

# JavaScript

## 脚本特效编程

## 给力起飞

苟英 曹海 白灵 高博

编著

想知道网页特效是怎么从无到有的吗？

想了解复杂的网页特效是怎么构建自己的模块吗？

知或者不知，本书就在这里，不增不减，循序渐进。

你即将有机会坐上网页特效设计大巴，从基础走到新技术，从框架模块走到项目实施。沿途你将欣赏到无数靓丽的实例风景，体验高速大巴的乐趣。



包含书中所有实例项目的源代码、  
素材及相关学习资料。



电子工业出版社

PUBLISHING HOUSE OF ELECTRONICS INDUSTRY  
http://www.phei.com.cn

# JavaScript 脚本特效编程给力起飞

苟 英 秦 涛 白 灵 高 博 编著

電子工業出版社

Publishing House of Electronics Industry

北京 · BEIJING

## 内 容 简 介

想知道网页特效是怎么从无到有的吗？想了解复杂的网页特效是怎么构建自己的模块的吗？

知或者不知，本书就在这里，不增不减，循序渐进。

你即将有机会坐上网页特效设计大巴，从基础走到新技术，从框架模块走到项目实现。沿途你将欣赏到无数靓丽的实例风景，体验高速大巴的乐趣。本巴士路线与以往有所不同，不仅会为你引入实例，也会向你逐步介绍 JavaScript 和其他网页设计基础知识。沿途停靠的站点包括：JavaScript 基础、JavaScript 语法、JavaScript 编程、页面交互信息的实现、Ajax 客户端技术、jQuery 框架、Ext JS 框架和 JavaScript 调试的利器 Firebug，以及使用 jQuery 实现在线留言板系统和一些门户网站的特效。结合每个站点的实例介绍，读者在每站学习后立刻就能投入实践。项目研发过程中的测试和开发是密不可分的，因此本巴士还在最后张贴了 Web 测试的知识，引入了网页调试工具 Firebug，并介绍了它的使用，让你不仅能够掌握 JavaScript 的知识，还能学到网页测试方面的内容，从而全面提高自身能力。

本书结合了丰富的开发经验及体会，将是广大网页爱好者及自学者的一个不错选择。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。  
版权所有，侵权必究。

### 图书在版编目 ( CIP ) 数据

JavaScript 脚本特效编程给力起飞 / 苟英等编著. —北京：电子工业出版社，2011.8  
ISBN 978-7-121-14046-4

I. ①J… II. ①苟… III. ①JAVA 语言—程序设计 IV. ①TP312

中国版本图书馆 CIP 数据核字 (2011) 第 134209 号

策划编辑：张月萍

责任编辑：贾 莉

印 刷：三河市鑫金马印装有限公司

装 订：电子工业出版社

北京市海淀区万寿路 173 信箱

邮编：100036

开 本：787×980 1/16

印张：25.25

字数：606 千字

印 次：2011 年 8 月第 1 次印刷

定 价：49.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：(010) 88254888。

质量投诉请发邮件至 zltts@phei.com.cn，盗版侵权举报请发邮件到 dbqq@phei.com.cn。

服务热线：(010) 88258888。

# 目 录

|                                  |    |
|----------------------------------|----|
| 第 1 篇 学习与积累 .....                | 1  |
| 第 1 章 JavaScript基础 .....         | 2  |
| 1.1 JavaScript与HTML .....        | 5  |
| 1.1.1 HTML 5 .....               | 5  |
| 1.1.2 HTML常用标签 .....             | 6  |
| 1.2.3 将JavaScript脚本嵌入HTML中 ..... | 20 |
| 1.2 编写JavaScript脚本工具 .....       | 22 |
| 1.3 简单的计算器 .....                 | 25 |
| 第 2 章 JavaScript语法 .....         | 27 |
| 2.1 JavaScript基础语法 .....         | 28 |
| 2.2 函数 .....                     | 39 |
| 2.2.1 特殊的内置函数 .....              | 39 |
| 2.2.2 创建自己的函数 .....              | 41 |
| 2.3 常用对象 .....                   | 43 |

|                              |    |
|------------------------------|----|
| 2.3.1 字符串对象 .....            | 44 |
| 2.3.2 Date对象 .....           | 56 |
| 2.3.3 Math对象 .....           | 60 |
| 2.4 数组 .....                 | 62 |
| 2.5 JavaScript错误 .....       | 68 |
| 2.6 JavaScript特性 .....       | 71 |
| 第3章 JavaScript编程 .....       | 74 |
| 3.1 DOM编程基础 .....            | 75 |
| 3.2 window对象 .....           | 76 |
| 3.2.1 window对象常用属性 .....     | 76 |
| 3.2.2 window对象常用方法 .....     | 78 |
| 3.2.3 window对象事件 .....       | 88 |
| 3.3 history和location对象 ..... | 90 |
| 3.3.1 history对象 .....        | 91 |
| 3.3.2 location对象 .....       | 91 |
| 3.4 document对象 .....         | 94 |
| 3.4.1 查询元素 .....             | 95 |

|                                 |     |
|---------------------------------|-----|
| 3.4.2 修改网页元素 .....              | 98  |
| 3.4.3 添加网页元素 .....              | 100 |
| 3.4.4 删除网页元素 .....              | 101 |
| 3.4.5 cookie.....               | 102 |
| 3.5 操作表格 .....                  | 104 |
| 3.5.1 表格的树型结构 .....             | 105 |
| 3.5.2 遍历行 .....                 | 105 |
| 3.5.3 添加行 .....                 | 107 |
| 3.5.4 删除行 .....                 | 109 |
| 3.6 下拉列表框的操作 .....              | 111 |
| 3.7 事件源的应用 .....                | 112 |
| 3.8 body对象 .....                | 114 |
| 3.9 form对象 .....                | 115 |
| 3.10 利用JavaScript创建对象 .....     | 119 |
| 3.11 JavaScript访问样式属性 .....     | 121 |
| 3.12 常用事件 .....                 | 123 |
| 3.13 JavaScript模块化和命名空间管理 ..... | 127 |

|                                   |     |
|-----------------------------------|-----|
| 3.13.1 模块化 .....                  | 127 |
| 3.13.2 命名空间管理 .....               | 128 |
| 3.14 正则表达式 .....                  | 131 |
| 3.14.1 正则表达式介绍 .....              | 131 |
| 3.14.2 正则表达式作用 .....              | 132 |
| 3.14.3 RegExp对象 .....             | 133 |
| 3.14.4 正则表达式语法参考 .....            | 133 |
| 3.14.5 String对象中与正则表达式有关的方法 ..... | 134 |
| 3.14.6 常见的验证方式 .....              | 137 |
| 3.15 JavaScript样式特效应用 .....       | 138 |
| 第 2 篇 提高与应用 .....                 | 156 |
| 第 4 章 页面交互信息的实现 .....             | 157 |
| 4.1 表单 .....                      | 158 |
| 4.2 表单控件 .....                    | 159 |
| 4.2.1 text文本框 .....               | 160 |
| 4.2.2 select下拉列表框 .....           | 160 |

|                                |     |
|--------------------------------|-----|
| 4.2.3 其他控件 .....               | 161 |
| 4.3 表单提交方式 .....               | 165 |
| 4.4 表单提交范例 .....               | 166 |
| 4.5 表单应用 .....                 | 167 |
| 4.6 表单验证 .....                 | 170 |
| 4.7 验证实现 .....                 | 176 |
| 4.7.1 输入框验证 .....              | 176 |
| 4.7.2 下拉列表框验证 .....            | 178 |
| 4.7.3 单选按钮验证 .....             | 181 |
| 4.7.4 复选框验证 .....              | 183 |
| 4.7.5 邮件地址验证 .....             | 187 |
| 第 5 章 Ajax客户端技术 .....          | 189 |
| 5.1 Ajax介绍 .....               | 190 |
| 5.1.1 Ajax技术的由来 .....          | 190 |
| 5.1.2 Ajax与JavaScript的关系 ..... | 191 |
| 5.2 Ajax程序范例 .....             | 191 |
| 5.2.1 使用Ajax完成验证 .....         | 191 |



|   |     |
|---|-----|
| 5.2.2 使用Ajax完成交互 .....                      | 195 |
| 5.3 一个注册的案例 .....                           | 197 |
| 5.4 使用 XMLHttpRequest对象与服务器端通信 .....        | 200 |
| 5.4.1 XMLHttpRequest对象 .....                | 201 |
| 5.4.2 使用open方法创建一个请求 .....                  | 202 |
| 5.4.3 使用 send 方法发送一个请求 .....                | 202 |
| 5.4.4 使用onreadystatechange 事件捕获请求状态变化 ..... | 203 |
| 5.4.5 使用 readyState 属性判断请求状态变化 .....        | 203 |
| 5.4.6 使用status属性判断请求的结果 .....               | 204 |
| 5.4.7 使用 responseText 获得返回的文本 .....         | 204 |
| 5.5 利用Ajax实现局部刷新 .....                      | 204 |
| 5.5.1 网页无闪自动局部刷新 .....                      | 204 |
| 5.5.2 表单局部刷新 .....                          | 206 |
| 5.6 实现注册页面 .....                            | 208 |
| 5.7 实时在线人数 .....                            | 215 |
| 第6章 jQuery框架 .....                          | 218 |
| 6.1 jQuery介绍 .....                          | 219 |

|       |                          |     |
|-------|--------------------------|-----|
| 6.1.1 | jQuery的由来 .....          | 219 |
| 6.1.2 | jQuery配置 .....           | 220 |
| 6.1.3 | jQuery常用语法及接口 .....      | 221 |
| 6.2   | jQuery程序范例 .....         | 223 |
| 6.2.1 | 选择器介绍 .....              | 223 |
| 6.2.2 | 选择器详解 .....              | 223 |
| 6.2.3 | 动态创建元素 .....             | 225 |
| 6.2.4 | 包装集元素管理 .....            | 227 |
| 6.2.5 | DOM操作：区分DOM属性和元素属性 ..... | 230 |
| 6.2.6 | 操作DOM属性 .....            | 231 |
| 6.2.7 | 修改元素的样式 .....            | 232 |
| 6.3   | 事件 .....                 | 236 |
| 6.3.1 | 事件和事件对象 .....            | 237 |
| 6.3.2 | jQuery中的事件 .....         | 239 |
| 6.4   | 利用jQuery实现页面特效 .....     | 243 |
| 6.5   | 实现鼠标单击留言切换高亮显示 .....     | 248 |
| 6.6   | 快餐在线 .....               | 249 |

|                                    |     |
|------------------------------------|-----|
| 第7章 Ext JS框架 .....                 | 254 |
| 7.1 Ext JS入门 .....                 | 254 |
| 7.1.1 获得Ext JS .....               | 254 |
| 7.1.2 应用Ext JS .....               | 255 |
| 7.1.3 Ext JS版的Hello Word .....     | 256 |
| 7.2 Ext JS框架基础及核心简介 .....          | 258 |
| 7.2.1 Ext JS类库简介 .....             | 258 |
| 7.2.2 Ext JS的组件 .....              | 259 |
| 7.3 Ext JS组件应用方法 .....             | 261 |
| 7.3.1 组件的使用 .....                  | 261 |
| 7.3.2 组件的配置属性 .....                | 264 |
| 7.3.3 事件处理 .....                   | 266 |
| 7.4 容器组件 .....                     | 269 |
| 7.4.1 面板 .....                     | 269 |
| 7.4.2 窗口Window及对话框MessageBox ..... | 274 |
| 7.4.3 布局概述 .....                   | 282 |
| 7.5 表格控件Grid.....                  | 284 |

|                                   |     |
|-----------------------------------|-----|
| 7.5.1 基本表格GridPanel.....          | 284 |
| 7.5.2 可编辑的表格EditorGridPanel ..... | 289 |
| 7.5.3 与服务器交互 .....                | 293 |
| 7.6 数据存储Store .....               | 295 |
| 7.6.1 Store.....                  | 295 |
| 7.6.2 Record .....                | 297 |
| 7.6.3 DataReader .....            | 298 |
| 7.6.4 DataProxy与自定义Store.....     | 300 |
| 7.7 使用表单Form.....                 | 301 |
| 7.7.1 FormPanel基本应用 .....         | 301 |
| 7.7.2 表单控件介绍 .....                | 303 |
| 7.8 Ext JS综合实例 .....              | 306 |
| 第3篇 综合与总结 .....                   | 311 |
| 第8章 高级特效范例 .....                  | 312 |
| 8.1 高级文字特效 .....                  | 312 |
| 8.2 高级图像特效 .....                  | 316 |
| 8.3 菜单特效 .....                    | 317 |
| 8.4 鼠标特效 .....                    | 330 |

|                                     |     |
|-------------------------------------|-----|
| 8.5 背景特效 .....                      | 332 |
| 8.6 页面特效 .....                      | 334 |
| 8.7 下载时间计算 .....                    | 336 |
| 8.8 游戏 .....                        | 341 |
| 第9章 jQuery实现在线留言板系统 .....           | 353 |
| 9.1 可折叠的留言板 .....                   | 353 |
| 9.2 浮动的留言板 .....                    | 359 |
| 第10章 门户网站首页特效 .....                 | 363 |
| 10.1 仿门户网站的幕布式Flash广告效果 .....       | 363 |
| 10.2 JavaScript实现仿 163 下拉广告效果 ..... | 370 |
| 10.3 浮动广告 .....                     | 373 |
| 10.4 设为首页和收藏本站JS脚本 .....            | 376 |
| 10.5 带缩略图的图片轮换代码 .....              | 378 |
| 第11章 JavaScript调试的利器Firebug.....    | 384 |
| 11.1 Firebug的安装 .....               | 385 |
| 11.2 Firebug的使用 .....               | 386 |

|        |                     |     |
|--------|---------------------|-----|
| 11.2.1 | Console 控制台 .....   | 386 |
| 11.2.2 | 查看和修改HTML.....      | 387 |
| 11.2.3 | CSS调试 .....         | 388 |
| 11.2.4 | 可视化的CSS标尺 .....     | 389 |
| 11.2.5 | 网络状况监视器 .....       | 389 |
| 11.2.6 | JavaScript调试器 ..... | 390 |
| 11.2.7 | DOM查看器 .....        | 390 |

## 前 言

JavaScript 是由 Netscape 公司开发的一套与超文本标记语言 HTML 紧密结合的脚本语言，为网页制作者提供了非常灵活的应用和发挥空间。JavaScript 已经成为当今网页特效设计语言中最流行、最成熟的一种。JavaScript 主要用于网页的交互性设计，功能十分强大，加上开发人员的设计技巧，实现的特效有时可以达到匪夷所思的地步。

本书旨在让初学 JavaScript 的读者快速而全面地掌握使用 JavaScript 语言进行 Web 开发的要点，立足于让读者快速入门、快速掌握、快速应用的目的。为了使读者更好地建立快速学习的目的，本书对知识点进行了归总，一些不必要、太过基础的知识点不讲或略讲，重要的知识点，结合例子详细说明，由浅入深一步一步系统性地带领读者学习 JavaScript，以达到事半功倍的效果。本书中还介绍了多个实例小程序的开发，读者不但可以通过它们熟悉基础知识点，更可以在日后的编程中参考或直接修改后使用。

本书最大的特点就是采用了大量的实例帮助读者学习 JavaScript，让读者可以非常容易、非常形象地理解所介绍的知识，甚至可以直接敲击代码先查看效果再学习。本书还介绍了网页测试的相关知识，让你成为 JavaScript 编程高手的同时，也成为网页测试的前驱，不仅能让你编写的 Web 系统功能强大，更能让它拥有强大的性能特点。目前还没有关于 YSLOW 的介绍的相关书籍，Web 系统测试又是目前重要的研究课题，相信读者在此可以了解更多的除了开发以外，有关测试的知识。

本书共 11 章，让你从完全不会 JavaScript 到精通 JavaScript，成为一个 JavaScript 的编程高手，并且能够实现性能优化。本书从最基本的 HTML 语法讲起，让你在学习时不会有这方面的障碍。然后为读者讲解 JavaScript 的起源，以及一些常识性的东西，再到基本的语法，到 JavaScript DOM 编程，一步一步由浅入深地为读者讲解。到最后，本书将所讲解的知识运用到具体的实例中去，让读者将学习到的知识运用到实际中。第 8 章中为读者收录了一些特效的 JavaScript 代码，读者除了可以用于学习之外，稍加修改还可以运用到自己的网页中，做出具有自己特色的东西。本书最后一章讲解了网页测试的工具 Firebug，让读者不仅能编程也能调试，从而大大提高 Web 系统的性能。

本书的第 1 篇是全书的基础，包括：

### 第 1 章 JavaScript 基础

本章主要介绍 JavaScript 的基础以及与 HTML 的关系，让你能立刻进入到 JavaScript 的世界中去。

### 第 2 章 JavaScript 语法

本章介绍了 JavaScript 的基本语法，帮助读者打好编程基础。

### 第 3 章 JavaScript 编程

本章主要介绍了 DOM 编程，window、history、location、document、body 和 form 对象，同时针对 JavaScript 模块化、样式特效等做了介绍。

第 2 篇在基础知识上进行了提高，着重介绍了如何将基础知识应用到实践中去，包括：

### 第 4 章 页面交互信息的实现

本章主要介绍了表单的知识，包括表单控件、提交方式、应用和范例。

### 第 5 章 Ajax 客户端技术

本章主要介绍了 Ajax 技术，并且以实例介绍如何用它实现一些网页特效。

### 第 6 章 jQuery 框架

本章主要介绍了 jQuery 框架技术，使用它实现一些网页特效，并完成快餐在线的简单功能。

### 第 7 章 Ext JS 框架

本章主要介绍了 Ext JS 框架技术，让读者了解其基础和核心、组件的应用方法，以及实例。

美妙的网页不是靠一个或者两个特效就能够完成的，它需要恰如其分地将各种特效加入到网页中。

第 3 篇通过案例综合地讲解了如何恰当应用各种特效的方法，以及各种特效的实现，包括：

### 第 8 章 高级特效范例

本章列举了很多用 JavaScript 实现的网页特效，读者可直接或者稍微修改一下进行使用。

### 第 9 章 jQuery 实现在线留言板系统

本章主要介绍了使用 jQuery 实现留言板系统。

### 第 10 章 门户网站首页特效

本章主要介绍了诸如网易、新浪等门户网站中的一些 Flash 广告效果等的特效。

### 第 11 章 JavaScript 调试的利器 Firebug

本章主要介绍了网页的调试工具 Firebug，它集 HTML 查看和编辑、JavaScript 控制台、网络状况监视器于一体，是开发 JavaScript、CSS、HTML 和 Ajax 的得力助手。Firebug 如同一把精巧的瑞士军刀，从各个不同的角度剖析 Web 页面内部的细节层面，给 Web 开发者带来很大的便利。

本书的内容是建立在朋友们研究成果的基础上，参与编写的有苟英、秦涛、白灵、高博、陈其，在本书完成之际要向提供帮助的朋友们表示衷心的感谢！

作者



## 第 3 章

### JavaScript 编程

---

本章将带领读者走进 JavaScript 中的 DOM 编程，主要介绍 DOM 编程的基础，以及常用对象如 window 对象、document 对象等，让读者领略 DOM 编程的艺术魅力，进一步提高读者的脚本编程能力。

## 3.1 DOM 编程基础

一个完整的 JavaScript 实现是由以下三个不同部分组成的：

- 核心（ECMAScript）
- 文档对象模型（DOM）
- 浏览器对象模型（BOM）

这里只对 DOM 做一个介绍。DOM 是“Document Object Model”（文档对象模型）的首字母缩写，是 HTML 和 XML 的应用程序接口（API）。DOM 把整个页面规划成由节点层级构成的文档。HTML 或 XML 页面的每个部分都是一个节点的衍生物。

请看如下的代码：

```
<html>
  <head>
    <title>Sample Page</title>
  </head>
  <body>
    <p>hello world!</p>
  </body>
</html>
```

这段代码可以用 DOM 绘制成一个节点层次图，如图 3-1 所示。

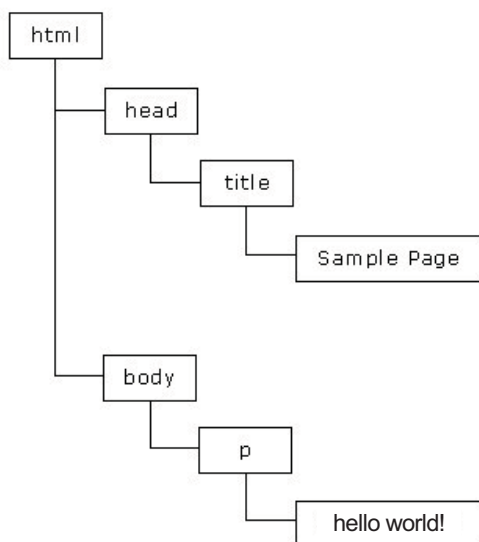


图 3-1 节点层次图

DOM 通过创建树来表示文档，从而使开发者对文档的内容和结构具有空前的控制力。用 DOM API 可以轻松地删除、添加和替换节点。DOM 定义了所有文档元素的对象和属性，以及访问它们的方法（接口）。

下面来看看这些常用的对象属性和方法。

## 3.2 window 对象

JavaScript 语言里的对象可以分为三种类型：

- 用户定义对象（user-defined object）：由程序员自行创建的对象。
- 内置对象（native object）：内置在 JavaScript 语言里的对象，如上一章讲的 Array、Math 和 Date 等。
- 宿主对象（host object）：由浏览器提供的对象。

编写 JavaScript 脚本时，用到的最基础的宿主对象就是 window 对象。在客户端 JavaScript 中，window 对象是全局对象，如果要访问其中的某个属性，可以不把 window 写出来，比如访问 window 对象的 document 属性时，可以只写“document”，而不必写“window.document”。它是一个顶层对象，而不是另一个对象的属性。

### 3.2.1 window 对象常用属性

window 对象的常用属性见表 3-1。

表 3-1 window 对象常用属性

| 属性名         | 描述                                      |
|-------------|---|
| closed      | 返回窗口是否已被关闭                              |
| name        | 设置或返回窗口或者框架的名称                          |
| document    | 对 document 对象的只读引用，该文档表示浏览器窗口中的 HTML 文档 |
| history     | 对 history 对象的只读引用，该对象包含当前浏览器访问过的 URL 信息 |
| location    | 对于窗口或框架的 location 对象，包含当前相关的 URL 信息     |
| screen      | 对 screen 对象的只读引用，该对象包含有关客户的屏幕和显示性能信息    |
| status      | 设置或者读取窗口状态栏的文本信息                        |
| innerheight | 返回窗口的文档显示区的高度                           |
| innerwidth  | 返回窗口的文档显示区的宽度                           |

(续表)

| 属性名                     | 描述  |
|-------------------------|---|
| opener                  | 返回对创建此窗口的窗口的引用, 当在界面 A 中打开另外一个窗口 B 的时候, 可以在页面 B 中使用该属性得到页面 A 的信息  |
| parent                  | 返回父窗口, 一般在框架中使用   |
| screenLeft<br>screenTop | 只读整数。声明了窗口的左上角在屏幕上的 $x$ 坐标和 $y$ 坐标。IE、Safari 和 Opera 支持 screenLeft 和 screenTop, 而 Firefox 和 Safari 支持 screenX 和 screenY |

下面来看一下如何使用这些属性:

```
<html>
<head>
<title>window 对象属性</title>
<script type="text/javascript">
    window.status="显示 window 对象常用属性! ";    //状态栏中显示的内容
    //得到 window 对象的属性
    var c=window.closed;
    var na=window.name;
    var sA=window.status;
    var h=window.innerHeight;
    var w=window.innerWidth;
    var sl=window.screenLeft;
    var sT=window.screenTop;
    //输出 window 对象的属性
    document.write("closed:"+c+"<hr>");
    document.write("name:"+na+"<hr>");
    document.write("status:"+sA+"<hr>");
    document.write("innerheight:"+h+"<hr>");
    document.write("innerwidth:"+w+"<hr>");
    document.write("screenLeft:"+sl+"<hr>");
    document.write("screenTop:"+sT+"<hr>");
</script>
</head>
<body>
</body>
</html>
```

将上面的代码保存为后缀名为 **html** 的文件, 打开后看到如图 3-2 所示的界面。

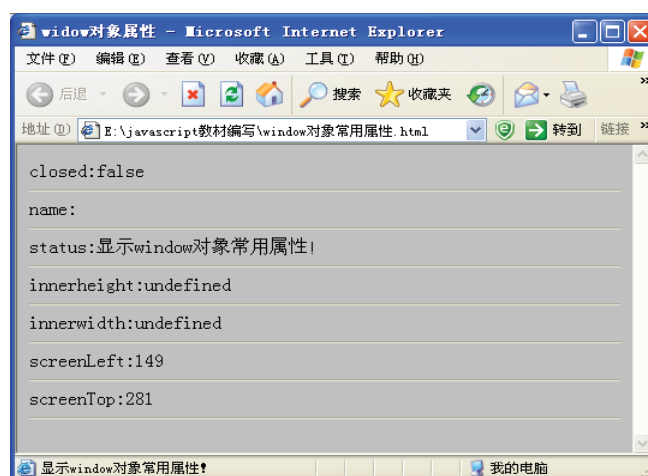


图 3-2 window 对象属性

### 3.2.2 window 对象常用方法

window 对象的常用方法见表 3-2。

表 3-2 window 对象的常用方法

| 方法名            | 描述                                     |
|----------------|--|
| alert()        | 显示包含应指定信息的对话框，对话框中只有一个确认按钮             |
| confirm()      | 显示一个带指定信息的确认框，确认框带确认和取消按钮，选择不同按钮返回不同结果 |
| prompt()       | 显示一个带提示信息的输入框                          |
| parseInt()     | 将指定字符串转换为整型数字                          |
| parseFloat()   | 将指定字符串转换为浮点型数字                         |
| open()         | 打开一个新窗体，并加载指定的 HTML 文档                 |
| close()        | 关闭当前的浏览器窗体                             |
| setTimeout()   | 设置定时器：在指定毫秒值之后指定某个指定的函数                |
| clearTimeout() | 取消某个定时器                                |

下面对这些方法逐一进行讲解。

#### (1) alert()

alert()方法用于显示带有一条指定消息和一个【确定】按钮的警告框。

**注意：**中文环境下，显示的是【确定】，若是英文环境则显示【OK】，以下都相同。

语法格式:

```
alert (message)
```

**message** 是指要在 **window** 上弹出的对话框中显示的纯文本（而非 **HTML** 文本）。

### 例 3-1

```
<html>
<head>
<script type="text/javascript">
  alert ("我的第一个对话框!")
</script>
</head>
<body>
</body>
</html>
```

将上面的代码保存后，双击打开，会看到如图 3-3 所示的界面。



图 3-3 alert 方法

### (2) confirm()

**confirm()** 方法的作用是弹出有两个按钮的对话框，一个是【确定】，一个是【取消】。单击【确定】按钮则返回真，否则为假。

语法格式:

```
confirm (value)
```

**value** 是指要在 **window** 上弹出的对话框中显示的纯文本（而非 **HTML** 文本），它有两个值，分别对应【确定】和【取消】。

### 例 3-2

```
<html>
<head>
<script type="text/javascript">
  function test() {
    var res=confirm("你今天开心吗? ");
    if(res){
```

```

    alert("相当开心! ");
}
else{
    alert("要开心哟! ");
}
}
</script>
</head>
<body>
<input type="button" value="confirm方法" onclick="test()" />
</body>
</html>

```

将上面的代码保存后，页面上会显示 `confirm方法` 按钮，单击后弹出如图 3-4 所示的对话框。



图 3-4 confirm 方法

### (3) prompt()

`prompt()` 方法用于显示可提示用户进行输入的对话框。

语法格式：

```
prompt(text, defaultText)
```

**text**：可选，要在对话框中显示的纯文本（而不是 HTML 格式的文本）。

**defaultText**：可选，默认的输入文本。

如果用户单击提示框中的【取消】按钮，则返回 `null`。如果用户单击【确定】按钮，则返回输入字段当前显示的文本。在用户单击【确定】按钮或【取消】按钮把对话框关闭之前，它将阻止用户对浏览器的所有输入。在调用 `prompt()` 时，将暂停对 JavaScript 代码的执行，在用户做出响应之前，不会执行下一条语句。

### 例 3-3

```

<html>
<head>
<script type="text/javascript">
function disp_prompt()
{
    var name=prompt("请输入你的名字: ", "")

```

```

if (name!=null && name!="")
{
    document.write("Hello " + name + " !")
}
}
</script>
</head>
<body>
<input type="button" onclick="disp_prompt()" value="prompt 方法" />
</body>
</html>

```

将上面的代码保存后，页面上会显示  按钮，单击后会弹出如图 3-5 所示的对话框。

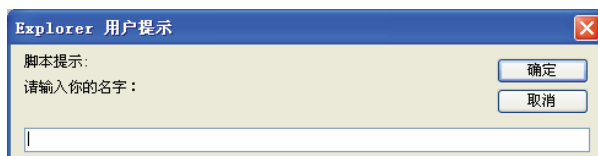


图 3-5 prompt 方法（一）

如果你什么都不输入，直接单击【取消】或者【关闭】按钮，则关闭该对话框；如果你在输入框中输入名字，如“美羊羊”，单击【确定】按钮，则在界面中会显示“Hello 美羊羊！”，如图 3-6 所示。

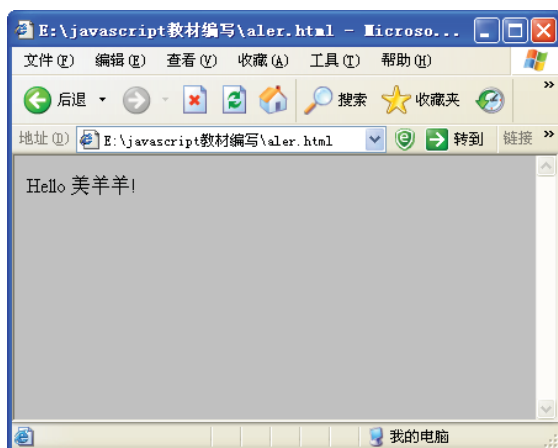


图 3-6 prompt 方法（二）

#### （4）open()

open() 方法用于打开一个新的浏览器窗口或查找一个已命名的窗口。

语法格式：

```
window.open("URL","窗口名","窗口特征")
```



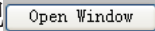
- URL：显示打开的页面地址；
- 窗口名：显示新窗口的标题；
- 窗口特征：用于设置打开窗口的特征，常用的窗口特征见表 3-3。

表 3-3 窗口特征

| 特征                    | 含义                  |
|-----------------------|---------------------|
| height=pixels         | 窗口文档显示区的高度，以像素计     |
| left=pixels           | 窗口的 x 坐标，以像素计       |
| location=yes no 1 0   | 是否显示地址字段。默认是 yes    |
| menubar=yes no 1 0    | 是否显示菜单栏。默认是 yes     |
| resizable=yes no 1 0  | 窗口是否可调节尺寸。默认是 yes   |
| scrollbars=yes no 1 0 | 是否显示滚动条。默认是 yes     |
| status=yes no 1 0     | 是否添加状态栏。默认是 yes     |
| titlebar=yes no 1 0   | 是否显示标题栏。默认是 yes     |
| toolbar=yes no 1 0    | 是否显示浏览器的工具栏。默认是 yes |
| top=pixels            | 窗口的 y 坐标            |
| width=pixels          | 窗口的文档显示区的宽度，以像素计    |

例 3-4

```
<html>
<head>
<script type="text/javascript">
function open_win()
{
window.open("http://www.baidu.com","百度","width=400,height=30,menubar=1,statusbars=0")
}
</script>
</head>
<body>
<input type=button value="Open Window" onclick="open_win()" />
</body>
</html>
```

将以上代码保存后用网页方式打开，页面中出现  按钮，如图 3-7 所示，单击该按钮，则打开网址为 <http://www.baidu.com> 的百度网页。

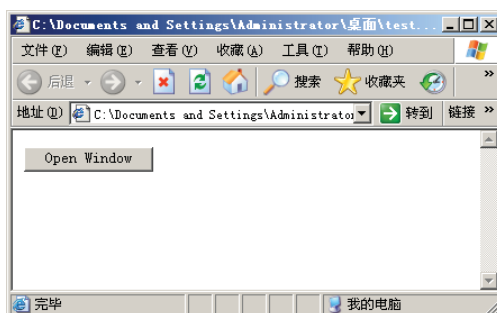


图 3-7 open 方法

### (5) close()

close() 方法用于关闭浏览器窗口。

语法格式:

```
window.close()
```

### 例 3-5

```

<html>
<head>
<script type="text/javascript">
function closeWin()
{
    myWindow.close()
}
</script>
</head>
<body>
<script type="text/javascript">
myWindow=window.open('', '', 'width=200,height=100')
myWindow.document.write("This is myWindow")
</script>
<input type="button" value="Close Window"
onclick="closeWin()" />
</body>
</html>

```

将以上代码保存后, 界面显示 “Close Window” 按钮 (如图 3-8 所示), 同时打开一个 “This is myWindow” 的新窗口 (如图 3-8 所示中的空白页), 单击 “Close Window” 按钮, 则新窗口会被关闭。

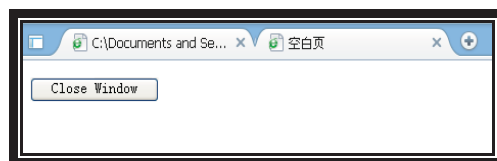


图 3-8 close 方法

### (6) parseInt()

返回由字符串转换得到的整数。

语法格式：

```
parseInt(numString, [radix])
```

**numString**：必选项。要转换为数字的字符串。

**radix**：可选项。2~36 之间的表示 **numString** 所保存数字的进制的值。如果没有提供，则前缀为 '0x' 的字符串被当做十六进制，前缀为 '0' 的字符串被当做八进制。所有其他字符串都被当做是十进制的。

#### 例 3-6

```
<html>
<body>
<script type="text/javascript">
document.write(parseInt("10") + "<br />")
document.write(parseInt("19",10) + "<br />")
document.write(parseInt("11",2) + "<br />")
document.write(parseInt("17",8) + "<br />")
document.write(parseInt("1f",16) + "<br />")
document.write(parseInt("010") + "<br />")
document.write(parseInt("He was 40") + "<br />")
</script>
</body>
</html>
```

上面代码显示的结果如图 3-9 所示。



图 3-9 字符串转换成整数

### (7) parseFloat()

`parseFloat()` 函数可解析一个字符串，并返回一个浮点数。

语法格式：

```
parseFloat(string)
```

**string**：必需。要被解析的字符串。

**注意：**只有字符串中的第一个数字会被返回。开头和结尾的空格是允许的。如果字符串的第一个字符不能被转换为数字，那么 `parseFloat()` 会返回 `NaN`。如果只想解析数字的整数部分，请使用 `parseInt()` 方法。

#### 例 3-7

```
<html>
<body>
<script type="text/javascript">
document.write(parseFloat("10") + "<br />")
document.write(parseFloat("10.00")+ "<br />")
document.write(parseFloat("10.33")+ "<br />")
document.write(parseFloat("34 45 66")+ "<br />")
document.write(parseFloat(" 60 ") + "<br />")
document.write(parseFloat("40 years")+ "<br />")
document.write(parseFloat("He was 40")+ "<br />")
</script>
</body>
</html>
```

上面代码显示的结果如图 3-10 所示。

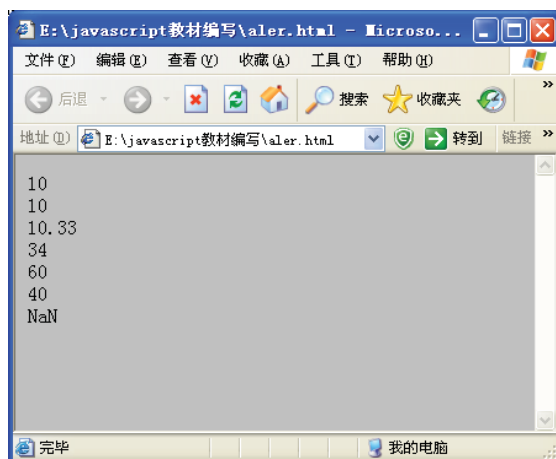


图 3-10 字符串转换成浮点数

### (8) setTimeout()

**setTimeout()** 方法用于在指定的毫秒数后调用函数或计算表达式。

语法格式：

```
setTimeout (code, millisec)
```

**code**：必需。要调用的函数后要执行的 JavaScript 代码串。

**millisec**：必需。在执行代码前需等待的毫秒数。

**setTimeout()** 只执行 **code** 一次。如果要多次调用，可使用 **setInterval()** 或者让 **code** 自身再次调用 **setTimeout()**。

#### 例 3-8

```
<html>
<head>
<script type="text/javascript">
function startTime()
{
var today=new Date()
var h=today.getHours()
var m=today.getMinutes()
var s=today.getSeconds()
m=checkTime(m)
s=checkTime(s)
document.getElementById('txt').innerHTML=h+":"+m+":"+s
t=setTimeout('startTime()',500)
}
function checkTime(i)
{
if (i<10)           // 当数字小于 10 时在前面加 0
    {i="0" + i}
    return i
}
</script>
</head>
<body onload="startTime()">
<div id="txt"></div>
</body>
</html>
```

效果显示如图 3-11 所示。

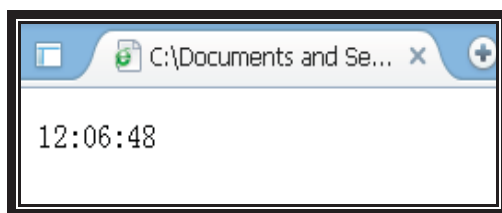


图 3-11 时间显示

### (9) clearTimeout()

clearTimeout() 方法可取消由 setTimeout() 方法设置的 timeout。

语法格式：

```
clearTimeout(id)
```

id：由 setTimeout() 返回的 id 值。该值标识要取消的延迟执行代码块。

### 例 3-9

```
<html>
<head>
<script type="text/javascript">
function startTime()
{
var today=new Date()
var h=today.getHours()
var m=today.getMinutes()
var s=today.getSeconds()
m=checkTime(m)
s=checkTime(s)
document.getElementById('txt').innerHTML=h+":"+m+":"+s
t=setTimeout('startTime()',500)
}
function checkTime(i)
{
if (i<10)
{i="0" + i}
return i
}
function stopCount()
{
clearTimeout(t)
}
</script>
```

```
</head>
<body onload="startTime()" ">
<div id="txt"></div>
<input type="button" value="Stop" onClick="stopCount()" ">
</body>
</html>
```

这个例子的作用是停止显示计时器，只需单击界面（如图 3-12 所示）上的 **Stop** 按钮即可。

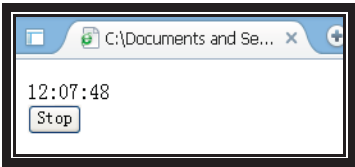


图 3-12 停止计时器

3.2.3 window 对象事件

window 对象有很多事件，方便对一些关键点进行操作，常用的 window 对象事件见表 3-4。

表 3-4 常用 window 对象事件

| 事件名            | 描述                     |
|----------------|------------------------|
| onload         | 装载完成后触发                |
| onunload       | 退出时触发                  |
| onbeforeunload | 退出时触发，会发生在 onunload 之前 |
| onhelp         | 显示帮助时触发                |
| onfocus        | 获得焦点时触发                |
| onblur         | 失去焦点时触发                |
| onerror        | 错误时触发                  |
| onresize       | 改变大小时触发                |
| onscroll       | 滚动时触发                  |
| onmove         | 当对象移动时触发               |

1) onload

onload 在使用时有三种方法：

方法一：直接在 HTML 标签中指定。这种方法用得很多。

格式：

```
<标签 ..... 事件="事件处理程序" [事件="事件处理程序" ...]>
```

## 例 3-10

```
<html>
<head>
</head>
<body onload="alert('网页读取完成，请慢慢欣赏! ')" onunload="alert('再见! ')">
</body>
</html>
```

这样定义的<body>标签，能在文档读取完毕的时候弹出一个对话框，写着“网页读取完成，请慢慢欣赏”；在用户退出文档（或者关闭窗口，或者到另一个页面去）的时候弹出“再见”（如图 3-13 所示）。

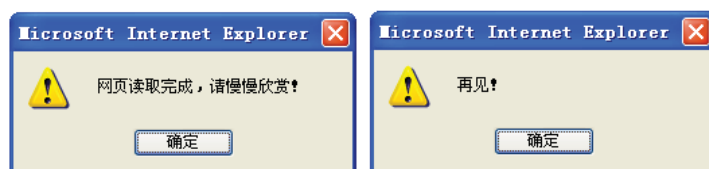


图 3-13 onload 对象（一）

**方法二：**编写特定对象特定事件的 JavaScript。这种用得比较少。

格式：

```
<script language="javascript" for="对象" event="事件">
...
(事件处理程序代码)
...
</script>
```

## 例 3-11

```
<html>
<head>
<script language="javascript" for="window" event="onload">
  alert('网页读取完成，请慢慢欣赏! ');
</script>
</head>
</body>
</html>
```

这段代码能在文档读取完毕的时候弹出一个对话框，写着“网页读取完成，请慢慢欣赏”（如图 3-14 所示）。





图 3-14 onload 对象 (二)

方法三：在 JavaScript 中说明。

格式：

```
<事件主角 (对象)>.<事件> = <事件处理程序>;
```

用这种方法要注意的是，“事件处理程序”是真正的代码，而不是字符串形式的代码。事件处理程序若是一个自定义函数，如无使用参数的需要，就不要加“()”。

### 例 3-12

```
<html>
<head>
<script language="javascript">
  function ignoreError() {
    return true;
  }
  window.onerror = ignoreError;
</script>
</head>
<body>
</body>
</html>
```

这个例子将 `ignoreError()` 函数定义为 `window` 对象的 `onerror` 事件的处理程序。它的效果是忽略该 `window` 对象下的任何错误（由引用不允许访问的 `location` 对象产生的“没有权限”错误是不能被忽略的）。

## 3.3 history 和 location 对象

在 DOM 树模型的结构中，`window` 对象处于树的顶点，是一个顶级对象。在 `window` 对象之下还有一些系统对象，如 `location`、`history`、`screen` 等，这些对象实际上也是 `window` 对象的一些属性，能对其里面的属性内容进行修改或者访问，但是不能创建这些对象。下面将讲解其中两个比较重要的对象，即 `history` 和 `location` 对象。

### 3.3.1 history 对象

当用户浏览网页时,浏览器保存了一个最近访问网页的 URL 列表,一般来说,在该列表中的 URL 是不能通过 JavaScript 获取的。DOM 模型中,这个列表使用 history 对象表示。history 对象是 window 对象的属性,可以通过 window.history 访问,当然也可以像 window 对象的其他方法属性一样,省略掉 window 直接使用 history 访问。使用 history 对象及其 back()或者 go()方法,可以实现与浏览器【前进】、【后退】按钮相同的功能。

#### 1. 属性

history 对象的属性比较少,常用的就只有 length 属性,通过该属性可以得到 history 保存的历史列表的长度,就是用户访问过的不同地址的数目。

history 对象有 current、previous 和 next 三个属性,用来存储历史列表中的 URL。但是,为了安全和保护隐私,这些对象在现在的浏览器中不能正常访问。

#### 2. 方法

history 对象常用方法如表 3-5 所示。

表 3-5 history 对象常用方法

| 方法名        | 描述  |
|------------|---|
| back()     | 载入历史列表中前一个网址,相当于按下【后退】按钮                      |
| forward()  | 载入历史列表中后一个网址,相当于按下【前进】按钮                      |
| go(number) | 加载 history 列表中的某个具体页面。要使用这个方法,必须在括号内指定一个正数或负数 |

history 对象的 back()方法相当于单击一次【后退】按钮,forward 方法相当于单击一次【前进】按钮。go()方法可以代替 back 和 forward,当 go 方法的参数为正数时表示前进、为负数时表示后退。例如:

```
history.go(1) //代表前进一页,相当于浏览器中的【前进】按钮,等价于 forward()
history.go(-1) //代表后退一页,相当于浏览器中的【后退】按钮,等价于 back() 方法
history.go(-2) //相当于按【后退】按钮两次,等价于执行两次 back() 方法
```

### 3.3.2 location 对象

在 DOM 模型的层次结构中,location 对象比 window 对象低一个层次,它相当于浏览器中的地址栏,包含了关于当前 URL 地址的信息。location 对象是 window 对象的一个部分,可通过 window.location 的方式来访问,当然也可以将 window 省略掉,直接使用 location 对象访问。

## 1. 属性

location 对象的属性保存了当前浏览器地址栏中的所有信息,如表 3-6 所示,表中使用的例子 URL 地址为: <http://www.google.cn/ig/china?hl=zh-CN>。

表 3-6 location 对象属性

| 属性名      | 描述         | 例                                      |
|----------|------------|--|
| protocol | 访问地址协议     | http                                   |
| hostname | 主机名        | www.google.cn                          |
| port     | 访问端口       | 80                                     |
| host     | 主机名及端口号    | www.google.cn:80                       |
| pathname | 访问路径名      | /ig/china                              |
| href     | 完整的 URL 路径 | http://www.google.cn/ig/china?hl=zh-CN |

## 2. 方法

location 对象常用方法如表 3-7 所示。

表 3-7 location 对象方法

| 方法名            | 描述                      |
|----------------|-------------------------|
| assign("url")  | 加载指定的新 HTML 文档          |
| reload()       | 重新加载当前 HTML 文档          |
| replace("url") | 用新的 HTML 文档替换当前 HTML 文档 |

通过设置 location 对象 href 属性的值 (或者调用 location 的一些方法), 可以实现网页之间的跳转, 比如下例中使用 location 实现单击按钮跳转页面的操作。

例 3-13

```
.....
<script type="text/javascript">
    function openGoogle() {
        //使用 location.href 属性实现跳转
        location.href="http://www.google.cn";
    }
    function openBaidu() {
        //使用 location.assign() 方法实现跳转
        location.assign("http://www.baidu.com");
    }
}
```

[illegible]

上例中，通过设置 `location.href` 的属性值实现单击按钮跳转到“google”首页，通过调用 `location.assign()` 方法实现单击按钮跳转到“百度”首页。当然也可以将跳转的代码直接写在按钮的 `<input/>` 标签中，如：

```
.....  
<center>  
    <input type="button" value="连接到 google"  
        onClick="javascript: location.href('http://www.google.cn'); " />  
    &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~  
    <input type="button" value="连接到百度"  
        onClick="javascript:location.assign('http://www.baidu.cn');" />  
</center>
```

运行例 3-13 的效果如图 3-15 所示。单击【连接到 google】按钮时将看到“google”的首页，如图 3-16 所示，单击【连接到百度】按钮时将看到“百度”的首页，如图 3-17 所示。



图 3-15 location 对象的使用



图 3-16 location 对象属性的使用



图 3-17 location 对象方法的使用

### 3.4 document 对象

每个载入浏览器的 HTML 文档都会成为 document 对象。document 对象使用户可以从脚本中对 HTML 页面中的所有元素进行访问。document 对象是 window 对象的一部分，可通过 window.document 属性对其进行访问，document 对象属性见表 3-8。

表 3-8 document 对象属性

| 属性               | 描述  |
|------------------|---|
| title            | 设置文档标题，等价于 HTML 的<title>标签                          |
| bgColor          | 设置页面背景色   |
| fgColor          | 设置前景色（文本颜色）   |
| linkColor        | 未单击过的链接颜色   |
| alinkColor       | 激活链接（焦点在此链接上）的颜色                                    |
| vlinkColor       | 已单击过的链接颜色   |
| URL              | 设置 URL 属性，从而在同一窗口打开另一网页                             |
| fileCreateDate   | 文件建立日期，只读属性   |
| fileModifiedDate | 文件修改日期，只读属性   |
| fileSize         | 文件大小，只读属性   |
| cookie           | 设置和读出 cookie  |
| charset          | 设置字符集（简体中文：gb2312）                                  |
| body             | 提供对 <body> 元素的直接访问。对于定义了框架集的文档，该属性引用最外层的 <frameset> |

使用 document 对象属性的语法格式为 “document.属性”。

document 对象方法见表 3-9。

表 3-9 document 对象方法

| 方法                     | 描述  |
|------------------------|---|
| close()                | 关闭用 document.open() 方法打开的输出流，并显示选定的数据                     |
| getElementById()       | 返回对拥有指定 id 的第一个对象的引用                                      |
| getElementsByName()    | 返回带有指定名称的对象集合   |
| getElementsByTagName() | 返回带有指定标签名的对象集合  |
| open()                 | 打开一个流，以收集来自任何 document.write() 或 document.writeln() 方法的输出 |
| write()                | 向文档写 HTML 表达式 或 JavaScript 代码                             |
| writeln()              | 等同于 write() 方法，不同的是在每个表达式之后写一个换行符                         |

### 3.4.1 查询元素

网页元素的查询主要使用 document 对象的 getElementById()、getElementsByName()、getElementsByTagName()方法和 Element 的几个属性来完成。

#### 1. 使用 getElementById 方法

按照 W3C 的标准，同一个 ID 名在一个 HTML 文档中只能出现一次，即使 HTML 文档不规范，使用 getElementById()方法也只能得到一个网页元素，而且是最前面的一个。

例 3-14

```
<html>
<head>
<title>通过 ID 查询网页元素</title>
<script type="text/javascript">
function queryByID() {
    //通过 ID 查询文本框元素
    var txt = document.getElementById("account");
    alert(txt.value); //显示文本框输入内容
}
</script>
</head>
<body>
    <div id="content">
        <input type="text" id="account" />
    </div>
    <hr size="1" color="#FF0000" />
    <input type="button" value="通过 ID 查询" onclick="queryByID()" />
</body>
</html>
```

```
</body>
</html>
```

运行后，单击按钮将看到如图 3-18 所示的效果。



图 3-18 使用 getElementById 方法

## 2. 通过 getElementsByName 查找

按照 W3C 的标准，同一个 name 名称可以在一个 HTML 文档中出现多次，所以使用 getElementsByName()方法将得到一组网页元素。

### 例 3-15

```
//.....
<script type="text/javascript">
function queryByName() {
    //通过 name 查询
    var list = document.getElementsByName("interest");
    //通过数组的 length 属性取得查询到的元素个数
    var len = list.length;
    //将每个元素的 value 属性值连接成一个字符串
    var str = "";
    for(var i=0;i<list.length;i++){
        str=str + list[i].value + "," ;
    }
    alert("查到的元素个数: " + len + "\n\n元素内容: " + str);
}
</script>
</head>
<body>
    <div id="content">爱好:
        <input type="checkbox" name="interest" value="跑步" />跑步
        <input type="checkbox" name="interest" value="篮球" />篮球
        <input type="checkbox" name="interest" value="游泳" />游泳
```

```

        <input type="checkbox" name="interest" value="登山" />登山
    </div>
    <hr size="1" color="#FF0000" />
    <input type="button" value="通过 name 属性查询" onClick="queryByName()" />
</body>
</html>

```

运行后，单击按钮将看到如图 3-19 所示的效果。

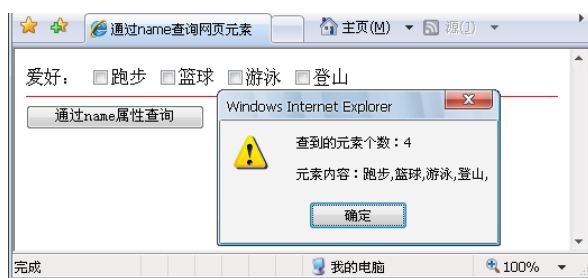


图 3-19 使用 `getElementsByName` 方法

### 3. 通过 `getElementsByTagName` 查找

在 HTML 文档中某一个标签可能出现多次，所以使用 `getElementsByTagName()` 方法将得到一组网页元素。

#### 例 3-16

```

<html>
<head>
<title>通过 ID 查询网页元素</title>
<script type="text/javascript">
function test() {
    //通过标签名查询
    var list = document.getElementsByTagName("input");
    //通过数组的 length 属性取得查询到的元素个数
    var len = list.length;
    //将每个元素的 value 属性值连接成一个字符串
    var str = "";
    for(var i=0;i<list.length;i++){
        str=str + "    " + list[i].type + ":" + list[i].value + "\n" ;
    }
    //输出查询到的内容
    alert("查到的元素个数: " + len + "\n\n元素类型和值内容: \n" + str);
}
</script>

```



```

</head>
<body>
    <div id="content">爱好:
        <input type="checkbox" name="interest" value="跑步" />跑步
        <input type="checkbox" name="interest" value="篮球" />篮球
    </div>
    <hr size="1" color="#FF0000" />
    <input type="button" value="通过标签名查询" onClick="test()" />
</body>
</html>

```

运行后将看到如图 3-20 所示的效果。

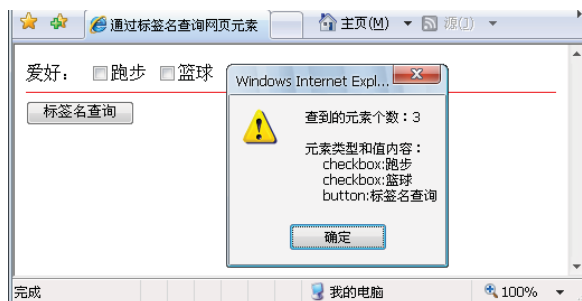


图 3-20 使用 getElementsByTagName() 方法

### 3.4.2 修改网页元素

当使用 document 提供的方法和 Element 的属性得到网页元素之后，就可以对元素的内容进行修改，如下例所示的“全选 / 全不选”的实现。

例 3-17

```

<html>
<head>
<title>全选</title>
<script type="text/javascript">
    //实现全选函数
    function choose(val) {
        //通过 name 属性值得到所有复选框
        var listCH = document.getElementsByName("kc");
        //循环修改复选框属性
        for(var i=0;i<listCH.length;i++){
            //修改复选框的状态
            listCH[i].checked=val;
        }
    }

```

```

    } </script>
</head>
<body>
<center>
<form>
    <h3>你希望学习的课程</h3>
    <a href="javascript:choose(true);">全选</a> /
    <a href="javascript:choose(false);">全不选</a>
    <hr size="1" />
    <div style="text-align:left; padding-left:140px;">
        <input type="checkbox" name="kc" value="0" />java 基础<br>
        <input type="checkbox" name="kc" value="1" />HTML + CSS + JavaScript <br>
        <input type="checkbox" name="kc" value="2" />java 核心<br>
        <input type="checkbox" name="kc" value="3" />java web<br>
        <input type="checkbox" name="kc" value="4" />SSH<br>
        <input type="checkbox" name="kc" value="5" />SqlServer2005<br>
        <input type="checkbox" name="kc" value="6" />Oracle10g<br>
    </div>
</form>
</center>
</body>
</html>

```

运行上例，单击【全选】按钮，将看到如图 3-21 所示的效果。当单击【全不选】按钮的时候，所有复选框的选中都将被去掉，如图 3-22 所示。



图 3-21 全选

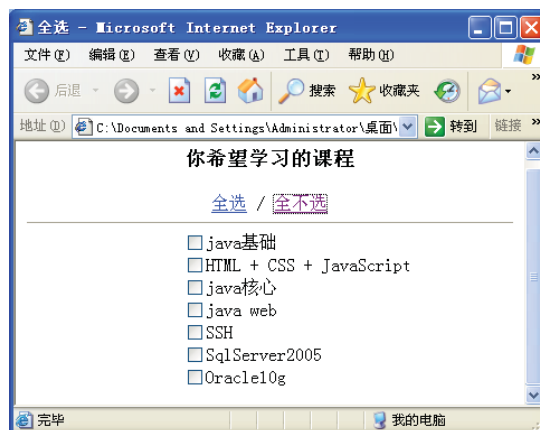


图 3-22 全不选

### 3.4.3 添加网页元素

添加网页元素主要使用 `document.createElement()` 方法和 `Element.appendChild()` 方法实现。使用 `document.createElement()` 方法用于创建一个元素，新创建的元素是独立的，和 HTML 文档没有任何关系，所以需要使用 `Element.appendChild()` 方法将新创建的网页元素添加到 DOM 中。如下例所示，在层中动态创建文本框。

例 3-18

```
<html>
<head>
<title>添加网页元素</title>
<script type="text/javascript">
    function addTxt() {
        // 创建一个 input 元素
        var txt = document.createElement("input");
        // 设置元素为文本框
        txt.type="text";
        txt.value="新添加的文本框";
        // 将文本框添加到层中
        var div = document.getElementById("disDiv");
        div.appendChild(txt);
    }
</script>
</head>
<body>
<center>
    <div id="disDiv" style="text-align:center; border:1px solid blue; width:450px;
height:100px;">
        </div>
        <input type="button" value="添加文本框" onclick="addTxt()" />
</center>
</body>
</html>
```

运行上例，当单击页面中的按钮的时候，层中将添加一个文本框，每单击一次执行一次 `addTxt()` 函数，每执行一次该函数添加一个文本框，效果如图 3-23 所示。

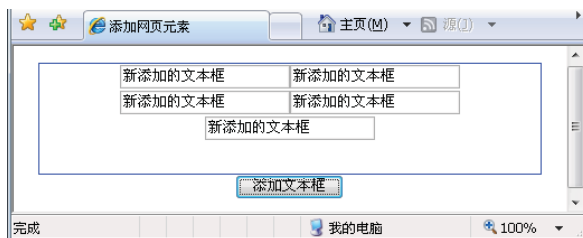


图 3-23 添加网页元素

### 3.4.4 删除网页元素

删除元素主要通过 `document` 的 `removeChild()` 方法实现。如下例所示，动态删除层中按钮元素。

例 3-19

```
<html>
<head>
<title>删除网页元素</title>
<script type="text/javascript">
    function delEle(){
        //得到层
        var div = document.getElementById("disDiv");
        //得到层的子元素
        var childList = div.childNodes;
        //如果层有子元素，就删除第一个子元素
        if(childList!=null && childList.length>0){
            div.removeChild(childList[0]);
        }
        else{
            alert("层中已经没有可以删除的子元素了!");
        }
    }
</script>
</head>
<body>
<center>
    <div id="disDiv" style="text-align:center; border:1px solid blue; width:450px;
padding:15px;">
        <input type="button" value="button1" />
        <input type="button" value="button2" />
        <input type="button" value="button3" />
        <input type="button" value="button4" />
    </div>
    <hr size="1" />
    <input type="button" value="删除层中的元素" onClick="delEle()" />
</center>
</body>
</html>
```

运行上例，单击一次按钮，层中的按钮就会减少一个。单击两次按钮之后的效果如图 3-24 所示。



图 3-24 删除网页元素

### 3.4.5 cookie

大家可能知道在 `document` 对象中有一个 `cookie` 属性。但是 `cookie` 是什么呢？`cookie` 就是所谓的缓存文件，也就是某些 Web 站点在您的硬盘上用很小的文本文件存储了一些信息，这些文件就称为 `cookie`。一般来说，`cookie` 是 CGI 或类似比 HTML 高级的文件、程序等创建的，但是 JavaScript 也提供了对 `cookie` 的很全面的访问权利。

每个 `cookie` 都是这样的：`<cookie 名>=<值>`。

`<cookie 名>`的限制与 JavaScript 的命名限制大同小异，只要你只用字母和数字命名，就完全没有问题了。`<值>`的要求也是“只能用可以用在 URL 编码中的字符”。

每个 `cookie` 都有失效日期，一旦电脑的时钟过了失效日期，这个 `cookie` 就会被删掉。不能直接删掉一个 `cookie`，但是可以用设定失效日期早于现在时刻的方法来间接删掉它。

每个网页或者说每个站点，都有它自己的 `cookie`，这些 `cookie` 只能由这个站点下的网页来访问，来自其他站点或同一站点下未经授权的区域的网页，是不能访问的。每一“组”`cookie` 有规定的总大小(大约 2KB 每“组”)，一旦超过最大的总大小，则最早失效的 `cookie` 会先被删除，以便让新的 `cookie` “安家”。

现在我们来学习使用 `document.cookie` 属性。

如果直接使用 `document.cookie` 属性，或者说用某种方法（如给变量赋值）来获得 `document.cookie` 的值，就可以知道在现在的文档中有多少个 `cookie`，每个 `cookie` 的名字和它的值。例如，在某文档中添加“`document.write(document.cookie)`”，结果显示：

```
name=kevin; email=kevin@kevin.com; lastvisited=index.html
```

这意味着，文档包含三个 `cookie`：`name`、`email` 和 `lastvisited`，它们的值分别是 `kevin`、`kevin@kevin.com` 和 `index.html`。可以看到，两个 `cookie` 之间是用分号和空格隔开的，于是可以用“`cookieString.split(';')`”方法得到每个 `cookie` 分开的一个数组。

先用 `var cookieString = document.cookie`，设定一个 `cookie` 的方法，对 `document.cookie` 赋值。与其他情况下的赋值不同，向 `document.cookie` 赋值不会删除原有的 `cookie`，而只会增添 `cookie` 或更改原有 `cookie`。赋值的格式如下：

```
document.cookie = 'cookieName=' + escape('cookieValue') + ';expires=' +
expirationDateObj.toGMTString();
```

是不是看得头晕了呢？`cookieName` 表示 `cookie` 的名称，`cookieValue` 表示 `cookie` 的值，`expirationDateObj` 表示存储着失效日期的日期对象名，如果不需要指定失效日期，则不需要第二行。不指定失效日期，则浏览器默认是在关闭浏览器（也就是关闭所有窗口）之后过期。

首先，`escape()` 方法为什么一定要用？因为 `cookie` 的值的 requirements 是“只能用可以用在 URL 编码中的字符”。知道了“`escape()`”方法是把字符串按 URL 编码方法来编码的，那只需要用一个“`escape()`”方法来处理输出到 `cookie` 的值，用“`unescape()`”来处理从 `cookie` 接收过来的值就万无一失了。而且这两个方法的最常用用途就是处理 `cookie`。其实设定一个 `cookie` 只用“`document.cookie = 'cookieName=cookieValue'`”这么简单，但是为了避免在 `cookieValue` 中出现 URL 里不准出现的字符，还是用一个 `escape()` 比较好。

然后，“`expires`”前面的分号，注意到就行了，是分号而不是其他。

最后，`toGMTString()` 方法，设定 `cookie` 的失效日期都是用 GMT 格式的时间的，其他格式的时间是没有作用的。

下面来实战一下。设定一个“`name=rose`”的 `cookie`，在三个月后过期。

### 例 3-20

```
var expires = new Date();
expires.setTime(expires.getTime() + 3 * 30 * 24 * 60 * 60 * 1000);
/* 三个月×一个月当做30天×一天24小时
   ×一小时60分×一分60秒×一秒1000毫秒 */
document.cookie = 'name=redrose;expires=' + expires.toGMTString();
```

上面为什么没有用 `escape()` 方法？这是因为知道 `redrose` 是一个合法的 URL 编码字符串，也就是说，`redrose = escape('redrose')`。一般来说，如果设定 `cookie` 时不用 `escape()`，那获取 `cookie` 时也不用 `unescape()`。

下面再编写一个函数，作用是查找指定 `cookie` 的值。

### 例 3-21

```
function getCookie(cookieName) {
    var cookieString = document.cookie;
    var start = cookieString.indexOf(cookieName + '=');
    // 加上等号的原因是避免在某些 cookie 的值里有
    // 与 cookieName 一样的字符串。
    if (start == -1) // 找不到
        return null;
    start += cookieName.length + 1;
```

```
var end = cookieString.indexOf(';', start);
if (end == -1) return unescape(cookieString.substring(start));
return unescape(cookieString.substring(start, end));
}
```

这个函数用到了字符串对象的一些方法，如果你不记得了，请快去查查。这个函数所有的 if 语句都没有带上 else，这是因为如果条件成立，程序运行的都是 return 语句，在函数里碰上 return，就会终止运行，所以不加 else 也没问题。该函数在找到 cookie 时，就会返回 cookie 的值，否则返回“null”。

现在要删除刚才设定的“name=redrose” cookie。

```
var expires = new Date();
expires.setTime(expires.getTime() - 1);
document.cookie = 'name=redrose;expires=' + expires.toGMTString();
```

可以看到，只需要把失效日期改成比现在日期早一点（这里是早 1 毫秒），再用同样的方法设定 cookie，就可以删掉 cookie 了。

### 3.5 操作表格

根据上面讲解的使用 document 和 Element 动态操作网页元素的方法，可以对页面中的表格及其数据进行动态操作，比如添加一行数据、修改一行数据，以及删除一行数据等。但是当使用上面的方式来实现表格的时候，会发现实现的代码非常烦琐，在 DOM 编程中对表格的操作可以变得更简单，即使用 table 对象自身的一些方法和属性。

表格对象用于动态操作数据的常用属性和方法如表 3-10 和表 3-11 所示。表格中行的属性和方法如表 3-12 和表 3-13 所示。

表 3-10 table 属性

| 属性名   | 描述                           |
|-------|------------------------------|
| rows  | 得到来自于 table 对象的 tr（表格行）对象的集合 |
| cells | 得到整个表格中所有单元格的集合              |

表 3-11 table 方法

| 属性名               | 描述                           |
|-------------------|------------------------------|
| insertRow()       | 在表格中创建新行（tr），并将行添加到 rows 集合中 |
| deleteRow(tr)     | 从表格及 rows 集合中删除指定行（tr）       |
| moveRow(tr,index) | 将表格行移动到新位置                   |

表 3-12 tr 属性

| 属性名             | 描述                       |
|-----------------|--------------------------|
| cells           | 得到表格行中所有单元格的集合           |
| sectionRowIndex | 得到行在行集合 rows 中的编号，从 0 开始 |

表 3-13 tr 方法

| 属性名             | 描述                                 |
|-----------------|------------------------------------|
| insertCell()    | 在表格行（tr）中创建新单元格，并将单元格添加到 cells 集合中 |
| deleteCell (td) | 从表格行及 cells 集合中删除指定单元格（td）         |

3.5.1 表格的树型结构

在操作表格元素之前，应该先熟悉表格在 DOM 模型中的整体结构。只有熟悉了表格在 DOM 的结构之后，才能得心应手地处理表格内的各个元素，实现动态操作表格的目的，如图 3-25 所示，展示了一个二行三列的表格在 DOM 中的结构。

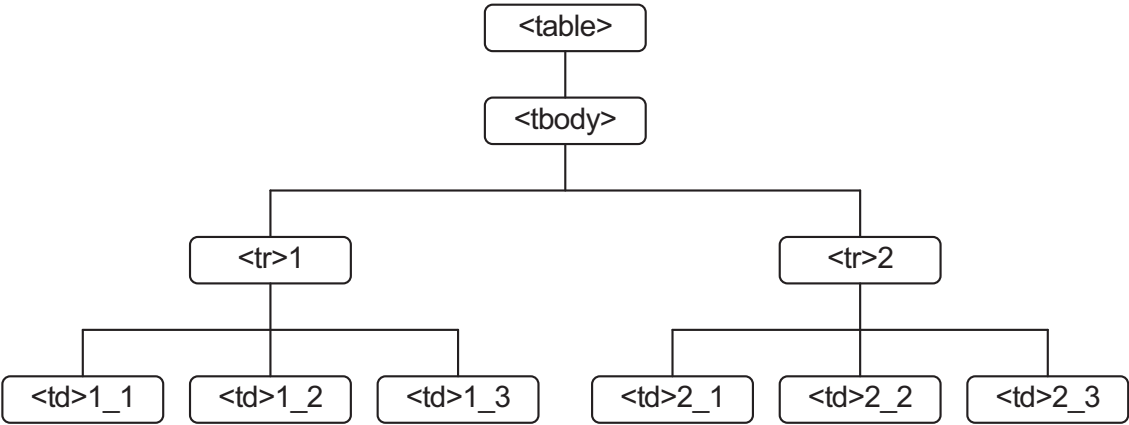


图 3-25 二行三列的表格在 DOM 中的结构

3.5.2 遍历行

表格中行的遍历使用表格 rows 属性即可，如下例中循环显示表格中各个学员的成绩。

例 3-22

```
<html>
<head>
<title>遍历表格</title>
<style type="text/css">
    body,table,th,td{ font-size:12px; }
```



```

#tabScore{ border-collapse:collapse; }
#tabScore th{ padding:5px; }
#tabScore td{ padding:5px; }
</style>
<script type="text/javascript">
    function disScore() {
        var tab = document.getElementById("tabScore");
        var rows = tab.rows; //得到表格所有的行
        for(var i=1;i<rows.length;i++){
            //得到每行的各项内容
            var name = rows[i].childNodes[0].innerText;
            var java = rows[i].childNodes[1].innerText;
            var html = rows[i].childNodes[2].innerText;
            var sql = rows[i].childNodes[3].innerText;
            var count = rows[i].childNodes[4].innerText;
            alert(name + ": " + java + "," + html + "," + sql + "," + count);
        }
    }
</script>
</head>
<body>
    <input type="button" value="查看表中成绩" onClick="disScore()" />
    <hr size="1" />
    <table id="tabScore" width="470" border="1" bordercolor="#003399" align=
"center">
        <tr bgcolor="#0099FF">
            <th>姓名</th><th>JAVA</th><th>HTML</th><th>SqlServer</th><th>总成绩</th>
        </tr>
        <tr>
            <td align="center">张三</td>
            <td align="center">87</td>
            <td align="center">92</td>
            <td align="center">79</td>
            <td align="center">258</td>
        </tr>
        <tr>
            <td align="center">李四</td>
            <td align="center">70</td>
            <td align="center">80</td>
            <td align="center">75</td>
            <td align="center">225</td>

```

```

        </tr>
    </table>
</body>
</html>

```

在上例中通过表格的 **rows** 属性得到表格的所有行，通过行的 **childNodes** 得到表格单元格，最后通过 **innerText** 得到单元格的文本内容。运行效果如图 3-26 所示。

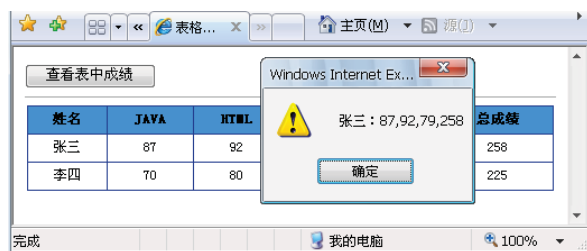


图 3-26 循环显示学生成绩

### 3.5.3 添加行

在表格中添加行要相对复杂一些，因为当添加一个行的时候还必须考虑到行中的单元格。比如例 3-22 中的表格，需要考虑姓名和四个成绩的单元格。所以添加行的时候需要使用到表格的 **insertRow()** 方法以及 **tr** 的 **insertCell** 方法。

当然，在使用 **insertRow** 和 **insertCell** 创建行和列的时候，并不需要将创建的元素添加到相应的 **tbody** 和 **tr** 上，因为这两个方法会自动实现该功能。如例 3-23 中将用户输入的成绩信息添加到表格中。

例 3-23

```

<html>
<head>
<title>添加行</title>
<style type="text/css">
    body,table,th,td,input{ font-size:12px; }
    #tabScore{ border-collapse:collapse; }
    #tabScore th{ padding:5px; }
    #tabScore td{ padding:5px; }
</style>
<script type="text/javascript">
    function addScore() {
        //得到输入的数据
        var name = document.getElementById("userName").value;
        var java = document.getElementById("java").value;
        var html = document.getElementById("html").value;
    }

```



```

    </table>
</center>
</body>
</html>

```

运行上例，并输入姓名和三项成绩，单击【添加】按钮，将看到表格中增加了一行数据。运行效果如图 3-27 所示。

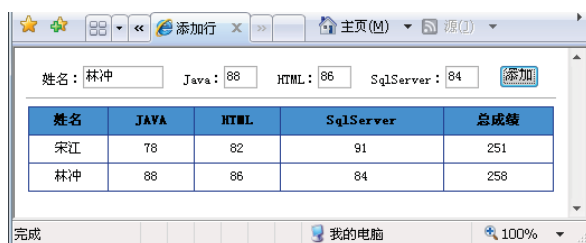


图 3-27 添加行

### 3.5.4 删除行

在 DOM 树模型中，当删除某一个元素的时候，该元素之下的所有子元素都将被删掉。同样在表格中删除表格中的行的时候，行下所有的单元格都将被删除掉，如下例中删除最后一行的学员成绩。

例 3-24

```

<html>
<head>
<title>删除表格中的行</title>
<style type="text/css">
    body,table,th,td,input{ font-size:12px; }
    #tabScore{ border-collapse:collapse; }
    #tabScore th{ padding:5px; }
    #tabScore td{ padding:5px; }
</style>
<script type="text/javascript">
    function delScore() {
        var tab = document.getElementById("tabScore");
        if(tab.rows.length>1){
            //通过行的索引编号删除行
            tab.deleteRow(tab.rows.length-1);
        }
        else{ alert("已经没有可以删除的学员成绩! "); }
    }
</script>

```

```

</head>
<body>
<center>
    <input type="button" value="删除最后一个学员的成绩" onClick="delScore()" />
    <hr size="1" />
    <table id="tabScore" width="470" border="1" bordercolor="#003399">
        <tr bgcolor="#0099FF">
            <th>姓名</th><th>JAVA</th><th>HTML</th><th>SqlServer</th><th>总成绩</th>
        </tr>
        <tr>
            <td align="center">宋江</td>
            <td align="center">87</td>
            <td align="center">92</td>
            <td align="center">79</td>
            <td align="center">258</td>
        </tr>
        <tr>
            <td align="center">林冲</td>
            <td align="center">70</td>
            <td align="center">80</td>
            <td align="center">75</td>
            <td align="center">225</td>
        </tr>
        <tr>
            <td align="center">李逵</td>
            <td align="center">100</td>
            <td align="center">100</td>
            <td align="center">100</td>
            <td align="center">300</td>
        </tr>
    </table>
</center>
</body>
</html>

```

当运行页面单击删除按钮时，表格中的数据将减少一行，直到成绩被删除完为止。运行效果如图 3-28 所示。



图 3-28 删除行

## 3.6 下拉列表框的操作

下拉列表框 **select** 的操作和其他操作有一些区别，它的选项存放在 **options** 集合中，可以使用该集合制作出非常漂亮的联动效果，如下例，实现了省份和城市的联动。

例 3-25

```
<html>
<head>
<title>联动效果</title>
<script type="text/javascript">
    function choose(){
        var pSel = document.getElementById("province").value;
        var city = document.getElementById("city");
        //删除下拉列表框中原有的所有选项
        city.options.length = 0;
        if(pSel == "重庆"){
            //创建下拉列表框的选项
            var cq = new Option("重庆","重庆");
            //将选项添加到下拉框中
            city.options.add(cq);
            var wz = new Option("万州","万州");
            city.options.add(wz);
            var kx = new Option("开县","开县");
            city.options.add(kx);
            var hc = new Option("合川","合川");
            city.options.add(hc);
        }
        else if(pSel == "四川"){
            var cd = new Option("成都","成都");
            city.options.add(cd);
            var djy = new Option("都江堰","都江堰");
            city.options.add(djy);
            var wc = new Option("汶川","汶川");
            city.options.add(wc);
            var ls = new Option("乐山","乐山");
            city.options.add(ls);
        }
    }
</script>
```



表 3-14 event 对象属性

| 属性名        | 描述                                |
|------------|-----------------------------------|
| button     | 该属性用于得到鼠标事件的按键（左键为 1、中键为 4、右键为 2） |
| keyCode    | 该属性用于得到键盘事件的按键值                   |
| x          | 设置或获取鼠标指针位置相对于父文档的 x 像素坐标         |
| y          | 设置或获取鼠标指针位置相对于父文档的 y 像素坐标         |
| srcElement | 设置或获取触发事件的对象                      |
| offsetX    | 设置或获取鼠标指针位置相对于触发事件的对对象的 x 坐标      |
| offsetY    | 设置或获取鼠标指针位置相对于触发事件的对对象的 y 坐标      |
| type       | 从 event 对象中获取事件名称                 |

例 3-26

```
<html>
<head>
<title>event 的使用</title>
<script type="text/javascript">
    function test(){
        var str = "";
        str += "事件类型: " + event.type + "\n";
        str += "鼠标 x 坐标: " + event.x + "\n";
        str += "鼠标 y 坐标: " + event.y + "\n";
        if(event.button == 0){
            str += "鼠标击键: 左键\n";
        }
        else if(event.button == 2){
            str += "鼠标击键: 右键\n";
        }
        else if(event.button == 4){
            str += "鼠标击键: 中键\n";
        }
        //得到触发对象
        var but = event.srcElement;
        str += but.type + ": " + but.value + "\n";
        alert(str);
    }
</script>
</head>
<body>
    <input type="button" value="[查看事件源]" onClick="test()" />
```



```
</body>
</html>
```

运行上例，事件源对象的属性值如图 3-30 所示。

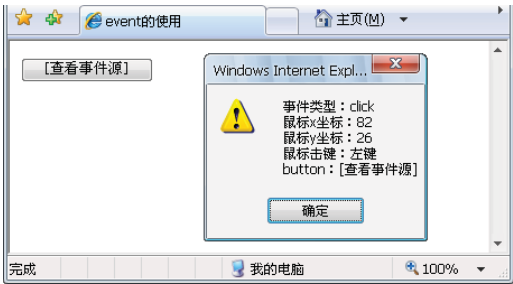


图 3-30 事件源的使用

# 3.8 body 对象

body 对象代表文档的主体，其属性见表 3-15。

表 3-15 body 对象的属性

| 属性        | 描述                |
|-----------|-------------------|
| className | 设置或返回元素的 class 属性 |
| dir       | 设置或返回文本的方向        |
| id        | 设置或返回 body 的 id   |
| lang      | 设置或返回元素的语言代码      |
| title     | 设置或返回元素的标题        |

下面一起来使用 body 对象，让文本内容随着右边的滚动条移动，效果如图 3-31 所示。

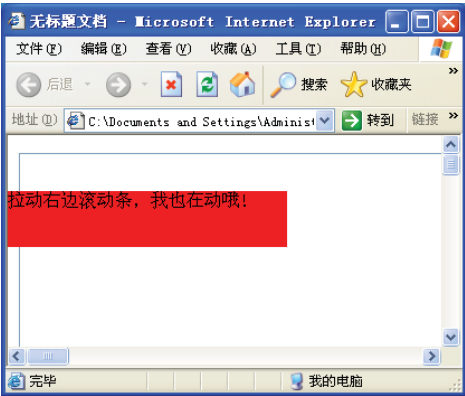


图 3-31 body 对象的使用

显示如图 3-31 所示效果的代码如下：

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>无标题文档</title>
<script language="javascript">
function dealscroll()
{
it.style.top=document.body.scrollTop+50;
it.style.left=document.body.scrollLeft;
}
</script>
<style type="text/css">
<!--
#it {
position: absolute;
left:50px;
top:50px;
background-color:#FF0000;
height:50px;
width:250px;
}
-->
</style>
</head>

<body onscroll="dealscroll()" >
<textarea rows="100" cols="500" id="textareal" name="textareal">
</textarea>
<div id="it">
    拉动右边滚动条，我也在动哦！
</div>
</body>
</html>
```

## 3.9 form 对象

form 对象代表一个 HTML 表单。在 HTML 文档中 <form> 每出现一次，form 对象就会被创建。form 对象方法见表 3-16。

表 3-16 form 对象属性

| 属性            | 描述                            |
|---------------|-------------------------------|
| acceptCharset | 服务器可接受的字符集                    |
| action        | 设置或返回表单的 action 属性            |
| enctype       | 设置或返回表单用来编码内容的 MIME 类型        |
| id            | 设置或返回表单的 id                   |
| length        | 返回表单中的元素数目                    |
| method        | 设置或返回将数据发送到服务器的 HTTP 方法       |
| name          | 设置或返回表单的名称                    |
| target        | 设置或返回表单提交结果的 Frame 或 Window 名 |

1. onsubmit 事件

form 对象的 onsubmit 属性指定了一个事件句柄函数。当用户单击了表单中的 reset 按钮而提交一个重置时，就会调用这个事件句柄函数。注意，当调用方法 form.submit()时，该处理器函数不会被调用。

如果 onsubmit 句柄返回 fasle，表单的元素就不会提交。如果该函数返回其他值或什么都没有返回，则表单会被提交。

语法格式：

```
form.onsubmit
```

2. 表单字段元素

■ 方法

- blur 方法：用于按表单字段元素失去焦点，焦点被移动到后台。
- focus 方法：用于按某个表单字段元素获得焦点。
- click 方法：单击事件。
- setCapture 方法：用于某个表单字段元素对象上捕获鼠标的事件。
- releaseCapture 方法：用于取消某个表单字段元素对象捕获鼠标对象的设置。
- select 元素对象的 add 方法：用于添加选择项。

■ 属性

- defaultValue 属性：设置或返回表单字段元素的默认值。
- disabled 属性：用于设置或返回表单字段元素的 disabled 的状态。

**form** 属性：返回表单字段元素所属于的 **form** 的对象。

**readOnly** 属性：设置或返回 **readOnly** 状态。

**title** 属性：设置或返回表单字段元素的 **title** 属性。

**value** 属性：设置或返回表单字段元素的当前取值。

**checked** 属性：设置或返回单选和复选的选中状态。

列表框（**select**）的专有属性：

**multiple** 属性：设置或返回列表框的 **multiple** 属性。

**selectedIndex** 属性：设置或返回列表框中被选中的项的索引号。

**options** 数组属性：一个包含列表框所有选择项对象的数组（`<option value="" selected>text</option>`）。

**text** 属性：代表选择项中的文本。

**value** 属性：代表选择项所对应的取值。

**selected** 属性：设置或返回选择项是否被选中。

**index** 属性：设置或返回选择项在所有的选择项中的顺序索引位置。

### 3. 表单事件

**onChange** 事件：当元素发生改变时发生。

**onSelect** 事件：对应当行或多行文本框被选中后的事件。

**onFocus** 事件：当获得焦点时产生的事件。

**onBlur** 事件：当失去焦点时产生的事件。

#### 例 3-27

```
<html>
<head>
<script type="text/javascript">
function showMethod()
{
    var x=document.getElementById("myForm")
    alert(x.method)
}
</script>
</head>
<body>
```

```

<form id="myForm" method="post">
名称: <input type="text" value="米老鼠" />
<input type="button" onclick="showMethod()" value="显示 method" />
</form>

</body>
</html>

```

显示效果如图 3-32 所示。

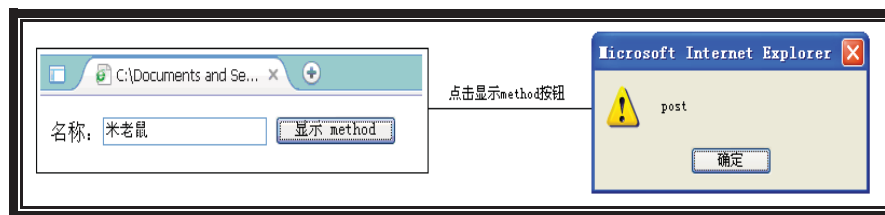


图 3-32 得到表单元素的提交方式

### 例 3-28

```

<html>
<head>
<script type="text/javascript">
function changeAction()
{
var x=document.getElementById("myForm")
alert("Original action: " + x.action)
x.action="index.asp"
alert("New action: " + x.action)
x.submit()
}
</script>
</head>
<body>
<FORM name="guideform">
<SELECT name="guidelinks">
<OPTION SELECTED value="http://www.baidu.com">所有
<OPTION value="http://www.sinna.com">重庆
<OPTION value="http://www.163.com">深圳
<OPTION value="http://www.淘宝.com">广州
</SELECT>
<INPUT type="button" name="go" value="Go!"
onClick="window.location=document.guideform.guidelinks.options

```

```
[document.guidedform.guidelinks.selectedIndex].value"> </FORM>
</body>
</html>
```

显示结果如图 3-33 所示。

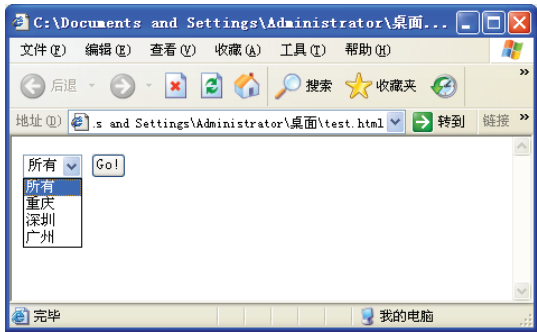


图 3-33 下拉列表框的制作

### 3.10 利用 JavaScript 创建对象

利用 JavaScript 创建对象只是为了创建一个简单的实体，这个实体包含了许多属性和方法。以下是一个对象构造器的例子： `function Student(name, age, colour) { this.name = name; this.age = age; this.colour = colour; }`。然后，可以这样来调用 Student 构造器 “`var somebody = new Student("Patrick", 22, "red");`” 或者从用户那里得到输入数值： `var sName = prompt("What is your name?");` `var sAge = prompt("How old are you this year?");` `var sColour = prompt("What is your favourite colour?");` `var somebody = new Student(pName, pAge, pColour);`。所有在 JavaScript 中的构造器，就像在 Java 中，都可以利用 new 关键字来进行调用。当一个 Student 利用它的属性 “`{ "Patrick", 22, "red" }`” 被创建（或者说被构造）的时候，这些数值就可以访问了，表 3-17 列出了关键字的值。

表 3-17 关键字数值

|   |           |
|---|-----------|
| <code>somebody.name</code> or <code>somebody["name"]</code>     | "Patrick" |
| <code>somebody.age</code> or <code>somebody["age"]</code>       | 22        |
| <code>somebody.colour</code> or <code>somebody["colour"]</code> | "red"     |

因为利用 name 增加了 Student 的属性，所以它们必须通过 name 来被引用。如果使用数组，那么就必须使用数组来引用，比如要用 `somebody[0]`，而不用 `somebody.name`。JavaScript 的数组和对象都允许这两种方法。另外，数组和对象都有一个本质的属性，即 `length`（长度）。在编写程序的时候，可以这样来使用它 “`objectName.length`”，它返回对象包含的元素或者属性的个数。

属性同样也可以从函数中创建。这样，只需增加一行到 **Student** 函数中就可以创建属性了：  
`function Student(name, age, colour) { this.name = name; this.age = age; this.colour = colour; this.birthYear = (new Date()).getFullYear() - this.age; }`。以上的 **Student** 函数定义了第四个属性：**birthYear**，它代表每一个 **Student** 的出生年份。这里要注意，增加的这一行是调用一个内置的 **Date** 构造器，它返回一个包含了当前日期和时间的 **Date** 对象，这是一个非常懒惰访问一个 **Date** 变量的方法。所以使用下面的代码会显得更准确些、更有可读性：`var today = new Date(); this.birthYear = today.getFullYear() - this.age;`。当然，有许多方法来对 JavaScript 程序进行“压缩”。通过插入对象到代码中，你就可以处理绝大多数的变量。这个 **Date** 对象比起上面定义的 **Student** 对象来说更复杂，因为它包括了访问方法（**accessor/get**）以及操作方法（**manipulator/set**）。同时，增加简单的对象方法到 JavaScript 中是可能的，下面是详细的代码：

```
<SCRIPT language="javascript"> <!-- Hide from older browsers
function Student(name, age, colour)
{ this.name = name; this.age = age; this.colour = colour; this.birthYear = (new
Date()).getFullYear() - this.age; this.toString = printStudent;
// 这里定义 the Student.toString() 方法
this.isOlder = isOlder; // 这里定义 Student.isOlder(Student) 方法
}
function printStudent()
{ var text = this.name + "was born in" + this.birthYear + "<br>"; text += "and is" +
this.age + " years old.<p>"; return text; }
function isOlder(otherStudent)
{
    // 这里定义是否第一个人是更老的
    return (this.age > otherStudent.age); // 返回布尔型数值
}
//下面的代码用于测试的函数
var body1 = new Student("Patrick", 22, "red"); var body2 = new Student("Betty", 21,
"green");
document.write(body1); //这里为 Student.toString() 创建一个调用
document.write(body2);
document.write(body1.name);
document.write((body1.isOlder(body2)) ? " is " : " is not "); // 是否更老?
document.write("older than " + body2.name); // Stop hiding -->
</SCRIPT>
```

通过为 **Student** 对象重载 **Student.toString()** 方法，可以将 **Student** 对象作为字符串显示出来。每当 **Student** 对象被作为字符串引用的时候，**printStudent** 返回的数值就决定了该显示什么。上面脚本的输出如下所示：

```
Patrick was born in 76 and is 22 years old. Betty was born in 77 and is 21 years old.
Patrick is older than Betty
```

从输出的结果可以看到，年份是以两位数字表示的，如“1976 年”只用“76”来表示。你可能想增加“19”到这两位数字的前面，不幸的是，“千年虫”的问题使得处理起来有点棘手。而不同的浏览器处理 `Date` 对象的 `getFullYear()` 方法是不同的，如表 3-18 所示。

表 3-18 `getFullYear` 方法

| Year ( 年份 ) | Navigator 浏览器 | IE 3.x 浏览器 | IE 4.x 浏览器 |
|-------------|---------------|------------|------------|
| 1998        | 98            | 98         | 98         |
| 1999        | 99            | 99         | 99         |
| 2000        | 2000          | 100        | 2000       |
| 2001        | 2001          | 101        | 2001       |

从上面表格中发现最新的浏览器支持 1999 年之后四位的年份格式（如 2000、2001）。不幸的是，老的浏览器处理年份的格式的时候给程序设计人员无尽的困惑。我们可以这样进行处理：

```
var thisYear = (new Date()).getFullYear();
thisYear = 1900 + (thisYear % 1900); //将年份转换为 IE 3.x 格式并且增加 1900 以得到真实的
//年份
```

比如，2002 年经过“`this Year%1900`”转换为 IE 3.x 格式 102，然后 102 再加上 1900 得到 2002。上面这两条语句可以适用于从 1900 年到 3799 年的处理，现在看起来这么长的时间是足够使用的。但是，为了避免类似于“千年虫”问题，应该再找出更好的解决方案。另外，新的 ECMA 标准包括了一个名为 `getFullYear()` 的函数，它返回完整的年份格式，但是这个函数只能被 Navigator 4 支持，在 IE 中是不能使用的。最后对程序再做一点改进，可以修改 `People` 构造器以转换年份为 YYYY 格式，具体代码如下：`this.birthYear = 1900 + (((new Date()).getFullYear() - this.age) % 1900);`。在 JavaScript 中使用对象的能力通常被许多程序设计人员所忽视，但是从本教程中，你应该可以体会到使用对象可以使程序员设计出功能更强大的应用程序。

## 3.11 JavaScript 访问样式属性

DOM（文档对象模型）的 `style` 对象是 HTML 对象的一个属性。`style` 对象提供了一组对应于浏览器所支持 CSS 属性的属性（比如 `color`、`fontWeight`、`fontStyle`）。每一个 HTML 对象都有一个 `style` 属性，可以使用这个属性访问 CSS 样式属性。在下面的示例中，当单击按钮时，将把 `<body>` 标签的字体样式修改为斜体。

例 3-29

```
.....
<body background="5_1_bg.jpg">
  <h1 name="header1" align="center">法拉利! (请注意这几个字)</h1>
```



```
<center><input type="button" onclick="styler()" value="请单击此处"/></center>
</body>
</html>
<script type="text/javascript">
    function styler(){
        document.body.style.fontStyle = "italic";
    }
</script>
```

在上例中：

- 定义函数 **styler()**，其功能是修改<body>标签的字体样式，也就是改变文档的默认字体样式。
- **style** 是 **body** 对象的一个属性，而 **fontStyle** 是 **style** 对象的一个属性。这条语句的功能是将 **body** 对象的 **fontStyle** 属性修改为 **italic**（斜体）。
- 定义一个按钮，并使用 **styler()**函数响应该按钮的单击操作，效果如图 3-34 所示。



图 3-34 使用 style 属性

JavaScript 的 **style** 对象提供了一组与 CSS 属性相对应的属性，这些属性对应 CSS 样式中的属性。为了方便使用，表 3-19 列出了 **style** 对象的常用属性。

表 3-19 style 对象的常用属性

| 属性                   | 意义     |
|----------------------|--------|
| font                 | 字体     |
| fontFamily           | 字体体系   |
| fontStyle            | 字体样式   |
| fontVariant          | 字体变体   |
| fontWeight           | 字体粗细   |
| backgroundAttachment | 背景附着方式 |
| backgroundImage      | 背景图像   |
| backgroundPosition   | 背景显示位置 |
| backgroundRepeat     | 背景平铺方式 |

(续表)

| 属性              | 意义     |
|-----------------|--------|
| backgroundColor | 背景颜色   |
| color           | 文字颜色   |
| letterSpacing   | 文字间隔   |
| textAlign       | 文字对齐方式 |
| textDecoration  | 文字装饰   |
| textIndent      | 文字缩进   |
| verticalAlign   | 垂直对齐方式 |
| wordSpacing     | 单词间距   |
| borderStyle     | 边框样式   |
| borderWidth     | 边框宽度   |
| margin          | 四周空白   |
| marginTop       | 上部空白   |
| marginBottom    | 下部空白   |
| marginLeft      | 左部空白   |
| marginRight     | 右部空白   |
| padding         | 内空白    |
| paddingTop      | 上部内空白  |
| paddingBottom   | 下部内空白  |
| paddingLeft     | 左部内空白  |
| paddingRight    | 右部内空白  |
| height          | 高度     |
| width           | 宽度     |

对照 CSS 属性和 style 对象的属性，在 CSS 属性中使用短横线（减号）分隔其名称的属性名，在 style 对象的属性中去掉了这个短横线。比如，在 CSS 中，属性 font-size、background-color、word-spacing 中都有短横线，而它们对应的 style 对象的属性 fontSize、backgroundColor、wordSpacing 都去掉了短横线。使用这种命名方法的原因是，JavaScript 的标识中不能包含短横线，并且标识符中包含单词时，后面单词的首字母要大写。

## 3.12 常用事件

在前面的章节中，已经多次使用了事件处理程序，只是没有明确介绍事件和事件处理程序，这里

将详细讲解。

事件是某些动作发生时产生的信号。**JavaScript** 能够感知这些信号，因此能够编写代码来响应这些信号，也就是响应这些事件。

**JavaScript** 事件是一种异步事件，也就是说，这些事件随时都可能发生。引起事件发生的动作称为触发事件，比如，当鼠标指针掠过某个按钮、用户单击了某个超链接、用户选择了某个复选框、用户在文本框中输入了某些文字时，都会触发相应的事件。

绝大多数情况下，都是由于用户对页面元素的操作而引发事件。但是，在加载页面、卸载页面时也会触发一些事件。事件是让 **HTML** 页面充满活力的源泉，也是运行 **JavaScript** 代码的主要途径。了解和掌握各种事件的触发时机，是活用 **JavaScript** 的基础。

事件处理程序的基本语法是：

```
name_of_handler = "javascript 代码或调用函数";
```

例如：

```
onMouseDown="alert('javaScript 入门到精通!') ";
```

再如：

```
<input type="button" ... onClick="check()"/>
```

常用事件如表 3-20 所示，制作网页时常用事件是学习的重点。

表 3-20 常用事件

| 事件名称        | 含义     | 作用对象                                | 详细说明   |
|-------------|--------|-------------------------------------|--|
| onClick     | 鼠标单击   | link、button、submit、reset、form、image | 已经很熟悉，如单击按钮、图片、文本框、列表框等                                    |
| onChange    | 内容发生改变 | input、select 和多行文本框（textarea）       | 如文本框的内容发生改变  |
| onBlur      | 失去焦点   | window、frame 所有表单对象（比如按钮、单选框）       | 当光标从某一控件移出时产生  |
| onFocus     | 获得焦点   | window、frame 所有表单对象（比如按钮、单选框）       | 如单击文本框时，该文本框就获得焦点（鼠标），产生 onFocus（获得焦点）事件                   |
| onMouseOver | 鼠标悬停事件 | link、链接中的图像                         | 当移动鼠标，停留在图片或文本框等的上方，就产生 onMouseOver（鼠标悬停）事件                |
| onMouseOut  | 鼠标移出事件 | link、链接中的图像                         | 当移动鼠标，离开图片或文本框的区域，就产生 onMouseOut（鼠标移出）事件                   |
| onMouseMove | 鼠标移动事件 | link、链接中的图像                         | 当鼠标在图片或层<DIV>、<SPAN>等 HTML 元素上方移动时，将产生 onMouseMove（鼠标移动）事件 |

(续表)

| 事件名称        | 含义       | 作用对象                | 详细说明   |
|-------------|----------|---------------------|--|
| onLoad      | 页面加载事件   | body、frameset、image | HTML 网页从网站服务器下载到本机后, 需要浏览器加载到内存中, 然后解释执行并显示。浏览器加载 HTML 网页时, 将产生 onLoad (页面加载) 事件 |
| onUnload    | 页面关闭事件   | body、frameset       | 当文档被关闭或重置时调用   |
| onMouseDown | 鼠标按下事件   | link、链接中的图像         | 当用户用任何鼠标按键单击对象时触发  |
| onMouseUp   | 鼠标弹起事件   | link、链接中的图像         | 当用户在鼠标位于对象之上时, 释放鼠标按键时触发   |
| onResize    | 窗口大小改变事件 | window              | 当用户改变窗口大小时产生, 如窗口最大化、窗口最小化、用鼠标拖动改变窗口大小等时   |

例 3-30

```

<html>
<head>
<title>onBlur 和 onFocus 的运用</title>
<style type="text/css">
    //平面文本框样式(了解)
    input{
        background-color:#0099FF;
        font-size:20px;
        border:1px solid;
    }
</style>
</head>
<body bgcolor="#FFFF66">
<form name="myform">
    <table align="center" width="400" border="0" cellspacing="0"
        cellpadding="0">
        <caption align="center"><h2>onBlur 和 onFocus</h2></caption>
        <tr>
            <td width="161" align="right">卡号: </td>
            <td width="239">
                <input type="text" name="card" id="card"
                    onblur="myfun2()" onfocus="myfun1()"
                    value="请注意格式:10XXXXXX" />
            </td>
        </tr>
    </table>
</form>

```

```

<tr>
  <td align="right">密码: </td>
  <td>
    <input type="text" name="pass" id="pass" />
  </td>
</tr>
</table>
</form>
</body>
</html>
<script type="text/javascript">
  function myfun1() { //文本获得鼠标焦点时 (onFocus) 调用的函数
    if (document.myform.card.value == "请注意格式:10XXXXXX") {
      document.myform.card.value = ""; //鼠标单击时清空文本框
    }
  }
  function myfun2() { //文本框失去鼠标焦点时 (onBlur) 调用的函数
    var a = document.myform.card.value;
    //判断是否是 10 打头, 并且为数字
    if (a.substr(0,2) != "10" || isNaN(a)) {
      alert("格式错误 , 请重新输入! ");
      document.myform.card.focus(); //让鼠标光标重新回到卡号文本框
      document.myform.card.select(); //并选中已输入的内容
    }
  }
}
</script>

```

当用户单击文本框时, 文本框获得鼠标的光标, 提示用户输入。这时习惯性地称该文本框得到焦点, 产生了获得焦点 **onFocus** 事件; 同样, 当用户填完数据, 鼠标移动到另一个输入框时, 习惯性地称该文本框失去焦点, 产生了失去焦点 **onBlur** 事件。

有的网页的文本框输入要求具有一定的格式, 很多网站将提示信息显示在文本框中, 当用户单击文本框准备输入时, 文本框中的提示信息将自动消失。例 3-30 的运行效果如图 3-35 所示。

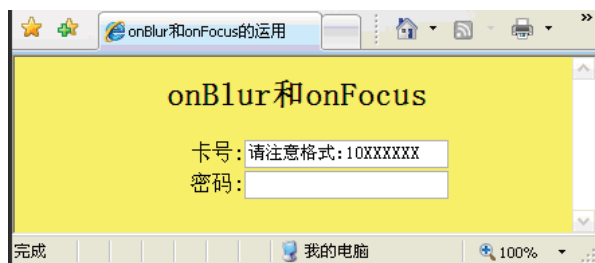


图 3-35 onBlur 和 onFocus 的运用

## 3.13 JavaScript 模块化和命名空间管理

### 3.13.1 模块化

模块化和编码思想与代码管理的便利度相关。其实，模块化思想和面向对象的思想基本相同，只不过所谓的“模块”是比“对象”更大而已。模块化致力于把完成同一个目的的功能函数通过良好的封装组合起来，并且保证其良好的复用性，在 **C#**、**Java** 等语言中把这样一个组合代码片段的思想称为面向对象的思想。这样做的好处有很多，如易用性、通用性、可维护性、可阅读性，等等。

而模块化无非就是在面向对象上的面向模块而已。把和同一个项目（模块）相关的功能封装有机地组合起来，通过一个共同的名字来管理，这就是模块化的思想。所以，相比面向对象而言，在代码架构上贯彻模块化的思想其实比面向对象的贯彻还更为容易一些。

不像 **C#**、**Java** 等这种本身就拥有良好模块化和命名空间机制的强类型语言。**JavaScript** 并没有为创建和管理模块提供任何语言功能。正因为这样，在做 **js** 的编码的某些时候，对于所谓的命名空间（**namespace**）的使用会显得有些过于随便，比如下面的代码：

```
var test = {} // namespace
(function(){
test.Class1 = function () {
    //TODO
}
...
test.Class2 = function () {
    //TODO
}
})();
```

如上，用一个全局变量或者全局对象就作为的命名空间（**namespace**），如此简单，甚至显得有些随便地委以它这么重大的责任。

在做一些项目的时候，或者建一些规模不大的网站时，简单地用这种方式来做命名空间的工作其实也够了，基本不会出什么大乱子。但是回归本质，如果有代码洁癖或者是在建立一个大规模的网站，亦或是一开始就抱着绝对优雅的态度和逻辑来做代码架构的话，或许该考虑更好一些的命名空间的注册和管理方式。

好了，回归正题，如上的方式，简单地通过全局对象来做命名空间已经能够很好地减少全局变量，规避变量名污染的问题。但是一旦网站规模变大，或者项目很多的时候，管理多个全局对象的命名空间依然会有问题。如果不巧发生了名字冲突，一个模块就会覆盖掉另一个模块的属性，导致其一或者两者都不能正常工作。而且出现问题之后，要去甄别也挺麻烦。所以需要一套机制或者工具，能在创

建命名空间的时候就判断是否已有重名。

另一方面，不同模块，亦即不同命名空间之间其实也不能完全独立，有时候也需要在不同命名空间下建立相同的方法或属性，这时方法或属性的导入和导出也会是个问题。

### 3.13.2 命名空间管理

由于命名空间是一个对象，拥有对象应该有的层级关系，所以在检测命名空间的可用性时，需要基于这样的层级关系去判断和注册，这在注册一个子命名空间时尤为重要。比如新注册了一个命名空间为 **test**，然后需要再注册一个命名空间为 **test.one**，其中，**one** 这个命名空间是 **test** 的子命名空间，它们应该拥有父子的关系。所以，在注册命名空间的时候需要通过 **'.'** 来分隔，并且进行逐一对应的判断。所以，注册一个命名空间的代码大概如下：

```
//建立命名空间，并返回主层命名空间
Module.createNamespace = function (name, version) {
    if (!name) throw new Error('name required');
    if (name.charAt(0) == '.' || name.charAt(name.length-1) == '.' ||
        name.indexOf('..') != -1) throw new Error('illegal name');
    var parts = name.split('.');
    var container = Module.globalNamespace;
    for (var i=0; i<parts.length; i++) {
        var part = parts[i];
        if (!container[part]) container[part] = {};
        container = container[part];
    }

    var namespace = container;
    if (namespace.NAME) throw new Error('module "'+name+'" is already defined');
    namespace.NAME = name;
    if (version) namespace.VERSION = version;
    Module.modules[name] = namespace;
    return namespace;
};
```

上面的 **Module** 是注册和管理命名空间的一个通用 **Module**，它本身作为一个“基模块”，拥有一个 **modules** 的模块队列属性，用来存储新注册的命名空间，正因为有了这个队列，才能方便地判断命名空间什么时候已经被注册了。

```
var Module;
if (!!Module && (typeof Module != 'object' || Module.NAME)) throw new Error("Name Space 'Module' already Exists!");
```

```
Module = {};
Module.NAME = 'Module';
Module.VERSION = 0.1;
Module.EXPORT = ['require',
    'importSymbols'];
Module.EXPORT_OK = ['createNamespace',
    'isDefined',
    'modules',
    'globalNamespace'];

Module.globalNamespace = this;
Module.modules = {'Module': Module};
```

上面代码最后一行就是一个命名空间队列，所有新建的命名空间都会放到里面去。结合先前的一段代码，基本就能很好地管理命名空间了。下面是一个创建命名空间的测试：

```
Module.createNamespace('test', 0.1); //注册一个名为 test 的 namespace，版本为 0.1
```

可以发现，新注册的 **test** 也添进了 **Module** 的 **modules** 队列里。大家也发现了，**Module** 里还有 **require**、**isDefined**、**importSymbols** 几个方法。

**require**（检测版本）和 **isDefined**（检测 **namespace** 是否已经注册）这两个方法并不难。

```
Module.isDefined = function (name) {
    return name in Module.modules;
};

Module.require = function (name, version) {
    if (!(name in Module.modules)) throw new Error('Module '+name+' is not defined');
    if (!version) return;

    var n = Module.modules[name];
    if (!n.VERSION || n.VERSION < version) throw new Error('version '+version+' or greater is required');
};
```

由于要的是一个通用的命名空间注册和管理的工具，所以在做标记导入或导出的时候，需要考虑到可配置性，不能全部导入或导出。所以就有了 **Module** 模板中的 **EXPORT** 和 **EXPORT\_OK** 两个 **Array**，作为存储允许导出的属性或方法的标记队列。其中，**EXPORT** 为 **public** 的标记队列，**EXPORT\_OK** 为可以自定义的标记队列，如果不需要分这么清楚，也可以只用一个标记队列，用来存放允许导出的标记属性或方法。

有了标记队列，做的导出操作就只针对 **EXPORT** 和 **EXPORT\_OK** 两个标记队列中的标记。

```
Module.importSymbols = function (from) {
    if (typeof from == 'string') from = Module.modules[from];
```



```
var to = Module.globalNamespace; //dafault
var symbols = [];
var firstsymbol = 1;

if (arguments.length>1 && typeof arguments[1] == 'object' && arguments[1] !=
null) {
    to = arguments[1];
    firstsymbol = 2;
}

for (var a=firstsymbol; a<arguments.length; a++) {
    symbols.push(arguments[a]);
}

if (symbols.length == 0) {
    if (from.EXPORT) {
        for (var i=0; i<from.EXPORT.length; i++) {
            var s = from.EXPORT[i];
            to[s] = from[s];
        }
        return;
    } else if (!from.EXPORT_OK) {
        for (var s in from) {
            to[s] = from[s];
            return;
        }
    }
}

if (symbols.length > 0) {
    var allowed;
    if (from.EXPORT || form.EXPORT_OK) {
        allowed = {};
        if (from.EXPORT) {
            for (var i=0; i<form.EXPORT.length; i++) {
                allowed[from.EXPORT[i]] = true;
            }
        }
        if (from.EXPORT_OK) {
            for (var i=0; i<form.EXPORT_OK.length; i++) {
                allowed[form.EXPORT_OK[i]] = true;
            }
        }
    }
}
```

```

        }
    }
}

for (var i=0; i<symbols.length; i++) {
    var s = symbols[i];
    if (!(s in from)) throw new Error('symbol '+s+' is not defined');
    if (!!allowed && !(s in allowed)) throw new Error(s+' is not public,
cannot be imported');
    to[s] = form[s];
}
}

```

`importSymbols` 这个方法中第一个参数为导出源空间，第二个参数为导入目的空间（可选，默认是定义的 `globalNamespace`），后面的参数也是可选，为想导出的具体属性或方法，默认是标记队列里的全部。

```

Module.createNamespace('test');
Module.createNamespace('one', 0.1);
me.EXPORT = ['define']
me.define = function () {
    this.NAME = '__one';
}
Module.importSymbols(one, test); //把 one 命名空间下的标记导入到 test 命名空间下

```

通过上例可以看到，本来定义在 `one` 命名空间下的方法 `define()` 就被导入到了 `test` 命名空间下。当然，这里说的导入、导出，其实只是复制，在 `one` 命名空间下依然能访问和使用 `define()` 方法。

## 3.14 正则表达式

让读者能把数小时辛苦而且易错的文本处理工作压缩在几分钟（甚至几秒钟）内完成，以前不可能的事，自从有了正则表达式后成为了现实。

### 3.14.1 正则表达式介绍

正则表达式最早是由数学家 **Stephen Kleene** 于 1956 年提出的，他是在对自然语言的递增研究成果的基础上提出来的。具有完整语法的正则表达式使用在字符的格式匹配方面，后来被应用到熔融信息技术领域。自从那时起，正则表达式经过几个时期的发展，现在的标准已经被 ISO（国际标准化组织）批准和被 **Open Group** 组织认定。

正则表达式并非一门专用语言，但它是可用于在一个文件或字符里查找和替代文本的一种标准。它具有两种标准：基本的正则表达式（BRE），扩展的正则表达式（ERE）。ERE 包括 BRE 功能和另外其他的概念。

许多程序中都使用了正则表达式，包括 `xsh`、`egrep`、`sed`、`vi` 以及在 UNIX 平台下的程序。它们可以被很多语言采纳，如 `HTML` 和 `XML`，这些采纳通常只是整个标准的一个子集。

### 3.14.2 正则表达式作用

简单的说，正则表达式是一种可以用于模式匹配和替换的强有力的工具。可以在几乎所有的基于 UNIX 系统的工具中找到正则表达式的身影，如 `vi` 编辑器、`Perl` 或 `PHP` 脚本语言以及 `awk` 或 `sed shell` 程序等。由此可见，正则表达式已经超出了某种语言或某个系统的局限，成为人们广为接受的概念和功能。

正则表达式可以让用户通过使用一系列的特殊字符构建匹配模式，然后把匹配模式与数据文件、程序输入以及 `Web` 页面的表单输入等目标对象进行比较，根据比较对象中是否包含匹配模式，执行相应的程序。

举例来说，正则表达式的一个最为普遍的应用就是用于验证用户在线输入的邮件地址的格式是否正确。如果通过正则表达式验证用户邮件地址的格式正确，用户所填写的表单信息就将会被正常处理；反之，如果用户输入的邮件地址与正则表达的模式不匹配，将会弹出提示信息，要求用户重新输入正确的邮件地址。由此可见，正则表达式在 `Web` 应用的逻辑判断中具有举足轻重的作用。

正则表达式是由普通字符（如字符 `a~z`）以及特殊字符（称为元字符）组成的文字模式。该模式描述在查找文字主体时，待匹配的一个或多个字符串。正则表达式作为一个模板，将某个字符模式与所搜索的字符串进行匹配。正则表达式使用正斜杠进行分隔，如 `“/abc/”` 就定义了一个简单的正则表达式。

- 测试字符串的某个模式。例如，可以对一个输入字符串进行测试，看该字符串是否存在一个电话号码模式或一个信用卡号码模式，这称为数据有效性验证。
- 替换文本。可以在文档中使用一个正则表达式来标识特定文字，然后可以全部将其删除或者替换为别的文字。
- 根据模式匹配从字符串中提取一个子字符串。可以用来在文本或输入字段中查找特定文字。

例如，如果需要搜索整个 `Web` 站点来删除某些过时的材料并替换某些 `HTML` 格式化标记，则可以使用正则表达式对每个文件进行测试，看在该文件中是否存在所要查找的材料或 `HTML` 格式化标记。用这个方法，就可以将受影响的文件范围缩小到包含要删除或更改的材料的那些文件。然后可以使用正则表达式来删除过时的材料，最后，可以再次使用正则表达式来查找并替换那些需要替换的标记。

JavaScript 提供了大量元字符用于控制和表达匹配模式。利用匹配模式，可以查找只包含数字、只包含字母或包含字母与数字的字符串，也可以查找以数字开头、后跟任意多个字母、以数字结尾的字符串，等等。

### 3.14.3 RegExp 对象

构造函数 `RegExp()` 有一个或两个字符串参数，它将创建一个新的 `RegExp` 对象。该构造函数的第一个参数是包含正则表达式主体的字符串，即稍候介绍的正则表达式文字量中出现在斜线对之间的文本。第二个参数是可选的，如果提供了此参数，它说明的就是该正则表达式的标志（`i`、`g` 或 `m`）。

这种创建正则表达式的方法常用于按照用户输入构造正则表达式或在程序运行过程中改变正则表达式的场合，这是一种动态方法，一般来说，运行速度要比文字量创建的正则表达式略慢一些。

使用构造函数 `RegExp()` 创建正则表达式的一般格式为：

```
var regexname=new RegExp("pattern"[,"flags"]);
```

其中，`regexname` 是变量名称，该变量用于保存新创建的正则表达式；`patten` 为指定匹配模式的正则表达式；`flags` 是零个或多个可选项，方括号表示 `flags` 参数可以省略。这里，`pattern` 和 `flags` 参数都可以是字符串变量。例如：

```
var country = new RegExp("中国");
var reobj = new RegExp("loveme","ig");
var str = "中*国";
var country1 = new RegExp(str); //使用字符串变量定义正则表达式
```

正则表达式对象 `RegExp` 提供了两个方法：`test()` 和 `exec()` 方法。这两个函数的语法分别如下。

```
test(string)
```

功能为：测试字符串 `string` 是否包含了匹配该正则表达式的子串，如果包含了这样的子串，那么函数返回 `true`，否则返回 `false`。

```
exec(string)
```

功能为：在字符串 `string` 中进行匹配搜索，并将结果保存在一个数组中返回；如果没有找到匹配的子串，那么返回 `null`。

### 3.14.4 正则表达式语法参考

正则表达式就是一个字符模式。与 `String` 对象类似，JavaScript 的正则表达式也是一个对象，它主要用于字符串的模式匹配。创建正则表达式有两种方法：文字量方法和使用构造函数 `RegExp()`。

#### 1. 使用文字量方法创建正则表达式

使用文字量方法创建正则表达式的方法为：将文字量的正则表达式赋值给一个变量。正则表达式是包含在两个斜杠之间的一个或多个字符，在后一个斜杠后面，可以指定一个或多个选项。例如，`/abc/`、

/abc/i、/ab+c/、/^d+\$/、/abc/gi 都是正则表达式。

使用文字量方法创建正则表达式的一般格式为：

```
var regexname=/pattern/flags
```

其中，**regexname** 是正则表达式对象实例名，用于保存新创建的正则表达式；**pattern** 为该正则表达式对象实例的匹配模式；**flags** 是可选参数，其可选项及意义如下。

- i: 忽略大小写，也就是说在对字符串进行正则匹配时，不区分大小写；
- g: 全局匹配，注明该选项时，正则表达式将匹配字符串中出现的所有匹配模式；
- m: 进行多行匹配。

### 3.14.5 String 对象中与正则表达式有关的方法

在 **String** 对象中和正则表达式有关的方法如表 3-21 所示。

表 3-21 String 对象中正则表达式方法

| 方法            | 说明  |
|---------------|---|
| match(regex)  | 使用指定的正则表达式匹配字符串，并返回包含匹配结果的数组。如果没有匹配结果，则返回 null。如果 regex 不是全局正则表达式，那么返回的数组与 RegExp.exec() 方法的执行结果相同；如果 regex 是全局正则表达式（包含了“g”属性），返回数组的元素包含了每一个匹配的结果 |
| search(regex) | 返回与 regex 匹配的字符串的开始位置。如果不匹配那么返回-1   |

下面让我们一起来领略一下正则表达式带来的震撼吧！

#### （1）身份证号码验证

- 身份证号码是 18 位数字，根据 GB11643-1999《居民身份证》定义制作；由 17 位本体码和一位校验码组成。
- 身份证号码前 6 位是地址码，按（GB/T2260）规定执行。
- 接着 8 位是年、月、日。
- 后三位是同年同月同日出生的人的顺序号，奇数表示男，偶数表示女。
- 最后一位是校验码。
- 因此正则表达式验证模式为：/^d{17}(d|X)\$/。

如图 3-36 所示为身份证号码验证的界面。



图 3-36 身份证号码合法性验证

要实现如图 3-36 所示效果的代码如下所示：

```
<html>
<head>
<title>用正则表达式验证身份证的合法性</title>
</head>
<body bgcolor="#FFCC99">
<form action="" method="get" name="myForm">
<table width="400" border="0" align="center" cellpadding="0" cellspacing="0">
  <caption align="center"><h2>验证身份证的合法性</h2></caption>
  <tr>
    <td width="214" align="right">身份证号:</td>
    <td width="186">
      <input type="text" name="cardid" size="18" />
    </td>
  </tr>
  <tr>
    <td colspan="2" align="center">
      <input name="check" onclick="checkCard()" type="button" id="check"
value="检测身份证号" />
      <input name="reset" type="reset" id="reset" value="重置" />
    </td>
  </tr>
</table>
</form>
</body>
</html>
<script type="text/javascript">
//验证身份证号
var vcity={ 11:"北京",12:"天津",13:"河北",14:"山西",15:"内蒙古",21:"辽宁",
```

```

22:"吉林",23:"黑龙江",31:"上海",32:"江苏",33:"浙江",34:"安徽",
35:"福建",36:"江西",37:"山东",41:"河南",42:"湖北",43:"湖南",44:"广东",
45:"广西",46:"海南",50:"重庆",51:"四川",52:"贵州",53:"云南",54:"西藏",
61:"陕西",62:"甘肃",63:"青海",64:"宁夏",65:"新疆",71:"台湾",
81:"香港",82:"澳门",91:"国外"};
function checkCard(){
    var isum=0;
    var re=/^[1-9]\d{5}[1-9]\d{3}((0\d)|(1[0-2]))((([0|1|2]\d)|3[0-1])((\d{4})|\d{3}[A-Z]))$/;    var cardidstr=document.getElementById('cardid').value;
    if (cardidstr.length>0 && cardidstr!=null) {
        if(!re.test(cardidstr)) {
            alert("身份证号不符合要求!");
            return false;
        }
        //检查地址是否符合要求
        if(vcity[parseInt(cardidstr.substr(0,2))]==null) {
            alert("身份地址不符合要求!");
            return false;
        }
        //出生日期
        Sbirthday = cardidstr.substr(6,4) + "-" + cardidstr.substr(10,2)+"-"+
cardidstr.substr(12,2);
        alert('身份证地址:'+vcity[parseInt(cardidstr.substr(0,2))]+"\\n"
        +"出生日期:"+sbirthday+"\\n"+"性别:"+ (cardidstr.substr(16,1)%2?"男":"女"));
    }
    else{
        alert("请输入身份证号!");
    }
}
</script>

```

## (2) 普通注册验证

校验用户姓名：只能输入 1~30 个以字母开头的字符串。

校验密码：只能输入 6~20 个字母、数字、下画线。

```

function isTrueName(s)
{
    var patrnr=/^[a-zA-Z]{1,30}$/;
    if (!patrnr.exec(s)) {
return false;
    }
    return true;
}

```

```
}  
function isPasswd(s)  
{  
    var patrn=/^(\w){6,20}$/;  
    if (!patrn.exec(s)) {  
return false;  
    }  
    return true ;  
}
```

### 3.14.6 常见的验证方式

(1) 校验是否全由数字组成。

```
function isDigit(s)  
{  
    var patrn=/^[0-9]{1,20}$/;  
    if (!patrn.exec(s)) {  
        return false;  
    }  
    return true;  
}
```

(2) 校验用户姓名：只能输入 1~30 个以字母开头的字符串。

```
function isTrueName(s)  
{  
    var patrn=/^[a-zA-Z]{1,30}$/;  
    if (!patrn.exec(s)) {  
return false;  
    }  
    return true;  
}
```

(3) 校验密码：只能输入 6~20 个字母、数字、下画线。

```
function isPasswd(s)  
{  
    var patrn=/^(\w){6,20}$/;  
    if (!patrn.exec(s)) {  
return false;  
    }  
    return true ;  
}
```



```
}
```

(4) 校验普通电话、传真号码：可以以“+”开头，除数字外，可含有“-”。

```
function isTel(s)
{
    var patrn=/^[+]{0,1}(\d){1,3}[ ]?([-]?((\d)|[ ]){1,12})+$/;
    if (!patrn.exec(s)) {
return false ;
    }
    return true;
}
```

(5) 校验 IP。

```
function isIP(s)
{
    var patrn=/^[0-9.]{1,20}$/;
    if (!patrn.exec(s)) {
return false;
    }
    return true
}
```

## 3.15 JavaScript 样式特效应用

### 1. 背景时钟

从本章中我们学习到了 JavaScript 编程的精髓，是不是有一种想写代码的冲动？那就让我们一起来完成背景时钟的制作吧。代码如下，将其保存为 HTML 文档。

```
<html>
<head>
<TITLE>背景时钟</TITLE>
<script language=JavaScript>
<!--//
function clockon() {
thistime= new Date()
//获取小时
var hours=thistime.getHours()
//获取分钟
var minutes=thistime.getMinutes()
//获取秒
var seconds=thistime.getSeconds()
```

```

//为个位数的时间前面加0
if (eval(hours) < 10) {hours="0"+hours}
if (eval(minutes) < 10) {minutes="0"+minutes}
if (seconds < 10) {seconds="0"+seconds}
thistime = hours+":"+minutes+": "+seconds
if(document.all) {
bgclocknoshade.innerHTML=thistime
bgclockshade.innerHTML=thistime
}
//document.layers Netscape 4.x 专有属性
if(document.layers) {
document.bgclockshade.document.write('<div id="bgclockshade"

style="position:absolute;visibility:visible;font-family:Verdana;color:FFAAAAA;font-
size:120px;top:10px;left:152px">'+thistime+'</div>')

document.bgclocknoshade.document.write('<div id="bgclocknoshade"

style="position:absolute;visibility:visible;font-family:Verdana;color:DDDDDD;font-
size:120px;top:10px;left:150px">'+thistime+'</div>')

document.close()
}
var timer=setTimeout("clockon()",200)
}
//-->
</script>
//引入CSS 样式
<link rel="stylesheet" href="../style.css"></head>
<body onLoad="clockon()" >
<div id="bgclockshade" style="position:absolute;visibility:visible;font-
family:Arial;color:FF8888;font-size:120px;top:102px;left:152px"></div>

<div id="bgclocknoshade" style="position:absolute;visibility:visible;font-
family:Arial;color:DDDDDD;font-size:120px;top:100px;left:150px"></div>

<div id="mainbody" style="position:absolute; visibility:visible">
</div>
</body>
</html>

```

**注意：**时钟显示的位置需要通过修正 top 和 left 参数来确定。top 表示层距离显示器顶部的像素值，left 表示距离左侧的像素值。

以网页方式打开，时钟会动态显示（参见图 3-37），去试试它的神奇吧！



图 3-37 背景时钟

## 2. 输入框样式特效

鼠标移过输入框，输入框背景色变色，实现代码如下：

```
<html>
<head>
  <title>输入框样式特效</title>
</head>

<body bgcolor="#FFCC99">
<table width="400" align="center">
  <caption align="center">
    输入框样式特效
  </caption>
  <tr>
    <td align="right">
      用户名：
    </td>
    <td>
      <!--为文本框添加鼠标移入、移出事件-->
      <input value="Type here" name="username" type="text"
        size="29" onmouseover="mouseOver(this)"
        onmouseout="mouseOut(this)"
        style="width: 106; height: 211; border:1px solid;"
        onfocus="clearText(this)" />
    </td>
  </tr>
</table>
</body>
</html>
```

```

<script type="text/javascript">
//鼠标移入时改变文本框样式
function mouseOver(input){
    input.style.borderColor = 'blue';
    input.style.backgroundColor = '#FFFF66';//黄色
}
//鼠标移出时恢复
function mouseOut(input){
    input.style.borderColor = '#000000';
    input.style.backgroundColor = '#ffffff';//白色
}
function clearText(input){
    input.value = "";
}
}
</script>

```

- 定义 **mosueOver** 函数，即当鼠标在文本框上面时，把文本框的边框色设置为蓝色，背景色设置为黄色。
- 定义 **mouseOut** 函数，即当鼠标光标移出文本框时，把文本框的边框设置为黑色，背景色恢复到白色。
- 定义 **clearText** 函数，即当鼠标光标焦点在文本框时，把文本框内的内容清除。运行效果如图 3-38（鼠标未移上去）和图 3-39（鼠标移上去）所示。



图 3-38 鼠标未移上去

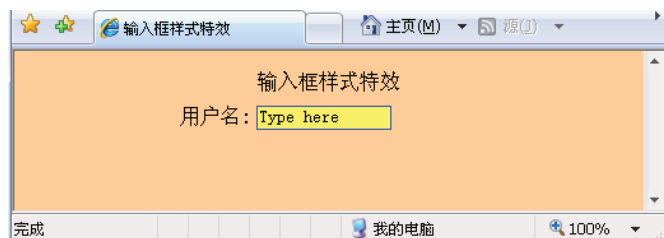


图 3-39 鼠标移上去

### 3. 背景改变的按钮

当鼠标移到按钮上时，按钮的背景色改变为另一种颜色；当鼠标移开后，按钮背景色又变回原来的颜色。实现代码如下：

```
<html>
<head>
<title>背景改变的按钮</title>
<style type="text/css">
    body{font-size:12px;}
    .free{
        width:283px;
        border:1px solid #C1C1C1;
        border-top:0;
        background:#F9F9F9 url(images/5_4_3.jpg) no-repeat center top;
    }
    /*鼠标移上时调用的样式*/
    .overStyle{
        background-image: url(images/5_4_1.jpg);
        color:#489379;
        width:102px;
        height:24px;
        text-align:center;
        border:0;
        font-weight:bold;
        clear:both;
        cursor:pointer;
        margin:0 0 0 96px;
    }
    /*鼠标移出时调用的样式*/
    .outStyle{
        background-image:url(images/5_4_2.jpg);
        color:#489379;
        width:102px;
        height:24px;
        text-align:center;
        border:0;
        font-weight:bold;
        clear:both;
        cursor:pointer;
        margin:0 0 0 96px;
    }
}
```

```

</style>
</head>
<body>
<table width="400" align="center" class="free">
  <tr><td colspan="2" height="80"></td></tr>
  <tr>
    <td>
      <select name="type">
        <option >会员名</option>
        <option selected="selected">邮箱名</option>
      </select>
    </td>
    <td><input name="email" type="text" size="15" >@sina.com</td>
  </tr>
  <tr>
    <td>密码</td>
    <td><input name="pass" type="text" size="15"></td>
  </tr>
  <tr>
    <td>&nbsp;</td>
    <td><input name="remem" type="checkbox">记住邮箱名</td>
  </tr>
  <tr>
    <td colspan="2">
      <input type="submit" value="登 录" class="outStyle"
        onmouseover="this.className='overStyle'"
        onmouseout="this.className='outStyle'" />
    </td>
  </tr>
</table>
</body>
</html>

```

本例中：

- 定义 **overStyle** 样式为鼠标移上时的样式，**outStyle** 鼠标移出时的样式。
- 在【登录】按钮的 **onmouseover** 和 **onmouseout** 时设置相应的样式，运行效果如图 3-40（鼠标移上“登录”按钮时）和图 3-41（鼠标移出“登录”按钮时）所示。



图 3-40 鼠标移上【登录】按钮时



图 3-41 鼠标移出【登录】按钮时

#### 4. 选项卡的制作

当在同一个页面中需要提供不同类别的浏览时，可以采用选项卡的形式。下面就将制作选项卡来完成不同内容的阅读。实现代码如下：

```
<html>
<head>
<title>Tab 选项卡</title>
<style type="text/css">
    //设置 id 是 tabNav 的 ul 的样式
    ul#tabNav{
        width:400px;
        list-style:none;
        border-bottom:solid 1px green;
        margin:0;
        padding-left:0;
        padding-bottom:26px;
        *padding-bottom:0;
    }
    //设置 id 是 tabNav 的 ul 下辖 li 的样式
    ul#tabNav li{
        float:left;
```

```

        height:25px;
        margin:0 10px -2px 0;//关键地方
        background-color:Green;
        color:black;
        border:solid 1px Green;
        border-bottom:0;
        padding:0;
    }
    ul#tabNav a:link,ul#tabNav a:visited{
        display:block;
        text-decoration:none;
        padding:5px 10px 3px 10px;
    }
    #tabContent{
        width:400px;
        height:200px;
        border:solid 1px green;
        border-top-width:0;
    }
    //选中后的样式
    #tabNav li.tabSelected
    {
        background-color:#FFCC00;
        color:#FFFFFF;
    }
</style>
<script type="text/javascript">
    function showContent(index){
        //根据事件获取相应的层
        var e = document.getElementById("content"+index);
        e.style.display = "";
        for(var i=1; i<4; i++){
            //判断层编号,并隐藏其他的层
            if(i != index){
                var e2 = document.getElementById("content"+i);
                e2.style.display = "none";
                document.getElementById("tab"+i).className="";
            }
            //改变选中层的类名,从而改变样式
            else{
                document.getElementById("tab"+i).className =

```



```

        "tabSelected";
    }
}
}
</script>
</head>
<body>
    <ul id="tabNav">
        <li onmouseover="showContent(1)" id='tab1' class="tabSelected">
            <a href="#" _fcksavedurl='#'>新闻</a>
        </li>
        <li onmouseover="showContent(2)" id='tab2' class="">
            <a href="#" _fcksavedurl='#'>体育</a>
        </li>
        <li onmouseover="showContent(3)" id='tab3' class="">
            <a href="#" _fcksavedurl='#'>娱乐</a>
        </li>
    </ul>
    <div id="tabContent">
        <div id="content1" >
            新闻内容
        </div>
        <div id="content2" style="display:none">
            体育内容
        </div>
        <div id="content3" style="display:none">
            娱乐内容
        </div>
    </div>
</body>
</html>

```

示例的实现原理就是当用户鼠标移动到类别标题时，显示对应的层并把其他类别对应的层给隐藏起来。运行效果如图 3-42 所示。

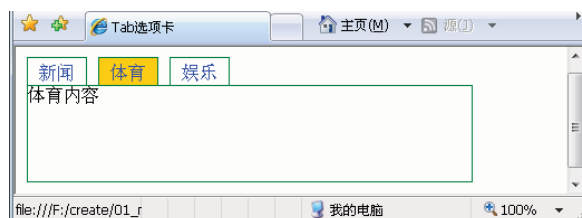


图 3-42 选项卡高亮显示

## 5. 菜单的制作

在浏览网页时，经常会遇到菜单，实现代码如下：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>二级菜单</title>
<style type="text/css">
/* 整体设置*/
#menu {
    margin:0;
    padding:0;
    width:610px;
    list-style-type:none;
    font:14px Arial;
}
#menu li {
    float:left;
    padding:0;
    margin:0 1px 0 0;
    width:150px;
}
/* 设置菜单项*/
#menu li dl {
    width:150px;/*ie6*/
    margin: 0;
    padding: 0 0 10px 0;
    background: #cb6 url(images/5_7_bottom.gif) no-repeat bottom left;
}
#menu li dt a,#menu li dd a{
    display:block;
}
/* 设置菜单项的 dt */
#menu li dt {
    margin:0;
    padding: 5px;
    text-align:center;
    border-bottom:1px solid #b00;
    background:
}
```

```
#menu li dt.orange {
    background:#fa5 url(images/5_7_top.gif) no-repeat top left;
}
#menu li dt.yellow {
    background:#ee5 url(images/5_7_top.gif) no-repeat top left;
}
#menu li dt.green {
    background:#5e5 url(images/5_7_top.gif) no-repeat top left;
}
#menu li dt.blue {
    background:#5cf url(images/5_7_top.gif) no-repeat top left;
}
#menu li dt a ,#menu li dt a:visited {
    display:block;
    color:#333;
    text-decoration:none;
}
/* 设置菜单项的 dd */
#menu li dd {
    margin:0;
    padding:0;
    color: #fff;
    background: #47a;
}
#menu li dd.last {
    border-bottom:1px solid #b00;
}
#menu li dd a, #menu li dd a:visited {
    display:block;
    color:#fff;
    text-decoration:none;
    padding:4px 5px 4px 20px;
    background: #47a url(images/5_7_arrow.gif) no-repeat 10px 10px;
    height:1em;
}
/*关闭子菜单*/
#menu li dd {
    display:none;
}
/* 设置鼠标响应 */
#menu li:hover dd , #menu li a:hover dd {
```

```

        display:block;
    }
    #menu li:hover, #menu li a:hover {
        border:0;
    }/*ie6*/
    #menu li dd a:hover {
        background: #147;
        color:#9cf;
    }
    /*针对 ie6 的设置*/
    #menu table {
        border-collapse:collapse;
        padding:0;
        margin:-1px;
        font-size:1em;
    }
</style>
</head>
<body>
<ul id="menu">
    <li>
        <dl>
            <dt class="orange"><a href="#">Artech Studio</a></dt>
            <dd><a href="#">Web Dev</a></dd>
            <dd><a href="#">Web Design</a></dd>
            <dd><a href="#">Books</a></dd>
            <dd class="last"><a href="#">Contact Us</a></dd>
        </dl>
    </li>
    <li>
        <dl>
            <dt class="yellow"><a href="#">Artech Store</a></dt>
            <dd><a href="#">Books</a></dd>
            <dd><a href="#">DVDs</a></dd>
            <dd><a href="#">Movies</a></dd>
            <dd class="last"><a href="#">Service</a></dd>
        </dl>
    </li>
    <li>
        <dl>
            <dt class="green"><a href="#">Artech Achi</a></dt>

```

```

        <dd><a href="#">Landscape</a></dd>
        <dd><a href="#">Plan</a></dd>
        <dd><a href="#">Design</a></dd>
        <dd class="last"><a href="#">Maps</a></dd>
    </dl>
</li>
<li>
    <dl>
        <dt class="blue"><a href="#">Artech Science</a></dt>
        <dd><a href="#">Physics</a></dd>
        <dd><a href="#">Maths</a></dd>
        <dd><a href="#">Chemistry</a></dd>
        <dd class="last"><a href="#">Courses</a></dd>
    </dl>
</li>
</ul>
</body>
</html>

```

菜单运行效果如图 3-43 所示。

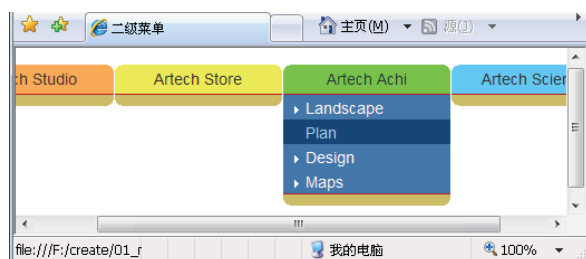


图 3-43 菜单效果

## 6. 操作表格及右键删除

通过前面的学习，相信读者已经掌握了 `document` 对象、`Element` 以及 `event` 对象的使用，下面通过一个综合的例子来理解这些对象在实际开发中的应用。

本例要实现对学员信息的添加和删除操作，并且删除要求使用右键菜单，鼠标移动要改变背景颜色。实现代码如下：

```

<html>
<head>
<title>学生信息管理</title>
<style type="text/css">
    body,table,tr,td,th,div,input{ font-size:12px;}
    .tr1{ background-color:#FFFFFF; }

```

```

        .tr2{ background-color:#FFCC66; }
</style>
<script type="text/javascript">
//设置时间联动
function setDay(){
    //获取年、月、日的下拉菜单
    var selYear = document.getElementById("year");
    var selMonth = document.getElementById("month");
    var selDay = document.getElementById("day");
    //获取年、月、日的值
    var year = selYear.value;
    var month = selMonth.value;
    var day=0;
    //判断月份从而确定天数
    if(month==1 || month==3 || month==5 || month==7 || month==8 || month==10 ||
month==12 ){
        day = 31;
    }
    else if(month == 4 || month == 6 || month == 9 || month == 11){
        day = 30;
    }
    else{
        if(((year%4==0) && (year%100!=0)) || year%400 ==0){
            day=29;
        }
        else{
            day =28;
        }
    }
    selDay.options.length=0; //将下拉列表框中原来的选项清空
    selDay.options.length=day;
    //生成日期
    for(var i=0;i<day;i++){
        selDay.options[i] = new Option(i+1,i+1);
    }
}
//记录删除行
var delRow=false;
//当鼠标点右键时显示删除菜单
function mousedown(tr){
    var e = event.button; //得到鼠标单击的键位

```

```

//1 左键, 2 右键, 4 中键
if (e==2) {
    var menu = document.getElementById("contextMenu");
    menu.style.display="block";
    //得到鼠标的位置
    var x = event.x;
    var y = event.y;
    menu.style.top = y;
    menu.style.left = x;
    delRow = tr;
}
}
//删除行
function deleteRow() {
    //得到表格
    var tab = document.getElementById("tab");
    //tr.sectionRowIndex //得到 tr 的索引编号
    tab.deleteRow(delRow.sectionRowIndex);
    var menu = document.getElementById("contextMenu");
    menu.style.display="none";
}
//添加数据
function addStu() {
    //取得输入的数据
    var id = document.getElementById("stuid").value;
    var name = document.getElementById("stuname").value;
    var score = document.getElementById("score").value;
    var bd = document.getElementById("year").value + "-" + document.
getElementById("month").value + "-" + document.getElementById("day").value;
    //在表格中创建行, 并将数据放入单元格中
    var tab = document.getElementById("tab");
    var row = tab.insertRow(); //在表格上面创建一行
    //给建的行注册事件
    row.onmouseover=function() {
        row.style.backgroundColor="#FFCC66";
    }
    row.onmouseout=function() {
        row.style.backgroundColor="#ffffff";
    }
    row.onmousedown=function() {
        var e = event.button; //得到鼠标单击的键位

```

```

//1 左键, 2 右键, 4 中键
if (e==2) {
    var menu = document.getElementById("contextMenu");
    menu.style.display="block";
    //得到鼠标的位置
    var x = event.x;
    var y = event.y;
    menu.style.top = y;
    menu.style.left = x;
    delRow = row;
}
}
//设置背景颜色
row.style.backgroundColor="#ffffff";
var td1 = row.insertCell();
td1.innerText=id;
var td2 = row.insertCell();
td2.innerText=name;
var td3 = row.insertCell();
td3.innerText=bd;
var td4 = row.insertCell();
td4.innerText=score;
}
</script>
</head>
<body>
<center>
学号: <input type="text" id="stuid" name="stuid" size="3" />&nbsp;
姓名: <input type="text" id="stuname" name="stuname" size="8" />&nbsp;
出生日期: <select id="year" onChange="setDay()" >
    <script>
        for(var i=1970;i<2010;i++){
            document.write("<option value="+i+">"+i+ "</option>");
        }
    </script>
</select>年
<select id="month" onChange="setDay()" >
    <script>
        for(var i=1;i<13;i++){
            document.write("<option value="+i+">"+i+"</option>");
        }
    </script>
    </select>月
    <select id="day" onChange="setDay()" >
    <script>
        for(var i=1;i<31;i++){
            document.write("<option value="+i+">"+i+"</option>");
        }
    </script>
    </select>日
    </body>
    </html>

```



```

        </script>
    </select>月
    <select id="day">
        <script>
            for(var i=1;i<32;i++){
                document.write("<option value="+i+">"+i+"</option>");
            }
        </script>
    </select>日
<br>
高考成绩: <input type="text" id="score" name="score" size="3" />&nbsp;
<input type="button" value="添加" onClick="addStu()" />
<hr>
<!-- 右键菜单 -->
<div id="contextMenu" style="position:absolute; padding:5px; width:70px; z-index:2;
background-color:#CCCCCC; display:none; border:1px solid #000000;">
    <label style="cursor:hand;" onClick="deleteRow()">删除</label>
</div>
<!--关闭 IE 本身的右键菜单 -->
<table onContextMenu="return false" id="tab" align="center" width="480" border="0"
cellspacing="1" bgcolor="#003366" cellpadding="5">
    <tr bgcolor="#CCCCCC">
        <th width="100">学号</th>
        <th width="200">姓名</th>
        <th width="300">出生年月</th>
        <th width="100">高考成绩</th>
    </tr>
    <tr bgcolor="#FFFFFF" onMouseDown="mousedown(this)"
onMouseOver="this.className='tr2'" onMouseOut="this.className='tr1'">
        <td>001</td>
        <td>007</td>
        <td>1977-7-7</td>
        <td>77</td>
    </tr>
</table>
</center>
</body>
</html>

```

实现效果如图 3-44 所示。



图 3-44 操作表格及右键删除

在实现中使用层“contextMenu”来实现右键菜单，需要注意的是，在表格使用右键菜单的时候，必须设置表格属性如下：

```
<table onContextMenu="return false" id="tab" .....
```

另外，通过 event 事件源来确定鼠标的位置，从而使得提示菜单随鼠标而显示。

//得到鼠标的位置

```
var x = event.x;
var y = event.y;
menu.style.top = y;
menu.style.left = x;
```

同时，大家应主要确定应该删除其中哪一行的方法：

① 首先记录要删除的行号是哪个：

```
function mousedown(tr) {
    .....
    delRow = tr;
}
```

② 再根据这里记录的行进行删除：

```
function deleteRow() {
    //得到表格
    var tab = document.getElementById("tab");
    //tr.sectionRowIndex //得到 tr 的索引编号
    tab.deleteRow(delRow.sectionRowIndex);
    var menu = document.getElementById("contextMenu");
    menu.style.display="none";
}
```

## 第 5 章

### Ajax 客户端技术

---

通过前面的学习，相信读者已经可以实现在页面中动态地添加一些元素，并使用表单和用户进行交互了。但是可能大家已经发现，所有的操作都是基于整个页面的，也就是说，所有的操作都要在刷新整个页面后才能够看到效果，那么，是否可以只刷新页面的一部分内容，就达到更新的目的呢？

## 5.1 Ajax 介绍

基于 XML 的异步 JavaScript, 简称 Ajax, 是当前 Web 创新 (称为 Web 2.0) 中的一个翘楚。感谢组成 Ajax 的各种技术, Web 应用的交互, 如 Flickr、Backpack 和 Google 在这方面已经有了质的飞跃。这个术语源自描述从基于网页的 Web 应用到基于数据的应用的转换。在基于数据的应用中, 用户需求的数据, 如联系人列表, 可以从独立于实际网页的服务器端取得并且可以被动态地写入网页中, 给缓慢的 Web 应用体验着色, 使之像桌面应用一样。

许多重要的技术和 Ajax 开发模式可以从现有的知识中获取。例如, 在一个发送请求到服务端的应用中, 必须包含请求顺序、优先级、超时响应、错误处理及回调, 其中许多元素已经在 Web 服务中包含了, 就像现在的 SOA。Ajax 开发人员拥有一个完整的系统架构知识。同时, 随着技术的成熟还会有许多地方需要改进, 特别是 UI 部分的易用性。

Ajax 开发与传统的 CS 开发有很大的不同。这些不同引入了新的编程问题, 最大的问题在于易用性。由于 Ajax 依赖浏览器的 JavaScript 和 XML, 浏览器的兼容性和支持的标准也变得和 JavaScript 的运行时性能一样重要了。这些问题中的大部分来源于浏览器、服务器和技术的组合, 因此必须理解如何才能最好地使用这些技术。

综合各种变化的技术和强耦合的客户服务端环境, Ajax 提出了一种新的开发方式。Ajax 开发人员必须理解传统的 MVC 架构, 这限制了应用层次之间的边界。同时, 开发人员还需要考虑 CS 环境的外部和使用 Ajax 技术来重定型 MVC 边界。最重要的是, Ajax 开发人员必须禁止以页面集合的方式来考虑 Web 应用, 而需要将其认为是单个页面。一旦 UI 设计与服务架构之间的范围被严格区分开来后, 开发人员就需要更新和变化的技术集合了。

### 5.1.1 Ajax 技术的由来

该技术在 1998 年前后得到了应用。允许客户端脚本发送 HTTP 请求 (XMLHTTP) 的第一个组件由 Outlook Web Access 小组写成。该组件原属于微软 Exchange Server, 并且迅速地成为了 Internet Explorer 4.0 的一部分。部分观察家认为, Outlook Web Access 是第一个应用了 Ajax 技术的成功的商业应用程序, 并成为包括 Oddpost 的网络邮件产品在内的许多产品的领头羊。2005 年初, 许多事件使得 Ajax 被大众所接受。Google 在它著名的交互应用程序中使用了异步通信, 如 Google 讨论组、Google 地图、Google 搜索建议、Gmail 等。对 Mozilla/Gecko 的支持使得该技术走向成熟, 变得更为易用。

随着网络时代的来临, 传统的应用程序开发从 Client/Server 结构开始向 Browser/Server 结构过渡。B/S 结构由于使用了通用的瘦客户端——浏览器, 可以在任何地方进行操作而不需要安装专门的软件, 也不再需要为每次系统应用的升级而修改客户端。然而, B/S 结构也有着一些固有的局限性, 例如, B/S 结构的浏览器无法向 C/S 结构那样发挥客户端 PC 的处理能力, 并不是在客户端处

理多个步骤之后再最终结果传递到服务器端，而是每次要进行一个动作，就要把请求发送到服务器端，再由服务器端返回结果，之后才能进行下一个请求，这样，其响应速度比 C/S 结构要慢很多。从用户体验的角度来讲，B/S 结构不如 C/S 结构。

这里从一个现实的例子开始讨论 Ajax 的运行机制。假如有一个互联网公司的开发人员，公司要开发一款邮件系统以便和其他公司进行竞争。先不去观察竞争对手正在做些什么，至少应该知道，这个邮件系统中最重要的部分是邮件的使用，而最先开始的部分应该是用户的产生，即用户注册功能。下面是传统的用户注册流程：用户在表单中输入了自己的用户名、密码以及一些个人信息，单击提交按钮，经客户端 JavaScript 验证没有问题，数据被传送到服务器端。服务器端程序检查后发现这个用户名是可以使用的，就会在服务器端的数据库中产生一条用户信息记录，之后用户就可以使用邮件服务了；如果服务器端程序发现这个用户提交的用户名已经被占用，则返回出错信息，让用户重新修改申请的用户名并要求其重复提交过程。这样，在简单的注册过程中，用户就有可能多次重复提交注册申请才能最终找到可用的用户名，而越是人气旺的网站系统，这种用户名注册冲突的现象发生的概率就越大。试想一下，如果某位用户想使用网站的服务，重复注册提交了好多次，每一次返回的结果都是“该用户名已经被占用了，请重新选择用户名”，当然会感到沮丧，甚至放弃使用网站服务。因此，如何提高用户体验就成了互联网开发人员要思考的重要课题。

Ajax 前景非常乐观，可以提高系统性能，优化用户界面。Ajax 现有直接框架 AjaxPro，可以引入 AjaxPro.2.dll 文件，可以直接在前台页面 JS 调用后台页面的方法。但此框架与表单验证有冲突。另外，微软也引入了 Ajax 组建，需要添加 AjaxControlToolkit.dll 文件，在控件列表中会出现相关控件。

### 5.1.2 Ajax 与 JavaScript 的关系

Ajax 的概念中最重要的也是最被忽视的是，它也是一种 JavaScript 编程语言。JavaScript 是一种黏合剂，使 Ajax 应用的各部分集成在一起。在大部分时间中，JavaScript 常被服务端开发人员认为是一种企业级应用，不需要使用的东西应该尽力避免。这种观点来自以前编写 JavaScript 代码的经历：繁杂而又易出错的语言。类似的，它也被认为将应用逻辑任意地散布在服务端和客户端中，这使得问题很难被发现而且代码很难重用。在 Ajax 中，JavaScript 主要被用来传递用户界面上的数据到服务端并返回结果。XMLHttpRequest 对象用来响应通过 HTTP 传递的数据，一旦数据返回到客户端就可以立刻使用 DOM 将数据放到网页上。

## 5.2 Ajax 程序范例

### 5.2.1 使用 Ajax 完成验证

关于可用性有一句金玉良言，即防患于未然，根本杜绝错误的发生。但是如果真的出现了错误，

你就要第一时间通知用户。在 Ajax 之前，基于 Web 的应用必须提交整个页面才能验证数据，或者要依赖复杂的 JavaScript 来检查表单。尽管有些检查确实很简单，可以使用 JavaScript 编写，但另外一些检查则不然，完全靠 JavaScript 编写是办不到的。当然，在客户端编写的每一个验证例程都必须在服务器上以某种方式重写，因为用户有可能禁用 JavaScript。

利用 Ajax，你不用再受这个限制，不再只是编写简单的客户端验证和重复的逻辑。现在，如果你想为用户提供更能体现交互性的体验，可以简单地调用为服务器编写的验证例程。在大多数情况下，这个逻辑编写起来更简单，测试也更容易，而且完全可以借助于现有的框架。

有人问，在应用中应该从哪里开始使用 Ajax，一般会建议从验证开始。你很可能要去掉一些 JavaScript，而且可以很容易地加入一些现有的服务器端逻辑。本节将介绍一个例子，这是最常见的验证之一——日期验证。

这个例子的 HTML 很简单。其中，有一个标准的输入框，以及相应的 onchange() 事件（当然，可以使用你认为合适的任何事件）会触发验证方法。可以看到，要调用标准的 createXMLHttpRequest() 方法，然后把输入值发送到 ValidationServlet servlet。callback() 函数从服务器得到结果，然后委托给 setMessage() 方法，这个方法会检查值，以确定用什么颜色显示消息。

### 例 5-1

```
<html>
<head>
<title>Using Ajax for validation</title>
<script type="text/javascript">
var xmlhttp;
function createXMLHttpRequest() {
//判断浏览器是否支持 ActiveX 控件
if (window.ActiveXObject) {
//支持则通过 ActiveXObject 的一个新实例来创建 XMLHttpRequest 对象
xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
}
//否则把 XMLHttpRequest 实现为一个本地 JavaScript 对象
else if (window.XMLHttpRequest) {
xmlhttp = new XMLHttpRequest();
}
}
function validate() {
createXMLHttpRequest();
var date = document.getElementById("birthDate");
var url = "servlet/ValidationServlet?birthDate=" + escape(date.value);
//设置请求目标 URL、方法和其他参数
```

```

xmlHttp.open("GET", url, true);
xmlHttp.onreadystatechange = callback;
xmlHttp.send(null);
}
function callback() {
if (xmlHttp.readyState == 4) {
if (xmlHttp.status == 200) {
var mes =
xmlHttp.responseXML.getElementsByTagName("message")[0].firstChild.data;
var val =
xmlHttp.responseXML.getElementsByTagName("passed")[0].firstChild.data;
setMessage(mes, val);
}
}
}
function setMessage(message, isValid) {
var messageArea = document.getElementById("dateMessage");
var fontColor = "red";
if (isValid == "true") {
fontColor = "green";
}
messageArea.innerHTML = "<font color=" + fontColor + ">"
+ message + " </font>";
}
</script>
</head>
<body>
<h1>Ajax 实例</h1>
生日: <input type="text" size="10" id="birthDate" onchange="validate();" />
<div id="dateMessage"></div>
</body>
</html>

```

得到的用户页面如图 5-1 所示。

服务器端代码也很简单。为简单起见，这里把验证代码放在 **Servlet** 中，而在生产环境中很可能把验证代码委托给验证服务。

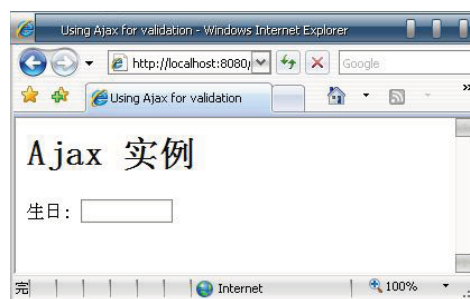


图 5-1 Ajax 页面

例 5-2

```
import java.io.*;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import javax.servlet.*;
import javax.servlet.http.*;

public class ValidationServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        //获取输入的生日
        boolean passed = validateDate(request.getParameter("birthDate"));
        response.setContentType("text/xml");
        response.setHeader("Cache-Control", "no-cache");
        String message = " You have entered an invalid date.";
        //判断生日输入是否正确
        if (passed) {
            message = " You have entered a valid date..";
        }
        //输出结果
        out.println("<response>");
        out.println("<passed>" + Boolean.toString(passed) + "</passed>");
        out.println("<message>" + message + "</message>");
        out.println("</response>");
        out.close();
    }
    private boolean validateDate(String date) {
        boolean isValid = true;
        //判断日期格式
        if(date != null) {
            //格式化日期
            SimpleDateFormat formatter= new SimpleDateFormat("MM/dd/yyyy");
            try {
                formatter.parse(date);
            } catch (ParseException pe) {
                System.out.println(pe.toString());
                isValid = false;
            }
        } else {
```



```
isValid = false;
}
//返回判断结果
return isValid;
}
}
```

当输入一个生日时，页面会将输入的内容发送到服务器端进行判断，并将结果显示在页面上，如图 5-2 所示。

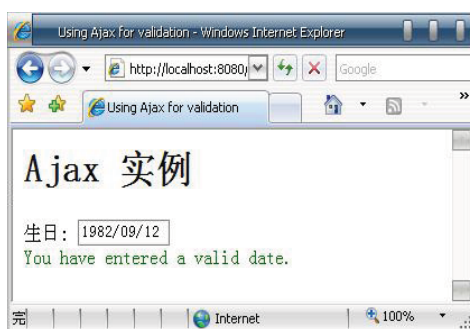


图 5-2 正确的生日验证

在这里大家发现，从 `servlet` 中返回的信息并没有刷新整个页面，而是在原页面上显示了出来。是不是比前面的方法好了很多？

## 5.2.2 使用 Ajax 完成交互

Ajax 的工作原理相当于在用户和服务器之间加了个中间层，使用户操作与服务器响应异步化。并不是所有的用户请求都提交给服务器，像一些数据验证和数据处理等，都交给 Ajax 引擎自己来做，只有确定需要从服务器读取新数据时，再由 Ajax 引擎代为向服务器提交请求。

在旧的交互方式中，由用户触发一个 HTTP 请求到服务器，服务器对其进行处理后再返回一个新的 HTML 页到客户端，每当服务器处理客户端提交的请求时，客户都只能空闲等待，并且哪怕只是一次很小的交互、只需从服务器端得到很简单的一个数据，都要返回一个完整的 HTML 页，而用户每次都要浪费时间和带宽去重新读取整个页面。

而使用 Ajax 后，用户感觉到几乎所有的操作都会很快响应，并且没有页面重载（白屏）的等待。

Ajax 的一个最大的特点是无须刷新页面，便可向服务器传输或读写数据（又称无刷新更新页面），这一特点主要得益于 XMLHTTP 组件 XMLHttpRequest 对象。这样就可以异步向服务器端发送请求，并处理响应结果，而无须刷新页面也不用每次将数据处理的工作提交给服务器来做，这样既减轻了服务器的负担又加快了响应速度、缩短了用户等候时间。

最早应用 XMLHTTP 的是微软，IE（IE5 以上）通过允许开发人员在 Web 页面内部使用

XMLHTTP ActiveX 组件扩展自身的功能,开发人员可以不用从当前的 Web 页面导航,而直接传输数据到服务器上或者从服务器取数据。这个功能是很重要的,因为它帮助减少了无状态连接的痛苦,它还可以排除下载冗余 HTML 的需要,从而提高进程的速度。Mozilla( Mozilla1.0 以上及 Netscape7 以上)做出的回应是创建它自己的继承 XML 代理类:XMLHttpRequest 类。Konqueror(和 Safari v1.2, 同样也是基于 KHTML 的浏览器)也支持 XMLHttpRequest 对象,而 Opera 也将在其 v7.6x+以后的版本中支持 XMLHttpRequest 对象。对于大多数情况,XMLHttpRequest 对象和 XMLHTTP 组件很相似,方法和属性也类似,只是有一小部分属性不支持。表 5-1 列出了 XMLHttpRequest 的对象方法,表 5-2 列出了 XMLHttpRequest 的对象属性。

XMLHttpRequest 对象在 JS 中的应用:

```
var xmlhttp = new XMLHttpRequest();
```

微软的 XMLHTTP 组件在 JS 中的应用如下例所示。

例 5-3

```
var xmlhttp = new ActiveXObject(Microsoft.XMLHTTP);
var xmlhttp = new ActiveXObject(Msxml2.XMLHTTP);
XMLHttpRequest 对象方法
if (typeof XMLHttpRequest == 'undefined') {
    XMLHttpRequest = function () {
        var msxmls = ['MSXML3', 'MSXML2', 'Microsoft']
        for (var i=0; i < msxmls.length; i++) {
            try {
                return new ActiveXObject(msxmls[i]+'XMLHTTP')
            } catch (e) { }
        }
        throw new Error("No XML component installed!")
    }
}
function createXMLHttpRequest() {
    try {
        //以 Mozilla 方式创建
        if (window.XMLHttpRequest) {
            return new XMLHttpRequest();
        }
        //以 IE 方式创建
        if (window.ActiveXObject) {
            return new ActiveXObject(getXMLPrefix() + ".XmlHttp");
        }
    }
}
```

```
catch (ex) {}
return false;
};
```

表 5-1 XMLHttpRequest 对象方法

| 方法                               | 描述                   |
|----------------------------------|----------------------|
| abort()                          | 停止当前请求               |
| getAllResponseHeaders()          | 作为字符串返回完整的 headers   |
| getResponseHeader("headerLabel") | 作为字符串返回单个的 header 标签 |

设置提交请求的目标 URL、方法和其他参数：

```
open("method", "URL" [, asyncFlag [, "userName" [, "password"]]])
```

发送请求：

```
send(content)
```

设置 header 并和请求一起发送：

```
setRequestHeader("label", "value")
```

表 5-2 XMLHttpRequest 对象属性

| 属性                 | 描述   |
|--------------------|--|
| onreadystatechange | 状态改变的事件触发器   |
| readyState         | 对象状态 (integer):<br>0 = 未初始化<br>1 = 读取中<br>2 = 已读取<br>3 = 交互中<br>4 = 完成 |
| responseText       | 服务器进程返回数据的文本版本   |
| responseXML        | 服务器进程返回数据的兼容 DOM 的 XML 文档对象  |
| status             | 服务器返回的状态码，如：404 = "文件未找到"、200 = "成功"                                   |
| statusText         | 服务器返回的状态文本信息   |

## 5.3 一个注册的案例

上一节中简单介绍了交互的对象，下面通过一个具体的例子来看一下。

实现如图 5-3 所示界面的注册功能，需要采用 Ajax 方式来完成用户注册。



图 5-3 Ajax 方式完成用户注册

在 HTML 中嵌入 Ajax 代码，页面表现形式与传统一致，为用户填写注册信息，当用户填写了注册用户名，准备开始填写注册密码时，页面状态开始变化。JavaScript 函数在第一个文本框失去焦点时被触发，生成一个检测用户名的 http 请求向服务器端发送，同时提醒用户正在检测用户名，如图 5-4 所示。

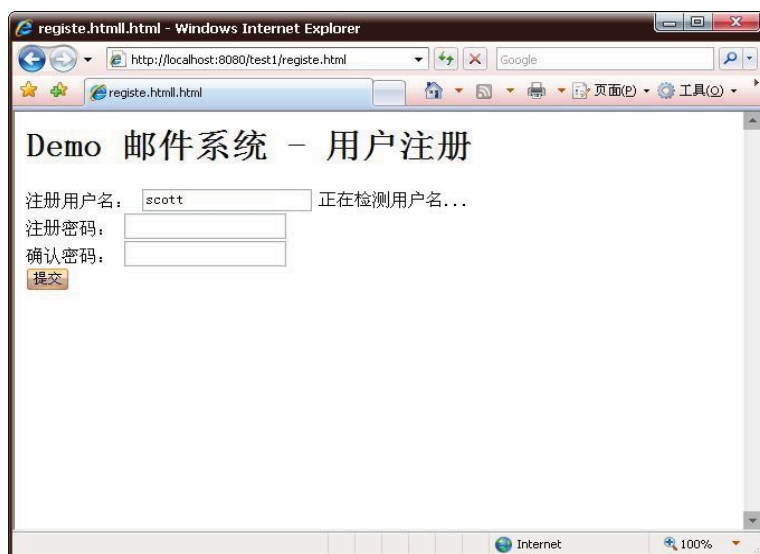


图 5-4 利用 Ajax 无刷新提交 http 请求

正如图 5-4 所显示的，用户的信息注册行为与用户名检测行为在同时进行。这时的用户并没有为检测用户名多做任何的工作。在一小段时间后，检测结果将会被返回给用户，如图 5-5 所示。

整个注册过程在 Ajax 技术的支持下，利用 JavaScript 的 HTML 操纵能力，使得用户不需要进行额外的操作。用户体验的效果逼近 C/S 结构。而整个案例的改进，仅仅是改良了客户端的 register.html 文件，添加了 JavaScript 代码和 HTML 元素。

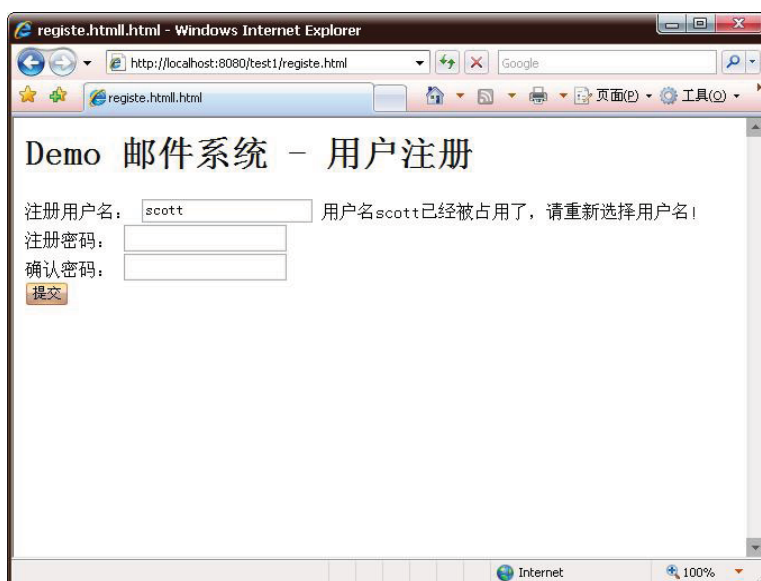


图 5-5 Ajax 返回用户名检测结果

首先还是编写一个 HTML 表单来接收用户输入。

源代码如下 (register.html) :

```
<H1>Demo 邮件系统 - 用户注册</H1>
<form name="regForm" id="regForm" action="DoRegister.jsp" method="post">
  注册用户名:
    <input type="text" name="userName" id="userName" onblur="checkUser();" />
    <label id="checkResult"></label>
  <br/>
  注册密码:
    <input type="password" name="password" id="password" /><br/>
  确认密码:
    <input type="password" name="password2" id="password2" /><br/>
    <input type="submit" value="提交" />
</form>
```

在 HTML 文件中对 id 为 `userName` 的文本框元素添加了 `onblur` 事件。这样, 当这个文本框失去焦点, 即用户填完了注册用户名, 准备继续填写注册密码时, 事件被触发。JavaScript 方法 `checkUser()` 被调用。下面是 `checkUser()` 的定义:

```
var req;
function checkUser() {
  var checkResult = document.getElementById("checkResult");
  checkResult.innerHTML="正在检测用户名...";
  var userName=document.getElementById("userName").value;
```

```

//得到浏览器中可以发送 http 请求的对象, 本例只对 IE5.5+有效
req=new ActiveXObject("Microsoft.XMLHttp");

req.open("get","http://localhost:8080/test1/CheckUserName.jsp?
checkUserName="+userName);
//javascript 特殊语法, 表明 req 的状态改变将调用 handleRequest 方法
req.onreadystatechange=handleRequest;
req.send(null);
}

```

为了便于理解, JavaScript 方法 `checkUser()` 被写得很简单。其中最关键的步骤就是方法将会生成一个 `XMLHttpRequest` 对象在 IE 浏览器中, 这个对象以 `ActiveX` 控件的形式出现。有了 `XMLHttpRequest` 对象之后, JavaScript 方法可以利用它直接发送异步请求到服务器端而不需要刷新当前浏览器页面。当服务器端返回处理结果时, `XMLHttpRequest` 对象的状态将会发生改变, 这样会触发相应的 JavaScript 方法被调用。在案例中, `handleRequest()` 方法就是用来处理服务器端返回的结果。

编写 `handleRequest` 方法:

```

function handleRequest() {
    var checkResult = document.getElementById("checkResult");
    //判断就绪状态
    if (req.readyState==4) {
        if (req.status==200) {
            checkResult.innerHTML=req.responseText;
        }
        else {
            alter("An error occurred:"+req.statusText);
        }
    }
}

```

`HandlerRequest()` 方法的逻辑也比较简单, 当服务器端有正常返回值时, 就在 `id` 为 `checkResult` 的 `label` 元素中添加一段 HTML, 其值就是由服务器端返回的数据。在上述的两个方法中, 都有陌生的 Ajax 对象和属性。在接下来的一节中, 会对 Ajax 技术中的对象和方法做完整的解释。

## 5.4 使用 XMLHttpRequest 对象与服务器端通信

在上面的例子中究竟用到了哪些技术呢?

### 5.4.1 XMLHttpRequest 对象

XMLHttpRequest 对象是整个 Ajax 开发的基础,它具有从客户端到服务器端异步发送 http 请求的能力。XMLHttpRequest 对象由 JavaScript 创建并使用。最早它出现在微软的 IE5.0 浏览器中,是以 ActiveX 控件的形式出现的。之后随着 XMLHttpRequest 对象具有发送异步请求的特殊作用,其他浏览器开始支持这个功能。这也是 Ajax 技术能够迅速走红的一个基础。如今大部分浏览器都可以支持 XMLHttpRequest 对象。下面是浏览器列表:

Microsoft IE5.0 以上

Apple Safari 1.2 以上

Firefox 1.0 以上

Netscape 7.1 以上

Opera7.6 以上

由于 XMLHttpRequest 对象不是一个通用的国际标准,不同的浏览器对其实现的方式也不相同。要想实现跨浏览器运行的 Ajax 程序,需要考虑所有的使用场合,这也使得生成 XMLHttpRequest 的代码变得复杂起来。

IE5.5 以上版本的 XMLHttpRequest 对象的创建方式。

```
//得到浏览器中可以发送 http 请求的对象,本例只对 IE5.5+有效
var req = new ActiveXObject("Microsoft.XMLHttp");
```

对于 IE5.0 版本,得到 XMLHttpRequest 对象的方式为:

```
//得到浏览器中可以发送 http 请求的对象,本例只对 IE5.0 有效
req=new ActiveXObject("Msxml2.XMLHttp");
```

为了支持所有的主流浏览器,比较通行的方法是使用 JavaScript 的 try-catch 程序块来尝试以不同的途径创建对象。代码如下:

```
function createXHR() {
    var xhr;
    try{
        xhr= new ActiveXObject("Msxml2.XMLHTTP");
    }catch(e) {
        try{
            xhr = new ActiveXObject("Microsoft.XMLHTTP");
        }catch (E) {
            xhr = false;
        }
    }
    if(!xhr&&typeofXMLHttpRequest!='undefined'){
```



```

        xhr = new XMLHttpRequest();
    }
    return xhr;
}

```

这样，在调用的时候就可以使用下面代码来得到跨浏览器的 `XMLHttpRequest` 对象了。

```

//得到跨浏览器的 XMLHttpRequest 对象
var req = createXHR();

```

有了 `XMLHttpRequest` 对象，就可以利用对象的方法向服务器端发送异步消息了。

### 5.4.2 使用 open 方法创建一个请求

使用 `XMLHttpRequest` 对象的第一步就是创建一个 `http` 请求，使用 `open` 方法。其原型为：

```
open(method,url,asynchronous,user,password);
```

其中的参数含义如下：

- `method`: `http` 请求方式，支持“`post`”和“`get`”。
- `url`: 服务器端程序的 `URL`。
- `asynchronous`: 定义 `request` 是否异步。如果 `asynchronous = true`，`send()`函数在发送 `request` 后马上返回，即异步请求；如果 `asynchronous = false`，`send()`函数必须在收到回执后才返回，即同步请求。默认值是 `true`。
- `username`: 用户名，有些 `url` 要求 `request` 带有用户名，一般不会用到。
- `password`: 密码，一般不会用到。

创建一个异步请求，代码如下：

```
req.open("get","http://localhost:8080/test1/CheckUserName.jsp?checkUserName="+userName);
```

### 5.4.3 使用 send 方法发送一个请求

在创建一个请求之后，可以使用 `send` 方法向服务器发送这个请求。其原型为：

```
send(body);
```

该方法只有一个参数，它表示要向服务器端发送的数据，其格式为字符串。使用 `get` 请求，所有提交字符串都显式地写在了 `url` 上，所以发送的数据为 `null`。

```
req.open("get","http://localhost:8080/test1/CheckUserName.jsp?checkUserName="+userName);
req.send(null);
```

如果在 `open` 的方法中指明是 `post` 请求，在 `send` 提交之前需要设置 `http` 头。例如，在案例中，如果可以，修改 `register.html` 源文件为：



```
req.open("post","http://localhost:8080/test1/CheckUserName.jsp");
req.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
req.send("checkUserName="+userName);
```

其他情况下，post 和 get 操作没有任何区别。

#### 5.4.4 使用 onreadystatechange 事件捕获请求状态变化

onreadystatechange 是 XMLHttpRequest 对象的参数，是用来定义此对象状态改变事件发生时的监听器，它的值对应一个 JavaScript 函数。在案例中对应的是定义的 handleRequest 方法。

```
req.open("get","http://localhost:8080/test1/CheckUserName.jsp?checkUserName="+
userName);
    //javascript 特殊语法，表明 req 的状态改变将调用 handleRequest 方法
req.onreadystatechange=handleRequest;
req.send(null);
```

注意 onreadystatechange 的值为 handleRequest，要与定义的 handleRequest 方法完全一致。下面是案例中处理服务器返回结果的方法定义：

```
function handleRequest(){
var checkResult = document.getElementById("checkResult");
//状态改变逻辑处理
}
```

#### 5.4.5 使用 readyState 属性判断请求状态变化

handleRequest 方法定义了当 XMLHttpRequest 对象的 readyState 属性发生改变时，根据属性变化采取相应的响应操作。readyState 的值表示了当前的请求状态，可能的状态有 5 个，见表 5-3。

表 5-3 readyState 的属性值

| readyState 的值 | 含义                       |
|---------------|--------------------------|
| 0             | open 方法还未开始调用            |
| 1             | open 方法已经调用，但未调用 send 方法 |
| 2             | send 方法已调用               |
| 3             | 请求已经发送成功，正在接收数据          |
| 4             | 应答已解析，数据接收成功             |

在整个请求过程中，onreadystatechange 事件在每一次 readyState 的值改变时都会被触发。通常在事件中判断 readyState 的值是在请求完毕时才做处理：

```
function handleRequest(){
    var checkResult = document.getElementById("checkResult");
    if(req.readyState==4){
```

```

        //请求完毕的处理代码
    }}

```

### 5.4.6 使用 status 属性判断请求的结果

**status** 记录了服务器端返回的 http 请求响应代码, 这个代码和以前所学习的代码是一致的。200 表示请求成功, 404 表示请求资源未找到, 500 表示内部服务器错误等。

```

if (req.readyState==4) {
    if (req.status==200) {
        //请求成功
        checkResult.innerHTML=req.responseText;
    }
    else{
        //网络连接失败等错误
        alter("An error occurred:"+req.statusText);
    }
}

```

上面的代码表明, 当 XMLHttpRequest 对象从服务器端获得正确的应答后, 将会调用的处理逻辑。

### 5.4.7 使用 responseText 获得返回的文本

当服务器成功地处理了请求并返回后, 可利用 XMLHttpRequest 对象的 **responseText** 属性来获得返回的结果。在案例中, 返回的结果被赋值到 **label** 上, 以显示用户名是否被注册。代码如下:

```
checkResult.innerHTML=req.responseText;
```

**responseText** 属性将会获取到从服务器端返回的 HTML 页面的信息。再利用 JavaScript 来完成对页面元素的属性赋值, 实现了页面无刷新地获取数据。

## 5.5 利用 Ajax 实现局部刷新

### 5.5.1 网页无闪自动局部刷新

在网页制作的过程中, 经常会遇到及时刷新数据的问题, 如果使用 `<meta http-equiv=refresh content="300">` 的方法, 会造成整个屏幕不断闪烁刷新的效果, 这会降低用户的操作满意度。所以需要一种可以实现无闪自动刷新数据的方法来解决以上问题。

**实例解决问题:**

希望实现用户在进入系统以后 (整个 session 的时效之内), 如果收到新邮件则发出声音提示。

实现思路:

### 1. 首页部分:

```
<body onload="init('<%=ses_userBean.getUsername()%>');"> // load时调用 init(user);
```

2. js 部分: 用 xmlhttp 实现页面局部刷新, 调用 check\_mail.jsp 对后台数据库进行检索判断并返回结果。

```
var xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
var checkresult=null;
var username =null;
function init(user){
    username=user;
    window.setInterval('Checkmail()',5000); //每隔 5 秒自动调用 Checkmail()
}
function Checkmail()
{
    xmlhttp.open("POST", "check_mail.jsp?uName="+username, false);
    xmlhttp.onreadystatechange = updatePage;
    xmlhttp.send();
}
function updatePage() {
    if (xmlhttp.readyState < 4) {
        test1.innerHTML="loading...";
    }
    if (xmlhttp.readyState == 4) {
        var response = xmlhttp.responseText;
        if(response==1){ //判断为假
            test1.innerHTML="&nbsp;";
            checkresult=1;
        }
        else{//判断为真
            test1.innerHTML="<img alt='新邮件' src='img/tp024.gif'><EMBED src='music/nudge.wma'
            hidden=true autostart=true loop=false>";
            checkresult=0;
        }
    }
}
```

3. check\_mail.jsp 用于处理客户端发送上来的请求, 并返回对数据库查询得到的结果。如果有结果返回 0, 没有返回 1。

```
<%@ page contentType="text/html; charset=GBK" %>
<%@ page errorPage="error/login_error.jsp"%>
```

```

<%@ page import="myweb.*" %>
<%@ page import="java.sql.*" %>
<%
String user=request.getParameter("uName");
Connection conn=null;
try{
conn=DBConnection.getConnection();
PreparedStatement pStat=conn.prepareStatement("select * from message
where r_name= '"+user+"' and status=0");
//查询SQL语句
ResultSet rs=pStat.executeQuery();
if(rs.next()){//有记录
response.getWriter().print(0);
}else{
response.getWriter().print(1);
}
}finally{
if(conn!=null) conn.close();
}
}%>

```

4. 首页结果显示。

将<span id="test1"></span>插入指定位置。

5. 数据库部分，以及数据库连接部分的内容请大家自己给出。

得到效果如图 5-6 所示。

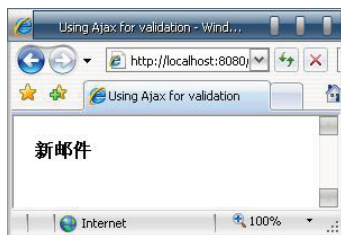


图 5-6 图片自动刷新

## 5.5.2 表单局部刷新

提交 form 表单之后，不会刷新页面，而是局部刷新。

如果使用 get 请求，注意中文乱码问题，jQuery 会先使用 iso8859-1 解码，然后发给服务器，如果使用 post 请求，则直接将中文内容提交给服务器解析。

使用 jquery-1.4.2.js 和 jquery.form.js 插件。

```
//如果创建新的js文件中,需要加上
document.write("<script type='text/javascript' src='jquery-1.4.2.js'></script>")
document.write("<script type='text/javascript' src='jquery.form.js'></script>")
// 动态加载页面
// id 显示页面的容器组件 ID2
// url 欲加载页面网址
// gop get 请求还是 post 请求, 默认 get
function loadPage(id, url, gop) {
$("#" + id).addClass("loader");
$("#" + id).append("Loading.....");
var type = (gop == "post" ? "post" : "get");//判断 gop 的请求类型
$.ajax( {
type : type;
url : url,
cache : false,
//分别定义出错时, 以及成功时要执行的方法
error : function() {
alert('加载页面' + url + '时出错! ')
},
success : function(content) {
$("#" + id).empty().append(content);
$("#" + id).removeClass("loader");
}
});
}
// 局部提交表单
function formSubmit(formId, divId, url) {
$('#' + formId).submit(function() {
$(this).ajaxSubmit( {
target : '#' + divId,
url : url,
error : function() {
alert('加载页面' + url + '时出错! ')
}
});
return false;
});
}
```

本例的思想已经给出, 剩下的部分请大家自己调试, 试试看能否完成。

## 5.6 实现注册页面

学习了 Ajax 的基本操作之后,大家应该已经迫不及待地想要实验一下这些新的功能了,接下来一起来做一个注册页面。

HTML 页面的完整代码如下:

```
<%@page language="java" contentType="text/html; charset=GBK"%>
<script language="javascript" type="text/javascript">
<!--

var http = getHTTPObject();
function handleHttpResponse() {
    //判断 xmlHttpRequest 对象的状态
    if(http.readyState == 4){
        if(http.status == 200){
            var xmlDocument = http.responseXML;
            //如果返回值不为空
            if(http.responseText!=""){
                //显示返回内容
                document.getElementById("showStr").style.display = "";
                document.getElementById("userName").style.background= "#FF0000";
                document.getElementById("showStr").innerText = http.responseText;
            }else{
                //如果返回值为空,不显示
                document.getElementById("userName").style.background= "#FFFFFF";
                document.getElementById("showStr").style.display = "none";
            }
        }
    }
    else{
        alert("你所请求的页面发生异常,可能会影响你浏览该页的信息!");
        alert(http.status);
    }
}

function handleHttpResponse1(){
    if(http.readyState == 4){
        if(http.status == 200){
            var xmlDocument = http.responseXML;
            if(http.responseText!=""){
```

```

        document.getElementById("comNmStr").style.display = "";
        document.getElementById("comNm").style.background= "#FF0000";
        document.getElementById("comNmStr").innerText = http.responseText;
    }else{
        document.getElementById("comNm").style.background= "#FFFFFF";
        document.getElementById("comNmStr").style.display = "none";
    }
}
else{
    alert("你所请求的页面发生异常, 可能会影响你浏览该页的信息!");
    alert(http.status);
}
}
}

//检测用户名称
function chkUser(){
    var url = "/chkUserAndCom";
    var name = document.getElementById("userName").value;
    url += ("&userName="+name+"&oprte=chkUser");
    http.open("GET",url,true);
    http.onreadystatechange = handleHttpResponse;
    http.send(null);
    return ;
}

//检测企业名称
function chkComNm(){
    var url = "/chkUserAndCom";
    var name = document.getElementById("comNm").value;
    url += ("&comName="+name+"&oprte=chkCom");
    http.open("GET",url,true);
    http.onreadystatechange = handleHttpResponsel;
    http.send(null);
    return ;
}

//该函数可以创建需要的 XMLHttpRequest 对象
function getHTTPObject(){
    var xmlhttp = false;
    if(window.XMLHttpRequest){

```

```

    xmlhttp = new XMLHttpRequest();
    if(xmlhttp.overrideMimeType){
        xmlhttp.overrideMimeType('text/xml');
    }
}
else{
    try{
        xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
    }catch(e){
        try{
            xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
        }catch(E){
            xmlhttp = false;
        }
    }
}
return xmlhttp;
}

//检测表单输入是否合法
function chkpassword()
{

//获取表单
var m=document.form1;
//判断密码输入是否合法
if(len(m.password.value) > 20 || len(m.password.value) < 5
|| !isStr(m.password.value))
{
    document.getElementById("passwordStr").style.display = "";
    document.getElementById("password").style.background= "#FF0000";
    document.getElementById("passwordStr").innerText = "对不起,密码必须为英文字母、数字或
下划线,长度为 5~20!";
}
else
{
    document.getElementById("password").style.background= "#FFFFFF";
    document.getElementById("passwordStr").style.display = "none";
}
}

```



```
//判断第二次输入是否合法、一致
function chkconfirmPassword()
{
    var m=document.form1;
    if (m.password.value != m.confirmPassword.value)
    {
        document.getElementById("confirmPasswordStr").style.display = "";
        document.getElementById("confirmPassword").style.background= "#FF0000";
        document.getElementById("confirmPasswordStr").innerText = "对不起,密码与重复密码不一致!";
    }
    else
    {
        document.getElementById("confirmPassword").style.background= "#FFFFFF";
        document.getElementById("confirmPasswordStr").style.display = "none";
    }
}
//判断所有文本框中的输入
function checkfield()
{
    var m=document.form1;
    if(m.userName.value.length==0)
    {
        alert("对不起,用户名必须为英文字母、数字或下画线,长度为 5~20。");
        m.userName.focus();
        return false;
    }
    if(m.password.value.length==0)
    {
        alert("对不起,密码必须为英文字母、数字或下画线,长度为 5~20。");
        m.password.focus();
        return false;
    }
    if (m.password.value != m.confirmPassword.value)
    {
        alert("对不起,密码与重复密码不一致!");
        m.confirmPassword.focus();
        return false;
    }
    if(m.comNm.value.length==0)
    {

```

```

alert("对不起,企业名称不能为空!!");
m.comNm.focus();
return false;
}
m.submit();
}
//-->
</script>
<body topmargin="0">
<form name="form1" method="post" action="/Control?act=Register">
<table width="100%">
<tr><td align="center">&nbsp;<H2>Ajax 演示程序</H1></td></tr>
<tr><td align="center">&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&----- 企业注册 By Alpha</td></tr>
</table>
<HR>
<table width="400" border="0" cellpadding="1" cellspacing="1" align="center" >
<tr>
<td><font color="red">*</font></td>
<td>用户账号:</td>
<td>

<div id="showStr" style="background-color:#FF9900;display:none"></div>
</td>
</tr>
<tr>
<td><font color="red">*</font></td>
<td>企业名称:</td>
<td>

<div id="comNmStr" style="background-color:#FF9900;display:none"></div>
</td>
</tr>
<tr>
<td><font color="red">*</font></td>
<td>用户密码:</td>
<td><input type="password" name="password" maxlength="20" style="background:
#FFFFFF" onBlur="chkpassword()"/>
<div id="passwordStr" style="background-color:#FF9900;display:none"></div>

```

```

    </td>
</tr>
<tr>
    <td><font color="red">*</font></td>
    <td>确认密码:</td>
    <td><input type="password" name="confirmPassword" maxlength="20" style=
"background:#FFFFFF" onBlur="chkconfirmPassword()" />
        <div id="confirmPasswordStr"
style="background-color:#FF9900;display:none"></div>
    </td>
</tr>
</table>
<div align="center">
    <input type="button" name="ok" value=" 确 定 " onclick="checkfield()" />
    &nbsp;&nbsp;<input type="reset" name="reset" value=" 取 消 " />
</div>
</form>
</body>
</html>

```

在页面部分中，通过 **JavaScript** 进行客户端的表单验证，主要验证表单的格式、输入是否正确。使用这种方式可以避免在提交之后才返回一个简单的输入错误，在网速较慢的情况下更能提高效率。

#### servlet 部分:

```

package com.event;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.beans.EBaseInfo;
public class CheckUserAndComNm {
    private String msgStr = "";
    protected void doGet(HttpServletRequest request,HttpServletResponse response)
    throws ServletException
    {
        EComBaseInfo info=new EComBaseInfo();
        //获取请求中的参数
        String oprate=request.getParameter("oprate").trim();
        String userName=request.getParameter("userName");
        String passWord=request.getParameter("password");
        String comName=request.getParameter("comName");

        try

```

```
{
//判断是否是检查用户名
    if(oprate.equals("chkUser"))
    {
        response.setContentType("text/html;charset=GB2312");
        if(userName.length() < 5 || userName.length() > 20)
        {
            msgStr = "对不起,用户名必须为字母、数字或下画线,长度为 5-20 个字符!";
        }
        else
        {
            boolean bTmp=info.findUser(userName);
//查询数据库中有无该用户名
            if(bTmp)
                msgStr ="对不起,此用户名已经存在,请更换用户名注册!";
            else
                msgStr ="";
        }
        response.getWriter().write(msgStr);
    }
//检查企业名
    else if(oprate.equals("chkCom"))
    {
        response.setContentType("text/html;charset=GB2312");
        if(comName.length() < 6 || comName.length() > 100)
        {
            msgStr = "对不起,公司名称长度为 6-100 个字符 (不包括字符内的空格)!";
        }
        else
        {
            boolean bTmp=info.findCom(comName);
//查找数据库中有无该企业名
            if(bTmp)
                msgStr ="对不起,此企业名称已经存在,请更换企业名称注册!";
            else
                msgStr ="";
        }
        response.getWriter().write(msgStr);
    }
}
catch(Exception ex)
```

```

{
}
finally
{
    request.setAttribute("url",url);
}
}
protected void doPost(HttpServletRequest request,HttpServletResponse response)
throws ServletException
{
    doGet(request,response);
}
}

```

在 **servlet** 部分中,进行数据库方面的验证,主要用于查看在数据库中有没有存在该用户或企业。实现效果如图 5-7 所示。

如果在输入的过程中有错误产生,那么显示一个层,在层中给出提示信息,如图 5-8 所示。



图 5-7 注册页面



图 5-8 错误页面

## 5.7 实时在线人数

**实时在线人数:** 该功能实现实时在线人数的统计,主要通过判断连接的 **session** 个数来实现对人数的统计。如果有一个 **session** 被创建,那么人数加 1,有一个 **session** 消亡,人数减 1,从而达到统计人数的效果。

```
<%@ page language="java" pageEncoding="gb2312"%>
```

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>自动刷新</title>
    <style type="text/css">
      <!--
        body {
          background-image: url(images/img.jpg);
        }
      -->
    </style>
  </head>
  <script language="javascript">
    var XMLHttpRequest;
//创建 XMLHttpRequest 对象
    function createXMLHttpRequest() {
      if(window.XMLHttpRequest) {
        //Mozilla 浏览器
        XMLHttpRequest = new XMLHttpRequest();
      }
      else if (window.ActiveXObject) {
        // IE 浏览器
        try {
          XMLHttpRequest = new ActiveXObject("Msxml2.XMLHTTP");
        } catch(e) {
          try {
            XMLHttpRequest = new ActiveXObject("Microsoft.XMLHTTP");
          } catch (e) {}
        }
      }
    }
//发送请求函数
    function sendRequest() {
      createXMLHttpRequest();
      var url = "Random.jsp";
      XMLHttpRequest.open("GET", url, true);
      XMLHttpRequest.onreadystatechange = processResponse;//指定响应函数
      XMLHttpRequest.send(null); // 发送请求
    }
// 处理返回信息函数

```

```

function processResponse() {
    if (XMLHttpRequest.readyState == 4) { // 判断对象状态
        if (XMLHttpRequest.status == 200) { // 信息已经成功返回, 开始处理信息
            var str =XMLHttpRequest.responseText;
            document.getElementById("show").innerHTML=(str+"人");
            setTimeout("sendRequest()", 726);
        } else { // 页面不正常
            window.alert("您所请求的页面有异常。");
        }
    }
}
}
</script>
<body onload =sendRequest()><br>
<center>
    <b>实时在线人数</b><br><br>
    <div id="show"></div>
</center>
</body>
</html>

```

实现效果如图 5-9 所示。



图 5-9 实时在线人数效果

## 第 6 章

### jQuery 框架

---

在前面的章节中，使用了 CSS/JavaScript 等进行编程，构建出来的网站虽然比只使用 HTML 要美观得多，但还有很多不足的地方。在这里我们来学习使用 jQuery 进一步地美化页面，同时实现更多的效果。



## 6.1 jQuery 介绍

### 6.1.1 jQuery 的由来

jQuery 由美国人 John Resig 创建，其实就是一个工具包，很多常用的功能已经被封装好，只要直接调用就可以了（类似 SDK），具体内部原理不用关心。

jQuery 是一套 JavaScript 脚本库。JavaScript 脚本库类似于 .NET 的类库，将一些工具方法或对象方法封装在类库中，方便用户使用。

脚本库能够完成编码逻辑，实现业务功能。使用 jQuery 将极大地提高编写 JavaScript 代码的效率，让写出来的代码更加优雅、更加健壮。同时网络上丰富的 jQuery 插件也让工作变得更加的灵活。

jQuery 有两种版本：一种为 `uncompressed` 版（未压缩版），主要用在开发中；另一种为 `Minified`（迷你版），当开发完毕了，就可以改用这个版本。两种版本的区别在于文件大小不一样，最终用户在浏览时可以减少加载 JavaScript 文件的等待时间。`Uncompressed` 版文件大小为 118KB，`Minified` 版文件大小为 56KB。

jQuery 有如下特点：

#### 1. 提供了强大的功能函数

使用这些功能函数，能够快速完成各种功能，而且会让代码异常简洁。

#### 2. 解决浏览器兼容性问题

JavaScript 脚本在不同浏览器的兼容性一直是 Web 开发人员的噩梦，常常一个页面在 IE7、Firefox 下运行正常，在 IE6 下就出现了莫名其妙的问题。针对不同的浏览器编写不同的脚本是一件痛苦的事情。有了 jQuery，就能够解决这一问题，比如在 jQuery 中的 `event` 事件对象已经被格式化成所有浏览器通用的对象，从前要根据 `event` 获取事件触发者，在 IE 下是 `event.srcElements`，而 Firefox 等浏览器下是 `event.target`。jQuery 则通过统一 `event` 对象，就可以在所有浏览器中使用 `event.target` 获取事件对象了。

#### 3. 实现丰富的 UI

jQuery 可以实现比如渐变弹出、图层移动等动画效果，以提供更好的用户体验。虽然 JavaScript 使用大量代码也可以实现，但是费心费力不说，写完后如果没有大量的帮助文档，过一段时间就忘记了。再开发类似的功能还要再次费心费力，如今使用 jQuery 就可以快速完成此类应用。

#### 4. 纠正错误的脚本知识

大部分开发人员对于 JavaScript 存在错误的认识。比如在页面中编写加载时即执行操作 DOM 的语句，在 HTML 元素或者 `document` 对象上直接添加“`onclick`”属性，不知道 `onclick` 其实是一个匿

名函数，等等。拥有这些错误脚本知识的技术人员也能完成所有的开发工作，但是这样的程序是不健壮的。当页面代码很小、用户加载很快时没有问题，当页面加载稍慢时，就会出现浏览器“终止操作”的错误。jQuery 提供了很多简便的方法解决这些问题，一旦使用 jQuery 就能纠正这些错误的知识，因为使用的都是标准的 jQuery 脚本编写方法。

### 6.1.2 jQuery 配置

第一个 jQuery 程序。

#### 1. 下载 jQuery 类库

jQuery 的项目下载放在了 Google Code 上，下载地址：

<http://code.google.com/p/jqueryjs/downloads/list>

上面的地址是总下载列表，里面有很多版本和类型的 jQuery 库，主要分为如下几类：

- Min：压缩后的 jQuery 类库，在正式环境上使用，如 jQuery-1.3.2.min.js。
- vsdoc：在 Visual Studio 中需要引入此版本的 jQuery 类库，才能启用智能感知，如 jQuery-1.3.2-vsdoc2.js。
- release 包：里面有没有压缩的 jQuery 代码，以及文档和示例程序，如 jQuery-1.3.2-release.zip。

#### 2. 编写程序

下面编写一个 jQuery 版的 Hello World 程序作为第一个 jQuery 程序。

例 6-1

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Hello World jQuery!</title>
<script type="text/javascript" src="scripts/jquery-1.3.2-vsdoc2.js"></script>
</head>
<body>
<div id="divMsg">Hello World!</div>
<input id="btnShow" type="button" value="显示" />
<input id="btnHide" type="button" value="隐藏" /><br />
<input id="btnChange" type="button"
  value="修改内容为 Hello World, too!" />
<script type="text/javascript" >
  $("#btnShow").bind("click", function(event) { $("#divMsg").show(); });
  $("#btnHide").bind("click", function(event) { $("#divMsg").hide(); });
</script>
</body>
</html>
```

```

    $("#btnChange").bind("click", function(event) { $("#divMsg").html("Hello
World, too!"); });
</script>
</body>

```

在使用 jQuery 之前，必须首先引入前面提到的类库。在本例中使用的如下语句实现了对类库的引入：

```

<script type="text/javascript"
src="scripts/jquery-1.3.2-vsdoc2.js"></script>

```

运行的效果如图 6-1 所示。



图 6-1 运行效果

如图 6-1 所示，该例子除了实现显示打印出“Hello World”之外，还实现了对打印内容的显示和隐藏，并可以将打印的内容修改为“Hello World, too!”。

对于以上功能的具体实现，将会在后面的章节中具体学习。

### 6.1.3 jQuery 常用语法及接口

无论是写程序还是看 API 文档，要时刻注意区分 Dom 对象和 jQuery 包装集。

#### 1. Dom 对象

在传统的 JavaScript 开发中，都是首先获取 Dom 对象，比如：

```

var div = document.getElementById("testDiv");
var divs = document.getElementsByTagName("div");

```

使用 document.getElementById 方法根据 id 获取单个 Dom 对象，或者使用 document.getElementsByTagName 方法根据 HTML 标签名称获取 Dom 对象集合。

另外，在事件函数中，可以通过在方法函数中使用 this 引用事件触发对象，或者使用 event 对象的 target(FF) 或 srcElement(IE6) 获取引发事件的 Dom 对象。

这里获取的都是 Dom 对象，Dom 对象也有不同的类型，如 input、div、span 等。Dom 对象只有有限的属性和方法，如图 6-2 所示。

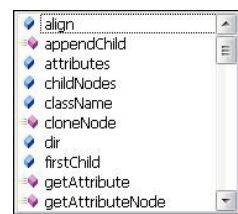


图 6-2 Dom 对象属性、方法

## 2. jQuery 包装集

jQuery 包装集可以说是 Dom 对象的扩充。在 jQuery 的世界中将所有的对象，无论是一个还是一组，都封装成一个 jQuery 包装集，比如获取包含一个元素的 jQuery 包装集：

```
var jQueryObject = $("#testDiv");
```

jQuery 包装集都是作为一个对象一起调用的。jQuery 包装集拥有丰富的属性和方法，jQuery 特有的属性和方法如图 6-3 所示。

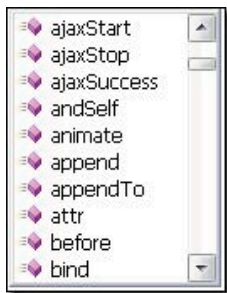


图 6-3 jQuery 特有的属性、方法

## 3. Dom 对象与 jQuery 对象的转换

### (1) Dom 转 jQuery 包装集

如果要使用 jQuery 提供的函数，首先就需要构造 jQuery 包装集。可以使用本文即将介绍的 jQuery 选择器直接构造 jQuery 包装集，比如：

```
$("#testDiv");
```

上面语句构造的包装集只含有一个 id 是 testDiv 的元素。或者已经获取了一个 Dom 元素，比如：

```
var div = document.getElementById("testDiv");
```

上面的代码中 div 是一个 Dom 元素，可以将 Dom 元素转换成 jQuery 包装集：

```
var domTojQueryObject = $(div);
```

### (2) jQuery 包装集转 Dom 对象

jQuery 包装集是一个集合，所以可以通过索引器访问其中的某一个元素：

```
var domObject = $("#testDiv")[0];
```

通过索引器返回的不再是 jQuery 包装集，而是一个 Dom 对象！jQuery 包装集的某些遍历方法，比如 each() 中，可以传递遍历函数，在遍历函数中的 this 也是 Dom 元素，比如：

```
$("#testDiv").each(function() { alert(this) })
```

如果要使用 jQuery 的方法操作 Dom 对象，用上面介绍过的转换方法即可：

```
$("#testDiv").each(function() { $(this).html("修改内容") })
```

## 6.2 jQuery 程序范例

### 6.2.1 选择器介绍

在 Dom 编程中只能使用有限的函数根据 id 或者 TagName 获取 Dom 对象。

在 jQuery 中则完全不同，jQuery 提供了异常强大的选择器用来获取页面上的对象，并且将对象以 jQuery 包装集的形式返回。首先来看看什么是选择器：

```
//根据 ID 获取 jQuery 包装集
var jQueryObject = $("#testDiv");
```

上面使用了 ID 选择器，选取 id 为 testDiv 的 Dom 对象并将它放入 jQuery 包装集，最后以 jQuery 包装集的形式返回。

“\$” 符号在 jQuery 中代表对 jQuery 对象的引用，“jQuery” 是核心对象，其中包含的方法，如表 6-1 所示。

表 6-1 jQuery 对象方法

| 方法                          | 说明   |
|-----------------------------|--|
| jQuery(expression, context) | 这个函数接收一个 CSS 选择器的字符串，然后用这个字符串去匹配一组元素         |
| jQuery(html, ownerDocument) | 根据 HTML 原始字符串动态创建 Dom 元素                     |
| jQuery(elements)            | 将一个或多个 Dom 对象封装 jQuery 函数功能（即封装为 jQuery 包装集） |
| jQuery(callback)            | \$(document).ready()的简写方式                    |

上面的方法就是选择器使用的核心方法。可以用“\$”代替 jQuery 让语法更简洁。

### 6.2.2 选择器详解

jQuery 的选择器支持 CSS3 选择器标准。标准选择器都可以在 jQuery 中使用。

jQuery 选择器按照功能主要分为“选择”和“过滤”，可以分别使用，也可以组合成一个选择器字符串同时使用。主要的区别是“过滤”作用的选择器是指定条件从前面匹配的内容中筛选，“过滤”选择器也可以单独使用，表示从全部“\*”中筛选，比如：

```
$(":[title]")
```

而“选择”功能的选择器则不会有默认的范围，因为作用是“选择”而不是“过滤”。

下面的选择器分类中，带有“过滤器”的分类表示是“过滤”选择器，否则就是“选择”功能的选择器。

基础选择器 Basics，如表 6-2 所示。

表 6-2 基础选择器

| 名称                              | 说明  | 举例   |
|---------------------------------|---|--|
| #id                             | 根据元素 id 选择                                      | \$("#divId"), 选择 id 为 divId 的元素                                  |
| element                         | 根据元素的名称选择                                       | \$( <code>a</code> ), 选择所有<a>元素                                  |
| .class                          | 根据元素的 CSS 类选择                                   | \$( <code>.bgRed</code> ), 选择所用 CSS 类为 bgRed 的元素                 |
| *                               | 选择所有元素  | \$( <code>*</code> ), 选择页面所有元素                                   |
| selector1, selector2, selectorN | 可以将几个选择器用“,”分隔开, 然后再拼成一个选择器字符串。会同时选中这几个选择器匹配的内容 | \$("#divId, a, .bgRed"), 选择所有标签为<a>的元素中, bgRed 类里 id 为 divId 的元素 |

基本过滤器 Basic Filters，如表 6-3 所示。

表 6-3 基本过滤器

| 名称             | 说明   | 举例   |
|----------------|--|--|
| :first         | 匹配找到的第一个元素                                   | 查找表格的第一行: \$( <code>tr:first</code> )  |
| :last          | 匹配找到的最后一个元素                                  | 查找表格的最后一行: \$( <code>tr:last</code> )  |
| :not(selector) | 去除所有与给定选择器匹配的元素                              | 查找所有未选中的 input 元素:<br>\$( <code>input:not(:checked)</code> )   |
| :even          | 匹配所有索引值为偶数的元素, 从 0 开始计数                      | 查找表格的第 1, 3, 5...行: \$( <code>tr:even</code> )   |
| :odd           | 匹配所有索引值为奇数的元素, 从 0 开始计数                      | 查找表格的第 2, 4, 6 行: \$( <code>tr:odd</code> )  |
| :eq(index)     | 匹配一个给定索引值的元素<br>注: index 从 0 开始计数            | 查找第 2 行: \$( <code>tr:eq(1)</code> )   |
| :gt(index)     | 匹配所有大于给定索引值的元素<br>注: index 从 0 开始计数          | 查找第 2 行、第 3 行, 即索引值是 1 和 2, 也就是比 0 大: \$( <code>tr:gt(0)</code> )  |
| :lt(index)     | 选择结果集中索引小于 N 的 elements<br>注: index 从 0 开始计数 | 查找第 1 行和第 2 行, 即索引值是 0 和 1, 也就是比 2 小:<br>\$( <code>tr:lt(2)</code> )   |
| :header        | 选择所有 h1、h2、h3 一类的 header 标签                  | 给页面内所有标题加上背景色:<br>\$( <code>:header</code> ).css( <code>"background"</code> , <code>"#EEE"</code> );   |
| :animated      | 匹配所有正在执行动画效果的元素                              | 只有对不在执行动画效果的元素执行一个动画特效:<br>\$( <code>"#run"</code> ).click(function(){ \$( <code>"div:not(:animated)"</code> ).animate({ left: <code>"+=20"</code> }, 1000); }); |

### 6.2.3 动态创建元素

#### 1. 错误的编程方法

使用 JavaScript 动态地创建元素时,有很多程序员通过直接更改某一个容器的 HTML 内容来进行。

例 6-2

```
<!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">
<head>
  <title>动态创建对象</title>
</head>
<body>
<div id="testDiv">测试图层</div>
<script type="text/javascript">
document.getElementById("testDiv").innerHTML =
  "<div style=\"border:solid 1px #FF0000\">动态创建的 div</div>";
</script>
</body>
</html>
```

上面的示例中通过修改 testDiv 的内容,想在页面上动态地添加一个 DIV 元素。但是请牢记,这是错误的做法!错误的原因如下:

- 在页面加载时改变了页面的结构。在 IE6 中,如果网络变慢或者页面内容太大,就会出现“终止操作”的错误。也就是说永远不要在页面加载时改变页面的 Dom 模型。
- 使用修改 HTML 内容添加元素,不符合 Dom 标准,在实际工作中也碰到过使用这种方法修改内容后,某些浏览器中并不能立刻显示添加的元素的情况,因为不同浏览器的显示引擎是不同的。但是如果使用 Dom 的 CreateElement 创建对象,在所有的浏览器中几乎都可以。但是在 jQuery 中,如果传入的值是一个完整的 HTML 字符串,那么内部也是使用 innerHTML,所以也不是完全否定 innerHTML 函数的使用。

所以,从现在开始请摒弃这种旧知识,使用下面介绍的正确方法编程。

#### 2. 创建新的元素

在 jQuery 中创建对象非常简单,比如创建一个 DIV 元素:

```
$("<div style=\"border:solid 1px #FF0000\">动态创建的 div</div>")
```

主要使用 jQuery 核心类库中的一个方法:

```
jQuery( html, ownerDocument )
```



Returns: jQuery

其中, HTML 参数是一个 HTML 字符串, 当 HTML 字符串是没有属性的元素时, 内部使用 `document.createElement` 创建元素, 比如:

```
//jQuery 内部使用 document.createElement 创建元素:
$("<div/>").css("border", "solid 1px #FF0000").html("动态创建的div").appendTo(testDiv);
```

否则使用 `innerHTML` 方法创建元素:

```
//jQuery 内部使用 innerHTML 创建元素:
$("<div style=\"border:solid 1px #FF0000\">动态创建的 div</div>").appendTo(testDiv);
```

### 3. 将元素添加到对象上

可以使用上面两种方式创建一个元素, 但是上面已经提到一定不要在页面加载时就改变页面的 DOM 结构, 比如添加一个元素。正确的做法是在页面加载完毕后, 添加或删除元素。传统做法上, 使用 `window.onload` 完成上述目的:

```
//DOM 加载完毕后添加元素
//传统方法
window.onload = function() {
testDiv.innerHTML = "<div style=\"border:solid 1px #FF0000\">
动态创建的 div</div>";
}
```

虽然能够在 DOM 完整加载后, 再添加新的元素, 但是不幸的是, 浏览器执行 `window.onload` 函数不仅仅是在构建完 DOM 树之后, 也是在所有图像和其他外部资源完整地加载并且在浏览器窗口显示完毕之后。所以, 如果某个图片或者其他资源加载很长时间, 访问者就会看到一个不完整的页面, 甚至在图片加载之前, 就执行了需要依赖动态添加的元素的脚本而导致脚本错误。

解决办法就是采用等 DOM 被解析后, 在图像和外部资源加载之前执行的函数。在 jQuery 中, 这一实现变得可行:

```
//jQuery 使用动态创建的$(document).ready(function)方法
$(document).ready(
    function() { testDiv.innerHTML =
"<div style=\"border:solid 1px #FF0000\">使用动态创建的$(document).ready(function)方法</div>"; }
);
```

或者使用简便语法:

```
//jQuery 使用$(function)方法
$(
    function() { testDiv.innerHTML +=
"<div style=\"border:solid 1px #FF0000\">使用$(function)方法</div>"; }
```



```
);
```

使用\$()将函数包装起来即可。而且可以在一个页面绑定多个函数,如果使用传统的 window.onload 则只能调用一个函数。

所以请大家将修改 DOM 的函数使用此方法调用。另外还要注意 document.createElement 和 innerHTML 的区别。如果可以, 请尽量使用 document.createElement 和\$("<div/>")的形式创建对象。

6.2.4 包装集元素管理

既然学会了动态创建元素, 接下来就学习把这些元素放入 jQuery 包装集中。

可以在 jQuery 包装集上调用下面这些函数, 用来改变原始 jQuery 包装集, 并且大部分返回的都是过滤后的 jQuery 包装集。

jQuery 提供了一系列的函数用来管理包装集, 分别介绍如下。

1. 过滤 Filtering

该包装集函数如表 6-4 所示。

表 6-4 过滤包装集

| 名称   | 说明  | 举例   |
|--|---|--|
| eq(index)  | 获取第 N 个元素   | 获取匹配的第二个元素:<br>\$("p").eq(1)   |
| filter(expr)                                     | 筛选出与指定表达式匹配的元素集合  | 保留带有 select 类的元素:<br>\$("p").filter(".selected")   |
| filter(fn)                                       | 筛选出与指定函数返回值匹配的元素集合<br>这个函数内部将对每个对象计算一次( 正如 '\$.each' )。如果调用的函数返回 false, 则这个元素被删除, 否则就会保留  | 保留子元素中不含有 ol 的元素:<br>\$("div").filter(function(index)<br>{return \$("ol", this).size() == 0; }); |
| is(expr)<br>注意: 这个函数返回的不是 jQuery 包装集而是 Boolean 值 | 用一个表达式来检查当前选择的元素集合, 如果其中至少有一个元素符合这个给定的表达式就返回 true<br>如果没有元素符合, 或者表达式无效, 都返回'false'。 'filter' 内部实际也是在调用这个函数, 所以, filter()函数原有的规则在这里也适用 | 由于 input 元素的父元素是一个表单元素, 所以返回 true:<br>\$("input[type='checkbox']").parent().is("form")           |

(续表)

| 名称               | 说明  | 举例   |
|------------------|---|--|
| map(callback)    | 将一组元素转换成其他数组（不论是否是元素数组）<br>你可以用这个函数来建立一个列表，不论是值、属性还是 CSS 样式，或者其他特别形式，都可以用\$.map()来方便地建立 | 把 form 中的每个 input 元素的值建立一个列表：<br>\$("p").append( \$("input").map(function(){<br>return \$(this).val(); }).get().join(", ")); |
| not(expr)        | 删除与指定表达式匹配的元素   | 从 p 元素中删除带有 select 的 ID 的元素：<br>\$("p").not( \$("#selected")[0] )  |
| slice(start,end) | 选取一个匹配的子集   | 选择第一个 p 元素：<br>\$("p").slice(0, 1);  |

## 2. 查找 Finding

该包装集的函数如表 6-5 所示。

表 6-5 查找包装集

| 名称                 | 说明  | 举例  |
|--------------------|---|---|
| add( expr )        | 把与表达式匹配的元素添加到 jQuery 对象中。这个函数可以用于连接分别与两个表达式匹配的元素结果集   | 动态生成一个元素并添加至匹配的元素中：<br>\$("p").add("<span>Again</span>")  |
| children( [expr] ) | 取得一个包含匹配的元素集合中每一个元素的所有子元素的元素集合<br>可以通过可选的表达式来过滤所匹配的子元素。注意：parents()将查找所有祖辈元素，而 children()只考虑子元素、不考虑所有后代元素 | 查找 DIV 中的每个子元素：<br>\$("div").children()   |
| closest( [expr] )  | 取得与表达式匹配的最新的父元素   | 为事件源最近的父类 li 对象更换样式：<br>\$(document).bind("click", function (e)<br>{ \$(e.target).closest("li").toggleClass("hilight"); }); |
| contents( )        | 查找匹配元素内部所有的子节点（包括文本节点）。如果元素是一个 iframe，则查找文档内容   | 查找所有文本节点并加粗：<br>\$("p").contents().not("[nodeType=1]").wrap("<b/>");  |

(续表)

| 名称                 | 说明   | 举例  |
|--------------------|--|---|
| find( expr )       | 搜索所有与指定表达式匹配的元素。这个函数是找出正在处理的元素的后代元素的好方法<br>所有搜索都依靠 jQuery 表达式来完成。这个表达式可以使用 CSS1~3 的选择器语法来写           | 从所有的段落开始, 进一步搜索下面的 span 元素, 与\$("p span")相同:<br>\$("p").find("span")  |
| next( [expr] )     | 取得一个包含匹配的元素集合中每一个元素紧邻的后面同辈元素的元素集合<br>这个函数只返回后面那个紧邻的同辈元素, 而不是后面所有的同辈元素(可以使用 nextAll )。可以用一个可选的表达式进行筛选 | 找到每个段落的后面紧邻的同辈元素:<br>\$("p").next()                                   |
| nextAll( [expr] )  | 查找当前元素之后所有的同辈元素。可以用表达式过滤   | 给第一个 DIV 之后的所有元素加个类:<br>\$("div:first").nextAll().addClass("after");  |
| offsetParent( )    | 返回第一个有定位的父类(比如 relative 或 absolute )   |   |
| parent( [expr] )   | 取得一个包含着所有匹配元素的唯一父元素的元素集合<br>可以使用可选的表达式来筛选  | 查找每个段落的父元素:<br>\$("p").parent()                                       |
| parents( [expr] )  | 取得一个包含着所有匹配元素的祖先元素的元素集合(不包含根元素)。可以通过一个可选的表达式进行筛选   | 找到每个 span 元素的所有祖先元素:<br>\$("span").parents()                          |
| prev( [expr] )     | 取得一个包含匹配的元素集合中每一个元素紧邻的前一个同辈元素的元素集合<br>可以用一个可选的表达式进行筛选<br>只有紧邻的同辈元素会被匹配到, 而不是前面所有的同辈元素                | 找到每个段落紧邻的前一个同辈元素:<br>\$("p").prev()                                   |
| prevAll( [expr] )  | 查找当前元素之前所有的同辈元素, 可以用表达式过滤  | 给最后一个之前的所有 DIV 加上一个类:<br>\$("div:last").prevAll().addClass("before"); |
| siblings( [expr] ) | 取得一个包含匹配的元素集合中每一个元素的所有唯一同辈元素的元素集合。<br>可以用可选的表达式进行筛选  | 找到每个 DIV 的所有同辈元素:<br>\$("div").siblings()                             |

### 3. 串联 Chaining

该包装集函数如表 6-6 所示。

表 6-6 串联包装集函数

| 名称        | 说明   | 举例  |
|-----------|--|---|
| andSelf() | 将之前匹配元素集合增加到当前匹配元素集合中，返回匹配元素集合   | 选取所有 DIV 以及内部的 p，并加上 border 类：<br>\$("div").find("p").andSelf().addClass("border"); |
| end()     | 回到最近的一个“破坏性”操作之前，即将匹配的元素列表变为前一次的状态<br>如果之前没有破坏性操作，则返回一个空集。所谓的“破坏性”就是指任何改变所匹配的 jQuery 元素的操作 | 选取所有的 p 元素，查找并选取 span 子元素，然后再回过来选取 p 元素：<br>\$("p").find("span").end()              |

### 6.2.5 DOM 操作：区分 DOM 属性和元素属性

一个 img 标签：

```

```

通常开发人员习惯将 id、src、alt 等叫做这个元素的“属性”，将其称为“元素属性”。但是在解析成 DOM 对象时，实际浏览器最后会将标签元素解析成“DOM 对象”，并且将元素的“元素属性”存储为“DOM 属性”。两者是有区别的。

虽然设置了元素的 src 是相对路径：images/image.1.jpg，但是在“DOM 属性”中都会转换成绝对路径：http://localhost/images/image.1.jpg。

甚至有些“元素属性”和“DOM 属性”的名称都不一样，比如上面的元素属性 class，转换为 DOM 属性后对应 className。

牢记，在 JavaScript 中可以直接获取或设置“DOM 属性”：

```
<script type="text/javascript">
    $(function() {
        var img1 = document.getElementById("hibiscus");
        alert(img1.alt);
        img1.alt = "Change the alt element attribute";
        alert(img1.alt);
    })
</script>
```

所以如果要设置元素的 CSS 样式类，要使用的是“DOM 属性”className 而不是“元素属性”class。

### 6.2.6 操作 DOM 属性

在 jQuery 中没有包装操作“DOM 属性”的函数，因为使用 JavaScript 获取和设置“DOM 属性”都很简单。在 jQuery 中，提供了 `each()` 函数用于遍历 jQuery 包装集，其中的 `this` 指针是一个 DOM 对象，所以可以应用这一点配合原生 JavaScript 来操作元素的 DOM 属性，如下所示。

```
$("#img").each(function(index) {  
    alert("index:" + index + ", id:" + this.id + ", alt:" + this.alt);  
    this.alt = "changed";  
    alert("index:" + index + ", id:" + this.id + ", alt:" + this.alt);  
});
```

下面是 `each` 函数的说明：

`Each( callback )` Returns: jQuery 包装集

对包装集中的每一个元素执行 `callback` 方法。其中，`callback` 方法接受一个参数，表示当前遍历的索引值，从 0 开始。

可以使用 JavaScript 中的 `getAttribute` 和 `setAttribute` 来操作元素的“元素属性”。

在 jQuery 中为你提供了 `attr()` 包装集函数，能够同时操作包装集中所有元素的属性，其使用方法如表 6-7 所示。

表 6-7 attr 函数

| 名称                              | 说明   | 举例  |
|---------------------------------|--|---|
| <code>attr( name )</code>       | 取得第一个匹配元素的属性值。通过这个方法可以方便地从第一个匹配元素中获取一个属性的值。如果元素没有相应属性，则返回 <code>undefined</code> | 返回文档中第一个图像的 <code>src</code> 属性值：<br><code>\$("#img").attr("src");</code>   |
| <code>attr( properties )</code> | 将一个“名/值”形式的对象设置为所有匹配元素的属性  | 为所有图像设置 <code>src</code> 和 <code>alt</code> 属性： <code>\$("#img").attr({ src: "test.jpg", alt: "Test Image" });</code>         |
| <code>attr( key, value )</code> | 为所有匹配的元素设置一个属性值  | 为所有图像设置 <code>src</code> 属性： <code>\$("#img").attr("src", "test.jpg");</code>   |
| <code>attr( key, fn )</code>    | 为所有匹配的元素设置一个计算的属性值<br>不提供值，而是提供一个函数，将这个函数计算的值作为属性值                               | 把 <code>src</code> 属性的值设置为 <code>title</code> 属性的值：<br><code>\$("#img").attr("title", function() { return this.src });</code> |
| <code>removeAttr( name )</code> | 从每一个匹配的元素中删除一个属性   | 将文档中图像的 <code>src</code> 属性删除： <code>\$("#img").removeAttr("src");</code>   |

当使用 ID 选择器时，常常返回只有一个对象的 jQuery 包装集，这个时候常使用 `attr(name)` 函数获得它的元素属性：

```
function testAttr1(event) {
    alert($("#hibiscus").attr("class"));
}
```

注意, `attr(name)` 函数只返回第一个匹配元素的特定元素属性值。而 `attr(key, name)` 会设置所有包装集中的元素属性:

```
//修改所有 img 元素的 alt 属性
$("#img").attr("alt", "修改后的 alt 属性");
```

而 `attr(properties)` 可以一次修改多个元素属性:

```
$("#img").attr({title:"修改后的 title", alt: "同时修改 alt 属性"});
```

另外,虽然可以使用 `removeAttr(name)` 删除元素属性,但是对应的 DOM 属性是不会被删除的,只会影响 DOM 属性的值。

比如,将一个 `input` 元素的 `readonly` 元素属性去掉,会导致对应的 DOM 属性变成 `false` (即 `input` 变成可编辑状态):

```
$("#inputTest").removeAttr("readonly");
```

## 6.2.7 修改元素的样式

可以修改元素 CSS 类或者直接修改元素的样式。

一个元素可以应用多个 CSS 类,但是不幸的是,在 DOM 属性中是用一个以空格分割的字符串存储的,而不是数组。所以,如果在原始 JavaScript 时代,想对元素添加或者删除多个属性时,都要自己操作字符串。

jQuery 让这一切变得异常简单,再也不用做那些无聊的工作了。

### 1. 修改 CSS 类

修改 CSS 类相关的 jQuery 方法如表 6-8 所示。

表 6-8 修改 CSS 类相关的方法

| 名称                                    | 说明                          | 实例  |
|---------------------------------------|-----------------------------|---|
| <code>addClass( classes )</code>      | 为每个匹配的元素添加指定的类名             | 为匹配的元素加上 'selected' 类:<br><code>\$("#p").addClass("selected");</code>     |
| <code>hasClass( class )</code>        | 判断包装集中是否至少有一个元素应用了指定的 CSS 类 | <code>\$("#p").hasClass("selected");</code>                               |
| <code>removeClass( [classes] )</code> | 从所有匹配的元素中删除全部或者指定的类         | 从匹配的元素中删除 'selected' 类:<br><code>\$("#p").removeClass("selected");</code> |
| <code>toggleClass( class )</code>     | 如果存在 (不存在) 就删除 (添加) 一个类     | 为匹配的元素切换 'selected' 类:<br><code>\$("#p").toggleClass("selected");</code>  |

(续表)

| 名称                           | 说明   | 实例   |
|------------------------------|--|--|
| toggleClass( class, switch ) | 当 switch 是 true 时添加类，当 switch 是 false 时删除类 | 每三次单击切换高亮样式：<br>var count = 0;<br>\$("p").click(function(){ \$(this).toggleClass("highlight", count++ % 3 == 0); }); |

使用上面的方法，可以对元素的 CSS 类像集合一样进行修改，再也不必手工解析字符串。

注意，addClass( class ) 和 removeClass( [classes] ) 的参数可以一次传入多个 CSS 类，用空格分割，比如：

```
$("#btnAdd").bind("click", function(event) { $("p").addClass("colorRed borderBlue"); });
```

removeClass 方法的参数可选，如果不传入参数则移除全部 CSS 类：

```
$("p").removeClass()
```

2. 修改 CSS 样式

同样，当想要修改元素的具体某一个 CSS 样式，即修改元素属性“style”时，jQuery 也提供了相应的方法，如表 6-9 所示。

表 6-9 修改 CSS 样式属性的方法

| 名称                 | 说明  | 实例   |
|--------------------|---|--|
| css( name )        | 访问第一个匹配元素的样式属性  | 取得第一个段落的 color 样式属性的值：<br>\$("p").css("color");                                  |
| css( properties )  | 把一个“名/值对”对象设置为所有匹配元素的样式属性<br>这是一种在所有匹配的元素上设置大量样式属性的最佳方式 | 将所有段落的字体颜色设为红色并且背景为蓝色：<br>\$("p").css({ color: "#ff0011", background: "blue" }); |
| css( name, value ) | 在所有匹配的元素中，设置一个样式属性的值<br>数字将自动转化为像素值                     | 将所有段落字体设为红色：<br>\$("p").css("color","red");                                      |

虽然可以通过获取属性、特性以及 CSS 样式来取得元素的几乎所有信息，但是请注意观察下面的实验，如例 6-3 所示。

例 6-3

```
<!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">
<head>
```



```

<title>获取对象宽度</title>
<script
type="text/javascript" src="scripts/jquery-1.3.2-vsdoc2.js"></script>
<script type="text/javascript">
    $(function()
    {
        alert("attr(\"width\") : "
        + $("#testDiv").attr("width")); //没有定义的
        alert("css(\"width\") : "
        + $("#testDiv").css("width")); //自动(ie6) 或 1264px(ff)
        alert("width() : " + $("#testDiv").width()); //正确的数值 1264
        alert("style.width : " + ($("#testDiv")[0].style.width )); //空值
    })
</script>
</head>
<body>
    <div id="testDiv">
        测试文本
    </div>
</body>
</html>

```

实现效果如图 6-4 所示。



图 6-4 获取宽度

希望获取测试图层的宽度，使用 **attr** 方法获取“元素特性”为 **undefined**，因为并没有为 **DIV** 添加 **width**。而使用 **css()** 方法虽然可以获取到 **style** 属性的值，但是在不同浏览器里返回的结果不同，IE6 下返回 **auto**，而 FF 下虽然返回了正确的数值，但是后面带有“px”。

针对上面的问题，jQuery 为常用的属性提供了获取和设置的方法，比如 **width()** 用来获取元素的宽度，而 **width(val)** 用来设置元素宽度。

下面这些方法可以用来获取元素的常用属性值。



1. 宽和高相关（height 和 width）

获取和宽、高相关的方法如表 6-10 所示。

表 6-10 获取宽和高

| 名称                      | 说明  | 举例                                   |
|-------------------------|---|--------------------------------------|
| height( )               | 取得第一个匹配元素当前计算的高度值（px）                                     | 获取第一段的高：<br>\$("p").height();        |
| height( val )           | 为每个匹配的元素设置 CSS 高度（eitht）属性的值<br>如果没有明确指定单位（如 em 或%），使用 px | 把所有段落的高设为 20：<br>\$("p").height(20); |
| width( )                | 取得第一个匹配元素当前计算的宽度值（px）                                     | 获取第一段的宽：<br>\$("p").width();         |
| width( val )            | 为每个匹配的元素设置 CSS 宽度（width）属性的值<br>如果没有明确指定单位（如 em 或%），使用 px | 将所有段落的宽设为 20：<br>\$("p").width(20);  |
| innerHeight( )          | 获取第一个匹配元素内部区域高度（包括补白、不包括边框）<br>此方法对可见和隐藏元素均有效             |                                      |
| innerWidth( )           | 获取第一个匹配元素内部区域宽度（包括补白、不包括边框）<br>此方法对可见和隐藏元素均有效             |                                      |
| outerHeight( [margin] ) | 获取第一个匹配元素外部高度（默认包括补白和边框）<br>此方法对可见和隐藏元素均有效                |                                      |
| outerWidth( [margin] )  | 获取第一个匹配元素外部宽度（默认包括补白和边框）<br>此方法对可见和隐藏元素均有效                |                                      |

关于获取高和宽的函数中，要注意“inner”、“outer”和“height/width”这三种函数的区别，如图 6-5 所示。

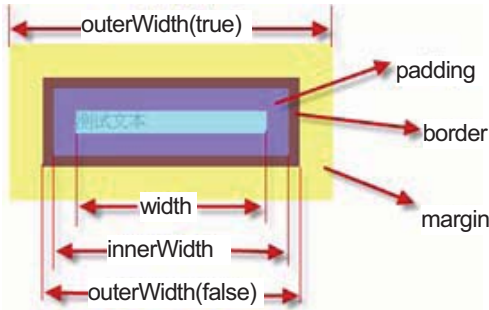


图 6-5 三种函数的区别

outerWidth 可以接受一个 Boolean 值参数表示是否计算 margin 值。

相信上图一目了然地反映了各个函数所取值的范围。图片以 `width` 为例进行说明, `height` 的各个函数同理。

## 2. 位置相关 Positioning

另外, 在一些设计弹出对象的脚本中, 常常需要动态获取弹出坐标并且设置元素的位置。

但是很多的计算位置的方法存在着浏览器兼容性问题, jQuery 提供了位置相关的各个函数, 如表 6-11 所示。

表 6-11 位置相关函数

| 名称                             | 说明  | 举例  |
|--------------------------------|---|---|
| <code>offset()</code>          | 获取匹配元素在当前窗口的相对偏移<br>返回的对象包含两个整型属性: <code>top</code> 和 <code>left</code> 。此方法只对可见元素有效                            | 获取第二段的偏移:<br><code>var p = \$("p:last");</code><br><code>var offset = p.offset();</code><br><code>p.html( "left: " + offset.left + ", top: " + offset.top );</code>                       |
| <code>position()</code>        | 获取匹配元素相对父元素的偏移<br>返回的对象包含两个整型属性: <code>top</code> 和 <code>left</code> 。为精确计算结果, 请在补白、边框和填充属性上使用像素单位。此方法只对可见元素有效 | 获取第一段的偏移:<br><code>var p = \$("p:first");</code><br><code>var position = p.position();</code><br><code>\$( "p:last" ).html( "left: " + position.left + ", top: " + position.top );</code> |
| <code>scrollTop()</code>       | 获取匹配元素相对滚动条顶部的偏移<br>此方法对可见和隐藏元素均有效  | 获取第一段相对滚动条顶部的偏移:<br><code>var p = \$("p:first");</code><br><code>\$( "p:last" ).text( "scrollTop:" + p.scrollTop() );</code>  |
| <code>scrollTop( val )</code>  | 传递参数值时, 设置垂直滚动条顶部偏移为该值<br>此方法对可见和隐藏元素均有效  | 设定垂直滚动条值:<br><code>\$( "div.demo" ).scrollTop(300);</code>  |
| <code>scrollLeft()</code>      | 获取匹配元素相对滚动条左侧的偏移<br>此方法对可见和隐藏元素均有效  | 获取第一段相对滚动条左侧的偏移:<br><code>var p = \$("p:first");</code><br><code>\$( "p:last" ).text( "scrollLeft:" + p.scrollLeft() );</code>  |
| <code>scrollLeft( val )</code> | 传递参数值时, 设置水平滚动条左侧偏移为该值<br>此方法对可见和隐藏元素均有效  | 设置相对滚动条左侧的偏移:<br><code>\$( "div.demo" ).scrollLeft(300);</code>   |

## 6.3 事件

事件是脚本编程的灵魂。本节内容也是 jQuery 学习的重点。本节将对 jQuery 中的事件处理以及事件对象进行详细的讲解。

### 6.3.1 事件和事件对象

首先看一下经常使用的添加事件的方式，如例 6-4 所示。

例 6-4

```
<!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">
<head>
  <title>javascript 中的事件</title>

<script
type="text/javascript" src="scripts/jquery-1.3.2-vsdoc2.js"></script>

  <script type="text/javascript">
    $(function()
    {
      //获取对象添加单击事件
      document.getElementById("testDiv2").onclick = showMsg;
    })

    function showMsg(event)
    {
      alert("!!!");
    }
  </script>
</head>
<body>
  <div id="testDiv1" onclick="showMsg();">单击事件 1</div>
  <div id="testDiv2">单击事件 2</div>
</body>
</html>
```

效果如图 6-6 所示。



图 6-6 单击事件

Web 中最常使用的事件添加方式是：为元素添加 `onclick` 元素属性。

为 `testDiv2` 添加 `onclick` 事件的方式是修改 Dom 属性。

在上一章中已经说明了什么是元素属性、什么是 Dom 属性。这两种方式的效果相同。当单击 DIV 时会显示提示框。

请注意，虽然效果相同，但是并不等效。

```
document.getElementById("testDiv2").onclick = showMsg;
```

上一条代码等效于：

```
<div id="testDiv1" onclick="alert('!!!');">单击事件 1</div>
```

注意到两者的区别了吗？常用的修改元素属性添加事件的方式，实际上是建立了一个匿名函数：

```
document.getElementById("testDiv1").onclick = function(event)
{
    alert("!!!");
};
```

这个匿名函数的签名和手写的 `showMsg` 签名相同，所以可以把 `showMsg` 直接赋值给 `onclick`。这种方式的弊端是：

- 只能为一个事件绑定一个事件处理函数。使用“=”赋值会把前面为此事件绑定的所有事件处理函数冲掉。
- 在事件函数（无论是匿名函数还是绑定的函数）中获取事件对象的方式在不同浏览器中要特殊处理。

IE 中，事件对象是 `window` 对象的一个属性。事件处理函数必须这样访问事件对象：

```
obj.onclick=function()
{
    var oEvent = window.event;
}
```

在 DOM 标准中，事件对象必须作为唯一参数传给事件处理函数：

```
obj.onclick=function()
{
    var oEvent = arguments[0];
}
```

除了使用 `argument[0]` 访问此参数，也可以指定参数名称，上面的代码等同于：

```
obj.onclick=function(oEvent)
{
}
```

目前兼容 DOM 的浏览器有 Firefox、Safari、Opera、IE7 等。

6.3.2 jQuery 中的事件

jQuery 提供了处理对象事件的一系列函数。上面的基础知识还是要弄懂，但是再也不用自己去实现处理多播事件委托的函数了。下面是在 jQuery 中最常使用的委托函数的 bind()方法举例：

```
$("#testDiv4").bind("click", showMsg);
```

为 id 是 testDiv4 的元素，添加触发 click 事件的处理函数 showMsg。

使用 jQuery 事件处理函数的好处有以下几点。

1. 添加的是多播事件委托

也就是为 click 事件又添加了一个方法，不会覆盖对象的 click 事件原有的事件处理函数。

```
$("#testDiv4").bind("click", function(event) { alert("one"); });
$("#testDiv4").bind("click", function(event) { alert("two"); });
```

单击 testDiv4 对象时，依次提示 “one” 和 “two” 。

2. 统一了事件名称

添加多播事件委托时，IE 中的事件名称前面有 “on” 。 但是使用 bind()函数时，不用区分 IE 和 DOM，因为内部 jQuery 已经统一了事件的名称。

3. 可以将对象行为全部用脚本控制

让 HTML 代码部分只注意 “显示” 逻辑。现在的趋势是将 HTML 的行为、内容与样式切分干净。其中用脚本控制元素行为，用 HTML 标签控制元素内容，用 CSS 控制元素样式。使用 jQuery 事件处理函数可以避免在 HTML 标签上直接添加事件。

下面是基础的 jQuery 事件处理函数，如表 6-12 所示。

表 6-12 事件处理函数 Event Handling

| 函数名称                     | 说明                                   | 举例   |
|--------------------------|--------------------------------------|--|
| bind( type, [data], fn ) | 为每一个匹配元素的特定事件（如 click）绑定一个事件处理器函数    | 当每个段落被单击的时候，弹出其文本：<br>\$("p").bind("click", function(){<br>alert( \$(this).text() ); });     |
| one( type, [data], fn )  | 为每一个匹配元素的特定事件（如 click）绑定一个一次性的事件处理函数 | 当所有段落被第一次单击的时候，显示所有其文本：<br>\$("p").one("click", function(){<br>alert( \$(this).text() ); }); |

(续表)

| 函数名称  | 说明   | 举例  |
|---|--|---|
| <code>trigger( event, [data] )</code>         | 在每一个匹配的元素上触发某类事件。<br>这个函数也会导致浏览器同名的默认行为的执行。比如, 如果用 <code>trigger()</code> 触发一个 'submit', 则同样会导致浏览器提交表单, 如果要阻止这种默认行为, 应返回 <code>false</code> 可以触发由 <code>bind()</code> 注册的自定义事件           | 给一个事件传递参数:<br><code>\$( "p" ).click( function (event, a, b) {<br/>// 一个普通的单击事件时, a 和 b 是 //undefined 类型<br/>// 如果用下面的语句触发, 那么 a //指向"foo", 而 b 指向"bar"<br/>}).trigger("click", ["foo", "bar"]);</code>  |
| <code>triggerHandler ( event, [data] )</code> | 这个特别的方法将会触发指定的事件类型上所有绑定的处理函数, 但不会执行浏览器默认动作   | 如果你对 一个 <code>focus</code> 事件执行了 <code>triggerHandler()</code> , 浏览器默认动作将不会被触发, 只会触发你绑定的动作:<br><code>\$( "#old" ).click(function(){<br/>\$( "input" ).trigger("focus"); });<br/>\$( "#new" ).click(function(){<br/>\$( "input" ).triggerHandler("focus"); });<br/>\$( "input" ).focus(function(){ \$( "&lt;span&gt;Focused!&lt;/span&gt;" ).appendTo("body").<br/>fadeOut(1000); });</code> |
| <code>unbind( type, fn )</code>               | <code>bind()</code> 的反向操作, 从每一个匹配的元素中删除绑定的事件<br>如果没有参数, 则删除所有绑定的事件。<br>可以将 <code>bind()</code> 注册的自定义事件取消绑定。<br>如果提供了事件类型作为参数, 则只删除该类型的绑定事件<br>如果把在绑定时传递的处理函数作为第二个参数, 则只有这个特定的事件处理函数会被删除 | 把所有段落的所有事件取消绑定:<br><code>\$( "p" ).unbind()</code><br>将段落的 <code>click</code> 事件取消绑定:<br><code>\$( "p" ).unbind( "click" )</code><br>删除特定函数的绑定, 将函数作为第二个参数传入:<br><code>var foo = function () {<br/>// 处理某个事件的代码<br/>};<br/>\$( "p" ).bind("click", foo);<br/>//当单击段落的时候会触发 foo<br/>\$( "p" ).unbind("click", foo);<br/>//再也不会被触发 foo</code>   |

常用时间函数:

1. `bind( type, [data], fn )` 函数举例`bind()` 是最常使用的函数, 可选的第二个参数 `data`, 可以把一些附加信息传递给事件处理函数:

```
function handler(event) {
    alert(event.data.foo); }
$( "p" ).bind("click", {foo: "bar"}, handler)
```

注意 `event` 参数的使用。jQuery 中统一了事件对象, 将事件对象作为事件处理函数的唯一参数传递。

**data** 参数也要通过 **event.data** 进行访问。为何要提供 **data** 参数呢？因为经常会碰到这样的问题：希望在事件处理中根据事件源的某些数据进行特殊处理。目前有下面两种解决方法。

（1）使用自定义元素属性存储数据。

比如：

```
<div id="testDiv5" customer="customer data 1">获取自定义数据-1</div>
```

在事件处理函数中获取数据：

```
$("#testDiv5").bind("click",  
function(event) { alert($(event.target).attr("customer")); });
```

**attr** 函数，用于获取元素的“元素属性”，而且可以获取自定义的元素属性，单击 **div** 后将显示，如图 6-7 所示。

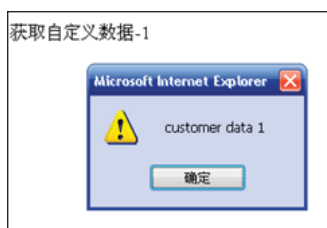


图 6-7 事件函数 1

（2）使用脚本将数据传递给事件处理函数。

```
<div id="testDiv6">获取自定义数据-2</div>
```

元素没有任何的自定义属性，添加事件处理函数时将额外的数据传递：

```
$("#testDiv6").bind("click", { customer: "customer data 2" }, function(event)  
{ alert(event.data.customer) });
```

单击 **div** 后的结果和方法（1）相同（如图 6-8 所示）。



图 6-8 事件函数 2

方法 1：便于存储和查找数据，但是自定义属性不能通过 W3C 验证。

方法 2：必须要自己想办法存储数据，并且要制定规则查找指定元素的数据。

从“开发人员”的角度，方法（1）要更加简单直观。但是缺点比较严重。所以如何取舍请大家自己决定。

`one( type,[data],fn )` 函数和 `bind` 一样，但是只执行一次。

## 2. `trigger( event,[data] )`和 `triggerHandler(event,[data] )`

虽然为元素绑定了某些事件，比如 `click`，但是有时希望在程序中触发这些事件，这两个函数可以实现此功能。

主要区别是 `trigger` 会触发浏览器默认的动作，而 `triggerHandler` 不会触发。

通过下面的例子可以明确地区分这两个函数。

例 6-5

```
<!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">
<head>
  <title>jQuery 事件处理:trigger 和 triggerHandler 示例</title>

<script
type="text/javascript" src="scripts/jquery-1.3.2-vsdoc2.js"></script>

  <script type="text/javascript">
    $(function()
    {
      $("#old").click(function()
      {
        $("#divResult").html("");
        //使用 trigger 函数绑定事件
        $("input").trigger("focus");
      });
      $("#new").click(function()
      {
        $("#divResult").html("");
        //使用 triggerHandler 函数绑定事件
        $("input").triggerHandler("focus");
      });
      $("input").focus(function(){
        $("<span>Focused!</span>").appendTo("#divResult");
      });
    })

  </script>
```



```
</head>
<body>
  <button id="old">
    .trigger("focus")</button>
  <button id="new">
    .triggerHandler("focus")</button><br />
  <br />
  <input type="text" value="To Be Focused" />
  <div id="divResult"></div>
</body>
</html>
```

当单击“.trigger”按钮时，会调用两次 Focused，并且 input 元素获得了焦点，如图 6-9 所示。单击“.triggerHandler”按钮时，只调用一次，并且 input 元素没有获得焦点，如图 6-10 所示。



图 6-9 trigger 函数



图 6-10 triggerHandler 函数

也就是说，trigger 函数发出了浏览器默认的获取焦点的行为，让 input 元素获得了焦点，所以再次调用了 focus 事件处理函数。

triggerHandler 只调用为 focus 事件绑定的事件处理函数，而不引发浏览器行为，所以最后 input 元素没有获得焦点。

## 6.4 利用 jQuery 实现页面特效

**实现文字高亮显示：**鼠标移动到图片上，图片透明显示的效果。

未移动到图片上时，如图 6-11 所示。



图 6-11 未移动鼠标到图片上时的效果

将鼠标移动到图片上后，透明显示，如图 6-12 所示。



图 6-12 移动鼠标到图片上时的效果

本特效使用 jQuery 完成。其中使用了 `hover()` 方法。

参考实现：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>hover() 方法</title>
<style type="text/css">
<!--
body{
    /* 设置背景图片，以突出透明度的效果 */
    background:url(bg1.jpg);
    margin:20px; padding:0px;
}
```

```

img{
    border:1px solid #FFFFFF;
}
-->
</style>
<script language="javascript" src="jQuery.min.js"></script>
<script language="javascript">
$(function(){
    $("img").hover(
        function(oEvent){
            //第一个函数相当于 mouseover 事件监听
            $(oEvent.target).css("opacity","0.5");
        },
        function(oEvent){
            //第二个函数相当于 mouseover 事件监听
            $(oEvent.target).css("opacity","1.0");
        }
    );
});
</script>
</head>
<body>
    
</body>
</html>

```

**实现下拉菜单：**在页面中实现下拉菜单。

HTML 部分代码如下所示，主要通过层实现和列表：

```

<div id="menu">
<ul>
<li><a href="">新闻</a>
    <ul>
        <li><a href="">国内新闻</a></li>
        <li><a href="">国际新闻</a>
            <ul><li><a href="">趣闻</a></li></ul>
        </li>
        <li><a href="">社会图片</a></li>
    </ul>
</li>
<li><a href="">娱乐</a>
    <ul>

```

```

    <li><a href="">音乐</a></li>
    <li><a href="">图书</a></li>
    <li><a href="">电影</a></li>
  </ul>
</li>
</ul>
</div>

```

CSS 控制代码如下所示:

```

ul,ol,li{list-style:none;padding:0px;margin:6px; float:left;}
#menu *{line-height:30px;}
//设置层中的 a 标签
#menu a{
    text-decoration:none;
    display:block;
}
//设置层中的 ul 标签
#menu ul{
    text-align:left;
    background:#FF0033;
}
//设置层中 class 为 arrow 的标签
#menu .arrow{
    padding-right:5px;
}
#menu>ul{height:30px;}
//设置层下的 ul 下的 li 标签
#menu>ul>li{
    text-align:center;
    display:inline-block;
    width:80px;
}
//设置层下的 ul 下的 li 中的 a 标签
#menu>ul>li>a{color:#FFFFFF;}
#menu>ul>li:hover{background:#FF3399;}

#menu>ul>li ul{
    display:none;
    width:150px;
    position:absolute;
    background:#C1Cd94;
    box-shadow:2px 2px 2px #000000;
}

```

```

-webkit-box-shadow:2px 2px 2px #000000;
-moz-box-shadow:2px 2px 2px #225599;
}

```

JS 的控制代码如下:

```

$(document).ready(function()
{
    $('#menu>ul>li>ul').find('li:has(ul:not(:empty))>a').append("<span
class='arrow'>></span>");
    //添加对于 li 标签,鼠标移动到其上时和鼠标离开时,其子标签 ul 的事件
    $('#menu>ul>li').bind('mouseover',function()    {
        $(this).children('ul').slideDown('fast');
    }).bind('mouseleave',function()
    {
        $(this).children('ul').slideUp('fast');
    });
    //添加对于最底层的 li 标签,鼠标移动到其上时和鼠标离开时,其子标签 ul 的事件
    $('#menu>ul>li>ul li').bind('mouseover',function()    {
        //移动到上面时展开
        $(this).children('ul').slideDown('fast');
    }).bind('mouseleave',function()
    {
        //移出时收起
        $(this).children('ul').slideUp('fast');
    });
});

```

实现效果如图 6-13 所示的菜单(鼠标未移上时)。



图 6-13 下拉菜单

当把鼠标移动到该菜单上时,菜单展开,如图 6-14 所示。

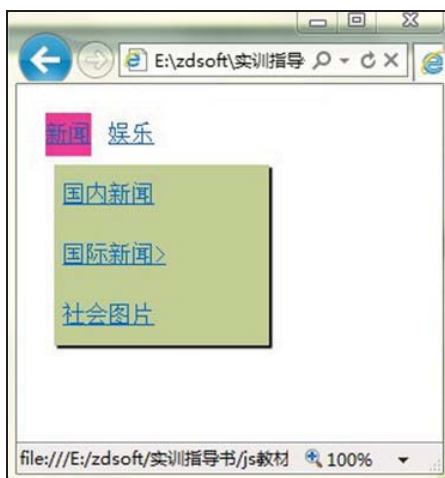


图 6-14 展开之后的下拉菜单

## 6.5 实现鼠标单击留言切换高亮显示

对文字单击实现高亮显示。没有单击的效果，如图 6-15 所示。

单击之后的效果，如图 6-16 所示。



图 6-15 没有单击的效果



图 6-16 高亮效果

要完成该功能可以使用 jQuery 更改 CSS 样式的方法。通过 toggleClass() 方法实现。

```
<html>
<head>
<title>toggleClass() 方法</title>
<style type="text/css">
<!--
//设置 p 标签
p{
    color:blue; cursor:help;
    font-size:13px;
    margin:0px; padding:5px;
}
//设置 class 为 highlight 的标签
```

```

.highlight{
    background-color:#FFFF00;
}
-->
</style>
<script language="javascript" src="jQuery.min.js"></script>
<script language="javascript">
$(function(){
    $("p").click(function(){
        //单击的时候不断切换
        $(this).toggleClass("highlight");
    });
});
</script>
</head>
<body>
    <p>高亮? </p>
</body>
</html>

```

## 6.6 快餐在线

本节实现快餐在线系统的设计。

其初始界面如图 6-17 所示。



图 6-17 快餐在线初始界面

每选中其中一个复选框后，可以弹出一组单选按钮菜单，并计算出合计的金额，如图 6-18 所示。

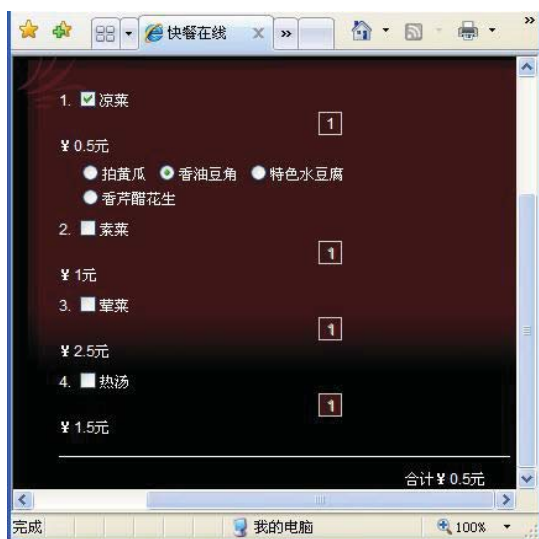


图 6-18 展开单选按钮菜单

- 计算总价时，对所有选中的复选框进行遍历，获取数量和单价进行计算。不用考虑具体选择的是哪种菜。
- 选中复选框时，根据其 `checked` 属性设置单选按钮组所在层的 `display` 属性的值为 `none` 或者 `block`。
- 每次改变选中状态时，都要将数量重新置为 1，并重新计算价格。

参考实现：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>快餐在线</title>
<style type="text/css">
<!--
//设置 body 标签
body{
    padding:0px;
    margin:165px 0px 0px 160px;
    font-size:12px;
    font-family:Arial, Helvetica, sans-serif;
    color:#FFFFFFF;
    background:#000000 url(bg2.jpg) no-repeat;
}
//设置 body 下面的层
```



```

body > div{
    margin:5px; padding:0px;
}
//设置层下面的 class 为 detail 的标签
div.detail{
    display:none;
    margin:3px 0px 2px 15px;
}
//设置层下面的 id 为 totalPrice 的标签
div#totalPrice{
    padding:10px 0px 0px 280px;
    margin-top:15px;
    width:85px;
    border-top:1px solid #FFFFFF;
}
//设置 input 标签
input{
    font-size:12px;
    font-family:Arial, Helvetica, sans-serif;
}
//设置 input 标签中 class 为 quantity 的标签
input.quantity{
    border:1px solid #CCCCCC;
    background:#3f1415; color:#FFFFFF;
    width:15px; text-align:center;
    margin:0px 0px 0px 210px
}
-->
</style>
<script language="javascript" src="jQuery.min.js"></script>
<script type="text/javascript">
function addTotal(){
    //计算总价格的函数
    var fTotal = 0;
    //对于选中的复选项进行遍历
    $("input:checkbox:checked").each(function(){
        //获取每一个的数量
        var iNum = parseInt($(this).parent().find("input[type=text]").val());
        //获取每一个的单价
        var fPrice = parseFloat($(this).parent().find("span[price]").attr("price"));
        fTotal += iNum * fPrice;
    });
}

```

```

    });
    $("#totalPrice").html("合计¥ "+fTotal+"元");
}
$(function() {
    $(".checkbox").click(function() {
        var bChecked = this.checked;
        //如果选中则显示子菜单
        $(this).parent().find(".detail").css("display",bChecked?"block":"none");
        $(this).parent().find("input[type=text]")
            //每次改变选中状态,都将值重置为 1,触发 change 事件,重新计算价格
            .attr("disabled",!bChecked).val(1).change()
            .each(function() {
                //需要聚焦判断,因此采用 each 来插入语句
                if(bChecked) this.focus();
            });
    });
    $("span[price] input[type=text]").change(function() {
        //根据单价和数量计算价格
        $(this).parent().find("span").text( $(this).val() * $(this).parent().attr(
            "price") );
        addTotal(); //计算总价格
    });
    //加载页面完全后,统一设置输入文本框
    $("span[price] input[type=text]")
        .attr({ "disabled":true, //文本框为 disable
                "value":"1", //表示份数的 value 值为 1
                "maxlength":"2" //最多只能输入两位数 (不提供 100 份以上)
            })
        .change(); //触发 change 事件,让 span 都显示出价格
});
</script>
</head>

<body>
<div>
<input type="checkbox" id="LiangCaiCheck"><label for="LiangCaiCheck">凉菜</label>
<span price="0.5"><input type="text" class="quantity"> ¥<span></span>元</span>
    <div class="detail">
        <label><input type="radio" name="LiangCai" checked="checked">拍黄瓜</label>
        <label><input type="radio" name="LiangCai">香油豆角</label>
        <label><input type="radio" name="LiangCai">特色水豆腐</label>
        <label><input type="radio" name="LiangCai">香芹醋花生</label>
    </div>
</div>
</body>
</html>

```

```

        </div>
</div>

<div>
<input type="checkbox" id="SuCaiCheck"><label for="SuCaiCheck">素菜</label>
<span price="1"><input type="text" class="quantity"> ¥<span></span>元</span>
    <div class="detail">
        <label><input type="radio" name="SuCai" checked="checked">虎皮青椒</label>
        <label><input type="radio" name="SuCai">醋溜土豆丝</label>
        <label><input type="radio" name="SuCai">金钩豆芽</label>
    </div>
</div>

<div>
<input type="checkbox" id="HunCaiCheck"><label for="HunCaiCheck">荤菜</label>
<span price="2.5"><input type="text" class="quantity"> ¥<span></span>元</span>
    <div class="detail">
        <label><input type="radio" name="HunCai" checked="checked"/>麻辣肉片</label>
        <label><input type="radio" name="HunCai">红烧牛柳</label>
        <label><input type="radio" name="HunCai">糖醋里脊</label>
    </div>
</div>

<div>
<input type="checkbox" id="SoupCheck"><label for="SoupCheck">热汤</label>
<span price="1.5"><input type="text" class="quantity"> ¥<span></span>元</span>
    <div class="detail">
        <label><input type="radio" name="Soup" checked="checked"/>西红柿鸡蛋汤
</label>
        <label><input type="radio" name="Soup">南瓜汤</label>
    </div>
</div>

<div id="totalPrice"></div>
</body>
</html>

```