pager).addClass('clickable');
}
$pager.find('span.page-number:first').addClass('active');
$pager.insertBefore($table);
$table.trigger('repaginate');
});
});
```

Advanced Row Striping

深入行背景色操作

As we saw earlier in the chapter, row striping can be as simple as two lines of code to alternate the background color:

```
$(document).ready(function() {
$('table.sortable tbody tr:odd').addClass('odd');
$('table.sortable tbody tr:even').addClass('even');
```

});

我们在这一章的前面部分看到，行背景色操作可以简单到用两行代码来做出背景色交替的效果。

$(document).ready(function() {

$('table.sortable tbody tr:odd').addClass('odd');

$('table.sortable tbody tr:even').addClass('even');

});

If we declare background colors for the odd and even classes as follows, we can see the rows in

alternating shades of gray:

tr.even {

background-color: #eee;

```
}
tr.odd {
background-color: #ddd;
}
```

如果我们为 odd 和 even 定义了背景色的话，就能够看到数据行会有交替的灰度梯度显示出来。

```
tr.even {
background-color: #eee;
}
tr.odd {
background-color: #ddd;
}
```

While this code works fine for simple table structures, if we introduce nonstandard rows into the table, such as sub-headings, the basic odd-even pattern no longer suffices. For example, suppose we have a table of news items grouped by year, with columns for date, headline, author, and topic. One way to express this information is to wrap each year's news items in a <tbody> element and use <th colspan="4"> for the subheading. Such a table's HTML (in abridged

form) would look like this:

```
<table class="striped">
<thead>
<tr>
<th>Date</th>
<th>Headline</th>
<th>Author</th>
<th class="filter-column">Topic</th>
</tr>
</thead>
<tbody>
<tr>
<th colspan="4">2007</th>
</tr>
<tr>
<td>Mar 11</td>
<td>SXSWi jQuery Meetup</td>
<td>John Resig</td>
<td>conference</td>
</tr>
<tr>
<td>Feb 28</td>
```

```
<td>jQuery 1.1.2</td>

<td>John Resig</td>

<td>release</td>

</tr>

<tr>

<td>Feb 21</td>

<td>jQuery is OpenAjax Compliant</td>

<td>John Resig</td>

<td>standards</td>

</tr>

<tr>

<td>Feb 20</td>

<td>jQuery and Jack Slocum's Ext</td>

<td>John Resig</td>

<td>third-party</td>

</tr>

</tbody>
```

```
<tbody>
<tr>
<th colspan="4">2006</th>
</tr>
<tr>
<td>Dec 27</td>
<td>The Path to 1.1</td>
<td>John Resig</td>
<td>source</td>
</tr>
<tr>
<td>Dec 18</td>
<td>Meet The People Behind jQuery</td>
<td>John Resig</td>
<td>announcement</td>
</tr>
<tr>
<td>Dec 13</td>
<td>Helping you understand jQuery</td>
<td>John Resig</td>
<td>tutorial</td>
```

```
</tr>
</tbody>
<tbody>
<tr>
<th colspan="4">2005</th>
</tr>
<tr>
<td>Dec 17</td>
<td>JSON and RSS</td>
<td>John Resig</td>
<td>miscellaneous</td>
</tr>
</tbody>
</table>
```

尽管上面的代码对于简单结构的代码来说是有效的，但是如果我们给表格中引入了非标准的行，

比如子页眉，那么原来的奇偶行的形式就不行了。比如，我们有一个新闻的表格，所有的新闻都

用年份进行了分组，列数据是日期、说明文、作者、标题。要显示这些信息的一种方法就是把每

一年的新闻放在 <tbody> 标签中并给子页眉使用 <th colspan="4">。这样的一个表格的 HTML 就像

下面的一样。

```
<table class="striped">

<thead>

<tr>

<th>Date</th>

<th>Headline</th>

<th>Author</th>

<th class="filter-column">Topic</th>

</tr>

</thead>

<tbody>

<tr>

<th colspan="4">2007</th>

</tr>

<tr>

<td>Mar 11</td>
```

```
<td>SXSWi jQuery Meetup</td>

<td>John Resig</td>

<td>conference</td>

</tr>

<tr>

<td>Feb 28</td>

<td>jQuery 1.1.2</td>

<td>John Resig</td>

<td>release</td>

</tr>

<tr>

<td>Feb 21</td>

<td>jQuery is OpenAjax Compliant</td>

<td>John Resig</td>

<td>standards</td>

</tr>

<tr>

<td>Feb 20</td>

<td>jQuery and Jack Slocum's Ext</td>

<td>John Resig</td>

<td>third-party</td>
```

```
</tr>
</tbody>
<tbody>
<tr>
<th colspan="4">2006</th>
</tr>
<tr>
<td>Dec 27</td>
<td>The Path to 1.1</td>
<td>John Resig</td>
<td>source</td>
</tr>
<tr>
<td>Dec 18</td>
<td>Meet The People Behind jQuery</td>
<td>John Resig</td>
<td>announcement</td>
</tr>
<tr>
<td>Dec 13</td>
<td>Helping you understand jQuery</td>
<td>John Resig</td>
```

<td>tutorial</td>

</tr>

</tbody>

<tbody>

<tr>

<th colspan="4">2005</th>

</tr>

<tr>

<td>Dec 17</td>

<td>JSON and RSS</td>

<td>John Resig</td>

<td>miscellaneous</td>

</tr>

</tbody>

</table>

With separate CSS styles applied to <th> elements within <thead> and <tbody>, a snippet of the table might look like this:

当<thead>和<tbody>内的<th>元素分别加载了 CSS 样式之后，整个表格看起来就像下面这个样子：

To ensure that the alternating gray rows do not override the color of the subheading rows, we need to

adjust the selector expression:

```
$(document).ready(function() {
$('table.striped tbody tr:not([th]):odd').addClass('odd');
$('table.striped tbody tr:not([th]):even').addClass('even');
});
```

为了让交替背景色不会影响到子页眉，我们需要修改一下程序：

```
$(document).ready(function() {
$('table.striped tbody tr:not([th]):odd').addClass('odd');
$('table.striped tbody tr:not([th]):even').addClass('even');
});
```

The added selector, :not([th]), removes any table row that contains a <th> from the matched set of

elements. Now the table will look like this:

增加了:not([th])选择符之后，交替背景色就不会应用到含有<th>元素的行了。现在整个表格看起来如下：

Three-color Alternating Pattern

三种背景色的模式

There may be times when we want to apply more complex striping. For example, we can apply a pattern of three

alternating row colors rather than just two. To do so, we first need to define another CSS rule for the third row. We'll

also reuse the odd and even styles for the other two, but add more appropriate class names for them:

```
tr.even,
tr.first {
background-color: #eee;
}
tr.odd,
tr.second {
background-color: #ddd;
}
tr.third {
background-color: #ccc;
}
```

可能有些时候，我们要给做出更加复杂的背景色。比如，将原来的两行的交替背景色改成三行的

交替背景色。为了能够做到这种效果，我们要给第三行定义一个新的CSS。原来的两行也可以使

用 odd 和 even 的 class 属性，但是有更加适合的名称给它们。

```
tr.even,
tr.first {
background-color: #eee;
}
```

```
tr.odd,
tr.second {
background-color: #ddd;
}
tr.third {
background-color: #ccc;
}
```

To apply this pattern, we start the same way as the previous example—by selecting all rows that are descendants of a <tbody>, but filtering out the rows that contain a <th>. This time, however, we attach the .each() method so that we can use its built-in index:

```
$(document).ready(function() {
$('table.striped tbody tr').not('[th]').each(function(index) {
//Code to be applied to each element in the matched set.
});
});
```

为了达到这种效果，我们可以像以前的例子一样，选出<tbody>标签中所有的行，并过滤掉<th>的行。但是这一次我们使用了.each()方法，这样我们就能够使用它的内建的索引了。

```
$(document).ready(function() {
$('table.striped tbody tr').not('[th]').each(function(index) {
//Code to be applied to each element in the matched set.
});
});
```

To make use of the index, we can assign our three classes to a numeric key: 0, 1, or 2. We'll do this by creating an object, or map:

```
$(document).ready(function() {
var classNames = {
0: 'first',
1: 'second',
2: 'third'
};
$('table.striped tbody tr').not('[th]').each(function(index) {
// Code to be applied to each element in the matched set.
});
});
```

为了使用索引，我们可以给这三个 class 赋上一个数字键：0、1、2。我们通过创建一个对象或一个图来达到这个目的。

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

```
var classNames = {
0: 'first',
1: 'second',
2: 'third'
};
$('table.striped tbody tr').not('[th]').each(function(index) {
// Code to be applied to each element in the matched set.
});
});
```

Finally, we need to add the class that corresponds to those three numbers, sequentially, and then repeat
the sequence. The modulus operator, designated by a %, is especially convenient for such calculations.
A modulus returns the remainder of one number divided by another. This modulus, or remainder value,
will always range between 0 and one less than the dividend. Using 3 as an example, we can see this
pattern:
3/3 = 1, remainder 0.

4/3 = 1, remainder 1.

5/3 = 1, remainder 2.

6/3 = 2, remainder 0.

7/3 = 2, remainder 1.

8/3 = 3, remainder 2.

And so on. Since we want the remainder range to be 0 - 2, we can use 3 as the divisor (second number)

and the value of index as the dividend (first number). Now we simply put that calculation in square

brackets after classNames to retrieve the corresponding class from the object variable as the .each()

method steps through the matched set of rows:

```
$(document).ready(function() {
var classNames = {
0: 'first',
1: 'second',
2: 'third'
};
$('table.striped tbody tr').not('[th]').each(function(index) {
$(this).addClass(classNames[index % 3]);
});
```

});

最后，我们就要根据这三个数字给对应的行加上 class。取模运算符%对于这种计算来说是非常方

便的。取模运算返回一个数被另一个数整除时的余数。这个余数永远介于 0 和除数之间。以 3 为

例：

3/3 = 1，余 0。

4/3 = 1，余 1。

5/3 = 1，余 2。

6/3 = 2，余 0。

7/3 = 2，余 1。

8/3 = 3，余 2。

由于我们需要让余数范围在 0、1、2 内，所以 3 就是除数而索引的值就是被除数。由于.each()方法

会对每一行执行相应的操作，所以现在我们在 classNames 后的方括号内执行这个计算，就能够得到对

应的 class。

```
$(document).ready(function() {
var classNames = {
0: 'first',
1: 'second',
2: 'third'
```

```
};
$('table.striped tbody tr').not('[th]').each(function(index) {
$(this).addClass(classNames[index % 3]);
});
});
```

With this code in place, we now have the table striped with three alternating background colors:

加入这段代码之后，这样我们的表格就有了三色的背景。

We could of course extend this pattern to four, five, six, or more background colors by adding key—value

pairs to the object variable and increasing the value of the divisor in classNames[index % n].

我们同样也可以扩展成更多的背景色的形式，只需要增加classNames的内容并修改

classNames[index % n]的除数。

Alternating Triplets

三行一组的交替背景色

Suppose we want to use two colors, but have each one display three rows at a time. For this, we can

employ the odd and even classes again, as well as the modulus operator. But we'll also reset the class

each time we're presented with a row containing <th> elements.

假设我们现在使用两种颜色的交替背景色，但是要每三行使用一种颜色。为了达到这种效果，就

要再用到 odd 和 even 的 class 以及取模运算。但是当行内含有<th>元素的时候，我们就要重新设置

class。

If we don't reset the alternating row class, we may be faced with unexpected colors after the first group

of rows is striped. So far, our example table has avoided such problems because the first group consists
of 12 rows, which, conveniently, is divisible by both 2 and 3. For the triplet striping scenario, we'll
remove two rows, leaving us with 10 in the first group, to emphasize the class resetting.

如果不重置 class 的话，可能就会在设置了第一组的背景色后，第二组的背景色不是希望的颜色。

到目前为止，我们的例子中的表格没有出现这种问题，因为它包含了 12 行数据，可以被 2 和 3 整
除。在这次的三行一组的交替背景色的例子中，我们去掉两行留下 10 行数据，突显出 class 重置的
作用。

We begin this striping technique by setting two variables, rowClass and rowIndex. We'll use the .each()
method this time as well, but rather than relying on the built-in index, we'll use a custom rowIndex
variable so that we can reset it on the rows with <th>:

```
$(document).ready(function() {
var rowClass = 'even';
var rowIndex = 0;
$('table.striped tbody tr').each(function(index) {
```

```
$(this).addClass(rowClass);
});
});
```

我们在这里的程序中设置了两个变量 rowClass 和 rowIndex。同样也使用.each()方法，和之前使用
内建的索引不同，我们要使用一个自定义的 rowIndex 变量，这样在遇到<th>的时候就能够对它进
行重置了。

```
$(document).ready(function() {
var rowClass = 'even';
var rowIndex = 0;
$('table.striped tbody tr').each(function(index) {
$(this).addClass(rowClass);
});
});
```

Notice that since we have removed the :not([th]) selector, we'll have to account for those subheading
rows within the .each(). But first, let's get the triplet alternation working properly. So far, every <tr> will
become <tr class="even">. For each row, we can check to see if the rowIndex % 3 equals 0. If it does,
we toggle the value of rowClass. Then we increment the value of

```
rowIndex:

$(document).ready(function() {

var rowClass = 'even';

var rowIndex = 0;

$('table.striped tbody tr').each(function(index) {

if (rowIndex % 3 == 0) {

rowClass = (rowClass == 'even' ? 'odd' : 'even');
```

```
};
$(this).addClass(rowClass);
rowIndex++;
});
});
```

因为我们没有用:not([th])选择符，所以就必须处理.each()中遇到的子页眉所在行。但是首先要先
让三行一组的交替背景色正常显示出来。到目前为止，每一个<tr>都变成了<tr class="even">。对
每一行来说，我们可以测试一下 rowIndex % 3 的结果是否为 0；如果是，就标记 rowClass 的值。然
后给 rowIndex 加 1。

```
$(document).ready(function() {
var rowClass = 'even';
var rowIndex = 0;
$('table.striped tbody tr').each(function(index) {
if (rowIndex % 3 == 0) {
rowClass = (rowClass == 'even' ? 'odd' : 'even');
};
$(this).addClass(rowClass);
rowIndex++;
```

```
});
});
```

A ternary, or conditional, operator is used to set the changed value of rowClass because of its

succinctness. That single line could be rewritten as:

```
if (rowClass == 'even') {
rowClass = 'odd';
} else {
rowClass = 'even';
}
```

三元运算符，或者叫作条件运算符，因为它很简洁，在这里用来对 rowClass 的值进行修改。上面

的单行代码可以写成：

```
if (rowClass == 'even') {
rowClass = 'odd';
} else {
rowClass = 'even';
}
```

In either case, the code now produces table striping that looks like this:

不管使用哪种形式，最后能够得到下面一样的效果：

Perhaps surprisingly, the subheading rows have retained their proper formatting. But let's not be fooled
by appearances. The 2007 subheading row is now set in the HTML as <tr class="odd"> and the 2006
row has <tr class="even">. In the stylesheet, however, the greater specificity of the element's rule
outweighs that of the two classes:

```
#content tbody th {
background-color: #6f93ce;
padding-left: 6px;
}
tr.even {
background-color: #eee;
}
tr.odd {
background-color: #ddd;
}
```

也许感到有点意外，子页眉所在的行仍然保持了原有的格式。但是不要被表面给愚弄了。2007 的
子页眉行被设置成了<tr class="odd">而 2006 的子页眉行成了<tr class="even">。但是在 CSS 中，由

于优先级的关系，这两行并没有显示成其他的样子。

```
#content tbody th {
background-color: #6f93ce;
padding-left: 6px;
}
tr.even {
background-color: #eee;
}
tr.odd {
background-color: #ddd;
```

}

Nevertheless, because the rowIndex numbering does not account for these subheading rows, we have

mis-classed rows from the start; this is evident because the first striping color change occurs after two

rows rather than three.

不过，由于 rowIndex 并没有针对这些子页眉行进行特殊处理，造成了从一开始就有了错误的行的

情况。很明显地就能看出，第一次背景色的变化是在两行数据之后而不是在三行数据之后发生

的。

We need to include another condition, checking if the current row contains a <th>. If it does, we'll set the

value of rowClass to subhead and set rowIndex to -1:

```
$(document).ready(function() {
var rowClass = 'even';
var rowIndex = 0;
$('table.striped tbody tr').each(function(index) {
if ($('th', this).length) {
rowClass = 'subhead';
rowIndex = -1;
```

```
} else if (rowIndex % 3 == 0) {
rowClass = (rowClass == 'even' ? 'odd' : 'even');
};
$(this).addClass(rowClass);
rowIndex++;
});
});
```

我们要考虑到另外一种情况，要检验当前行是否含有<th>标签。如果有，就把 rowClass 赋值成 subhead 并把 rowIndex 赋值成−1。

```
$(document).ready(function() {
var rowClass = 'even';
var rowIndex = 0;
$('table.striped tbody tr').each(function(index) {
if ($('th', this).length) {
rowClass = 'subhead';
rowIndex = -1;
} else if (rowIndex % 3 == 0) {
rowClass = (rowClass == 'even' ? 'odd' : 'even');
};
$(this).addClass(rowClass);
rowIndex++;
```

```
});
});
```

With rowIndex at −1 for the subheading rows, the variable will be incremented to 0 for the next row—

precisely where we want it to start for each group of striped rows. Now we can see the striping with each

year's articles beginning with three light colored rows and alternating three at a time between lighter and

darker:

在子页眉行处把 rowIndex 赋值成-1，在下一行内 rowIndex 就会变成 0

——这正好就是我们所需要的

每一组初始时的 rowIndex 的值。现在我们可以看到每一年内的新闻，

都是三行一组浅色背景，三

行一组深色背景交替出现。

A final note about this striping code—while the ternary operator is indeed concise, it can get confusing
when the conditions get more complex. The sophisticated striping variations can be more easily
managed by using basic if–else conditions instead:

```
$(document).ready(function() {
var rowIndex = 0;
$('tbody tr').each(function(index) {
if ($('th',this).length) {
$(this).addClass('subhead');
rowIndex = -1;
} else {
if (rowIndex % 6 < 3) {
$(this).addClass('even');
}
else {
$(this).addClass('odd');
}
};
rowIndex++;
});
```

```
});
```

这是最后的背景色设置的代码，由于三元运算符实在是很简洁，在条件很复杂的时候它可能会让

人感觉很难懂。使用基本的 if-else 语句就可以很容易的来控制背景色的变化了。

```
$(document).ready(function() {
var rowIndex = 0;
$('tbody tr').each(function(index) {
```

```
if ($('th',this).length) {
$(this).addClass('subhead');
rowIndex = -1;
} else {
if (rowIndex % 6 < 3) {
$(this).addClass('even');
}
else {
$(this).addClass('odd');
}
};
rowIndex++;
});
});
```

Now we've achieved the same effect as before, but also made it easier to include additional else if conditions.

现在我们就得到了和之前一样的效果，但是使用了 else if 条件语句就能更容易地实现这种效果。

Row Highlighting

行的高亮

Another visual enhancement that we can apply to our news article table is row highlighting based on
user interaction. Here we'll respond to clicking on an author's name by highlighting all rows that have
the same name in their author cell. Just as we did with the row striping, we can modify the appearance of
these highlighted rows by adding a class:

```
#content tr.highlight {
background: #ff6;
}
```

另外一个强化界面显示的方法，就是在用户进行操作的时候，把新闻表格内的行进行高亮显示。
这里我们要实现当点击一个作者的名字的时候，要对所有有相同名字的行进行高亮显示。就像刚
才处理背景色一样，我们可以给需要高亮的行加上一个 class.

```
#content tr.highlight {
background: #ff6;
}
```

It's important that we give this new highlight class adequate specificity for the background color to
override that of the even and odd classes.
给高亮的 class 的背景色进行定制以覆盖原有的 even 和 odd 的 class

是非常重要的。

Now we need to select the appropriate cell and attach the .click() method to it:

```
$(document).ready(function() {
var column = 3;
$('table.striped td:nth-child(' + column + ')' )
.click(function() {
// Do something on click.
});
});
```

现在就需亚找出适当的单元格，并对其绑定.click()事件。

```
$(document).ready(function() {
var column = 3;
$('table.striped td:nth-child(' + column + ')' )
.click(function() {
// Do something on click.
});
```

});

Notice that we use the :nth-child(n) pseudo-class as part of the selector expression, but rather than
simply including the number of the child element, we pass in the column variable. We'll need to refer to
the same nth-child again, so using a variable allows us to change it in only one place if we later decide to
highlight based on a different column.

注意以下，我们使用了:nth-child(n)作为选择符表达式的一部份，并传递了变量 column，而不是简
单的包涵了子元素的数量。如果稍后要根据另外一列进行高亮的时候，我们需要再次引用到 nthchild，
所以使用一个变量来进行控制的话我们就只需要修改一个地方就可以了。

和 JavaScript 的索引不同，:nth-child(n)不是从 0 而是从 1 开始。

When the user clicks a cell in the third column, we want the cell's text to be compared to that of the same
column's cell in every other row. If it matches, the highlight class will be toggled. In other words, the
class will be added if it isn't already there and removed if it is. This way, we can click on an author cell

to remove the row highlighting if that cell or one with the same author has already been clicked：

```
$(document).ready(function() {
$('table.striped td:nth-child(' + column + ')' )
.click(function() {
var thisClicked = $(this).text();
$('table.striped td:nth-child(' + column + ')') .each(function(index) {
if (thisClicked == $(this).text()) {
$(this).parent().toggleClass('highlight');
};
});
});
})
```

当用户点击第三列中的单元格时，我们要把这一格的文本和该列中的其他单元格中的文本进行比
较。如果匹配，就会加上高亮的 class。换句话来说，就是在没有这个 class 的时候会加上该 class，
而有这个 class 的时候就会移除它。这样一来，如果有相同的作者的单元格已经被点击的时候，我
们就可以点击作者的单元格来取消行的高亮。

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

```
$('table.striped td:nth-child(' + column + ')' )
.click(function() {
var thisClicked = $(this).text();
$('table.striped td:nth-child(' + column + ')') .each(function(index) {
if (thisClicked == $(this).text()) {
$(this).parent().toggleClass('highlight');
};
});
});
})
```

The code is working well at this point, except when a user clicks on two authors' names in succession.

Rather than switching the highlighted rows from one author to the next as we might expect, it adds the second clicked author's rows to the group that has class="highlight".

To avoid this behavior, we can add an else statement to the code, removing the highlight class for any row that does not have the same author name as the one clicked:

```
$(document).ready(function() {
$('table.striped td:nth-child(' + column + ')' )
```

```
.click(function() {
var thisClicked = $(this).text();
$('table.striped td:nth-child(' + column + ')' ).each(function(index) {
if (thisClicked == $(this).text()) {
$(this).parent().toggleClass('highlight');
} else {
$(this).parent().removeClass('highlight');
};
});
});
})
```

除了当用户连续点击两个作者的名字的时候有点问题，这个代码在这里还是能够正常工作的。和
我们期望的从一个作者名字的高亮状态切换到另一个作者名字的高亮状态的情况有所不同，这段
代码会给第二次点击的作者所在的行都加上 class="highlight"。为了避免这种情况，我们可以给代
码加上 else 语句，把所有没有相同作者名字的行都除去高亮的 class.

```
$(document).ready(function() {
$('table.striped td:nth-child(' + column + ')' )
```

```
.click(function() {
var thisClicked = $(this).text();
$('table.striped td:nth-child(' + column + ')' ).each(function(index) {
if (thisClicked == $(this).text()) {
$(this).parent().toggleClass('highlight');
} else {
$(this).parent().removeClass('highlight');
};
});
});
})
```

Now when we click on Rey Bango, for example, we can see all of his articles much more easily:

比如我们现在点击 Rey Bango，就能很容易地看到他的所有文章。

If we then click on John Resig's name in any one of the cells, the highlighting will be removed from

Rey Bango's rows and added to John's.

如果我们接着点击 John Resig 的名字，Rey Bango 所在的行就会取消高亮，而 John Resig 所在的行

会加上高亮效果。

Tooltips

提示信息

Although the row highlighting might be a useful feature, so far it's not apparent to the user that the

feature even exists. We can begin to remedy this situation by giving all author cells a clickable class,

which will change the cursor to a pointer when a user hovers the mouse cursor over them:

```
$(document).ready(function() {
$('table.striped td:nth-child(' + column + ')' ).addClass('clickable').click(function() {
var thisClicked = $(this).text();
$('table.striped td:nth-child(' + column + ')' ).each(function(index) {
if (thisClicked == $(this).text()) {
```

```
$(this).parent().toggleClass('highlight');
} else {
$(this).parent().removeClass('highlight');
};
});
})
```

```
})
```

尽管行的高亮可能是个有用的东西，但是对于用户来说它还不够详细直接。现在我们要通过给所

有的作者单元格加上一个 clickable 的 class 来改善这种情况，效果就是当用户将鼠标移动到作者单

元格上的时候，鼠标会变成手掌的形式。

```
$(document).ready(function() {
$('table.striped td:nth-child(' + column + ')' ).addClass('clickable').click(function() {
var thisClicked = $(this).text();
$('table.striped td:nth-child(' + column + ')' ).each(function(index) {
if (thisClicked == $(this).text()) {
$(this).parent().toggleClass('highlight');
} else {
$(this).parent().removeClass('highlight');
};
});
})
})
```

The clickable class is a step in the right direction, for sure, but it

still doesn't tell the user what will

happen when the cell is clicked. As far as anyone knows (without looking at the code, of course) that

clicking could send the user to another page. Some further indication of what will happen upon clicking

is in order.

增加 clickable 的 class 是正确的一步，不过用户没有点击的情况下他们还是不知道接下来会发生什

么。任何人（没有见过代码）见到这种情况都知道点击之后会跳转到另外一个页面。对于点击之

后会发生什么情况，应该需要有一定的提示。

Tooltips are a familiar feature of many software applications, including web browsers. We can simulate a

tooltip with custom text, such as Click to highlight all rows authored by Rey Bango, when the mouse

hovers over one of the author cells. This way, we can alert users to the effect their action will have.

提示信息是很多软件都有的一个特性，包括 Web 浏览器。我们可以用自定义的文字来实现提示信

息，比如当鼠标移动到作者的单元格时，会出现点击这里会使所有 Rey Bango 的行出现高亮。这

样，我们就告诉了用户点击之后会出现怎样的效果。

We're going to create three functions—showTooltip, hideTooltip, and positionTooltip—outside any event
handlers and then call or reference them as we need them. Let's start with positionTooltip, which we'll
reference when the mouse moves over any of the author cells:

```
var positionTooltip = function(event) {
var tPosX = event.pageX - 5;
var tPosY = event.pageY + 20;
$('div.tooltip').css({top: tPosY, left: tPosX});
};
```

下面我们创建三个函数——showTooltip、hideTooltip 和 positionTooltip——要在所有事件的外面定
义，然后就可以在需要的时候调用它们了。先从 positionTooltip 开始，在鼠标移动到作者的单元
格的时候会调用这个函数。

```
var positionTooltip = function(event) {
var tPosX = event.pageX - 5;
var tPosY = event.pageY + 20;
$('div.tooltip').css({top: tPosY, left: tPosX});
};
```

Here we use the pageX and pageY properties of event to set the top and left positions of the tooltip.

When we reference the function in the .mousemove() method, tPosX will refer to 5 pixels to the left of the mouse cursor while tPosY will refer to 20 pixels below the cursor. We can attach this method to the same chain as the one being used already for .click():

```
$(document).ready(function() {
var positionTooltip = function(event) {
```

```
var tPosX = event.pageX - 5;
var tPosY = event.pageY + 20;
$('div.tooltip').css({top: tPosY, left: tPosX});
};
$('table.striped td:nth-child(' + column + ')' ).addClass('clickable').click(function() {
// ...Code continues...
})
.mousemove(positionTooltip);
});
```

这里我们使用 event 的属性 pageX 和 pageY 来设置提示信息的位置。当我们引用了.mousemove()方法里面的函数时，tPosX 会被设置成鼠标位置的左边 5 像素而 tPosY 会被设置成鼠标的下面 20 像素。我们可以给.click()所使用的函数绑定这个方法。

```
$(document).ready(function() {
var positionTooltip = function(event) {
var tPosX = event.pageX - 5;
var tPosY = event.pageY + 20;
$('div.tooltip').css({top: tPosY, left: tPosX});
};
```

```
$('table.striped td:nth-child(' + column + ')' ).addClass('clickable').click(function() {
// ...Code continues...
})
.mousemove(positionTooltip);
});
```

So, we've positioned the tooltip already, but we still haven't created it. That will be done in the
showTooltip function.

这样，我们就确定了提示信息的位置，不过我们还没有创建它。在 showTooltip 函数中会完成这个
功能。

The first thing that we do in the showTooltip function is remove all tooltips. This may seem
counterintuitive, but if we are going to show the tooltip each time the mouse cursor hovers over an
author cell; we don't want a proliferation of these tooltips appearing with each new cell hovered over:

```
var showTooltip = function(event) {
$('div.tooltip').remove();
};
```

在 showTooltip 函数中首先要做的就是除去所有的信息提示。这看起

来可能是违反了常规的，但是

如果我们要在鼠标移动到作者单元格上显示出提示信息的时候，我们不希望每一个单元格上都显

示出所有的提示信息。

```
var showTooltip = function(event) {
$('div.tooltip').remove();
};
```

Now we're ready to create the tooltip. We can wrap the entire and its contents in a $() function and

then append it to the document's body:

```
var showTooltip = function(event) {
$('div.tooltip').remove();
var $thisAuthor = $(this).text();
$('<div class="tooltip">Click to highlight all articles written by ' +
$thisAuthor + '</
div>').appendTo('body');
};
```

现在我们就要创建提示信息了。我们可以把整个 和它里面的内容都放在$()函数内并把它加

入到页面的上。

```
var showTooltip = function(event) {
$('div.tooltip').remove();
```

```
var $thisAuthor = $(this).text();
```

```
$('<div class="tooltip">Click to highlight all articles written by ' +
$thisAuthor + '</
div>').appendTo('body');
};
```

When the mouse cursor hovers over an author cell with Rey Bango in it, the tooltip will read, Click to highlight all articles written by Rey Bango. Unfortunately, the tooltip will appear at the bottom of the page. That's where the positionTooltip function comes in. We simply place that at the end of the showTooltip function:

```
var showTooltip = function(event) {
$('div.tooltip').remove();
var $thisAuthor = $(this).text();
$('<div class="tooltip">Click to highlight all articles written by ' +
$thisAuthor + '</
div>').appendTo('body');
positionTooltip(event);
};
```

当鼠标移动到含有 Rey Bango 的单元格时，就会读取提示信息：点击这里会使所有 Rey Bango 的

行出现高亮。可惜，提示信息会在页面的地步出现。这就是要引入 positionTooltip 函数的原因。

我们把它放在 showTooltip 的最后面。

```
var showTooltip = function(event) {
$('div.tooltip').remove();
var $thisAuthor = $(this).text();
$('<div class="tooltip">Click to highlight all articles written by ' +
$thisAuthor + '</
div>').appendTo('body');
positionTooltip(event);
};
```

The tooltip still won't be positioned correctly, though, unless we free it from its default postion:static

property. We can do that in the stylesheet:

```
.tooltip {
position: absolute;
z-index: 2;
background: #efd;
border: 1px solid #ccc;
padding: 3px;
}
```

提示信息的位置还是不正确，除非我们修改它的一些属性。可以这样

做：

```
.tooltip {
position：absolute；
z—index：2；
background：＃efd；
border：1px solid ＃ccc；
padding：3px；
}
```

The style rule also gives the tooltip a z—index higher than that of the surrounding elements to ensure that
it is layered on top of them, as well as sprucing it up with a background color, a border, and some
padding.

现在的CSS给提示信息加了一个z—index参数，使其高于周围的元素，这样就可以把提示信息放置
在最外层显示，同样也要给提示信息架上背景色、边框和填充。

Finally, we write a simple hideTooltip function：

```
var hideTooltip = function() {
$('div.tooltip').remove()；
}；
```

最后，我们写出了一个简单的 hideTooltip 函数。

```
var hideTooltip = function() {
$('div.tooltip').remove();
};
```

And now that we have functions for showing, hiding, and positioning the tooltip, we can reference them at the appropriate places in our code:

```
$(document).ready(function() {
var column = 3;
// Position the tooltip.
var positionTooltip = function(event) {
var tPosX = event.pageX - 5;
var tPosY = event.pageY + 20;
$('div.tooltip').css({top: tPosY, left: tPosX});
};
// Show (create) the tooltip.
var showTooltip = function(event) {
$('div.tooltip').remove();
var $thisAuthor = $(this).text();
$('<div class="tooltip">Click to highlight all articles written by ' +
$thisAuthor + '</
div>').appendTo('body');
```

```
positionTooltip(event);
};
// Hide (remove) the tooltip.
var hideTooltip = function() {
$('div.tooltip').remove();
};
$('table.striped td:nth-child(' + column + ')' ).addClass('clickable').click(function(event) {
var thisClicked = $(this).text();
$('table.striped td:nth-child(' + column + ')' ).each(function(index) {
if (thisClicked == $(this).text()) {
$(this).parent().toggleClass('highlight');
} else {
$(this).parent().removeClass('highlight');
};
});
})
.hover(showTooltip, hideTooltip)
.mousemove(positionTooltip);
})
```

现在我们就有了能够显示、隐藏和定位提示信息的函数了，这样就可

以在代码里合适的地方调用

它们。

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

var column = 3;

// Position the tooltip.

var positionTooltip = function(event) {

var tPosX = event.pageX - 5;

var tPosY = event.pageY + 20;

$('div.tooltip').css({top: tPosY, left: tPosX});

};

// Show (create) the tooltip.

var showTooltip = function(event) {

$('div.tooltip').remove();

var $thisAuthor = $(this).text();

$('<div class="tooltip">Click to highlight all articles written by ' +

$thisAuthor + '</

div>').appendTo('body');
```

```
positionTooltip(event);
};
// Hide (remove) the tooltip.
var hideTooltip = function() {
$('div.tooltip').remove();
};
$('table.striped td:nth-child(' + column + ')' ).addClass('clickable').click(function(event) {
var thisClicked = $(this).text();
$('table.striped td:nth-child(' + column + ')' ).each(function(index) {
if (thisClicked == $(this).text()) {
$(this).parent().toggleClass('highlight');
} else {
$(this).parent().removeClass('highlight');
};
});
})
.hover(showTooltip, hideTooltip)
.mousemove(positionTooltip);
})
```

Note that the .hover() and .mousemove() methods are referencing functions that are defined elsewhere.

As such, the functions take no parentheses. Also, because positionTooltip(event) is called inside

showTooltip, the tooltip is immediately positioned on hover; it then continues to be referenced as the

mouse cursor is moved over the cell due to the function's placement inside the .mousemove() method.

The tooltip now looks like this:

现在.hover()和.mousemove()就调用了之前定义的函数。同样，这些函数没有使用括号。而且，因

为 positionTooltip(event)是在 showTooltip 内部调用的，当鼠标移上去的时候，提示信息就立刻定位

好了；接下来由于在.mousemove()方法中也有这个函数，当鼠标移开的时候也会调用它。提示信

息现在看起来就是这个样子：

Everything works fine now, with a tooltip that appears when we hover over an author cell, moves with
the mouse movement, and disappears when we move the mouse cursor out of the cell. The only problem
is that the tooltip continues to suggest clicking on a cell to highlight the articles even after those articles
have been highlighted:

一切看起来都很正常，不管是鼠标移动到作者单元格上出现提示信息，还是鼠标移动时提示信息
业移动，或者鼠标移开时提示信息消失。唯一的问题就是点击单元格完成行高亮的操作之后，这
个提示信息还是会出现，告诉我们点击单元格会进行高亮操作。

What we need is a way to change the tooltip if the row has the highlight class. Fortunately, we have the showTooltip function, in which we can place a conditional test to check for the class. If the current cell's parent <tr> has the highlight class, we add un- in front of the word highlight when we create the tooltip:

```
$(document).ready(function() {
var highlighted = "";
// Code continues...
var showTooltip = function(event) {
$('div.tooltip').remove();
var $thisAuthor = $(this).text();
if ($(this).parent().is('.highlight')) {
highlighted = 'un-';
} else {
highlighted = '';
};
$('<div class="tooltip"> Click to ' + highlighted + 'highlight all articles written by '
+ $thisAuthor + '</div>').appendTo('body');
positionTooltip(event);
```

```
};
};
```

我们要做的就是在行高亮之后，取消提示信息。幸运的是，我们有 showTooltip 函数，可以在这个

函数里加上对 class 的判断。如果当前单元格所在的<tr>有了高亮的 class，我们就在创建提示信息

的时候把高亮的文字内容修改成取消高亮。

```
$(document).ready(function() {
var highlighted = "";
// Code continues...
var showTooltip = function(event) {
$('div.tooltip').remove();
var $thisAuthor = $(this).text();
if ($(this).parent().is('.highlight')) {
highlighted = 'un-';
} else {
highlighted = '';
};
$('<div class="tooltip"> Click to ' + highlighted + 'highlight all
articles written by '
+ $thisAuthor + '</div>').appendTo('body');
positionTooltip(event);
```

```
};
```

```
};
```

Our tooltip task would now be finished were it not for the need to trigger the tooltipchanging behavior

when a cell is clicked as well. For that, we need to call the showTooltip function inside the .click() event

handler:

```
$(document).ready(function() {
// Code continues...
.click(function(event) {
var thisClicked = $(this).text();
$('table.striped td:nth-child(' + column + ')' ).each(function(index) {
if (thisClicked == $(this).text()) {
$(this).parent().toggleClass('highlight');
} else {
$(this).parent().removeClass('highlight')
};
});
showTooltip.call(this, event);
})
// Code continues...
```

```
});
```

如果不是因为点击单元格的时候要触发提示信息变化的行为，我们的提示工作就应该结束了。为

了达到这个功能，我们要在 click()事件中调用 showTooltip 这个函数。

```
$(document).ready(function() {
// Code continues...
.click(function(event) {
var thisClicked = $(this).text();
$('table.striped td:nth-child(' + column + ')' ).each(function(index) {
if (thisClicked == $(this).text()) {
$(this).parent().toggleClass('highlight');
} else {
$(this).parent().removeClass('highlight')
};
});
showTooltip.call(this, event);
})
// Code continues...
});
```

By using the JavaScript call() function, we can invoke showTooltip as if it were defined within

the .click() handler. Therefore, this inherits the scope of .click().

Additionally, we pass in event so that

we can use its pageX and pageY information for the positioning.

通过使用 JavaScript 的 call()函数，如果 showTooltip 在.click()事件中定义了，我们就可以进行调用
了。因此，它继承了.click()的作用域。而且我们把 event 传递进去，这样就可以使用 pageX 和
pageY 来进行定位了。

Now the tooltip offers a more intelligent suggestion when the hovered row is already highlighted.

现在当我们把鼠标移动到已经高亮的行的时候，显示的提示信息就更加智能化了。

Collapsing and Expanding

收缩与展开

When large sets of data are grouped in tables, as each year's set of articles are in our News page,
collapsing, or hiding, a section's contents can be a convenient way to get a broad view of all of the table's
data without having to scroll so much.

当一大堆数据在表格内进行分组显示的时候，比如新闻页面上按照年份来对文章进行分组，这种
情况下收缩或隐藏一些内容对于表格中内容的显示就会更加方便，我们也不用总是来卷动屏幕
了。

To make the sections of the news article table collapsible, we first prepend a minussymbol image to each
subheading row's first cell. The image is inserted with JavaScript, because if JavaScript is not available
for the row collapsing, the image might confuse those who expect clicking on it to actually trigger some

kind of event：

```
$(document).ready(function() {
var toggleMinus = '../icons/bullet_toggle_minus.png';
var togglePlus = '../icons/bullet_toggle_plus.png';
var $subHead = $('tbody th:first-child');
$subHead.prepend('<img src="' + toggleMinus + '" alt="collapse
this section" />');
});
```

为了让表格中的数据能够进行收缩，我们先给每个子页眉行的第一个单元格准备了一个负号的图
片。这个图片使用 JavaScript 插入，因为如果 JavaScript 不能够用于行的收缩时，这个图片就无
效，也就不会让用户产生疑惑，不会出现用户点击了图片而什么都没有发生的情况。。

```
$(document).ready(function() {
var toggleMinus = '../icons/bullet_toggle_minus.png';
var togglePlus = '../icons/bullet_toggle_plus.png';
var $subHead = $('tbody th:first-child');
$subHead.prepend('<img src="' + toggleMinus + '" alt="collapse
this section" />');
});
```

Note that we set variables for the location of both a minussymbol and a plussymbol image. This way we

can change the image's src attribute when the image is clicked and the rows are collapsed or expanded.

请注意我们给负号图片和正号图片都设置了相应的变量。这样一来我们就可以在点击图片和表格

行进行收缩和展开时来修改图片的路径。

Next we use the .addClass() method to make the newly created images appear clickable:

```
$(document).ready(function() {
var toggleMinus = '../icons/bullet_toggle_minus.png';
var togglePlus = '../icons/bullet_toggle_plus.png';
var $subHead = $('tbody th:first-child');
$subHead.prepend('<img src="' + toggleMinus + '" alt="collapse this section" />');
$('img', $subHead).addClass('clickable');
});
```

接下来我们使用了.addClass()方法让新建立的图片能够被点击。

```
$(document).ready(function() {
var toggleMinus = '../icons/bullet_toggle_minus.png';
var togglePlus = '../icons/bullet_toggle_plus.png';
var $subHead = $('tbody th:first-child');
```

```
$subHead.prepend('<img src="' + toggleMinus + '" alt="collapse
this section" />');
$('img', $subHead).addClass('clickable');
});
```

Finally, we can add code inside a .click() method to do the collapsing and expanding. A condition will
check the current value of the clicked image's src attribute. If it equals the file path represented by the
toggleMinus variable, then all of the other <tr> elements within the same <tbody> will be hidden, and
the src attribute will be set to the value of the togglePlus variable. Otherwise, these <tr> elements will be
shown and the src will change back to the value of toggleMinus:

```
$(document).ready(function() {
var toggleMinus = '../icons/bullet_toggle_minus.png';
var togglePlus = '../icons/bullet_toggle_plus.png';
var $subHead = $('tbody th:first-child');
$subHead.prepend('<img src="' + toggleMinus + '" alt="collapse
this section" />');
$('img', $subHead).addClass('clickable')
.click(function() {
var toggleSrc = $(this).attr('src');
```

```
if ( toggleSrc == toggleMinus ) {
$(this).attr('src', togglePlus)
.parents('tr').siblings().fadeOut('fast');
```

```
} else{
$(this).attr('src', toggleMinus)
.parents('tr').siblings().fadeIn('fast');
};
});
})
```

最终，我们在.click()方法中加入代码来实现收缩和展开。代码中间会检验当前被点击的图片的路径。如果它等于变量 toggleMinus 的值，那么在当前<tbody>内所有<tr>都会被隐藏起来，而图片路径会被设置成 togglePlus 的值。否则，这些<tr>元素就会显示出来而图片途径会被设置成 toggleMinus 的值。

```
$(document).ready(function() {
var toggleMinus = '../icons/bullet_toggle_minus.png';
var togglePlus = '../icons/bullet_toggle_plus.png';
var $subHead = $('tbody th:first-child');
$subHead.prepend('<img src="' + toggleMinus + '" alt="collapse
this section" />');
$('img', $subHead).addClass('clickable')
.click(function() {
```

```
var toggleSrc = $(this).attr('src');
if ( toggleSrc == toggleMinus ) {
$(this).attr('src', togglePlus)
.parents('tr').siblings().fadeOut('fast');
} else{
$(this).attr('src', toggleMinus)
.parents('tr').siblings().fadeIn('fast');
};
});
})
```

With this code in place, clicking on the minus-symbol image next to 2007 makes the table look like this:

加入这段代码之后，点击靠近 2007 的负号图片，就会收缩表格：

The 2007 news articles aren't removed; they are just hidden until we click the plussymbol image that
now appears in that row.
2007 年内的文章并没有被移出，它们只是被隐藏了，只要我们点击现在出现的加号图片它们就能
够显示出来。
Table rows present particular obstacles to animation, since browsers use different values (table-row and
block) for their visible display property. The .hide() and .show() methods, without animation, are always
safe to use with table rows. As of jQuery version 1.1.3, .fadeIn() and .fadeOut() can be used as well.
表格的行给动态显示带了一些困难，因为各个浏览器给它们的显示属性使用了不同的值（tablerow
和 block）。.hide()和.show()方法没有动态显示效果，和表格行一起搭配使用永远都是安全
的。到了 jQuery 1.1.3 中，也可以使用.fadeIn()和.fadeOut()。
Filtering
过滤
Earlier we examined sorting and paging as techniques for helping users focus on relevant portions of a

table's data. We saw that both could be implemented either with serverside technology or with
JavaScript. Filtering completes this arsenal of data arrangement strategies. By displaying to the user only
the table rows that match a given criterion, we can strip away needless distractions.

前面我们使用了排序和分页来帮助用户对表格中的数据作处理。我们可以看到这两种技术都可以
在服务器端或是使用 JavaScript 来实现。过滤功能是对排序和分页的补充，它完善页面上数据处
理的功能。用户给定一个条件，就会只显示出符合条件的数据行，过滤掉那些不需要的行。

We have already seen how to perform a type of filter, highlighting a set of rows. Now we will extend this
idea to actually hiding rows that don't match the filter.

我们已经知道了通过行的高亮如何实现过滤。现在我们要扩展这个功能，来隐藏那些不符合过滤
条件的数据行。

We can begin by creating a place to put our filter buttons. In typical fashion, we insert these controls
using JavaScript so that people without scripting available do not see the options:

```
$(document).ready(function() {
$('table.filterable').each(function() {
var $table = $(this);
$table.find('th').each(function (column) {
if ($(this).is('.filter-column')) {
var $filters = $('<div class="filters"><h3>Filter by ' +
$(this).text() + ':</h3></
div>');
```

```
$filters.insertBefore($table);
}
});
});
});
```

我们先从创建过滤按钮开始。像前面的做法一样，我们使用 JavaScript 来创建这些控件，这样如果有用户禁用了 JavaScript 就不会看到这些控件。

```
$(document).ready(function() {
$('table.filterable').each(function() {
var $table = $(this);
$table.find('th').each(function (column) {
if ($(this).is('.filter-column')) {
var $filters = $('<div class="filters"><h3>Filter by ' +
$(this).text() + ':</h3></
div>');
$filters.insertBefore($table);
}
});
});
});
```

We get the label for the filter box from the column headers, so that this code can be reused for other

tables quite easily. Now we have a heading awaiting some buttons:

我们要从列的页眉中得到过滤所需要的标签，这样其他的表格就能复用这段代码。现在我们的页

眉需要加入一些按钮。

Filter Options

过滤条件

Now we can move on to actually implementing a filter. To start with, we will add filters for a couple of

known topics. The code for this is quite similar to the author highlighting example from before:

```
var keywords = ['conference', 'release'];
$.each(keywords, function (index, keyword) {
$('<div class="filter"></div>').text(keyword)
.bind('click', {'keyword': keyword}, function(event) {
$table.find('tbody tr').each(function() {
if ($('td', this).filter(':nth-child(' + (column + 1) + ')').text()
==
event.data['keyword']) {
$(this).show();
}
else if ($('th',this).length == 0){
$(this).hide();
}
});
$(this).addClass('active').siblings().removeClass('active');
```

```
}).addClass('clickable').appendTo($filters);
});
```

现在我们就可以继续往下实现过滤器了。首先要给一些已知的 Topic 加上过滤器。过滤器的代码
和行的高亮非常相像。

```
var keywords = ['conference', 'release'];
$.each(keywords, function (index, keyword) {
$('<div class="filter"></div>').text(keyword)
.bind('click', {'keyword': keyword}, function(event) {
$table.find('tbody tr').each(function() {
if ($('td', this).filter(':nth-child(' + (column + 1) + ')').text()
==
event.data['keyword']) {
$(this).show();
}
else if ($('th',this).length == 0){
$(this).hide();
}
});
```

```
 $(this).addClass('active').siblings().removeClass('active');
}).addClass('clickable').appendTo($filters);
});
```

Starting with a static array of keywords to filter by, we loop through and create a button for each. Just as in the paging example, we need to use the data parameter of .bind() to avoid accidental closure problems. Then, in the click handler, we compare each cell against the keyword and hide the row if there is no match. We must check whether the row is a subheader, to avoid hiding those in the process.

从用于过滤的关键字静态数组开始，我们进行了循环操作并给每个关键字创建了一个按钮。就如同分页的例子一样，我们需要使用.bind()的参数来避免意外的闭包问题。然后，在 click 事件中，我们将每个单元格和关键字进行比较，如果不匹配的话就隐藏这一行。我们必须要检验这一行是不是子页眉，以防止在处理的时候把子页眉也隐藏了。

Both of the buttons now work as advertised:

现在两个按钮可以正常工作了：

Collecting Filter Options from Content

从内容中收集过滤条件

Now we need to expand the filter options to cover the range of available topics in the table. Rather than hard-coding all of the topics, we can gather them from the text that has been entered in the table. We can change the definition of keywords to read:

```
var keywords = {};
$table.find('tbody tr td').filter(':nth-child(' + (column + 1) + ')').each(function() {
keywords[$(this).text()] = $(this).text();
});
```

现在我们需要扩展过滤条件以适用于表格中所有的 Topic。不用给所有的 Topic 编程，我们可以把它们从表格中收集起来。我们可以定义 keywords 数组来保存。

```
var keywords = {};
$table.find('tbody tr td').filter(':nth-child(' + (column + 1) + ')').each(function() {
keywords[$(this).text()] = $(this).text();
});
```

This code relies on two tricks:

By using a map rather than an array to hold the keywords as they are found, we eliminate duplicates automatically.

jQuery's $.each() function lets us operate on arrays and maps identically, so no later code has to change. Now we have a full complement of filter options:

这段代码有两个小技巧：

使用了图而不是数组来保存找到的关键字，我们可以消除重复的关键字。

jQuery 的$.each()函数能够让我们像操作数组那样操作图，所以不需要修改后面的代码。现在

我们对于过滤条件就有了一个完整的组件：

Reversing the Filters

还原过滤

For completeness, we need a way to get back to the full list after we have filtered it. Adding an option for all topics is pretty straightforward:

```
$('<div class="filter">all</div>').click(function() {
$table.find('tbody tr').show();
$(this).addClass('active').siblings().removeClass('active');
```

```
}).addClass('clickable active').appendTo($filters);
```

为了功能的完整，我们需要在过滤完成之后，加上还原所有数据的功能。直接加上一个显示所有

内容的按钮是最直接了当的。

```
$('<div class="filter">all</div>').click(function() {
$table.find('tbody tr').show();
$(this).addClass('active').siblings().removeClass('active');
}).addClass('clickable active').appendTo($filters);
```

This gives us an all button that simply shows all rows of the table again. For good measure we mark it as

active to begin with.

这样我们就能看到一个 all 的按钮，用来显示表格中的所有行。作为额外的功能我们把它一开始设

置成活动的。

Interacting with Other Code

和其他的代码整合

We learned with our sorting and paging code that we can't treat the various features we write as islands.

The behaviors we build can interact in sometimes surprising ways; for this reason, it is worth revisiting

our earlier efforts to examine how they coexist with the new filtering capabilities we have added.

从排序和分页的代码中我们知道了，不能把写出来的功能当成孤立的东西。我们做出来的功能有

时候会通过一些非常奇怪的方式互相影响；因为这个原因，我们需要回顾一下前面的代码，来检

验一下它们是不是能够和新加入的过滤功能共存。

Row Striping

行条纹

The advanced row striping we put in place earlier is confused by our new filters. Since the tables are not

re-striped after a filter is performed, rows retain their coloring as if the filtered rows were still present.

由于加入了过滤功能，之前的行条纹的功能就会受到影响。由于表格在过滤完成之后没有重新编

排条纹，所以在排序完成后，每个行就会保持原来的颜色。

To account for the filtered rows, the striping code needs to be able to find them. We can add a class on the rows when they are filtered:

```
$(document).ready(function() {
$('table.filterable').each(function() {
var $table = $(this);
$table.find('th').each(function (column) {
if ($(this).is('.filter-column')) {
var $filters = $('<div class="filters"><h3>Filter by ' +
$(this).text() + ':</h3></div>');
var keywords = {};
$table.find('tbody tr td').filter(':nth-child(' + (column + 1) +
')')
.each(function() {
keywords[$(this).text()] = $(this).text();
});
$('<div class="filter">all</div>').click(function() {
$table.find('tbody tr').show().removeClass('filtered');
$(this).addClass('active').siblings().removeClass('active');
$table.trigger('stripe');
}).addClass('clickable active').appendTo($filters);
```

```
$.each(keywords, function (index, keyword) {
$('<div class="filter"></div>').text(keyword)
.bind('click', {'keyword': keyword}, function(event) {
$table.find('tbody tr').each(function() {
if ($('td', this).filter(':nth-child(' + (column + 1) + ')').text()
== event.data['keyword']) {
$(this).show().removeClass('filtered');
}
else if ($('th',this).length == 0) {
$(this).hide().addClass('filtered');
}
});
$(this).addClass('active').siblings().removeClass('active');
$table.trigger('stripe');
}).addClass('clickable').appendTo($filters);
});
$filters.insertBefore($table);
}
```

```
 });
});
});
```

为了处理过滤之后的行，设置行背景色的代码要能够找到它们。我们可以给被过滤的行加上一个

class。

```
$(document).ready(function() {
$('table.filterable').each(function() {
var $table = $(this);
$table.find('th').each(function (column) {
if ($(this).is('.filter-column')) {
var $filters = $('<div class="filters"><h3>Filter by ' +
$(this).text() + ':</h3></div>');
var keywords = {};
$table.find('tbody tr td').filter(':nth-child(' + (column + 1) +
')')
.each(function() {
keywords[$(this).text()] = $(this).text();
});
$('<div class="filter">all</div>').click(function() {
$table.find('tbody tr').show().removeClass('filtered');
```

```
$(this).addClass('active').siblings().removeClass('active');
$table.trigger('stripe');
}).addClass('clickable active').appendTo($filters);
$.each(keywords, function (index, keyword) {
$('<div class="filter"></div>').text(keyword)
.bind('click', {'keyword': keyword}, function(event) {
$table.find('tbody tr').each(function() {
if ($('td', this).filter(':nth-child(' + (column + 1) + ')').text()
== event.data['keyword']) {
$(this).show().removeClass('filtered');
}
else if ($('th',this).length == 0) {
$(this).hide().addClass('filtered');
}
});
$(this).addClass('active').siblings().removeClass('active');
$table.trigger('stripe');
}).addClass('clickable').appendTo($filters);
});
$filters.insertBefore($table);
}
});
```

```
});
});
```

Whenever the current filter changes, we trigger the stripe event. This uses the same trick we implemented when making our pager aware of sorting—adding a new custom event. We have to rewrite the striping code to define this event:

```
$(document).ready(function() {
$('table.striped').each(function() {
$(this).bind('stripe', function() {
var rowIndex = 0;
$('tbody tr:not(.filtered)', this).each(function(index) {
if ($('th',this).length) {
$(this).addClass('subhead');
rowIndex = -1;
```

```
 } else {
if (rowIndex % 6 < 3) {
$(this).removeClass('odd').addClass('even');
}
else {
$(this).removeClass('even').addClass('odd');
}
};
rowIndex++;
});
});
$(this).trigger('stripe');
});
});
```

当前的过滤发生改变的时候，我们就触发 stripe 事件。这跟我们之前在排序的时候加上分页是所
使用的技巧是一样的——增加一个新的自定义事件。我们需要重写设置条纹背景的代码来定义这
个事件。

```
$(document).ready(function() {
$('table.striped').each(function() {
```

```
$(this).bind('stripe', function() {
var rowIndex = 0;
$('tbody tr:not(.filtered)', this).each(function(index) {
if ($('th',this).length) {
$(this).addClass('subhead');
rowIndex = -1;
} else {
if (rowIndex % 6 < 3) {
$(this).removeClass('odd').addClass('even');
}
else {
$(this).removeClass('even').addClass('odd');
}
};
rowIndex++;
});
});
$(this).trigger('stripe');
});
});
```

The selector to find table rows now skips filtered rows. We also must remove obsolete classes from

rows, as this code may now be executed multiple times. With both the new event handler and its triggers

in place, the filtering operation respects row striping:

找到表格行的选择符现在就跳过了被过滤的行。我们也必须给这些行出去不需要的 class，因为这
段代码可能需要多次执行。加入了新的事件处理和触发器之后，过滤的操作就和行条纹可以一起
正常工作了。

Expanding and Collapsing

展开与收缩

The expanding and collapsing behavior added earlier also conflicts with our filters. If a section is
collapsed and a new filter is chosen, then the matching items are displayed, even if in the collapsed
section. Conversely, if the table is filtered and a section is expanded, then all items in the expanded
section are displayed regardless of whether they match the filter.

前面加入的展开与收缩的功能和过滤的功能有冲突。如果一部份行被收缩，而一个新的过滤操作
被执行了，那么就连被收缩的部分的符合过滤匹配的数据也都显示出来了。相反，如果表格执行
了过滤而一部分数据被展开，那么这一块里所有的数据都会显示出来而不管它们是否匹配过滤条
件。

Since we have added the filtered class to all rows when they are removed by a filter button, we can

check for this class inside our collapser's click handler:

```
var toggleSrc = $(this).attr('src');
if ( toggleSrc == toggleMinus ) {
$(this).attr('src', togglePlus)
.parents('tr').siblings().addClass('collapsed').fadeOut('fast');
} else{
$(this).attr('src', toggleMinus)
.parents('tr').siblings().removeClass('collapsed') .not('.filtered')
.fadeIn('fast');
};
```

由于我们给所有被过滤的行加上了过滤的 class，我们可以在收缩的操作中对这个 class 进行检验。

```
var toggleSrc = $(this).attr('src');
if ( toggleSrc == toggleMinus ) {
```

```
 $(this).attr('src', togglePlus)
.parents('tr').siblings().addClass('collapsed').fadeOut('fast');
} else{
$(this).attr('src', toggleMinus)
.parents('tr').siblings().removeClass('collapsed') .not('.filtered')
.fadeIn('fast');
};
```

While we are collapsing or expanding rows, we add or remove another new class on the rows. We need this class to solve the other half of the problem. The filtering code can use the class to ensure that a row should be shown when the filter changes:

```
$table.find('tbody tr').each(function() {
if ($('td', this).filter(':nth-child(' + (column + 1) + ')').text()
== e.data['keyword']) {
$(this).removeClass('filtered').not('.collapsed').show();
}
else if ($('th',this).length == 0) {
$(this).addClass('filtered').hide();
}
});
```

当我们收缩或者展开行的时候，就会给这些行加上或者出去一个新的 class。我们要使用这个 class 去解决另外一半问题。过滤的代码可以使用这个 class 来确保在过滤发生改变的时候某一行是否要显示出来。

```
$table.find('tbody tr').each(function() {
if ($('td', this).filter(':nth-child(' + (column + 1) + ')').text()
== e.data['keyword']) {
$(this).removeClass('filtered').not('.collapsed').show();
}
else if ($('th',this).length == 0) {
$(this).addClass('filtered').hide();
}
});
```

Now our features play nicely, each able to hide and show the rows independently.

现在我们的功能都能正常使用了，表格行可以正常的隐藏和现实。

The Finished Code

最终代码

Our second example page has demonstrated table row striping, highlighting, tooltips, collapsing/ expanding, and filtering. Taken together, the JavaScript code for this

page is:

```
$(document).ready(function() {
var highlighted = "";
var column = 3;
var positionTooltip = function(event) {
var tPosX = event.pageX;
var tPosY = event.pageY + 20;
$('div.tooltip').css({top: tPosY, left: tPosX});
};
var showTooltip = function(event) {
$('div.tooltip').remove();
var $thisAuthor = $(this).text();
if ($(this).parent().is('.highlight')) {
highlighted = 'un-';
} else {
highlighted = '';
};
$('<div class="tooltip">Click to ' + highlighted +
```

```
 'highlight all articles written by ' +
$thisAuthor + '</div>').appendTo('body');
positionTooltip(event);
};
var hideTooltip = function() {
$('div.tooltip').remove();
};
$('table.striped td:nth-child(' + column + ')' ).addClass('clickable').click(function(event) {
var thisClicked = $(this).text();
$('table.striped td:nth-child(' + column + ')' )
.each(function(index) {
if (thisClicked == $(this).text()) {
$(this).parent().toggleClass('highlight');
} else {
$(this).parent().removeClass('highlight');
};
})
showTooltip.call(this, event);
})
.hover(showTooltip, hideTooltip)
```

```
.mousemove(positionTooltip);
});
$(document).ready(function() {
$('table.striped').each(function() {
$(this).bind('stripe', function() {
var rowIndex = 0;
$('tbody tr:not(.filtered)', this).each(function(index) {
if ($('th',this).length) {
$(this).addClass('subhead');
rowIndex = -1;
} else {
if (rowIndex % 6 < 3) {
$(this).removeClass('odd').addClass('even');
}
else {
$(this).removeClass('even').addClass('odd');
}
}
rowIndex++;
});
});
$(this).trigger('stripe');
```

```
});
})
$(document).ready(function() {
$('table.filterable').each(function() {
var $table = $(this);
$table.find('th').each(function (column) {
if ($(this).is('.filter-column')) {
var $filters = $('<div class="filters"><h3>Filter by ' +
$(this).text() + ':</h3></div>');
var keywords = {};
$table.find('tbody tr td').filter(':nth-child(' + (column + 1) + ')')
```

```
 .each(function() {
keywords[$(this).text()] = $(this).text();
})
$('<div class="filter">all</div>').click(function() {
$table.find('tbody
tr').removeClass('filtered') .not('.collapsed').show();
$(this).addClass('active').siblings().removeClass('active');
$table.trigger('stripe');
}).addClass('clickable active').appendTo($filters);
$.each(keywords, function (index, keyword) {
$('<div class="filter"></div>').text(keyword)
.bind('click', {'keyword': keyword}, function(event) {
$table.find('tbody tr').each(function() {
if ($('td', this).filter(':nth-child(' + (column + 1) + ')').text()
== event.data['keyword']) {
$(this).removeClass('filtered').not('.collapsed') .show();
}
else if ($('th',this).length == 0) {
$(this).addClass('filtered').hide();
}
});
```

```
$(this).addClass('active').siblings().removeClass( 'active');
$table.trigger('stripe');
}).addClass('clickable').appendTo($filters);
});
$filters.insertBefore($table);
}
});
});
});
$(document).ready(function() {
var toggleMinus = '../icons/bullet_toggle_minus.png';
var togglePlus = '../icons/bullet_toggle_plus.png';
var $subHead = $('tbody th:first-child');
$subHead.prepend('<img src="' + toggleMinus + '" alt="collapse
this section" />');
$('img', $subHead).addClass('clickable')
.click(function() {
var toggleSrc = $(this).attr('src');
if ( toggleSrc == toggleMinus ) {
$(this).attr('src', togglePlus)
.parents('tr').siblings().addClass('collapsed').fadeOut('fast');
} else {
```

```
$(this).attr('src', toggleMinus)
.parents('tr').siblings().removeClass('collapsed') .not('.filtered')
.show().fadeIn('fast');
};
});
})
```

我们的第二个示例的页面演示了表格交替行背景色、高亮、提示信息、收缩与展开、过滤。综合了所有功能的 JavaScript 代码如下：

```
$(document).ready(function() {
var highlighted = "";
var column = 3;
var positionTooltip = function(event) {
var tPosX = event.pageX;
var tPosY = event.pageY + 20;
```

```
 $('div.tooltip').css({top: tPosY, left: tPosX});
};
var showTooltip = function(event) {
$('div.tooltip').remove();
var $thisAuthor = $(this).text();
if ($(this).parent().is('.highlight')) {
highlighted = 'un-';
} else {
highlighted = '';
};
$('<div class="tooltip">Click to ' + highlighted +
'highlight all articles written by ' +
$thisAuthor + '</div>').appendTo('body');
positionTooltip(event);
};
var hideTooltip = function() {
$('div.tooltip').remove();
};
$('table.striped td:nth-child(' + column +
')' ).addClass('clickable').click(function(event) {
var thisClicked = $(this).text();
```

```
$('table.striped td:nth-child(' + column + ')' )
.each(function(index) {
if (thisClicked == $(this).text()) {
$(this).parent().toggleClass('highlight');
} else {
$(this).parent().removeClass('highlight');
};
})
showTooltip.call(this, event);
})
.hover(showTooltip, hideTooltip)
.mousemove(positionTooltip);
});
$(document).ready(function() {
$('table.striped').each(function() {
$(this).bind('stripe', function() {
var rowIndex = 0;
$('tbody tr:not(.filtered)', this).each(function(index) {
if ($('th',this).length) {
$(this).addClass('subhead');
rowIndex = -1;
} else {
```

```
if (rowIndex % 6 < 3) {
$(this).removeClass('odd').addClass('even');
}
else {
$(this).removeClass('even').addClass('odd');
}
}
rowIndex++;
});
});
$(this).trigger('stripe');
});
```

```
 })
$(document).ready(function() {
$('table.filterable').each(function() {
var $table = $(this);
$table.find('th').each(function (column) {
if ($(this).is('.filter-column')) {
var $filters = $('<div class="filters"><h3>Filter by ' +
$(this).text() + ':</h3></div>');
var keywords = {};
$table.find('tbody tr td').filter(':nth-child(' + (column + 1) +
')')
.each(function() {
keywords[$(this).text()] = $(this).text();
})
$('<div class="filter">all</div>').click(function() {
$table.find('tbody
tr').removeClass('filtered') .not('.collapsed').show();
$(this).addClass('active').siblings().removeClass('active');
$table.trigger('stripe');
}).addClass('clickable active').appendTo($filters);
$.each(keywords, function (index, keyword) {
```

```
$('<div class="filter"></div>').text(keyword)
.bind('click', {'keyword': keyword}, function(event) {
$table.find('tbody tr').each(function() {
if ($('td', this).filter(':nth-child(' + (column + 1) + ')').text()
== event.data['keyword']) {
$(this).removeClass('filtered').not('.collapsed') .show();
}
else if ($('th',this).length == 0) {
$(this).addClass('filtered').hide();
}
});
$(this).addClass('active').siblings().removeClass( 'active');
$table.trigger('stripe');
}).addClass('clickable').appendTo($filters);
});
$filters.insertBefore($table);
}
});
});
});
$(document).ready(function() {
var toggleMinus = '../icons/bullet_toggle_minus.png';
```

```
var togglePlus = '../icons/bullet_toggle_plus.png';
var $subHead = $('tbody th:first-child');
$subHead.prepend('<img src="' + toggleMinus + '" alt="collapse
this section" />');
$('img', $subHead).addClass('clickable')
.click(function() {
var toggleSrc = $(this).attr('src');
if ( toggleSrc == toggleMinus ) {
$(this).attr('src', togglePlus)
.parents('tr').siblings().addClass('collapsed').fadeOut('fast');
} else {
$(this).attr('src', toggleMinus)
.parents('tr').siblings().removeClass('collapsed') .not('.filtered')
.show().fadeIn('fast');
```

```
  };
});
})
```

Summary

总结

In this chapter, we have explored some of the ways to slice and dice the tables on our sites, reconfiguring
them into beautiful and functional containers for our data. We have covered sorting data in tables, using
different kinds of data (words, numbers, dates) as sort keys along with paginating tables into easilyviewed
chunks. We have learned sophisticated row striping techniques and JavaScript-powered tooltips.
We have also walked through expanding and collapsing as well as filtering and highlighting of rows that
match the given criteria.

在这一章中，我们研究了对于表格的操作的一些方式，而且根据数据显示做了界面和功能优化。
我们在表格中做了排序处理，使用了不同的数据类型来作为排序的键，还使用了分页的功能。我
们也学习了复杂的交替行背景色的设置以及使用了 JavaScript 来实

现提示信息。我们同样也实现

了展开与收缩数据行，还有过滤和高亮显示数据行。

We've even touched briefly on some quite advanced topics, such as sorting and paging with server-side

code and AJAX techniques, dynamically calculating page coordinates for elements, and writing a jQuery

plug-in.

我们甚至已经接触到了一些很高级的话题了，例如服务器端的排序和分页、AJAX 技术、动态配

置元素属性和编写 jQuery 插件。

As we have seen, properly semantic HTML tables wrap a great deal of subtlety and complexity in a

small package. Fortunately, jQuery can help us easily tame these creatures, allowing the full power of

tabular data to come to the surface.

正如我们已经看到的，适当在表格中使用 HTML 能够包含大量的奇妙和复杂的东西。很幸运

jQuery 能够帮助我们很容易就掌握这些特性，能够将表格数据的全部特点展现出来。

Appendix A Online Resources

附录 A 在线资源

I can't remember what I used to know. Somebody help me now and let me go —Devo, "Deep Sleep"

The following online resources represent a starting point for learning more about jQuery, JavaScript, and
web development in general, beyond what is covered in this book. There are far too many sources of
quality information on the web for this appendix to approach anything resembling an exhaustive list.
Furthermore, while other print publications can also provide valuable information, they are not noted
here.

下面的在线资源为我们找到了一个起点，能够帮助我们呢学习更多关于 jQuery、JavaScript 以及这
本书中还没有提及的 Web 开发的内容。对于这篇附录来说，网络上有太多的信息了，要列出一个
详细没有遗漏的清单是不可能的。此外，因为其他的一些印刷出版物也能够提供一些有价值的信
息，在这里并没有把他们收录其中。

jQuery Documentation

jQuery 文档

jQuery Wiki

jQuery Wiki

The documentation on jquery.com is in the form of a wiki, which means that the content is editable by

the public. The site includes the full jQuery API, tutorials, getting started guides, a plug-in repository,

and more:

http://docs.jquery.com/

Jquery.com 上的文档采用了 wiki 的形式,这就意味着任何人都可以修改编辑这上面的内容。官方

网站上收录了完整的 jQuery API、jQuery 指南、新手指南和一个插件资源库,更多的内容请访

问:

http://docs.jquery.com/。

jQuery API

jQuery API

On jQuery.com, the API is available in two locations — the documentation section and the paginated API

browser.

在 jQuery.com 上,jQuery 的 API 可以在两个地方看到——jQuery 文档部分和 API Browser。

The documentation section of jQuery.com includes not only jQuery methods, but also all of the jQuery

selector expressions:

http://docs.jquery.com/Selectors

http://docs.jquery.com/

http://jquery.com/api

jQuery.com 的文档部分不仅仅收录了 jQuery 的方法，还收录了所有的 jQuery 的选择符表达式

(Selector Expression):

http://docs.jquery.com/Selectors

http://docs.jquery.com/

http://jquery.com/api

jQuery API Browser

jQuery API Browser

J.rn Zaeferrer has put together a convenient tree-view browser of the jQuery API with a search feature

and alphabetical or categorical sorting:

http://jquery.bassistance.de/api-browser/

J.rn Zaeferrer 把所有的 jQuery API 收集在一个非常方便直观的树型结构浏览器内，并且提供了搜

索功能，以及按字母顺序和分类类型排序:

http://jquery.bassistance.de/api-browser/

Visual jQuery

可视化的 jQuery

This API browser designed by Yehuda Katz is both beautiful and convenient. It also provides quick

viewing of methods for a number of jQuery plug-ins:

http://www.visualjquery.com/

由 Yehuda Katz 设计的 API Browser 是既美观又方便的。它也提供了对大量的 jQuery 插件的快速访

问：

http://www.visualjquery.com/

Web Developer Blog

Sam Collet keeps a master list of jQuery documentation, including downloadable versions and cheat

sheets, on his blog:

http://webdevel.blogspot.com/2007/01/jquery-documentation.html

Sam Collet 保留了一份有关 jQuery 的主要清单，包括可供下载的各个版本和速记表格在他的 Blog

上：

http://webdevel.blogspot.com/2007/01/jquery-documentation.html

JavaScript Reference

JavaScript 参考

Mozilla Developer Center

Mozilla 开发者中心

This site has a comprehensive JavaScript reference, a guide to programming with JavaScript, links to

helpful tools, and more:

http://developer.mozilla.org/en/docs/JavaScript/

这个站点有一份全面的 JavaScript 参考，使用 JavaScript 进行编程的指南和一些有用的工具的链

接，更多内容请访问：

http://developer.mozilla.org/en/docs/JavaScript/

Dev.Opera

Dev.Opera

While focused primarily on its own browser platform, Opera's site for web developers includes a

number of useful articles on JavaScript:

http://dev.opera.com/articles/

虽然 Opera 主要关注与它自己的浏览器平台，不过它为 Web 开发人员提供的社区内有许多关于

JavaScript 的有用的文章：

http://dev.opera.com/articles/

Quirksmode

Quirks 模式

Peter-Paul Koch's Quirksmode site is a terrific resource for understanding differences in the way

browsers implement various JavaScript functions, as well as many CSS properties:

http://www.quirksmode.org/

Peter-Paul Koch 的 Quirksmode 的网站是一个非常令人恐怖的资源，它会让你理解不同浏览器在实

现 JavaScript 函数和 CSS 属性的工作方式上的差异：

http://www.quirksmode.org/

JavaScript Toolbox

JavaScript 工具箱

Matt Kruse's JavaScript Toolbox offers a large assortment of homespun JavaScript libraries, as well as

sound advice on JavaScript best practices and a collection of vetted JavaScript resources elsewhere on

the Web:

http://www.javascripttoolbox.com/

Matt Kruse 的 JavaScript 工具箱提供了一个庞大的 JavaScript 库的分类，可以看作是 JavaScript 最优方

法的第一选择，以及经过测试确定有效的 JavaScript 的资源的集合，

具体请访问：

http://www.javascripttoolbox.com/

JavaScript Code Compressors

JavaScript 代码压缩工具

Packer

Packer

This JavaScript compressor/obfuscator by Dean Edwards is used to compress the jQuery source code.

It's available as a web-based tool or as a free download. The resulting code is very efficient in file size, at

a cost of a small increase in execution time:

http://dean.edwards.name/packer/

http://dean.edwards.name/download/#packer

这个由 Dean Edwards 开发的 JavaScript 压缩工具是用来压缩 jQuery 源代码的。它既可以作为一个在

线的工具，也可以通过下载免费得到。经过处理的代码在文件大小上可以看到很明显的压缩效

果，而这样换来的代价只是会增加少许的执行时间：

http://dean.edwards.name/packer/

http://dean.edwards.name/download/#packer

JSMin

JSMin

Created by Douglas Crockford, JSMin is a filter that removes comments and unnecessary white space

from JavaScript files. It typically reduces file size by half, resulting in faster downloads:

http://www.crockford.com/javascript/jsmin.html

由 Douglas Crockford 开发的 JSMin 是一个能够过滤 JavaScript 源文件中的注释和多余的空格的工

具。它的特点是能够将文件大小变为原来的一半，让文件下载变得更快：

http://www.crockford.com/javascript/jsmin.html

Pretty Printer

Pretty Printer

This tool prettifies JavaScript that has been compressed, restoring line breaks and indentation where

possible. It provides a number of options for tailoring the results:

http://www.prettyprinter.de/

这个工具能够将被压缩过的 JavaScript 代码还原，如果原处有换行符和缩进的话会恢复它们。对

于还原的结果的格式，它为用户提供了很多选择。

(X)HTML Reference

(X)HTML 参考

W3C Hypertext Markup Language Home Page

W3C 超文本标记语言主页

The World Wide Web Consortium (W3C) sets the standard for (X)HTML, and the HTML home page is a great launching point for its specifications and guidelines:

http://www.w3.org/MarkUp/

W3C 组织设立了(X)HTML 标准，如果要了解它的规范和获得它的指南，请访问

HTML (Hypertext Markup Language，超文本标记语言）的主页：

http://www.w3.org/MarkUp/

CSS Reference

CSS 参考

W3C Cascading Style Sheets Home Page

W3C 层叠样式表主页

The W3C's CSS home page provides links to tutorials, specifications, test suites, and other resources:

http://www.w3.org/Style/CSS/

W3C 的 CSS（Cascading Style Sheets，层叠样式表）主页提供了新手指南、CSS 规范、测试组件以

及其他的资源：

http://www.w3.org/Style/CSS/

Mezzoblue CSS Cribsheet

Mezzoblue CSS Cribsheet

Dave Shea provides this helpful CSS cribsheet in an attempt to make the design process easier, and

provides a quick reference to check when you run into trouble：

http://mezzoblue.com/css/cribsheet/

Dave Shea 提供了一份非常有用的 CSS cribsheet，力图使设计过程更加简单，并且当你碰到问题的

时候能够作为一个快捷的参考：

http://mezzoblue.com/css/cribsheet/

Position Is Everything

Position Is Everything

This site includes a catalog of CSS browser bugs along with explanations of how to overcome them：

http://www.positioniseverything.net/

这个站点收录了一份目录，内容包括众多浏览器下的 CSS 的问题，以及如何解决它们：

http://www.positioniseverything.net/

XPath Reference

XPath 参考

W3C XML Path Language Version 1.0 Specification

W3C XML 定位语言 1.0 版规范

Although jQuery's XPath support is limited, the W3C's XPath Specification may still be useful for those

wanting to learn more about the variety of possible XPath selectors:

http://www.w3.org/TR/xpath

尽管 jQuery's 的 XPath 支持是有限的，但是 W3C 的 XPath 规范还是能够对那些想要学习更多关于

XPath 选择符的人们帮上一些忙：

http://www.w3.org/TR/xpath

TopXML XPath Reference

TopXML XPath 参考

The TopXML site provides helpful charts of axes, node tests, and functions for those wanting to learn

more about XPath:

http://www.topxml.com/xsl/XPathRef.asp

TopXML 的站点提供了非常有用的轴和节电测试的图表，还为那些想学习更多 XPath 知识的人们提

供了一些函数：

http://www.topxml.com/xsl/XPathRef.asp

MSDN XPath Reference

MSDN XPath 参考

The Microsoft Developer Network website has information on XPath syntax and functions:

http://msdn2.microsoft.com/en-us/library/ms256115.aspx

MSDN (Microsoft Developer Network) 站点有一些关于 XPath 语法和函数的内容：

http://msdn2.microsoft.com/en-us/library/ms256115.aspx

Useful Blogs

有用的 Blog

The jQuery Blog

jQuery 官方 Blog

John Resig and other contributors to the official jQuery blog posts announcements about new versions

and other initiatives among the project team, as well as occasional tutorials and editorial pieces.

http://jquery.com/blog/

John Resig 和其他 jQuery 官方 Blog 的投稿者们发布一些项目组内关于新版本和其他动态的公告，

也会有新手指南和社论：

http://jquery.com/blog/

Learning jQuery

学习 jQuery

Karl Swedberg, Jonathan Chaffer, Brandon Aaron, et al. are running a blog for jQuery tutorials,

examples, and announcements:

http://www.learningjquery.com/

Karl Swedberg, Jonathan Chaffer, Brandon Aaron 等人开通了一个 Blog，放上了一些 jQuery 的新手指

南、例子和公告：

http://www.learningjquery.com/

Jack Slocum's Blog

Jack Slocum 的 Blog

Jack Slocum, the author of the popular EXT suite of JavaScript components, writes about his work and

JavaScript programming in general：

http://www.jackslocum.com/blog/

Jack Slocum，JavaScript 的一个非常流行的组件 EXT 的作者，在这里记述他的工作和 JavaScript 的

开发：

http://www.jackslocum.com/blog/

Web Standards with Imagination

想象中的 Web 标准

Dustin Diaz's blog features articles on web design and development, with an emphasis on JavaScript：

http://www.dustindiaz.com/

Dustin Diaz 的 Blog 主要记载了关于 Web 设计和开发的文章，尤其是强调了 JavaScript：

http://www.dustindiaz.com/

Snook

Snook

Jonathan Snook's general programming/web-development blog：

http://snook.ca/

Jonathan Snook 的关于编程和 Web 开发的 Blog：

http://snook.ca/

I Can't

I Can't

Three sites by Christian Heilmann provide blog entries, sample code, and lengthy articles related to

JavaScript and web development:

http://icant.co.uk/

http://www.wait-till-i.com/

http://www.onlinetools.org/

Christian Heilmann 的三个站点都跟 JavaScript 和 Web 开发有关，一个是 Blog，一个收录了示范代

码，一个则都是冗长的文章：

http://icant.co.uk/

http://www.wait-till-i.com/

http://www.onlinetools.org/

DOM Scripting

DOM 脚本编程

Jeremy Keith's blog picks up where the popular DOM scripting book leaves off—a fantastic resource for

unobtrusive JavaScript:

http://domscripting.com/blog/

Jeremy Keith 的 Blog 收录了最受欢迎的 DOM 编程的书籍，无须多说明，绝对是 JavaScript 的一个梦

幻般的资源：

http://domscripting.com/blog/

As Days Pass By

As Days Pass By

Stuart Langridge experiments with advanced use of the browser DOM：

http://www.kryogenix.org/code/browser/

Stuart Langridge 的有关浏览器 DOM 高级运用的实验：

http://www.kryogenix.org/code/browser/

A List Apart

A List Apart

A List Apart explores the design, development, and meaning of web content, with a special focus on web

standards and best practices：

http://www.alistapart.com/

A List Apart 探究了 Web 内容的设计、开发和意义，特别关注与 Web 标准和最优方法：

http://www.alistapart.com/

Particletree

Particletree

Chris Campbell, Kevin Hale, and Ryan Campbell started a blog that provides valuable information on

many aspects of web development:

http://particletree.com/

Chris Campbell, Kevin Hale 和 Ryan Campbell 开通了一个 Blog，收录了很多 Web 开发各方面的有用

的信息：

http://particletree.com/

The Strange Zen of JavaScript

Scott Andrew LePera's weblog about JavaScript quirks, caveats, odd hacks, curiosities and collected

wisdom. Focused on practical uses for web application development:

http://jszen.blogspot.com/

Scott Andrew LePera 的关于 JavaScript 的各种各样技巧的 Blog。这个 Blog 关注于 Web 应用程序开发

的实际运用：

http://jszen.blogspot.com/

Web Development Frameworks Using jQuery

使用 jQuery 的 Web 开发框架

As developers of open-source projects become aware of jQuery, many are incorporating the JavaScript

library into their own systems. The following is a brief list of some of the early adopters:

当开源项目的开发者们开始知道 jQuery 时，许多人把 JavaScript 库整合到他们自己的系统中。下面是一些早期的框架：

Drupal: http://drupal.org/

Joomla Extensions: http://extensions.joomla.org/

Pommo: http://pommo.org/

SPIP: http://www.spip.net/

Textpattern: http://textpattern./

Trac: http://trac.edgewall.org/

WordPress: http://wordpress.org/

For a more complete list, visit the Sites Using jQuery page at: http://docs.jquery.com/Sites_Using_jQuery

如果需要更加详细的清单，请访问 Sites Using jQuery：

http://docs.jquery.com/Sites_Using_jQuery

Appendix B Development Tools

附录 B 开发工具

When a problem comes alongYou must whip it —Devo, "Whip It"

Documentation can help in troubleshooting issues with our JavaScript applications, but there is no

replacement for a good set of software development tools. Fortunately, there are many software

packages available for inspecting and debugging JavaScript code, and most of them are available for

free.

文档可以帮助我们在进行 JavaScript 编程中处理一些问题，但是，没有什么东西能够替代开发工

具在编程中的地位。幸运的是，有很多软件插件和扩展包可以检查并调试 JavaScript 代码，并且

这些插件绝大多数都是免费的。

Tools for Firefox

FireFox 工具

Mozilla Firefox is the browser of choice for the lion's share of web developers, and therefore has some

of the most extensive and well-respected development tools.

Mozilla Firefox 是强悍的 Web 开发人员对于浏览器的第一选择，因此，

它有一些扩展性非常好并

且评价非常不错的开发工具。

Firebug

Firebug

The Firebug extension for Firefox is indispensable for jQuery development：

http://www.getfirebug.com/

Some of the features of Firebug are ：

An excellent DOM inspector for finding names and selectors for pieces of the document

CSS manipulation tools for finding out why a page looks a certain way and changing it

An interactive JavaScript console

A JavaScript debugger that can watch variables and trace code execution

FireFox 的 Firebug 插件对于 jQuery 开发来说是不可或缺的：

http://www.getfirebug.com/

Firebug 的特色有以下几点：

非常卓越的 DOM 搜索器，用于找到页面内元素的名称和选择符。

CSS 操作工具，用于找出页面

交互的 JavaScript 控制台

能够监视变量和追踪代码执行的 JavaScript 调试器

Web Developer Toolbar

Web 开发者工具条

This not only overlaps Firebug in the area of DOM inspection, but also contains tools for common tasks

like cookie manipulation, form inspection, and page resizing. You can also use this toolbar to quickly

and easily disable JavaScript for a site to ensure that functionality degrades gracefully when the user's

browser is less capable:

http://chrispederick.com/work/web-developer/

这套工具并不只是和 Firebug 在 DOM 搜索的功能上有交集，而且还包含了通常任务的工具，例如

Cookie 操作，表单搜索和页面大小调整。当用户的浏览器性能不够好时，通过使用这个工具条，

你可以快速而方便的禁用网站的 JavaScript 以确保性能不会很快地下降。

http://chrispederick.com/work/web-developer/

Venkman

Venkman

Venkman is the official JavaScript debugger for the Mozilla project. It provides a troubleshooting

environment that is reminiscent of the GDB system for debugging

programs that are written in other

languages.

http://www.mozilla.org/projects/venkman/

Venkman 是 Mozilla 项目的官方 JavaScript 调试器。它提供了一个调试环境，这个环境是 GDB 系统的

前身，能够调试用其他语言编写的程序：

http://www.mozilla.org/projects/venkman/

Regular Expressions Tester

正则式测试器

Regular expressions for matching strings in JavaScript can be tricky to craft. This extension for Firefox

allows easy experimentation with regular expressions using an interface for entering search text:

http://sebastianzartner.ath.cx/new/downloads/RExT/

JavaScript 中匹配字符串的正则式是一门非常讲究技巧的技术。FireFox 的这个扩展插件提供了一

个能够输入搜索字符串的界面来进行正则式的测试。

http://sebastianzartner.ath.cx/new/downloads/RExT/

Tools for Internet Explorer

IE 工具

Sites often behave differently in IE than in other web browsers, so having debugging tools for this

platform is important.

在 IE 中，站点的外观看起来和在其他的浏览器下经常有所不同，所以拥有 IE 下的调试工具也是非

常重要的。

Microsoft Internet Explorer Developer Toolbar

微软 IE 开发者工具条

The Developer Toolbar primarily provides a view of the DOM tree for a web page. Elements can be

located visually, and modified on the fly with new CSS rules. It also provides other miscellaneous

development aids, such as a ruler for measuring page elements:

http://www.microsoft.com/downloads/details.aspx?FamilyID=e59c

3964-672d-4511-

bb3e-2d5e1db91038

微软 IE 开发者工具条主要提供了对于网页中 DOM 元素的树形浏览的功能。所有元素都可以进行

可视化的显示，并且用新的 CSS 规则修改。同样它还提供了其他各种各样的开发援助，比如定位

页面元素的标尺。

http://www.microsoft.com/downloads/details.aspx?FamilyID=e59c

3964-672d-4511-

bb3e-2d5e1db91038

Microsoft Visual Web Developer

微软可视化 Web 开发者

Microsoft’s Visual Studio package can be used to inspect and debug JavaScript code：

http://msdn.microsoft.com/vstudio/express/vwd/

To run the debugger interactively in the free version (Visual Web Developer Express), follow the process

outlined here：

http://www.berniecode.com/blog/2007/03/08/how-to-debug-javascript-with-visual-web-developerexpress/

微软的 Visual Studio 可以用于检查和调试 JavaScript 代码：

http://msdn.microsoft.com/vstudio/express/vwd/

如果要在免费版本中交互式地运行调试器，请按照下面步骤：

http://www.berniecode.com/blog/2007/03/08/how-to-debug-javascript-with-visual-web-developerexpress/

DebugBar

DebugBar

The DebugBar provides a DOM inspector as well as a JavaScript console for debugging：

http://www.debugbar.com/

DebugBar 提供了 DOM 搜索器，和 JavaScript 控制台一样用于调试：

http://www.debugbar.com/

Drip

Drip

Memory leaks in JavaScript code can cause performance and stability issues for Internet Explorer. Drip

helps to detect and isolate these memory issues:

http://Sourceforge.net/projects/ieleak/

To learn more about a common cause of Internet Explorer memory leaks, see Appendix C, JavaScript

Closures.

JavaScript 代码中的内存溢出会造成 IE 的性能和稳定性出现问题。Drip 能够找出并隔离这些危险代

码。

http://Sourceforge.net/projects/ieleak/

如果要学习更多关于 IE 内存溢出的原因，请参照附件 C 部分 JavaScript 闭包

Tools for Safari

Safari 工具

Safari remains the new kid on the block as a development platform, but there are still tools available for

situations in which code behaves differently in this browser than elsewhere.

Safari 是近来新兴的开发平台和浏览器，但是仍然有很多工具用于处理 JavaScript 代码在这个浏览

器和其他浏览器有不同表现的情况。

Web Inspector

Web Inspector

Nightly builds of Safari include the ability to inspect individual page elements and collect information

especially about the CSS rules that apply to each one.

http://trac.webkit.org/projects/webkit/wiki/Web%20Inspector

这个工具能够搜索独立页面的元素还能够收集页面内的信息，尤其是每个元素所应用的 CSS 规

则。

http://trac.webkit.org/projects/webkit/wiki/Web%20Inspector

Drosera

Drosera

Drosera is the JavaScript debugger for Safari and other WebKit-driven applications. It enables

breakpoints, variable watching, and an interactive console.

http://trac.webkit.org/projects/webkit/wiki/Drosera

Drosera 是 Safari 和其他的 WebKit 驱动的应用程序的 JavaScript 调

试器。它支持断点、变量监视和交

互式控制台。

http://trac.webkit.org/projects/webkit/wiki/Drosera

Other Tools

其他工具

Firebug Lite

Firebug Lite

Though the Firebug extension itself is limited to the Firefox web browser, some of the features can be

replicated by including the Firebug Lite script on the web page. This package simulates the Firebug

console, including allowing calls to console.log() to work in all browsers and not raise JavaScript errors:

http://www.getfirebug.com/lite.htmlDevelopment Tools [340]

尽管 Firebug 插件只能用于 FireFox 浏览器，但是它的一些特点能够在通过在页面中使用 Firebug

Lite 脚本体现出来。这个工具模拟了 Firebug 控制台，允许调用 console.log()函数运行在所有的浏览

器中并且不会引起 JavaScript 报错。

http://www.getfirebug.com/lite.htmlDevelopment Tools [340]

TextMate jQuery Bundle

TextMate jQuery 包

This extension for the popular Mac OS X text editor TextMate provides syntax highlighting for jQuery

methods and selectors, code completion for methods, and a quick API reference from within your code.

The bundle is also compatible with the E text editor for Windows:

http://www.learningjquery.com/2006/09/textmate-bundle-for-jquery

Mac 操作系统上非常流行的文本编辑器 TextMate 的这个插件提供了 jQuery 方法和选择符的动态高

亮、代码的自动完成功能和快速的 API 参考。这个包同样也适用于 Windows 下的 E 文本编辑器。

http://www.learningjquery.com/2006/09/textmate-bundle-for-jquery

Charles

Charles

When developing AJAX-intensive applications, it can be useful to see exactly what data is being sent

between the browser and the server. The Charles web debugging proxy displays all HTTP traffic

between two points, including normal web requests, HTTPS traffic, Flash remoting, and AJAX

responses:

http://www.xk72.com/charles/

当开发高强度 AJAX 应用程序时，这个工具可以详细地看到有些什么数据在浏览器和服务器之前

传送。Charles 的网络调试代理服务器显示出两点之间的所有 HTTP 通信，包括普通的 Web 请

求、HTTPS 通信，远程刷新和 AJAX 回应。

http://www.xk72.com/charles/

Aptana

Aptana

This Java-based web development IDE is free and cross-platform. Along with both standard and

advanced code editing features, it incorporates a full copy of the jQuery API documentation.

http://www.aptana.com/

这个基于 Java 的 Web 开发 IDE 是免费的而且是跨平台的。除了有标准代码编辑和高级代码编辑的

特征外，它还包含了一份完整的 jQueryAPI 文档。

http://www.aptana.com/

Appendix C JavaScript Closures

附录 C JavaScript 闭包

Let's close our eyes togetherNow can you see how good it's going to be? —Devo, "Pink Jazz Trancers"

Throughout this book, we have seen many jQuery methods that take functions as parameters. Our examples have thus created, called, and passed around functions time and again. While usually we can do this with only a cursory understanding of the inner JavaScript mechanics at work, at times side effects of our actions can seem strange if we do not have knowledge of the language features. In this appendix, we will study one of the more esoteric (yet prevalent) types of functions, called closures.

贯穿整本书，我们已经看到了许多将函数作为参数的 jQuery 的方法。我们的例子一直围绕着函数的建立、调用和传递。但是我们常常只是对于 JavaScript 内在工作机制有一个浅显的了解，有时候如果我们没有关于 JavaScript 语言特性的相关知识，我们的操作产生的副作用汇看起来非常奇怪。在这篇附录里，我们将学习一个更加深奥而又流行的函数的类型，

称为闭包。

Inner Functions

内部函数

JavaScript is fortunate to number itself among the programming languages that support inner function

declarations. Many traditional programming languages, such as C, collect all functions in a single toplevel

scope. Languages with inner functions, on the other hand, allow us to gather small utility functions

where they are needed, avoiding namespace pollution.

很幸运 JavaScript 是众多支持内部函数声明的语言中的一员。很多传统的编程语言，比如 C，将所

有的函数都置于最高级的作用域中。从另一方面来说，支持内部函数的语言能够让我们在需要的

地方放置那些小型而有用的函数，从而避免了命名空间的资源浪费。。

An inner function is simply a function that is defined inside of another function. For example:

```
function outerFun() {
function innerFun() {
alert('hello');
}
}
```

内部函数就是在函数内部定义的函数，例如：

```
function outerFun() {
function innerFun() {
alert('hello');
}
}
```

The innerFun() is an inner function, contained within the scope of outerFun(). This means that a call to innerFun() is valid within outerFun(), but not outside of it. The following code results in a JavaScript error:

```
function outerFun() {
function innerFun() {
alert('hello');
}
}
innerFun();
```

innerFun()就是一个内部函数，包含在 outerFun()的作用域中。这意味着在 outerFun()里调用 innerFun()是可行的，而不是在 outerFun()外面调用。下面的代码会导致 JavaScript 错误：

```
function outerFun() {
```

```
function innerFun() {

alert('hello');

}
```

```
}
innerFun();
```

We can trigger the alert, though, by calling innerFun() from within outerFun():

```
function outerFun() {
function innerFun() {
alert('hello');
}
innerFun();
}
outerFun();
```

通过调用 outerFun()来调用 innerFun()，我们可以触发 alert 对话框：

```
function outerFun() {
function innerFun() {
alert('hello');
}
innerFun();
}
outerFun();
```

This technique is especially handy for small, single-purpose functions. For example, algorithms that are

recursive but have a non-recursive API wrapper are often best expressed with an inner function as a
helper.

这个技巧对于那些小型的、单一的函数来说是非常便利的。比如，那些没有递归 API 的递归算法
就经常用内部函数的方法来处理。

The Great Escape

转义

The plot thickens when function references come into play. Some languages, such as Pascal, do allow
the use of inner functions for the purpose of code hiding, and those functions are forever entombed
within their parent functions. JavaScript, on the other hand, allows us to pass functions around just as if
they were any other kind of data. This means inner functions can escape their captors.

The escape route can wind in many different directions. For example, suppose the function is assigned to
a global variable:

```
var globVar;
function outerFun() {
function innerFun() {
```

```
alert('hello');
}
globVar = innerFun;
}
outerFun();
globVar();
```

当函数引用引入到程序中的时候，程序的结构就变得不一样了。有些语言，比如 Pascal，允许使

用内部函数达到代码隐藏的目的，而这些函数在它们的父函数内就永远成为了坟墓一样的东西。

从另一方面来说，JavaScript 允许我们将函数作为另外一种类型的数据来进行传递。这意味着内

部函数可以进行转义。

转义的方式可以有很多种。比如，假设函数被赋给一个全局变量：

```
var globVar;
function outerFun() {
function innerFun() {
alert('hello');
}
globVar = innerFun;
}
```

```
outerFun();
globVar();
```

The call to outerFun() after the function definition modifies the global variable globVar. It is now a reference to innerFun(). This means that the later call to globVar() operates just as an inner call to innerFun() would, and the alert is displayed. Note that a call to innerFun() from outside of outerFun() still results in an error! Though the function has escaped by way of the reference stored in the global variable, the function name is still trapped inside the scope of outerFun().

A function reference can also find its way out of a parent function through a return value:

```
function outerFun() {
function innerFun() {
alert('hello');
}
return innerFun ;
var globVar = outerFun();
globVar();
```

在函数 outerFun()的调用之后，全局变量 globVar 的值被修改。现在它是指向 innerFun()的引用。这
意味着后面调用 globVar()的操作就相当于内部调用 innerFun()，而且 alert 对话框会出现。请注意在
outerFun()的外面调用 innerFun()是会引发错误的。尽管已经通过全局变量保存引用对函数进行了
转义，但是函数的名称还是只在 outerFun()的作用域内有效。

还可以通过父函数的返回值来操作函数引用：

```
function outerFun() {
function innerFun() {
alert('hello');
}
return innerFun ;
var globVar = outerFun();
globVar();
```

Here, there is no global variable modified inside outerFun(). Instead, outerFun() returns a reference to
innerFun(). The call to outerFun() results in this reference, which can be stored and called itself in turn,
triggering the alert again.

在这里，outerFun()内没有修改任何全局变量。相反的是，outerFun()返回了对 innerFun()的引用。

对 outerFun()的调用返回了这个引用，这样就能够保存并调用这个引用，触发 alert 对话框。

The fact that inner functions can be invoked through a reference even after the function has gone out of

scope means that JavaScript needs to keep referenced functions available as long as they could possibly

be called. Each variable that refers to the function is tracked by the JavaScript runtime, and once the last

has gone away the JavaScript garbage collector comes along and frees up that bit of memory.

函数可以通过引用在作用域的外部进行调用的这个事实，意味着一旦被引用的函数需要调

用，JavaScript 就要把这些函数一直保持着。每个引用了函数的变量都被 JavaScript 运行时环境监

视着，一旦不再调用函数，JavaScript 垃圾回收器会开始工作并释放该变量占用的内存。

Variable Scoping

变量作用域

Inner functions can of course have their own variables, which are restricted in scope to the function

itself:

```
function outerFun() {
```

```
function innerFun() {
var innerVar = 0;
innerVar++;
alert(innerVar);
}
```

```
return innerFun;
```

内部函数也有属于自己的变量，这些变量都限制在内部函数的作用域内。

```
function outerFun() {
function innerFun() {
var innerVar = 0;
innerVar++;
alert(innerVar);
}
return innerFun;
```

Each time the function is called, through a reference or otherwise, a new variable innerVar is created, incremented, and displayed:

```
var globVar = outerFun();
globVar(); // Alerts "1"
globVar(); // Alerts "1"
var innerVar2 = outerFun();
innerVar2(); // Alerts "1"
innerVar2(); // Alerts "1"
```

不管是通过引用或是其他方法调用函数，总是有一个新的变量 innerVar 被创建，自增并显示出

来。

```
var globVar = outerFun();
globVar(); // 显示"1"
globVar(); // 显示"1"
var innerVar2 = outerFun();
innerVar2(); // 显示"1"
innerVar2(); // 显示"1"
```

Inner functions can reference global variables, in the same way as any other function can:

```
var globVar = 0;
function outerFun() {
function innerFun() {
globVar++;
alert(globVar);
}
return innerFun;
}
```

内部函数可以引用全局变量，同样其他函数也可以。

```
var globVar = 0;
function outerFun() {
function innerFun() {
globVar++;
```

```
alert(globVar);
}
return innerFun;
}
```

Now our function will consistently increment the variable with each call:

```
var globVar = outerFun();
globVar(); // Alerts"1"
globVar(); // Alerts "2"
var globVar2 = outerFun();
globVar2(); // Alerts "3"
globVar2(); // Alerts "4"
```

现在每次调用函数的时候，显示的变量都会自增。

```
var globVar = outerFun();
```

```
globVar(); // 显示"1"
globVar(); // 显示"2"
var globVar2 = outerFun();
globVar2(); // 显示"3"
globVar2(); // 显示"4"
```

But what if the variable is local to the parent function? Since the inner function inherits its parent's scope, this variable can be referenced too:

```
function outerFun() {
var outerVar = 0;
function innerFun() {
outerVar++;
alert(outerVar);
}
return innerFun;
}
```

但是如果变量是父函数的局部变量又会怎么样呢？因为内部函数是在父函数的作用域内，所以这个变量也可以被引用。

```
function outerFun() {
var outerVar = 0;
```

```
function innerFun() {
outerVar++;
alert(outerVar);
}
return innerFun;
}
```

Now our function calls have more interesting behavior:

```
var globVar = outerFun();
globVar(); // Alerts "1"
globVar(); // Alerts "2"
var globVar2 = outerFun();
globVar2(); // Alerts "1"
globVar2(); // Alerts "2"
```

现在函数的调用有了更有趣的表现。

```
var globVar = outerFun();
globVar(); // 显示"1"
globVar(); // 显示"2"
var globVar2 = outerFun();
globVar2(); // 显示"1"
globVar2(); // 显示"2"
```

We get a mix of the two earlier effects. The calls to innerFun() through each reference increment

innerVar independently. Note that the second call to outerFun() is not resetting the value of innerVar, but
rather creating a new instance of innerVar, bound to the scope of the second function call. The upshot of
this is that after the above calls, another call to globVar() will alert 3, and a subsequent call to globVar2()
will also alert 3. The two counters are completely separate.
我们得到了前面两种效果的组合。每次通过引用调用 innerFun()就分别给 innerVar 进行了自增操
作。注意第二次调用 outerFun()时没有重置 innerVar，而是在第二次函数调用时的作用域内重新创
建了 innerVar 的实例。结果就是在前两次调用之后，再次调用 globVar()就会输出 3，而后面并发的
调用 globVar2()也会输出 3。这两种情况下的计数都是完全独立的。
When a reference to an inner function finds its way outside of the scope in which the function was
defined, this creates a closure on that function. We call variables that are not local to the inner function
free variables, and the environment of the outer function call closes them. Essentially, the fact that the

function refers to a local variable in the outer function grants the variable a stay of execution. The
memory is not released when the function completes, as it is still needed by the closure.

当一个内部函数的引用发现在该函数定义的作用域外面进行了函数的调用时，将会创建一个该函
数的“闭包”。我们把那些不是内部函数的局部变量的变量称为“自由变量”，而外部函数调用时的
环境和这些变量是隔离的。本质上，在外部函数中函数引用局部变量会延缓对该变量的内存的回
收。。当函数结束的时候内存还没有释放，因为这块内存还要用到。

Interactions between Closures

闭包间的交互

When more than one inner function exists, closures can have effects that are not as easy to anticipate.

Suppose we pair our incrementing function with another function, this time incrementing by two:

```
function outerFun() {
var outerVar = 0;
function innerFun() {
outerVar++;
```

```
alert(outerVar);
}
function innerFun2() {
outerVar = outerVar + 2;
alert(globVar);
}
return {'innerFun': innerFun, 'innerFun2': innerFun2};
}
```

当有一个以上的内部函数时，闭包产生的效果可能没有预期所想像的那样简单。假设我们把自增函数和另外一个每次让变量自增 2 的函数放在一起。

```
function outerFun() {
var outerVar = 0;
function innerFun() {
outerVar++;
alert(outerVar);
}
function innerFun2() {
outerVar = outerVar + 2;
alert(globVar);
}
return {'innerFun': innerFun, 'innerFun2': innerFun2};
```

```
}
```

We return references to both functions, using a map to do so (this illustrates another way in which

reference to an inner function can escape its parent). Both functions can be called through the references:

```
var globVar = outerFun();
globVar.innerFun(); // Alerts "1"
globVar.innerFun2(); // Alerts "3"
globVar.innerFun(); // Alerts "4"
var globVar2 = outerFun();
globVar2.innerFun(); // Alerts "1"
globVar2.innerFun2(); // Alerts "3"
globVar2.innerFun(); // Alerts "4"
```

我们使用了一个 Map 返回了两个函数的引用（这个描述了将内部函数的引用从父函数中进行转义

的另外一种方法。）。两个函数都通过饮用来进行调用。

```
var globVar = outerFun();
globVar.innerFun(); // 显示"1"
globVar.innerFun2(); // 显示"3"
globVar.innerFun(); // 显示"4"
var globVar2 = outerFun();
```

globVar2.innerFun(); // 显示"1"

globVar2.innerFun2(); // 显示"3"

globVar2.innerFun(); // 显示"4"

The two inner functions refer to the same local variable, so they share the same closing environment.

When innerFun() increments outerVar by 1, this sets the new starting value of outerVar when

innerFun2() is called. Once again, though, we see that a subsequent call to outerFun() creates new

instances of these closures with a new closing environment to match. Fans of object-oriented

programming will note that we have in essence created a new object, with the free variables acting as

instance variables and the closures acting as instance methods. The variables are also private, as they

cannot be directly referenced outside of their enclosing scope, enabling true object-oriented data privacy.

这两个内部函数引用了相同的局部变量，所以它们有同一个闭包环境。当 innerFun()给 outerVar 自

增 1 的时候，就是给 innerFun2()调用时的 outerVar 设定了初始值。尽管，我们看到了对 outerFun()的

并发的调用，创建了一个新的闭包环境，但是面向对象编程的爱好者们会注意到，实际上我们已
经创建了一个新的对象，这个对象拥有属于自己的内部变量和方法。这些变量都是私有化的，因
为它们不可能在作用域外面被直接引用，这就是真正的面向对象的数据私有化。

Closures in jQuery

jQuery 中的闭包

The methods we have seen throughout the jQuery library often take at least one function as a parameter.

For convenience, we often use anonymous functions so that we can define the function behavior right
when it is needed. This means that functions are rarely in the top-level namespace; they are usually inner
functions, which means they can quite easily become closures.

我们在 jQuery 库中常见到的方法常常至少把一个函数作为参数。为了方便，我们常常使用匿名函
数以便于在需要的时候我们能够正确定义函数的行为。这意味着函数很少都会在最高层的命名空
间中，它们通常都是内部函数，这就意味着可以很容易地使用闭包。

Arguments to $(document).ready()

$(document).ready()的参数

Nearly all of the code we write using jQuery ends up getting placed inside a function as an argument to $(document).ready(). We do this to guarantee that the DOM has loaded before the code is run, which is usually a requirement for interesting jQuery code. When a function is created and passed to .ready(), a reference to the function is stored as part of the global jQuery object. This reference is then called at a later time, when the DOM is ready.

在我们所有用 jQuery 编写的代码中，都会把一个函数作为$(document).ready()的参数。我们这么做是为了确保在代码运行前 DOM 已经加载完成，这常常是 jQuery 代码让人关注的条件之一。当一个函数被创建并传递给.ready()，一个指向这个函数的引用就被当作是 jQuery 全局对象的一部分保存起来了。当 DOM 完成加载之后，就会调用这个引用。

We usually place the $(document).ready() construct at the top level of the code structure, so this function is not really a closure. However, since our code is usually written inside this function, everything else is an inner function:

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

```
var readyVar = 0;
function outerFun() {
function innerFun() {
readyVar++;
alert(readyVar);
return innerFun;
}
var readyVar2 = outerFun();
readyVar2();
```

});

我们常常把$(document).ready()放在代码的最上层，所以这个函数不是一个真正的闭包。然而，

因为我们的代码经常是写在函数内的，所以所有的东西都成了内部函数。

```
$(document).ready(function() {
var readyVar = 0;
function outerFun() {
function innerFun() {
readyVar++;
alert(readyVar);
return innerFun;
}
var readyVar2 = outerFun();
readyVar2();
});
```

This looks like our global variable example from before, except now it is wrapped in a $

(document).ready() call as so much of our code always is. This means that readyVar is not a global

variable, but a local variable to the anonymous function. The variable

readyVar2 gets a reference to a

closure with readyVar in its environment.

这样看起来像是我们以前的全局变量的例子，有所不同的和我们大多数的代码一样它被置于$

(document).ready()的调用内。这意味着，readyVar 就不是一个全局变量了，对于这个匿名函数来

说是一个局部变量。变量 readyVar2 得到了

The fact that most jQuery code is inside a function body is useful, because this can protect against some

namespace collisions. For example, it is this feature that allows us to use jQuery.noConflict() to free up

the $ shortcut for other libraries, while still being able to define the shortcut locally for use within $

(document).ready().

jQuery 代码存在于函数体内的事实是非常有用的，因为这能够避免命名空间的冲突。例如，我们

能够使用 jQuery.noConflict()来释放出$快捷方式供其他库使用，但是也可以在$(document).ready()

内定义快捷方式来使用。

Event Handlers

事件处理器

The $(document).ready() construct usually wraps the rest of our

code, including the assignment of event

handlers. Since handlers are functions, they become inner functions and since those inner functions are

stored and called later, they become closures. A simple click handler can illustrate this:

```
$(document).ready(function() {
var readyVar = 0;
$('.trigger').click(function() {
readyVar++;
alert(readyVar);
});
});
```

$(document).ready()通常包含了我们其他的代码，包括事件处理器的任务分配。因为事件处理器

都是函数，所以它们就都是内部函数，也因为这些内部函数都是在后面存储和调用的，它们就是

闭包的。一个快速的 click 事件可以用下面的代码实现。

```
$(document).ready(function() {
var readyVar = 0;
$('.trigger').click(function() {
readyVar++;
lert(readyVar);
```

```
});
```

});

Because the variable readyVar is declared inside of the .ready() handler, it is only available to the jQuery

code inside this block and not to outside code. It can be referenced by the code in the .click() handler,

however, which increments and displays the variable. Because a closure is created, the same instance of

readyVar is referenced each time the button is clicked. This means that the alerts display a continuously

incrementing set of values, not just 1 each time.

因为变量 readyVar 是在.ready()内部声明的，所以它只能在.ready()的内部有效，在外部无效。它可

以被.click()内的代码引用，用于增加变量的值和显示变量。因为闭包已经建立了，所以每次发生

按钮的 click 事件的时候，会引用 readyVar 的同一个实例。这意味着每次显示的值都是不断增加

的，而不是每次都是显示 1。

Event handlers can share their closing environments, just like other functions can:

$(document).ready(function() {

var readyVar = 0;

```
$('.add').click(function() {
readyVar++;
alert(readyVar);
});
$('.subtract').click(function() {
readyVar--;
alert(readyVar);
});
});
```

事件处理器能够能够共享闭包环境，跟其他的函数一样。

```
$(document).ready(function() {
var readyVar = 0;
$('.add').click(function() {
readyVar++;
alert(readyVar);
});
$('.subtract').click(function() {
readyVar--;
alert(readyVar);
});
});
```

Since both of the functions reference the same variable, the

incrementing and decrementing operations

of the two buttons affect the same value rather than being independent.

因为这两个函数都引用了同一个变量，所以这两个按钮的点击会对同一个变量进行增和减的操

作。

These examples have used anonymous functions, as has been our custom in jQuery code. This makes no

difference in the construction of closures. For example, we can write an anonymous function to report

the index of an item within a jQuery object:

```
$(document).ready(function() {
$('li').each(function(index) {
$(this).click(function() {
alert(index);
});
});
});
```

这些例子使用了匿名函数，而且是我们自定义的 jQuery 代码。这和闭包的创建没有区别。例如，

我们可以编写一个匿名函数使用 jQuery 对象来返回一个元素的索引。

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

```
 $('li').each(function(index) {
$(this).click(function() {
alert(index);
});
});
});
```

Because the innermost function is defined within the .each() callback, this code actually creates as many
functions as there are list items. Each of these functions is attached as a click handler to one of the items.
The functions have index in their closing environment, since it is a parameter to the .each() callback.
This behaves the same way as the same code with the click handler written as a named function:

```
$(document).ready(function() {
$('li').each(function(index) {
function clickHandler() {
alert(index);
}
$(this).click(clickHandler);
});
```

```
});
```

因为最内层的函数是使用了.each()的回调进行定义的，所以这部分代码实际上创建了和 li 的元素
数量相等的函数。每个函数都被绑定到每个元素的 click 事件上。因为这些函数是回调的.each()的
一个参数，所以它们在闭包环境中都有索引。在把 click 事件写在一个命名过的函数中，上面的代
码也能产生同样的效果。

```
$(document).ready(function() {
$('li').each(function(index) {
function clickHandler() {
alert(index);
}
$(this).click(clickHandler);
});
});
```

The version with the anonymous function is just a bit shorter. The position of this named function is still
relevant, however:

```
$(document).ready(function() {
$('li').each(function(index) {
function clickHandler() {
```

```
alert(index);
}
$(this).click(clickHandler);
});
});
```

这个使用了匿名函数的版本比较短，但是命名的函数的位置仍然是相关的。

```
$(document).ready(function() {
function clickHandler() {
alert(index);
}
$('li').each(function(index) {
$(this).click(clickHandler);
});
});
```

This version will trigger a JavaScript error whenever a list item is clicked, because index is not found in

the closing environment of clickHandler(). It remains a free variable, and so is undefined in this context.

```
 $(document).ready(function() {
function clickHandler() {
alert(index);
}
$('li').each(function(index) {
$(this).click(clickHandler);
});
});
```

这个版本的代码会在点击一个列表元素的时候引发 JavaScript 错误，因为在 clickHandler()的闭包环境中找不到索引。索引参数是一个自由变量，所以在 clickHandler()的上下文中没有定义。

Memory Leak Hazards

内存溢出的危险

JavaScript manages its memory using a technique known as garbage collection. This is in contrast to low-level languages like C, which require programmers to explicitly reserve blocks of memory and free them when they are no longer being used. Other languages such as Objective-C assist the programmer by implementing a reference counting system, which allows the user

to note how many pieces of the
program are using a particular piece of memory so it can be cleaned up when no longer used. JavaScript
is a high-level language, on the other hand, and generally takes care of this bookkeeping behind the
scenes.
JavaScript 使用众所周知的垃圾回收来管理自己的内存。这和低级的语言，形成了鲜明的对比，
比如 C 就要求程序员们精确的分配内存和进行回收。其他的语言比如 Objective-C 通过实现了引用
统计系统帮助程序员们进行开发，引用统计系统能够帮助我们统计程序使用内存的情况，这样当
这些内存不再被使用的时候就可以进行释放。JavaScript 是一门高级语言，从另一方面来说，比
起表面来，它更关注内在的东西。
Whenever a new memory-resident item such as an object or function comes into being in JavaScript
code, a chunk of memory is set aside for this item. As the object gets passed around to functions and
assigned to variables, more pieces of code begin to point to the object. JavaScript keeps track of these
pointers, and when the last one is gone, the memory taken by the

object is released. Consider a chain of

pointers:

当一个新的内存常驻对象变成 JavaScript 代码的时候，系统会为这个对象分配一块内存。当这个

对象作为参数传给函数或是赋值给变量时，更多的代码块就会指向这个对象。JavaScript 不断追

踪这些指针，当最后一个被销毁时，这个对象占用的内存就被释放。

好比链式指针一样：

Here object A has a property that points to B, and B has a property that points to C. Even if object A here

is the only one that is a variable in the current scope, all three objects must remain in memory because of

the pointers to them. When A goes out of scope, however (such as at the end of the function it was

declared in), then it can be released by the garbage collector. Now B has nothing pointing to it, so can be

released, and finally C can be released as well.

这个对象 A 有一个属性指向对象 B，对象 B 有一个属性指向对象 C。尽管只有 A 是当前作用域内的

变量，但是由于有指针的存在，所有的三个对象都保存在内存中。当 A 在作用域内被销毁时（比

如函数执行结束），它占用的内存就会被垃圾回收器释放。现在没有

任何东西指向 B 了，所以 B

会被释放，最后 C 也被释放。

More complicated arrangements of references can be harder to deal with:

Now we've added a property to object C that refers back to B. In this case, when A is released, B still has
a pointer to it from C. This reference loop needs to be handled specially by JavaScript, which must
notice that the entire loop is isolated from the variables that are in scope.

有时候会碰到更加复杂的引用的处理：

现在我们给 C 增加了一个回指向 B 的属性。在这种情况下，当 A 被释放时，B 还拥有一个从 C 得到
的指针。这种引用循环需要被 JavaScript 特殊处理，尤其需要注意这个循环和作用域中的变量是
相互隔离的。

Accidental Reference Loops

意外的引用循环

Closures can cause reference loops to be inadvertently created. Since functions are objects that must be
kept in memory, any variables they have in their closing environment are also kept in memory:

闭包可能会不经意的引起引用循环。因为函数也是放在内存中的对

象，任何在函数的闭包环境中

的变量也都保存在内存中。

```
function outerFun() {
var outerVar = {};
function innerFun() {
alert(outerVar);
};
outerVar.innerFun = innerFun;
return innerFun;
};
```

Here an object called innerFun is created, and referenced from within the inner function innerFun().

Then a property of outerVar that points to innerFun() is created, and innerFun() is returned. This creates

a closure on innerFun() that refers to innerFun, which in turn refers back to innerFun(). But the loop can

be more insidious than this:

```
function outerFun() {
var outerVar = {};
function innerFun() {
alert(outerVar);
};
```

```
outerVar.innerFun = innerFun;
return innerFun;
};
```

这里建立了一个名叫 innerFun 的对象，在内部函数 innerFun().调用了这个变量。outerVar 的 innerFun 属性得到了 innerFun()的引用，这样就创建了 innerFun()的闭包以及 innerFun 和 innerFun()的循环引用。但是引用循环可能比这个更加危险。

```
 function outerFun() {
var outerVar = {};
function innerFun() {
alert('hello');
};
outerVar.innerFun = innerFun;
return innerFun;
};
```

Here we've changed innerFun() so that it no longer refers to outerVar. However, this does not break the loop. Even though outerVar is never referred to from innerFun(), it is still in innerFun()'s closing environment. All variables in the scope of outerFun() are implicitly referred to by innerFun() due to the closure. So, closures make it easy to accidentally create these loops.

```
function outerFun() {
var outerVar = {};
function innerFun() {
alert('hello');
};
outerVar.innerFun = innerFun;
```

```
return innerFun;
};
```

这里我们修改了 innerFun()，它就不会再引用 outerVar。但是这样做并没有打破引用循环。尽管 outerVar 从来没有在 innerFun()内被引用，但是它还是在 innerFun()的闭包环境内。因为所有在 outerFun()作用域内的变量都被 innerFun()间接的引用。所以，闭包让不经意地产生引用循环变得更加容易。

The Internet Explorer Memory Leak Problem

IE 内存溢出问题

All of this is generally not an issue because JavaScript is able to detect these loops and clean them up when they become orphaned. Internet Explorer, however, has difficulty handling one particular class of reference loops. When a loop contains both DOM elements and regular JavaScript objects, IE cannot release either one because they are handled by different memory managers. These loops are never freed until the browser is closed, which can eat up a great deal of memory over time. A common cause of such a loop is a simple event handler:

```
$(document).ready(function() {
var div = document.getElementById('foo');
div.onclick = function() {
alert('hello');
}
});
```

其实基本上这不能称作是一个问题，因为 JavaScript 能够找出和清除那些孤立的引用循环。然而
IE 在处理一类特殊的引用循环是会有问题。当一个引用循环包含了 DOM 元素和 JavaScript 对象
时，IE 不能够释放任何一个对象，因为这两者是使用不同的内存管理器处理的。不关闭浏览器就
不会释放这些引用循环，因此这可能会占用大量的内存。一个简单的事件处理器可能就会造成这
种引用循环。

```
$(document).ready(function() {
var div = document.getElementById('foo');
div.onclick = function() {
alert('hello');
}
});
```

When the click handler is assigned, this creates a closure with div in

the closing environment. But div

now contains a reference back to the closure, and the resulting loop

can't be released by Internet

Explorer even when we navigate away from the page.

当分配了一个 click 事件处理器，这将创建变量 div 的闭包环境。但是现在 div 包含了对闭包的回

调，因此这样产生的引用循环是不能被 IE 释放的，就算我们打开别的页面，这段内存还是没有释

放。

The Good News

好消息

Now let's write the same code, but using normal jQuery constructs:

```
$(document).ready(function() {
var $div = $('#foo');
$div.click(function() {
alert('hello');
});
});
```

现在我们写下同样的代码，但是使用的是 jQuery 的方式。

```
$(document).ready(function() {
var $div = $('#foo');
$div.click(function() {
alert('hello');
});
});
```

Even though a closure is still created causing the same kind of loop as before, we do not get an IE
memory leak from this code. Fortunately, jQuery is aware of the potential for leaks, and manually
releases all of the event handlers that it assigns. As long as we faithfully adhere to using jQuery event
binding methods for our handlers, we need not fear leaks caused by this particular common idiom.

尽管创建的闭包仍然会引起引用循环，但是我们不会得到 IE 内存溢出的问题了。幸运的
是，jQuery 是知道内存溢出的潜在危险，并会手动释放事先分配好的事件处理器。只要我们一直
使用 jQuery 事件绑定方法，就不需要担心 IE 造成的内存溢出问题。

This doesn't mean we're completely out of the woods; we must continue to take care when we're
performing other tasks with DOM elements. Attaching JavaScript objects to DOM elements can still
cause memory leaks in Internet Explorer; jQuery just helps make this situation far less prevalent.

这并不意味着我们完完全全就是安全的，在处理 DOM 元素的时候我们必须还是要小心。在 IE 下
将 JavaScript 对象绑定到 DOM 元素上仍然可能会引起内存溢出，

jQuery 只会减少这个情况发生的

概率。

Conclusion

结语

JavaScript closures are a powerful language feature. They are often quite useful in hiding variables from

other code, so that we don't tread on variable names being used elsewhere. Due to jQuery's frequent

reliance on functions as method arguments, they can also be inadvertently created quite often.

Understanding them allows us to write more efficient and concise code, and with a bit of care and the

use of jQuery's built-in safeguards we can avoid the memory-related pitfalls they can introduce.

JavaScript 的闭包是非常强大的语言特性。在隐藏代码上闭包是非常有用的，这样我们就不用操

心其他的地方是否有变量重名的情况。由于 jQuery 常常依靠将函数作为方法的参数，闭包往往在

不经意之间就建立起来了。理解闭包能够让我们编写出更多有效和简洁的代码，而且只需要少许

的注意加上 jQuery 内建的安全机制我们能够避免闭包所带来的内存相关的问题。