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

第一章 开始 Jquery

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，jQuery 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。jQuery 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 open-source 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：HTML 文档本身，CSS 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="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8"/>
<title>Through the Looking-Glass</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 Looking-Glass</h1>
<div class="author">by Lewis Carroll</div>
<div class="chapter" id="chapter-1">
<h2 class="chapter-title">1. Looking-Glass 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">"&amp;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="poem-title">YKCOWREBBAJ</h3>
<div class="poem-stanza">
<div>sevot yhtils eht dna ,gillirb sawT'</div>
<div>;ebaw eht ni elbmig dna eryg diD</div>
<div>,sevogorob eht erew ysmim llA</div>
<div>.ebargtuo shtar emom eht dnA</div>
</div>
</div>
<p>She puzzled over this for some time, but at last a bright
thought struck her. <span class="spoken">"Why, it's a
Looking-glass book, of course! And if I hold it up to a
glass, the words will all go the right way again."</span></p>
<p>This was the poem that Alice read.</p>
<div class="poem">
<h3 class="poem-title">JABBERWOCKY</h3>
<div class="poem-stanza">
<div>'Twas brillig, and the slithy toves</div>
<div>Did gyre and gimble in the wabe;</div>
<div>All mimsy were the borogoves,</div>
<div>And the mome raths outgrabe.</div>
</div>
</div>
</div>
</div>
</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。

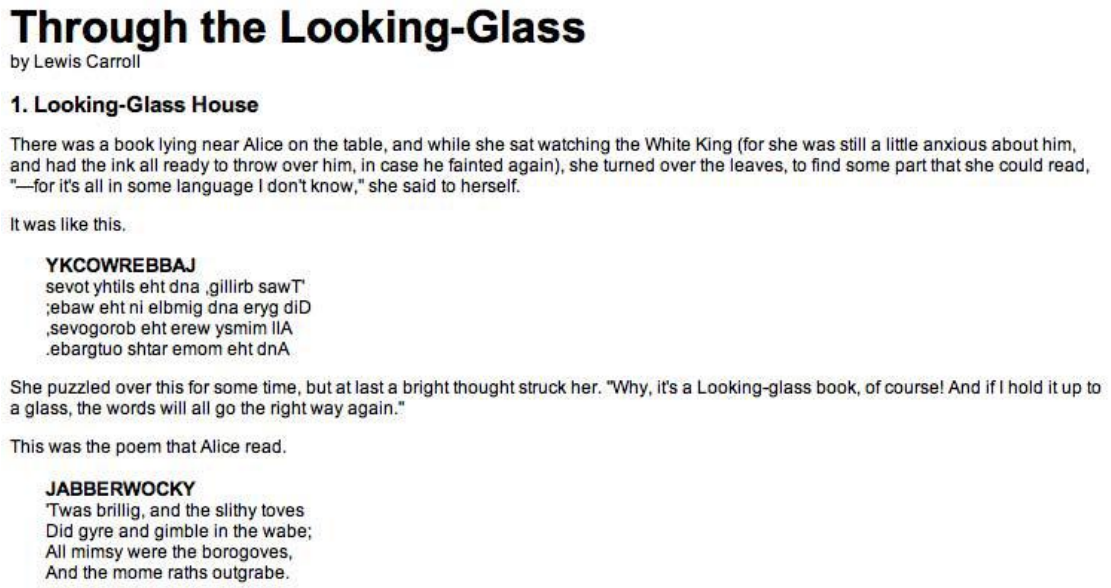
Immediately following the normal HTML preamble, the stylesheet is loaded. For this example, we'll use a Spartan(斯巴达的) one.
马上跟着常规的 HTML 导言，样式表已被加载。

```
body {
font: 62.5% Arial, Verdana, sans-serif;
}
h1 {
font-size: 2.5em;
margin-bottom: 0;
}
h2 {
font-size: 1.3em;
margin-bottom: .5em;
}
h3 {
font-size: 1.1em;
margin-bottom: 0;
}
.poem {
margin: 0 2em;
}
.emphasized {
font-style: 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：



Through the Looking-Glass

by Lewis Carroll

1. Looking-Glass House

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, "—for it's all in some language I don't know," she said to herself.

It was like this.

YKCOWREBBAJ
sevot yhtils eht dna ,gillirb sawT'
;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 Looking-glass 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.

JABBERWOCKY
'Twas brillig, and the slithy toves
Did gyre and gimble in the wabe;
All mimsy were the borogoves,
And the mome raths outgrabe.

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() {
  $('.poem-stanza').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 poem-stanza 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() {
  $('.poem-stanza').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();">
```

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() {
  $('.poem-stanza').addClass('emphasized');
});
```

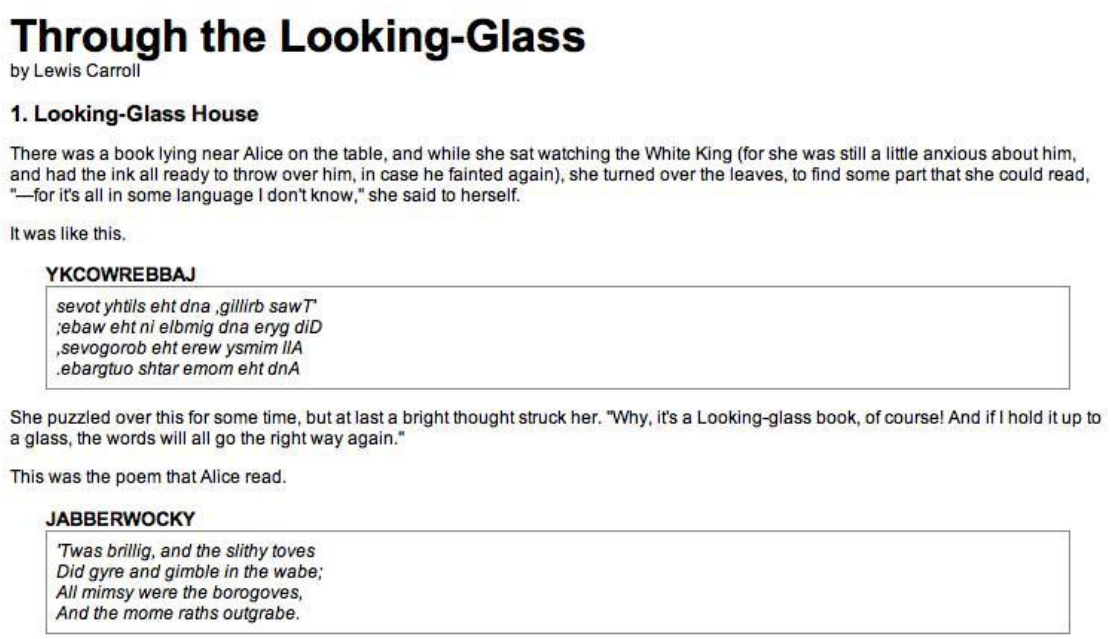
用 function 这个关键词而不是用一个函数名，我们详细地定义了一个我们需要的函数，不

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>
<div>
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
<p>This is yet another paragraph.</p>
</div>
</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。<p> 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: $('#some-id') gets the single element in the document that has the corresponding some-id

ID.
A class: $('.some-class') gets all elements in the document that have a class of some-class.

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="selected-plays">
<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>
<li>Macbeth</li>
<li>Romeo and Juliet</li>
</ul>
</li>
<li>Histories
<ul>
<li>Henry IV (<a href="mailto:henryiv@king.co.uk">email</a>)
<ul>
```

```
<li>Part I</li>
<li>Part II</li>
</ul>
<li><a href="http://www.shakespeare.co.uk/henryv.htm">
Henry V</a></li>
<li>Richard II</li>
</ul>
</li>
</ul>
```

注意第一个 <ul> 有一个 selected-plays ID，但 <li> 标签没有类，也没有任何的样式，这个列表看似像：

- Comedies
  - As You Like It
  - All's Well That Ends Well
  - A Midsummer Night's Dream
  - Twelfth Night
- Tragedies
  - Hamlet
  - Macbeth
  - Romeo and Juliet
- Histories
  - Henry IV (email)
    - Part I
    - Part II
  - Henry V
  - Richard II

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;
  list-style: 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。

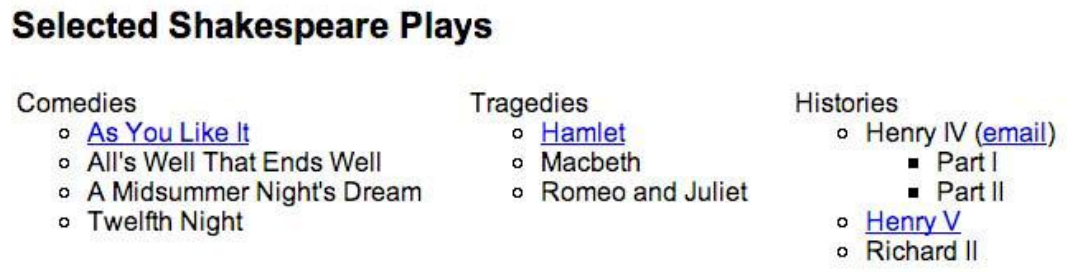
为了说明 jQuery 的选择器的用法我们动态地添加它到列表的最高级项，只有 **Comedies**、**Tragedies** 和 **Histories**，而不是将 horizontal 类直接附加到我们的 HTML 中。

```
$(document).ready(function() {
    $('#selected-plays > li').addClass('horizontal');
});
```

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* selected-plays (#selected-plays)（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 pseudo-class** （保留原意，否定伪类）方法获得所有子级的项目。

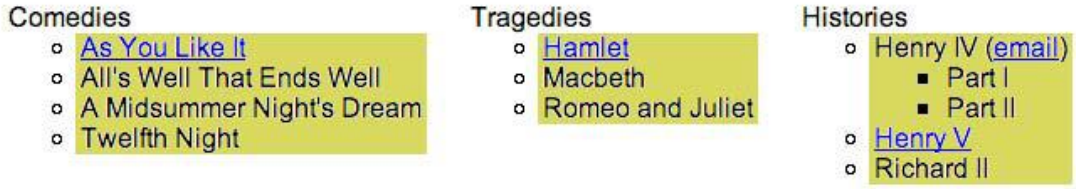
```
$(document).ready(function() {
$('#selected-plays > li').addClass('horizontal');
$('#selected-plays li:not(.horizontal)').addClass('sub-level');
});
```

这时我们得到每个列表项：

1、 是否 ID 为 selected-plays 的元素的子孙

2、 本身没有类 horizontal

当我们添加 sub-level 类到这些项目中，他们得到灰黄色的背景，在样式表中定义：.sub-level {background-color: #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。jQuery 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 {
text-decoration: none; /* remove internal link underline */
border-bottom: 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');
});
```

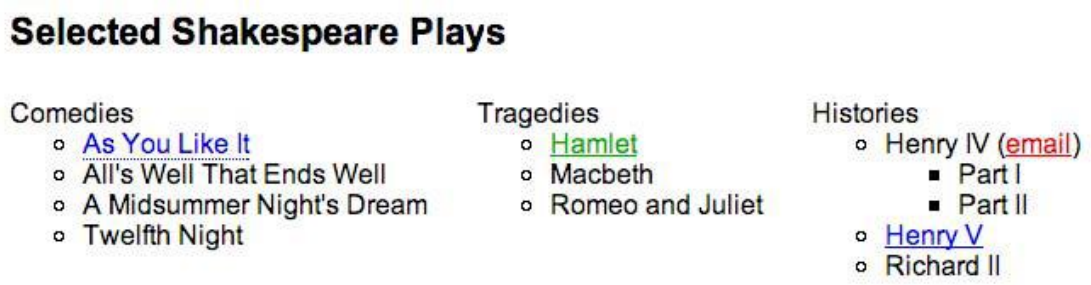
这里，mysite.com 在 href 的值出可出现在任何地方。如果我们也想在 mysite.com 里包含链接到任何子域名，那么这是特别重要的。

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:nth-child(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 {
background-color: #ffc; /* pale yellow for odd rows */
}
.even {
background-color: #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：

As You Like It	Comedy
All's Well that Ends Well	Comedy
Hamlet	Tragedy
Macbeth	Tragedy
Romeo and Juliet	Tragedy
Henry IV, Part I	History
Henry V	History

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 {font-weight: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：

As You Like It	Comedy
All's Well that Ends Well	Comedy
Hamlet	Tragedy
Macbeth	Tragedy
Romeo and Juliet	Tragedy
Henry IV, Part I	History
Henry V	History

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('table-heading');
$('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 table-heading 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('table-heading');
$('tr:not([th]):even').addClass('even');
$('tr:not([th]):odd').addClass('odd');
$('td:contains("Henry")').next().addClass('highlight');
});
```

根据加上 table-heading 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：

Title	Category
As You Like It	Comedy
All's Well that Ends Well	Comedy
Hamlet	Tragedy
Macbeth	Tragedy
Romeo and Juliet	Tragedy
Henry IV, Part I	History
Henry V	History

.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：

Title	Category	Year Published
As You Like It	Comedy	
All's Well that Ends Well	Comedy	1601
Hamlet	Tragedy	1604
Macbeth	Tragedy	1606
Romeo and Juliet	Tragedy	1595
Henry IV, Part I	History	1596
Henry V	History	1599

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 #my-element 的元素的 tag name 的话，可以用：

```
var myTag = $('#my-element').get(0).tagName;
```

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