



附本书源代码光盘

珍藏版

网站开发非常之旅

强 锋 科 技

技术支持:QQ群(21948169) / 论坛(<http://www.rzchina.net>)

陈 华 编著

# Ajax

## 从入门到精通

- ☑ 讲解 Ajax 的原理、开发、调试、跨浏览器兼容等高级技术
- ☑ 用 JavaScript 实现最流行的 Ajax 框架和组件
- ☑ 解读最经典的 JavaScript 框架 Prototype.js
- ☑ 详细介绍 Web 开发技术的优化
- ☑ 分析 JavaScript 的面向对象编程和 Ajax 的数据形式 JSON
- ☑ 配有源代码光盘, 提高学习效率

清华大学出版社

# 网站开发非常之旅



## 丛书特色：

- ✓ 作者均为有丰富的网络编程经验的一线开发人员
- ✓ 全面攻克网络编程领域的技术难点与热点
- ✓ 极大地提升您的网络应用开发水准
- ✓ 确保技术的先进性、实用性和深入性
- ✓ 贯穿丰富的应用实例，真正做到学以致用
- ✓ 专门设计了综合案例，让您领会项目开发的整体思路
- ✓ 配多媒体光盘讲述界面操作，既节省篇幅，又容易上手

ISBN 978-7-302-17642-8



9 787302 176428 >

定价:76.00元(附光盘1张)



## 网站开发非常之旅

# Ajax 从入门到精通

**强锋科技**

陈 华 编著

010-62585888 13501151833 地址: 北京海淀区中关村大街28号 邮编: 100080 电话: 010-62585888 13501151833 传真: 010-62585888 13501151833

男妙 (910) 目錄別註詳圖

8-5421-506-7-859 7821

莫五學、陳明升費

© 2002 Blackwell Science Ltd

12345678910111213141516171819202122232425262728293031323334353637383940414243444546474849505152535455565758596061626364656667686970717273747576777879808182838485868788899091929394959697989910010110210310410510610710810911011111211311411511611711811912012112212312412512612712812913013113213313413513613713813914014114214314414514614714814915015115215315415515615715815916016116216316416516616716816917017117217317417517617717817918018118218318418518618718818919019119219319419519619719819920020120220320420520620720820921021121221321421521621721821922022122222322422522622722822923023123223323423523623723823924024124224324424524624724824925025125225325425525625725825926026126226326426526626726826927027127227327427527627727827928028128228328428528628728828929029129229329429529629729829930030130230330430530630730830931031131231331431531631731831932032132232332432532632732832933033133233333433533633733833934034134234334434534634734834935035135235335435535635735835936036136236336436536636736836937037137237337437537637737837938038138238338438538638738838939039139239339439539639739839940040140240340440540640740840941041141241341441541641741841942042142242342442542642742842943043143243343443543643743843944044144244344444544644744844945045145245345445545645745845946046146246346446546646746846947047147247347447547647747847948048148248348448548648748848949049149249349449549649749849950050150250350450550650750850951051151251351451551651751851952052152252352452552652752852953053153253353453553653753853954054154254354454554654754854955055155255355455555655755855956056156256356456556656756856957057157257357457557657757857958058158258358458558658758858959059159259359459559659759859960060160260360460560660760860961061161261361461561661761861962062162262362462562662762862963063163263363463563663763863964064164264364464564664764864965065165265365465565665765865966066166266366466566666766866967067167267367467567667767867968068168268368468568668768868969069169269369469569669769869970070170270370470570670770870971071171271371471571671771871972072172272372472572672772872973073173273373473573673773873974074174274374474574674774874975075175275375475575675775875976076176276376476576676776876977077177277377477577677777877978078178278378478578678778878979079179279379479579679779879980080180280380480580680780880981081181281381481581681781881982082182282382482582682782882983083183283383483583683783883984084184284384484584684784884985085185285385485585685785885986086186286386486586686786886987087187287387487587687787887988088188288388488588688788888989089189289389489589689789889990090190290390490590690790890991091191291391491591691791891992092192292392492592692792892993093193293393493593693793893994094194294394494594694794894995095195295395495595695795895996096196296396496596696796896997097197297397497597697797897998098198298398498598698798898999099199299399499599699799899910001001100210031004100510061007100810091010101110121013101410151016101710181019102010211022102310241025102610271028102910301031103210331034103510361037103810391040104110421043104410451046104710481049105010511052105310541055105610571058105910601061106210631064106510661067106810691070107110721073107410751076107710781079108010811082108310841085108610871088108910901091109210931094109510961097109810991100110111021103110411051106110711081109111011111112111311141115111611171118111911201121112211231124112511261127112811291130113111321133113411351136113711381139114011411142114311441145114611471148114911501151115211531154115511561157115811591160116111621163116411651166116711681169117011711172117311741175117611771178117911801181118211831184118511861187118811891190119111921193119411951196119711981199120012011202120312041205120612071208120912101211121212131214121512161217121812191220122112221223122412251226122712281229123012311232123312341235123612371238123912401241124212431244124512461247124812491250125112521253125412551256125712581259126012611262126312641265126612671268126912701271127212731274127512761277127812791280128112821283128412851286128712881289129012911292129312941295129612971298129913001

7082610 臺灣省海山縣志

化学工业出版社 编

宇千 190 厘米 空 160 厘米 重 60 公斤 1946 年 12 月 20 日 本 国

清华大学出版社

北京

## 内 容 简 介

本书从开发 Ajax 应用所需要具备的基本能力开始,逐步深入地向读者介绍 JavaScript 的基本知识、高级技巧、开发工具的使用、开发框架的介绍、浏览器兼容性的问题、调试技巧、Ajax 应用程序的组成、不同的数据组织方式,以及开发过程中经常会遇到的问题,并对 Ajax 应用程序的架构进行了探讨,是一本全面的、适合各种水平层次的读者阅读和学习的教程。

本书作为比较全面的 Ajax 教程书籍,涵盖了基本语言介绍 (JavaScript)、DHTML 技术 (DOM)、Ajax 技术核心知识、面向对象的 JavaScript、数据组织方式 (XML 和 JSON) 等知识,并对开发工具和调试技巧以及流行的应用框架进行了介绍。尤其是本书的第 10~15 章,介绍了 Ajax 的高级技术,包括对 XML 和 JSON 的深入介绍,以及对 Ajax 应用程序架构的探讨、JavaScript 面向对象编程的技巧等。

本书对于 PHP、ASP、Java、.NET 程序员有很大的帮助,同时适合 Web 前端开发人员、Web 性能检测人员、需要掌握 Ajax 技术的测试人员、Web 架构师、学习 Web 高级技术的初级程序员以及所有的 Web 普通开发人员作为必备参考用书使用。

本书封面贴有清华大学出版社防伪标签,无标签者不得销售。

版权所有,侵权必究。侵权举报电话:010-62782989 13701121933

### 图书在版编目 (CIP) 数据

Ajax 从入门到精通/陈华编著. —北京:清华大学出版社,2008.9  
(网站开发非常之旅)

ISBN 978-7-302-17642-8

I. A… II. 陈… III. 计算机网络-程序设计 IV. TP393.09

中国版本图书馆 CIP 数据核字 (2008) 第 073249 号

责任编辑:朱英彪 张丽萍

封面设计:张 岩

版式设计:杨 洋

责任校对:焦章英

责任印制:李红英

出版发行:清华大学出版社

地 址:北京清华大学学研大厦 A 座

<http://www.tup.com.cn>

邮 编:100084

社 总 机:010-62770175

邮 购:010-62786544

投稿与读者服务:010-62776969, c-service@tup.tsinghua.edu.cn

质 量 反 馈:010-62772015, zhiliang@tup.tsinghua.edu.cn

印 刷 者:清华大学印刷厂

装 订 者:北京市密云县京文制本装订厂

经 销:全国新华书店

开 本:203×260 印 张:40 字 数:901 千字

(附光盘 1 张)

版 次:2008 年 9 月第 1 版 印 次:2008 年 9 月第 1 次印刷

印 数:1~5000

定 价:76.00 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题,请与清华大学出版社出版部联系调换。联系电话:(010)62770177 转 3103 产品编号:028894-01



# 丛书序

“不积跬步，无以至千里。”

——中国思想家荀子

“网站开发非常之旅”这套书自 2006 年开始已经陆续推出了近 10 个品种，这些品种大都受到了读者的热烈欢迎。自出版后这套书中的大多数图书已经多次印刷，销售居同类图书前列。对于计算机图书，这是一个让人振奋的结果。

根据几年来读者的反馈可以看出，他们比较一致地认为这套书定位明确，内容有特色，编写质量较好，看后能学到真正有用的东西，而且售后服务和技术支持做得非常好。从中我们不难看出这套书受欢迎的原因。

为了让读者更加全面地了解这套书，下面具体介绍这套书的定位、内容、特色和读者对象等。

## 丛书定位

根据读者的实际需求，本丛书并不追求面面俱到，而是定位于实用，并注重对基本知识点的掌握和对基本技能的提升。突出体现在以下几点：

- ☑ 每本书大体都对应着相应的工作岗位，着重让读者掌握一项技能，使他们在学完一本书以后，可以将所学应用到实际工作中去。
- ☑ 讲解概念，但并不拘泥于概念，而是侧重于对概念的灵活运用，从而让读者在实践中加深对概念的理解和对基本知识的掌握。
- ☑ 不安排纯演示性实例，那种实例通常没有任何应用价值，读者很难通过它而达到较好的应用水平。本丛书中的实例大多选自于实际开发，是作者多年开发经验的总结。读者通过对这些实例的学习，可以真正体验实际的开发过程，从而将所学应用于实践。

## 丛书书目

### 第一批：

《ASP 网络编程从入门到精通》

《ASP+SQL Server 典型网站建设案例》

《ASP.NET 网络数据库开发实例精解》

《服务器配置全攻略》

### 第二批：

《ASP.NET 2.0 网络编程从入门到精通》

《ASP.NET 2.0+SQL Server 网络应用系统  
开发案例精解》

《Dreamweaver 网页制作与色彩搭配全攻略》

《JavaScript 网页特效实例大全》

《HTML 网页设计参考手册》

## 第三批:

《JSP 网络编程从入门到精通》

《PHP 网络编程从入门到精通》

《PHP+MySQL 经典案例剖析》

《Ajax 经典案例开发大全》

《Dreamweaver 网页设计与制作完全手册》

《Dreamweaver+ASP 动态网站开发从入门到精通》

《CSS 标准网页布局开发指南》

## 第四批:

《JavaScript 从入门到精通》

《Ajax 从入门到精通》

《SQL 技术与网络数据库开发详解》

《网页制作与网站建设技术大全》

《Flex 从入门到精通》

## 丛书特色

- ☑ 本丛书按照网站开发的流程组织内容, 从最初的服务器配置, 到后台编码的实现, 再到前台网页设计等内容均有涉及, 真正实现了网站开发一条龙。学完本丛书, 读者可以全面掌握网站开发的各项技术。
- ☑ 作者均为有丰富的网络编程经验的一线开发人员, 可以确保技术的先进性、实用性和深入性。
- ☑ 内容讲解到位, 避免空洞, 每个知识点都配有实例, 读者可以上机操作, 体会其中的奥妙。
- ☑ 贯穿丰富的应用实例, 并专门设计了综合案例, 让读者真正做到学以致用, 并领会项目开发的整体思路。
- ☑ 配源代码光盘, 以帮助读者快速学习并提高实际编程能力。
- ☑ 提供必要的售后服务。丛书的每本书都有相应的技术论坛与作者 [martt0656@163.com](mailto:martt0656@163.com) (见前言), 读者若有疑问, 可以按此寻求帮助。

## 读者对象

- ☑ 网站建设及网络开发人员
- ☑ 网页设计及制作人员
- ☑ 网站维护人员
- ☑ 网页制作爱好者
- ☑ 大中专院校的学生
- ☑ 毕业设计的学生

“不积跬步, 无以至千里”, 这句话可能最能概括这套书成功的真正原因。这套书要想走得远, 就得一步一个脚印地做好各个环节的工作。从对读者需求的调研, 到确立写作思路, 到选择作者, 到后期编辑及制作, 乃至技术服务等, 都要一一落实。只有这样, 才能给这套书的继续发展奠定基础, 才能让这套书继续成为广大读者的良师益友。希望这套书能一如既往地以高质量、高品质和读者见面, 同时也希望这套书能继续受到读者的关注和青睐。



# 前 言

随着 Web 应用的不断深入,用户对 Web 应用程序提出了更高的要求。在这种情况下, Ajax 技术应运而生,并引领了一阵 Web 2.0 的风潮。Ajax 技术本身并不是新的技术,而是对已有的成熟技术的整合应用,并提出了 Web 开发的新思路。当前 Ajax 技术已经得到广泛的应用,并被用户所推崇,这也对 Web 开发工作者提出了更高的要求。掌握 Ajax 风格的 Web 应用程序开发技术成为每一个 Web 开发工作者的当务之急。

Ajax 的核心技术由 XMLHttpRequest、JavaScript、XML、HTML 及 CSS 等组成。本书假定读者已经具备一定基础的 HTML 和 CSS 技能,着重向读者介绍 JavaScript 的基础知识和高级技巧、XMLHttpRequest 对象的使用、数据的组织(XML 和 JSON)、Ajax 开发模式、常见问题及解决方案以及最流行的 Ajax 开发框架等内容,涵盖了 Ajax 技术从入门到精通需要的大部分知识。

## 本书的内容安排

本书共 5 篇正文和 3 个附录,循序渐进地讲述 Ajax 技术的各个知识点:

准备篇(第 1 章)介绍 Ajax 的概念和历史,并向读者介绍两个 Ajax 技术运用的经典案例: Google Suggest 和 Gmail,然后向读者演示一个简单的 Ajax 应用程序 Hello World,让读者对 Ajax 有一个直观的了解。

Ajax 技术构成篇(第 2~5 章)介绍 Ajax 的开发语言 JavaScript 的基本知识和常用对象、DOM 文档对象模型的概念和操作,以及 Ajax 开发所需要使用的工具。

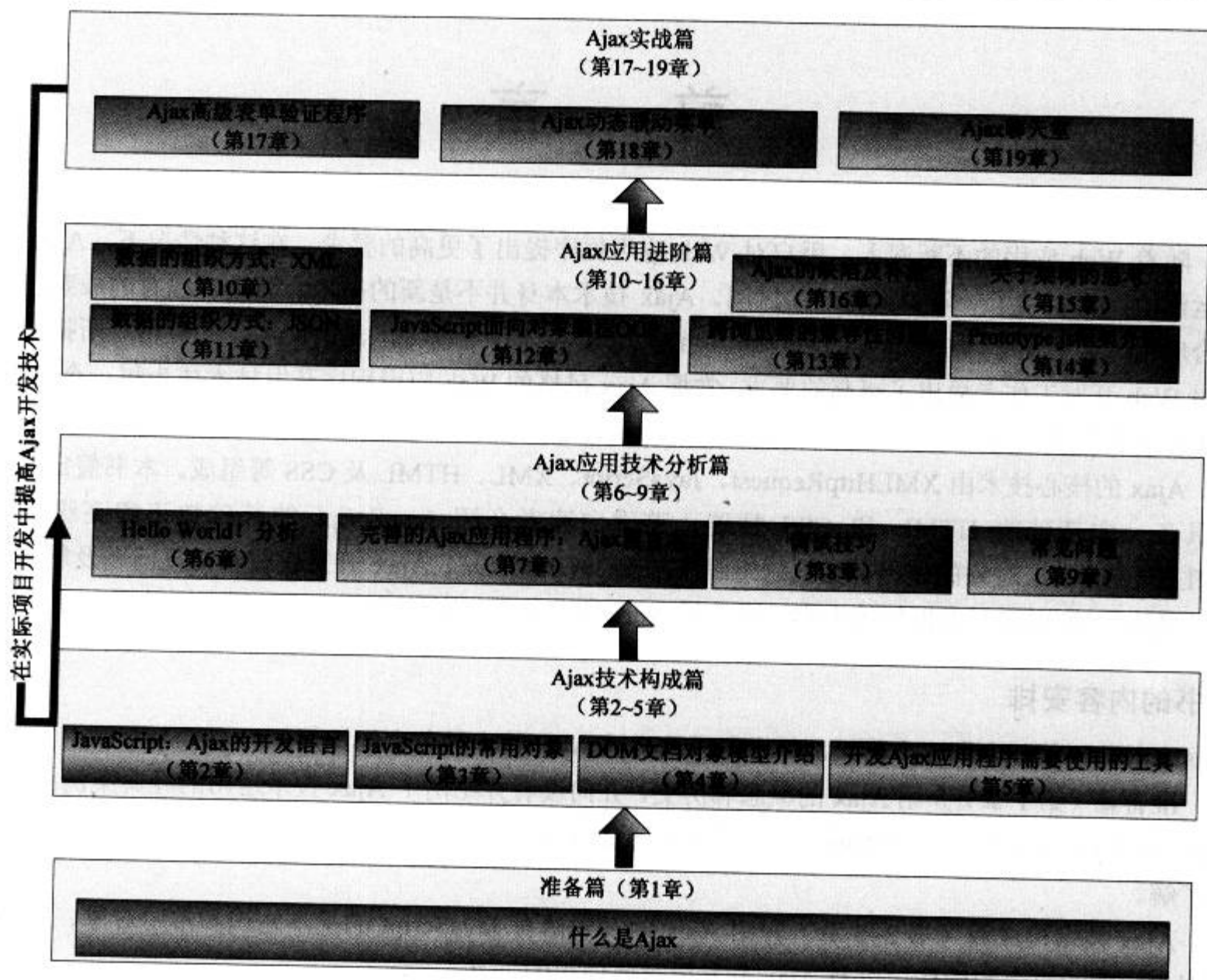
Ajax 应用技术分析篇(第 6~9 章)介绍 Ajax 技术的核心 XMLHttpRequest 对象的所有属性和方法,并实现了一个完善的 Ajax 应用程序: Ajax 留言本,然后介绍调试技巧以及开发中的常见问题。

Ajax 应用进阶篇(第 10~16 章)详细介绍 Ajax 应用程序的数据组织方式: XML 和 JSON、JavaScript 面向对象编程的技巧、浏览器兼容性问题的分析及解决、使用 JSVM 进行代码组织、使用 Prototype.js 进行开发以及关于 Web 应用程序架构的思考和 Ajax 技术的缺陷和补救办法。

Ajax 实战篇(第 17~19 章)演示如何基于 JSVM 开发高级 Ajax 表单验证程序、Ajax 联动菜单,以及基于 Prototype.js 开发一个 Ajax 聊天室。

附录(附录 A~附录 C)提供了 JavaScript 速查手册、HTML DOM 速查手册以及 Prototype 速查手册。

读者可以遵循本书的写作顺序,参考以下流程图阅读本书。



## 本书的特点

本书编者从自己学习和使用 Ajax 技术的经验出发, 不仅向读者介绍了 Ajax 技术的相关知识, 还包含了开发技巧、调试技巧、代码组织、架构等多方面的内容。本书的主要特点如下所示:

- 本书的编排采用循序渐进的方式, 适合初级、中级读者逐步掌握复杂的 Ajax 技术及其 JavaScript 类库。
- 本书采用了大量的实例来结合知识点的讲解, 以让读者更好地将书中的知识融入到实际的开发中。
- 所有实例都具有代表性和实际意义, 着重解决工作中的实际问题。
- 对于有特点的实例进行详细的解释和分析, 帮助读者理解和模拟实践。
- 对于工作中经常遇到的问题, 需要注意的关键点予以特别注释。
- 按递进关系进行案例组织, 使用新旧知识点时相互关联, 对比分析易于理解。
- 本书采用技术要点、详细介绍、运行效果等多种方式进行讲解, 系统性和可用性较强。



## 适合阅读本书的读者

- 所有 Web 前端开发人员。
- 希望了解或掌握 Ajax 技术的 Web 后端开发人员。
- 技术经理、架构师等技术决策者。
- 所有对 Ajax 和 Web 开发技术感兴趣的程序员。

本书由陈华编写，其他参与编写、资料整理和光盘制作的人员有陈杰、陈冠军、项宇峰、于咏泽、冯浩楠、刘军、刘晶晶、刘辉、刘长江、吴荣、孙海民、孙爱荣、张亚丹、张军华、李家玉、李爱芝、李静、王全、王嘉、王晓天、王永刚、石光成、纪超、胡永、贾凯、赵美青、陆壮飞和马忠超等，他们对本书的编写和出版做了大量的工作，在此一并表示诚挚感谢。

如果您在阅读本书过程中遇到问题，可以发电子邮件至 [martt0656@163.com](mailto:martt0656@163.com) 以获得帮助，我们会尽力为您解答。

编 者

# 目 录

## 第 1 篇 准备篇

第 1 章 什么是 Ajax .....	3	1.3.2 Gmail 主界面 .....	9
1.1 Ajax 概述 .....	4	1.3.3 Inbox (收件箱) 工作区域 .....	10
1.1.1 关键技术: XMLHttpRequest .....	4	1.3.4 Web Clip (网络剪辑) .....	11
1.1.2 Ajax 名词的由来 .....	4	1.3.5 邮件操作区域 .....	11
1.2 经典案例 1: 搜索关键词建议系统 (Google Suggest) .....	5	1.3.6 操作邮件 .....	12
1.3 经典案例 2: 优秀的电子邮件服务 系统 (Gmail) .....	7	1.4 第一个 Ajax 应用程序: Hello World! .....	13
1.3.1 注册 Gmail .....	7	1.5 小结 .....	14

## 第 2 篇 Ajax 技术构成篇

第 2 章 JavaScript: Ajax 的开发语言 .....	17	2.3.4 基本类型和引用类型 .....	27
2.1 JavaScript 概述 .....	18	2.4 表达式和运算符 .....	28
2.1.1 对 JavaScript 的误解 .....	18	2.4.1 表达式 .....	28
2.1.2 JavaScript 的版本 .....	19	2.4.2 算术运算符 .....	28
2.2 数据类型和值 .....	19	2.4.3 相等运算符 .....	30
2.2.1 数字 .....	20	2.4.4 关系运算符 .....	31
2.2.2 字符串 .....	20	2.4.5 赋值运算符 .....	33
2.2.3 布尔值 .....	21	2.4.6 逻辑运算符 .....	33
2.2.4 函数 .....	22	2.4.7 字符串运算符 .....	34
2.2.5 对象 .....	23	2.4.8 其他运算符 .....	34
2.2.6 数组 .....	24	2.5 语句 .....	34
2.2.7 null 值 .....	24	2.5.1 声明变量: var 语句 .....	35
2.2.8 undefined 值 .....	24	2.5.2 流程控制: if 语句 .....	35
2.3 JavaScript 的变量 .....	25	2.5.3 流程控制: else if 语句 .....	37
2.3.1 变量的类型 .....	25	2.5.4 流程控制: switch 语句 .....	37
2.3.2 变量的声明 .....	25	2.5.5 循环: while 语句 .....	39
2.3.3 变量的作用域 .....	26	2.5.6 循环: do/while 语句 .....	39
		2.5.7 循环: for 语句 .....	40



2.5.8 遍历: for/in 语句 .....	41	3.3.5 用于模式匹配的 RegExp 方法 .....	63
2.5.9 控制语句: break 语句 .....	41	3.3.6 常用正则表达式 .....	64
2.5.10 控制语句: continue 语句 .....	43	3.4 日期和时间: Date 对象 .....	64
2.5.11 函数语句: function 语句 .....	44	3.4.1 Date 对象的创建 .....	65
2.5.12 函数返回值: return 语句 .....	44	3.4.2 读取和设置日期及时间的各个 部分 .....	66
2.5.13 抛出异常: throw 语句 .....	45	3.4.3 日期和时间的换算 .....	69
2.5.14 异常处理: try/catch/finally 语句 .....	45	3.5 复杂的数学运算: Math 对象 .....	71
2.5.15 空语句 .....	46	3.5.1 小数的取整 .....	71
2.6 JavaScript 的函数 .....	46	3.5.2 得到随机数 .....	71
2.6.1 函数的定义和调用 .....	47	3.5.3 幂运算 .....	72
2.6.2 实际参数列表: arguments 属性的 使用 .....	48	3.5.4 最大值和最小值 .....	72
2.7 大小写敏感性 .....	49	3.6 操作浏览器窗口: window 对象 .....	72
2.8 JavaScript 的注释 .....	49	3.6.1 使用对话框 .....	72
2.9 JavaScript 的保留字 .....	50	3.6.2 改变窗口状态栏的信息 .....	74
第 3 章 JavaScript 的常用对象 .....	51	3.6.3 延迟执行和定时执行 .....	74
3.1 保存多个数据元素的容器: Array 对象 .....	52	3.6.4 URL 的获取和跳转 .....	76
3.1.1 数组的创建 .....	52	3.6.5 历史记录的前进和后退 .....	77
3.1.2 得到数组的长度 .....	52	3.6.6 控制窗口的大小和位置 .....	81
3.1.3 添加、删除和替换数组元素 .....	53	3.6.7 打开和关闭窗口 .....	85
3.1.4 得到数组片段 .....	54	3.6.8 获得焦点和失去焦点 .....	88
3.1.5 反转数组 .....	54	3.6.9 取得用户显示器的信息 .....	88
3.1.6 将数组转换成字符串 .....	54	3.6.10 取得用户浏览器的信息 .....	88
3.1.7 数组元素的排序 .....	55	3.7 操作 HTML 文档: document 对象 .....	89
3.2 字符串的处理: String 对象 .....	55	3.7.1 文档的输出 .....	89
3.2.1 获取字符串的长度 .....	55	3.7.2 文档的标题 .....	90
3.2.2 字符串的截取 .....	56	3.7.3 文档的图像 .....	91
3.2.3 字符串的替换 .....	56	3.7.4 文档的超链接 .....	92
3.2.4 大小写转换 .....	56	3.7.5 文档的表单 .....	93
3.2.5 将字符串转换成数组 .....	57	3.8 应用实例 .....	94
3.2.6 拼接字符串的优化方法 .....	57	3.8.1 在网页上显示自动更新的日期和 时间 .....	94
3.3 正则表达式: RegExp 对象 .....	58	3.8.2 一个简单的小游戏: Lucky Seven (幸运 7) .....	95
3.3.1 创建正则表达式 .....	58	3.9 小结 .....	99
3.3.2 正则表达式的语法规则 .....	59	第 4 章 DOM 文档对象模型介绍 .....	101
3.3.3 正则表达式的属性 .....	61	4.1 基本概念 .....	102
3.3.4 用于模式匹配的 String 方法 .....	62		

4.1.1	树形结构	102
4.1.2	结点的类型和组成	103
4.1.3	结点之间的关系	103
4.2	结点的引用	103
4.2.1	根据 id 属性引用结点	104
4.2.2	根据 name 属性引用结点	105
4.2.3	根据标签名引用结点	106
4.2.4	引用父结点	107
4.2.5	引用子结点	108
4.2.6	引用相邻的结点	110
4.3	结点的操作	112
4.3.1	创建元素结点	112
4.3.2	创建文本结点	112
4.3.3	添加结点	113
4.3.4	插入子结点	116
4.3.5	替换子结点	117
4.3.6	复制结点	119
4.3.7	删除子结点	120
4.3.8	读取结点属性	124
4.3.9	添加和修改属性结点	124
4.3.10	删除属性结点	125
4.4	控制元素的样式	125
4.4.1	获取和设置元素的 css 类	125
4.4.2	获取和设置元素样式	126
4.5	事件处理	129

4.5.1	事件模型和传播机制	129
4.5.2	注册事件处理程序	130
4.5.3	注销事件处理程序	134
4.5.4	事件对象	136
4.5.5	常用事件	139
4.6	应用实例	140
4.6.1	悬浮的广告	140
4.6.2	可拖动的层	142
4.7	小结	144

## 第 5 章 开发 Ajax 应用程序需要使用的工具

5.1	开发工具: Aptana	146
5.1.1	Aptana 的下载和安装	146
5.1.2	Aptana 的界面介绍	146
5.1.3	Aptana 的使用	148
5.1.4	Aptana 的更新	157
5.1.5	Aptana 的卸载	157
5.2	Firefox	158
5.2.1	错误控制台: Error Console	158
5.2.2	优秀的调试插件: Firebug	158
5.3	HTTP 调试工具: Fiddler	165
5.3.1	Fiddler 的下载和安装	166
5.3.2	使用 Fiddler	166
5.4	小结	167

## 第 3 篇 Ajax 应用技术分析篇

### 第 6 章 Hello World! 分析

6.1	XMLHttpRequest 对象详解	172
6.1.1	初始化请求	172
6.1.2	设置请求的 HTTP 头信息	173
6.1.3	发送请求	174
6.1.4	获取请求的当前状态	176
6.1.5	指定请求状态改变时的事件处理句柄	178
6.1.6	返回当前请求的 HTTP 状态码	178

6.1.7	从返回信息中获取指定的 HTTP 头	179
6.1.8	获取返回信息的所有 HTTP 头	180
6.1.9	取得返回的数据	181
6.1.10	取消当前请求	183
6.2	搭建基本的 Ajax 开发框架	183
6.2.1	创建 XMLHttpRequest 对象	183
6.2.2	发送请求和回调函数	184
6.2.3	一个封装好的基本 Ajax 应用程序开发框架	186

6.3 小结 .....	191	8.1.3 控制台的命令行功能 .....	234
<b>第 7 章 完善的 Ajax 应用程序: Ajax</b>		8.1.4 断点、单步执行和变量信息 .....	236
留言本 .....	193	8.1.5 在其他浏览器中使用 Firebug 的 控制台 .....	239
7.1 留言本的需求 .....	194	8.1.6 屏蔽测试代码 .....	241
7.2 留言本的基本设计 .....	194	<b>8.2 使用 Aptana 的集成调试功能 .....</b>	<b>241</b>
7.2.1 系统环境 .....	194	8.2.1 配置集成调试环境 .....	241
7.2.2 留言的数据和操作 .....	194	8.2.2 启动调试 .....	242
7.2.3 数据库设计 .....	195	8.2.3 断点、单步执行和变量信息 .....	244
7.2.4 后台功能模块 .....	195	8.2.4 使用 console.log 和 dump 输出文本 信息 .....	246
7.2.5 前台脚本功能模块 .....	195	8.2.5 使用 aptana.trace 输出调用堆栈 信息 .....	247
7.2.6 系统文件结构和文件清单 .....	196	8.2.6 使用断言 .....	248
7.3 留言本的实现 .....	196	8.2.7 屏蔽调试代码 .....	249
7.3.1 创建数据库 .....	196	<b>8.3 小结 .....</b>	<b>250</b>
7.3.2 完成前台界面: index.html .....	197	<b>第 9 章 常见问题 .....</b>	<b>251</b>
7.3.3 完成前台界面: 样式表 .....	197	9.1 编码的处理 .....	252
7.3.4 完成后台功能模块: 数据库操作 模块 .....	199	9.1.1 文件编码与声明编码 .....	252
7.3.5 完成后台功能模块: 留言本逻辑 处理模块 .....	203	9.1.2 Ajax 请求乱码 .....	253
7.3.6 完成后台功能模块: 接口模块 .....	205	9.1.3 发送数据乱码 .....	254
7.3.7 完成前台功能模块: 基本 Ajax 功能 模块 .....	205	9.2 控制缓存 .....	256
7.3.8 完成前台功能模块: 读取和发送 留言 .....	208	9.3 选择合适的请求方式 .....	257
7.3.9 整合留言本程序 .....	211	9.4 控制多个 Ajax 请求 .....	258
7.4 留言本的功能测试 .....	212	9.4.1 轮询模式 .....	258
7.5 小结 .....	214	9.4.2 事件响应模式 .....	259
<b>第 8 章 调试技巧 .....</b>	<b>215</b>	9.5 Ajax 请求的安全性 .....	260
8.1 深入解析 Firebug 的调试功能 .....	216	9.5.1 身份验证 .....	260
8.1.1 检查常规错误 .....	216	9.5.2 防范 SQL 注入 .....	260
8.1.2 完善的 log 功能 .....	219	9.5.3 防范 JavaScript 注入 .....	261

## 第 4 篇 Ajax 应用进阶篇

<b>第 10 章 数据的组织方式: XML .....</b>	<b>265</b>	10.2 XML 语法规范 .....	266
10.1 XML 概述 .....	266	10.2.1 XML 声明 .....	267
		10.2.2 根节点 .....	267



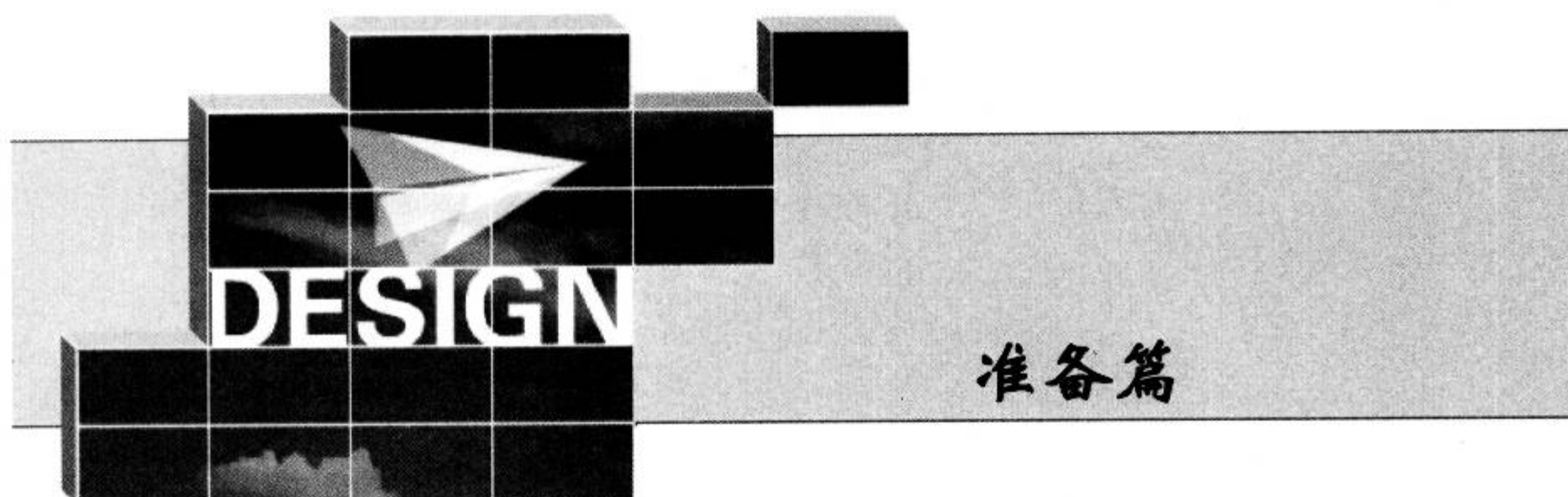
10.2.3 开始和结束标记 .....	267	10.6.6 坐标轴 .....	307
10.2.4 属性 .....	267	10.6.7 运算符 .....	308
10.2.5 合理地嵌套包含 .....	268	10.6.8 路径表达式的步语法 .....	308
10.2.6 大小写敏感性 .....	268	10.6.9 XPath 函数 .....	309
10.2.7 空白被保留 .....	268	10.7 格式化 XML 工具: XSL .....	309
10.2.8 XML 的注释 .....	268	10.7.1 基本示例 .....	310
10.2.9 元素的命名 .....	269	10.7.2 XSL 声明 .....	313
10.2.10 转义字符 .....	270	10.7.3 使用模板 .....	313
10.2.11 CDATA 部件 .....	270	10.7.4 取得数据 .....	316
10.3 XML 命名空间 .....	271	10.7.5 使用 for-each 元素代替模板 .....	316
10.4 XML Schema .....	272	10.7.6 使用 sort 元素进行排序 .....	318
10.4.1 基本示例 .....	273	10.7.7 流程控制 .....	319
10.4.2 定义元素 .....	276	10.7.8 创建元素和属性 .....	320
10.4.3 简单类型 .....	276	10.7.9 指定输出格式 .....	321
10.4.4 复合类型 .....	278	10.8 在客户端格式化 XML .....	322
10.4.5 定义属性 .....	279	10.9 跨浏览器的 XML 开发框架:	
10.4.6 默认值 .....	279	zXML .....	326
10.4.7 约束特殊值 .....	280	10.10 应用实例: Ajax 文章列表程序	
10.4.8 列表类型 .....	280	(XML) .....	341
10.4.9 联合类型 .....	281	10.10.1 确认需求 .....	341
10.4.10 匿名类型定义 .....	281	10.10.2 系统设计 .....	341
10.4.11 简单的复合类型 .....	283	10.10.3 系统实现: 创建数据库 .....	343
10.4.12 混合内容 .....	283	10.10.4 系统实现: 完成后台模块 .....	343
10.4.13 任意类型 .....	284	10.10.5 系统实现: 完成前台界面 .....	347
10.4.14 分组和引用 .....	284	10.10.6 系统实现: 完成前台模块 .....	351
10.4.15 命名冲突 .....	285	10.10.7 系统实现: 编写 xsl 样式表 .....	357
10.4.16 关联 XML 与 XML Schema .....	286	10.10.8 整合系统 .....	358
10.5 XML DOM .....	287	10.10.9 系统测试 .....	359
10.5.1 创建 XML DOM 对象 .....	287	10.11 小结 .....	363
10.5.2 加载 XML 文档 .....	287	第 11 章 数据的组织方式: JSON .....	365
10.5.3 加载 XML 片段 .....	294	11.1 JSON 的语法结构 .....	366
10.5.4 取得 XML 内容 .....	296	11.1.1 JSON 的基本结构 .....	366
10.6 强大的检索工具: XPath .....	297	11.1.2 JSON 中值的类型 .....	367
10.6.1 基本示例 .....	297	11.2 JSON 的语言支持 .....	369
10.6.2 选取节点 .....	306	11.2.1 在 JavaScript 中读取 JSON 数据 .....	369
10.6.3 谓词 .....	306	11.2.2 在 JavaScript 中输出 JSON 数据 .....	371
10.6.4 通配符 .....	307	11.2.3 在 PHP 中使用 JSON .....	378
10.6.5 使用多个路径 .....	307		



11.2.4 在 C# 中输出 JSON 数据 .....	380	第 13 章 跨浏览器的兼容性问题 .....	421
11.2.5 在 C# 中输出带类型说明的 JSON 数据 .....	384	13.1 使用 getElementById 代替 idName .....	422
11.2.6 在 C# 中读取 JSON 数据 .....	389	13.2 表单元素的引用问题 .....	423
11.2.7 更多语言支持 .....	393	13.3 访问集合对象成员的问题 .....	425
11.3 JSON 的优点和不足 .....	393	13.4 读取自定义属性的问题 .....	426
11.3.1 JSON 的优点 .....	394	13.5 常量的定义问题 .....	427
11.3.2 JSON 的不足 .....	395	13.6 input 元素的 type 属性读写问题 .....	428
11.4 将 XML 转换为 JSON .....	395	13.7 模态窗口的问题 .....	429
11.5 小结 .....	399	13.8 frame 的操作问题 .....	431
第 12 章 JavaScript 面向对象编程 (OOP) .....	401	13.9 innerText 的问题 .....	433
12.1 JavaScript 中的类 .....	402	13.10 对父元素的引用问题 .....	436
12.1.1 创建类 .....	402	13.11 getElementsByName 的问题 .....	436
12.1.2 类的属性 .....	402	13.12 outerText 的问题 .....	438
12.1.3 类的方法 .....	403	13.13 outerHTML 的问题 .....	440
12.1.4 公有属性和私有属性 .....	403	13.14 小结 .....	442
12.1.5 公有方法和私有方法 .....	405	第 14 章 Prototype.js 框架介绍 .....	443
12.1.6 静态属性和静态方法 .....	406	14.1 Prototype.js 常用方法介绍 .....	444
12.1.7 原型对象 prototype .....	406	14.1.1 使用 \$ 方法代替 document .getElementById .....	444
12.2 JavaScript 中的继承 .....	407	14.1.2 使用 \$\$ 方法获得元素引用 .....	444
12.2.1 对象冒充 .....	407	14.1.3 根据 css 类名取得元素集合 .....	444
12.2.2 使用对象冒充实现继承 .....	408	14.1.4 使用 Try.these 尝试运行多个 函数 .....	446
12.2.3 继承原型对象中的属性和方法 .....	409	14.1.5 使用 \$F 方法来获得表单元素 的值 .....	446
12.2.4 封装继承方法 .....	410	14.1.6 数组迭代的简化方法 .....	447
12.3 更多技巧 .....	411	14.1.7 设置元素的样式 .....	447
12.3.1 属性的封装 .....	411	14.1.8 切换元素的隐现状态 .....	448
12.3.2 实现多态 .....	412	14.1.9 序列化表单值 .....	449
12.3.3 命名空间 .....	413	14.1.10 转换 HTML 标签 .....	451
12.3.4 实现短类名 .....	414	14.2 基于 Prototype.js 的类和继承 .....	452
12.4 使用 JSVM 进行代码组织 .....	416	14.2.1 使用 Class.create() 创建类 .....	452
12.4.1 下载和配置 JSVM .....	416	14.2.2 Prototype.js 中的继承 .....	453
12.4.2 路径和文件名的约定 .....	417	14.3 Prototype.js 中的事件处理 .....	454
12.4.3 编写类 .....	417	14.3.1 注册事件处理函数 .....	454
12.4.4 类的按需加载 .....	418	14.3.2 注销事件处理函数 .....	455
12.4.5 在类中引用其他类 .....	419		
12.5 小结 .....	420		

14.4	Prototype.js 的 Ajax 功能 .....	456	15.2.3	融入式 Web 应用程序 .....	470
14.4.1	Ajax.Request 方法详解 .....	457	15.3	Ajax 为 REST 带来新的契机 .....	470
14.4.2	用 Ajax.Updater 更新界面 .....	459	15.3.1	缓存 Ajax 程序引擎 .....	471
14.4.3	用 Ajax.PeriodicalUpdater 定时更新 界面 .....	460	15.3.2	缓存 Ajax 数据 .....	471
14.4.4	使用 Form.request 方法无刷新提交 表单 .....	460	15.4	小结 .....	471
14.5	基于 Prototype.js 的留言本程序 .....	462	第 16 章	Ajax 的缺陷及补救 .....	473
14.6	小结 .....	465	16.1	搜索引擎的收录问题 .....	474
第 15 章	关于架构的思考 .....	467	16.1.1	问题产生的原因 .....	474
15.1	REST 架构模式 .....	468	16.1.2	解决办法 .....	474
15.2	Web 应用程序的发展史 .....	468	16.2	前进和后退的问题 .....	483
15.2.1	提供静态文档的 Web 站点 .....	469	16.2.1	问题产生的原因 .....	483
15.2.2	早期的动态 Web 应用程序 .....	469	16.2.2	Firefox 下的解决办法 .....	483
			16.2.3	在 IE 下的解决办法 .....	489
			16.3	小结 .....	495
<b>第 5 篇 Ajax 实战篇</b>					
第 17 章	Ajax 高级表单验证程序 .....	499	19.2.1	系统结构 .....	536
17.1	确定需求 .....	500	19.2.2	实体及数据库设计 .....	537
17.2	基本设计 .....	500	19.2.3	后台功能模块 .....	537
17.3	代码实现 .....	501	19.2.4	请求控制器 .....	537
17.3.1	实现 EventManager 类 .....	501	19.2.5	前台界面 .....	537
17.3.2	实现 Request 类 .....	503	19.2.6	前台功能模块 .....	538
17.3.3	实现 Validator 类 .....	506	19.3	实例代码 .....	539
17.3.4	实现 FormItemValidator 类 .....	510	19.3.1	建立数据库 .....	539
17.3.5	实现 FormValidator 类 .....	514	19.3.2	实现后台功能模块 .....	540
17.4	测试 .....	517	19.3.3	请求控制器 .....	547
第 18 章	Ajax 动态联动菜单 .....	523	19.3.4	界面 HTML 和 CSS 代码 .....	548
18.1	确定需求 .....	524	19.3.5	前台功能模块 .....	551
18.2	基本设计 .....	524	19.3.6	加入表情图标 .....	564
18.3	实例代码 .....	524	19.3.7	整合程序 .....	565
18.4	测试 .....	528	19.4	测试 .....	567
第 19 章	Ajax 聊天室 .....	535	附录 A	JavaScript 速查手册 .....	571
19.1	确定需求 .....	536	附录 B	HTML DOM 速查手册 .....	589
19.2	基本设计 .....	536	附录 C	Prototype 速查手册 .....	608

# 第 1 篇



## 第1章 什么是Ajax

# 第1章

设计

DESIGN

第1章 设计



# 第 1 章

## 什么是 Ajax

- » Ajax 概述
- » 经典案例 1: 搜索关键词建议系统 (Google Suggest)
- » 经典案例 2: 优秀的电子邮件服务系统 (Gmail)
- » 第一个 Ajax 应用程序: Hello World!
- » 小结

当前 Web 开发领域最新的时髦术语 Ajax, 到底代表着什么含义? 它从何发展而来? 在当今 Web 开发领域中扮演什么样的角色, 有着什么样的影响? 未来又将去往何处? 本章将带领读者走进 Ajax 的世界。

本章首先在“Ajax 概述”中对 Ajax 作基本的介绍, 讲解其含义和发展历史。然后介绍两个经典的 Ajax 应用案例: Google Suggest 和 Gmail。最后, 笔者将和读者一起来创建第一个 Ajax 应用程序“Hello World!”。

## 1.1 Ajax 概述

Ajax 全称是 Asynchronous JavaScript and XML，即异步 JavaScript 和 XML，这也概括了 Ajax 应用程序的基本特点：由 JavaScript 编写、程序异步执行、用 XML 来封装和传输数据。

Ajax 并不是什么新技术，而是将各种已经存在的成熟技术，采用一种崭新的开发模式重新揉合在一起，使得古老的 B/S 方式的 Web 开发焕发了新的活力，迎来了第二个春天。具体来说，Ajax 主要由以下技术组成：

- ☐ HTML
- ☐ CSS
- ☐ DOM (HTML DOM 和 XML DOM)
- ☐ JavaScript
- ☐ XML
- ☐ XMLHttpRequest

### 1.1.1 关键技术：XMLHttpRequest

可以看到，除了 XMLHttpRequest 技术，其他技术都是目前得到理解和广泛使用的基于 Web 标准的技术。而 XMLHttpRequest 技术正是整个 Ajax 技术体系的核心，是 Ajax 得以区分传统 Web 应用程序的标志。

XMLHttpRequest 最初叫做 XMLHTTP，由微软（Microsoft）于 1999 年在 IE5.0 中率先推出。XMLHttpRequest 为运行于浏览器中的 JavaScript 脚本提供了一种在页面内与服务器通信的手段，这样就使得页面内的 JavaScript 可以在不刷新页面的前提下，向服务器提交数据，及从服务器获取数据。

XMLHttpRequest 的出现打破了传统 Web 应用程序的固有模式，即 Web 应用程序再也不一定是由一组连续的页面组成。在传统的 Web 应用程序中，用户大部分交互性的操作，都需要等待页面的切换并刷新整个页面，而在页面刷新的间隙中，用户只能“傻傻地”等待，什么也做不了。XMLHttpRequest 的出现改变了这一笨拙的开发模式，使得用户可以不切换页面就能提交数据到服务器，并能局部更新页面以显示更新后的数据，为用户提供了更好的用户体验。

正是由于 XMLHttpRequest 的优越性，使得其他厂商的浏览器纷纷开始支持 XMLHttpRequest。几乎所有的主流浏览器，例如 IE、Firefox、Netscape、Opera 以及苹果（Apple）公司推出的 Safari，全部都支持这个技术。如今，XMLHttpRequest 已经成为事实标准。

### 1.1.2 Ajax 名词的由来

虽然 XMLHttpRequest 的出现已经有近十个年头，但是 Ajax 这个概念却是直到 2005 年 2 月才被 Adaptive Path 公司负责用户体验策略的主管 Jesse James Garrett 正式提出。他在 2005 年 2 月 18 日发表了一篇题为《Ajax: A New Approach to Web Applications》（Ajax：一个 Web 应用的新途径）的论文，

明确地将 Ajax 定义为 Asynchronous JavaScript and XML, 这就是 Ajax 这个名词的由来。

而 Ajax 技术应用的先驱和主要推动者则是著名的搜索引擎公司: Google。Google 在 2005 年推出的 Gmail, 充分展示了 Ajax 的魅力所在, 吸引了无数用户。从此, 在全世界范围内逐渐刮起了一阵 Ajax 的旋风。

Google 公司在之后相继推出的新版 Google Groups、Google Suggest 和 Google Maps 中, 都给予了 Ajax 充分的表现空间。Google Suggest 能够根据用户的输入立刻显示关键词的建议和相关条目的搜索结果数量。Google Maps 给用户呈现出可以任意放大、缩小和移动地图的超凡体验。而后, Google 公司更是基于 Ajax 构建出了 Gmail Chat、电子表格、日历等接近传统桌面应用程序体验的 Web 应用程序, 将 Ajax 和 Web 的发展又往前推进了一大步。

与此同时, 其他厂商也不甘示弱。微软在推出的 MSN Space 和 MSN Virtual Earth 服务中都大量采用了 Ajax 技术, 并且还提供了一个名为 Atlas 的基于 ASP.NET 2.0 的 Ajax 开发框架(最终被命名为 ASP.NET Ajax)。另一个广为人知的 Ajax 应用, 则是被雅虎收购的 Flickr.com 提供的一种基于 Ajax 技术的图片管理服务。虽然其功能并不是十分强大, 但是用户可以非常方便地在浏览器中对图片的标题、描述和标签等内容进行修改, 这些改变在用户结束输入的同时就被保存了下来, 并留在了网页上。

随着 Ajax 应用的逐渐广泛和深入, Ajax 应用程序的开发效率成为了一个重要的问题。在这时, 先后涌现出了一大批优秀的 Ajax 应用框架: 从底层开发框架 Prototype.js 到 UI 框架 YUI/YUI-ext, 再到代码组织框架 JSVM 等; 从早期单纯的客户端框架, 到后期与服务端整合的开发框架, 如 ASP.NET Ajax、GWT 等, 都标志着 Ajax 技术正在逐渐走向成熟和辉煌。

## 1.2 经典案例 1: 搜索关键词建议系统 (Google Suggest)

1.1 节向读者介绍了 Ajax 的基本概念和发展历史。为了让读者对 Ajax 技术有更深刻和直观的了解, 本节和 1.3 节特别准备了两个经典的 Ajax 应用案例给读者。这两个案例来自 1.1 节中提到的 Ajax 技术主要的推动者: Google 公司, 其分别是搜索关键词建议系统 Google Suggest 和邮件服务系统 Gmail。

Google Suggest 是 Google 公司推出的一项搜索关键词建议系统。在中文版的 Google 网站, 是默认开启这项功能的, 读者可以直接访问 <http://www.google.cn> 来体验 Google Suggest 的功能。而在英文版的 Google 网站中, Google Suggest 默认被关闭了, 所以需要访问专门的页面 <http://www.google.com/webhp?complete=1&hl=en>。中文版的 Google 搜索界面如图 1.1 所示。



图 1.1 Google 中文版搜索界面

Google Suggest 的概念其实非常简单, 就是当用户在搜索框输入关键词时, 程序自动根据用户输入的一部分关键词来检索相关建议词及其搜索结果的数量, 并以下拉列表框的样式将这些建议词显示在输入框的下方, 供用户选择。

如图 1.2 所示, 在搜索框中输入“AJAX”后, 搜索框下方出现了一个下拉列表框, Google Suggest



自动选择了 10 条建议词，并显示在这个列表框中。当用户改变输入框中的关键词时，下拉列表框中的建议词也会立即随之发生相应的改变。如图 1.3 所示，在搜索框中继续输入“框架”后，下面的下拉列表框的内容马上切换成了与“AJAX 框架”相关的建议词内容。

AJAX	
ajax教程	961,000 结果
ajax技术	4,260,000 结果
ajax基础教程	615,000 结果
ajax框架	398,000 结果
ajax.net	852,000 结果
ajax 框架	404,000 结果
ajax in action	7,560,000 结果
ajax 教程	958,000 结果
ajaxpro	89,100 结果
ajax4jsf	277,000 结果
关闭	

图 1.2 Google Suggest 效果图 1

AJAX框架	
ajax框架比较	268,000 结果
ajax框架汇总	57,700 结果
ajax框架下载	358,000 结果
ajax框架 java	372,000 结果
ajax框架推荐	328,000 结果
ajax框架 .net	415,000 结果
ajax框架 下载	360,000 结果
ajax框架 dwr	39,200 结果
ajax框架 比较	276,000 结果
ajax框架选择	248,000 结果
关闭	

图 1.3 Google Suggest 效果图 2

据分析，这些建议词是按照搜索次数由大到小排列的，而每次显示的建议词不会超过 10 个。当用户单击选择其中一个建议词，或者用键盘上的方向键选择一个建议词并按确认键（Enter 键）后，页面就会跳转而显示相应的搜索结果，效果如图 1.4 和图 1.5 所示。

ajax框架	
ajax框架比较	268,000 结果
ajax框架汇总	57,700 结果
ajax框架下载	358,000 结果
ajax框架 java	372,000 结果
ajax框架推荐	328,000 结果
ajax框架 .net	415,000 结果
ajax框架 下载	360,000 结果
ajax框架 dwr	39,200 结果
ajax框架 比较	276,000 结果
ajax框架选择	248,000 结果
关闭	

图 1.4 选择建议词

Google 网页 图片 资讯 地图 更多

ajax框架 比较 Google 搜索

所有网页 中文网页 简体中文网页 中国的网页

网页 约有 184,000 个结果

**三种Ajax框架使用比较-理论, 实践, 总结-博客园**  
 三种Ajax框架使用比较, 随着Web 2.0 被越来越多的人所了解和接受, 做为其重要的组成部分, Ajax技术也正被越来越多的人所应用。然而其复杂的开发过程和低下的开发效率, 使像我这样没有多少传统Web开发经验的程序员望而却步。幸好, 各种各样的Ajax开发框架的 ...  
[www.cnblogs.com/hjfl1223/archive/2006/03/02/341627.html](http://www.cnblogs.com/hjfl1223/archive/2006/03/02/341627.html) - 62k - 类似网页

**我的一些看法: 关于AJAX框架的比较-老赵点滴-博客园**  
 禾口王[匿名] 是啊, 关键是“合适”。不过只用某一种“AJAX特性”的应用的确比较少, 而且说实话ASP.NET AJAX的与别家的“劣势”其实也不明显。所以我对于选择ASP.NET下的AJAX框架的意见是“用ASP.NET AJAX总没错, 用别的要斟酌。”...  
[www.cnblogs.com/JeffreyZhao/archive/2006/11/27/about\\_ajax\\_framework\\_comparison.html](http://www.cnblogs.com/JeffreyZhao/archive/2006/11/27/about_ajax_framework_comparison.html) - 119k - 类似网页  
[\[ www.cnblogs.com站内的其它相关链接 \]](#)

**AJAX框架简介 51CTO.COM 中国最大的网络技术网站**  
 关键词: AJAX 阅读提示: DWR和Buffalo都是Web Remoting框架, 区别在于DWR使用自定义的简单文本协议, 而Buffalo ... 因此Buffalo解析大数据量可能会比较慢, 然而可以适用于多种服务器端和客户端, 并且burlap协议的完整性和支持的数据类型更加丰富 ...  
[tech.51cto.com/art/200603/23958.htm](http://tech.51cto.com/art/200603/23958.htm) - 54k - 类似网页

**AJAX技术框架及开发工具- CNBRUCE'S BLOG(布鲁斯狼)**  
 其中关于DWR和Buffalo之间的比较, 它们都是Web Remoting框架, 区别在于: DWR使用自定义的简单文本协议, 而Buffalo使用burlap协议。... 2006年9月据Ajaxian.com调查主流的AJAX框架结果显示Prototype是最受欢迎的AJAX框架, 比例为43% ...  
[www.cnbruce.com/blog/showlog.asp?log\\_id=989](http://www.cnbruce.com/blog/showlog.asp?log_id=989) - 52k - 类似网页

图 1.5 搜索结果

通过上面的演示，读者可以发现，在建议词显示和更新的过程中，整个页面并没有任何刷新的行为发生，这些建议词好像一开始就写在页面中一样，但显然这是不可能的。那么，这些建议词的数据是如何获取，并更新显示在页面上的呢？

(1) 当用户输入关键词时，自动触发 Google Suggest 的监听程序。

(2) 程序获取用户输入的内容，然后以 Ajax 的方式，在页面内部制造 HTTP 请求，将输入内容



发送给 Google 的数据服务器。

(3) 数据服务器接收到请求后立即按照用户输入的内容进行检索和组织数据, 然后返回给驻留在页面内部的 Google Suggest 程序。

(4) 程序接收到数据服务器返回的数据后, 立即对数据进行分析 and 再组织, 然后将建议词显示在页面上供用户选择。

在用户输入关键词后极短的时间内, Google Suggest 就已经完成了一次与服务器的交互, 在用户毫无察觉的情况下, 就已经获取了服务器即时返回的数据来供用户进一步操作。整个流程如图 1.6 所示。

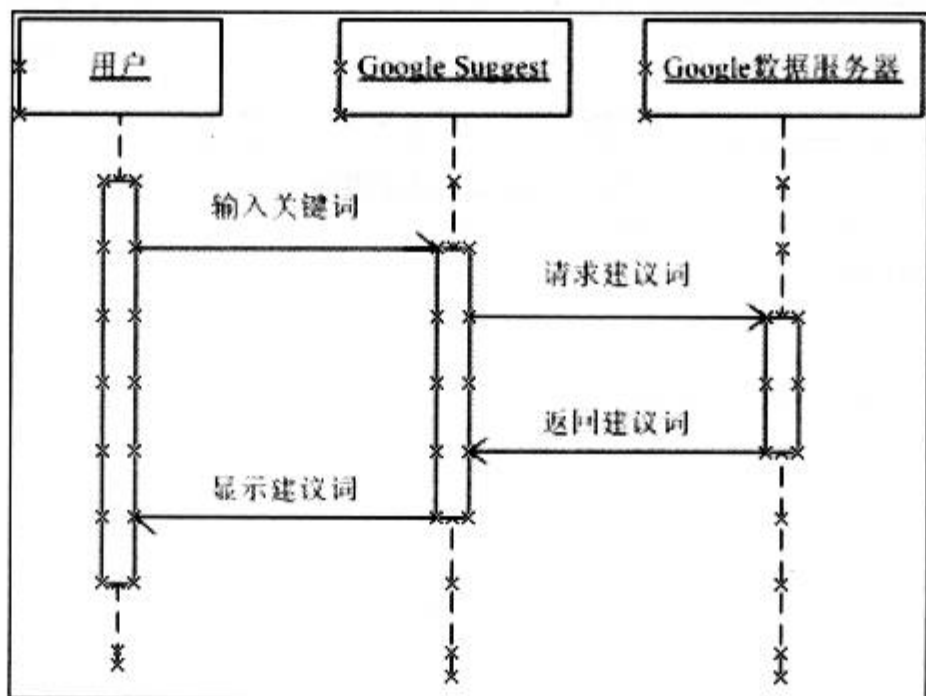


图 1.6 Google Suggest 序列图

对 Google Suggest 的介绍就到这里, 1.3 节将向读者介绍优秀的电子邮件服务系统: Gmail。

## 1.3 经典案例 2: 优秀的电子邮件服务系统 (Gmail)

1.2 节向读者介绍了一个经典的 Ajax 案例: Google Suggest。本节将会向读者介绍一个更为庞大和复杂的经典 Ajax 案例: Google 公司出品的优秀电子邮件服务系统 Gmail。

### 1.3.1 注册 Gmail

注册 Gmail 和普通的邮箱几乎没有区别, 步骤如下:

- (1) 使用浏览器访问 <http://www.gmail.com>, 进入 Gmail 的登录界面, 如图 1.7 所示。
- (2) 单击屏幕右下角的 Sign up for Gmail 超链接, 可以建立新的 Gmail 账号。注册界面如图 1.8 所示。
- (3) 在填写登录名时, 可以使用 “check availability!” 按钮来检测所填写的登录名是否可用, 检测结果将无刷新的显示在文本框下面, 这也是 Ajax 技术的典型应用。效果如图 1.9 所示。

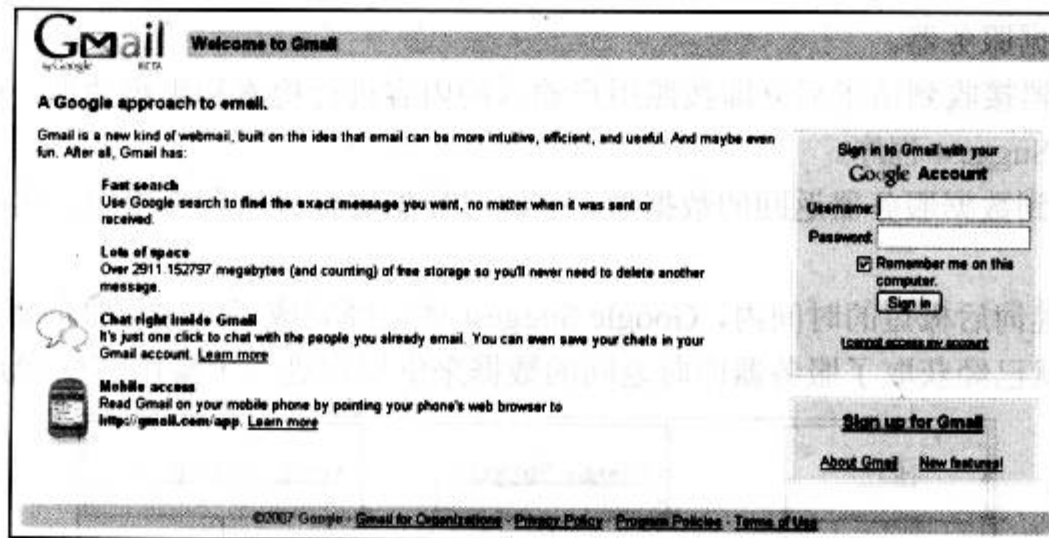


图 1.7 Gmail 登录界面

图 1.8 Gmail 注册界面

图 1.9 检查登录名是否可用

在注册完成后，就可以登录使用 Gmail 了。

### 1.3.2 Gmail 主界面

登录后, Gmail 的主界面如图 1.10 所示。

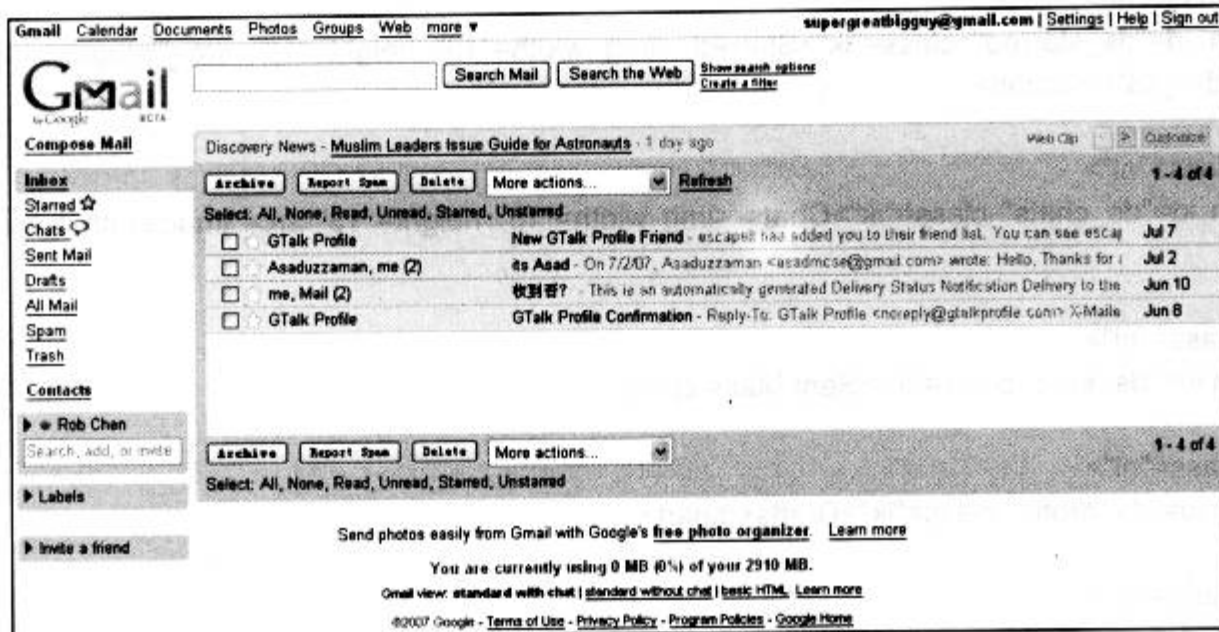


图 1.10 Gmail 主界面

Gmail 的主界面主要由页面左边的导航区域和右边的工作区域两部分组成。整个界面简洁明快,没有多余的装饰性图片和其他的广告,在提供用户良好的视觉体验的同时,也减少了页面加载所需要的时间。

先看导航栏。Inbox 按钮被一个蓝色的半圆角矩形的色块所包围,从而标明当前所处的位置是 Inbox。值得注意的是,这些导航按钮并没有用到 a 标签,而是使用 span 标签,其 HTML 代码如下所示。

```
<div id="nds" style="padding-top: 9px;">
  <table cellpadding="0" cellspacing="0" border="0" class="cv">
    <tbody>
      <tr>
        <td>
          <b class="rnd">
            <b class="rnd1"/>
            <b class="rnd2"/>
          </b>
        </td>
      </tr>
      <tr class="an">
        <td>
          <b style="margin-left: 8px;" id="ds_inbox" class="lk">Inbox</b>
        </td>
      </tr>
      <tr>
        <td>
          <b class="rnd">
            <b class="rnd2"/>
            <b class="rnd1"/>
          </b>
        </td>
      </tr>
    </tbody>
  </table>
</div>
```

```

        </b>
      </td>
    </tr>
  </tbody>
</table>
<div class="nl">
  <span id="ds_starred" class="lk">Starred </span>
</div>
<div class="nl">
  <span id="ds_chats" class="lk">Chats 
</span>
</div>
<div class="nl">
  <span id="ds_sent" class="lk">Sent Mail</span>
</div>
<div class="nl">
  <span id="ds_drafts" class="lk">Drafts</span>
</div>
<div class="nl">
  <span id="ds_all" class="lk">All Mail</span>
</div>
<div class="nl">
  <span id="ds_spam" class="lk">Spam</span>
</div>
<div class="nl">
  <span id="ds_trash" class="lk">Trash</span>
</div>
</div>

```

在 Ajax 应用程序中,经常需要设置一些按钮供用户操作,而又不需要页面跳转,所以会使用更简单的、没有默认行为和样式的 `span` 标签来替代 `a` 标签。

### 1.3.3 Inbox (收件箱) 工作区域

Inbox (收件箱) 的工作区域界面如图 1.11 所示。



图 1.11 Inbox 的工作区域



### 1.3.4 Web Clip (网络剪辑)

首先来看第一行内容，这是一个 Web Clip，即网络剪辑，相当于一个 RSS 阅读器的功能。Gmail 允许用户保存多个 RSS 源的地址到 Gmail 中，当用户访问 Gmail 时，Gmail 就会自动读取 RSS 的内容后显示在页面上，每次只显示一条记录。用户可以单击右边的向前和向后按钮来阅读其他的记录，如图 1.12 所示。

图 1.12 Web Clip 翻页按钮

Gmail 在页面加载时，会读取最新的几条 Web Clip 记录，保存在浏览器的缓存中。当用户单击 Web Clip 翻页按钮时，如果请求的数据在缓存中存在，则直接读取缓存并将新的记录更新显示在页面上。如果请求的数据在缓存中不存在时，就会发起 Ajax 请求到后台的服务器读取新的若干条记录，保存在浏览器缓存中，然后更新页面上的记录。用户感觉页面上好像保存了所有的 Web Clip 记录，而实际上是分段读取的。

### 1.3.5 邮件操作区域

WebClip 下面的部分就是邮件的操作区域。最近收到的邮件会以列表的形式显示在工作区中，如图 1.13 所示。

<input type="checkbox"/>	GTalk Profile	New GTalk Profile Friend - escapell has added you to their friend list. You can see es	Jul 7
<input type="checkbox"/>	Asaduzzaman, me (2)	its Asad - On 7/2/07, Asaduzzaman <asadmcse@gmail.com> wrote: Hello, Thanks f	Jul 2
<input type="checkbox"/>	me, Mail (2)	收到否? - This is an automatically generated Delivery Status Notification Delivery to 1	Jun 10

图 1.13 Gmail 邮件列表

每条邮件记录都由以下 5 个部分组成：复选框、星形图标、邮件标题、邮件具体内容片段和邮件接收日期。复选框让用户可以选择一条或者多条邮件记录来进行操作。而星形图标则可以让用户自己来标注邮件的重要性。单击空心的星形图标，图标切换成黄色实心的五角星，这时邮件被标注为重要。单击已经被标注的星形图标，则又切换成普通状态，如图 1.14 所示。

<input type="checkbox"/>	GTalk Profile	New GTalk Profile Friend - escapell has added you to their friend list. You can see es	Jul 7
<input checked="" type="checkbox"/>	☆ Asaduzzaman, me (2)	its Asad - On 7/2/07, Asaduzzaman <asadmcse@gmail.com> wrote: Hello, Thanks f	Jul 2
<input type="checkbox"/>	me, Mail (2)	收到否? - This is an automatically generated Delivery Status Notification Delivery to 1	Jun 10

图 1.14 星形标注

当用户切换标注状态以后，Gmail 会立即发送一个 Ajax 请求到后台服务器，以保存用户的操作结果。这样，当用户刷新页面，或者退出系统后再重新登录，邮件的标注状态仍然显示为用户操作后的样式。

Gmail 为用户提供了 Archive (存档)、Report Spam (报告垃圾邮件)、Delete (删除) 3 个快捷操作按钮。同时也提供了 6 个分组选择按钮，分别是：All (选择所有邮件)、None (取消所有选择)、Read (选择所有已读邮件)、Unread (选择所有未读邮件)、Starred (选择所有被星形标注的邮件)、Unstarred (选择所有未被星形标注的邮件)。除此之外，Gmail 还提供了一个下拉列表框来提供更多的操作选择以及一个刷新按钮，如图 1.15 所示。

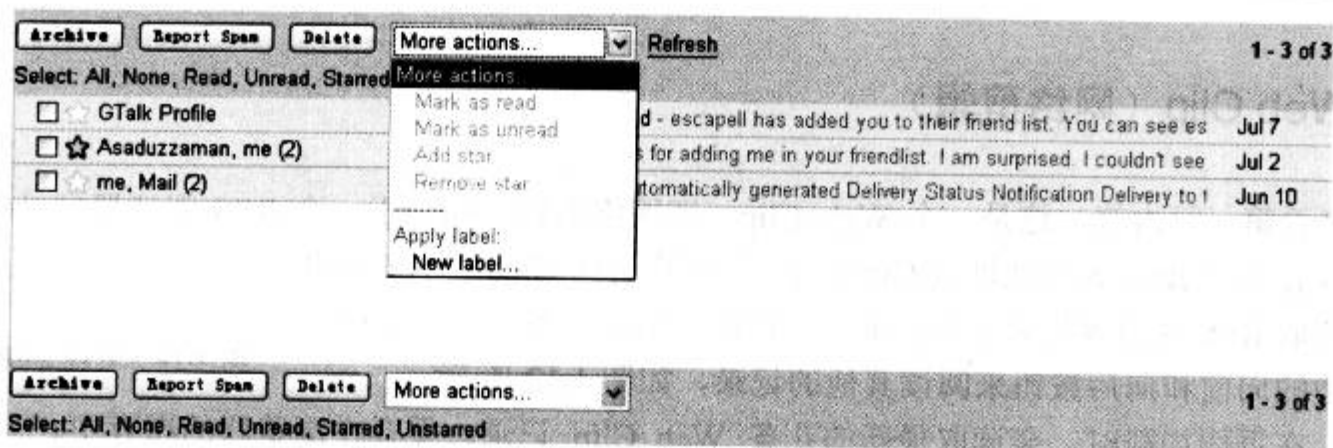


图 1.15 更多操作

### 1.3.6 操作邮件

当选择一封邮件时，这条记录的背景色会发生变化，从而标识已经被选中，如图 1.16 所示。

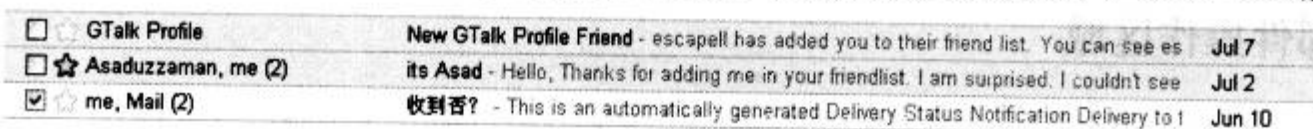


图 1.16 选择邮件

单击 Delete（删除）按钮，删除邮件，这时 Gmail 发送一个 Ajax 请求到后台服务器以删除数据库中相应的记录，同时在页面右上角显示一个表示当前请求状态的进度框。当后台服务器处理完毕后，请求返回，进度框隐藏，在工作区上方添加醒目的文字提示用户邮件删除成功，同时无刷新的更新界面，删除刚才选中的邮件记录，此时整个操作就完成了。效果如图 1.17 和图 1.18 所示。

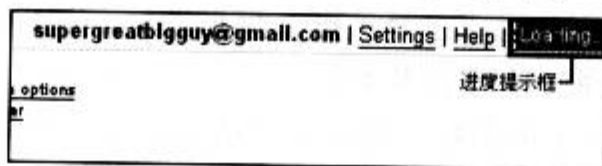


图 1.17 显示请求状态

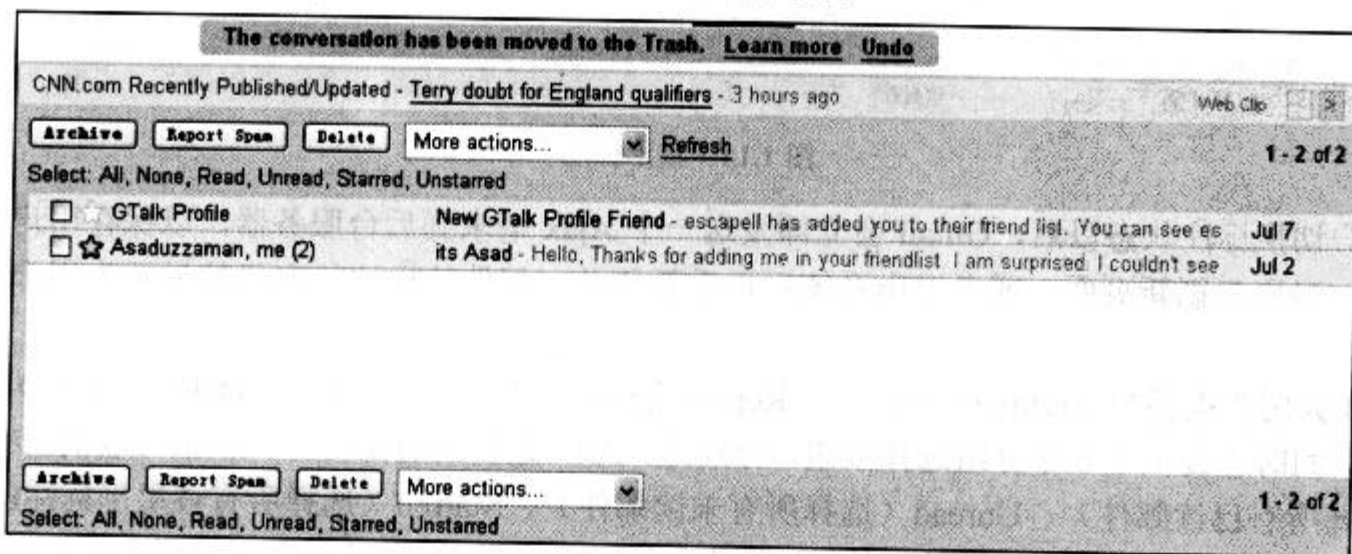


图 1.18 删除成功

Gmail 所有的操作都是类似的风格，切换界面以及针对邮件的操作，都是用户做出相应操作后，会

显示一个进度框来提示当前操作正在进行，操作完成后进度框消失，并出现醒目的文字提示，接着无刷新的更新界面。这里不再赘述。

Gmail 这一模式后来也被广泛地借鉴，特别是进度框的显示和隐藏。因为在 Ajax 应用程序中，用户的一些操作并没有发生页面跳转，数据交互在页面内部进行，而在交互完成前页面不会有任何变化，一旦请求所用的时间过长，用户可能会误认为操作没有被正常处理转而进行其他的误操作或者直接放弃操作离开页面。所以在请求过程中设置一个进度框来提示用户当前操作正在进行中，就很有必要了。

相信读者此时已经对 Ajax 应用程序有了直观的认识，Gmail 就介绍到这里。更多的细节，读者可以自己去体验，从而更深入地感受 Ajax 的魅力。

## 1.4 第一个 Ajax 应用程序：Hello World!

1.2 节和 1.3 节向读者介绍了两个 Ajax 应用的经典案例：Google Suggest 和 Gmail。本节将向读者展示本书第一个 Ajax 程序实例，正如同很多程序设计教程的实例都从 Hello World 开始一样，本书的第一个例子名字就叫“Hello World!”。

“Hello World!”的流程很简单，当页面打开时，页面的脚本程序向后台程序发送 Ajax 请求，从而获得后台程序输出的问候语：Hello World!，然后以对话框的形式显示出来。前台页面 HelloWorld.html 代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Hello World</title>
<script type="text/javascript">
    try
    {
        var xmlhttp = new XMLHttpRequest();
    }
    catch(e)
    {
        var xmlhttp = new ActiveXObject('Microsoft.XMLHTTP');
    }
    xmlhttp.open('GET','hello_world.php',true);
    xmlhttp.onreadystatechange = function()
    {
        if(xmlhttp.readyState == 4 && xmlhttp.status == 200)
        {
            alert(xmlhttp.responseText);
        }
    }
    xmlhttp.send();
</script>
```

```
</head>
```

```
<body>
```

```
</body>
```

```
</html>
```

后台 `hello_world.php` 代码如下所示。

```
<?
```

```
    echo 'Hello World';
```

```
?>
```

在浏览器中访问 `HelloWorld.html`，运行结果如图 1.19 所示。

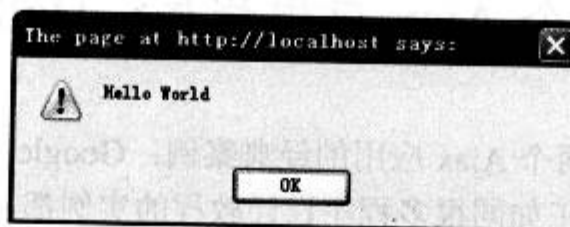


图 1.19 Hello World! 运行结果

## 1.5 小 结

本章主要是让读者对 Ajax 技术有一个初步的认识。通过 1.1 节 Ajax 概述，读者可以了解到 Ajax 技术实际上是由几种成熟的技术所组成的一个综合性的应用，通过 `XMLHttpRequest` 对象来发送异步请求，从而创造了一种新的 Web 开发模式。接着向读者介绍了两个经典的 Ajax 案例：Google Suggest 和 Gmail，让读者认识了现实中的 Ajax 应用程序到底是什么模样，了解了广泛应用的 Ajax 程序的一些基本模式。最后一节向读者展示了一个基本的 Ajax 应用程序的代码。在第 2 章中，将会辅以实例着重向读者介绍 Ajax 的主要开发语言：JavaScript。



# 第 2 篇



## Ajax 技术构成篇

第2章 JavaScript: Ajax的开发语言

第3章 JavaScript的常用对象

第4章 DOM文档对象模型介绍

第5章 开发Ajax应用程序需要使用的工具

# 第 2 章

Ajax 技术概述

- 第 2 章 Ajax 技术概述
- 第 3 章 JavaScript 常用对象
- 第 4 章 DOM 文档对象模型介绍
- 第 5 章 开发 Ajax 应用所需使用的工具

# 第 2 章

## JavaScript: Ajax 的开发语言

- » JavaScript 概述
- » 数据类型和值
- » JavaScript 的变量
- » 表达式和运算符
- » 语句
- » JavaScript 的函数
- » 大小写敏感性
- » JavaScript 的注释
- » JavaScript 的保留字

通过第1章的学习，读者已经了解到 JavaScript 是 Ajax 技术的主要组成部分。Ajax 应用程序的客户端代码都依靠 JavaScript 来实现，所以，掌握 JavaScript 是学习 Ajax 技术的必要条件。本章将详细介绍 JavaScript 的数据类型和值、变量、表达式、运算符、语句和函数等基本知识，并辅以实例加以说明。



## 2.1 JavaScript 概述

JavaScript 最初叫作 LiveScript，是网景（Netscape）公司在设计浏览器时，一方面为了辅助本身发展的一种名为 LiveWire 的技术，另一方面为了提高网页的互动性，而创造的一种程序设计语言。后来网景公司与 Sun 公司合作，为了辅助 Java 在网页程序方面的设计，而共同发展 LiveScript 语言，并且将其改名为 JavaScript，这就是 JavaScript 的由来。

JavaScript 是一种轻量级的、解释性的程序设计语言，而且具备一定面向对象的能力。JavaScript 与操作系统无关，它依赖于 JavaScript 的解释引擎。浏览器通过内嵌 JavaScript 的解释引擎从而获得了对 JavaScript 的处理能力。通过在网页中嵌入<script>标签，将 JavaScript 脚本添加到网页中，使得网页不再是简单的静态 HTML，而是包含了控制浏览器的程序、动态创建 HTML 内容和与用户交互的程序，即动态 HTML（DHTML）。

在语法结构上，JavaScript 继承了 C 语言的一些特点，诸如 if 语句、while 循环和&&运算符这样的结构。但是与 C、C++、Java 等程序设计语言所不同的是，JavaScript 是一种无类型（弱类型）的语言，这就是说，它的变量不需要具有一个明确的数据类型。而且，JavaScript 中的对象也与 C 中的结构或 C++和 Java 中的对象不太一样，它更像 Perl 中的关联数组。另外，JavaScript 面向对象的继承机制也与 C++和 Java 大相径庭。

### 2.1.1 对 JavaScript 的误解

一直以来，人们对 JavaScript 有着很多误解，此时很有必要在读者进一步了解它之前对其进行澄清。

#### 1. JavaScript 不是 Java 的简化版

在最新的金山词霸中查找 JavaScript，翻译结果如图 2.1 所示。

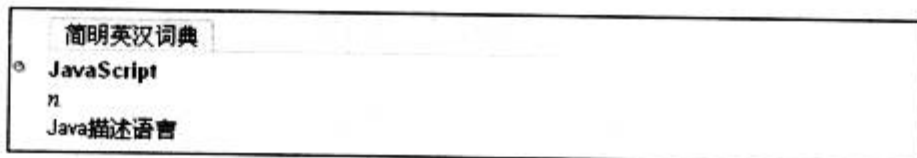


图 2.1 JavaScript 在金山词霸中的翻译

JavaScript 被翻译成了 Java 描述语言，这是不准确的。JavaScript 与 Sun Microsystems 公司的程序设计语言 Java，除了名字和语法上有些相似之外，其实是完全不相干的。而通常的误解，就是将 JavaScript 当作了 Java 的简化版本。

#### 2. JavaScript 并不简单

JavaScript 在很多人心目中都是一种简单的脚本语言的形象。而脚本语言与程序设计语言的差别在于：脚本语言往往很简单，它们是提供给非程序设计人员的程序设计语言。JavaScript 的很多特性使得它更加灵活，让程序设计的新手能更容易地使用，而非程序设计人员也可以用它来做一些简单的工作。其实 JavaScript 是一种具有丰富特性的程序设计语言，它和其他所有的语言一样复杂，甚至比某些语言还要复杂得多。当一个程序设计人员没有扎实的 JavaScript 基础，要想进行复杂的任务，例如搭建大型



Ajax 应用时, 就会显得举步维艰了。

### 3. JavaScript、JScript 与 ECMAScript

很多人并不清楚 JavaScript、JScript 与 ECMAScript 之间的联系和区别。因为 JavaScript 作为网页的客户端脚本语言非常成功, 使得微软 (Microsoft) 在 1996 年 8 月将之引入了自己的浏览器 Internet Explorer 3.0 中, 并取名为 JScript。1996 年 11 月, 网景 (Netscape) 将 JavaScript 提交给欧洲计算机制造商协会进行标准化, 并在 1997 年 6 月被采纳推出 ECMA-262 标准, 并将其命名为 ECMAScript。JavaScript 与 JScript 都与 ECMAScript 相容, 并包含超出 ECMAScript 的功能。本书中统一使用 JavaScript 来引用这些版本中的一个, 包括微软 (Microsoft) 公司的 JScript。

## 2.1.2 JavaScript 的版本

JavaScript 发展到现在, 经历了多次版本的更替。现在主流的浏览器中使用的版本都是基于 ECMAScript v3 的实现, 本书也主要针对这个版本做相关介绍。详细的版本记录如表 2-1 所示。

表 2-1 JavaScript 版本记录

版 本	说 明
JavaScript1.0	最初版本。由Netscape2实现
JavaScript1.1	引入真正的Array (数组) 对象, 修正大量重要错误。由Netscape3实现
JavaScript1.2	引入swtich语句、正则表达式和大量新特性。基本符合ECMA v1, 但有一些不兼容性。由Netscape4实现
JavaScript1.3	完全符合ECMA v1。由Netscape4.5实现
JavaScript1.4	只在Netscape服务器产品中实现
JavaScript1.5	引入异常处理, 符合ECMA v3。由Mozilla和Netscape6实现
JScript1.0	相当于JavaScript1.0。由IE3的早期版本实现
JScript2.0	相当于JavaScript1.1。由IE3的后期版本实现
JScript3.0	相当于JavaScript1.3, 符合ECMA v1。由IE4实现
JScript4.0	还没有基于任何浏览器的实现
JScript5.0	支持异常处理, 符合ECMA v3。由IE5实现
JScript5.5	相当于JavaScript1.5, 完全符合ECMA v3。由IE5.5和IE6实现
ECMA v1	第一个标准版本, 标准化了JavaScript1.1的基本特性, 并添加了一些新特性。没有标准化swtich语句和正则表达式
ECMA v2	该标准的维护版本, 添加了说明, 但没有定义任何新特性
ECMA v3	标准化了swtich语句、正则表达式和异常处理

## 2.2 数据类型和值

所有的计算机程序都是通过操作各种不同的值 (value) 来运行的, 而值的类型称为数据类型 (data type)。程序设计语言最基本的特性之一, 就是它所支持的数据类型的集合。JavaScript 支持 3 种最基本的数据类型: 数字、字符串和布尔值, 另外还支持两种比较特殊的小数据类型: null (空) 和 undefined

(未定义)，它们各自只被定义了一个可以被使用的值。除此之外，JavaScript 还支持复合数据类型：对象、数组、函数以及一些对象类。下面将对它们一一介绍。

### 2.2.1 数字

数字 (number) 是最基本的数据类型。与其他程序设计语言不同，JavaScript 并不区分整型数值和浮点数值，所有的数字都是由浮点型表示的。JavaScript 采用 IEEE754 标准定义的 64 位浮点格式来表示数字，所能表示的最大值为  $\pm 1.7976931348623157 \times 10^{303}$ ，最小值是  $\pm 5 \times 10^{-324}$ 。常见使用方法如下所示。

```
var a = 1000;
var b = 3.1415926;
var c = 4.14e2;
var d = 3.28E-5;
```

注意：4.14e2 及 3.28E-5 为指数计数法，即等同于  $4.14 \times 10^2$  和  $3.28 \times 10^{-5}$ 。

当一个数字大于 JavaScript 所能表示的最大值时，会返回一个特殊的无穷大值：Infinity。同样，当一个负值比 JavaScript 所能表示的最小的负值还要小时，结果就是负无穷大：-Infinity。当一个数值的运算产生了未定义的结果或者是发生错误后返回时，这时结果会是一个非数字的特殊值：NaN，它与包括自己在内的任何值都不相等，所以 JavaScript 提供了一个专门的函数 isNaN() 来检测它，相关的函数 isFinite() 可以用来检测一个 NaN、Finite 或 -Finite 值。

JavaScript 提供了一些常量来表示一些特殊的数字相关的值，如表 2-2 所示。

表 2-2 JavaScript 数字常量列表

常 量	说 明
Infinity	表示无穷大的特殊值
NaN	特殊的非数字值
Number.NaN	特殊的非数字值
Number.MAX_VALUE	可表示的最大值
Number.MIN_VALUE	可表示的最小值
Number.POSITIVE_INFINITY	表示正无穷大的特殊值
Number.NEGATIVE_INFINITY	表示负无穷大的特殊值

### 2.2.2 字符串

字符串 (string) 是由单引号或者双引号括起来的 Unicode 字符序列，其中可以含有 0 个或多个字符。与 C、C++ 或 Java 不同的是：JavaScript 并没有 char 这种单个字符的数据类型，所以要表示单个字符，只能用长度为 1 的字符串来代替。

```
var str1 = 'Hello World!';
var str2 = "Hi, my friend!";
```

```
var str3 = 'b';  
var str4 = " ";
```

用引号界定的字符串中可以包含单引号，用单引号界定的字符串中可以再包含双引号。如果用单引号界定的字符串中需要再使用单引号，或者用双引号界定的字符串中需要再使用双引号，则可以使用转义符\（反斜线）对其进行转移。

```
var str1 = 'Please say: "Hello".';  
var str2 = "I know what 'JavaScript' is. ";  
var str3 = 'I don\'t know.';  
var str4 = "We usually use \" to mark a string.";
```

转义符还可以和一些特定的字符结合来组成转义序列，以用来表示一些特殊的符号，例如换行符的转义序列为\n。更多转义序列可查看表 2-3。

表 2-3 JavaScript 的转义序列

序 列	所表示的字符
\0	NUL 字符
\b	退格符
\t	水平制表符
\n	换行符
\v	垂直制表符
\f	换页符
\r	回车符
\"	双引号
'	单引号
\\	反斜线符
\xXX	由两位十六进制数XX指定的Latin-1字符
\uXXXX	由4位十六进制数XXXX指定的Unicode字符

注意：如果转义符\位于表 2-3 中所示的字符之外的字符前时，则忽略转义符。如\a等同于 a。

### 2.2.3 布尔值

布尔值（bool）只有两个值：true 和 false，用于表示某个事务为真还是为假。在 JavaScript 中，布尔值通常用在作为比较的结果和流程控制中。

```
if(a == 4)  
{  
    alert("a 等于 4");  
}  
else  
{  
    alert("a 不等于 4");  
}
```

在上例代码中，表达式 `a == 4`，判断变量 `a` 的值是否与 4 相等，返回一个表示比较结果的布尔值。如果这个值为真（`true`），则执行 “`alert("a 等于 4");`” 这行代码；否则，则执行 “`alert("a 不等于 4");`” 这行代码。

## 2.2.4 函数

函数（`function`）是一个可执行的程序代码段。函数被定义后，可以多次被程序所调用。JavaScript 的函数可以带有实际参数或形式参数，用于指定这个函数执行计算时所需要使用到的一个或者多个值，在函数执行完后可以返回一个值来表示计算结果。一个计算两数和的函数代码如下所示。

```
function add(a,b)           //函数声明，函数名为 add，接受两个参数：a 和 b
{                             //函数体开始
    var c;                   //声明变量 c
    c = a + b;               //计算 a 和 b 的和并保存到变量 c 中
    return c;                //将 c 的值作为函数的返回值返回
}                             //函数体结束
```

这里为了让读者更容易理解，书写的比较繁琐，其实可以更简洁地表达这个函数，代码如下所示。

```
function add(a,b)
{
    return a + b;
}
```

下面是调用它的代码。

```
var a = add(1,3);           //a = 4
var b = add(1,2);           //b = 3
var c = add(a,b);           //c = 7
```

JavaScript 一个非常重要的特性就是 JavaScript 代码可以对函数进行操作。在许多其他语言中，函数都只是语法结构，它们可以被定义和调用，但却不是数据类型。而 JavaScript 中的函数却是一个真正的数据类型，这使得函数可以被存储在变量、数组和对象中，而且还可以把函数作为参数传递给其他函数，给语言带来了巨大的灵活性。

将函数保存在变量中，代码如下所示。

```
var add = function(a,b)     //使用匿名方法声明函数，并保存在 add 变量中
{
    return a + b;
}
var c = add(1,2);           //调用 add 变量中保存的函数，运行结果为 c = 3
```

将函数作为参数传递，代码如下所示。

```
function add(a,b)           //声明一个计算两数和的函数，函数名为 add
{
    return a + b;
```



```

}
function exec(a,b,func)  //声明一个函数名为 exec 的函数, 接受 3 个参数: a、b、func
{
    return func(a,b);    //以 a 和 b 为参数调用函数 func, 并返回计算结果
}
var c = exec(1,2,add);    //以 1、2 和函数 add 作为参数调用函数 exec, 运行结果为 c = 3

```

函数还可以被保存到对象的属性中, 此时往往被当作对象的方法来使用, 代码如下所示。

```

var person = new Object();    //创建一个对象:person
person.name = 'Robin Chen';    //给 person 对象添加一个 name 属性, 其值为'Robin Chen'
person.getName = function()    //给 person 对象添加一个 getName 属性
                                //并声明一个匿名函数保存在该属性中
{
    return this.name;    //返回当前对象的 name 属性
}
var name = person.getName();    //调用 getName 方法 (即调用 getName 属性中所保存的函数)
                                //运行结果为 name = 'Robin Chen'

```

## 2.2.5 对象

对象 (object) 是已经命名的数据的集合, 这些被命名的数据通常被作为对象的属性来使用, 数据的名称是对象的属性名, 而数据的值则是对象的属性值。对象的属性本身可以是任何的 JavaScript 数据类型。创建一个对象可以使用 new 关键字创建对象实例, 也可以使用对象直接量的语法。

```

var person = new Object();
person.name = 'Robin Chen';
person.sex = 'male';

```

与下面的写法是等价的。

```

var person = {
    name:'Robin Chen',
    sex:'mail'
}

```

**注意:** 当使用诸如 person.name = 'Robin Chen' 的语句时, 程序首先会检查 person 对象是否包含 name 属性, 如果包含, 则将 name 属性的值改变为 'Robin Chen'; 如果不包含, 则会为 person 对象创建一个名为 name 的属性, 并将其值设置为 'Robin Chen'。

对象还可以作为关联数组来使用, 代码如下所示。

```

var name = person.name;
var sex = person.sex;

```

和下面的写法是等价的。

```

var name = person['name'];
var sex = person['sex'];

```

## 2.2.6 数组

数组 (array) 和对象一样, 都是数据的集合, 所不同的是: 对象中每一个数据元素都有一个名字, 而数组中的每一个数据元素都有一个编号 (下标)。数组的下标是从 0 开始的。例如有一数组名字为 `a`, 则 `a[0]` 表示第一个元素, `a[1]` 表示第二个元素, 依此类推。数组的元素可以为任何 JavaScript 数据类型。JavaScript 没有多维数组, 但是数组的元素可以为其他数组。JavaScript 是一种无类型的语言, 所以 JavaScript 中数组的元素不需要具有相同的类型。

数组的创建也和对象的创建一样, 支持两种方式, 代码如下所示。

```
var a = new Array();  
a[0] = 1;  
a[1] = 'Robin Chen';  
a[2] = new Object();
```

和下面的写法是等价的。

```
var a = [1, 'Robin Chen', new Object()];
```

`new Array()` 还可以带参数使用。当参数只有一个时, 如果该参数为数字, 则会返回一个含有该参数所标识的元素数量的数组, 如果数字不是整数, 则会发生错误并抛出异常。如果该参数不是数字, 或者提交的参数不止一个时, 则参数会被按照书写顺序作为返回的数组的元素保存下来。

```
var a = new Array(8);
```

数组 `a` 有 8 个元素。

```
var a = new Array(3.14);
```

则发生错误, 提示: 数组长度必须为有限正整数。

```
var a = new Array('abc');
```

数组 `a` 有 1 个元素, 为 'abc'。

```
var a = new Array(1, 3.14, 'abc');
```

数组 `a` 有 3 个元素, 按照下标顺序依次为: 1、3.14、'abc'。

## 2.2.7 null 值

JavaScript 中的 `null` 关键字是一种特殊的值, 它表示“没有值”。它是独一无二的, 有别于所有其他类型的值。`null` 通常用来标识和判断目标对象是否有意义。当一个变量的值为 `null` 时, 就可以判断它的值不是有效的数字、字符串、布尔值、对象和数组。

## 2.2.8 undefined 值

同 `null` 一样, `undefined` 也是一种特殊的值, 它的意思是“未声明的、未定义的”。当使用了一个

未声明的变量，或者使用了一个声明了但并未赋值的变量，或者使用了一个并不存在的对象的属性/方法时，程序就会返回这个值。注意，`undefined` 和 `null` 虽然不同，但是表达式 `undefined == null` 却是返回 `true`，所以如果要比较 `undefined` 值和 `null` 值，需要使用全等运算符`===`。

## 2.3 JavaScript 的变量

变量是计算机编程中一个重要的概念。变量是一个可以存储值的名称，这些值可以在程序运行中被改变。JavaScript 的变量可以存储任何 JavaScript 支持的数据类型的值。本章将向读者介绍如何使用 JavaScript 的变量。

### 2.3.1 变量的类型

JavaScript 是一种无类型（弱类型）程序设计语言，这表示 JavaScript 的变量可以存放任何类型的值，而其他强类型语言，诸如 C 或者 Java 的变量，都只能存放特定类型的值。JavaScript 的变量可以在程序中被先后赋予不同数据类型的值，而这在 C 或者 Java 中是不允许的。

```
var a = 1;
a = 'ajax';
```

这在 JavaScript 中是合法的。由于 JavaScript 的变量缺乏类型规则，所以在程序处理过程中，会在必要的时候自动将变量做类型转换，参考以下代码。

```
var a = 1;
var b = 'ajax';
var c = a + b;      //c = '1ajax';
```

数字型的变量 `a` 和字符串型的变量 `b` 在做`+`运算时，变量 `a` 会自动被转换成字符串后进行运算。

### 2.3.2 变量的声明

使用一个 JavaScript 的变量之前，必须先声明它。变量的声明使用 `var` 关键字。

```
var a;
var b;
```

也可以一次声明多个变量。

```
var a,b;
```

也可以在声明变量的同时初始化变量。

```
var a = 1, b = 'ajax';
var c = a + b;
```

如果在声明变量时没有初始化变量，则变量会有一个默认的初始值：`undefined`。

### 2.3.3 变量的作用域

变量的作用域是指在程序中定义这个变量的区域。变量按照作用域的不同，一般分为全局变量和局部变量。全局变量是指变量的作用域是全局性的，在整个应用程序中，处处都有该变量的定义，也就是在任何地方都可以使用它。局部变量的作用域是局部的，只能在局部范围内被使用。在 JavaScript 中，在函数内声明的变量是局部变量，其有效范围是整个函数体。函数的参数也是局部变量。而在函数体外声明的变量，则是全局变量。参考以下代码。

```
var globalVar = 'Global';    //globalVar 为全局变量,作用域是整个应用程序
function add(a,b)            //参数 a、b 均为局部变量,作用域是函数 add
{
    var localVar = a + b;    //globalVar 为局部变量,作用域是函数 add
    return localVar;
}
```

在函数体内如果存在与全局变量同名的局部变量，则该局部变量的优先级比同名的全局变量高。

```
var a = 'global';
function test()
{
    var a = 'local';
    return a;
}
var b = test();           //b = 'local'
```

在 JavaScript 中，如果给一个未定义的变量赋值，则 JavaScript 会自动为该变量进行全局性的隐式定义，而这一特性在函数体内也有效果。

```
function test()
{
    var a = 'local';
}
test();
alert(a);                //运行结果为 undefined
```

因为在函数 test 内声明的变量 a 使用了 var 关键字，所以 a 是一个局部变量，则在函数体外无法使用，返回 undefined。

```
function test()
{
    a = 'global';
}
test();
alert(a);                //运行结果为 global
```

而在这段代码中，a 是由 JavaScript 隐式声明的，其作用域是全局性的，所以在函数体外仍然可以继续使用。需要特别说明的是，在函数体内声明的变量，在代码中无论其声明的位置在哪里，它在整



个函数体中都是有定义的。参考下面的代码。

```
var a = 'global';
function test()
{
    alert(a);           //运行结果为 undefined
    var a = 'local';
    alert(a);           //运行结果为 local
}
test();
```

在函数 test 中, 变量 a 虽然在第二行才被定义, 但是整个函数体内都有局部变量 a 的定义, 全局变量 a 的定义被隐藏了。所以函数 test 中第一行的 alert(a) 的运行结果并不是 global, 这时虽然存在局部变量 a 的定义, 但是 a 并没有被初始化, 所以只有一个默认的值 undefined。

注意: 在实际编码过程中, 建议读者声明所有变量都使用 var 关键字, 并且在函数体内将变量的声明都放在最前面, 这样可以避免一些不必要的错误。

### 2.3.4 基本类型和引用类型

所有的数据类型可以被分为两种: 基本类型和引用类型。基本类型在内存中具有固定的大小, 而引用类型则没有固定的大小。基本类型的变量存储的是值的本身, 而引用类型的变量存储的是对值的引用。通常引用的形式是指针或者内存地址, 它告诉变量在哪里可以找到这个值。理解基本类型和引用类型的差别, 对 JavaScript 学习是很重要的。看下面一段代码。

```
var a = 1;
var b = a;
a = a + 1;
alert(a);           //结果为 2
alert(b);           //结果为 1
```

数字是基本类型, 所以 var b = a 实际是创建了一个新的副本, 在 a 的值改变后, b 的值仍然没有发生变化。再看下面一段代码。

```
var a = new Object();
var b = a;
a.name = 'object';
alert(b.name);      //结果为 object
```

object 为引用类型, 所以 var b = a 实际是 b 获得对 a 的值的引用, 即 a 和 b 指向同一个对象。所以在给 a 添加属性 name 时, b 同时也获得了这个属性和相应的值。

提示: 一般来说, 简单的数据类型, 如数字、布尔值, 都是基本类型, 而复合数据类型, 如对象、数组、函数, 都是引用类型。比较特殊的是字符串, 字符串具有可变长度, 所以并不是基本类型, 但是它的行为却又与引用类型不一致, 通常更多的把字符串作为基本类型来理解和使用。

## 2.4 表达式和运算符

2.3 节为读者介绍了 JavaScript 的变量，本节主要介绍 JavaScript 的表达式和运算符。JavaScript 的表达式和运算符与 C、C++ 或者 Java 的很相似。如果读者是一位有经验的 C、C++ 或者 Java 程序员，那么可以快速浏览本节的内容；如果不是，那么读者可以详细地阅读本节有关 JavaScript 的表达式和运算符的知识介绍。

### 2.4.1 表达式

表达式是关键字、变量、常量和运算符的组合，可以用于执行运算、处理字符或测试数据。JavaScript 的解释引擎可以计算表达式，并返回一个结果值。

最简单的表达式可以是一个变量名，或者是直接量，而表达式的值就是变量的值或者直接量本身。更复杂的表达式，经常由简单表达式和运算符组合而成。常见表达式如下所示。

```
3.14
'Hello world'
{name:'Robin',sex:'male'}
[1,3,'hi']
true
null
/Java[sS]cript/
function(x){alert(x);}
i
1 + 3
i / 2
a + (b - 4) * 5
```

### 2.4.2 算术运算符

算术运算指的是数学中最基本的加减乘除等运算。JavaScript 的算术运算符分别介绍如下。

#### 1. 加法运算符 (+)

运算符 “+” 可以对数字进行加法运算，返回两个数字的和。

```
1000 + 100      //返回 1100
```

当把 “+” 运算符单独用在数值之前时，将返回该数值本身。一般而言，这种使用方法没有太大的意义，但如果为了明确指示数值的符号，则可以这样使用。

```
+ 3              //等价于 3
+ 100            //等价于 100
```

## 2. 减法运算符 (-)

运算符“-”计算两个运算数的差。

```
1000 - 100      //返回 900
100 - 1000      //返回 -900
```

当把“-”运算符单独用在数值之前时,将对数值进行取反操作。

```
var a = 100;
var b = -a;      //b = -100
```

## 3. 乘法运算符 (\*)

乘法运算符“\*”会把两个运算数相乘,并返回乘积。

```
3 * 5           //返回 15
```

## 4. 除法运算符 (/)

除法运算符“/”会把两个运算符相除,并返回结果。

```
5 / 2           //返回 2.5
```

注意:除数为0的表达式运算结果为正无穷或负无穷,而0/0的结果为特殊值NaN。

## 5. 模运算符 (%)

模运算符“%”返回两个运算数相除的余数,余数的符号和第一个运算符相同。

```
10 % 3          //返回 1
-10 % 3         //返回 -1
6.4 % 2.1       //返回 0.1
```

## 6. 递增运算符 (++)

递增运算符“++”是对运算数进行递增运算,它可以放在运算数的前面或者后面使用。当递增运算符放在运算数的前面时,会被看作前递增运算符,即先对运算数进行递增,然后用递增后的结果进行后续计算。

```
var a = 1;
var b = ++a;    //a = 2, b = 2
```

当递增运算符放在运算数的后面时,会被看作后递增运算符,即先用运算数的原始值进行计算,而后再对运算数进行递增。

```
var a = 1;
var b = a++;    //a = 2, b = 1
```

## 7. 递减运算符 (--)

递减运算符“--”是对运算数进行递减运算,同递增运算符一样,也分为前递减运算符和后递减运算符两种。

```
/*前递减运算*/
var a = 1;
```

```
var b = -- a;    //a = 0, b = 0
/*后递减运算*/
var c = 1;
var d = c --;    //c = 0, d = 1
```

### 2.4.3 相等运算符

相等运算符用来比较两个值，根据比较结果返回一个布尔值。广义的相等运算符包含相等运算符（==）、等同运算符（===）、不等运算符（!=）和不等同运算符（!==）。

#### 1. 相等运算符（==）

使用相等运算符来比较两个值是否相等，相等则返回 true，否则返回 false。对于简单类型，如数字、布尔值和字符串，比较的是两者的值是否相等。

```
1 == 1           //返回 true
2 == 1           //返回 false
true == true     //返回 true
true == false    //返回 false
'ajax' == 'ajax' //返回 true
'ajax' == 'AJAX' //返回 false
'ajax' == 'javascript' //返回 false
```

对于引用类型，如对象、数组等，比较的是两者的引用是否相同。也就是说，如果有两个数组，它们所包含的元素都是一样的，但是比较的结果仍然为 false。

```
var a = new Array(1,2,3);
var b = a;
var c = new Array(3,1,2);
var d = new Array(1,2,3);
a == b       //返回 true
a == c       //返回 false
a == d       //返回 false
```

更详细的比较规则是：当两个值的类型相同时，就比较它们的值或者引用是否完全相等。当两个值类型不同时，它们也有可能相等：

- ❑ null 与 undefined 相等。
- ❑ 数字和字符串比较，先将字符串转换成数字，再比较值。
- ❑ 布尔值跟其他类型的值比较时，先将布尔值转换成数字再进行比较。true 转换为 1，false 转换为 0。
- ❑ 当数字、布尔值或字符串与对象比较时，先将对象转换成简单类型再进行比较。JavaScript 的内置对象一般会先尝试 valueOf() 转换，再尝试 toString() 转换。比较特殊的是 Date 类，它会先执行 toString() 转换。

```
var a;           //a 为 undefined
null == a        //返回 true
1 == '1'         //返回 true
```



```
1 == true      //返回 true
0 == false     //返回 true
'1,2,3' == [1,2,3] //返回 true
```

## 2. 等同运算符 (===)

等同运算符 (===) 的作用与相等运算符 (==) 一致, 但是比较得更为严格。详细的比较规则为: 如果两个值的类型不同, 则它们不等同; 如果两个值都是 `null` 或都是 `undefined`, 则它们等同。`null` 与 `undefined` 不等。

```
var a;          //a 为 undefined
var b;          //b 为 undefined
a === b         //返回 true
null == a       //返回 false
1 == '1'        //返回 false
1 == true       //返回 false
0 == false      //返回 false
'1,2,3' == [1,2,3] //返回 false
```

## 3. 不等运算符 (!=)

不等运算符 (!=) 的比较规则和相等运算符 (==) 相同, 但是比较结果相反。

## 4. 不等同运算符 (!==)

不等同运算符 (!=) 的比较规则和等同运算符 (===) 相同, 但是比较结果相反。

### 2.4.4 关系运算符

关系运算符用来测试两个值之间的关系, 如果指定关系成立, 则返回 `true`, 否则返回 `false`。关系运算符包含比较运算符、`in` 运算符以及 `instanceof` 运算符。

#### 1. 比较运算符

比较运算符用于确定两个值的相对顺序, 它包含:

##### □ 大于运算符 (>)

当第一个运算数大于第二个运算数时, 返回 `true`, 否则返回 `false`。

```
4 > 1      //返回 true
1 > 2      //返回 false
```

##### □ 小于运算符 (<)

当第一个运算数小于第二个运算数时, 返回 `true`, 否则返回 `false`。

```
4 < 1      //返回 false
1 < 2      //返回 true
```

##### □ 大于等于运算符 (>=)

当第一个运算数大于或者等于第二个运算数时, 返回 `true`, 否则返回 `false`。

```
1 >= 0      //返回 true
1 >= 1      //返回 true
```

```
2 >= 1    //返回 true
1 >= 2    //返回 false
```

#### □ 小于等于运算符 (<=)

当第一个运算数小于或者等于第二个运算数时，返回 true，否则返回 false。

```
1 <= 0    //返回 false
1 <= 1    //返回 true
2 <= 1    //返回 false
1 <= 2    //返回 true
```

比较运算符的运算数可以是任意类型，但是比较运算只能在数字和字符串上执行。当两个运算数类型不同时，或者运算数为非数字和字符串类型时，JavaScript 会先对运算数进行适当的类型转换。具体规则如下：数字和字符串比较时，会先将字符串转换成数字类型再进行比较。如果字符串不能转换成有效的数字，则被转换成 NaN，比较结果始终为 false；当运算数为对象时，会优先转换成数字进行比较。如果不能转换成有效数字，则转换成字符串进行比较。如果不能转换成数字或者字符串，则比较结果始终为 false。

注意：当比较两个字符串时，实际比较的是字符串首字母的 ASCII 码的大小。

```
1 > 'a'    //返回 false
1 < 'a'    //返回 false
'a' < 'b'  //返回 true
'abc' < 'b' //返回 true
```

### 2. in 运算符

in 运算符用来检查第一个运算数是否为第二个运算数的属性名，如果是，则返回 true，否则返回 false。

```
var a = {x:1,y:2};
'x' in a    //返回 true
'y' in a    //返回 true
'z' in a    //返回 false
```

如果第二个运算数为数组，则会检查第一个运算数是否为数组包含的索引之一。

```
var a = ['abc',10,1000];
'abc' in a  //返回 false
10 in a     //返回 false
1000 in a   //返回 false
0 in a      //返回 true
1 in a      //返回 true
2 in a      //返回 true
3 in a      //返回 false
```

### 3. instanceof 运算符

instanceof 运算符用来检查第一个运算数是否为第二个运算数的实例，如果是，则返回 true，否则返回 false。

```
var date = new Date();
date instanceof Date    //返回 true
date instanceof Object  //返回 true(所有对象都是 Object 类的实例)
date instanceof String  //返回 false
```

## 2.4.5 赋值运算符

赋值运算符(=)可以将运算符右边运算数的值赋给左边的运算数,它要求左边的运算数为变量、数组的元素或者对象的属性,而右边的运算数可以为任意类型的值。

```
var a = new Array();
a[0] = 1;
var b = new Object();
b.property = 'property';
b.method = function(){alert(1);}
```

因为在 JavaScript 中“=”被定义为一个运算符,所以以下写法也是被允许的。

```
var a = 1;
var b = 2;
var c = 1 + (a = b) * 2;    //运行结果: a = 2, b = 2, c = 5
```

表达式 a=b 的值就是 a 的值。“=”运算符还可以与其他一些运算符结合使用。

```
a += 1;    //等同于 a = a + 1
a -= 1;    //等同于 a = a - 1
a *= 1;    //等同于 a = a * 1
a /= 1;    //等同于 a = a / 1
a %= 1;    //等同于 a = a % 1
a <<= 1;   //等同于 a = a << 1
a >>= 1;   //等同于 a = a >> 1
a >>>= 1;  //等同于 a = a >>> 1
a &= 1;    //等同于 a = a & 1
a |= 1;    //等同于 a = a | 1
a ^= 1;    //等同于 a = a ^ 1
```

## 2.4.6 逻辑运算符

逻辑运算符通常用来做针对布尔值的操作,主要包含逻辑与(&&),逻辑或(||)和逻辑非(!=) 3 个运算符。

### 1. 逻辑与(&&)运算符

逻辑与运算符作用于两个布尔值,当它们都为 true 时返回 true,否则返回 false。

```
true && true    //返回 true
true && false    //返回 false
false && true    //返回 false
false && false   //返回 false
```

```
4 > 1 && 4 > 2    //返回 true
3 < 1 && 3 <= 3    //返回 false
1 == 1 && 1 <= 2   //返回 true
```

## 2. 逻辑或 (||) 运算符

逻辑或运算符作用于两个布尔值，当它们至少有一个值为 true 时，返回 true，否则返回 false。

```
true || true      //返回 true
true || false     //返回 true
false || true     //返回 true
false || false    //返回 false
```

## 3. 逻辑非 (!) 运算符

逻辑非运算符只作用于单个布尔值，是对布尔值的取反操作。布尔值为 true，则返回 false，布尔值为 false，则返回 true。

```
!true             //返回 false
!false            //返回 true
```

注意：当逻辑运算符的运算数不为布尔值时，JavaScript 会先将其转换成布尔值后进行运算。一般转换规则是：null、undefined、0 和空字符串会被转换成 false，其他的转换成 true。

### 2.4.7 字符串运算符

所有能够作用于字符串的运算符，称为字符串运算符。除了前面介绍的相等运算符和比较运算符外，字符串运算符还包含“+”运算符。

“+”运算符用于连接两个字符串。

```
var str = 'abc' + 'def';    //str = 'abcdef'
```

当“+”运算符作用于一个字符串和一个其他类型的运算数时，其他类型的运算数会先被转换成字符串类型再进行运算。

```
var str = 1 + 'def';    //str = '1def'
```

### 2.4.8 其他运算符

除了上面介绍的运算符外，JavaScript 还有一些其他的运算符。如按位运算符、条件运算符、typeof 运算符、new 运算符、delete 运算符、void 运算符等，在使用到时再作简单介绍。

## 2.5 语 句

在 2.4 节中向读者介绍了 JavaScript 常用的运算符，本节向读者介绍 JavaScript 的语句。一个 JavaScript 程序就是若干语句的集合，每个语句都是程序的组成部分。JavaScript 的语句可以分为表达式语句和复合语句。表达式语句就是一些不同的表达式，而复合语句则是可以包含其他语句的组合格



句。JavaScript 规定每个复合语句可以有一个子语句，可以通过大括号将多个其他语句包含起来组成语句块放入子语句中。下面是一些表达式语句。

```
var str = 'Hello' + 'world';  
(a + 1) * 10;  
window.alert('Hi');
```

下面是一些复合语句。

```
if(a > 1)  
{  
    a++;  
    alert(a);  
}
```

技巧：每个语句都以分号(;)来标识结束。JavaScript 规定当一个语句单独占一行时，可以省略结尾的分号，但是并不推荐这样做。

### 2.5.1 声明变量：var 语句

读者在 2.3 节中已经见过 var 的使用，var 语句用来声明一个或多个变量，语法如下所示。

```
var name_1 [= value_1] [, ... , name_n [= value_n ]];
```

var 后面是一个或者多个需要声明的变量，之间用逗号分隔。每个变量都可选择一个初始化的表达式，用于指定其初始值，如果没有初始化的表达式，则变量初始值为 undefined。

```
var a = 0, b, c = 'ajax', d = new Array();
```

### 2.5.2 流程控制：if 语句

JavaScript 中最基本的流程控制语句，就是 if 语句。if 语句有两种形式：if 和 if...else，先来看第一种。if 语句的基本语法如下所示。

```
if(表达式)  
{  
    子语句  
}
```

下面是一个常见的示例。

```
var a = 1, b = 2;  
if(a < b)  
{  
    alert('a 小于 b');  
}  
if(a > b)  
{  
    alert('a 大于 b');  
}
```

if 语句先计算 if 后面括号中表达式的值为 true 还是为 false。如果为 true，则执行后面紧跟着的子语句；如果为 false，则跳过子语句，如图 2.2 所示。

在上面的代码中， $a < b$  的结果为 true，子语句 `alert('a 小于 b')` 被执行，而  $a > b$  为 false，则子语句 `alert('a 大于 b')` 没有被执行。if 语句第二种形式 if... else 的基本语法如下所示。

```
if(表达式)
{
    子语句
}
else
{
    子语句
}
```

使用 if...else 结构的示例如下所示。

```
var a = 1, b = 2;
if(a > b)
{
    alert('a 大于 b');
}
else
{
    alert('a 小于 b');
}
```

if...else 语句先计算 if 后面括号中表达式的值为 true 还是 false。如果为 true，则执行后面紧跟着的子语句，执行完后跳出整个语句；如果为 false，则跳过 if 后面的子语句转而执行 else 后面的子语句，如图 2.3 所示。

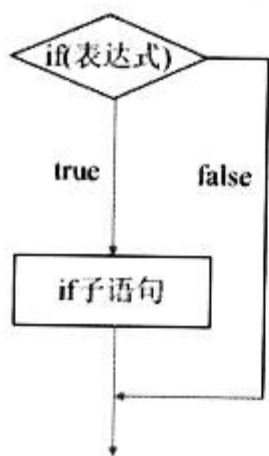


图 2.2 if 语句流程图

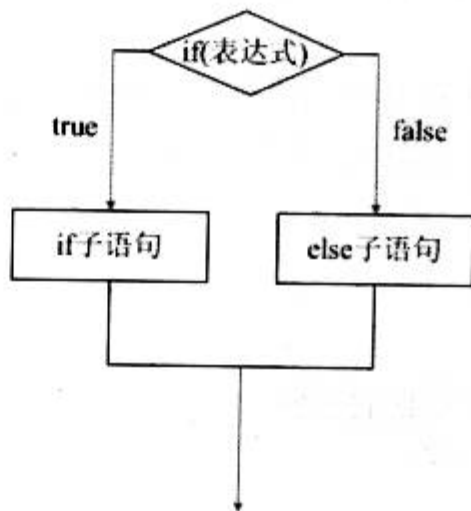


图 2.3 if...else 语句流程图

在上面的代码中表达式  $a > b$  的值为 false，则程序跳过 if 子语句 `alert('a 大于 b')`，直接执行 else 子语句 `alert('a 小于 b')`，然后跳出整个 if...else 语句。

### 2.5.3 流程控制: else if 语句

if...else 语句可以在两个子语句中选择其中之一执行, 如果需要在多条语句中选择一条语句执行, 则可以使用 else if 语句。else if 语句其实并不是一种真正的 JavaScript 语句, 而是 if...else 语句的一种变化形式。else if 语句的示例如下所示。

```
var a = 10;
if(a > 20)
{
    alert('a 大于 20');
}
else if(a > 15)
{
    alert('a 大于 15');
}
else
{
    alert('a 不大于 15');
}
```

与下面的代码是等价的。

```
var a = 10;
if(a > 20)
{
    alert('a 大于 20');
}
else
{
    if(a > 15)
    {
        alert('a 大于 15');
    }
    else
    {
        alert('a 不大于 15');
    }
}
```

### 2.5.4 流程控制: switch 语句

if 语句会在程序执行的流程中产生一个分支, 而重复搭配使用的 else if 则可以执行多个分支, 当所有分支都需要重复使用一个变量时, 多次的重复计算就形成了一种浪费, 这时可以使用 switch 来代替。switch 语句的基本语法如下所示。

```
switch(表达式)
{
```

```

case 值 1:
    子语句 1
case 值 2:
    子语句 2
...
case 值 n:
    子语句 n
[default:
    子语句]
}

```

其中 default 部分是可选的。switch 语句接受一个表达式，先计算出表达式的值，然后查找第一个与表达式的值相匹配的 case 标签，执行其子语句，如果没有与之匹配的 case 标签并且程序提供了 default 语句，则执行 default 标签的子语句。当找到一个匹配的 case 标签并执行完其子语句后，switch 并不会自动跳出整个语句，而是顺序执行后面的子语句。switch 语句流程如图 2.4 所示。

下面是一个 switch 语句的示例。

```

var a = 1;
switch(a)
{
    case 0:
        alert('0');
    case 1:
        alert('1');
    case 2:
        alert('2');
    default:
        alert('3');
}

```

上面程序执行后会先后弹出 3 个对话框，内容分别为：1、2、3。

更多的时候，只需要执行与表达式的值相匹配的 case 标签的子语句，这时可以使用 break 语句（break 语句在后面会做专门介绍）。在子语句中加上 break 语句，当程序遇到 break 时，就会跳出整个 switch 语句，流程如图 2.5 所示。

下面是一个使用 break 的 switch 语句示例。

```

var a = 1;
switch(a)
{
    case 0:
        alert('0');

```

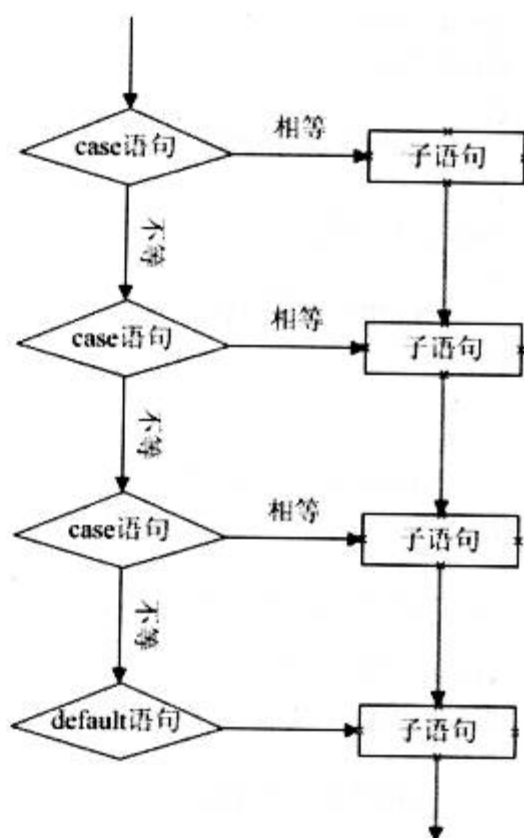


图 2.4 switch 语句流程图

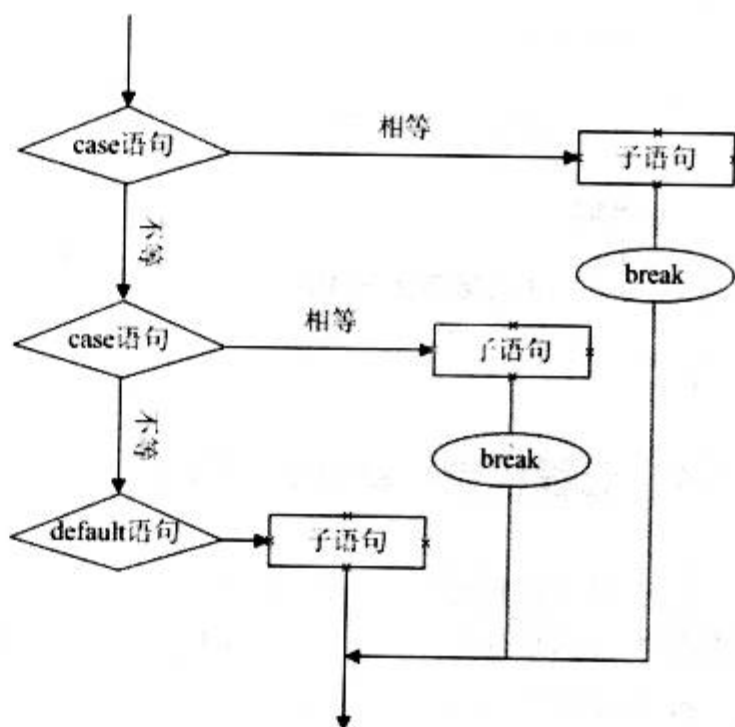


图 2.5 使用 break 的 switch 语句



```
        break;
    case 1:
        alert('1');
        break;
    case 2:
        alert('2');
        break;
    default:
        alert('3');
}
```

程序运行后只会弹出一个对话框，其内容为 1。

### 2.5.5 循环：while 语句

while 语句允许 JavaScript 执行重复的动作，其基本语法如下所示。

```
while(表达式)
{
    子语句
}
```

while 语句在每次循环中都会先计算表达式的值，如果值为 true，则执行其子语句，并在执行完后进入下一次循环；如果值为 false，则跳出 while 语句。while 语句的流程如图 2.6 所示。

下面是一个 while 语句的示例。

```
var i = 1;
while(i <= 10)
{
    i++;
}
alert(i);
```

当 i 小于或者等于 10 时，都会执行子语句 i++，while 语句循环 10 次，程序输出结果为 11。

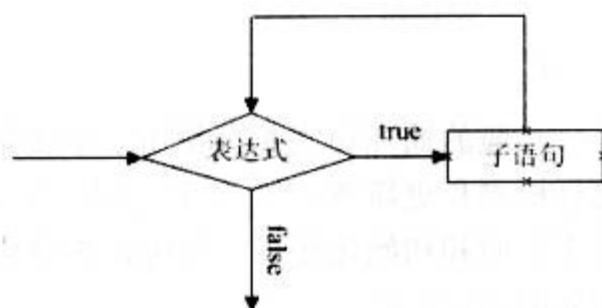


图 2.6 while 语句流程图

### 2.5.6 循环：do/while 语句

do/while 语句与 while 语句非常相似，所不同的是 do/while 是在语句底部检查表达式的值，所以其子语句至少会被执行一次。其基本语法如下所示。

```
do
{
    子语句
}
while(表达式);
```

do/while 语句的流程如图 2.7 所示。

下面是一个 do/while 语句的示例。

```
var i = 1;
do
{
    i++;
}
while(i > 10);
alert(i);
```

在上述程序中，虽然 i 的初始值为 1，表达式  $i > 10$  的值为 false，但是子语句还是执行了一次，所以程序执行结果为 2。

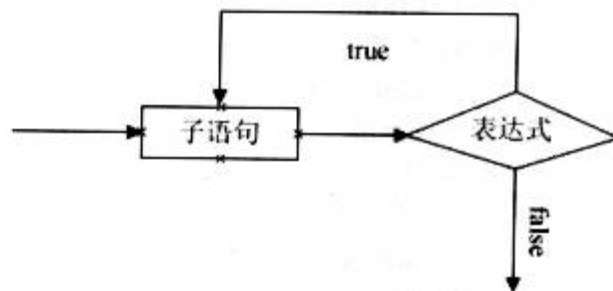


图 2.7 do/while 语句流程图

## 2.5.7 循环：for 语句

for 语句与 while 语句一样，同样用于循环操作，但是它提供了比 while 语句更方便的语法结构。其基本语法如下所示。

```
for(表达式 1;表达式 2;表达式 3)
{
    子语句
}
```

一般的循环模式都会使用一个变量来控制语句的流程，其包含变量的初始化、将变量代入表达式进行检测和更新变量值 3 个过程。在 for 语句中明确地将这 3 个过程声明为语法的一部分：表达式 1 用来声明和初始化变量，表达式 2 用来检查是否继续循环，表达式 3 用来更新变量的值。for 语句的流程如图 2.8 所示。

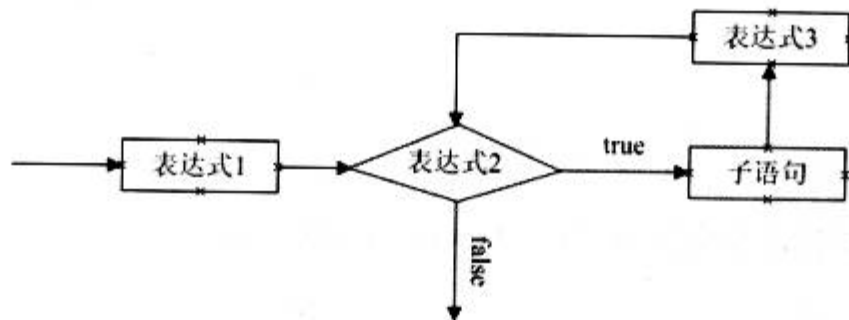


图 2.8 for 语句流程图

下面是一个 for 语句的使用示例。

```
for(var i = 0, sum = 0; i <= 10; i++)
{
    sum += i;
}
alert(sum);
```

运行结果为 55。

## 2.5.8 遍历: for/in 语句

for 语句还有一个不同的用法, 就是与 in 搭配使用。其语法如下所示。

```
for(变量 in 对象)
{
    子语句
}
```

for/in 语句会遍历所传入的对象的每一个属性, 将属性名存入传入的变量中并执行子语句。for/in 语句的流程如图 2.9 所示。

下面是一个 for/in 语句的使用示例。

```
var person = {
    name:'Robin Chen',
    sex:'male',
    age:18,
    grow:function(){
        this.age++;
    }
}
for(o in person){
    document.write(o + ' = ' + person[o] + '<br/>');
}
```

运行结果如图 2.10 所示。

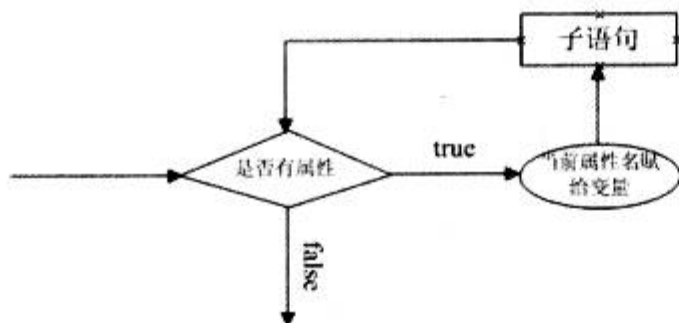


图 2.9 for/in 语句流程图

```
name=Robin Chen
sex=male
age=18
grow=function () { this.age++; }
```

图 2.10 运行结果

在一些情况下, for/in 语句并不能列举出对象所有的属性。一些对象的属性被标记成了只读的、永久的 (不可删除) 或不可列举的, 这些属性使用 for/in 语句不能列举出来。虽然所有用户定义的属性都可以被列举, 但是许多内部属性/方法都是不可列举的。

## 2.5.9 控制语句: break 语句

break 语句会使程序立即跳出包含在最内层的循环或者跳出一个 switch 语句, 它的语法如下所示。

break;

回顾 2.5.4 小节的内容, 读者已经见过如何在 switch 语句中使用 break 语句, 下面的例子展示如何在一个循环语句中使用 break 语句。

```
var a = new Array(1,5,20,2,100,553,23);
for(var i = 0; i < a.length; i++)
{
    if(a[i] > 50)
    {
        alert('数组中第一个大于 50 的数字是' + a[i]);
        break;
    }
}
```

程序运行结果为“数组中第一个大于 50 的数字是 100”。

break 语句还有一种不同的使用方法, 就是搭配标签使用。JavaScript 允许用标识符和冒号来标识一个语句, switch 语句中的 case 和 default 都是此类标识符。标识符可以是除 JavaScript 关键字以外的任意字符组合, 参考下面的示例代码。

```
myloop:
for(var i = 0; i < 10; i++)
{
    alert(i);
}
```

break 与标签结合的语法如下所示。

break 标签名;

注意: break 和标签名中间不能有换行。

上面计算数组中第一个大于 50 的数字的程序可以修改为如下代码。

```
var a = new Array(1,5,20,2,100,553,23);
myloop:
for(var i = 0; i < a.length; i++)
{
    if(a[i] > 50)
    {
        alert('数组中第一个大于 50 的数字是' + a[i]);
        break myloop;
    }
}
```

break 和标签的组合更多地用在多层循环结构中。

```
outerloop:
for(var i = 1; i <= 10; i++)
{
    innerloop:
    for(var j = 1; j <= 10; j++)
```



```
{
    if(j == 8)
    {
        break innerloop;
    }
    if(i * j == 24)
    {
        break outerloop;
    }
}
alert("i = " + i + ", j = " + j);
```

运行结果为 “i = 4, j = 6”。

### 2.5.10 控制语句: continue 语句

continue 语句用于循环中, 跳出当前循环而直接进入下一次循环。continue 语句只能用在 while、do/while、for、for/in 语句中。与 break 的语法一样, continue 可以单独使用, 也可以和标签搭配使用。

```
continue;
continue 标签名;
```

下面是简单的 continue 使用示例, 代码如下。

```
for(var i = 0, total = 0; i < 5; i++)
{
    if(i == 2)
    {
        continue;
    }
    total += i;
}
alert(total);
```

程序运行结果为 8。再来看一个复杂一点的搭配标签使用的示例, 代码如下。

```
outerloop:
for(var i = 0; i < 10; i++)
{
    innerloop:
    for(var j = 0; j < 10; j++)
    {
        if(i == 2)
        {
            continue outerloop;
        }
        if(j == 2)
        {
```

```
        continue innerloop;
    }
}
}
```

### 2.5.11 函数语句：function 语句

function 语句用来定义一个 JavaScript 的函数。在 2.2 节中已经向读者介绍了一些有关函数的知识。关于函数详细的介绍会在 2.6 节中呈现给读者。

### 2.5.12 函数返回值：return 语句

return 语句用在函数体中，用于指定当前函数的返回值，并跳出当前函数。其语法如下所示。

```
function 函数名
{
    ...
    return 返回值;
}
```

先来看一个简单的例子，代码如下。

```
function add(x,y)
{
    return x + y;
}
var a = add(1,2);    //a = 3
```

在上述代码中，return 后面的是一个表达式，程序会先计算表达式的值，然后将此值作为返回值返回。return 可以放在函数体内的任何位置，一个函数内也可以有多个 return 语句。

```
function compare(x,y)
{
    if(x > y)
    {
        return 1;
    }
    else if(x == y)
    {
        return 0;
    }
    else
    {
        return -1;
    }
}
```

return 也可以不带返回值单独使用, 这时函数的返回值为 undefined。

### 2.5.13 抛出异常: throw 语句

throw 语句用于抛出异常。所谓异常是一个信号, 表示当前程序出现了某种异常的状态或者错误。抛出一个异常, 就是发送一个异常的信号。这个信号可以被捕捉和处理。throw 语句的语法如下所示。

throw 异常信息;

例如, 在一个函数中, 如果传入的参数大于某个指定的值, 则抛出一个异常, 代码如下所示。

```
function checkAge(age)
{
    if(age < 18)
    {
        throw new Error('年龄未满 18 岁');
    }
    return age;
}
```

### 2.5.14 异常处理: try/catch/finally 语句

利用 throw 语句可以抛出一个异常, 处理异常则需要使用 try/catch/finally 语句。其基本语法如下所示。

```
try
{
    子语句
}
catch(变量)
{
    子语句
}
[finally
{
    子语句
}]
```

其中 finally 子句是可选的。使用 try/catch/finally 语句时, 当 try 的子语句有异常抛出, 程序会将异常信息赋给 catch 后的变量并跳入 catch 的子语句执行; 如果 try 的子语句没有异常抛出, 则不执行 catch 的子语句。而如果语句有定义 finally 部分, 则无论是否有异常抛出, 最终都会执行 finally 的子语句。try/catch/finally 语句的流程如图 2.11 所示。

下面是一个异常处理的例子, 代码如下。

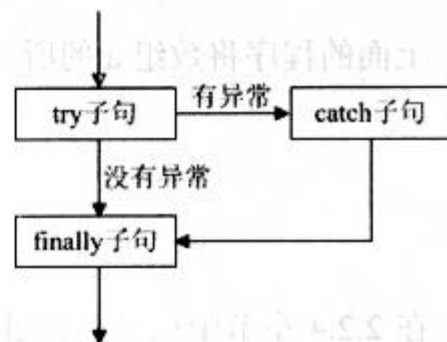


图 2.11 try/catch/finally 语句流程图

```
function checkAge(age)
{
    if(age < 18)
    {
        throw new Error('年龄未满 18 岁');
    }
    return age;
}
try
{
    var age = checkAge(17);
}
catch(e)
{
    alert(e.message);
}
finally
{
    if(typeof age == 'undefined')
    {
        age = 18;
    }
}
```

在上面的例子中，由于传入函数 `checkAge` 的参数 17 小于 18，所以抛出了异常“年龄未满 18 岁”，然后程序跳转到 `catch` 子句执行，变量 `e` 保存了错误对象，`e` 的 `message` 属性则为错误信息。最后程序进入 `finally` 子句执行，执行结果为 `age=18`。

### 2.5.15 空语句

空语句即是空的语句，什么也不做。

;

空语句在某些特定的情况下会很有作用，例如创建一个具有空循环体的循环。

```
var a = new Array(1,3,6,4,2,3,4,1,10);
for(var i = 0; i < a.length; a[i++] = 0);
```

上面的程序将数组 `a` 的所有元素的值都改为 0。

## 2.6 JavaScript 的函数

在 2.2.4 小节中已经介绍过作为数据类型的函数。在本节中将详细地向读者介绍 JavaScript 函数的相关知识。函数是 JavaScript 中一个既重要又复杂的内容。本节先从语法角度出发，向读者介绍函数的定义和调用，然后再向读者介绍一个最常用的函数属性：`arguments` 实际参数列表。



### 2.6.1 函数的定义和调用

下面向读者介绍函数的定义和调用的相关知识。

#### 1. 函数的定义

函数的定义有两种方式，一种是使用 `function` 语句来定义，另外一种就是用 `Function` 构造函数。用 `function` 语句定义函数，其语法如下所示。

```
function 函数名([参数 1[,参数 2,...]])  
{  
    函数体  
}  
var 函数名 = function([参数 1[,参数 2,...]])  
{  
    函数体  
}
```

下面是一些使用 `function` 语句定义函数的例子，代码如下。

```
//一个计算两数和的函数  
function add(x,y)  
{  
    return x + y;  
}  
//比较两个值大小的函数  
var compare = function(x,y)  
{  
    if(x > y)  
    {  
        return 1;  
    }  
    else if(x == y)  
    {  
        return 0;  
    }  
    else  
    {  
        return -1;  
    }  
}
```

用 `Function` 构造函数定义函数，其语法如下所示。

```
var 函数名 = new Function(["参数 1","参数 2",...]) "函数体"
```

`Function` 构造函数接受若干字符串型参数，最后一个参数为函数体内容，其余的是函数的形式参数列表。例如，上面的两个例子可以改写成下面的形式。

```
//一个计算两数和的函数  
var add = new Function('x','y','return x + y');
```

//比较两个值大小的函数

```
var compare = new Function('x','y','if(x > y){return 1;}else if(x == y){return 0;}else{return -1}');
```

## 2. 函数的调用

函数被定义后，就可以在程序中被调用。调用方法是在函数名后面加上括号（），如果需要传递参数，则将参数写在括号中，多个参数用逗号分隔。其语法如下所示。

函数名([参数 1 [,参数 2 [,...,参数 n]]]);

例如，调用上面示例中所声明的函数，代码如下。

```
add(1,3);           //返回 4
add(10,100);        //返回 110
add(-1,4);          //返回 3
compare(1,1);        //返回 0
compare(1,2);        //返回 -1
compare(2,1);        //返回 1
```

当某些函数只需要执行一次时，可以声明匿名函数并立即调用。

```
(function(a,b)
{
    return(a + b);
})(1,2);           //返回 3
```

## 2.6.2 实际参数列表：arguments 属性的使用

虽然在定义函数时，可以为其指定一个形式参数列表，但是实际调用时，传入的参数个数可以与形式参数列表一致。实际传入的参数会被存储到函数的 `arguments` 属性中。`arguments` 属性就像一个数组，但并不是真正的数组，只是具有一些数组的特点。例如，`arguments` 属性有 `length` 属性，`arguments` 属性可以使用下标索引，`arguments[0]` 是传入的第一个参数，`arguments[1]` 是传入的第二个参数，依此类推。请参考下面示例中的函数。

```
function add()
{
    var sum = 0;
    for(var i = 0; i < arguments.length; i++)
    {
        sum += arguments[i];
    }
    return sum;
}
```

函数 `sum` 在定义时，并没有指定形式参数，但是函数体内是根据实际参数进行判断，计算实际传入的所有参数的和，或者将传入所有参数连接为一个字符串。下面是调用函数 `sum` 的例子。

```
add(1,2,3,4,5,6);           //返回 21
add('Ajax',' study');       //返回 'Ajax study'
```

`arguments` 还具有一个名为 `callee` 的属性, 保存了对当前函数的引用。下面是一个利用 `callee` 实现递归算法, 来进行十进制到二进制转换的例子。

```
function recursion(x)
{
    if(typeof arguments[1] == 'undefined')
    {
        var num = "";
    }
    else
    {
        var num = arguments[1];
    }
    var y = parseInt(x / 2);
    num = x % 2 + num;
    if(y == 1)
    {
        return '1' + num;
    }
    else
    {
        num = arguments.callee(y,num);
    }
    return num;
}
//调用
recursion(3);      //返回 11
recursion(4);      //返回 100
recursion(27);     //返回 11011
```

## 2.7 大小写敏感性

JavaScript 是大小写敏感的, 例如, `a` 和 `A` 是两个不同的变量。这点对于 C、C++ 或者 Java 程序员来说, 不是什么问题, 而对于长期从事 VB 或者 VBS 程序开发的读者, 可能会带来代码习惯上的变化。按照一般惯例, JavaScript 在大小写使用上, 应遵循以下规则:

- 常量名全部大写, 如 `NAME`、`CONFIGURATION` 等。
- 变量名小写, 如 `name`、`age`、`sex` 等。如果变量名由多个单词组成, 则从第二个单词开始, 单词首字母大写, 其他字母还是小写, 如 `myName`、`myAge`、`mySex`。

## 2.8 JavaScript 的注释

JavaScript 支持 C 和 C++ 型的注释方法。JavaScript 会把处于 “//” 到一行结尾中间的任何文本都当作注释忽略掉。除此之外, “/\*” 和 “\*/” 之间的任何文本也会当作注释被忽略。“//” 与 “/\*...\*/” 的

区别是“//”只能注释单行内容，而“/\*...\*/”则可以跨越多行内容，但是其中不能有嵌套的注释。下面是注释的例子。

```
//单行注释
/*
    多行注释
    多行注释
*/
```

## 2.9 JavaScript 的保留字

JavaScript 会保留一些词语，这些词语一部分组成了现有的 JavaScript 语法，另外一部分则为了以后的升级扩展而被预留。这些保留字都不能作为变量、函数名或者标签名来使用。表 2-4 列出了 JavaScript 目前使用的保留字。

表 2-4 JavaScript 目前使用的保留字

使用的保留字					
break	do	if	switch	typeof	case
else	in	this	var	catch	false
instanceof	throw	void	continue	finally	new
true	while	default	for	null	try
with	delete	function	return		

表 2-5 列出了 JavaScript 预留的保留字。

表 2-5 JavaScript 预留的保留字

预留的保留字					
abstract	double	goto	native	static	boolean
enum	implements	package	super	byte	export
import	private	synchronized	char	extends	int
protected	throws	class	final	interface	public
transient	const	float	long	short	volatile
debugger					

关于 JavaScript 的基本知识就介绍到这里，第 3 章开始给读者介绍 JavaScript 一些常用对象。



# 第 3 章

## JavaScript 的常用对象

- » 保存多个数据元素的容器: Array 对象
- » 字符串的处理: String 对象
- » 正则表达式: RegExp 对象
- » 日期和时间: Date 对象
- » 复杂的数学运算: Math 对象
- » 操作浏览器窗口: window 对象
- » 操作 HTML 文档: document 对象
- » 应用实例
- » 小结

第 2 章向读者介绍了 JavaScript 的基本知识, 在实际编码过程中, 还会经常用到这样一些对象: 它们由 JavaScript 语言本身提供或者由执行 JavaScript 的浏览器提供, 包含丰富的内置属性和方法。灵活运用这些对象, 可以使 JavaScript 的行为更丰富, 功能更强大, 也可以使编码更简单。下面向读者一一介绍。

## 3.1 保存多个数据元素的容器：Array 对象

第 2 章已经向读者介绍了 JavaScript 的数组。数组是一种数据类型，它包含了被编码的值。每个编码的值称为数组的一个元素，编码被称作下标。每个元素都可以存储任意数据类型的值，同一个数组的不同元素，可以具有不同的数据类型，这点和强类型语言如 Java、C++ 等不同。接下来向读者介绍数组对象的一些常用方法。

### 3.1.1 数组的创建

在 2.2.6 小节中已经向读者介绍了数组的两种创建方式，即由数组直接量创建数组和利用 `Array()` 构造函数来创建数组。参考下面的示例代码。

```
//直接量创建数组
var a = [0,1,2,'abc',new Object()];
var b = [];                                //空数组
var c = [123,['ab',2],'3',22];            //嵌套的数组
//Array 构造函数创建数组
var d = new Array(10);                    //10 个元素的数组
var e = new Array(0,1,2,'abc');
var f = new Array(123,new Array('ab',2),'3',22);
```

### 3.1.2 得到数组的长度

Array 对象提供了一个属性 `length`，让程序可以得到数组中所包含的元素个数，即数组的长度。如有一数组 `a`，则数组的长度为 `a.length`，示例代码如下所示。

```
var a = [1,2,'abc'];
var b = a.length;           //b = 3
```

`length` 属性经常在遍历数组时使用。下面是一个使用 `length` 属性遍历数组的例子。

```
var a = [1,2,3,4,5];
for(var i = 0; i < a.length; i++)
{
    a[i]++;
}
```

在给数组添加新元素时，也可以使用 `length` 属性。

```
var a = [1,2,3,4,5];
a[a.length] = 6;      //a = [1,2,3,4,5,6]
```

上面的两段示例程序其实都巧妙地运用了数组的 `length` 属性总比数组的最大索引大 1 的特点，因为数组的索引是从 0 开始计算的。

数组的 `length` 属性是可写的，也就是说可以直接改变数组的长度。如果给 `length` 属性所赋的新值

比当前 `length` 属性的值大, 那么将会添加新的未定义的元素到数组末尾来填充数组到指定的长度。如果所赋的新值比当前值要小, 则会在数组末尾删除一些元素, 参考下面的示例代码。

```
var a = [1,2,3,4,5];
a.length = 3;           // a = [1,2,3]
a.length = 5;           // a = [1,2,3,undefined,undefined]
```

### 3.1.3 添加、删除和替换数组元素

对数组最常用的操作, 就是添加、删除或者替换数组元素。添加数组元素可以使用 `unshift` 方法或者 `push` 方法。使用数组的 `unshift` 方法可以将一个新的元素添加到数组的开头, 数组中所有原有元素都往后移动一位, 数组长度增加 1。 `unshift` 方法的使用示例如下所示。

```
var a = [1,2,3,4,5];
a.unshift(0);           //a = [0,1,2,3,4,5]
var b = new Array('abc','test');
b.unshift('yes');
b.unshift(32);          //b = [32,'yes','abc','test']
```

使用数组的 `push` 方法可以将一个新的元素添加到数组的末尾, 数组其他元素位置不变, 数组长度增加 1。 `push` 方法的使用示例如下所示。

```
var a = [1,2,3,4,5];
a.push(0);              //a = [1,2,3,4,5,0]
var b = new Array('abc','test');
b.push('yes');
b.push(32);             //b = ['abc','test','yes',32]
```

当需要删除数组中的元素时, 可以使用 `shift` 或者 `pop` 方法。与 `unshift` 方法相反, `shift` 方法删除数组的第一个元素, 数组中其他元素向前移动一位, 数组长度减少。 `shift` 方法的使用示例如下所示。

```
var a = [1,2,3,4,5];
a.shift();              //a = [2,3,4,5];
a.shift();              //a = [3,4,5];
```

使用 `pop` 方法, 则会删除数组中最后一个元素, 其他元素位置保持不变, 数组长度减少 1。下面是 `pop` 方法的使用示例。

```
var a = [1,2,3,4,5];
a.pop();               //a = [1,2,3,4]
a.pop();               //a = [1,2,3]
```

上面介绍的 4 个方法, 只能针对数组首位的元素进行操作, 如果需要对其他元素进行操作, 则需要使用 `splice` 方法, 而且不像前面介绍的 4 个方法, `splice` 方法可以一次操作多个数组元素。 `splice` 方法的第一个参数指定了要插入或者删除的元素在数组中的位置 (相对于索引的位置, 从 0 开始计算), 第二个参数指定了要从数组中删除的元素的个数, 如果这个参数被省略, 则会删除从第一个参数所指定的位置开始, 到数组结束的所有元素。 `splice` 方法的使用示例如下所示。

```
var a = [1,2,3,4,5];  
a.splice(1,1);           //a = [1,3,4,5]  
var b = ['a',3,'s'];  
b.splice(1);             //b = ['a']
```

注意：在 IE 中，splice 方法在只有一个参数时，没有被正确执行，没有任何元素被删除。

需要说明的是：splice 方法并不返回数组操作后的结果，而是返回被删除的元素所组成的新数组。在上面的示例中，a.splice(1,1)返回的是数组[2]。

splice 方法的前两个参数指定了需要删除的元素，而在这两个参数之后还可以有任意多个额外的参数，这些参数会在删除操作完成后，被当作数组的新元素，依次从第一个参数所指定的位置插入到数组中。使用 splice 方法插入新数组元素，代码如下。

```
var a = [1,2,3,4,5];  
a.splice(1,1,'a','b','c'); //a = [1,'a','b','c',3,4,5]
```

### 3.1.4 得到数组片段

利用 Array 对象的 slice 方法，可以得到数组的一个片段，或者说子数组。它的两个参数指定了要返回的数组片段的起始点。数组片段由 slice 方法的第一个参数指定位置开始，到第二个参数指定位置结束的所有元素组成，但不包含第二个参数指定位置的元素。如果省略掉第二个参数，则会返回从第一个参数指定位置的元素到数组结束的所有元素。slice 方法使用示例如下所示。

```
var a = [1,2,3,4,5];  
a.slice(1);           //返回[2,3,4,5]  
a.slice(2,4);         //返回[3,4]
```

slice 方法的参数还可以为负数。当传递给 slice 方法的参数为负数时，位置是从数组末尾开始从后往前计算，例如-3 就是数组的倒数第 3 个元素。下面是一些使用负数作为 slice 方法的示例。

```
var a = [1,2,3,4,5];  
a.slice(-3);          //返回[3,4,5]  
a.slice(2,-1);        //返回[3,4]  
a.slice(-4,-2);       //返回[2,3]
```

### 3.1.5 反转数组

利用 Array 对象的 reverse 方法可以将数组所有元素反转，示例如下所示。

```
var a = [1,2,3,4,5];  
a.reverse();          // a = [5,4,3,2,1]
```

### 3.1.6 将数组转换成字符串

利用 Array 对象的 join 方法可以将数组的所有元素使用指定的字符串连接成一个新的字符串。join



方法可以接受一个参数，如果参数被提供，则使用该参数提供的字符串来分隔和连接数组的所有元素到新的字符串；如果参数没有被提供，则默认使用“,”。join 方法使用示例如下所示。

```
var a = [1,2,3,4,5];
a.join();           //返回'1,2,3,4,5'
a.join('-');        //返回'1-2-3-4-5'
a.join('*');        //返回'1**2**3**4**5'
```

### 3.1.7 数组元素的排序

利用 Array 对象的 sort 方法可以对数组的元素进行排序。当 sort 方法不带参数执行时，在排序时会将数组元素转换成字符串，并按照字符串首个字符的 ASCII 码从小到大排列，如果数组含有未定义的元素，则这些元素会被放到数组末尾。一个使用 sort 方法排序的例子如下所示。

```
var a = ['d','csd',32,1,'ef1','2as'];
a.sort();           //返回[1,'2as',32,'csd','d','ef1']
```

如果需要 sort 方法按照某种指定的规则来对数组进行排序，则需要给 sort 方法传递一个比较函数作为参数。在 sort 方法对数组进行排序时，会从数组中每次取出两个元素传递给排序函数，然后根据排序函数的返回值来决定这两个元素的位置，其过程与冒泡排序算法类似。如果需要第一个元素排在第二个元素之后，则排序函数返回一个大于 0 的数；如果需要第一个元素排在第二个元素之前，则排序函数返回一个小于 0 的数；如果需要保持两个元素的相对位置不变，则排序函数返回 0。例如，可以使用自定义的排序规则来实现倒序排列，代码如下所示。

```
var a = [4,1,23,413,1234,2,31,39];
a.sort(function(x,y)
{
    return y - x;
});
```

上例程序的运行结果为 a = [1234,413,39,31,23,4,2,1]。

## 3.2 字符串的处理：String 对象

在 2.2.2 小节中已经向读者介绍了字符串的一些知识，本节将向读者介绍 String 对象的一些常用属性和方法。

### 3.2.1 获取字符串的长度

同 Array 对象类似，String 对象也有一个表示字符串长度的 length 属性，而与 Array 对象的 length 属性所不同的是：String 对象的 length 属性表示的是字符串所包含的字符数，而且不能被人为地修改。使用 length 属性获取字符串长度的例子如下所示。

```
var a = "";  
a.length;           //返回 0  
var b = 'Ajax';  
b.length;           //返回 4  
'Ajax study'.length; //返回 10
```

### 3.2.2 字符串的截取

用 Array 对象的 slice 方法可以获得一个数组的段落, String 对象也提供了 substring 方法来截取一段字符串的片段。同 slice 方法一样, substring 方法可以接受两个参数来指定截取范围, 当第二个参数被省略时, 默认截取到字符串的结尾。使用 substring 方法截取字符串的例子如下所示。

```
var a = 'I love ajax';  
var b = a.substring(2,4);    // b = 'lo'  
var c = a.substring(2);      // c = 'love ajax'
```

### 3.2.3 字符串的替换

使用 String 对象的 replace 方法可以将字符串中指定的内容替换成新的内容并返回一个新的字符串。replace 方法接受两个参数, 第一个为需要被替换的子字符串, 第二个为替换的内容。当执行 replace 方法时, 程序会在字符串中查找所有与第一个参数相符的片段, 并替换为第二个参数指定的内容。replace 方法完成替换后会返回一个新的字符串, 而不会对原字符串产生影响。下面是一些使用 replace 方法的示例。

```
var a = 'I love ajax';  
var b = a.replace(' ','_');    // b = 'I_love_ajax'  
var c = a.replace('a','o');    // c = 'I love ojax'  
var d = a.replace('ajax','AJAX'); // d = 'I love AJAX'  
// a = 'I love ajax'
```

replace 还有一种更为复杂的使用方法, 就是将一个正则表达式对象当作第一个参数传递给 replace, 则字符串中所有与该正则表达式相匹配的内容都会被替换, 借助正则表达式的灵活性, 使得 replace 方法变得非常强大。使用正则表达式的 replace 方法示例如下所示。

```
var a = 'My telephone number is 1323-440-2573';  
var b = a.replace(/d/, '*');    // b = 'My telephone number is ****-***-*****'  
var c = a.replace(/te.*one/, 'mobile phone'); // c = 'My mobile phone number is 1323-440-2573'  
var d = a.replace(/^(S+)/, 'Hello, Jimmy.$1'); // d = 'Hello, Jimmy.My telephone number is 1323-440-2573'
```

关于正则表达式的相关知识, 会在 3.3 节中向读者作详细介绍。

### 3.2.4 大小写转换

String 对象提供了两个方法来作大小写转换的操作: toLowerCase 方法和 toUpperCase 方法。

`toLowerCase` 方法将字符串中的所有大写字母转换为小写字母, `toUpperCase` 则将字符串中的所有小写字母转换为大写字母。但是这两个方法并不改变原有字符串, 而是转换后返回一个新的字符串。一个大小写转换的示例如下所示。

```
var a = 'I love ajax';
var b = a.toUpperCase();
var c = a.toLowerCase();
```

运行结果为 `a = 'I love ajax'`、`b = 'I LOVE AJAX'`、`c = 'i love ajax'`。

### 3.2.5 将字符串转换成数组

使用 `split` 方法可以将字符串根据指定的子字符串分割成若干元素, 并将这些元素按照顺序在字符串中的先后排列返回一个包含这些元素的数组。`split` 方法可以理解为 `Array` 对象的 `join` 方法的逆运算。`split` 方法使用示例如下所示。

```
var a = '00-12-13-DF-3A';
var b = a.split('-');           // b = ['00','12','13','DF','3A']
var c = 'I love ajax';
var d = c.split(' ');          // d = ['I','love','ajax']
var e = 'a=1&b=2&c=3'.split('&'); // e = ['a=1','b=2','c=3']
```

当传入 `split` 的参数为一个空字符串时, `split` 会将字符串中的每一个字符单独作为一个数组元素。

```
var a = 'ajax study';
var b = a.split('');           // b = ['a','j','a','x',' ','s','t','u','d','y']
var c = '123456789';
var d = c.split('');           // d = ['1','2','3','4','5','6','7','8','9']
var e = 'prototype'.split(''); // e = ['p','r','o','t','o','t','y','p','e']
```

当将 `split` 方法的参数省略掉时, `split` 会将整个字符串作为一个数组元素, 返回一个长度为 1 的数组。

```
var a = 'ajax study';
var b = a.split();             // b = ['ajax study']
var c = 'I love ajax';
var d = c.split();             // d = ['I love ajax']
var e = 'prototype'.split();   // e = ['prototype']
```

### 3.2.6 拼接字符串的优化方法

JavaScript 的字符串更像是一个常量, 其值是不可以被更改的, 在进行字符串的连接操作时, 实际上是创建了新的字符串而非在原有字符串的基础上进行修改, 所以当进行大量字符串拼接的操作时, 会带来性能损耗的问题, 一般的解决办法是用结合数组的 `join` 方法来达到拼接字符串的效果。

```
var a = 'I';
var b = 'love';
```

```
var c = 'ajax';  
var d = a + b + c;           // d = 'I love ajax'
```

上面的代码中，为了计算变量 `d` 的值，进行了两次 “+” 运算，这样实际上在运算过程中生成了两个新的字符串实例。常用优化的办法如下所示。

```
var a = 'I';  
var b = 'love'  
var c = 'ajax';  
var d = [a,b,c].join(' ');   // d = 'I love ajax'
```

将变量 `a`、`b`、`c` 赋给一个数组作为数组元素，然后调用数组的内置方法 `join` 来达到连接字符串的效果，运算过程中没有多余的字符串实例被生成。

### 3.3 正则表达式: RegExp 对象

3.2 节向读者介绍了 `String` 对象的一些常用属性和方法，本节向大家介绍使用广泛，并且和 `String` 对象紧密关联的正则表达式对象：`RegExp`。正则表达式是一个描述字符模式的对象，或者说是用某种模式去匹配一类字符串的一个公式。很多程序设计语言都支持正则表达式，JavaScript 中的正则表达式由 `RegExp` 对象表示。

#### 3.3.1 创建正则表达式

创建一个正则表达式，可以使用 `RegExp` 构造函数和直接量两种方式。就像字符串直接量被包含在一对引号中一样，正则表达式直接量被包含在一对斜线 (/) 中。例如，创建一个正则表达式的方法如下所示。

```
var reg1 = new RegExp('\\d+');  
var reg2 = new RegExp('[1-9a-zA-Z]*$');  
var reg3 = /\d+/;  
var reg4 = /^[1-9a-zA-Z]*$/;
```

其中 `reg1` 与 `reg3` 是等价的，而 `reg2` 与 `reg4` 是等价的。

还可以在创建正则表达式时为其指定一个标志，来说明高级匹配模式的规则。当使用 `RegExp` 构造函数创建正则表达式对象时，需要指定的标志作为第二个参数被传入。当使用直接量方法来创建一个正则表达式对象时，直接在标识表达式结束的斜线 (/) 后面写上标志就可以了。使用标志的正则表达式的示例如下所示。

```
var reg1 = new RegExp('\\d+', 'g');  
var reg2 = new RegExp('[1-9a-zA-Z]*$', 'm');  
var reg3 = /\d+/g;  
var reg4 = /^[1-9a-zA-Z]*$/i;
```

可选的标志及其含义如表 3-1 所示。



表 3-1 正则表达式的标志

标 志	含 义
i	执行匹配时忽略大小写
g	执行一个全局匹配，即找到所有匹配项而不是找到第一个就停止
m	多行匹配模式

正则表达式的标志可以自由搭配使用，参考下面的示例代码。

```
var reg1 = new RegExp('\\d+', 'gi');
var reg2 = new RegExp('[1-9a-zA-Z]*$', 'im');
var reg3 = /d+/gm;
var reg4 = /^[1-9a-zA-Z]*$/img;
```

### 3.3.2 正则表达式的语法规则

正则表达式由字符直接量、元符号和各种转义序列组成。字符直接量匹配的是它本身，除了字母字符和数字外，还可以通过转义符 (\) 开头的转义序列来支持一些非字母字符。表 3-2 列出了这些字符。

表 3-2 正则表达式的直接量字符

字 符	匹 配
字母或数字字符	自身
\o	NUL 字符
\t	水平制表符
\v	垂直制表符
\n	换行符
\r	回车符
\f	换页符
\xnn	由十六进制nn指定的拉丁字符
\uxxxx	由十六进制xxxx指定的Unicode字符
\cx	控制字符x

在方括号 ([]) 中放入单独的直接量，就组成了字符类。一个字符类可以和它所包含的任何单个字符相匹配，例如[abc]能匹配"a"、"b"和"c"。由于字符类很常用，所以正则表达式利用元符号和转义序列预定义了一些特别含义的字符类，如表 3-3 所示。

表 3-3 正则表达式的字符类

字 符	匹 配
[...]	括号内包含的任意字符
[^...]	括号内不包含的任意字符
.	除换行符和其他Unicode行中止符之外的其他字符

续表

字 符	匹 配
\w	大小写字母、数字和下划线
\W	非大小写字母、数字和下划线
\s	空白字符
\S	非空白字符
\d	数字字符
\D	非数字字符

下面是一些使用字符类的示例。

/\d\d/	匹配两位数字
/[jJ]avascript/	匹配'Javascript'和'javascript'
/a.b/	匹配'abb'、'anb~'等
/[^0-9]/	匹配任何单个非数字字符
/[a-z]/	匹配 a~z 的所有小写字母

上面介绍的直接量和字符类都只能表达单位字符，如果要重复类型的多位字符，则需要使用{}、\*、+或者? 元符号。关于这些符号的介绍如表 3-4 所示。

表 3-4 正则表达式的重复符号

符 号	含 义
{m,n}	匹配前一项最少m次，最多n次
{m,}	匹配前一项最少m次
{m}	匹配前一项m次
?	匹配前一项0次或者1次
+	匹配前一项最少1次。等价于{1,}
*	匹配前一项0次或者多次。等价于{0,}

下面是一些使用重复符号的示例。

/\d{1,3}/	匹配 1~3 位数字，如'1'、'123'、'32'等
/a{3}/	匹配'aaa'
/a{1,}/	匹配'a'、'aa'、'aaa'等
/a?/	匹配' '和'a'
/a+/	等价于/a{1,}/
/a*/	匹配' '、'a'、'aa'等。等价于/a{0,}/

在重复匹配时，正则表达式会尽可能多地匹配更多的字符，这种特性称之为贪婪匹配。如果在表达式后面加上元符号?，则会进行非贪婪模式的匹配，匹配必要的尽量少的字符。例如，当使用/a+/匹配字符串'aaab'时，会匹配字符串的前3位'aaa'，而当使用/a+?/匹配时，则只会匹配第一位'a'。

正则表达式还可以指定选择项、为表达式分组和引用前面子表达式所匹配的字符，相关的元符号如表 3-5 所示。

表 3-5 正则表达式选择、分组和引用符号

符 号	含 义
	选择。匹配符号左边的子表达式或者右边的子表达式
(...)	组合。将几个项目组合为一个单元，这个单元可以被重复符号所使用，还可以被引用
(?:...)	只组合。只组合为一个单元，但不能被引用
\n	n是一个数字。表示的是表达式中第n个可以被引用的组合，从左到右按书写顺序排序。\\n匹配的是与其所引用的分组匹配字符一样的字符

下面是一些分组、选择和引用符号的使用示例。

```

/a|b/           匹配'a'、'b'
/(a|b)n\\1/     匹配'ana'、'bnb'
/(?:m+)(a|b)n\\1/  其中\\1 匹配的是 a|b 的匹配结果，而不是 m+的匹配结果

```

正则表达式还有一些字符和转义序列匹配的是字符串中的特定位置，称之为锚字符，如表 3-6 所示。

表 3-6 正则表达式的锚字符

字 符	含 义
^	匹配字符串的开头。在多行匹配模式中，匹配一行的开头
\$	匹配字符串的结尾。在多行匹配模式中，匹配一行的结尾
\\b	匹配一个词语的边界
\\B	匹配所有非词语边界的字符
(?:p)	正前向声明。要求接下来的字符与模式p匹配，但是匹配结果不包含p匹配的字符
(?!p)	反前向匹配。要求接下来的字符不与模式p匹配

下面是一些锚字符的使用示例。

```

/^JavaScript/   匹配'JavaScript is cool'但是不匹配'I love JavaScript'
/m$/           匹配所有以 m 结尾的字符串，如'am'、'boom'
/^JavaScript$/  匹配'JavaScript'但不匹配'JavaScript is cool'和'I love JavaScript'
\\bam/         匹配'I am Robin'但不匹配'ram'
/\\B+$/        匹配'IloveJavaScript'但不匹配'I love JavaScript'
/a(?:a)/      匹配'aa'但不匹配'ab'
/a(?:!a)/     匹配'ab'但不匹配'aa'

```

注意：如果想匹配元符号或者具有特殊含义的符号时，只需要在相应字符前加上转义符反斜线(\\)即可。

### 3.3.3 正则表达式的属性

每个 RegExp 对象都有 5 个属性。属性 source 是一个只读的字符串，它存放的是正则表达式的文本，示例代码如下所示。

```
var reg = new RegExp('[1-9a-zA-Z]*$', 'im');
var text = reg.source;           // text = '[1-9a-zA-Z]*$'
```

属性 `global` 是一个只读的布尔值，它表示正则表达式是否启用了全局匹配模式，示例代码如下所示。

```
var reg1 = new RegExp('\\d+', 'gi');
var reg2 = new RegExp('[1-9a-zA-Z]*$', 'im');
alert(reg1.global);              // true
alert(reg2.global);              // false
```

属性 `ignoreCase` 同属性 `global` 类似，也是一个只读的布尔值，它表示正则表达式是否启用了忽略大小写的匹配模式，示例代码如下所示。

```
var reg1 = new RegExp('\\d+', 'g');
var reg2 = new RegExp('[1-9a-zA-Z]*$', 'im');
alert(reg1.ignoreCase);          // false
alert(reg2.ignoreCase);          // true
```

属性 `multiline` 也是一个只读属性，它表示正则表达式是否启用了多行匹配模式，示例代码如下所示。

```
var reg1 = new RegExp('\\d+', 'g');
var reg2 = new RegExp('[1-9a-zA-Z]*$', 'im');
alert(reg1.multiline);           // false
alert(reg2.multiline);           // true
```

最后一个属性 `lastIndex` 是一个可读可写的整数，它记录了在全局匹配模式下，在字符串中下一次开始匹配的位置。这个属性的使用将在本书 3.3.5 小节中向读者介绍。

### 3.3.4 用于模式匹配的 String 方法

在 3.2.3 小节中读者已经知道 `String` 对象的 `replace` 方法可以结合正则表达式使用。其实 `String` 对象有 4 个方法可以结合正则表达式使用，其中最简单的，就是 `search` 方法。`search` 方法以正则表达式为参数，返回第一个与之匹配的子字符串的开始位置，如果没有任何子字符串与之匹配，则返回 -1。`search` 方法的使用示例如下所示。

```
var str = 'I love JavaScript and Ajax';
alert(str.search(/l.*e/));        // 匹配'love'，返回 2
alert(str.search(/s/));           // 匹配'I'和'love'之间的空格，返回 1
alert(str.search(/.(?:Script)/)); // 匹配 Script 前面的 a，返回 10
```

注意：如果传递给 `search` 方法的参数不是正则表达式，则参数会被传递给 `RegExp` 构造函数，被转换成正则表达式。同时，`search` 并不支持全局匹配模式，它总是返回找到的第一个子字符串的位置。

方法 `replace` 执行替换操作。在 3.2.3 一节中读者已经知道如何使用 `replace` 方法，这里补充说明一点：当正则表达式中有分组时，在 `replace` 方法的第二个参数中，可以使用符号 `$` 加数字来引用这些分



组所匹配的内容。使用\$符号引用分组内容的示例如下所示。

```
var str1 = 'ajax.php?a=1&b=2&c=3';
var str2 = str1.replace(/^\.\?([a-z]+=[0-9]+&?)+$/i, 'the query string is : "$1".');
alert(str2); // 输出 'the query string is : "a=1&b=2&c=3".'
```

方法 `match` 是最常用的 `String` 方法，它接受一个正则表达式作为参数（如果不是正则表达式，则会被转换成正则表达式），返回的是一个包含匹配结果的数组。数组的第一个元素是匹配的子字符串，从第二个元素开始，是正则表达式的分组所匹配的内容。如果正则表达式被设置成了全局匹配模式，则返回的数组就是字符串中所有匹配的子字符串。`match` 方法的使用示例如下所示。

```
var str1 = 'http://www.robchen.cn/?action=guest';
var a = str1.match(/^http:VV([V]+V(\?)(.))*?$/);
// a = ['http://www.robchen.cn/?action=guest', 'www.robchen.cn', '?action=guest', '?', 'action=guest']
var str2 = 'a=1&b=2&c=3&d=4&e=5&f=6';
var b = str2.match(/[a-z]=[0-9]/g);
// b = ['a=1', 'b=2', 'c=3', 'd=4', 'e=5', 'f=6']
```

如果 `match` 作用于一个非全局模式的正则表达式，则返回的数组会有两个额外属性：`index` 和 `input`。`index` 属性包含的是所匹配的子字符串在原字符串中的位置，而 `input` 属性则保存了原字符串的一个副本，示例代码如下所示。

```
var str = 'a=1&b=2&c=3&d=4&e=5&f=6';
var a = str.match(/([a-z]+)=[0-9]+/);
alert(a.index); // a = ['a=1', 'a', '1']
alert(a.input); // 输出 0
// 输出 'a=1&b=2&c=3&d=4&e=5&f=6'
```

最后一个是 `split` 方法，在 3.2.5 一节中已经向读者介绍了接受一个字符串参数的 `split`。`split` 方法还可以接受一个正则表达式作为参数，如下所示。

```
var str = 'a=1&b=2+c=3|d=4@e=5%f=6';
var a = str.split(/&|+|@|%/); // a = ['a=1', 'b=2', 'c=3', 'd=4', 'e=5', 'f=6']
```

### 3.3.5 用于模式匹配的 RegExp 方法

`RegExp` 定义了两个用于模式匹配的方法，它们的行为和前面介绍的 `String` 方法很类似。第一个是 `exec` 方法，它与 `String` 对象的 `match` 方法很相似，只是它接受的参数是一个字符串，它会在此字符串中进行匹配查找，并返回一个数组。与 `match` 方法不同的是：无论正则表达式是否开启了全局匹配模式，`exec` 方法每次只匹配一个结果，返回的数组第一个元素是匹配的子字符串，从第二个元素开始是正则表达式中各个分组所匹配的内容。

只是当正则表达式开启了全局匹配模式时，每次成功匹配后，正则表达式的 `lastIndex` 属性都会被改写以标识此次匹配的子字符串的位置，在下次执行 `exec` 方法进行匹配时，会从 `lastIndex` 属性标识的位置开始查找。当没有任何匹配时，`lastIndex` 属性会被重置为 0，并返回 `null`。这个特性使得程序员可以反复匹配一个字符串从而得到所有匹配的子字符串的所有信息。下面是一个使用 `exec` 方法的示例。

```

var str = 'a=1&b=2+c=3|d=4@e=5%f=6';
var reg = /([a-z])=([0-9])/g;
var result;
while((result = reg.exec(str)) != null)
{
    document.write(result.join(',') + '<br/>');
}

```

程序运行结果如下所示。

```

a=1,a,1
b=2,b,2
c=3,c,3
d=4,d,4
e=5,e,5
f=6,f,6

```

另一个方法是 `test`，这个方法比较简单，它接受一个字符串作为参数，然后进行匹配检测。如果字符串能够被匹配，则返回 `true`，否则返回 `false`。例如，检查一个浮点数，代码如下所示。

```

var reg = /^(+|-)?(0|[1-9][0-9]*\.[0-9]*[1-9])$/;
reg.test('abc');           // 返回 false
reg.test('+123');          // 返回 true
reg.test('3.14');          // 返回 true
reg.test('-0.23');         // 返回 true

```

### 3.3.6 常用正则表达式

下面列出了一些常用的正则表达式，供读者参考学习。

- ❑ 匹配中文字符：`[\u4e00-\u9fa5]`
- ❑ 匹配双字节字符（包括汉字）：`[\x00-\xff]`
- ❑ 匹配 HTML 标记：`<(.*)>.*<\/1>|<(.*) \>\/`
- ❑ 匹配 E-mail 地址：`\/^w+([-.]w+)*@w+([-.]w+)*\.[w+([-.]w+)*$/i`
- ❑ 匹配 URL 地址：`\/^http:\/\/([w-]+\.)+[w-]+(\/[w- .?%&=]*)?$/i`
- ❑ 匹配字符串首尾的空格：`\/(^s*)(s*$)/`
- ❑ 匹配颜色代码：`^#[a-fA-F0-9]{6}/`
- ❑ 匹配身份证号码：`\/^d{15}(d{2}[xX0-9])?$/`
- ❑ 匹配电话号码：`\/^((d{3}\)|d{3}-)?(0d{2,3}|d{2,3}-)?[1-9]d{6,7}$/`
- ❑ 匹配整数：`\/^(+|-)?[1-9]d*$/`
- ❑ 匹配浮点数：`\/^(+|-)?(0|[1-9][0-9]*)(\.[0-9]*[1-9])?$/`

## 3.4 日期和时间：Date 对象

对时间的处理是程序设计中经常需要做的事情。在 JavaScript 中，时间由 `Date` 对象表示，一个 `Date`

对象表示一个日期和时间值。Date 对象提供了丰富的方法来对这些值进行操作。下面向读者介绍如何创建一个 Date 对象，以及 Date 对象的各种常用属性和方法。

### 3.4.1 Date 对象的创建

创建一个 Date 对象，需要使用 Date 构造函数。当不带参数调用 Date 构造函数时，返回的是一个表示当前时间的对象，如下所示。

```
var now = new Date();
```

当传入的参数为一个数字时，那么这个数字将作为日期的内部数字表示，其单位为毫秒（ms），如下所示。

```
var time = new Date(1192883138912);
```

当传入的参数为一个字符串时，那么这个字符串会被作为日期和时间的字符串表示，如下所示。

```
new Date('Sat, 20 Oct 2005 12:32:26 GMT');
```

字符串的格式必须遵循下面的规则：

- ❑ 短日期可以使用“/”或“-”作为日期分隔符，但是必须用月/日/年的格式来表示，例如“10/20/96”。
- ❑ 以“July 10 1995”形式表示的长日期中的年、月、日可以按任何顺序排列，年份值可以用两位数字表示，也可以用 4 位数字表示。如果使用两位数字来表示年份，那么该年份必须大于或等于 70。
- ❑ 括号中的任何文本都被视为注释。这些括号可以嵌套使用。
- ❑ 逗号和空格被视为分隔符。允许使用多个分隔符。
- ❑ 月和日的名称必须具有两个或两个以上的字符。如果两个字符所组成的名称不是独一无二的，那么该名称就被解析成最后一个符合条件的月或日。例如，Ju 被解释为七月而不是六月。
- ❑ 在所提供的日期中，如果所指定的星期几的值与按照该日期中剩余部分所确定的星期几的值不符合，那么该指定值就会被忽略。例如，尽管 1996 年 11 月 9 日实际上是星期五，Tuesday November 9 1996 也还是可以被接受并进行解析的。但是结果 Date 对象中包含的是 Friday November 9 1996。
- ❑ 符合全球标准时间（UTC）和格林威治标准时间（GMT）的格式。
- ❑ 小时、分钟和秒钟之间用冒号分隔，不需要指明的部分可以省略。“10:”、“10:11”和“10:11:12”都是有效的。
- ❑ 如果使用 24 小时计时的时钟，那么为中午 12 点之后的时间指定 PM 是错误的。例如“23:15 PM”就是错误的。
- ❑ 包含无效日期的字符串是错误的。例如，一个包含有两个年份或两个月份的字符串就是错误的。有效的参数示例如下所示。

```
Sat Oct 20 2007 20:34:56 GMT+0800  
Sat, 20 Oct 2007 12:34:56 GMT  
2007/10/20 12:34:56
```

当传入的参数为 2~7 个数字时，它们分别指出了日期和时间的各个组成部分，除了前两个字段（年和月）外，其他都是可选的。其语法如下所示。

```
new Date(year,month,day,hours,minutes,seconds,ms);
```

注意：month 参数的取值范围为 0~11，分别对应一年中的 12 个月份，即 month 参数是从 0 开始计算的。

### 3.4.2 读取和设置日期及时间的各个部分

当得到一个 Date 对象后，经常需要得到日期和时间的各个组成部分以进行进一步的操作和处理，这时可以使用下列方法。

- ❑ `getFullYear()`：返回对象中的年份部分，用 4 位数表示。
- ❑ `getMonth()`：返回对象中的月份部分（从 0 开始计算）。
- ❑ `getDate()`：返回对象所代表的一月中的第几天。
- ❑ `getDay()`：返回对象所代表的一周中的第几天。
- ❑ `getHours()`：返回对象中的小时部分。
- ❑ `getMinutes()`：返回对象中的分钟部分。
- ❑ `getSeconds()`：返回对象中的秒部分。
- ❑ `getMilliseconds()`：返回对象中的毫秒部分。
- ❑ `getTime()`：返回对象的内部毫秒表示。
- ❑ `getTimezoneOffset()`：返回对象所表示的日期的本地时间和 UTC 时间之间的时间差，以分钟为单位。

看下面一段程序。

```
var date = new Date(2007,9,20,12,03,30,444);
document.write('年份为' + date.getFullYear());
document.write('<br />');
document.write('月份为' + (parseInt(date.getMonth()) + 1));
document.write('<br />');
document.write('为当月的第' + date.getDate() + '天');
document.write('<br />');
document.write('为本周中的第' + date.getDay() + '天');
document.write('<br />');
document.write('小时部分为' + date.getHours());
document.write('<br />');
document.write('分钟部分为' + date.getMinutes());
document.write('<br />');
document.write('秒数部分为' + date.getSeconds());
document.write('<br />');
document.write('毫秒部分为' + date.getMilliseconds());
document.write('<br />');
document.write('内部毫秒数为' + date.getTime());
document.write('<br />');
document.write('与世界时间的时间差为' + date.getTimezoneOffset() + '分钟');
```



系统时区为东八(+8)区, 运行结果如下所示。

年份为 2007  
月份为 10  
为当月的第 20 天  
为本周中的第 6 天  
小时部分为 12  
分钟部分为 3  
秒数部分为 30  
毫秒部分为 444  
内部毫秒数为 1192853010444  
与世界时间的时间差为-480 分钟

在上述方法中, 除了 `getTime` 和 `getTimezoneOffset` 之外, 得到的都是本地时间, 如果要得到 UTC 时间(全球标准时间), 则可以在方法中相应的名词前加上 `UTC` 组成新的方法。将上面的例程做适当修改, 代码如下所示。

```
var date = new Date(2007,9,20,12,03,30,444);
document.write('年份为' + date.getUTCFullYear());
document.write('<br />');
document.write('月份为' + (parseInt(date.getUTCMonth()) + 1));
document.write('<br />');
document.write('为当月的第' + date.getUTCDate() + '天');
document.write('<br />');
document.write('为本周中的第' + date.getUTCDay() + '天');
document.write('<br />');
document.write('小时部分为' + date.getUTCHours());
document.write('<br />');
document.write('分钟部分为' + date.getUTCMinutes());
document.write('<br />');
document.write('秒数部分为' + date.getUTCSeconds());
document.write('<br />');
document.write('毫秒部分为' + date.getUTCMilliseconds());
document.write('<br />');
document.write('内部毫秒数为' + date.getTime());
```

运行结果如下所示。

年份为 2007  
月份为 10  
为当月的第 20 天  
为本周中的第 6 天  
小时部分为 4  
分钟部分为 3  
秒数部分为 30  
毫秒部分为 444  
内部毫秒数为 1192853010444

上述方法中, 如果将动词 `get` 改成 `set`, 则组成新的方法(除 `getDay` 方法和 `getTimezoneOffset` 方法外), 可以用来设置相应的日期部分, 这些方法如下。

- ☐ `set[UTC]Date()`: 设置对象的月中的某一天, 采用本地时间或者世界标准时间。
- ☐ `set[UTC]FullYear()`: 设置对象的年份部分, 采用本地时间或者世界标准时间。
- ☐ `set[UTC]Hours()`: 设置对象的小时部分, 采用本地时间或者世界标准时间。
- ☐ `set[UTC]Milliseconds()`: 设置对象的毫秒部分, 采用本地时间或者世界标准时间。
- ☐ `set[UTC]Minutes()`: 设置对象的分钟部分, 采用本地时间或者世界标准时间。
- ☐ `set[UTC]Month()`: 设置对象的月份部分, 采用本地时间或者世界标准时间。
- ☐ `set[UTC]Seconds()`: 设置对象的秒数部分, 采用本地时间或者世界标准时间。
- ☐ `setTime()`: 使用毫秒的形式设置对象的各个部分。

这些方法的使用示例如下所示。

```
var date = new Date();
date.setFullYear(2007);
date.setMonth(9);
date.setDate(20);
date.setHours(12);
date.setMinutes(03);
date.setSeconds(30);
date.setMilliseconds(444);
document.write('年份为' + date.getFullYear());
document.write('<br />');
document.write('月份为' + (parseInt(date.getMonth()) + 1));
document.write('<br />');
document.write('为当月的第' + date.getDate() + '天');
document.write('<br />');
document.write('为本周中的第' + date.getDay() + '天');
document.write('<br />');
document.write('小时部分为' + date.getHours());
document.write('<br />');
document.write('分钟部分为' + date.getMinutes());
document.write('<br />');
document.write('秒数部分为' + date.getSeconds());
document.write('<br />');
document.write('毫秒部分为' + date.getMilliseconds());
document.write('<br />');
document.write('内部毫秒数为' + date.getTime());
```

运行结果如下所示。

```
年份为 2007
月份为 10
为当月的第 20 天
为本周中的第 6 天
小时部分为 12
分钟部分为 3
秒数部分为 30
毫秒部分为 444
内部毫秒数为 1192853010444
```

setTime 方法的使用示例如下所示。

```
var date = new Date();
date.setTime(1192853010444);
document.write('年份为' + date.getFullYear());
document.write('<br />');
document.write('月份为' + (parseInt(date.getMonth()) + 1));
document.write('<br />');
document.write('为当月的第' + date.getDate() + '天');
document.write('<br />');
document.write('为本周中的第' + date.getDay() + '天');
document.write('<br />');
document.write('小时部分为' + date.getHours());
document.write('<br />');
document.write('分钟部分为' + date.getMinutes());
document.write('<br />');
document.write('秒数部分为' + date.getSeconds());
document.write('<br />');
document.write('毫秒部分为' + date.getMilliseconds());
document.write('<br />');
document.write('内部毫秒数为' + date.getTime());
```

运行结果如下所示。

```
年份为 2007
月份为 10
为当月的第 20 天
为本周中的第 6 天
小时部分为 12
分钟部分为 3
秒数部分为 30
毫秒部分为 444
内部毫秒数为 1192853010444
```

### 3.4.3 日期和时间的换算

在程序设计中，经常需要对日期和时间对象作换算，例如计算 5 天后是什么日期，或者 3 个小时后是几点钟。在 JavaScript 中，日期的换算需要搭配 get 类方法和 set 类方法同时完成，例如计算 5 天后的日期，代码如下所示。

```
var date = new Date();
date.setDate(date.getDate() + 5);
```

计算 3 个小时后的时间，代码如下所示。

```
var date = new Date();
date.setHours(date.getHours() + 3);
```

使用过 VBScript 的读者一定不会对 dateAdd 函数感到陌生，下面是用 JavaScript 来实现这个函数

功能的实例，一般的日期换算都可以利用这个函数来完成。

```
/*
 * 函数名: dateAdd
 * 参数说明:
 *     datepart:需要操作的时间部分，可选的值为
 *         'y':年数
 *         'm':月数
 *         'd':一月中的天数
 *         'w':一周中的天数
 *         'h':小时数
 *         'n':分钟数
 *         's':秒数
 *         'l':毫秒数
 *     number:改变的数量，为正整数或者负整数。
 *     date:被操作的 date 对象实例
 * 返回值: 被操作后的 date 对象
 */
function dateAdd(datepart,number,date)
{
    switch(datepart)
    {
        case 'y':
            date.setFullYear(date.getFullYear()+number);
            break;
        case 'm':
            date.setMonth(date.getMonth()+number);
            break;
        case 'd':
            date.setDate(date.getDate()+number);
            break;
        case 'w':
            date.setDate(date.getDate()+7*number);
            break;
        case 'h':
            date.setHours(date.getHours()+number);
            break;
        case 'n':
            date.setMinutes(date.getMinutes()+number);
            break;
        case 's':
            date.setSeconds(date.getSeconds()+number);
            break;
        case 'l':
            date.setMilliseconds(date.getMilliseconds()+number);
            break;
    }
    return date;
}
```



## 3.5 复杂的数学运算: Math 对象

基本的 JavaScript 运算符只能做一些简单的数学运算, 当需要做幂运算等高级数学运算时, 就需要使用 Math 对象。JavaScript 内置的 Math 对象提供了很多静态方法来完成复杂的数据运算, 下面介绍最常用的一些方法。

### 3.5.1 小数的取整

在程序设计中, 经常需要对一些小数进行取整操作, 这时可以使用 Math 对象的 ceil 和 floor 方法。ceil 方法提供了对一个浮点数的上舍入操作, 即返回的是一个大于或者等于原数的整数。floor 方法提供了对一个浮点数的下舍入操作, 即返回的是一个小于或者等于原数的整数。由于 Math 对象提供的都是静态方法, 所以调用其方法时需要使用如下格式。

Math.方法名(参数...);

这两个方法的使用示例如下所示。

```
var a = 3.14;
Math.ceil(a);      //返回 4
Math.floor(a);     //返回 3
var b = -12.4;
Math.ceil(b);      //返回 -12
Math.floor(b);     //返回 -13
```

### 3.5.2 得到随机数

随机数就是计算机随机产生的数字序列, 在实际应用中也会经常使用, 例如, 一个摇奖程序, 需要每次从 0~9 这 10 个数中随机抽取一个数, 这时可以使用 Math 对象的 random 方法。Math.random() 总是返回一个介于 0~1 之间的随机数, 通过对这个随机数的处理, 可以得到任意想要的随机数。例如, 随机得到 0~9 中的任意一个数, 代码如下所示。

```
//随机取得 0~9 之间的任意整数
function getRandom()
{
    var rnd = Math.random();           //生成一个随机数种子
    var number = Math.floor(rnd * 10); //将种子乘以 10 然后向下舍入, 得到一个大于等于 0 且小于 10 的整数
    return number;
}
//测试
var a = new Array();
for(var i = 0; i < 10; i++)
{
    a[i] = getRandom();
}
```

```
}  
document.write(a.join(','));
```

读者可以自己动手尝试一下，程序每次运行结果都会不同。

### 3.5.3 幂运算

当需要进行幂运算时，可以使用 `Math` 对象的 `pow` 方法。该方法接受两个参数，第一个参数为底数，第二个参数为幂数。例如，计算一个正方形的面积，代码如下所示。

```
//计算正方形的面积，参数 length 为边长，返回值是正方形的面积  
function getSquareArea(length)  
{  
    var area = Math.pow(length,2);    //计算面积，为边长的 2 次方  
    return area;  
}  
//测试  
var a1 = getSquareArea(12);           // a1 = 144  
var a2 = getSquareArea(33);           // a2 = 1089  
var a3 = getSquareArea(13.22);        // a3 = 174.7684
```

### 3.5.4 最大值和最小值

当需要在—组数中选出最大值或者最小值时，可以使用 `max` 方法和 `min` 方法。两个方法很类似，都接受 0 个或者多个数字作为参数，只是它们的返回值不同：`max` 方法返回参数列表中的最大值，而 `min` 方法返回参数列表中的最小值；如果参数数量为 0，`max` 方法则返回 `-Infinity`，而 `min` 方法返回 `Infinity`；如果有一个参数为 `NaN`，或者是不能转化为数字类型的其他数据类型，两个方法都返回 `NaN`。例如，对一个班级的某科考试成绩进行处理，得到最高分和最低分，代码如下所示。

```
var maximal = Math.max(90,84,100,99,73,58,82,64,79,93,78);    // maximal = 100  
var minimum = Math.min(90,84,100,99,73,58,82,64,79,93,78);    // minimum = 58
```

最高分为 100 分，而最低分为 58 分。

## 3.6 操作浏览器窗口：window 对象

在编写客户端脚本程序时，`window` 对象是一个最常用也最重要的对象之一，每个浏览器的窗口和框架都由 `window` 对象来表示。`window` 对象是客户端 JavaScript 的全局对象，也是客户端对象层次的根。`window` 对象定义了很多属性和方法，本节将向读者介绍如何使用这些属性和方法。

### 3.6.1 使用对话框

对话框允许程序向用户以对话窗口的方式呈现信息，并且能够根据用户的操作取得特定的返回值，

依次来完成最基本的交互功能。JavaScript 中的对话框有 3 种形式，分别对应着 window 对象的 3 个方法：alert、confirm 和 prompt。

alert 方法接受一个字符串作为参数，在被调用时，会在当前浏览器中打开一个小的对话窗口来显示所接受的字符串。这个窗口会有一个确定按钮，用户单击 OK 按钮会关闭对话窗口，在用户关闭对话窗口之前，对话窗口会始终保持焦点。例如，alert('I love Ajax')的效果如图 3.1 所示。

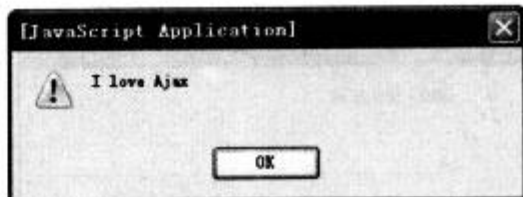


图 3.1 alert 对话框

注意：调用 window 对象的属性和方法时，可以省略“window.”，直接写属性名或者方法名。

对话框的外观会因为浏览器的种类和版本的不同而有一些细小的差别，但是基本功能不变。

confirm 方法同 alert 方法的工作方式很类似，同样接受一个字符串参数，并以一个对话窗口来呈现它。但是与 alert 方法不同的是，confirm 对话窗口会多一个“取消 (cancel)”的按钮，并且 confirm 方法会有返回值。当用户单击“确认 (OK)”按钮后，对话框消失，confirm 方法返回 true，而如果单击的是“取消 (Cancel)”按钮，则 confirm 方法返回 false。例如，在一个邮件系统中，当用户需要删除一封邮件时，一般会先提示用户是否确认删除该邮件，以免用户误操作，代码如下所示。

```
if(confirm('是否确认删除此邮件'))
{
    // do something
    alert('邮件已经被删除');
}
else
{
    // do something
    alert('您取消了该邮件的删除操作');
}
```

当程序运行时，显示的界面如图 3.2 所示。当用户单击 OK 按钮时，显示如图 3.3 所示。如果用户单击的是 Cancel 按钮，则显示如图 3.4 所示的界面。



图 3.2 confirm 对话框

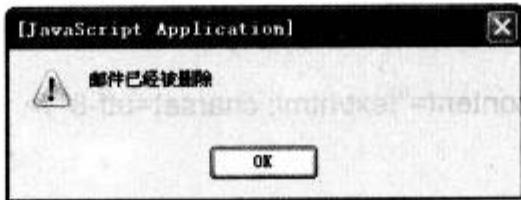


图 3.3 单击 OK 按钮后



图 3.4 单击 Cancel 按钮后

最后一个是 prompt 方法，它同样接受一个字符串参数。在 prompt 方法被调用时，会打开一个附有该字符串作为说明，带一个文本框以及确定、取消按钮的对话窗口。如果给 prompt 传递第二个参数，则该参数会作为对话框中文本框的默认值。prompt 方法同样有返回值，当用户单击“确定”按钮后，会把文本框中的内容作为返回值返回；而当用户单击“取消”按钮后，则会返回 null。例如，要求用户输入密码，如下所示。

```
var password = prompt('请输入您的密码',0);
alert(password);
```

```

</script>
</head>

<body>
<input type="button" id="btn" name="btn" value="click me" />
</body>
</html>

```

在上面的例子中，用户单击 click me 按钮后，此按钮会被禁用，然后在 5s 后恢复原状。其界面如图 3.9 所示。单击 click me 按钮后，界面如图 3.10 所示。在 5s 后，按钮恢复原状，如图 3.11 所示。

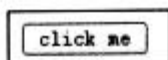


图 3.9 setTimeout 示例程序界面



图 3.10 单击 click me 按钮后

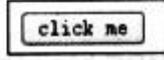


图 3.11 按钮恢复原状

与 setTimeout 方法不同的是，setInterval 方法是每次时间间隔后都会执行一次，除非被中断。它接受的参数与 setTimeout 一致：第一个参数为需要执行的代码片段、函数或者方法；第二个参数为每次执行的时间间隔，单位为毫秒。下面是一个利用 setInterval 函数来实现简单的计时器的例子。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>setTimeout demo</title>
<script>
var i = 0;
function refreshContainter()
{
    i++;
    document.getElementById('containter').innerHTML = i/100;
}
window.onload = function()
{
    t = window.setInterval(refreshContainter,1);
}
</script>
</head>

<body>
<span id="containter">0</span>
</body>
</html>

```

如图 3.12 所示，当页面被打开时，页面上会出现一个不断增加的数字，标识当前页面被打开的时间，以秒为单位。

当设置一个 timeout 或者 interval 时，可以将其保存在一个变量中，以后访问这个变量，就可以获得对这个 timeout 或者 interval 的引用，代码如下所示。

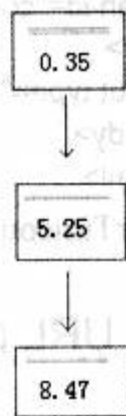


图 3.12 计时器界面效果



```
var timeout = window.setTimeout(function(){alert(1);},1000);
var interval = window.setInterval(function(){alert(2);},1000);
```

设置好 timeout 或者 interval 后, 如果取消这个设置, 停止延迟执行或者定时执行, 可以使用对应的 clearTimeout 和 clearInterval 方法, 代码如下所示。

```
window.clearTimeout(timeout);
window.clearInterval(interval);
```

现在修改上面计时器的例子, 给页面上添加一个按钮, 当单击该按钮时, 页面停止计时, 代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>setTimeout demo</title>
<script>
var i = 0;
function refreshContainter()
{
    i ++;
    document.getElementById('containter').innerHTML = i/100;
}
window.onload = function()
{
    var interval = window.setInterval(refreshContainter,1);
    document.getElementById('btnStop').onclick = function()
    {
        window.clearInterval(interval);
    }
}
</script>
</head>

<body>
<span id="containter">0</span>
<br />
<input type="button" id="btnStop" value="Stop" />
</body>
</html>
```

clearTimeout 用法与 clearInterval 一样, 这里不再赘述。

### 3.6.4 URL 的获取和跳转

window 对象提供了 location 属性来标识当前页面的 URL 地址。例如, 有一名为 demo.html 的页面, 代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>location demo</title>
<script>
alert(window.location);
</script>
</head>

<body>
</body>
</html>

```

其 URL 地址为 `http://localhost/demo.html`，访问页面时，效果如图 3.13 所示。

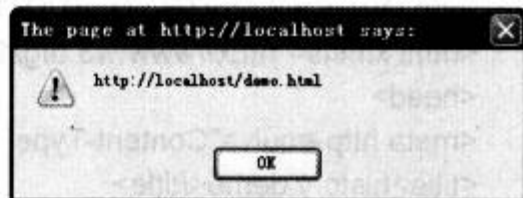


图 3.13 location 示例

`location` 属性其实本身也是一个对象，如果给这个对象的 `href` 属性赋值，当前页面就会发生跳转。在下面这个例子中，页面上有一个文本框和一个按钮，用户在文本框中输入想要到达的 URL 地址，然后单击该按钮，页面便会跳转到所填写的地址。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>location.href demo</title>
<script type="text/javascript">
function doJump()
{
    var url = document.getElementById('txtUrl').value;
    window.location.href = url;
}
</script>
</head>

<body>
<input type="text" id="txtUrl" />
<br />
<input type="button" id="btnJump" value="Jump" onclick="doJump();" />
</body>
</html>

```

### 3.6.5 历史记录的前进和后退

`window` 对象的 `history` 属性保存浏览器的历史记录。同 `location` 一样，`history` 本身也是一个对象，

它提供了 go、back 和 forward 3 种方法对历史记录进行操作。go 方法接受一个正整数或者负整数作为参数，其语法格式如下所示。

```
history.go(n);
```

当 n 为正整数时，浏览器会查找浏览记录队列中当前页面的第 n 项所指向的页面，并跳转。当 n 为负整数时，浏览器会查找浏览记录队列中当前页面的第前 n 项所指向的页面，并跳转。back 和 forward 方法则比较简单，其作用分别相当于 history.go(-1) 和 history.go(1)，即相当于单击浏览器上的后退和前进按钮的效果。下面是由 4 个 html 页面组成的程序演示了如何使用这些方法。

第一个页面 history\_demo\_1.html 的代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>history demo</title>
<script type="text/javascript">
function historyBack()
{
    history.back();
}
function historyForward()
{
    history.forward();
}
function historyGo()
{
    var n = parseInt(document.getElementById('txtHistpry').value);
    history.go(n);
}
</script>
</head>

<body>
<h1>History demo page 1</h1>
<p>
    <a href="history_demo_2.html">history_demo_2.html</a>
</p>
<input type="text" id="txtHistpry" /><input type="button" id="btnGo" value="history.go" onclick="historyGo();" />
<br />
<input type="button" id="btnBack" value="history.back" onclick="historyBack();" /><input type="button"
id="btnForward" value="history.forward" onclick="historyForward();" />
</body>
</html>
```

第二个页面 history\_demo\_2.html 的代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>history demo</title>
<script type="text/javascript">
function historyBack()
{
    history.back();
}
function historyForward()
{
    history.forward();
}
function historyGo()
{
    var n = parseInt(document.getElementById('txtHistpry').value);
    history.go(n);
}
</script>
</head>

<body>
<h1>History demo page 2</h1>
<p>
    <a href="history_demo_1.html">history_demo_1.html</a>
    <a href="history_demo_3.html">history_demo_3.html</a>
</p>
<input type="text" id="txtHistpry" /><input type="button" id="btnGo" value="history.go" onclick="historyGo();" />
<br />
<input type="button" id="btnBack" value="history.back" onclick="historyBack();" /><input type="button"
id="btnForward" value="history.forward" onclick="historyForward();" />
</body>
</html>

```

第三个页面 history\_demo\_3.html 的代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>history demo</title>
<script type="text/javascript">
function historyBack()
{
    history.back();
}

```



```
<body>
<h1>History demo page 4</h1>
<p>
  <a href="history_demo_1.html">history_demo_3.html</a>
</p>
<input type="text" id="txtHistpry" /><input type="button" id="btnGo" value="history.go" onclick="historyGo();" />
<br />
<input type="button" id="btnBack" value="history.back" onclick="historyBack();" /><input type="button"
id="btnForward" value="history.forward" onclick="historyForward();" />
</body>
</html>
```

### 3.6.6 控制窗口的大小和位置

window 对象提供了一组方法,使得 JavaScript 可以控制浏览器窗口的大小和位置:resizeTo、resizeBy、moveTo 和 moveBy。

#### 1. resizeTo 方法

resizeTo 可以将窗口改变为指定大小,它接受两个参数,分别指定窗口的宽和高。下面是一个使用 resizeTo 方法的示例。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>resizeTo demo</title>
<script type="text/javascript">
function resizeWindow()
{
  resizeTo(300,200);
}
</script>
</head>

<body>
<input type="button" id="btnResize" value="Resize window" onclick="resizeWindow();" />
</body>
</html>
```

程序界面如图 3.14 所示。单击 Resize window 按钮,窗口被调整至 300×200,如图 3.15 所示。

#### 2. resizeBy 方法

resizeBy 方法同样用于改变窗口大小,而与 resizeTo 所不同的是,resizeBy 方法是通过接受的两个参数,相对于原窗口的大小进行修改。如果参数的符号为正,则在原窗口的宽或高的基础上增加参数的值,如果参数的符号为负,则在原窗口的宽或高的基础上减少参数的值。下面是该方法的示例。



图 3.14 程序界面

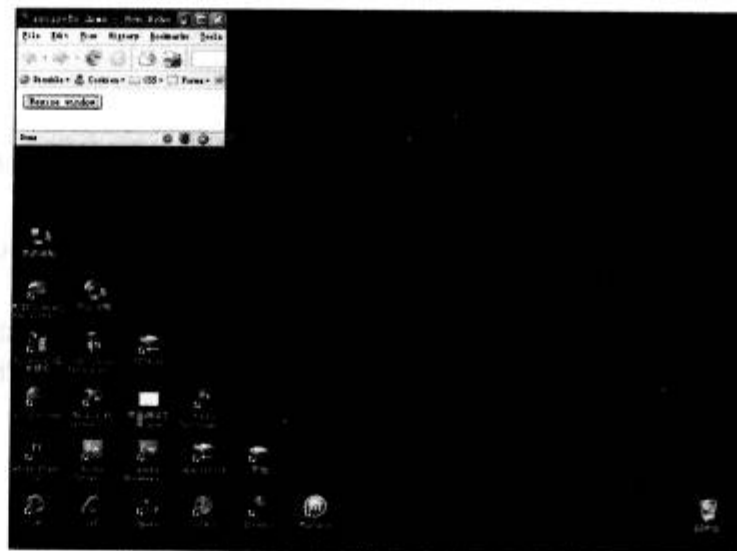


图 3.15 resizeTo 演示程序运行结果

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>resizeBy demo</title>
<script type="text/javascript">
function resizeWindow()
{
    resizeBy(-300,-200);
}
</script>
</head>

<body>
<input type="button" id="btnResize" value="Resize window" onclick="resizeWindow();" />
</body>
</html>

```

程序界面如图 3.16 所示。

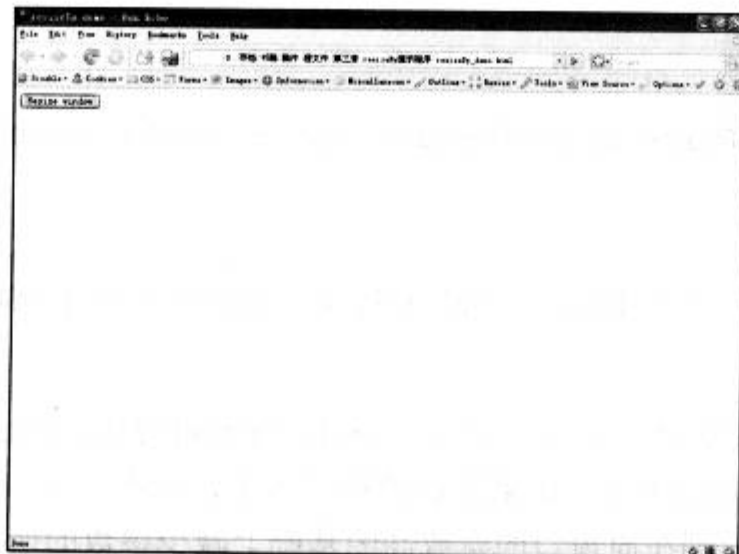


图 3.16 resizeBy 演示程序界面



图 3.19 moveTo 演示程序界面



图 3.20 moveTo 程序演示结果

#### 4. moveBy 方法

moveBy 方法与 resizeBy 方法类似，是在原窗口坐标的基础上进行修改。其使用方法见下面的演示程序。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>moveBy demo</title>
<script type="text/javascript">
resizeTo(300,200);
function moveWindow()
{
    moveBy(200,100);
}
</script>
</head>

<body>
<input type="button" id="btnMove" value="Move
window" onclick="moveWindow();" />
</body>
</html>
```

程序界面如图 3.21 所示。单击 Move window 按钮后，窗口向右移动 200 像素，向下移动 100 像素，如图 3.22 所示。当再次单击 Move window 按钮时，窗口再次被移动，如图 3.23 所示。

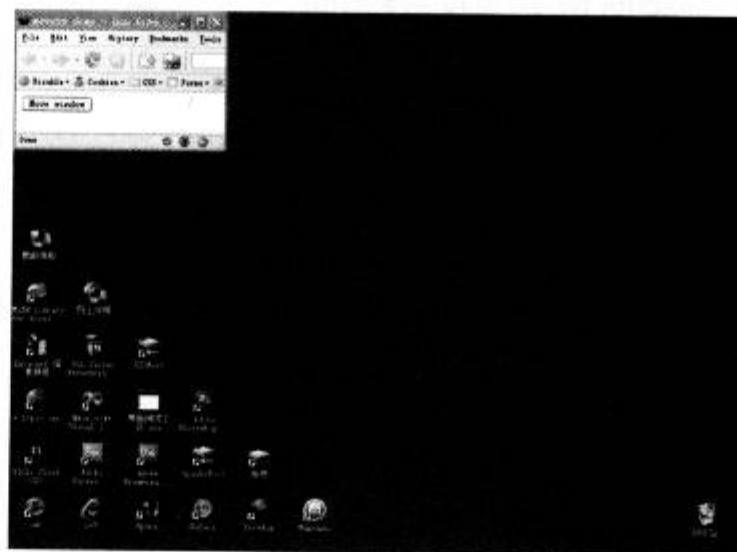


图 3.21 moveBy 演示程序界面



图 3.22 moveBy 演示程序结果（一）



图 3.23 moveBy 演示程序结果（二）

### 3.6.7 打开和关闭窗口

window 对象提供了 open 和 close 方法来让 JavaScript 控制窗口的打开和关闭。

open 方法可以打开一个新的浏览器窗口，返回的是这个新窗口的 window 对象。open 方法有 4 个可选参数：第一个参数是要被打开的窗口的 URL，如果这个参数被省略了，那么将会打开一个空窗口；第二个参数是新打开的窗口的名字，这个名字可以作为<a>标签或者<form>标签的 target 属性值，如果指定的是一个已经存在的窗口的名字，则 open 方法将使用这个已经存在的窗口而不是打开一个新窗口；第三个参数指明了窗口的一些特性，例如窗口的大小、窗口是否有地址栏、是否有状态栏等；第四个参数是一个布尔值，只有在 open 方法使用一个已经存在的窗口时才会生效，其作用是声明由第一个参数指定的 URL 是应该替换历史记录中的当前项，还是创建一个新项，默认为后者。open 方法返回的是新窗口的 window 对象，使得 JavaScript 可以获得对新窗口的引用，从而对新窗口进行操作。用 open 方法打开的窗口，如果是和原窗口在同一域内（即同一个域名下），则新窗口的 window 对象的 opener 属性保存了对原窗口的引用。一个完整的 open 方法使用示例如下所示。

```
var newWin = window.open('demo.html', 'demo', 'height=100, width=400, top=0, left=0, toolbar=no, menubar=no,
scrollbars=no, resizable=no, location=no, status=no', false);
```

对第三个参数的说明如表 3-7 所示。

表 3-7 窗口特性参数说明

height	窗口高度
width	窗口宽度
top	窗口距屏幕上方的距离
left	窗口距屏幕左侧的距离
toolbar	是否显示工具栏。yes 显示，no 不显示
menubar	是否显示菜单栏。yes 显示，no 不显示
scrollbars	是否显示滚动栏。yes 显示，no 不显示
resizable	是否允许改变窗口大小。yes 允许，no 不允许
location	是否显示地址栏。yes 显示，no 不显示
status	是否显示状态栏。yes 显示，no 不显示



close 方法比较简单, 该方法没有参数, 作用是关闭窗口。当关闭的是一个由 open 方法打开的窗口时, 窗口会直接被关闭。如果关闭的窗口不是由 open 方法打开的, 则浏览器会先显示一个对话框让用户确认是否关闭窗口, 如图 3.24 所示。

下面是一个用弹出窗口来模拟 confirm 对话框的演示程序, 程序中演示了如何使用 open 和 close 方法。主页面 main.html 的代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>main window</title>
<script type="text/javascript">
var subWindow = null;
function openDialogWindow(msg)           //打开新窗口, 并设置窗口的提示语
{
    subWindow = window.open('sub.html','subWindow','height=150, width=200, top=200, left=200, toolbar=no,
menubar=no, scrollbars=no, resizable=no,location=no, status=no',true);
    subWindow.onload = function()
    {
        subWindow.document.getElementById('label').innerHTML = msg;
    }
}
function showDialogResult(result)         //显示用户的操作结果
{
    document.getElementById('msgBox').innerHTML = 'You had click the "' + result + '" button.';
}
</script>
</head>

<body>
<input type="button" id="btnOpenWindow" value="Show dialog window" onclick="openDialogWindow('Confirm
Dialog Window');" />
<br />
<div id="msgBox"></div>
</body>
</html>
```

被打开页面 sub.html 的代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>sub window</title>
<script type="text/javascript">
function closeDialogWindow(result)       //显示用户的操作结果到父窗口并关闭当前窗口
{
```

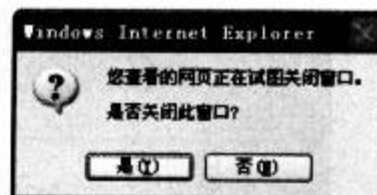


图 3.24 关闭窗口

```
        window.opener.showDialogResult(result);
        window.close();
    }
</script>
</head>

<body>
<div id="label"></div>
<br />
<input type="button" value="OK" id="btnOK" onclick="closeDialogWindow('OK');" />
<input type="button" value="Cancel" id="btnCancel" onclick="closeDialogWindow('Cancel');" />
</body>
</html>
```

打开 main.html，界面如图 3.25 所示。



图 3.25 main.html 界面

单击 Show dialog window 按钮后，显示子窗口，如图 3.26 所示。



图 3.26 显示子窗口

单击 OK 按钮，子窗口关闭，原窗口出现提示文字，显示用户单击的是哪一个按钮，如图 3.27 所示。同样，如果单击 Cancel 按钮，效果如图 3.28 所示。

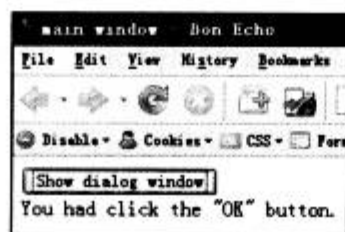


图 3.27 单击 OK 按钮

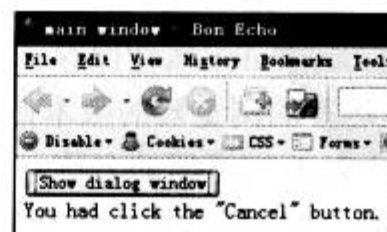


图 3.28 单击 Cancel 按钮

### 3.6.8 获得焦点和失去焦点

调用 window 对象的 focus 方法, 可以使窗口获得焦点; 而调用 blur 方法, 则可以让窗口失去焦点。在调用 focus 方法时, 系统会将窗口移动到窗口堆栈的顶部, 使窗口可见。在使用 open 方法打开新窗口时, 浏览器会自动在窗口堆栈顶部创建窗口, 但如果 open 方法第二个参数指定的窗口名已经存在, open 方法并不会自动让那个窗口可见, 这时就可以使用 focus 方法。

### 3.6.9 取得用户显示器的信息

window 对象的 screen 属性是一个记录了客户端显示器信息的对象, 它提供有关显示器大小和可用颜色数量的信息。属性 availWidth 和 availHeight 是实际可用的显示器大小, 它们排除了诸如 windows 任务栏等所占用的空间, 也可以理解为由显示器提供给浏览器使用的空间大小。属性 colorDepth 指定可以显示的颜色数以 2 为底的对数。下面的程序利用 screen 对象创建了一个宽 400 像素、高 200 像素的窗口, 这个窗口相对于显示器水平和垂直居中。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>screen demo</title>
<script type="text/javascript">
var width = screen.availWidth;
var height = screen.availHeight;
window.resizeTo(400,200);
window.moveTo(width / 2 - 200 , height / 2 - 100);
</script>
</head>

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

运行结果如图 3.29 所示。



图 3.29 水平和垂直居中的窗口

### 3.6.10 取得用户浏览器的信息

window 对象的 navigator 属性保存了客户端浏览器的总体信息。navigator 是一个对象, 它主要有 5 个属性来描述正在运行的浏览器的版本信息。

- ❑ appName: Web 浏览器的名称。
- ❑ appVersion: 浏览器的版本号和其他版本信息。这里的版本号是浏览器的内部版本号, 所以并不一定与显示给用户的版本号一致。

- ❑ **userAgent**: 浏览器进行 HTTP 请求时, 在 USER-AGENT 头信息中所包含的数据, 通常包含 **appName** 和 **appVersion** 中的信息。
- ❑ **appName**: 浏览器的代码名。Netscape 和 IE 都是用 Mozilla 作为这一属性的值。
- ❑ **platform**: 运行浏览器的硬件平台。

下面是一个显示用户浏览器相关信息的例程。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>navigator demo</title>
</head>
<body>
<script type="text/javascript">
document.write('appName:' + navigator.appName);
document.write('<br />');
document.write('appVersion:' + navigator.appVersion);
document.write('<br />');
document.write('userAgent:' + navigator.userAgent);
document.write('<br />');
document.write('appName:' + navigator.appCodeName);
document.write('<br />');
document.write('platform:' + navigator.platform);
</script>
</body>
</html>
```

## 3.7 操作 HTML 文档: document 对象

**document** 对象是 **window** 对象的属性, 由于其重要性, 所以在此节中单独向读者介绍。每个 **window** 对象都有 **document** 属性, 该属性引用的是窗口中显示的 HTML 文档的 **Document** 对象。在之前的许多演示程序中, 读者已经接触了 **document.write** 方法, 除此之外, **document** 对象还提供了一些描述 HTML 文档的整体信息的属性, 已经针对 HTML 文档进行操作的方法。下面向读者介绍 **document** 对象最常用的属性和方法。

### 3.7.1 文档的输出

**document** 对象提供了 **write** 方法使得 JavaScript 可以附加文本到当前打开的文档中, 在之前的许多示例中读者已经接触过 **write** 方法的使用。**document** 对象还提供了一个与 **write** 方法类似的方法: **writeln**。**writeln** 方法的使用和 **write** 方法一样, 所不同的是会在输出的内容后面自动加上一个换行符。



注意: write 方法和 writeln 方法只能在文档被解析时使用。如果在文档解析完成后调用 write 或 writeln 方法, 则输出的内容会覆盖当前文档的内容。

### 3.7.2 文档的标题

document 对象的 title 属性描述了位于 HTML 文档中<title>和</title>中间的内容, 即文档的标题, JavaScript 可以读取和设置这个值。例如, 下面的示例通过操作 title 属性来改变浏览器的标题。

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

<body>
<script type="text/javascript">
function modifyDocumentTitle()           //修改文档标题并更新页面内容
{
    var title = document.getElementById('txtTitle').value;
    document.title = title;
    document.getElementById('title').innerHTML = title;
}
document.write('<h1 id="title">' + document.title + '</h1>');
</script>
<br />
<input type="text" id="txtTitle" />
<input type="button" id="btnEdit" value="modify document title" onclick="modifyDocumentTitle();" />
</body>
</html>
```

程序界面如图 3.30 所示。程序首先读取文档的 title 内容, 并输出在文档中, 然后提供一个文本框和一个按钮让用户修改文档的 title 内容。在文本框中输入 “I love Ajax” 并单击 modify document title 按钮, 效果如图 3.31 所示。

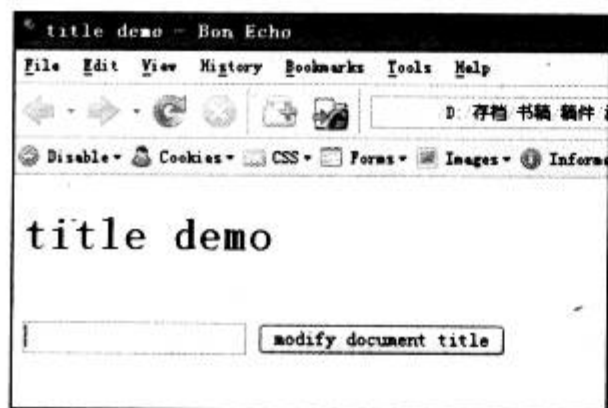


图 3.30 document.title 演示程序界面



图 3.31 document.all 程序结果演示

可以看到，文档的标题和文档中的内容都已经被修改为 I love Ajax。

### 3.7.3 文档的图像

document 对象的 images 属性是一个数组，保存了对当前 HTML 文档中所有图像的引用。在下面的演示程序中，通过遍历 images 数组来给文档中所有的图片加上边框。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>document.images demo</title>
<script type="text/javascript">
function addImageBorder()
{
    for(var i = 0; i < document.images.length; i++)
    {
        document.images[i].style.border = '1px solid #000'; //style 是一个对象，提供了对 HTML 元素 CSS 样
        式的操作接口
    }
}
</script>
</head>

<body>
<p>
    
    
    
    
</p>
<p>
    <input type="button" id="btnAddBorder" value="Add image border" onclick="addImageBorder()" />
</p>
</body>
</html>
```

程序界面如图 3.32 所示。单击 Add image border 按钮后，页面上所有图片都被加上 1 像素宽的黑色边框，如图 3.33 所示。

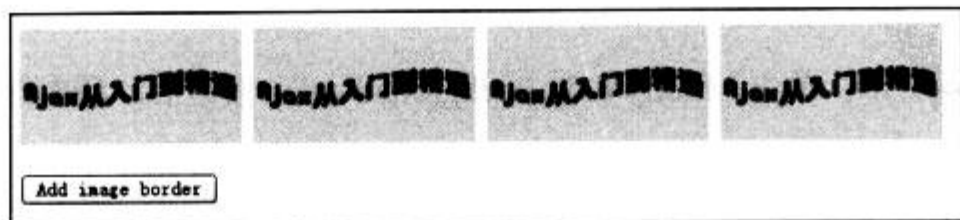


图 3.32 document.images 演示程序界面

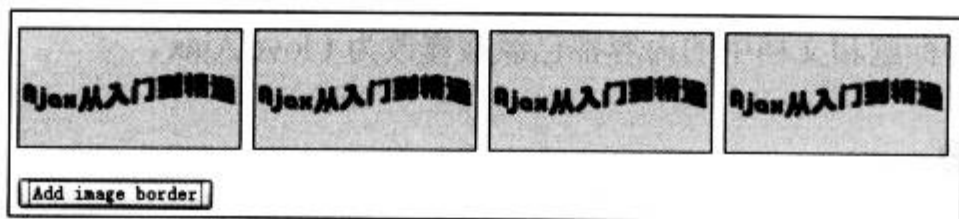


图 3.33 document.images 程序结果演示

### 3.7.4 文档的超链接

同 images 属性类似，links 属性也是一个数组，它保存了文档中所有超链接的引用。下面的例子演示了如何给一个 HTML 文档中所有的超链接加上黄色的背景色，演示代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>document.links demo</title>
<script type="text/javascript">
function modifyLinksBgColor()
{
    for(var i = 0; i < document.links.length; i++)
    {
        document.links[i].style.backgroundColor = '#FF0';
    }
}
</script>
</head>

<body>
<p><a href="http://www.robchen.cn">Ajax 从入门到精通 </a></p>
<p><a href="http://www.robchen.cn">Ajax 从入门到精通 </a></p>
<p><a href="http://www.robchen.cn">Ajax 从入门到精通 </a></p>
<p><a href="http://www.robchen.cn">Ajax 从入门到精通 </a></p>
<p><a href="http://www.robchen.cn">Ajax 从入门到精通 </a></p>
<br />
<input type="button" id="btnModify" value="Modify links background color" onclick="modifyLinksBgColor()" />
</body>

</html>
```

程序界面如图 3.34 所示。单击 Modify links background color 按钮后，所有超链接被加上黄色的背景色，如图 3.35 所示。

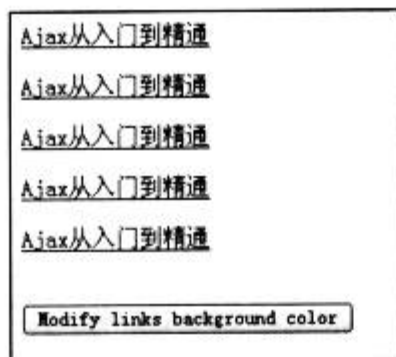


图 3.34 document.links 演示程序



图 3.35 document.links 程序结果演示

### 3.7.5 文档的表单

document 对象的 forms 属性是一个数组，保存了当前文档中所有表单的引用。表单是 HTML 中最重要的元素之一，是用户与服务器进行数据交互的重要工具。下面的例子演示了如何使用 forms 属性来重置页面中所有的表单，代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>document.forms demo</title>
<script type="text/javascript">
function resetAllForms()
{
    for(var i = 0; i < document.forms.length; i++)
    {
        document.forms[i].reset();
    }
}
</script>
</head>

<body>
<h1>Registracton</h1>
<h2>Account</h2>
<form name="fmAccount" id="fmAccount" method="post">
    <label for="txtUsername">Username</label>
    <br />
    <input type="text" id="txtUsername" name="txtUsername" />
    <br />
    <label for="txtPassword">Password</label>
    <br />
    <input type="password" id="txtPassword" name="txtPassword" />
    <br />
</form>
<h2>Contact</h2>
```



```
<form name="fmContact" id="fmContact" method="post">
  <label for="txtEmail">Email</label>
  <br />
  <input type="text" id="txtEmail" name="txtEmail" />
  <br />
  <label for="txtAddress">Address</label>
  <br />
  <input type="text" id="txtAddress" name="txtAddress" />
  <br />
</form>
<br />
<input type="button" id="btnReset" value="Reset all" onclick="resetAllForms()" />
</body>
</html>
```

程序界面如图 3.36 所示。先向两个表单中填写数据，如图 3.37 所示。然后单击 Reset all 按钮，两个表单的数据都被重置了，如图 3.38 所示。

Registration

Account

Username  
Password

Contact

Email  
Address

Reset all

图 3.36 document.forms 演示程序

Registration

Account

Username  
robin  
Password  
\*\*\*\*\*

Contact

Email  
robchen@126.com  
Address  
China

Reset all

图 3.37 填写数据

Registration

Account

Username  
Password

Contact

Email  
Address

Reset all

图 3.38 重置表单数据

## 3.8 应用实例

前面几节，向读者介绍了 JavaScript 常用的对象及其方法和属性。本节将向读者展示两个综合运用这些知识的实例，一个是在网页上显示自动更新的日期和时间，另外一个则是一个简单的小游戏 Lucky Seven。

### 3.8.1 在网页上显示自动更新的日期和时间

本实例通过操作 Date 对象及利用 setInterval 方法，来制作在网页上自动更新的日期和时间。代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>date demo</title>
<script type="text/javascript">
function getCurrentTimeString()
{
    var currentDate = new Date();
    var year = currentDate.getFullYear();
    var month = currentDate.getMonth() + 1;
    var date = currentDate.getDate();
    var hour = currentDate.getHours();
    var minutes = currentDate.getMinutes();
    var seconds = currentDate.getSeconds();
    var timeString = year + '-' + month + '-' + date + ' ' + hour + ':' + minutes + ':' + seconds;
    return timeString;
}
function refreshTime()
{
    document.getElementById('time').innerHTML = getCurrentTimeString(); //元素的 innerHTML 属性可以读
    取和设置元素内的 HTML 文本内容
}
window.onload = function()
{
    setInterval(refreshTime,1000);
}
</script>
</head>

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

```

### 3.8.2 一个简单的小游戏：Lucky Seven（幸运7）

Lucky Seven 是一种最早来源于街头游戏厅的游戏。Lucky Seven 包含着一组图，开始游戏后，在 3 个容器中将这组图快速地轮换显示。玩家依次按下 3 个按钮，来停止 3 个图形容器的滚动。当 3 个容器中的图片都停止后，如果图片都是 7，就会中奖并得到奖金。下面这个实例是使用 JavaScript 来简单地实现这个游戏。

主要页面 lucky\_seven.html 的代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>lucky seven</title>
<link type="text/css" rel="stylesheet" href="style.css" />
<script type="text/javascript" src="luckyseven.js"></script>
<script type="text/javascript">
window.onload = function()
{
    init();
}
</script>
</head>

<body>
<div id="box1" class="box">1</div>
<div id="box2" class="box">2</div>
<div id="box3" class="box">3</div>
<div class="clear"></div>
<div id="msg" style="display:none;"></div>
<input type="button" id="btnStart" value="Start" />
<input type="button" id="btnStop1" value="Stop" />
<input type="button" id="btnStop2" value="Stop" />
<input type="button" id="btnStop3" value="Stop" />
</body>
</html>

```

样式表文件 style.css 的代码如下所示。

```

@charset "utf-8";
/*
 *   Lucky Seven Style
 *   Author: Robin Chen ( robchen@126.com )
 */
*{
    margin:0;
    padding:0;
}
.clear {
    clear:both;
}
.box {
    width:100px;
    border:1px solid black;
    float:left;
    margin:10px;
    background-color:#eee;
    font-size:36px;
    text-align:center;
    line-height:36px;
    padding:5px 0;
}

```

```
}
#box1 {
    margin-left:50px;
}
input {
    border:1px solid #666;
}
#msg {
    border:1px solid #069;
    background-color:#FF0;
    width:460px;
    margin:10px 0;
}
#btnStop1 {
    margin-left:35px;
}
#btnStop2 {
    margin-left:75px;
}
#btnStop3 {
    margin-left:75px;
}
```

JavaScript 文件 luckyseven.js 的代码如下所示。

```
/*
 *   Lucky Seven
 *   Author: Robin Chen ( robchen@126.com )
 */

/*
 *   取得一个 1~9 之间的随机数
 */
function getRandom()
{
    var rnd = Math.floor(Math.random() * 10);
    return rnd;
}

/*
 *   开始游戏,设置定时期定时更新 3 个容器的内容,并绑定按钮事件处理
 */
function startGame()
{
    document.getElementById('btnStart').disabled = true;
    document.getElementById('msg').style.display = 'none';
    interval1 = setInterval(function()
    {
        document.getElementById('box1').innerHTML = getRandom();
    },1);
    interval2 = setInterval(function()
    {

```

```

        document.getElementById('box2').innerHTML = getRandom();
    },1);
    interval3 = setInterval(function()
    {
        document.getElementById('box3').innerHTML = getRandom();
    },1);
    document.getElementById('btnStop1').onclick = function()
    {
        clearInterval(interval1);
        interval1 = null;
        check();
    }
    document.getElementById('btnStop2').onclick = function()
    {
        clearInterval(interval2);
        interval2 = null;
        check();
    }
    document.getElementById('btnStop3').onclick = function()
    {
        clearInterval(interval3);
        interval3 = null;
        check();
    }
}
/*
 * 检查是否 3 个容器都已经停止滚动
 */
function check()
{
    if(interval1 == null && interval2 == null && interval3 == null)
    {
        endGame();
    }
}
/*
 * 结束游戏, 判断是否为 3 个 7, 如果是, 则显示中奖信息, 如果不是, 提示未中奖
 */
function endGame()
{
    if(document.getElementById('box1').innerHTML == '7' && document.getElementById('box2').innerHTML
    == '7' && document.getElementById('box3').innerHTML == '7')
    {
        document.getElementById('msg').style.display = 'block';
        document.getElementById('msg').innerHTML = 'Lucky Seven!!!';
    }
    else
    {
        document.getElementById('msg').style.display = 'block';
    }
}

```



```
        document.getElementById('msg').innerHTML = 'Bad Luck!!!';
    }
    document.getElementById('btnStart').disabled = false;
}
/*
 * 初始化程序
 */
function init()
{
    document.getElementById('btnStart').onclick = function()
    {
        startGame();
    }
}
```

程序界面如图 3.39 所示。单击 Start 按钮，3 个容器内的数字开始快速变换。单击容器下的 Stop 按钮，可以停止相应容器内数字的变换。当 3 个容器都停止动作时，游戏结束并显示结果，如图 3.40 所示。

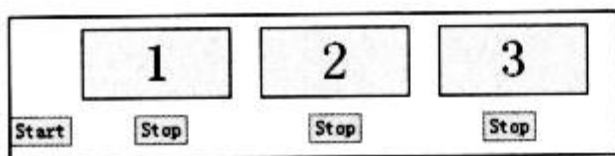


图 3.39 Lucy Seven 程序界面

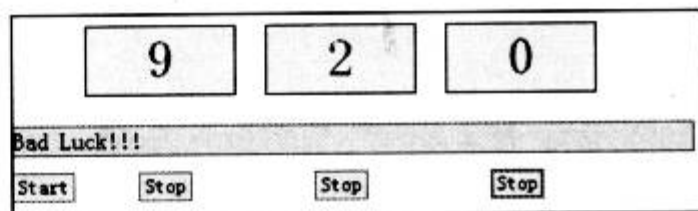


图 3.40 程序运行结果

### 3.9 小 结

本章主要向读者介绍了 JavaScript 的一些常用对象，其中包括数组对象（Array）、字符串对象（String）、正则表达式对象（RegExp）、日期和时间对象（Date）、数学运算对象（Math）、浏览器窗口对象（window）和文档对象（document），并辅以实例详细介绍了其常用的属性和方法。最后，向读者展示了两个综合应用这些知识的实例。

```
document.getElementById(msg).innerHTML = 'Bad Luck!!!';
```

```
document.getElementById(btnStart).disabled = false;
```

• 初始设置

```
function init()
```

```
{ document.getElementById(btnStart).onclick = function()
```

```
{ startGame();
```

图 3.39 显示，单击 Start 按钮，3 个容器内的数字开始快速变换。单击容器下的 Stop 按钮，可以停止相应容器内数字的变换。当 3 个容器都停止动作时，游戏结束并显示结果，如图 3.40 所示。



图 3.40 游戏运行结果



图 3.39 Lucky Seven 游戏界面

### 3.9 小结

本章主要向大家介绍了 JavaScript 的一些常用对象，其中包括数组对象 (Array)、字符串对象 (String)、正则表达式对象 (RegExp)、日期和时间对象 (Date)、数学运算对象 (Math)、浏览器窗口对象 (Window) 和文档对象 (Document)，并分别详细地介绍了其常用的属性及方法。最后，向大家展示了两个综合应用这些知识的实例。

# 第 4 章

## DOM 文档对象模型介绍

- » 基本概念
- » 结点的引用
- » 结点的操作
- » 控制元素的样式
- » 事件处理
- » 应用实例
- » 小结

DOM 是 Document Object Module 的缩写，即文档对象模型。DOM 是表示文档、访问和操作文档元素的应用程序接口 (API)，所有支持 JavaScript 的 Web 浏览器都支持 DOM。本书中介绍的 DOM 实际上是指 W3C DOM，即由 World Wide Web 委员会定义的标准文档对象模型，它包含了传统 Web 浏览器所实现的 DOM 模型的所有特性。DOM 支持对 HTML 及 XML 的操作。按照 DOM 的作用对象来划分，可以把它分为如下 3 个模块。

- ☐ 核心模块：提供基本的功能，定义了对 HTML 和 XML 文档的基本操作。如查找结点、操作结点等。
- ☐ HTML DOM：定义了针对 HTML 操作的功能。
- ☐ XML DOM：定义了针对 XML 操作的功能。

本章主要介绍核心模块和 HTML DOM 的知识，关于 XML DOM 将会在第 10 章向读者介绍。

## 4.1 基本概念

本节向读者介绍 DOM 的基本概念，包括 DOM 的体系结构（树形结构）、结点及其组成部分、结点的类型以及结点之间的关系。

### 4.1.1 树形结构

在 DOM 中，HTML 文档的层次结构被表示为一个树形结构。树的根结点是一个表示当前 HTML 文档的 Document 对象，树的每个子结点表示 HTML 文档中的不同内容。参考下面的 HTML 文档。

```
<html>
<head>
<title>demo</title>
</head>

<body>
  <h1>topic</h1>
  <p>something <strong>important</strong></p>
</body>
</html>
```

用树形图表达其结构，如图 4.1 所示。

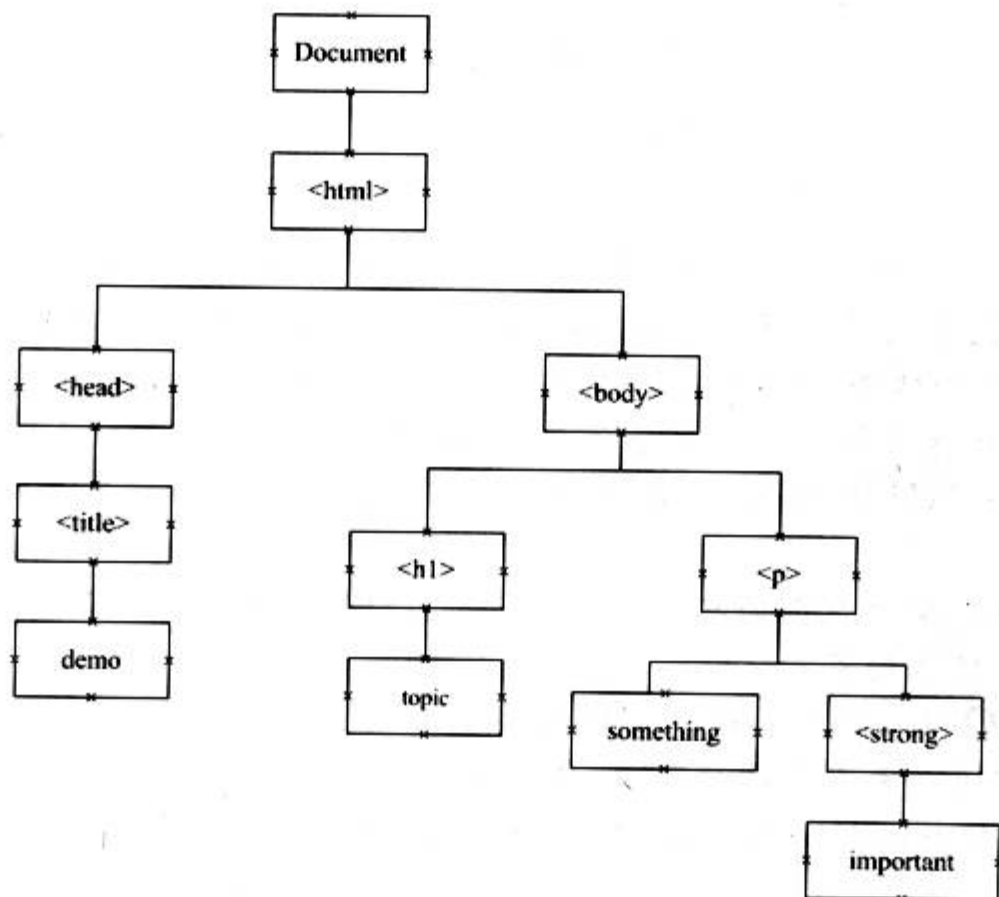


图 4.1 树形结构图



4.1.2 结点的类型和组成

每个结点都由一个 Node 对象表示，Node 对象提供了 `nodeType` 属性来表示结点的类型。DOM 为不同类型的结点提供了相应的接口，当知道一个结点为某种类型时，则可以使用相应的接口所定义的属性和方法。常见的结点类型及其 `nodeType` 值如表 4-1 所示。

表 4-1 常见结点类型和 `nodeType` 值

接 口	<code>nodeType</code> 值	说 明
Element	1	元素
Attr	2	属性
Text	3	文本
Comment	8	注释
Document	9	文档
DocumentFragment	11	文档片段

HTML 中最常见的就是元素结点，一个元素结点由标签名、属性结点和文本结点组成，例如一个表示链接的元素。

```
<a href="http://www.robchen.cn">my site</a>
```

其中 `a` 是标签名，`href="http://www.robchen.cn"` 是属性结点，`href` 是属性名，`http://www.robchen.cn` 是其属性值，`my site` 是文本结点。

4.1.3 结点之间的关系

结点与结点之间通常有 3 种关系：父子关系、兄弟关系和祖孙关系。在图 4.1 中，结点 `<html>` 是结点 `<head>` 和结点 `<body>` 的父结点，`<head>` 和 `<body>` 结点是 `<html>` 结点的子结点，而 `<head>` 和 `<body>` 互为兄弟关系。同样，`<body>` 结点是 `<h1>` 和 `<p>` 结点的父结点，`<h1>` 和 `<p>` 结点是 `<body>` 结点的子结点，`<h1>` 和 `<p>` 结点互为兄弟结点。一个结点的父结点以上级别的结点，称为这个结点的祖先结点，这个结点称为祖先结点的子孙结点。例如 `<html>` 结点是 `<p>` 结点的祖先结点，`<p>` 结点是 `<html>` 结点的子孙结点。DOM 为 Node 对象提供了一组属性来表达这些关系，使得程序可以非常方便地获得对结点的引用。关于这些属性的知识将在 4.2 节中向读者介绍。

4.2 结点的引用

需要对一个结点做相应操作时，首先需要获得对这个结点的引用。DOM 定义了大量的属性和方法可以使程序方便地获得对目标结点的引用。下面向读者一一介绍。



### 4.2.1 根据 id 属性引用结点

在 HTML 中, 可以给结点添加一个 id 属性, 从而通过 document 对象的 getElementById 方法来查找拥有指定 id 属性值的结点。参考下面的程序。

```
<!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>demo</title>
<script type="text/javascript">
window.onload = function()                                //页面加载完成后执行函数体内的代码
{
    var topic = document.getElementById('topic');         //获取 id 为 topic 的结点的引用
    alert(topic.innerHTML);                               //弹出对话框显示该结点内的 HTML 内容
}
</script>
</head>

<body>
    <h1 id="topic">topic</h1>
    <p>something <strong>important</strong></p>
</body>
</html>
```

程序中, 在页面加载完成后, 首先通过 getElementById 方法获取了 id 为 topic 的结点的引用, 然后用对话框显示该结点内的 HTML 内容。效果如图 4.2 所示。

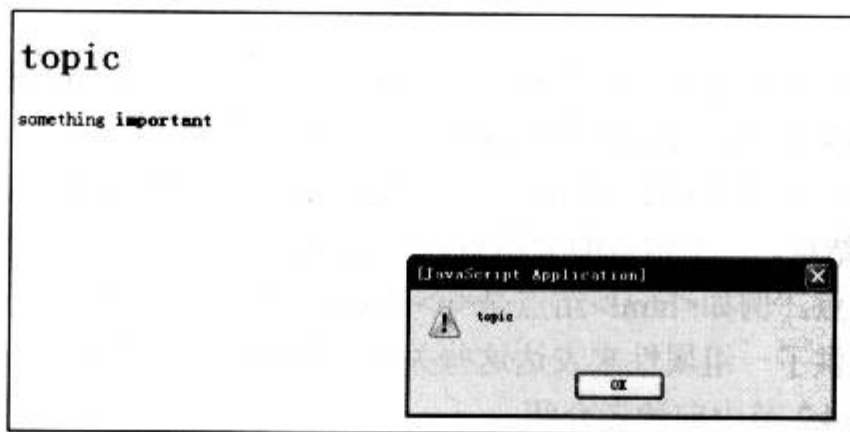


图 4.2 getElementById 演示程序

引用结点必须在结点被加载完成后才允许, 否则会发生错误。由于浏览器解析 HTML 代码是按照书写顺序由上而下进行的, 所以引用结点的 JavaScript 程序必须书写在该结点在 HTML 文档中书写位置的后面。如果引用结点的程序存放在函数或方法中, 则调用该函数或方法的代码必须位于该结点的后面或者在结点加载完后的某个事件的处理程序中。例如在上例中, 就是利用了 window 对象的 onload 事件, 这个事件在页面所有元素被加载完成后触发, 这样就保证了所引用的结点对象已经是加载完成的。关于事件的知识将会在本章的 4.5 节中向读者介绍。

### 4.2.2 根据 name 属性引用结点

通过 document 对象的 `getElementsByName` 方法可以取得文档中所有具有指定 `name` 属性的结点的集合, 该方法返回的是一个数组。例如, 在读取一个表单数据时, 可以通过该方法取得表单中所有 `checkbox` 控件并读取选中控件的值, 参考下面的实例。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>getElementsByName demo</title>
<script type="text/javascript">
function showCheckedData()
{
    var elms = document.getElementsByName('demo');
    var values = [];
    for(var i = 0; i < elms.length; i++)
    {
        if(elms[i].checked)
        {
            values.push(elms[i].value);
        }
    }
    alert(values.join(','));
}
</script>
</head>

<body>
<form action="" method="post">
<label>1:<input type="checkbox" name="demo" value="1" /></label>
<label>2:<input type="checkbox" name="demo" value="2" /></label>
<label>3:<input type="checkbox" name="demo" value="3" /></label>
<label>4:<input type="checkbox" name="demo" value="4" /></label>
<label>5:<input type="checkbox" name="demo" value="5" /></label>
<br />
<input type="button" value="show checked data" onclick="showCheckedData();" />
</form>
</body>
</html>
```

程序界面如图 4.3 所示。选择其中的一些复选框, 如图 4.4 所示。

图 4.3 `getElementsByName` 演示程序

图 4.4 选择复选框

单击 show checked data 按钮, 程序会获取所有 name 属性为 demo 的结点, 这里是全部的 checkbox。然后遍历取得的所有元素, 并判断该 checkbox 是否被选中。如果当前 checkbox 被选中, 则将 checkbox 的 value 属性值添加到 values 数组中。最后输出所有被选中的值, 如图 4.5 所示。

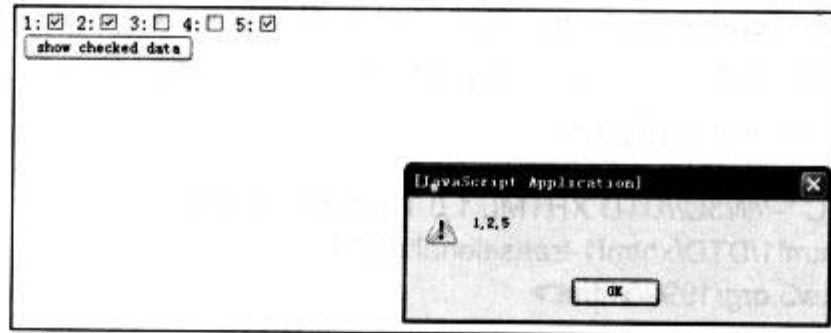


图 4.5 输出所有被选中的值

### 4.2.3 根据标签名引用结点

Node 对象提供了 `getElementsByTagName` 方法来查找所有标签名与给定参数一致的下属结点, 该方法返回一个数组。在介绍 document 对象的 links 属性时, 已经向读者介绍过一个改变文档中所有链接背景色的示例, 现在用 `getElementsByTagName` 方法来重写这个示例, 代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>getElementsByTagName demo</title>
<script type="text/javascript">
function modifyLinksBgColor()
{
    var links = document.body.getElementsByTagName('a');
    for(var i = 0; i < links.length; i++)
    {
        links[i].style.backgroundColor = '#FF0';
    }
}
</script>
</head>

<body>
<p><a href="http://www.robchen.cn">Ajax 从入门到精通 </a></p>
<p><a href="http://www.robchen.cn">Ajax 从入门到精通 </a></p>
<p><a href="http://www.robchen.cn">Ajax 从入门到精通 </a></p>
<p><a href="http://www.robchen.cn">Ajax 从入门到精通 </a></p>
<p><a href="http://www.robchen.cn">Ajax 从入门到精通 </a></p>
<br />
<input type="button" id="btnModify" value="Modify links background color" onclick="modifyLinksBgColor()" />
</body>

</html>
```

其中 `document.body` 是一个特殊的属性，特制 HTML 文档中的 `body` 元素。程序界面如图 4.6 所示。单击 `Modify links background color` 按钮后，所有链接的背景色变为黄色，如图 4.7 所示。

图 4.6 `getElementsByTagName` 演示程序

图 4.7 程序演示结果

#### 4.2.4 引用父结点

Node 对象提供了 `parentNode` 属性来引用当前结点的父结点。在下面的示例中，程序给页面所有的 `li` 时间注册了 `click` 事件的处理程序，单击 `li` 元素，则在指定的 `div` 中显示父结点的 `id`，代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>parentNode demo</title>
<script type="text/javascript">
window.onload = function()
{
    var lis = document.body.getElementsByTagName('li');
    for(var i = 0; i < lis.length; i++)
    {
        lis[i].onclick = function()
        {
            document.getElementById('msg').innerHTML = 'the current node's parent is : <strong>' +
this.parentNode.id + '</strong>';           // this 指针指向当前对象，在这里指向 lis[i]
        }
    }
}
</script>
<style type="text/css">
li {cursor:pointer;}
#msg {background-color:#FF0;}
</style>
</head>

<body>
<h1>parentNode demo</h1>
<h2>Navigation</h2>
```

```

<ul id="navigation">
  <li>Home</li>
  <li>News</li>
  <li>Products</li>
  <li>GuestBook</li>
</ul>
<h3>Sub Navigation</h3>
<ul id="subNavigation">
  <li>World News</li>
  <li>Company News</li>
</ul>
<div id="msg"></div>
</body>
</html>

```

程序界面如图 4.8 所示。当单击 Navigation 下的 Home、News、Products、GuestBook 时，在页面下方出现提示文字，显示当前结点的父结点 id 为 navigation，如图 4.9 所示。当单击 Sub Navigation 下的 World News 和 Company News 时，提示当前结点的父结点 id 为 subNavigation，如图 4.10 所示。

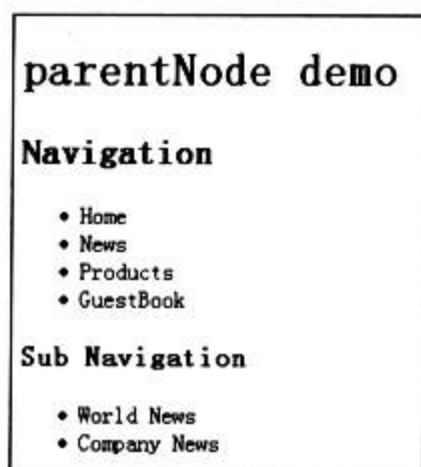


图 4.8 parentNode 演示程序

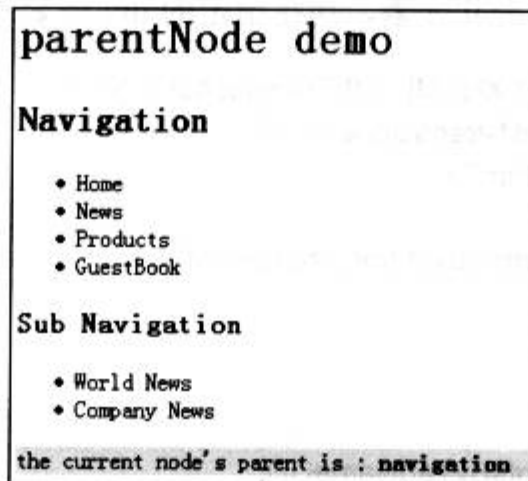


图 4.9 父结点 id 为 navigation

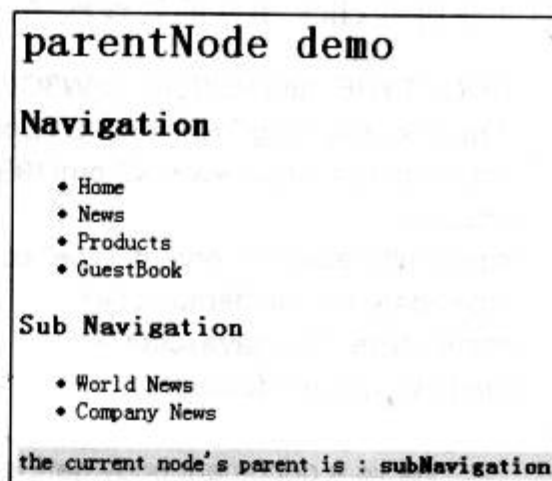


图 4.10 父结点 id 为 subNavigation

## 4.2.5 引用子结点

Node 对象提供了 3 个属性来引用其直属子结点，分别是 `childNodes`、`firstChild` 和 `lastChild`。`childNodes` 属性来引用其所有的直属子结点。`firstChild` 属性等于 `childNodes` 返回的元素集合中的第一个元素。`lastChild` 属性等于 `childNodes` 返回的元素集合中的最后一个元素。在下面的示例中，首先获取文档中的 `ul` 元素，然后通过 `firstChild`、`lastChild` 和 `childNodes` 属性给第一个 `li` 结点和最后一个 `li` 结点以及剩下的其他结点设置 3 种不同的背景色。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>child node demo</title>

```



```

<script type="text/javascript">
window.onload = function()
{
    var ul = document.getElementById('parent');
    ul.firstChild.style.backgroundColor = '#FF0';
    ul.lastChild.style.backgroundColor = '#F00';
    for(var i = 1; i < ul.childNodes.length - 1; i++)
    {
        ul.childNodes[i].style.backgroundColor = '#eee';
    }
}
</script>
</head>

<body>
    <ul id="parent">
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
    </ul>
</body>
</html>

```

在 IE 下其运行结果如图 4.11 所示。

但是在 gecko 核心的浏览器下，例如 Firefox，则得不到预期效果。因为 gecko 核心的浏览器会自动添加一些空白的文本结点来填充页面中的空白部分。所以在 gecko 内核的浏览器下，ul 的直属子元素就不仅仅是 li 元素，也包含了一些空白的文本结点。为了达到一样的效果，将上面的示例做一些修改，修改后的代码如下所示。

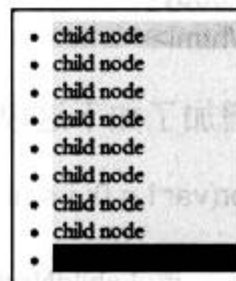


图 4.11 IE 下运行结果

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>child node demo</title>
<script type="text/javascript">
window.onload = function()
{
    var ul = document.getElementById('parent');
    for(var i = 0; i < ul.childNodes.length; i++)
    {

```

```

        if(ul.childNodes[i].nodeType == 3)
            ul.childNodes[i].parentNode.removeChild(ul.childNodes[i]);
    }
    ul.firstChild.style.backgroundColor = '#FF0';
    ul.lastChild.style.backgroundColor = '#F00';
    for(var i = 1; i < ul.childNodes.length - 1; i++)
    {
        ul.childNodes[i].style.backgroundColor = '#eee';
    }
}
</script>
</head>

<body>
    <ul id="parent">
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
    </ul>
</body>
</html>

```

其中增加了如下这段程序。

```

for(var i = 0; i < ul.childNodes.length; i++)
{
    if(ul.childNodes[i].nodeType == 3)
        ul.childNodes[i].parentNode.removeChild(ul.childNodes[i]);
}

```

其作用是删除 ul 的直属子结点中结点类型为 3 的结点，即文本结点，这样就达到了在 IE 下一样的效果。

#### 4.2.6 引用相邻的结点

Node 对象的 previousSibling 和 nextSibling 属性保存了结点的上一个和下一个兄弟结点的引用。在下面的示例中，通过给 li 结点定义事件处理程序，使得当鼠标划过 li 结点时，li 结点本身的背景色变为红色，其相邻两个 li 结点的背景色变为黄色，当鼠标划离 li 元素时，回复原样。示例代码如下。

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

```

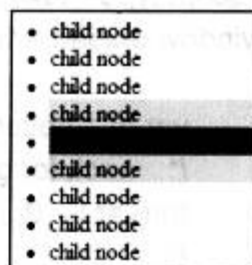
```

<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>brother child demo</title>
<script type="text/javascript">
window.onload = function()
{
    var ul = document.getElementById('parent');
    /* hack for gecko */
    for(var i = 0; i < ul.childNodes.length; i++)
    {
        if(ul.childNodes[i].nodeType == 3)
            ul.childNodes[i].parentNode.removeChild(ul.childNodes[i]);
    }
    for(var i = 0; i < ul.childNodes.length; i++)
    {
        ul.childNodes[i].onmouseover = function()    // 定义当鼠标划过时需要执行的程序
        {
            if(this.previousSibling)                // this 指针指向当前对象, 这里指 ul.childNodes[i]
            {
                this.previousSibling.style.backgroundColor = '#ff0';
            }
            if(this.nextSibling)
            {
                this.nextSibling.style.backgroundColor = '#ff0';
            }
            this.style.backgroundColor = '#f00';
        }
        ul.childNodes[i].onmouseout = function()    // 定义当鼠标划离时需要执行的程序
        {
            if(this.previousSibling)                // this 指针指向当前对象, 这里指 ul.childNodes[i]
            {
                this.previousSibling.style.backgroundColor = '#fff';
            }
            if(this.nextSibling)
            {
                this.nextSibling.style.backgroundColor = '#fff';
            }
            this.style.backgroundColor = '#fff';
        }
    }
}
</script>
</head>

<body>
    <ul id="parent">
        <li>child node</li>
        <li>child node</li>

```

```
<li>child node</li>
<li>child node</li>
<li>child node</li>
<li>child node</li>
<li>child node</li>
<li>child node</li>
<li>child node</li>
</ul>
</body>
</html>
```



效果如图 4.12 所示。

图 4.12 演示程序效果

## 4.3 结点的操作

4.2 节向读者介绍了各种获得结点引用的方法，本节向读者介绍针对结点的基本操作。DOM 提供了丰富的方法来支持对结点的基本操作，即创建、添加、修改和删除结点。本节将会辅以实例向读者详细介绍。

### 4.3.1 创建元素结点

当需要创建一个元素结点时，可以使用 `document` 对象的 `createElement` 方法。该方法接受一个标识需要创建的元素标签名的字符串参数，返回对被创建的结点的引用。例如，创建一个 `div` 元素。

```
document.createElement('div');
```

创建一个 `span` 元素。

```
document.createElement('span');
```

当使用 `createElement` 方法创建元素后，元素并没有被立即加入到当前的 DOM 树中，而是被存放在内存中。只有再使用添加结点的相关方法进行操作，才能真正将元素加入 DOM 树中。关于添加结点的方法，将会在后面几节中向读者介绍。

### 4.3.2 创建文本结点

使用 `createTextNode` 方法可以创建一个文本结点，该方法接受一个字符串作为创建的文本结点的文本值。示例代码如下。

```
document.createTextNode('It is a text node');
```

同 `createElement` 方法一样，使用 `createTextNode` 方法创建的文本结点并没有立即被添加到 DOM 树中，而是需要调用添加结点的相关方法。

### 4.3.3 添加结点

Node 对象提供了 `appendChild` 方法来将程序创建的结点添加到父结点的直属子结点列表的末尾。该方法也可作用于已经存在于 DOM 树中的结点，执行方法后会改变结点在 DOM 树中的位置，而不是插入一个新的结点。其语法格式如下所示。

```
node.appendChild(newNode);
```

现在结合本节中介绍的创建和添加结点的方法，来制作一个简单的无刷新留言本程序。当然这不是一个真正的留言本，它没有与后台程序和数据库的数据交互，只是模拟客户端的效果，代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>create & add node demo</title>
<style type="text/css">
*{
    margin:0;
    padding:0;
}
html {
    background-color:#eee;
    height:100%;
}
body {
    padding:15px;
    font-size:11px;
    width:500px;
    background-color:#fff;
    height:100%;
    font-family:Tahoma;
    border-left:20px solid #ccc;
}
ul {
    list-style:none;
    border-top:1px solid #999;
    height:350px;
    overflow-x:auto;
    overflow-y:scroll;
}
span {
    font-weight:bold;
    font-size:12px;
}
```



```

li {
    border-bottom: 1px dashed #666;
    line-height: 20px;
}
form {
    margin-top: 10px;
    border-top: 1px solid #999;
}
label {
    display: block;
    line-height: 20px;
    font-weight: bold;
    cursor: pointer;
    background-color: #999;
    color: #fff;
    margin: 3px 0;
    padding-left: 5px;
    width: 100%;
}
#txtName, #txtContent {
    width: 100%;
    font-size: 11px;
}
#btnSubmit {
    display: block;
    margin-top: 3px;
    border: 1px solid #666;
    padding: 2px 5px;
    width: 100%;
}
</style>
<script type="text/javascript">
function submitMsg()
{
    var name = document.getElementById('txtName').value;
    var content = document.getElementById('txtContent').value;
    var span = document.createElement('span');
    var nameText = document.createTextNode(name);
    span.appendChild(nameText);
    var p = document.createElement('p');
    var contentText = document.createTextNode(content);
    p.appendChild(contentText);
    var li = document.createElement('li');
    li.appendChild(span);
    li.appendChild(p);
    document.getElementById('msgList').appendChild(li);
}
</script>
</head>

```

//取得 name 值  
 //取得 message 值  
 //创建 span 结点  
 //创建文本结点, 文本值为 name 值  
 //将文本结点添加到 span 结点中  
 //创建 p 结点  
 //创建文本结点, 文本值为 message 值  
 //将文本结点添加到 p 结点中  
 //创建 li 结点  
 //将 span 结点添加到 li 结点中  
 //将 p 结点添加到 li 结点中  
 //将 li 结点添加到 msgList 结点中

```

<body>
  <h1>Guest Book</h1>
  <ul id="msgList">
    <li>
      <span>Robin Chen</span>
      <p>Welcome, My friends.</p>
    </li>
  </ul>
  <form name="fmMsg" id="fmMsg" action="?" method="post">
    <h2>Message</h2>
    <label for="txtName">name</label>
    <input name="txtName" type="text" id="txtName" value="guest" onfocus="this.select();" />
    <label for="txtContent">Message</label>
    <textarea name="txtContent" rows="4" id="txtContent" onfocus="this.select();">something to
say...</textarea>
    <input type="button" value="Click here to submit your message!" id="btnSubmit"
onclick="submitMsg();" />
  </form>
</body>
</html>

```

程序界面如图 4.13 所示。在表单中填写名字和信息内容，如图 4.14 所示。

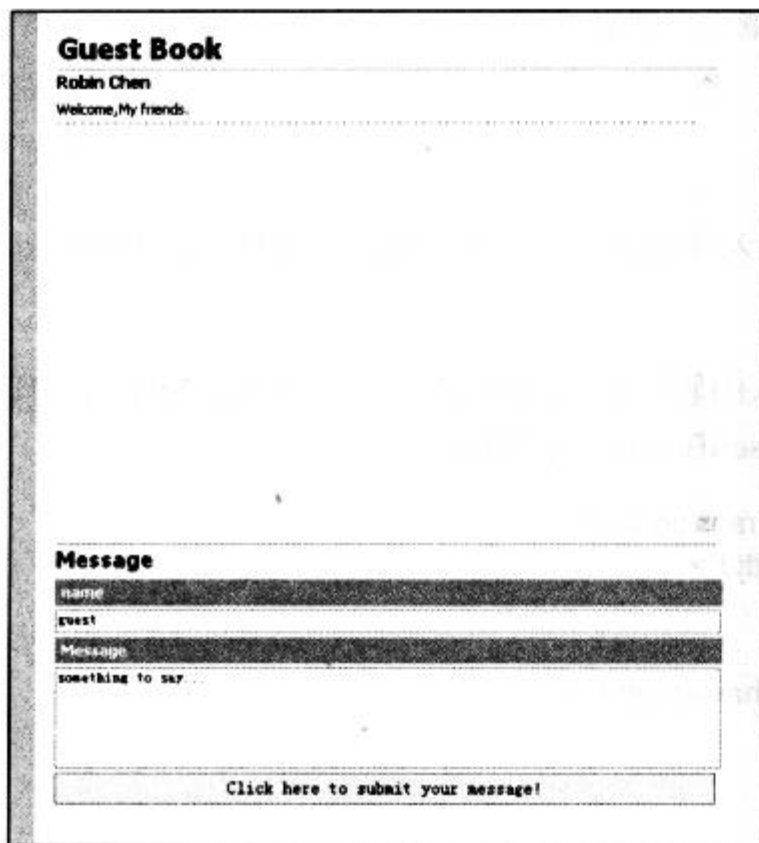


图 4.13 Guest Book 程序界面

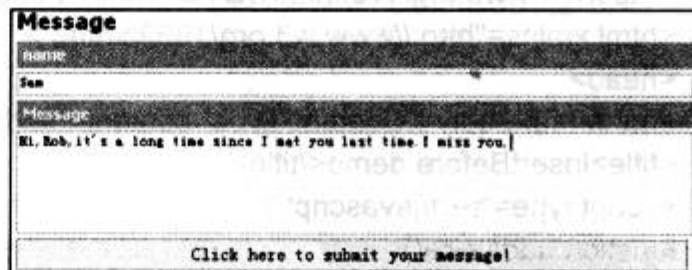


图 4.14 填写留言表单

填写完成后，单击下面的 Click here to submit your message! 按钮提交留言，则所填写的留言立即被显示在上面的列表中，如图 4.15 所示。

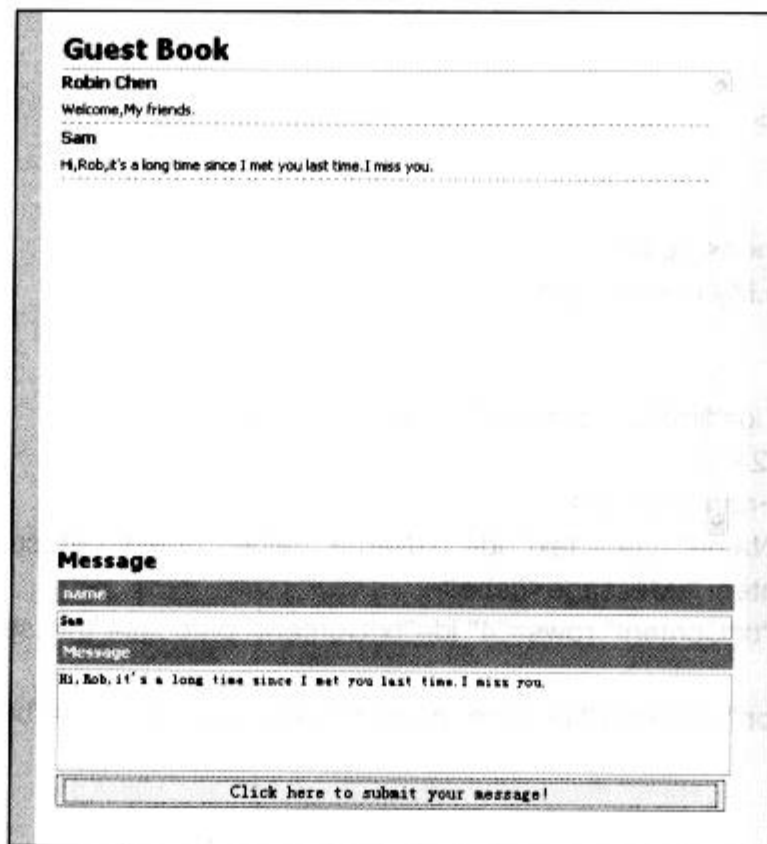


图 4.15 程序运行结果

这里并不是真正的提交留言，而只是模拟了客户端的效果，这是典型的 Ajax 用户体验。一个功能完整的、包含更多特性的 Ajax 留言本，将会在第 7 章中向读者呈现。

#### 4.3.4 插入子结点

Node 对象提供了 `insertBefore` 方法来将新结点插入到指定子结点的前面，语法格式如下所示。

```
parentNode.insertBefore(newNode,childNode);
```

其中 `parentNode` 指的是父结点，`newNode` 是需要插入的新结点，`newNode` 会被作为 `parentNode` 的直属子结点，插入 `childNode` 的前面。下面是一个使用 `insertBefore` 方法的示例。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>insertBefore demo</title>
<script type="text/javascript">
function addNode()
{
    var pos = parseInt(document.getElementById('txtPos').value) || 0;
    var li = document.createElement('li');
    var text = document.createTextNode('new node');
    li.appendChild(text);
    li.style.backgroundColor = '#ff0';
    var ul = document.getElementById('parent');
```

```

var lis = ul.getElementsByTagName("li");
if(pos >= lis.length)
{
    alert('error index');
    return;
}
ul.insertBefore(li,lis[pos]);
}
</script>
</head>

<body>
    <ul id="parent">
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
    </ul>
    <label for="txtPos">node position:</label>
    <input type="text" id="txtPos" />
    <input type="button" value="add" onclick="addNode();" />
</body>
</html>

```

程序界面如图 4.16 所示。

在文本框中输入需要插入的位置（从 0 开始），然后单击 add 按钮，程序会创建一个文本值为 new node、拥有黄色背景色的新 li 结点插入到 ul 元素中，其位置为文本框中输入的位置，如图 4.17 和图 4.18 所示。

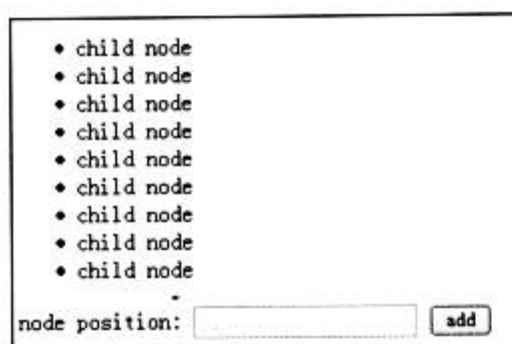


图 4.16 程序界面

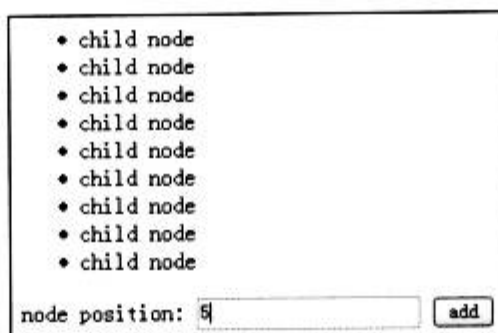


图 4.17 指定插入位置

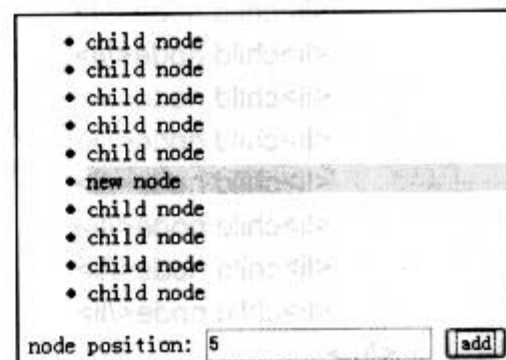


图 4.18 插入结点

### 4.3.5 替换子结点

Node 对象提供了 replaceChild 方法来使用一个新的结点替换一个子结点，语法格式如下所示。

```
parentNode.replaceChild(newNode,childNode);
```

其中 parentNode 为父结点, newNode 为新结点, childNode 为要被替换的子结点。现在来修改 4.3.4 小节的示例,使得新结点不直接插入,而是替换指定位置的旧结点,代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>replaceChild demo</title>
<script type="text/javascript">
function replaceNode()
{
    var pos = parseInt(document.getElementById('txtPos').value) || 0;
    var li = document.createElement('li');
    var text = document.createTextNode('new node');
    li.appendChild(text);
    li.style.backgroundColor = '#ff0';
    var ul = document.getElementById('parent');
    var lis = ul.getElementsByTagName('li');
    if(pos >= lis.length)
    {
        alert('error index');
        return;
    }
    ul.replaceChild(li,lis[pos]);
}
</script>
</head>

<body>
    <ul id="parent">
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
        <li>child node</li>
    </ul>
    <label for="txtPos">node position:</label>
    <input type="text" id="txtPos" />
    <input type="button" value="replace" onclick="replaceNode();" />
</body>
</html>
```

程序界面如图 4.19 所示。在文本框中输入“3”，然后单击 replace 按钮，效果如图 4.20 所示。



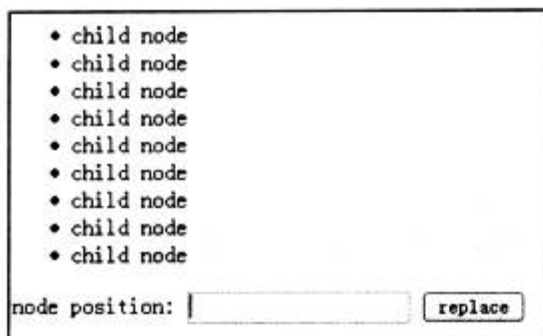


图 4.19 程序界面

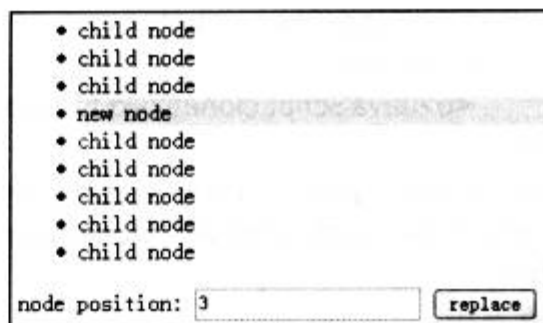


图 4.20 替换结点

### 4.3.6 复制结点

Node 对象提供了 cloneNode 方法来得到 Node 对象的一个副本。cloneNode 方法接受一个布尔值参数，来标识返回的结点副本中是否包含原结点的子结点。当参数为 true 时，则包含子结点；当参数为 false 时，则不包含子结点。下面是一个使用 cloneNode 方法的示例，代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>cloneNode demo</title>
<script type="text/javascript">
function clone()
{
    var p = document.getElementsByTagName('p')[0];
    var newNode = p.cloneNode(false);
    document.getElementById('container').appendChild(newNode);
}
function cloneWithChildNodes()
{
    var p = document.getElementsByTagName('p')[0];
    var newNode = p.cloneNode(true);
    document.getElementById('container').appendChild(newNode);
}
</script>
<style type="text/css">
p {
    line-height:20px;
    background-color:#ff0;
    height:20px;
    width:400px;
}
</style>
</head>
<body>
```

```

<h1>cloneNode demo</h1>
<div id="containter">
  <p>JavaScript cloneChild function demo</p>
</div>
<input type="button" value="clone node without child nodes" onclick="clone();" />
<input type="button" value="clone node with child nodes" onclick="cloneWithChildNodes();" />
</body>
</html>

```

为了方便演示，程序中给 p 标签设置了黄色的背景色和固定的高宽。程序界面如图 4.21 所示。单击 clone node without child nodes 按钮，会复制一个 p 结点插入到页面中。新结点不包含原 p 结点的子结点，所以显示的是一个空的色块，如图 4.22 所示。

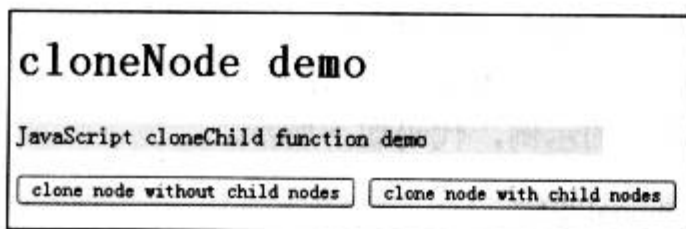


图 4.21 cloneNode 演示程序界面

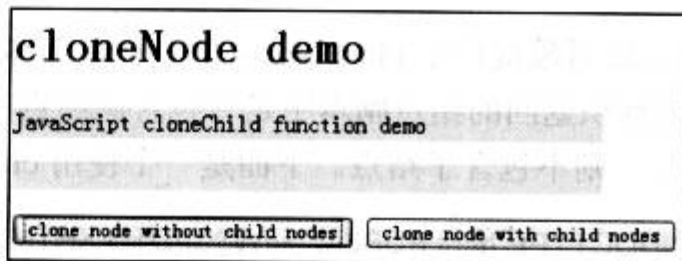


图 4.22 不包含子结点

单击 clone node with child nodes 按钮，会复制一个包含原 p 结点的子结点的 p 结点插入到页面中，效果如图 4.23 所示。

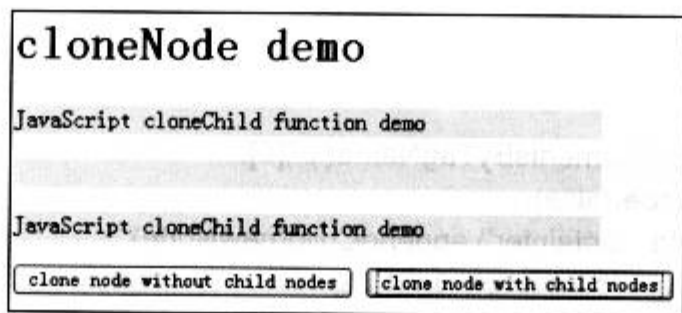


图 4.23 包含子结点

### 4.3.7 删除子结点

Node 对象提供了 removeChild 方法来删除一个直属子结点，该方法接受一个参数，为需要参数的子结点的引用。基本语法如下所示。

```
parentNode.removeChild(childNode);
```

当将 removeChild 方法结合 parentNode 属性使用时，可以更方便地删除一个结点，例如有一结点 oNode，可以使用以下方法来删除这个结点。

```
oNode.parentNode.removeChild(oNode);
```

下面的示例，通过模拟一个信息列表管理程序的客户端效果，来向读者展示 removeChild 方法的使用，其代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>removeChild demo</title>
<style type="text/css">
* {
    margin:0;
    padding:0;
}
body {
    font-size:12px;
    font-family:Tahoma;
    padding-left:20px;
}
ul {
    list-style:none;
    border:1px solid #666;
    background-color:#eee;
    width:400px;
    padding:10px;
    margin:10px 0;
}
li {
    width:100%;
    border-bottom:1px dashed #666;
    padding:5px 0;
}
.btn {
    float:right;
}
</style>
<script type="text/javascript">
/*
 * 删除选中的结点
 */
function deleteSelectedItems()
{
    var cbs = document.getElementsByName('cb');
    var list = document.getElementById('list');
    var lis = list.getElementsByTagName('li');
    var deletetltems = []; //需要删除的结点的集合
    for(var i = 0; i < cbs.length; i++)
    {
        /*
         * 如果 li 中的 checkbox 被选中, 则将该 li 结点加入到 deletetltems 数组中
         */
        if(cbs[i].checked)
        {
            deletetltems.push(lis[i]);
        }
    }
}

```

```

    }
    for(var i = 0; i < deletetItems.length; i++)
    {
        list.removeChild(deletetItems[i]);           //删除结点
    }
}
/*
 * 在 document 的 click 事件中判断是否单击了 li 中的按钮，并删除 li 元素
 */
document.onclick = function(e)
{
    var evt = arguments[0] || event;
    var elm = evt.target || evt.srcElement;
    if(elm.type == 'button' && elm.className == 'btn')
    {
        var li = elm.parentNode;
        li.parentNode.removeChild(li);
    }
}
</script>
</head>

<body>
    <h1>removeChild demo</h1>
    <ul id="list">
        <li>
            <input type="button" value="delete" class="btn" />
            <input type="checkbox" name="cb" />
            <span>Wildfires May Bring Other Hardships</span>
        </li>
        <li>
            <input type="button" value="delete" class="btn" />
            <input type="checkbox" name="cb" />
            <span>Sentence in Teen Sex Case Ruled Illegal</span>
        </li>
        <li>
            <input type="button" value="delete" class="btn" />
            <input type="checkbox" name="cb" />
            <span>What's the Biggest Shocker of the Week?</span>
        </li>
        <li>
            <input type="button" value="delete" class="btn" />
            <input type="checkbox" name="cb" />
            <span>Spears Custody Case Closed to Media</span>
        </li>
        <li>
            <input type="button" value="delete" class="btn" />
            <input type="checkbox" name="cb" />
            <span>Many States Facing Water Shortages</span>
        </li>
        <li>
            <input type="button" value="delete" class="btn" />

```

```

        <input type="checkbox" name="cb" />
        <span>Student's Death Tied to Staph 'Superbug'</span>
    </li>
    <li>
        <input type="button" value="delete" class="btn" />
        <input type="checkbox" name="cb" />
        <span>Astronauts Prepare to Open Station Room</span>
    </li>
    <li>
        <input type="button" value="delete" class="btn" />
        <input type="checkbox" name="cb" />
        <span>Lawmaker's Remarks Offend the Dutch</span>
    </li>
    <li>
        <input type="button" value="delete" class="btn" />
        <input type="checkbox" name="cb" />
        <span>Rice Looks to History for Peace Effort</span>
    </li>
    <li>
        <input type="button" value="delete" class="btn" />
        <input type="checkbox" name="cb" />
        <span>US to Order Diplomats to Serve in Iraq</span>
    </li>
</ul>
<input type="button" value="delete selected items" onclick="deleteSelectedItems();" />
</body>
</html>

```

程序界面如图 4.24 所示。列表程序在每条信息前面提供了一个复选框，使用户可以一次选择多条记录操作，在每条记录后面和整个列表下面分别提供了 delete 按钮和 delete selected items 按钮，来删除单条记录和多条用户选中的记录。现在来尝试删除第一条记录，直接单击第一条记录后面的 delete 按钮，效果如图 4.25 所示。



图 4.24 演示程序界面



图 4.25 删除第一条记录



记录被成功删除。再来选择最后 3 条记录，分别选中最后 3 条记录前面的复选框，如图 4.26 所示。然后单击 delete selected items 按钮，删除选中的记录，效果如图 4.27 所示。



图 4.26 选中最后 3 条记录

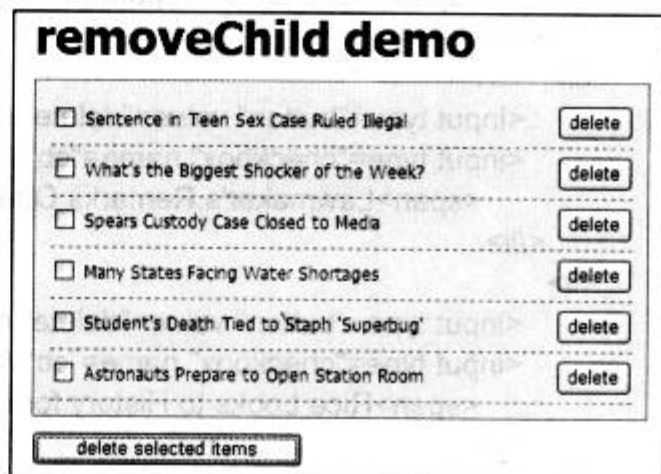


图 4.27 删除选中的记录

程序中使用了 event 对象来对 click 事件的目标进行定位。关于 event 对象的使用，将在本章后面的章节中向读者作详细介绍。

### 4.3.8 读取结点属性

使用 Node 对象的 `getAttribute` 方法可以得到结点的某一属性的值，该方法接受一个属性名作为参数，返回指定属性的值。例如有一 `a` 元素。

```
<a id="link" href="http://www.robchen.cn">Robin Chen</a>
```

则 `document.getElementById("link").getAttribute("href")` 返回 `http://www.robchen.cn`。除此之外，还可以直接使用读取对象属性的方式来读取结点的属性，例如：`document.getElementById("link").href` 同样返回 `http://www.robchen.cn`。

### 4.3.9 添加和修改属性结点

当需要添加或者修改一个属性结点时，可以使用 Node 对象的 `setAttribute` 方法。W3C 并没有为这两部操作提供两部不同的方法，而是将这两项功能集中到了 `setAttribute` 这一个方法上。`setAttribute` 方法的基本语法格式如下所示。

```
node.setAttribute(AttributeName,attValue);
```

其中 `node` 为包含属性结点的元素结点，`attName` 为属性名，`attValue` 为属性值。当 `attName` 为 `node` 已有属性时，则将属性值更新为 `attValue` 的值；如果 `node` 不包含属性名为 `attName` 的属性，则添加一个新的属性，属性名为 `attName`，属性值为 `attValue`。

回顾 4.3.8 小节中的示例。

```
<a id="link"href="http://www.robchen.cn">Robin Chen</a>
```

当希望用一个新窗口打开链接时，可以给 a 元素增加 target 属性。

```
document.getElementById("link").setAttribute("target","_blank");
```

当需要修改链接地址时，例如修改为 http://www.prototypejs.org，可以利用如下代码。

```
document.getElementById("link").setAttribute("href","http://www.prototypejs.org");
```

同读取属性值一样，添加和修改属性也可以使用对象属性的操作方式，例如，上面示例中添加 target 属性和修改 href 属性，可以写成下面的形式。

```
document.getElementById("link").target = "_blank";  
document.getElementById("link").href = "http://www.prototypejs.org";
```

#### 4.3.10 删除属性结点

当需要删除一个结点的某个属性时，可以使用 removeAttribute 方法。removeAttribute 方法接受一个参数，表明了需要删除的属性的名称。其基本语法格式如下所示。

```
node.removeAttribute(attName);
```

例如有一 a 元素。

```
<a id="link"href="http://www.robchen.cn" target="_blank">Robin Chen</a>
```

如果需要删除 target 属性，则可以使用如下代码。

```
document.getElementById("link").removeAttribute("target");
```

### 4.4 控制元素的样式

4.3 节向读者介绍了常见的针对结点的基本操作。DOM 也提供了接口来支持针对元素结点 CSS 样式的操作。通过设置元素的 class 属性和操作元素的 style 属性，可以达到控制元素 CSS 样式的目的。本节将向读者介绍相关的知识。

#### 4.4.1 获取和设置元素的 css 类

通过设置元素的 class 属性，可以为元素指定一个 css 类来设置元素的样式。下面示例中，通过设置链接的 class 属性来给链接添加黄色的背景，代码如下所示。

```
<!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>  
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  
<title>css class demo</title>
```

```

<style type="text/css">
.link {
    background-color:#ff0;
}
input {
    margin-top:10px;
}
</style>
<script type="text/javascript">
function setCssClass()
{
    document.getElementById('link').setAttribute('class','link');
}
</script>
</head>

<body>
    <h1>css class name demo</h1>
    <a id="link" href="http://www.robchen.cn" target="_blank">Robin Chen</a><br />
    <input type="button" value="set css class" onclick="setCssClass();" />
</body>
</html>

```

程序界面如图 4.28 所示。单击 set css class 按钮，css 类.link 的样式应用到了链接上，链接被加上了黄色的背景色，效果如图 4.29 所示。

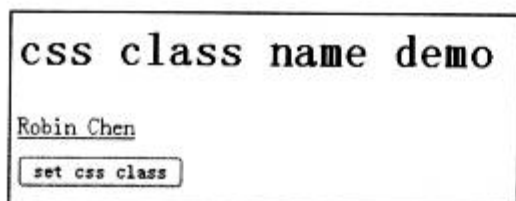


图 4.28 程序界面

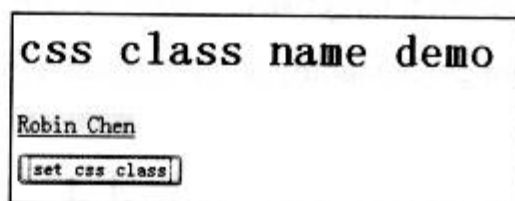


图 4.29 设置链接样式

但上面的示例在 IE 中并不能顺利执行，因为 IE 并不支持通过 DOM 设置结点的 class 属性来改变样式。如果使用操作对象属性的方法，则可以顺利解决这个问题。但是因为 class 是 JavaScript 的保留字，所以不能直接使用 `oNode.class = "..."` 这样的语法，W3C 规定了使用 `className` 来代替。将上例中的 `setCssClass` 函数进行修改，代码如下所示。

```

function setCssClass()
{
    document.getElementById('link').className = 'link';
}

```

则可以成功兼容 Gecko 和 IE 核心的浏览器。

#### 4.4.2 获取和设置元素样式

DOM 为 Node 对象定义了 style 属性，以此作为对 CSS 样式操作的接口。元素的 style 属性是一个

对象，保存了元素的 CSS 样式信息。例如 `node.style.backgroundColor` 保存了背景色的信息，`node.style.color` 保存了文字颜色的信息。

`style` 的对象的属性按照特定的命名规则与 `css` 属性一一对应。命名规则很简单，只需要遵守一些基本的原则。`css` 属性如果由多个单词组成，则省略掉单词之间的横线，将单词连接在一起，从第二个单词开始首字母大写，而如果 `css` 属性只是单个单词，则直接使用。例如 `css` 中的 `background-color` 属性，在 `style` 中为 `backgroundColor`，而 `color` 属性则直接为 `color`。如果 `css` 属性与 JavaScript 的保留字冲突，则在单词前面加上 `style`，单词首字母大写。例如 `css` 的 `float` 属性在 `style` 中为 `styleFloat`。在下面的例程中，通过用户自己指定文字大小来改变页面中文章的字体大小，代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>style demo</title>
<style type="text/css">
body {
    font-size:11px;
    text-align:center;
    font-family:Tahoma;
}
p {
    margin:0 auto;
    width:600px;
    text-indent:2em;
    text-align:left;
    padding-top:10px;
    line-height:18px;
}
div {
    background-color:#eee;
    line-height:18px;
}
a {
    margin:3px 5px;
    color:#000;
}
a:hover {
    color:#f00;
}
</style>
<script type="text/javascript">
function setSize(size)
{
    document.getElementById('content').style.fontSize = size + 'px';
}
</script>
</head>
```



```

<body>
  <h1>网事如风——一把没有扶手的椅子的故事</h1>
  <div>
    <span>选择字体大小:</span>
    <a href="#" onclick="setSize(11);return false;">11px</a>
    <a href="#" onclick="setSize(12);return false;">12px</a>
    <a href="#" onclick="setSize(13);return false;">13px</a>
  </div>
  <p id="content">她从来不上网，只是喜欢坐在我的椅子上笑着看我聊天、收发邮件或在联众上下象棋。虽然她并不懂得象棋。我也曾试图让她自己坐在电脑前，可她说什么也不肯。于是她坐在椅子的左扶手上，右手搭着椅子靠背的那个姿势就成了亘古不变的画面。我每胜一局的时候，习惯于将鼠标一扔喝一喝“over!”，便向后一仰，倒入她柔弱却温暖的臂弯。而她就会“咯咯”地笑着，那满眼的温柔就会像春水一样在我脸上层层地荡漾开来。

  谁曾想到呼啸的风声竟然撕裂了这永恒的画面！那是正起春风的时候，我一如既往地送她到车站，她悄悄指着旁边的一位先生，问我喜不喜欢他那种颜色的领带，我摇着头说不喜欢——刹那间一辆挂着河北牌照的大货车突然冲过隔离带，径直向我们轰然撞来……我宁愿死去也不愿看到那撕心裂肺的一幕！我无法相信眼前的现实，我多么希望这只是一个长夜里未醒的恶梦！我住院的日子里，终日恍恍惚惚以为自己也已随她去了，总觉得生存和死之间也没有太大的距离。这4年来，她竟早已在不知不觉间融入了我的生命！

  回到家里，我坐在电脑前的椅子上，靠着空荡的椅背才相信了这不争的事实。往事像潮水一样汹涌袭来，想起从此以后不会再有人问起“你到底爱我还是爱它”。我一动不动地坐着，麻木地任由眼泪从颧骨一滴滴落在键盘上，视如不见，心若刀割。就这样不知过了多久，我下意识地选了几张她的照片，配上“此情可待”那首我们曾经最喜欢的钢琴曲作了个屏保。然后我用一张床单盖上了我平日一刻也不能释手的爱机，也希望能盖上那段痛彻心扉的回忆。

  今年元宵节晚上，我没敢出去看缤纷绚丽的礼花，我怕看见自己在熙熙攘攘的人群中被焰火映出，长长的落寞的影子！我无聊地用手指敲打着茶几，突然间电视里传出一曲多么熟悉的旋律，那段如泣如诉的钢琴曲竟然使我鬼使神差地坐在尘封了好久的电脑前，握鼠标的手竟然也不再颤抖！

  我又重新回到了网络世界，注册了一个新的自己，废弃了几乎所有信箱，所有的密码用的都是她的名字。我就像一个在虚拟空间中游荡的幽灵，一个跋涉于无边网络的流浪汉，只是有时偶尔到搜狐的一个叫“谁解相思”的聊天室去停留片刻，只因为那里曾是我和她在网络世界中最后一个一起去过的地方。我并不迷信，但仍无数次地幻想美丽善良的她会从冥冥中发一封 E-mail 给我，而我唯一留下的那个只有她知道信箱，也会为了她保留到我生命的最后一刻。

  转眼春风又起，椅子也还是那把椅子，只是我锯掉了它的扶手。在我向后仰的时候，也开始习惯于把两只手交叉起双叠在脑后，回想着那“咯咯”的笑声和那双荡漾着温柔的黑眼睛……</p>
</body>
</html>

```

程序界面如图 4.30 所示。

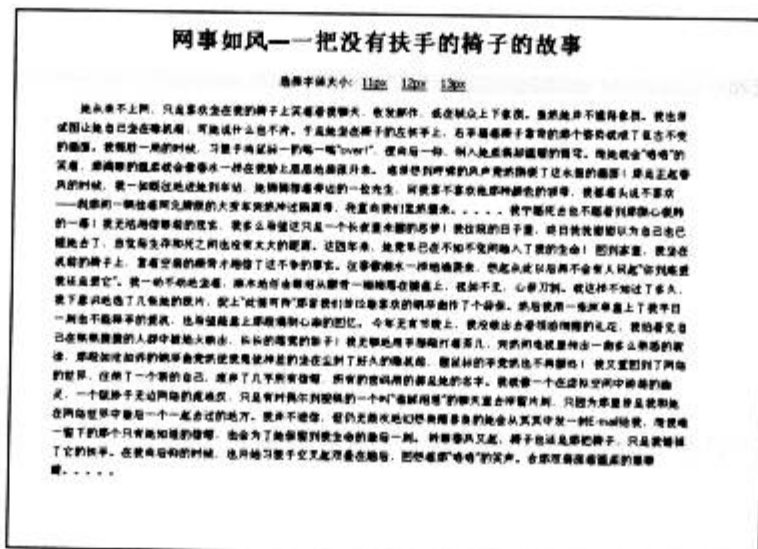


图 4.30 style 演示程序界面



在文章标题的下面提供了 3 种字体大小供用户选择, 即 11px、12px 和 13px。程序通过给 a 元素添加 onclick 来指定 click 事件的处理程序为 setSize 函数, 并在处理程序中设置 p 元素的字体大小样式。因为 a 是链接标签, 系统赋予的默认行为会发生页面跳转, 所以在 onclick 属性值的后面加上 return false 来阻止系统默认行为。文章默认字体大小是 11px, 当单击 12px 和 13px 的链接按钮时, 效果如图 4.31 和图 4.32 所示。

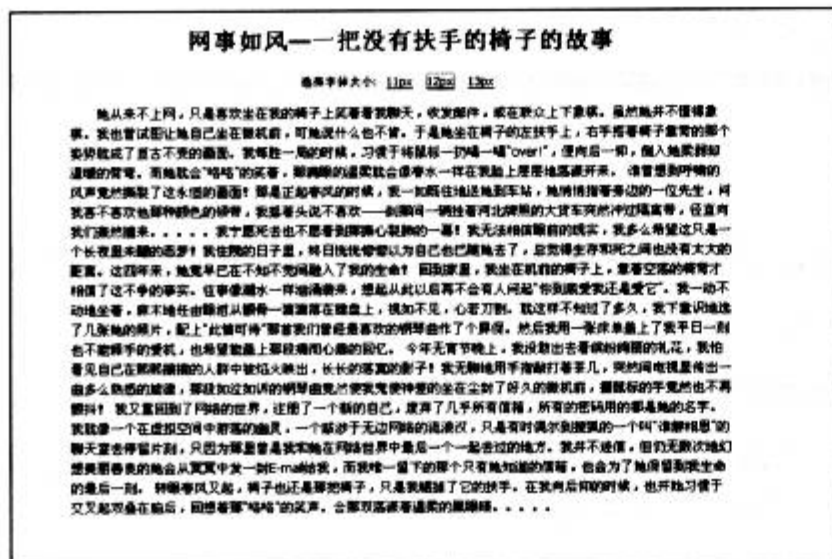


图 4.31 选择 12px

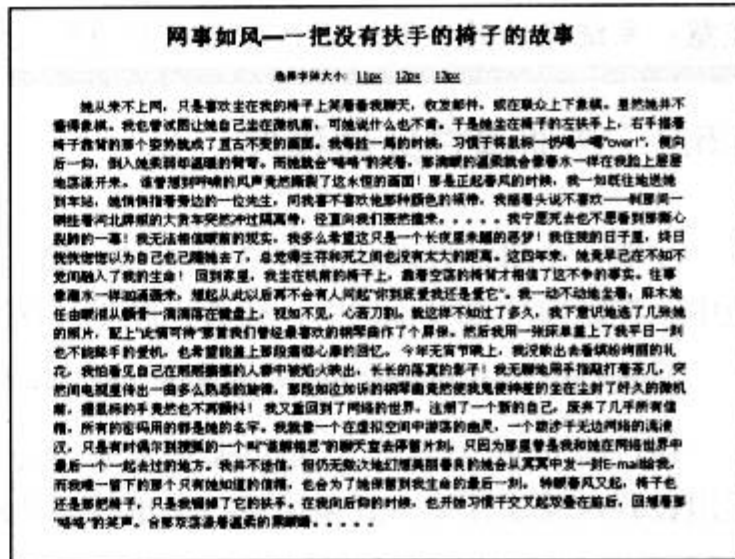


图 4.32 选择 13px

## 4.5 事件处理

事件处理是 DOM 中最重要的组成部分。事件驱动是现代面向对象编程的基本方法, 完善的事件机制使 JavaScript 程序可以根据特定的事件来触发不同的执行方法, 使得程序可以更具有交互性和智能化。在本节之前的许多示例中, 读者已经见过事件的使用。本节将向读者详细介绍 DOM 的事件模型。

### 4.5.1 事件模型和传播机制

目前主流浏览器所使用的事件模型主要为标准事件模型和 IE 事件模型。IE 事件模型由 IE 浏览器使用, 而标准事件模型由 W3C 制订, 由 Netscape 等浏览器实现。

两种事件模型的差异首先体现在事件对象上。在 IE 事件模型中, window 对象提供了一个 event 属性来保存当前事件对象, 所以可以在程序中像使用全局变量一样来使用事件对象。而在标准事件模型中, 事件对象是在事件被触发时生成, 然后作为参数传递给事件处理程序, 所以在标准事件模型中事件是局部的。关于事件对象将在 4.5.4 小节中向读者作详细介绍。

事件模型的差异还体现在事件的传播机制上。在标准事件模型中, 事件的传播分为三个阶段。第一个阶段是捕捉 (capturing) 阶段。在这个阶段, 事件从 document 对象沿着文档树向下往目标结点传播, 如果在传播过程中的任一结点注册了处理该事件的事件处理程序, 则程序会被执行。即在这个阶段目标结点的任何祖先结点都有机会来处理事件。第二个阶段发生在目标结点自身, 注册在目标结点上的合适的事件处理程序将被执行。第三个阶段是起泡阶段。在这个阶段中, 事件将会沿着文档树从

目标结点回传给 document 对象。所有的事件都受捕捉机制的支配，但并非所有的事件都起泡。一般而言，按照事件的类型，可以将事件分为两类：输入事件和高级语义事件。输入事件指的是用户操作产生的事件，例如移动鼠标、单击按键等。高级语义事件指的是由系统内部触发的事件，例如窗体加载完成的事件、图片加载完成的事件、表单提交的事件等。所有高级语义事件都不会起泡。在 IE 事件模型中，只支持起泡形式的事件传播，而不支持捕捉形式的事件传播。

注意：系统默认的事件处理程序都在起泡阶段被处理，例如的 click 事件等。

## 4.5.2 注册事件处理程序

注册一个事件处理程序有 3 种方法。第一种方法是作为结点的属性直接将事件处理程序写在属性值中。在之前的很多示例中，读者已经见过这种方法。

```
<input id="btnSetSize" type="button" onclick="setSize(11);" />
```

通过给 input 结点增加 onclick 属性来注册一个 click 事件的处理程序。另外一种与之类似的方法是采用设置对象属性的方式来注册一个事件处理程序。例如上面的例子可以改为下面的形式。

```
<input id="btnSetSize" type="button"/>
<script type="text/javascript">
document.getElementById('btnSetSize').onclick = function()
{
    setSize(11);
}
</script>
```

通过给结点对象的 onclick 属性赋一个函数值，来达到给结点的 click 事件注册一个事件处理程序的目的。最后一种方法，也是推荐使用的方法，就是采用标准事件模型规定的方法来注册事件处理程序。使用关键字 addEventListener 来注册一个事件处理程序，其基本语法如下所示。

```
obj.addEventListener(eventName,eventHandler,useCapture);
```

其中 obj 为目标结点对象。eventName 是一个表达事件名称的字符串，例如"click"、"dblclick"、"mouseover"、"keydown"等。eventHandler 是事件处理函数，在指定的事件发生时调用该函数，函数被调用时将会被传入一个事件对象作为参数。useCapture 是一个布尔值，如果为 true，则指定的事件处理程序将在事件传播的捕捉阶段用于捕捉事件；如果为 false，则当事件直接发生在对象上，或者发生在对象的子孙结点并起泡到对象上时，处理函数被调用，默认为 false。

使用 addEventListener 注册事件处理程序，可以给目标的同一事件注册多个事件处理程序。这样带来的好处就是使得程序可以模块化开发，而不用担心彼此的冲突。例如，在某个模块中需要使用按钮的 click 事件来提交表单，而在另外一个模块中需要使用按钮的 click 事件来显示状态信息，这时就可以直接使用 addEventListener 为按钮的 click 事件注册两个不同的处理函数，来达到预期的要求。下面的示例向读者演示了这一特性。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>addEventListener demo</title>
<style type="text/css">
#demo {
    width:300px;
    height:300px;
    border:1px solid black;
    margin:30px;
}
</style>
<script type="text/javascript">
function init()                                //注册 div 的 mouseover 和 mouseout 事件处理程序
{
    var div = document.getElementById('demo');
    div.addEventListener('mouseover',setBackgroundColor,false);
    div.addEventListener('mouseover',setBorderStyle,false);
    div.addEventListener('mouseout',removeBackgroundColor,false);
    div.addEventListener('mouseout',removeBorderStyle,false);
}
function setBackgroundColor()                  //设置 div 的背景色为红色
{
    document.getElementById('demo').style.backgroundColor = '#f00';
}
function removeBackgroundColor()               //设置 div 的背景色为白色
{
    document.getElementById('demo').style.backgroundColor = '#fff';
}
function setBorderStyle()                      //设置 div 的边框粗细为 10 像素
{
    document.getElementById('demo').style.border = '10px solid black';
}
function removeBorderStyle()                   //设置 div 的边框粗细为 1 像素
{
    document.getElementById('demo').style.border = '1px solid black';
}
window.addEventListener('load',init,false);    //在 window 的 load 事件被触发时执行 init 函数
</script>
</head>

<body>
<h1>addEventListener demo</h1>
<div id="demo"></div>
</body>
</html>

```

在上面示例的 init 函数中，为 div 的 mouseover 事件和 mouseout 事件分别注册了两个处理函数。然后通过将 init 函数注册给 window 的 load 事件，来使 init 函数在窗体加载完成时被调用。使用实现了



标准事件模型的浏览器，例如 Firefox 打开示例程序，界面如图 4.33 所示。

将鼠标移到 div 上，以此来触发 div 的 mouseover 事件。这时注册给 mouseover 事件的 setBackground-Color 函数和 setBorderStyle 函数都被调用，div 的边框粗细和背景色被改变的效果如图 4.34 所示。

再将鼠标移开，则 div 的 mouseout 事件被触发。这时注册给 mouseout 事件的 removeBackgroundColor 函数和 removeBorderStyle 函数被调用，div 的边框粗细和背景色恢复原状，如图 4.35 所示。

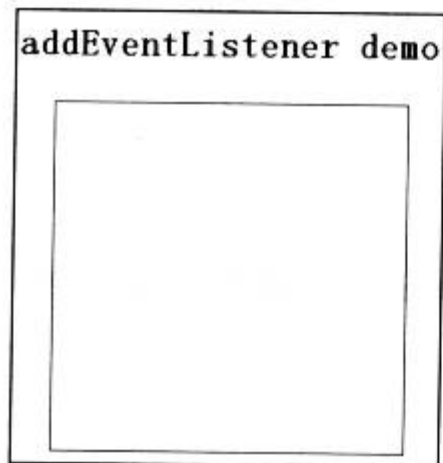


图 4.33 addEventListener 演示程序界面

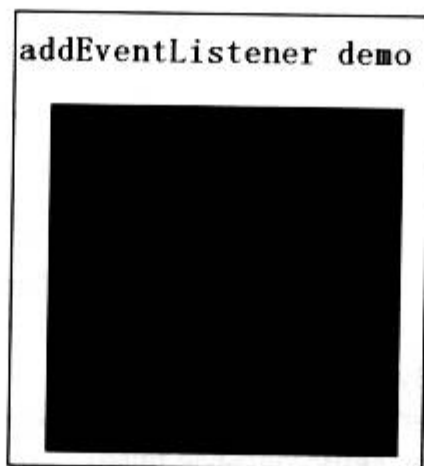


图 4.34 鼠标划过效果

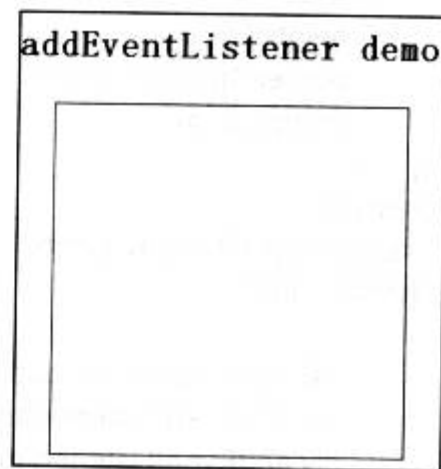


图 4.35 鼠标移开效果

在 IE 5.0 及以后的版本中，微软在其 IE 事件模型中加入了 attachEvent 来对应实现 addEventListener 的功能。其基本语法如下所示。

```
obj.attachEvent('on' + eventName,eventHandler);
```

其中 obj 为目标结点，eventName 为事件名，eventHandler 为处理函数。与 addEventListener 不同的是，attachEvent 的第一个参数，是在事件名前面加上 on，例如 mouseover 事件，需要写为 onmouseover，click 事件需要写为 onclick。而又由于 IE 事件模型并不支持捕捉形式的事件传播，所以没有 useCapture 参数。在 IE 事件模型下上面的示例需要进行一些修改，代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>attachEvent demo</title>
<style type="text/css">
#demo {
width:300px;
height:300px;
border:1px solid black;
margin:30px;
}
</style>
<script type="text/javascript">
function init()
{
var div = document.getElementById('demo');
div.attachEvent('onmouseover',setBackgroundColor);
div.attachEvent('onmouseover',setBorderStyle);
```

//注册 div 的 mouseover 和 mouseout 事件处理程序

```

        div.attachEvent('onmouseout',removeBackgroundColor);
        div.attachEvent('onmouseout',removeBorderStyle);
    }
    function setBackgroundColor()                //设置 div 的背景色为红色
    {
        document.getElementById('demo').style.backgroundColor = '#f00';
    }
    function removeBackgroundColor()            //设置 div 的背景色为白色
    {
        document.getElementById('demo').style.backgroundColor = '#fff';
    }
    function setBorderStyle()                   //设置 div 的边框粗细为 10 像素
    {
        document.getElementById('demo').style.border = '10px solid black';
    }
    function removeBorderStyle()                //设置 div 的边框粗细为 1 像素
    {
        document.getElementById('demo').style.border = '1px solid black';
    }
    window.attachEvent('onload',init);          //在 window 的 load 事件被触发时执行 init 函数
</script>
</head>

<body>
<h1>attachEvent demo</h1>
<div id="demo"></div>
</body>
</html>

```

鼠标滑入和滑出 div 的效果与之前的示例完全一致。

因为在实际项目开发中经常需要注册事件处理程序，所以一般的做法是对 `addEventListener` 和 `attachEvent` 进行封装，以此来解决浏览器的兼容性问题，减少程序开发的代码量，加快开发速度。下面是一个经过简单封装的事件注册程序。

```

function addEvent(obj,name,handler,useCapture)
{
    if(window.event)
    {
        obj.attachEvent('on' + name,handler);
    }
    else
    {
        obj.addEventListener(name,handler,useCapture);
    }
}

```

在上面的函数中，通过判断 `window` 是否含有 `event` 属性，来判断浏览器使用的哪种事件模型。如果是 IE 事件模型，则使用 `attachEvent` 来注册事件处理程序，这时可以省略 `useCapture` 参数。如果是标准时间模型，则使用 `addEventListener` 来注册事件处理程序。



### 4.5.3 注销事件处理程序

当不再需要一个事件处理程序时，可以将其注销。使用结点属性或者对象属性注册的事件处理程序，可以通过将对象的相应属性设置为 null 来注销该事件处理程序。使用 `attachEvent` 或 `addEventListener` 注册的事件处理程序，可以使用对应的 `detachEvent` 或 `removeEventListener` 来注销。下面的示例演示了如何注销一个事件处理程序。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>remove event handler demo</title>
<script type="text/javascript">
window.onload = function()
{
    document.getElementById('btnB').onclick = clickEventHandler;
    addEvent(document.getElementById('btnC'),'click',clickEventHandler,false);
}
function removeClickEvent()                //注销事件处理函数
{
    document.getElementById('btnA').onclick = null;
    document.getElementById('btnB').onclick = null;
    removeEvent(document.getElementById('btnC'),'click',clickEventHandler,false);
    document.getElementById('msg').innerHTML = 'The test buttons' click event handler have been
removed.';
}
function clickEventHandler()                //click 事件处理函数
{
    alert('test');
}
function addEvent(obj,name,handler,useCapture)    //封装好的注册事件函数
{
    if(window.event)
    {
        obj.attachEvent('on' + name,handler);
    }
    else
    {
        obj.addEventListener(name,handler,useCapture);
    }
}
function removeEvent(obj,name,handler,useCapture)    //封装好的注销事件函数
{
    if(window.event)
    {
        obj.detachEvent('on' + name,handler);
    }
}
```

```

    }
    else
    {
        obj.removeEventListener(name,handler,useCapture);
    }
}
</script>
</head>

<body>
<h1>remove event handler demo</h1>
<div id="msg"></div>
<input type="button" id="btnA" onclick="clickEventHandler();" value="test button A" />
<input type="button" id="btnB" value="test button B" />
<input type="button" id="btnC" value="test button C" />
<br />
<input type="button" value="remove click event" onclick="removeClickEvent()" />
</body>
</html>

```

程序界面如图 4.36 所示。

在程序中，通过添加 test button A 按钮的 onclick 结点属性，给其 click 事件注册了处理函数 clickEventHandler；通过在 window 的 load 事件中给 test button B 按钮对象添加 click 属性，给其 click 事件注册了处理函数 clickEventHandler；通过 addEvent 函数给 test button C 按钮的 click 事件注册了处理函数 clickEventHandler。单击按钮 test button A、test button B 或者 test button C，都会调用 clickEventHandler 函数，弹出对话框，效果如图 4.37 所示。



图 4.36 程序界面

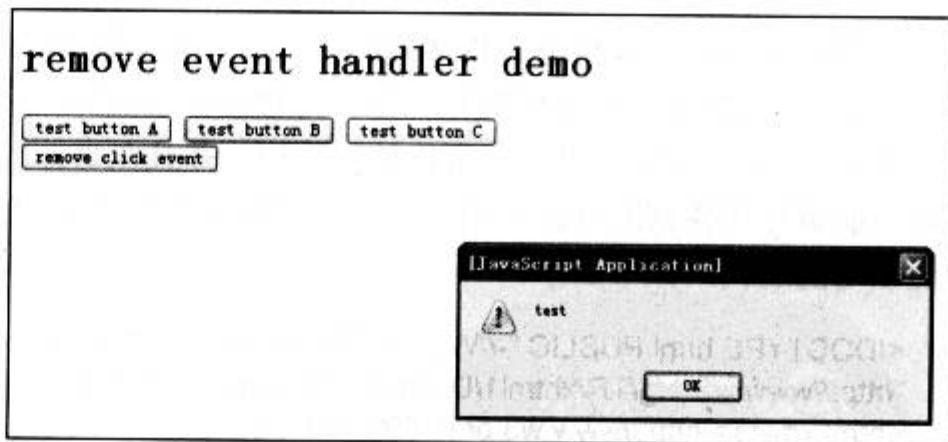


图 4.37 单击测试按钮

单击 remove click event 按钮，触发其 click 事件处理函数 removeClickEvent。在这个函数中，将 test button A 按钮和 test button B 按钮的 click 属性都设置为 null，再通过封装好的注销事件函数 removeEvent，注销了 test button C 的 click 事件处理程序。然后函数在页面上输出一条提示信息，表示测试按钮的事件处理程序都已经被注销。这时再单击 3 个测试按钮，不会有任何反应，界面如图 4.38 所示。

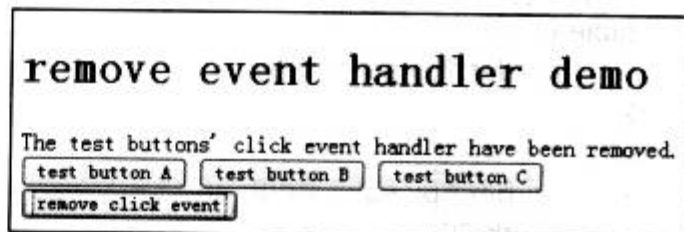


图 4.38 注销事件处理程序

#### 4.5.4 事件对象

在前面介绍事件模型时，已经向读者简单介绍过事件对象的作用，就是在事件发生时生成事件对象，保存到 window 对象的 event 属性中（IE 事件模型），或者当作第一个参数传递给事件处理程序（标准事件模型）。事件对象保存了事件的相关信息，例如事件的类型、发生事件的目标元素等。表 4-2 介绍了最常用的事件对象的属性和方法，更完整的说明参阅本书附录 B 中的 DOM 速查手册。

表 4-2 事件对象常用属性和方法

属性/方法名（标准事件模型）	属性/方法名（IE 事件模型）	说 明
target	srcElement	发生事件的目标元素
type	type	发生事件的类型，如click、mouseover等
keyCode	keyCode	声明了keydown、keyup事件的按键代码
clientX、clientY	clientX、clientY	事件发生时鼠标的位置相对于窗口左上角的坐标
preventDefault()	returnValue	当将IE事件模型中事件对象的returnValue设置为false时，或者调用标准事件模型中事件对象的preventDefault方法，可以阻止浏览器执行与事件相关的默认动作
stopPropagation()	cancelBubble	当将IE事件模型中事件对象的cancelBubble设置为true，或者调用标准事件模型中事件对象的stopPropagation方法时，将阻止事件继续起泡传播
altKey、ctrlKey、shiftKey	altKey、ctrlKey、shiftKey	这3个布尔属性分别标识了在鼠标事件发生时，是否按住了Alt键、Ctrl键或Shift键

灵活运用事件对象和事件传播机制，可以大量节省 JavaScript 代码、增强 JavaScript 解决问题的能力。例如，当需要为一个表格的每一个单元格都注册 click 事件处理程序时，不需要一个一个地去注册，而只需要给 table 对象注册一个事件处理程序，并且在事件的传播阶段进行处理。通过处理事件对象来获取当前事件传播到的目标元素，进一步判断是否为需要的单元格，从而决定是否执行事件处理程序。实现该功能的代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>event object demo</title>
<style type="text/css">
table {
border:1px solid #999;
}
td {
border:1px solid #999;
width:30px;
height:30px;
background-color:#eee;
```

```

font-size:12px;
line-height:30px;
text-align:center;
cursor:pointer;
}
</style>
<script type="text/javascript">
function addEvent(obj,name,handler,useCapture)           //封装好的注册事件函数
{
    if(window.event)
    {
        obj.attachEvent('on' + name,handler);
    }
    else
    {
        obj.addEventListener(name,handler,useCapture);
    }
}
function mouseoverEventHandler(e)                         //鼠标滑过事件处理函数
{
    var evt = e || event;                                //如果 e 存在，则返回 e，否则返回 event 对象
    /* 如果 evt 对象存在 target 属性，则返回该属性值，否则返回 srcElement 属性值 */
    var elm = evt.target || evt.srcElement;
    if(elm.tagName.toLowerCase() == 'td')                 //判断当前事件目标元素是否为 td 元素
    {
        elm.style.border = '1px solid #f00';             //如果是 td 元素，则改变 td 的边框颜色
    }
}
function mouseoutEventHandler(e)                         //鼠标滑出事件处理程序
{
    var evt = e || event;
    var elm = evt.target || evt.srcElement;
    if(elm.tagName.toLowerCase() == 'td')
    {
        elm.style.border = '1px solid #999';             //如果是 td 元素，则还原 td 的边框颜色
    }
}
function init()                                           //初始化程序
{
    var tb = document.getElementById('tb');
    addEvent(tb,'mouseover',mouseoverEventHandler,false);
    addEvent(tb,'mouseout',mouseoutEventHandler,false);
}
window.onload = init;                                     //在窗体加载完成时初始化程序
</script>
</head>

<body>
    <table border="0" cellspacing="2" cellpadding="0" id="tb">
        <tr>

```

```

        <td>1</td>
        <td>2</td>
        <td>3</td>
        <td>4</td>
        <td>5</td>
    </tr>
    <tr>
        <td>6</td>
        <td>7</td>
        <td>8</td>
        <td>9</td>
        <td>10</td>
    </tr>
    <tr>
        <td>11</td>
        <td>12</td>
        <td>13</td>
        <td>14</td>
        <td>15</td>
    </tr>
    <tr>
        <td>16</td>
        <td>17</td>
        <td>18</td>
        <td>19</td>
        <td>20</td>
    </tr>
    <tr>
        <td>21</td>
        <td>22</td>
        <td>23</td>
        <td>24</td>
        <td>25</td>
    </tr>
</table>
</body>
</html>

```

程序界面如图 4.39 所示。

当鼠标滑过单元格时，单元格边框变成红色，滑开后，回复原状，如图 4.40 和图 4.41 所示。



1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

图 4.39 程序界面



1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

图 4.40 程序效果演示（一）



1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

图 4.41 程序效果演示（二）



读者可能会对程序中这样的语法感到不解。

```
var evt = e || event;
var elm = evt.target || evt.srcElement;
```

这其实是对事件模型差异的一个兼容性处理。由于在标准事件模型中，事件对象是作为参数传递给事件处理函数的，所以在变量 `e` 中保存的就是传递进来的事件对象。而在 IE 事件模型中，事件对象是保存在 `window` 的 `event` 属性中，这时变量 `e` 的值为 `undefined`。

```
var evt = e || event;
```

实际上是利用了逻辑或运算符返回第一个为真（或者能够转化为真值）的表达式的值这一特性，来取得 `e` 或者 `event` 中有意义的一个值。同样，为了解决标准事件模型和 IE 事件模型对事件目标元素的描述差异，使用

```
var elm = evt.target || evt.srcElement;
```

来统一获取目标元素。

### 4.5.5 常用事件

在学习了事件处理函数的注册和注销，以及事件对象的使用后，读者一定很想知道有哪些事件可以使用。表 4-3 列出了常用的事件及其支持的 HTML 元素。

表 4-3 常用事件

事件类型	支持的元素	触发时间
change	<input>、<select>、<textarea>	改变<select>元素的选中项或其他表单元素的值，在其失去焦点时触发
click	绝大部分元素	鼠标键按下并释放时触发
dblclick	绝大部分元素	双击鼠标并释放时触发
focus	<a>、<area>、<button>、<input>、<label>、<select>、<textarea>	元素获得焦点时触发
blur	<a>、<area>、<button>、<input>、<label>、<select>、<textarea>	元素失去焦点时触发
load	<body>、<frameset>、<iframe>、<img>、<object>	文档或者元素装载完成时触发
abort	<img>、<object>	元素取消装载时触发
unload	<body>、<frameset>	卸载文档或者框架集时触发
keyup	<body>、<input>、<textarea>	按下键盘键并释放时触发
keydown	<body>、<input>、<textarea>	按下键盘键时触发
keypress	<body>、<input>、<textarea>	释放键盘时触发
mousedown	绝大部分元素	按下鼠标键时触发
mousemove	绝大部分元素	鼠标移动时触发
mouseover	绝大部分元素	鼠标移动到元素上时触发
mouseout	绝大部分元素	鼠标离开元素时触发

续表

事件类型	支持的元素	触发时间
mouseup	绝大部分元素	释放鼠标时触发。在click事件之前
submit	<form>	提交表单时触发，在表单真正被提交之前
reset	<form>	表单被重置时触发，在表单真正被重置之前
resize	<body>、<frameset>、<iframe>	改变窗口或者框架集大小时触发
scroll	<body>	滚动条位置发生变化时触发
select	<input>、<textarea>	选中文本时触发

## 4.6 应用实例

本节综合本章中向读者介绍的有关 DOM 的知识，来编写两个常见的 JavaScript 应用实例。一个是悬浮的广告，另外一个可是可拖动的层。建议读者自己将代码输入到编辑器中，并保存为 HTML 页面，然后到浏览器中查看效果，以加深记忆和理解。对于示例中出现的一些本章中未说明的内容，可以查阅本书附录 B 中的 DOM 速查手册来了解其用法。

### 4.6.1 悬浮的广告

读者朋友们在很多网站上都可以看见这样的广告：广告的图片或文字始终停留在页面的某个位置，无论如何拖动滚动条，广告都会跟随屏幕的滚动而滚动。下面是一个实现该效果的实例，读者可以将代码保存成页面，并用浏览器访问来查看其效果。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>float ads demo</title>
<style type="text/css">
<!--
*{
    margin:0;
    padding:0;
}
body{
    font-family:Tahoma;
    height:1000px;
    background-color:#eee;
}
#floatAds {
    position:absolute;
    width:200px;
    height:50px;
```

```

z-index:1;
left: 11px;
top: 100px;
background-color:#000099;
font-size:20px;
text-align:center;
line-height:50px;
color:#fff;
}
-->
</style>
<script type="text/javascript">
function addEvent(obj,name,handler,useCapture)           //封装好的注册事件函数
{
    if(window.event)
    {
        obj.attachEvent('on' + name,handler);
    }
    else
    {
        obj.addEventListener(name,handler,useCapture);
    }
}
var interval = null;           //定时器
var timeout = null;           //延迟器
var targetPosition;           //浮动目标位置
/*
 * 初始化程序,注册 scollEventHandler 处理函数到 window 对象的 scroll 事件
 */
function init()
{
    addEvent(window,'scroll',scollEventHandler,false);
}
/*
 * scroll 事件处理函数,当 scroll 事件被触发时, 设置一个延迟器, 在 50 毫秒后执行 startFloat 函数。如果延迟器
  已经被设置, 则取消之前设置的延迟器, 然后重新设置。
 */
function scollEventHandler()
{
    if(timeout)
    {
        clearTimeout(timeout);
    }
    setTimeout(startFloat,50);
}
/*
 * 开始调整浮动位置。先计算出广告层需要到达的位置, 然后设置一个定时器定时执行 doFloat 函数。如果定时
  器已经被设置, 则取消之前设置的定时器, 然后重新设置。
 */
function startFloat()

```

```

{
    var ads = document.getElementById('floatAds');
    targetPosition = (document.body.scrollTop || document.documentElement.scrollTop) + 100;
    if(parseInt(ads.style.top) != targetPosition)
    {
        if(interval != null)
        {
            clearInterval(interval);
        }
        interval = setInterval(doFloat,1);
    }
}
/*
 * 通过改变广告层 css 样式的 top 属性，来调整广告层的位置，每次改变 1 像素。当广告层到达目标位置后，取消定时器。
 */
function doFloat()
{
    var ads = document.getElementById('floatAds');
    var currentPosition = ads.offsetTop; //读取广告层的当前位置
    if(currentPosition < targetPosition) //如果当前位置小于目标位置，则将广告层 css 样式的
top 属性增加 1 像素
    {
        ads.style.top = currentPosition + 1 + 'px';
    }
    else if(currentPosition > targetPosition) //如果当前位置大于目标位置，则将广告层 css 样式的
top 属性增加 1 像素
    {
        ads.style.top = currentPosition - 1 + 'px';
    }
    else
    {
        clearInterval(interval); //如果当前位置等于目标位置，则取消定时器
        interval = null;
    }
}
window.onload = init; //在窗口加载完成时初始化程序
</script>
</head>

<body>
<div id="floatAds">Advertisement</div>
</body>
</html>

```

#### 4.6.2 可拖动的层

用过 Google 个性化首页或者 QQ 空间装扮功能的读者，一定会对其能够自由拖动各个内容层的操作印象深刻，下面的实例就是简单实现层的拖动这一功能。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>drag div demo</title>
<style type="text/css">
<!--
#dragDiv {
    position: absolute;
    width: 200px;
    height: 115px;
    z-index: 1;
    left: 40px;
    top: 51px;
    background-color: #3300CC;
    cursor: pointer;
}
-->
</style>
<script type="text/javascript">
function addEvent(obj,name,handler,useCapture)           //封装好的注册事件函数
{
    if(window.event)
    {
        obj.attachEvent('on' + name,handler);
    }
    else
    {
        obj.addEventListener(name,handler,useCapture);
    }
}
var draggable = false;                                   //当前是否是拖动的开关
var mousePosition = [];                                  //鼠标位置缓存
/*
 * mousedown 事件处理函数。当 mousedown 事件触发时，将 draggable 开关设置为 true，并且读取鼠标位置存入 mousePosition 数组。
 */
function mousedownEventHandler(evt)
{
    draggable = true;
    var evt = evt || event;
    var mouseX = (document.body.scrollLeft || document.documentElement.scrollLeft) + evt.clientX;
    var mouseY = (document.body.scrollTop || document.documentElement.scrollTop) + evt.clientY;
    mousePosition = [mouseX,mouseY];
}
/*
 * mousemove 事件处理函数。当 mousemove 事件触发时，读取鼠标当前位置和鼠标位置缓存比较，然后将差值增加到 div 的坐标中，达到移动 div 的效果。
 */

```



```

function mousemoveEventHandler(evt)
{
    if(!draggable)                                //如果 draggable 为 false, 则不执行拖动
    {
        return;
    }
    var evt = evt || event;
    var mouseX = (document.body.scrollLeft || document.documentElement.scrollLeft) + evt.clientX;
    var mouseY = (document.body.scrollTop || document.documentElement.scrollTop) + evt.clientY;
    div.style.left = div.offsetLeft + mouseX - mousePosition[0] + 'px';
    div.style.top = div.offsetTop + mouseY - mousePosition[1] + 'px';
    mousePosition = [mouseX,mouseY];              //每一次拖动完成后, 更新鼠标位置缓存
}
/*
 * mouseup 事件处理函数。当 mouseup 事件触发时, 将 draggable 开关设置为 false, 停止拖动。
 */
function mouseupEventHandler()
{
    draggable = false;
}
function init()                                    //初始化程序
{
    div = document.getElementById('dragDiv');
    addEvent(div,'mousedown',mousedownEventHandler,false);
    addEvent(document,'mousemove',mousemoveEventHandler,false);
    addEvent(div,'mouseup',mouseupEventHandler,false);
}
window.onload = init;                             //在窗体加载完成时初始化程序
</script>
</head>

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

```

## 4.7 小 结

本章向读者介绍了 DOM（文档对象模型）的相关知识。首先介绍了 DOM 的层次结构，是一个以 document 对象为根结点的树形结构，DOM 结点的类型和组成，以及结点之间的关系。然后向读者介绍了引用结点的各种方法，有直接通过 id、name 或标签名引用的方法，也有通过结点之间的关系引用的间接引用方法。接着向读者介绍了针对结点的各种操作，包括创建结点、添加结点、删除结点、替换结点对属性的操作。之后向读者介绍了如何通过 DOM 定义的接口来控制结点的样式，包括控制结点的 class 属性和操作 style 对象。然后向读者介绍了 DOM 的事件模型，其中包括标准事件模型和 IE 事件模型，讲解了如何注册和注销事件处理程序，分析了事件对象并罗列了常用的事件。最后，向读者展示了两个常见的应用实例：浮动广告和拖动层。

# 第 5 章

## 开发 Ajax 应用程序需要使用的工具

- » 开发工具: Aptana
- » Firefox
- » HTTP 调试工具: Fiddler
- » 小结

在前面几章中，向读者介绍了开发 Ajax 应用程序的语言：JavaScript 的基本知识和常用对象，然后向读者介绍了操作 HTML 文档的接口 DOM（文档对象模型）。读者已经明白如何编写一个操作 HTML 文档的 JavaScript 程序。工欲善其事，必先利其器。选择好的开发工具会使实际编码的工作变得更加简单和高效。本章向读者推荐一些开发 Ajax 应用程序可以使用的工具，例如 Aptana Web IDE、firebug 等。

## 5.1 开发工具: Aptana

一直以来, 由于缺少专业的 JavaScript 开发工具, 使得大多数开发者只能使用简单的文本编辑器, 例如 NotePad, 或者稍微支持 JavaScript 语法提示的开发工具, 例如 Dreamweaver、Visual Studio 等, 进行 JavaScript 和 Ajax 的开发, 使得 JavaScript 开发效率和调试成为了开发者的难点问题。直到 Aptana 的“横空出世”。Aptana 是一款使用 Java 开发、基于 JVM 的专业 JavaScript 开发工具。

Aptana 提供了强大的代码提示、浏览器兼容性提示、错误提示、代码自动完成等功能, 并且提供了基于 Firefox 的调试功能, 同时支持各大主流的 Ajax 开发框架, 包括 prototype、JQuery、YUI-ext 等。Aptana 提供了开发者良好的编码体验, 能够支持开发者进行快速高效的 Ajax 应用程序开发。下面让读者来认识这款优秀的开发工具: Aptana。

### 5.1.1 Aptana 的下载和安装

Aptana 是采用 Java 开发的、免费的、开源的、跨平台的 WEB IDE。因为采用 Java 开发, 所以需要有 JVM 的支持。Aptana 的官方网站是 <http://www.aptna.com>。官方网站上的下载频道提供了 Aptana 的下载, 其地址为 <http://www.aptna.com/download/>。打开下载页面, 如图 5.1 所示。

网站上针对不同操作系统提供 win 版本、Mac 版本和 Linux 版本的 Aptana 完整版的下载, 同时还提供了一个作为 Eclipse 插件的版本下载。这里以 win 版本为例, 单击下载链接跳转到页面底部, 如图 5.2 所示。

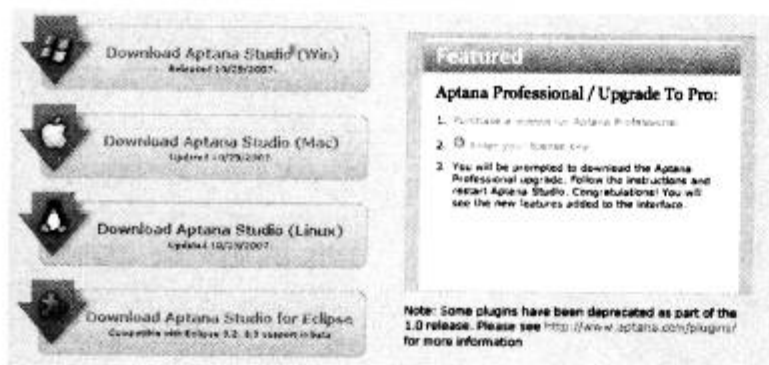


图 5.1 下载 Aptana (一)

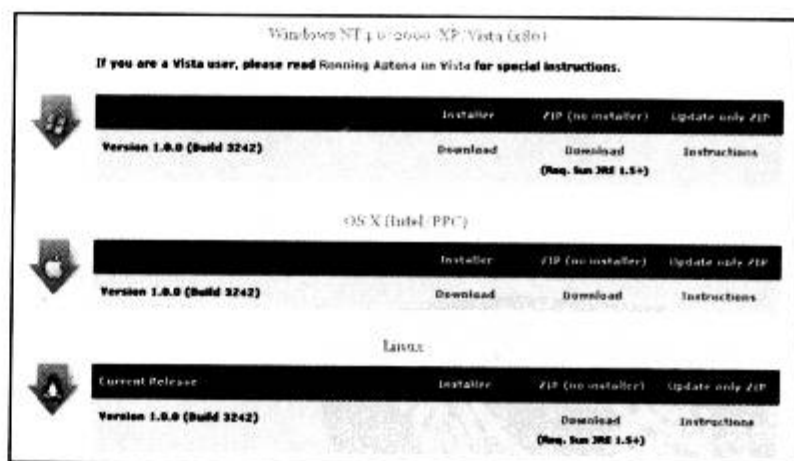


图 5.2 下载 Aptana (二)

单击 Windows NT4.0/2000/XP/Vista(x86)下面的 Download 链接即可开始下载。下载完成后, 运行所下载的 Aptana 安装文件开始安装。安装过程与一般的应用软件安装方式相同, 这里不再赘述。

### 5.1.2 Aptana 的界面介绍

Aptana 安装完成后, 会在桌面和“开始→程序”中生成快捷方式。双击快捷方式启动 Aptana, 如



图 5.3 所示。

第一次启动时, Aptana 会提示用户选择工作区, 即保存项目的路径, 如图 5.4 所示。



图 5.3 启动 Aptana

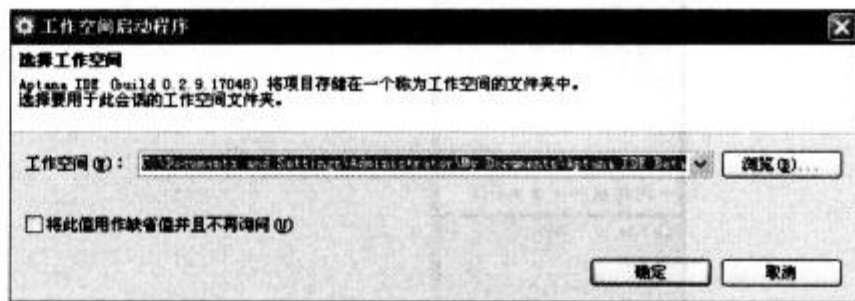


图 5.4 设置工作区

一般设置为 Apache 或者 IIS 的虚拟目录。如果希望以后启动 Aptana 都使用这个工作区, 则可以将图中的“将此值用作默认值并且不再询问”复选框选中, 设置选择的工作区为默认值即可。单击“确定”按钮, 进入 Aptana 工作界面, 如图 5.5 所示。

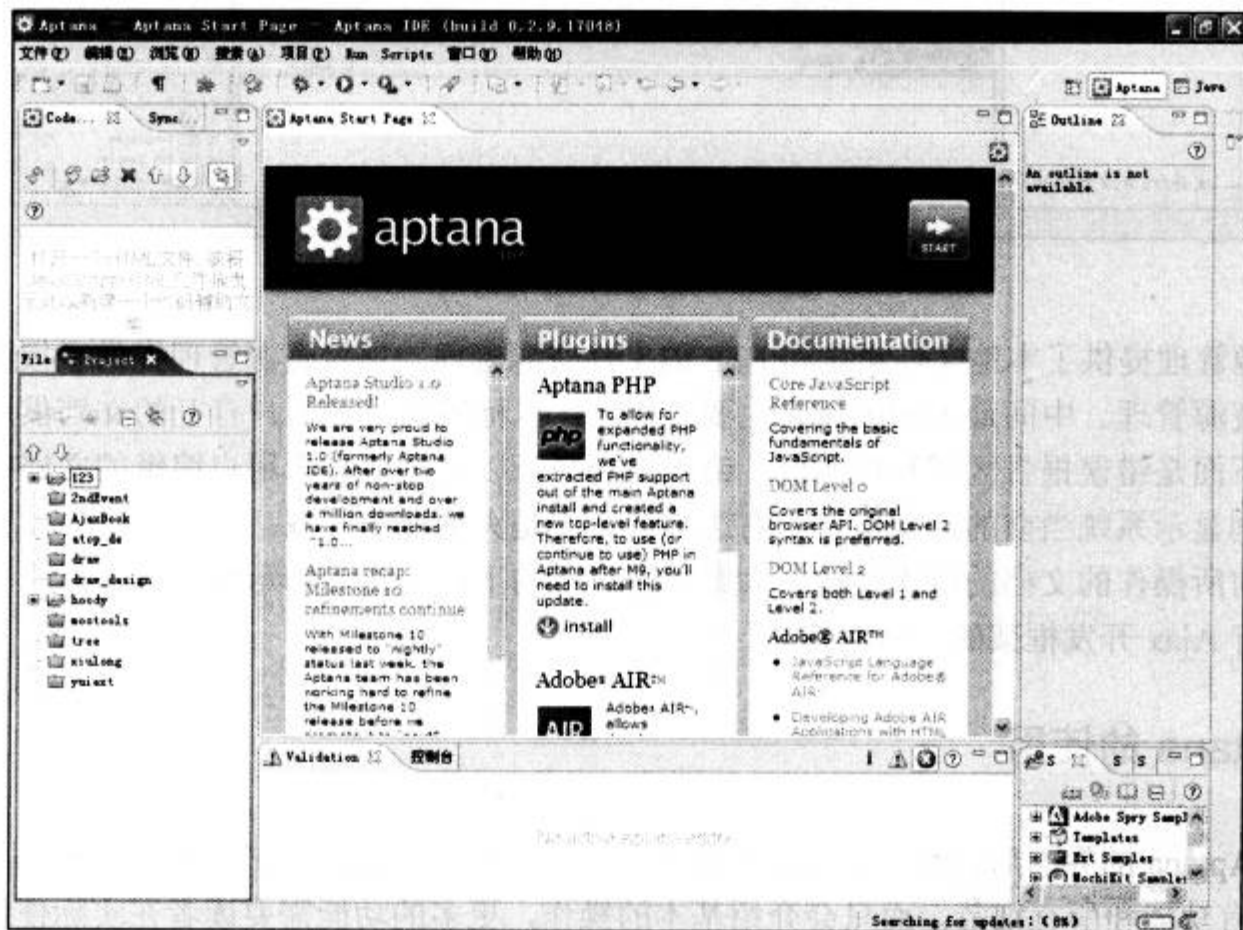


图 5.5 Aptana 界面

Aptana 启动时默认会打开一个叫 Aptana Start Page 的标签页, 其中包含了 Aptana 网站上最新的一些内容。如果用户不需要, 可以将其关闭。

现在来看 Aptana 的工作界面。如图 5.6 所示, 按照界面各部分的功能划分为不同的区域。

Aptana 界面最上面是功能菜单, 功能菜单包含了 Aptana 完整功能的导航。然后是一些快捷操作按钮, 提供了诸如新建和打开文档, 保存、打印、打开插件试图, 调试及运行等功能的快捷操作。左边第一个区域由两个标签页组成, 分别是代码辅助文案和 FTP, 提供了当前文档的辅助信息和 FTP 管理工具。左边下面的区域是文件/项目资源管理工具, 也是由两个标签页组成。

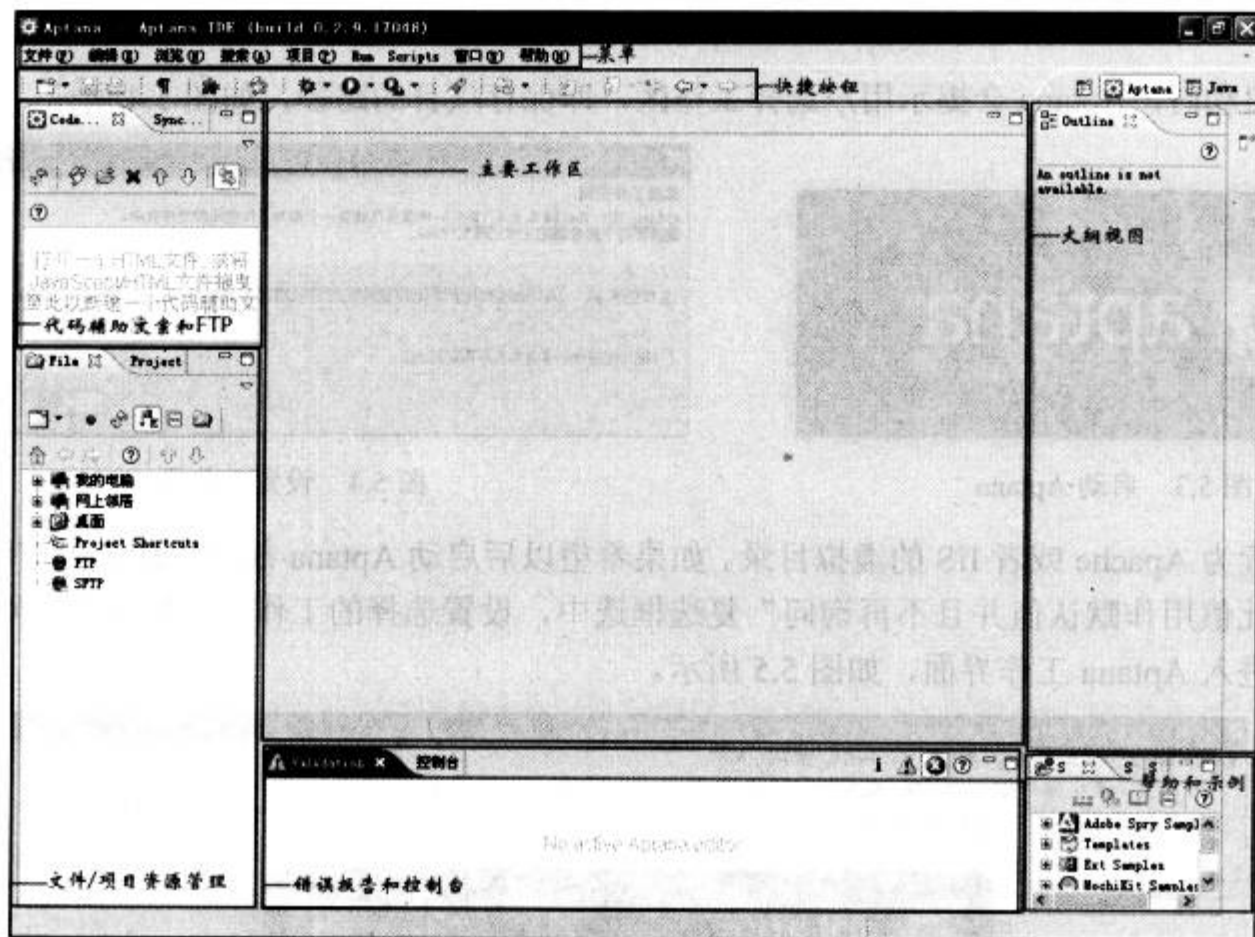


图 5.6 Aptana 的工作区域

文件资源管理提供了本地文件系统和 FTP 的文件资源管理。项目资源管理提供了按照工作区内的项目分组的资源管理。中间最大的区域是主要工作区，以标签页显示用户打开的文档供用户编辑。主要工作区的下面是错误报告区域和控制台。错误报告区可以实时地显示用户编辑的文档中所出现的错误。控制台则显示系统当前的活动信息。右边部分首先是大纲视图区域。这个区域会以大纲视图的形式来显示当前所操作的文档的层次结构。大纲视图区域的下面是帮助和示例区域，显示一些相关的帮助信息和流行 Ajax 开发框架的一些示例。

### 5.1.3 Aptana 的使用

认识了 Aptana 的界面后，本节开始向读者介绍如何使用 Aptana。本书并不会完整地介绍如何使用 Aptana 的所有功能和所有细节，而只会介绍基本的操作，更多的功能需要读者在实际使用过程中去摸索和体验。

#### 1. 创建项目

在开始一个项目时，首先需要创建一个 Aptana 项目。创建项目有 3 种方式。

- (1) 第一种是选择功能菜单的“文件→新建→项目”命令，如图 5.7 所示。
- (2) 第二种方式是使用快捷按钮的新建按钮，如图 5.8 所示。
- (3) 最后一种方式，就是将文件/项目资源管理器切换到项目视图，然后右击视图中的空白区域，在弹出的快捷菜单中选择“新建→项目”命令。



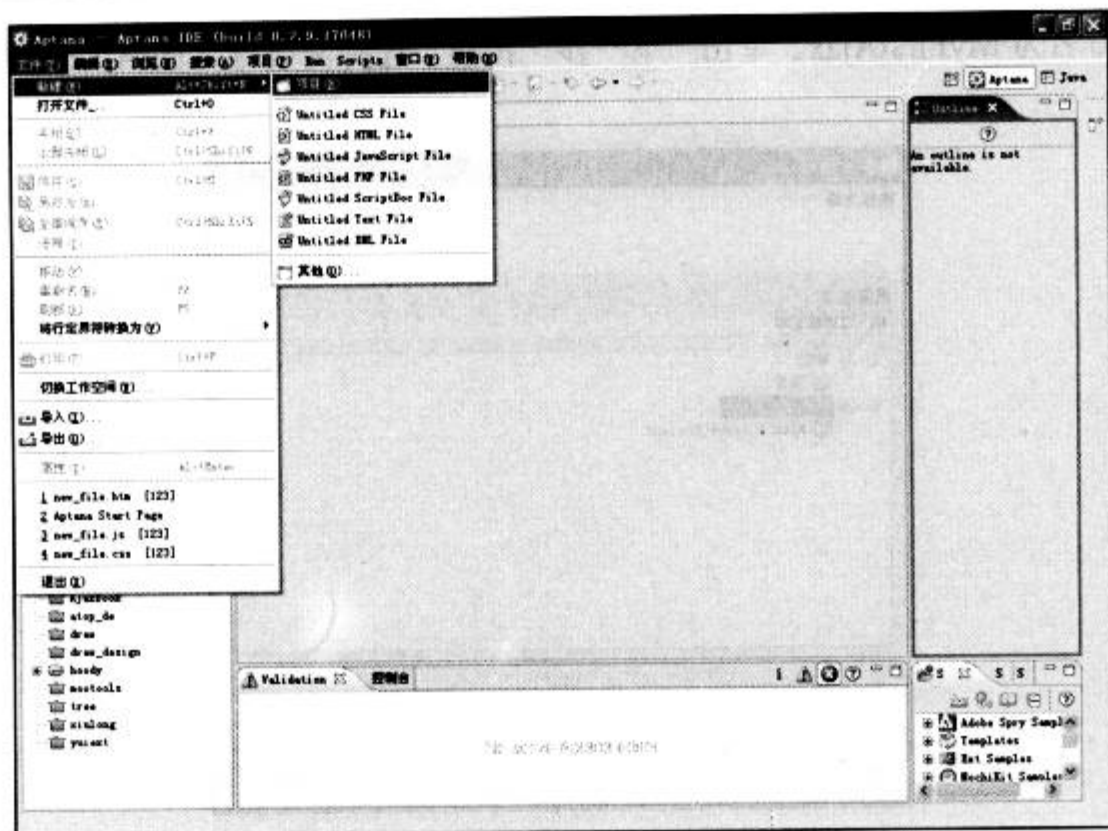


图 5.7 使用功能菜单新建项目

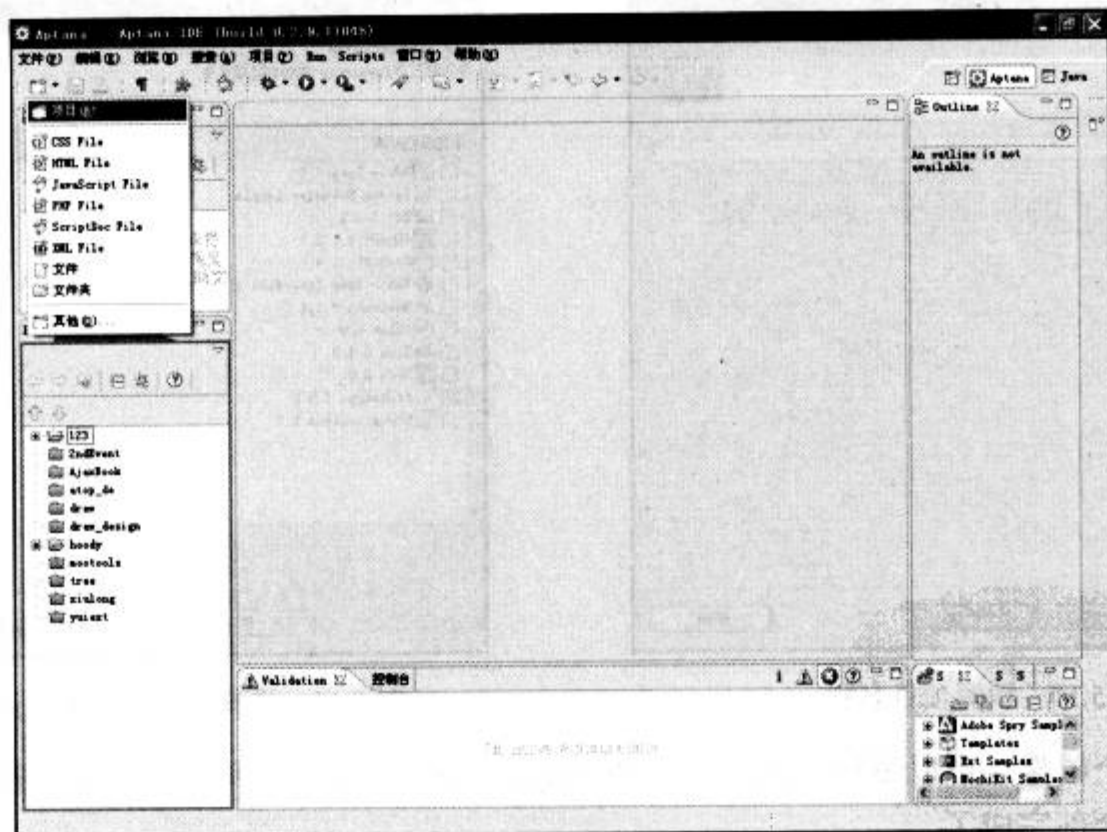


图 5.8 使用快捷按钮新建项目

创建项目时，首先会让用户选择所需要创建的项目类型，如图 5.9 所示。

Aptana 默认提供了常规项目类型和 Ajax Projects 两种项目类型，用户可以通过添加 Aptana 插件来让 Aptana 支持更多的项目类型，例如 Ruby 项目、Rails 项目等。这里选择 Ajax Projects 的 Ajax Library Project，这个项目类型能让用户基于 Aptana 支持的主流 Ajax 开发框架进行项目开发。单击“下一步”按钮，填写项目名称，如图 5.10 所示。

这里将项目命名为 MyFirstAjax。单击“下一步”按钮，选择需要导入的 Ajax 开发框架，如图 5.11 所示。



图 5.9 选择项目类型



图 5.10 填写项目名称

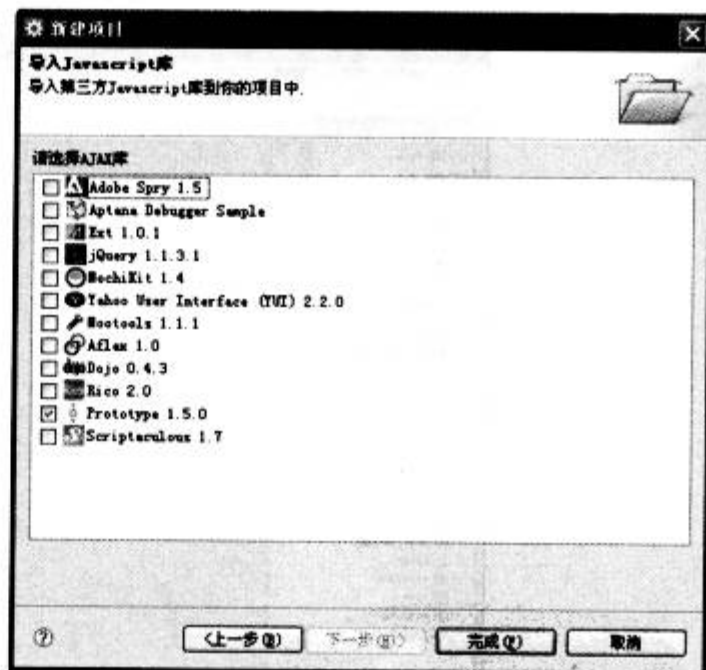


图 5.11 选择需要导入的 Ajax 开发框架

如果需要多个框架，可以多选。这里选中 Prototype 1.5.0 复选框，单击“完成”按钮。到这里，项目的创建工作就全部完成了。

注意：Prototype 是一个优秀的 Ajax 基础开发框架，1.5.0 是其版本号。官方网站是 <http://www.prototypejs.org>。本书将会在第 14 章中向读者介绍这个框架的使用方法。

## 2. 辅助文案

创建完成后，Aptana 自动打开一个 Prototype 框架的示例，页面名称为 prototype\_sample.htm。这时 Aptana 的各个工作区发生了相应的变化，如图 5.12 所示。

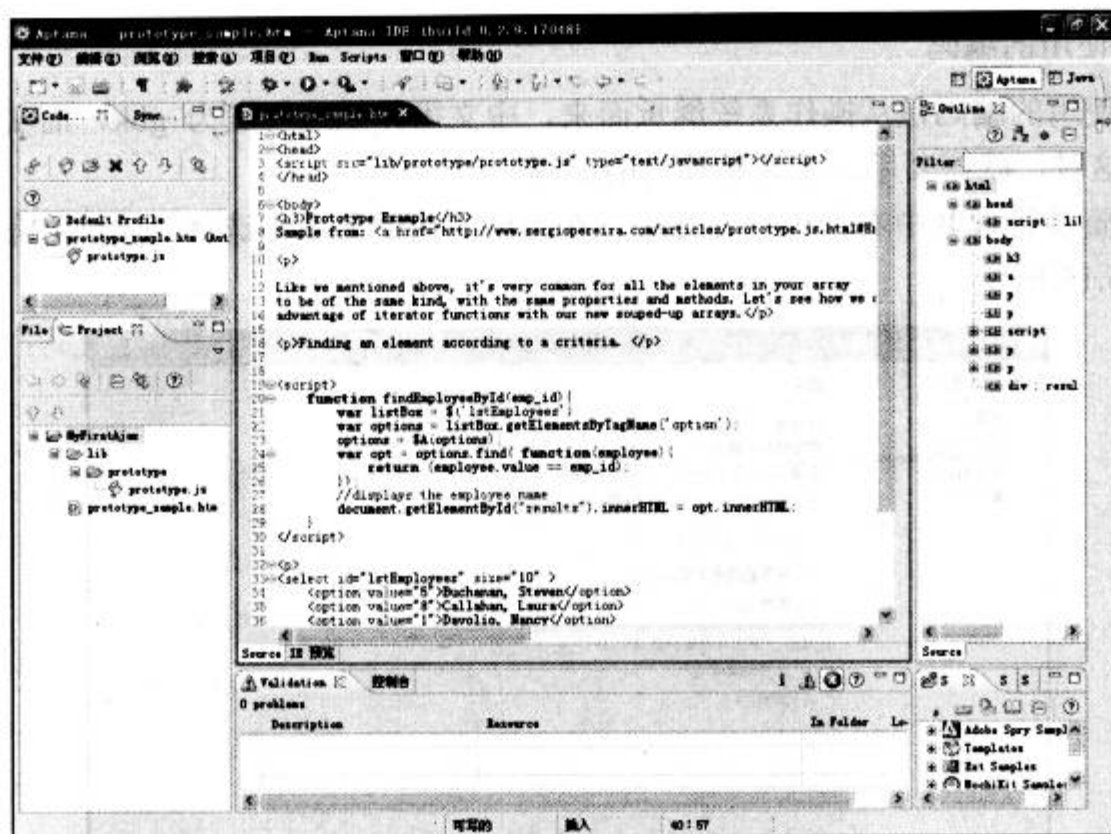


图 5.12 完成创建

首先来看辅助文案区域。辅助文案区域中增加了一个以当前打开的页面名字命名的文件夹，文件夹中是当前页面所引用的文件，图中是 `prototype.js`。将鼠标移动到文件上，会出现一个 tooltip 来显示此文件在文件系统中的位置，如图 5.13 所示。

### 3. 项目资源管理

在项目资源管理区域中，出现了刚才创建的项目，以树形图的方式显示了项目中所包含的所有文件，如图 5.14 所示。

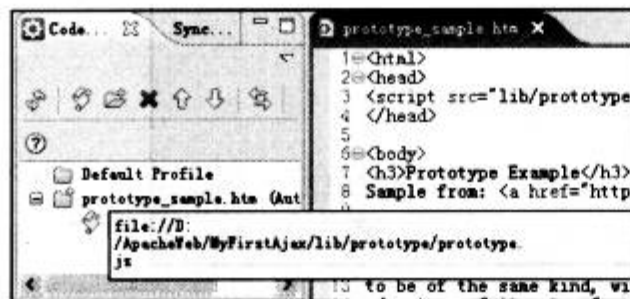


图 5.13 显示文件位置

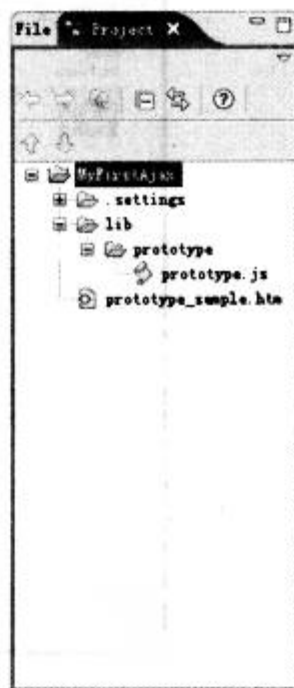


图 5.14 项目资源管理器

右击选择文件或者文件夹，可以进行相关的操作，例如删除、移动、重命名、查看属性等。

#### 4. 更改项目使用的编码

Aptana 默认使用的编码是从操作系统继承而来，中文操作系统一般为 gbk，而实际项目使用的编码通常是 utf-8，这时，就需要手动来调整项目的编码。

(1) 在项目资源管理区中右击项目文件夹，在弹出的快捷菜单中选择“属性”命令，打开项目的属性面板，如图 5.15 所示。

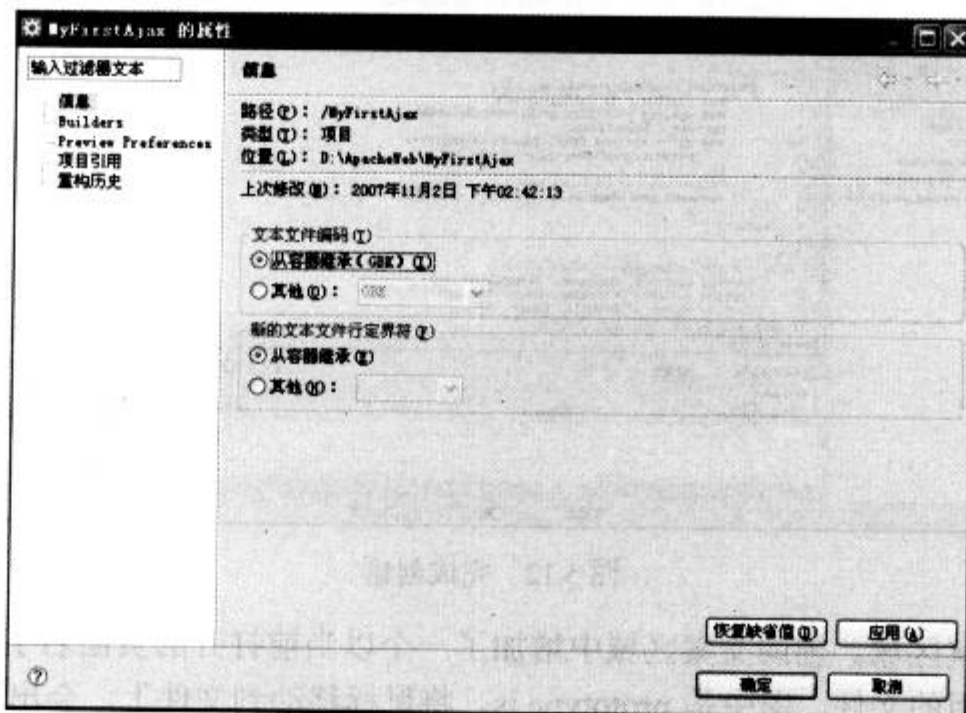


图 5.15 属性面板

(2) 在“信息”栏中的“文本文件编码”选项区域默认选中的是“从容器继承 (GBK)”单选按钮。将该单选按钮切换到“其他”，并在其下拉列表框中选择 utf-8 选项，如图 5.16 所示。

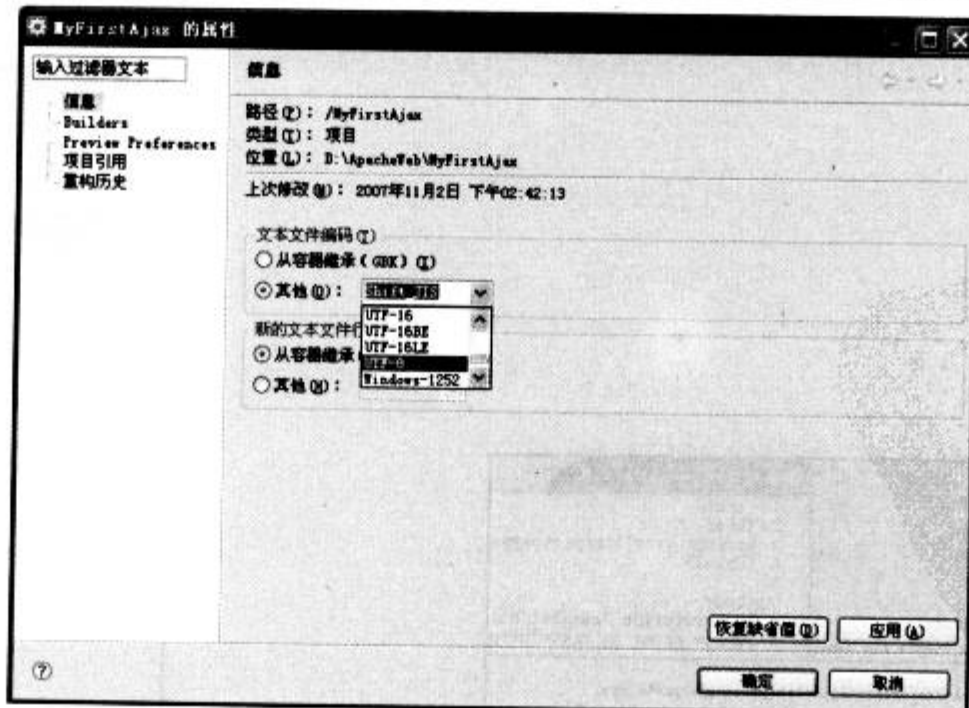


图 5.16 选择文件编码

(3) 然后单击“确定”按钮，保存设置。这时项目使用的编码就变为 utf-8。



## 5. 大纲视图

大纲视图区域以树形图的形式显示了示例文档的层次结构。因为打开的是一个 HTML 文档，所以大纲视图显示的是一个以 html 节点为根的、包含了页面所有元素的树形图，如图 5.17 所示。

在大纲视图中单击元素，可以将主工作区域中打开的文档的焦点定位到所选择的元素上，如图 5.18 所示。

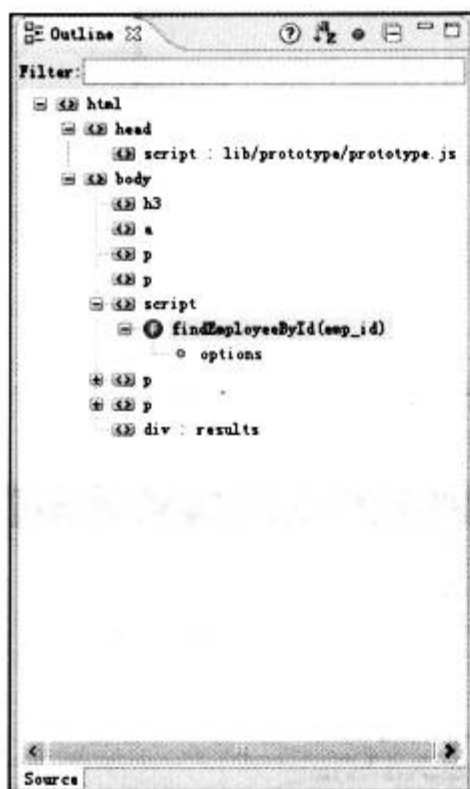


图 5.17 大纲视图

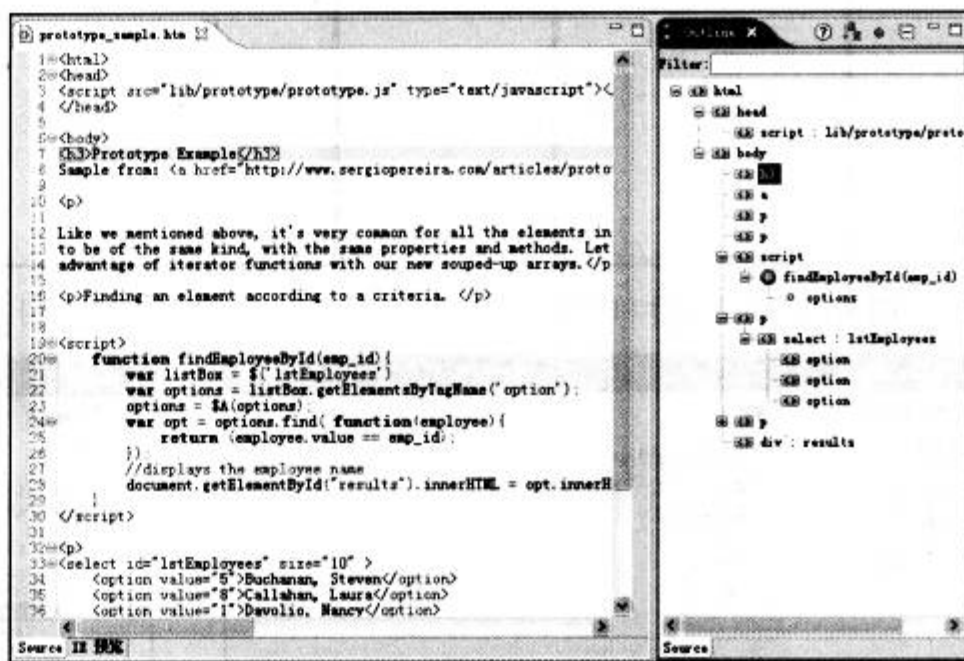


图 5.18 定位元素

在主工作区中，h3 标签被加上了灰色的方框来表示当前在大纲视图中被选中，h3 所在的行背景色也发生了变化。

## 6. 创建文件和文件夹

创建文件和文件夹的方式与创建项目类似，可以使用功能菜单、快捷按钮或是项目资源管理器中的邮件菜单来创建文件。

在 lib 目录下创建一个文件夹，并将其命名为 class，来存放创建的 JavaScript 文件。然后创建一个 JavaScript 文件，将其命名为 test.js，如图 5.19 所示。

## 7. 设置文件默认内容

test.js 被创建后，会自动在主工作区打开，如图 5.20 所示。

可以看到，新建 test.js 文件中被自动加入了一段内容。用户可以更改默认加入的文档内容。

(1) 选择功能菜单中的“窗口→首选项”命令打开首选项面板，如图 5.21 所示。

(2) 打开首选项面板后，在左侧的导航菜单中选择“Aptana→Editors→JavaScript Editor”命令，打开 Aptana 的 JavaScript 编辑器设置，如图 5.22 所示。



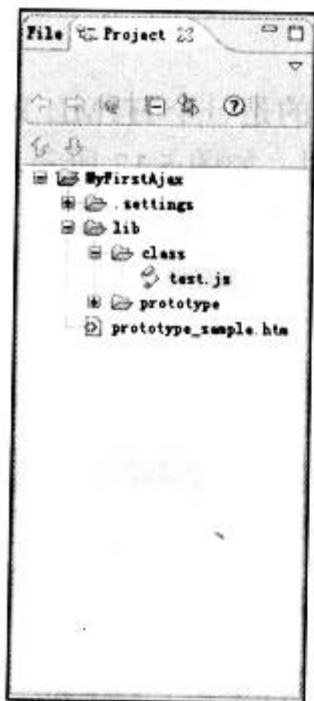


图 5.19 新建文件和文件夹

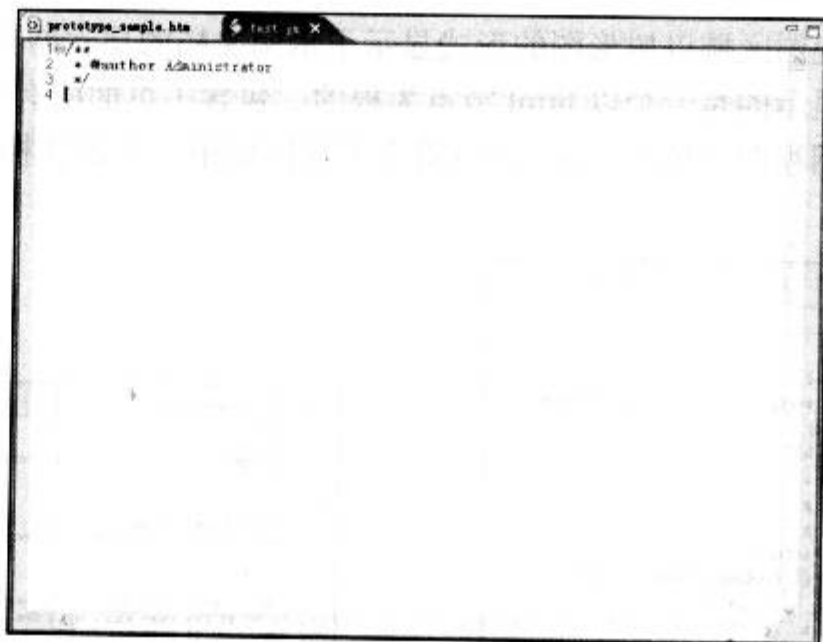


图 5.20 test.js

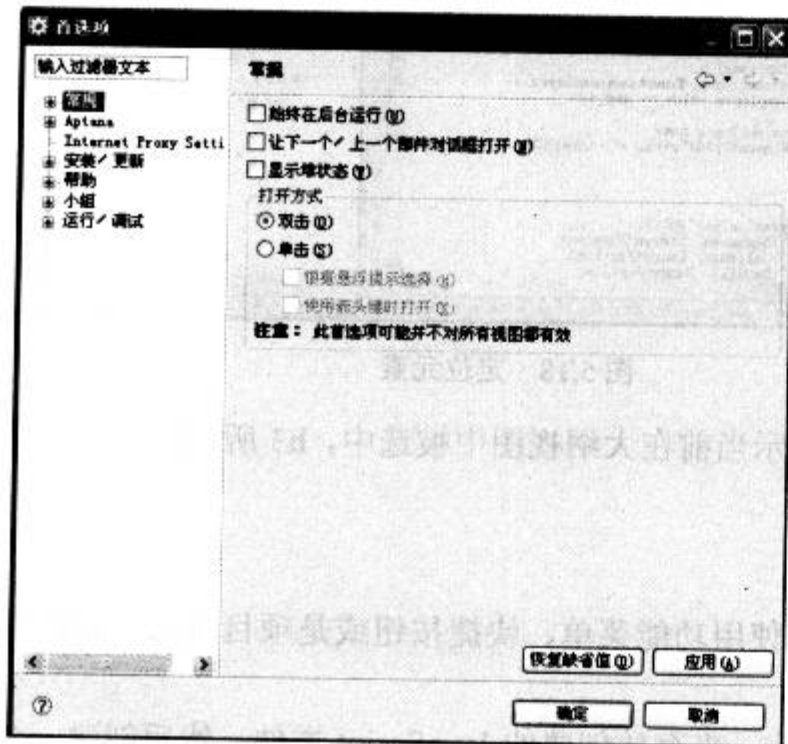


图 5.21 首选项面板

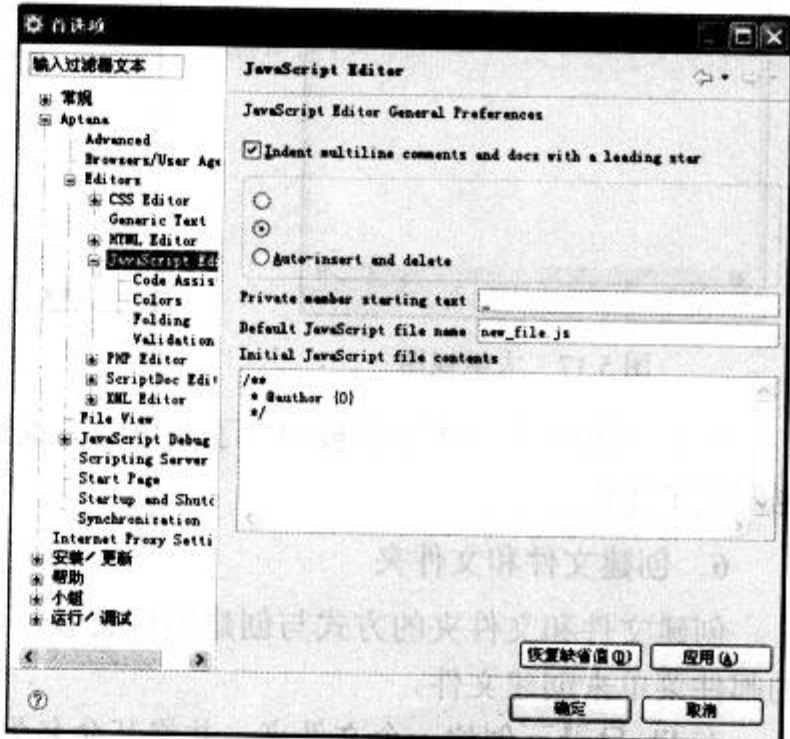


图 5.22 JavaScript 编辑器设置

(3) 在右边的面板中，可以看到 Initial JavaScript file contents 这一项，即文档的默认内容。现在将其修改为如图 5.23 所示的内容并保存。

(4) 保存完毕后，再新建一个 JavaScript 文件 test\_2.js，可以看到，默认加入的内容已经改变为刚才设置的内容，如图 5.24 所示。

(5) 由于 Aptana 在创建 HTML 文档时，默认的 DTD 是 HTML4.0 strict 的，如果用户需要使用其他的 DTD，例如 XHTML，则需要手动修改。为了避免每次修改的麻烦，也可以在 HTML 的默认内容中将 DTD 改变为 XHTML。打开首选项面板，并选择“Aptana→HTML Editor”命令，修改文档默认内容保存即可，如图 5.25 所示。

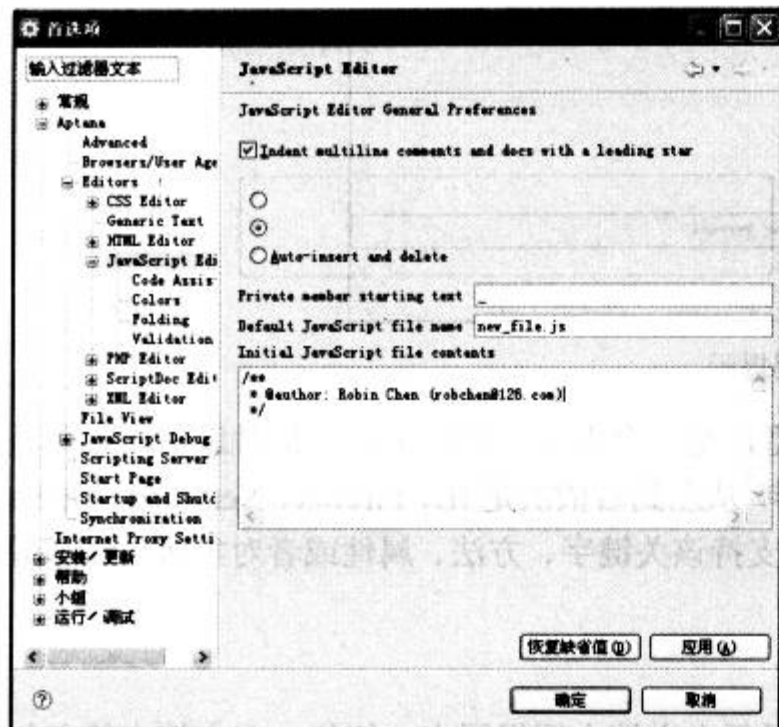


图 5.23 设置默认内容

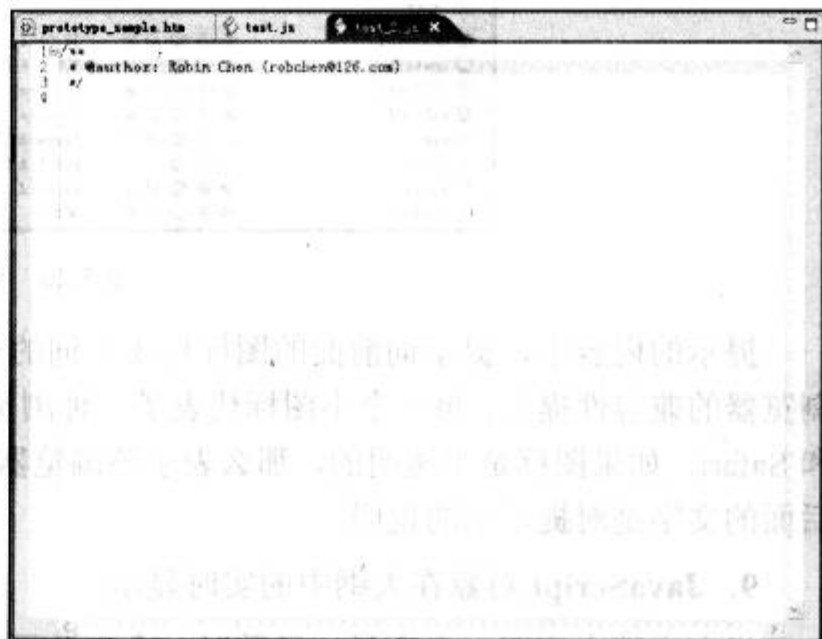


图 5.24 test\_2.js

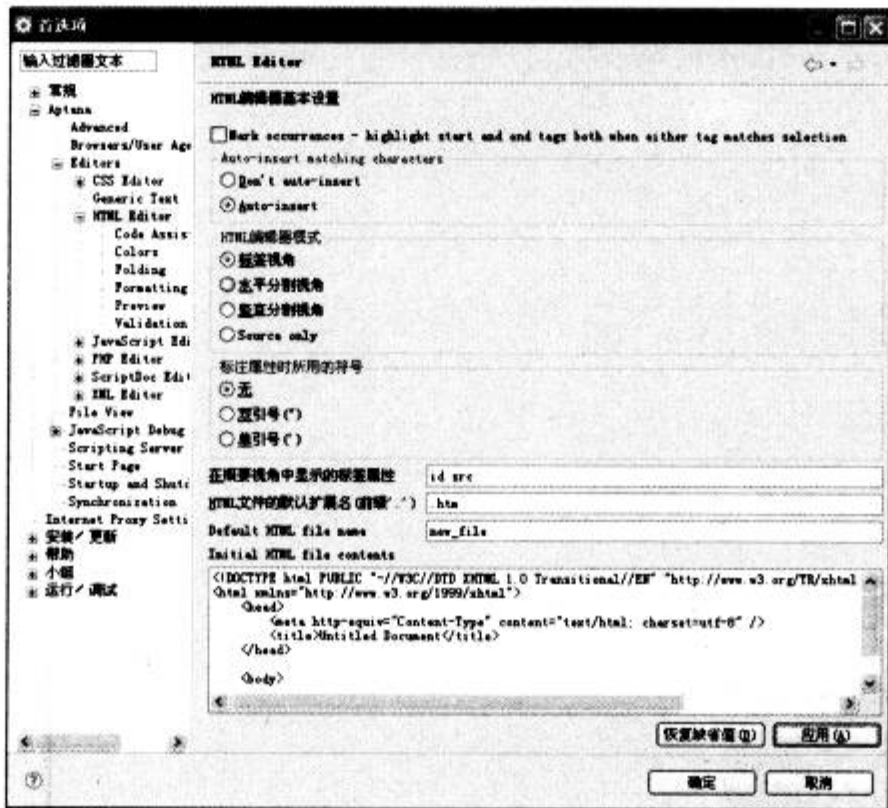


图 5.25 修改 HTML 文档默认内容

首选项还提供了更多的功能设置，读者可以自己进行查看。

## 8. 代码提示功能

Aptana 最让人兴奋的功能之一，就是其 JavaScript 代码提示功能。用 Visual Studio 开发过 C# 或者 VB 应用程序的读者应该会对其实时提示的功能印象深刻。而 Aptana 针对 JavaScript 的代码提示功能也毫不逊色。回到 test\_2.js，在主工作区中进行编辑。读者可以看到，随着键盘的输入，Aptana 会自动根据输入的内容来提示相关的 JavaScript 方法和属性，如图 5.26 所示。

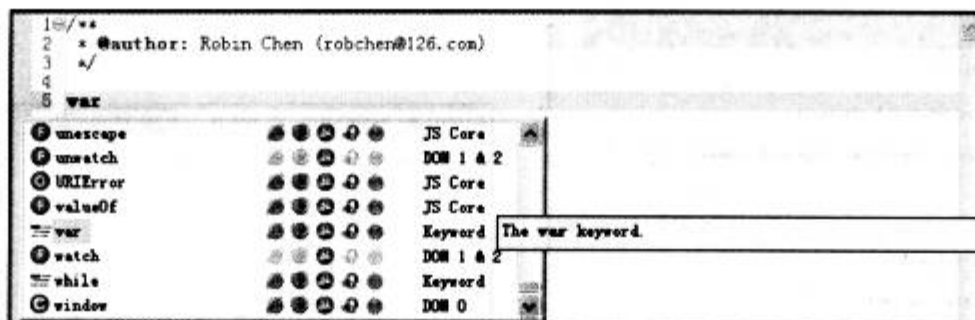


图 5.26 代码提示

提示的内容中，提示词前面的图标标识了词的类型，是一个方法、关键字、还是对象。词后面是浏览器的兼容性提示，每一个小图标代表了一种浏览器，从左到右依次是 IE、Firefox、Netscape、Opera 和 Safari。如果图标是半透明的，那么表示该浏览器不支持该关键字、方法、属性或者对象等。提示最后面的文字是对提示词的说明。

### 9. JavaScript 对象在大纲中的实时显示

当在文档中声明一个变量、函数时，会自动更新显示在文档大纲视图中，例如，在文档中输入如下代码。

```
var a = 1;
var b = [1,2,3,4,5];
var c = {
    p1:1,
    P2:2,
    p3:function()
    {
        return 3;
    }
}
function test()
{
    var a = 1;
    var b = 2;
    return a + b;
}
var b = function()
{
    return 1;
}
```

大纲视图更新为如图 5.27 所示。

### 10. 错误报告

Aptana 还有一个很方便的特性，就是在编辑文档时，如果所编写的程序有语法上的错误，则错误会立即被 Aptana 醒目地标示出来，并在错误报告区域中显示，如图 5.28 所示。



图 5.27 大纲视图



图 5.28 错误提示

#### 5.1.4 Aptana 的更新

当需要更新 Aptana 时,可以选择导航菜单中的“帮助→Check for Aptana Updates Now”命令,Aptana 将会自动连接上更新服务器并检查更新。如果有可用更新,则 Aptana 会弹出提示窗口,让用户确认是否进行更新安装。如果没有提示,则会弹出提示用户当前版本无须更新。

#### 5.1.5 Aptana 的卸载

当需要卸载 Aptana 时,可以进入系统控制面板的添加删除程序,删除 Aptana,如图 5.29 所示。



图 5.29 卸载 Aptana

## 5.2 Firefox

Firefox 是由 Mozilla 基金会开发的一个自由的、开放源代码的浏览器，适用于 Windows、Linux 和 MacOS 平台。它体积小速度快，并且有丰富的免费插件可以使用。Firefox 给普通用户提供了良好的上网体验。Firefox 同样也是面向开发者友好的。Firefox 的一些高级特性，以及丰富多样的免费插件，使得 Firefox 成为了 Web 开发工作者手中的调试利器。最新版本的 Firefox 可以在 <http://www.mozilla.com/> 下载。

### 5.2.1 错误控制台：Error Console

Firefox 提供了错误控制台来显示浏览器解析网页时遇到的错误，包括 JavaScript 错误、CSS 错误等。通过查看错误控制台，可以让开发者迅速了解程序的缺陷。开启错误控制台的方法是在 Firefox 功能菜单中选择“工具→错误控制台”命令。错误控制台的界面如图 5.30 所示。

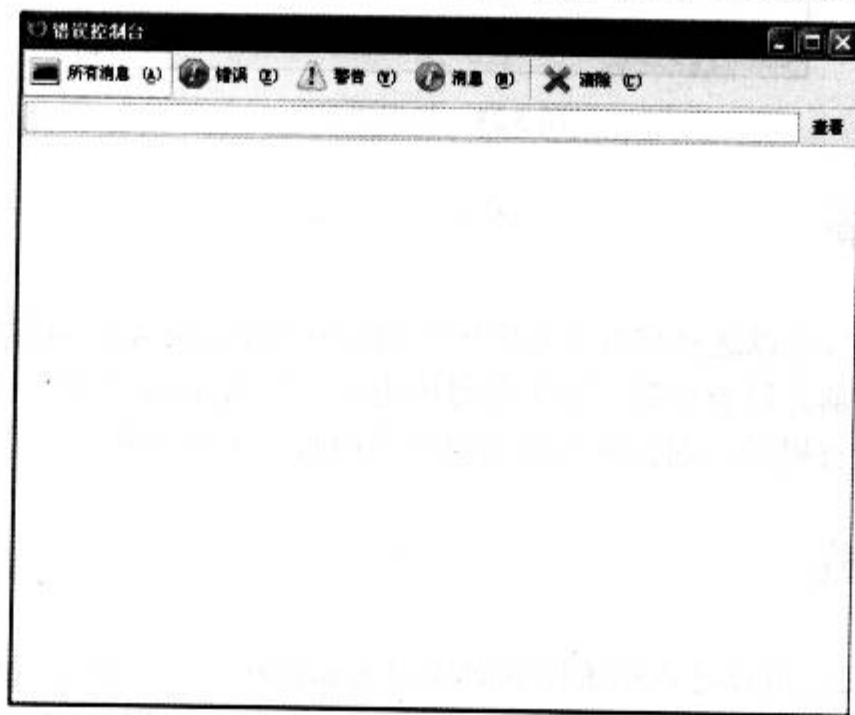


图 5.30 错误控制台的界面

错误控制台中的信息分为 3 种：错误、警告和消息。错误控制台提供了相应的 3 个按钮来查看这 3 类信息，还提供了一个“所有消息”按钮来查看所有的信息。在这 4 个按钮右边，是一个清除按钮，可以用来清除当前显示的信息。

### 5.2.2 优秀的调试插件：Firebug

#### 1. 安装 Firebug

Firebug 是 Firefox 下的一款开发类插件，它集 HTML 查看和编辑、JavaScript 控制台、网络状况监



视器于一体，是开发和调试 JavaScript、CSS、HTML 和 Ajax 的得力助手。只需要用 Firefox 访问 <https://addons.mozilla.org/en-US/firefox/addon/1843> 并单击页面上的 Install Now 按钮，就可以完成 Firebug 的下载和安装，如图 5.31 和图 5.32 所示。

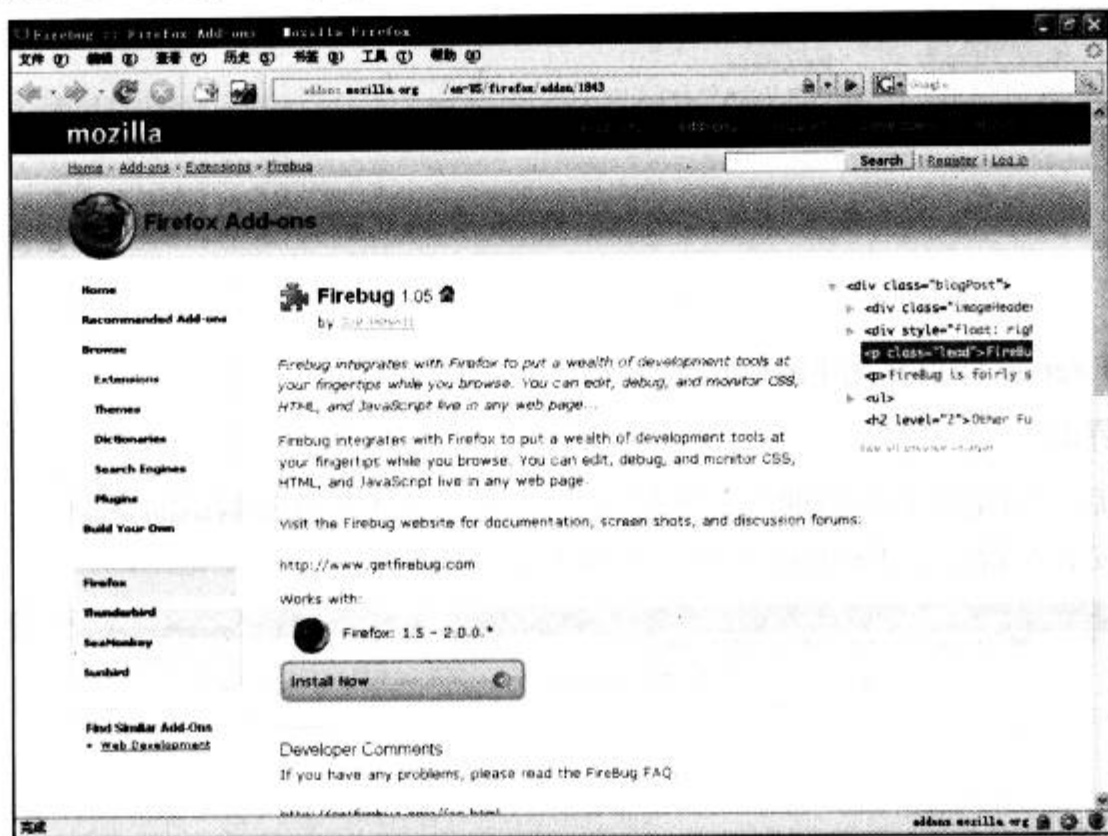


图 5.31 下载 Firebug

安装完成后，需要重启 Firefox，如图 5.33 所示。



图 5.32 安装 Firebug

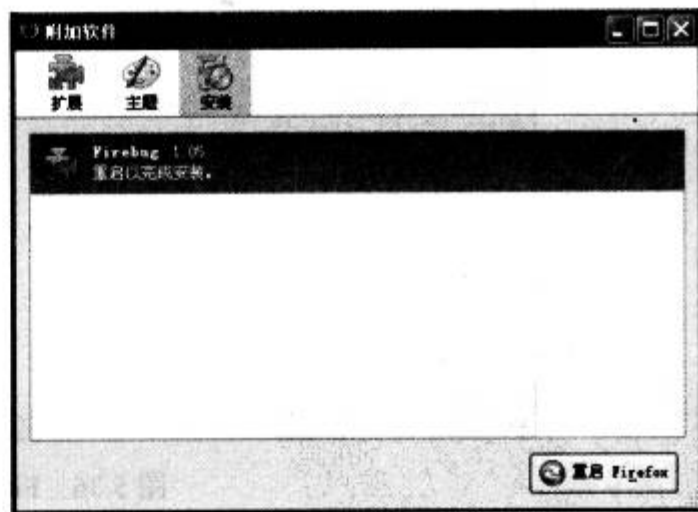


图 5.33 重启 Firefox

## 2. 启用 Firebug

Firebug 安装后，默认是关闭的。这时，在 Firefox 右下角可以看见一个灰色的图标，如图 5.34 所示。

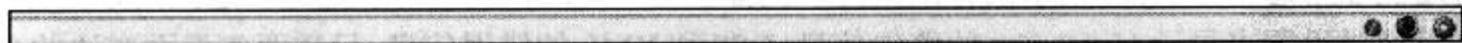


图 5.34 灰色图标

单击灰色图标，弹出 Firebug 界面，界面上提示 Firebug 被禁用，如图 5.35 所示。

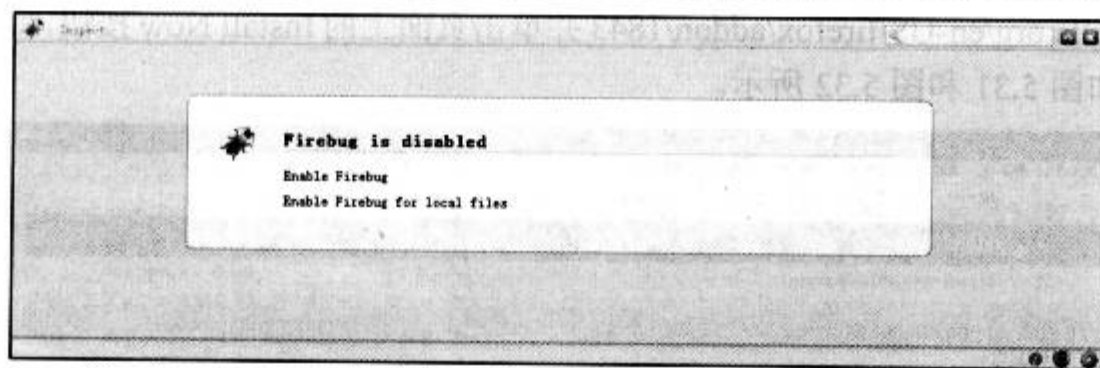


图 5.35 启用 Firebug

单击 Enable Firebug 即可启用 Firebug。

### 3. Firebug 界面

Firebug 启用后，浏览器右下角的灰色图标变为绿色。在打开需要测试的页面后，可以单击右下角绿色的图标或者按 F12 键打开 Firebug 界面，如图 5.36 所示。

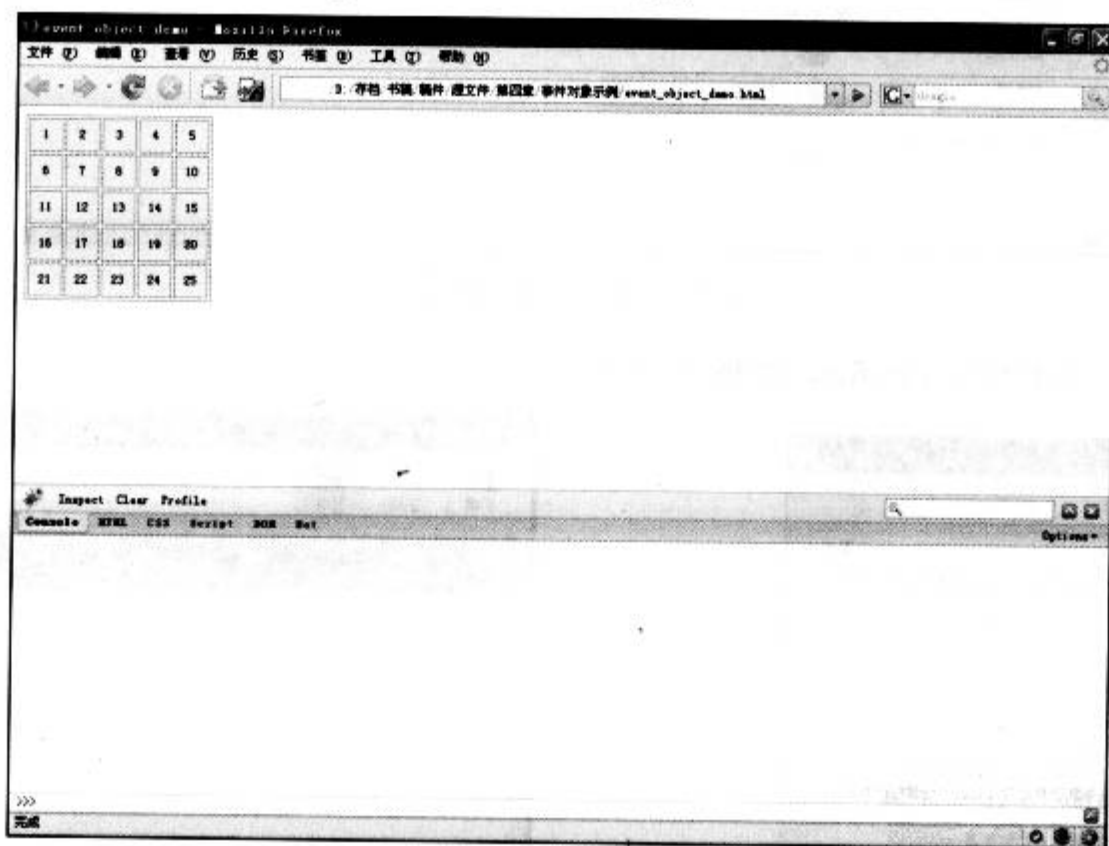


图 5.36 Firebug 工作界面

Firebug 有 6 个主要的标签按钮：Console（控制台）、HTML（HTML 查看器）、CSS（CSS 查看器）、Script（脚本查看器）、DOM（DOM 查看器）和 Net（网络状况监视器）。

### 4. Console（控制台）

控制台能够显示当前页面中的 JavaScript 错误信息，并且能提示出错的文件和行号，以方便调试。控制台还能显示每一个 Ajax 请求的详细信息，例如参数、HTTP 头、请求地址、返回的数据等。控制台还可以通过设置来显示 JavaScript 的警告信息、CSS 和 XML 的错误信息，只需要在界面右边的 Options 菜单中将需要显示的内容选中即可，如图 5.37 所示。

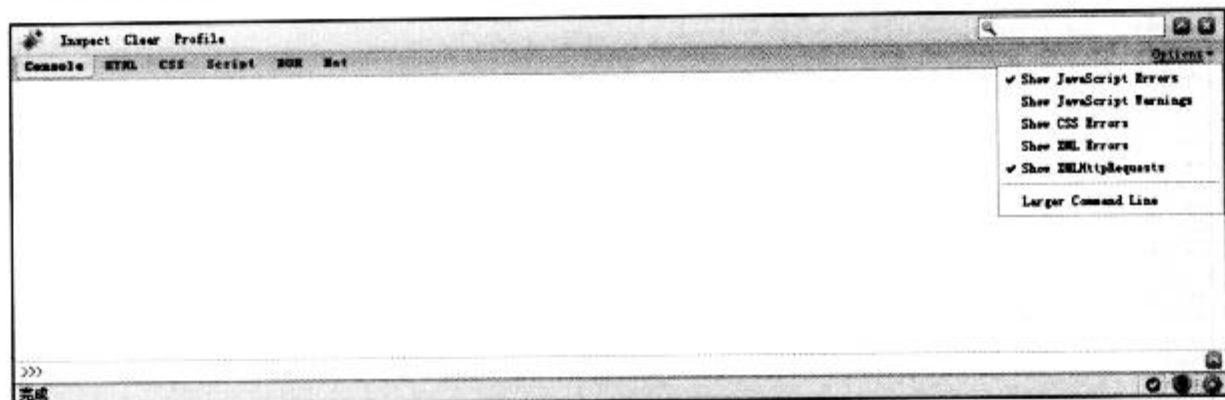


图 5.37 设置控制台的显示信息

控制台还有一个重要的功能，就是可以查看脚本的日志。有过 JavaScript 开发经验的读者，一定不会对调试程序时反复使用 `alert` 陌生。Firebug 的控制台提供了另一个选择，就是 `console.log`。最简单的打印日志的语法如下所示。

```
console.log("Hello World");
```

如果需要同时输入多个参数，也可以使用这样的语法。

```
console.log(1,3,8,"Ajax",test);
```

下面用一个示例来演示日志的使用。代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>console log demo</title>
<script type="text/javascript">
console.log('start loop');
var sum = 0;
for(var i = 0; i < 10; i++)
{
    console.log('loop' + i);
    sum += i;
    console.log('sum is' + sum);
}
console.log('end loop');
</script>
</head>

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

用 Firefox 打开页面，在 Firebug 的控制台中输出了日志信息，如图 5.38 所示。

Firebug 的日志输出有多种可选的格式和语法，甚至可以定制颜色输出，比起单调的 `alert` 会更加方便。关于日志输出更详细的介绍可参见本书的第 8 章：调试技巧。

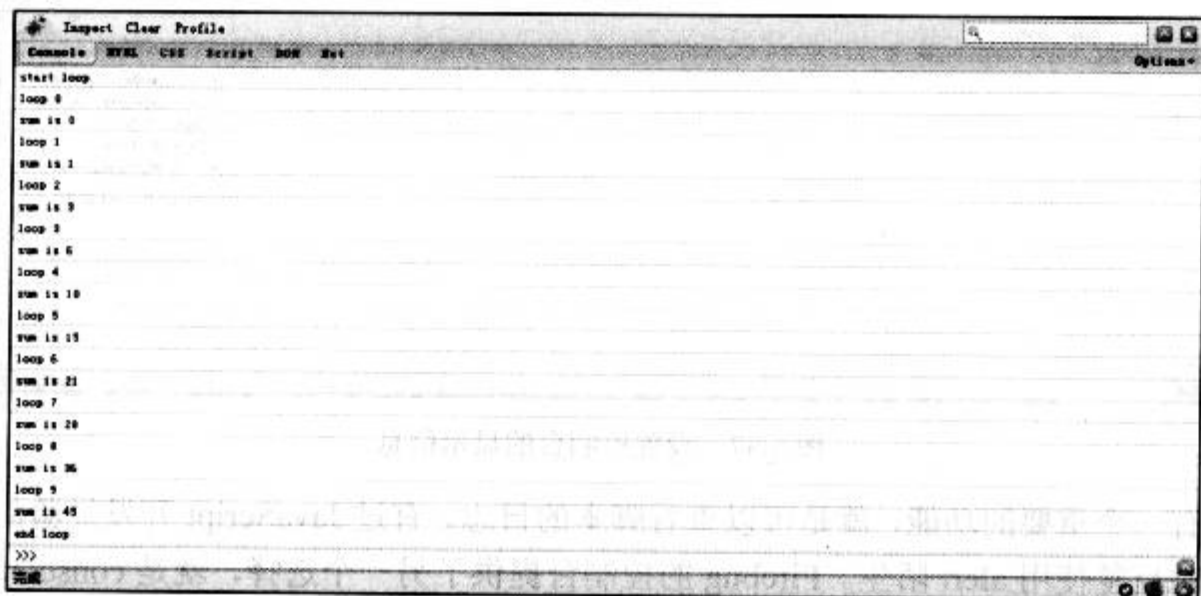


图 5.38 控制台日志

## 5. HTML 查看器

HTML 查看器显示的是经过格式化的当前文档的 HTML 代码，以树形结构显示了节点之间的从属关系。源代码上方还标识出了 DOM 的层次，清楚地列出了当前选择元素的父子节点以及根节点，如图 5.39 所示。

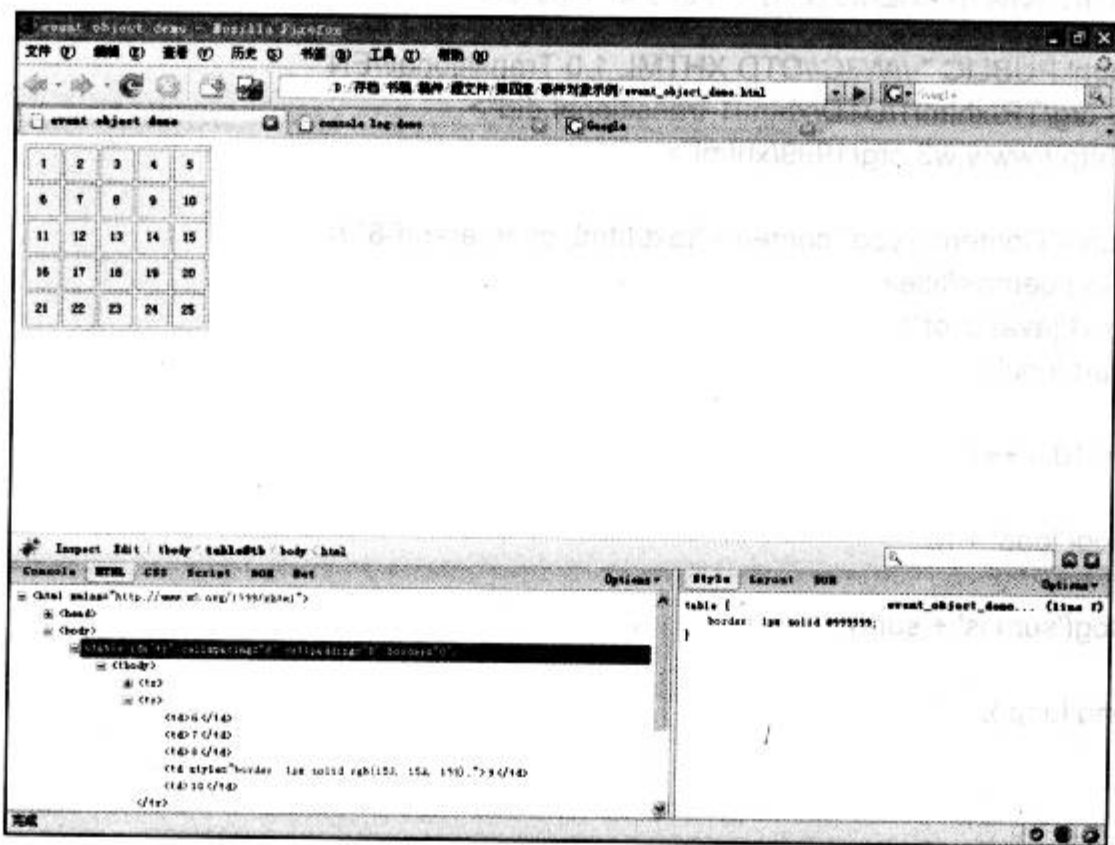


图 5.39 HTML 查看器

还可以直接右击页面上需要检查的元素，在弹出的快捷菜单中选择 **Inspector Element** 命令将 HTML 查看器定位到该元素上，如图 5.40 所示。

与浏览器默认的查看源代码不同，HTML 查看器所呈现的内容能够体现 HTML 的动态变化。如果页面中的脚本改变了 HTML 元素的结构或者样式，都可以实时地在 HTML 查看器中得到体现。HTML 查看器不仅可以查看，还可以对 HTML 内容进行编辑，并将编辑的结果实时更新到页面上。双击节点

可以切换到代码编辑视图, 如图 5.41 所示。



图 5.40 直接选择元素

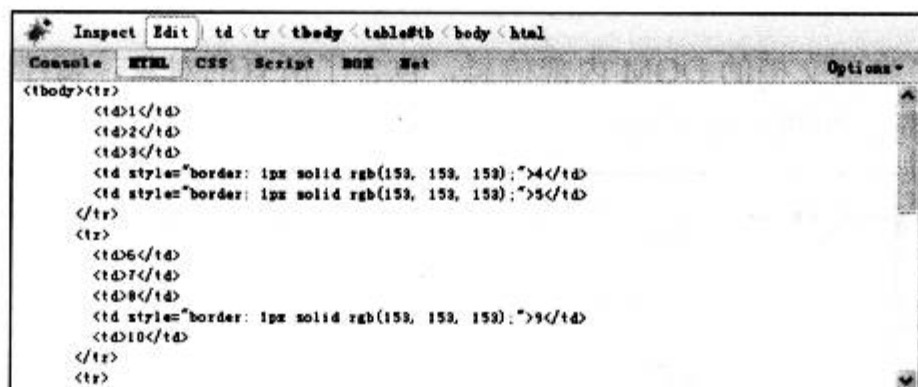


图 5.41 编辑视图

## 6. CSS 查看器

CSS 查看器显示了当前页面上所有被定义的 CSS 样式, 包含了通过引入 CSS 文件定义的、<style> 标签内定义的样式、元素 style 属性中定义的样式、通过节点的 style 对象定义的样式, 如图 5.42 所示。

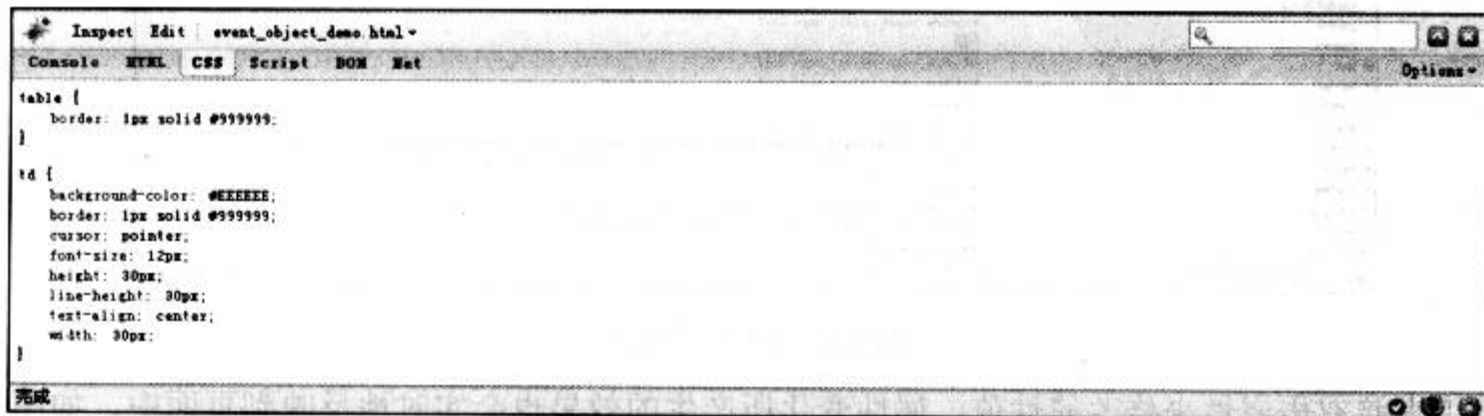


图 5.42 CSS 查看器



同 HTML 查看器一样, CSS 查看器同样可以实时显示页面样式的变化。而且可以直接在 CSS 查看器中编辑样式, 页面上会根据编辑的结果实时地做出变化, 如图 5.43 所示。

## 7. 脚本查看器

脚本查看器提供了页面中脚本的查看和调试的功能。脚本查看器的界面虽小, 但是单步调试、设置断点、变量查看一个不少, 如图 5.44 所示。



图 5.43 编辑 CSS 样式

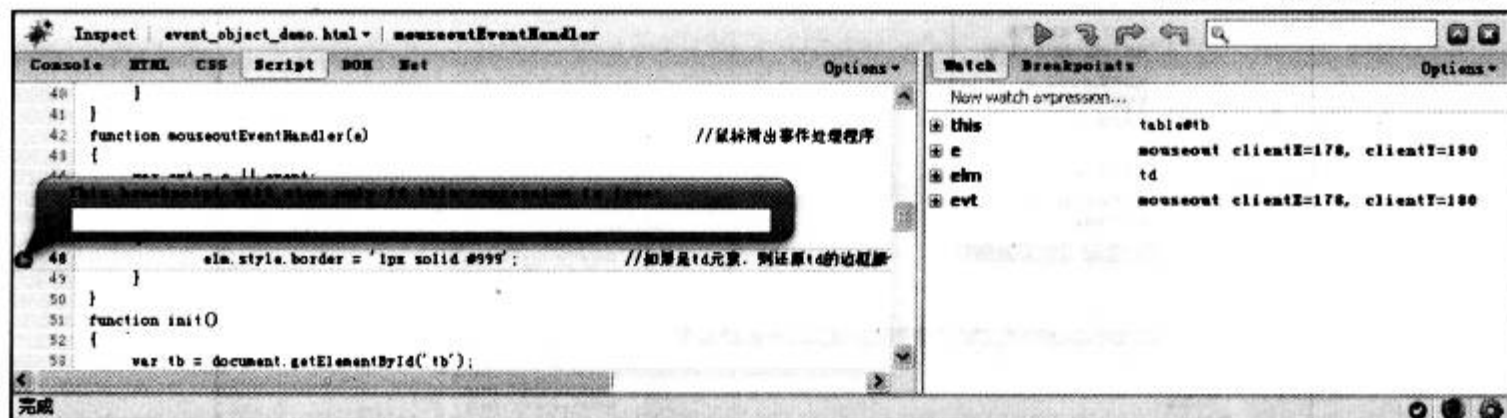


图 5.44 脚本查看器

## 8. DOM 查看器

DOM 查看器显示了当前文档的 DOM 内部信息, 包含了所有的对象、属性和事件, 并且可以根据脚本的操作而实时地更新, 如图 5.45 所示。

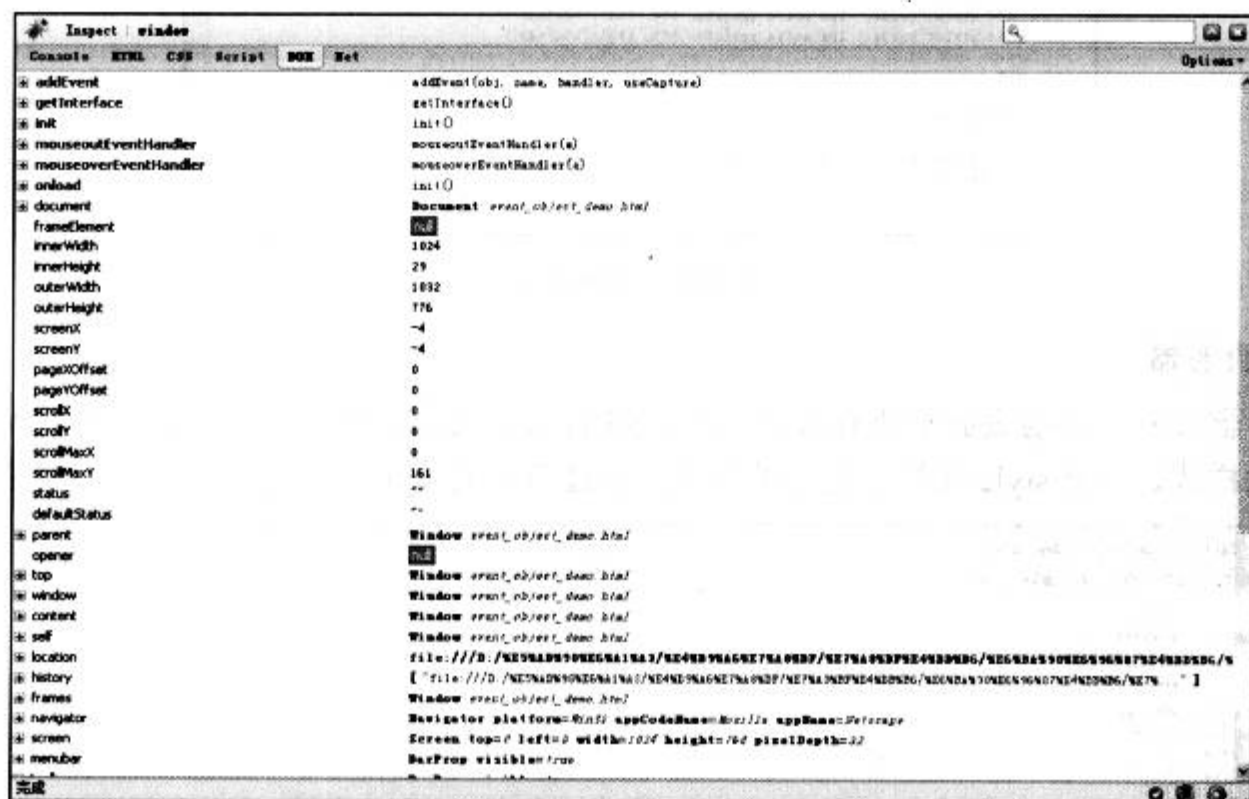


图 5.45 DOM 查看器

可以直接双击属性来修改属性值, 属性变化所产生的效果也会实时地反映到页面中, 如图 5.46

所示。

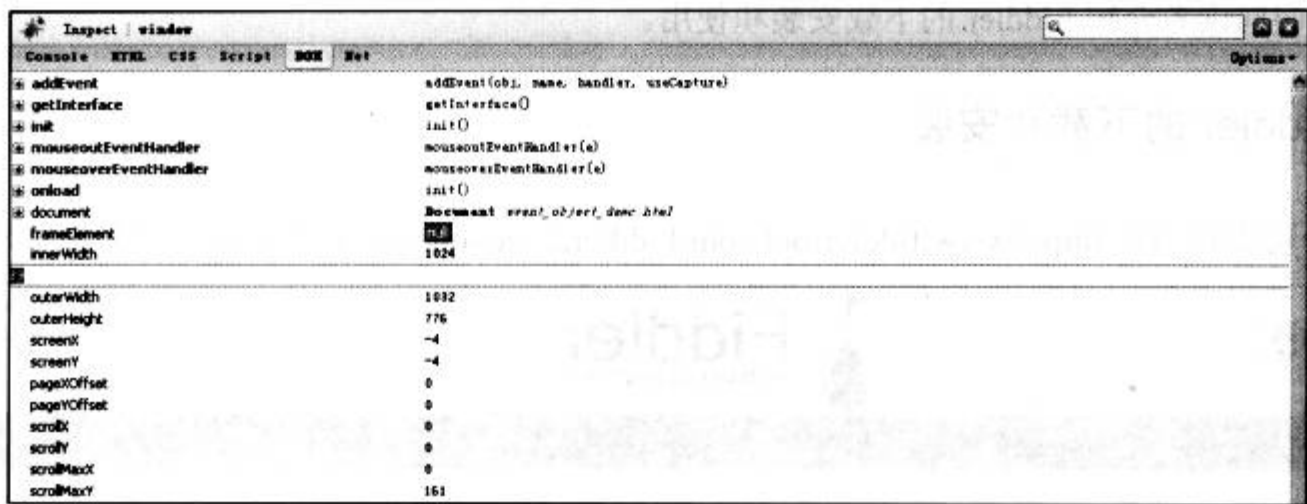


图 5.46 修改 DOM 属性

## 9. 网络状况查看器

网络状况查看器显示了在打开一个页面时所发生的所有 HTTP 请求及其细节情况。例如请求的 HTTP 头和输出的数据、请求所花费的时间、所下载的文件大小、域名等。还提供了请求数量、数据流量和花费时间的统计。网络状况查看器可以让开发者很方便地测试程序的加载性能，如图 5.47 所示。

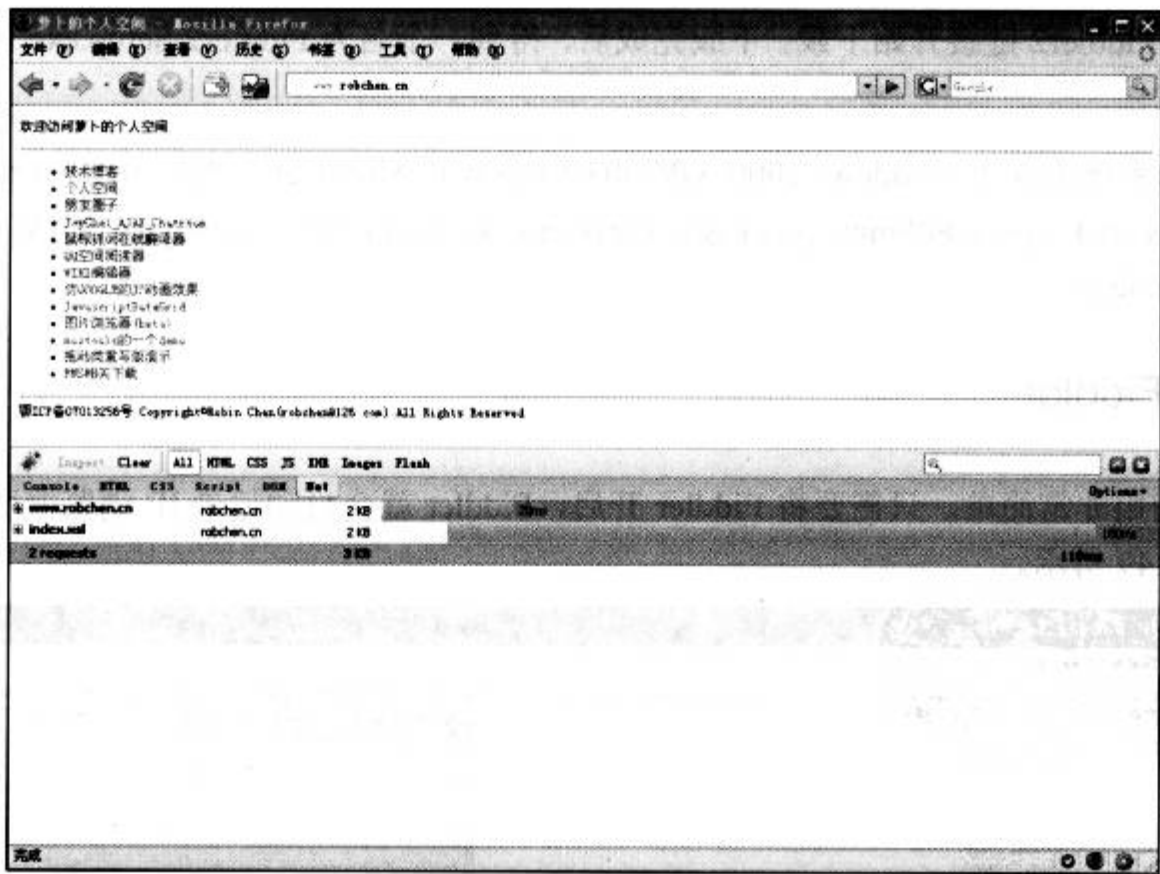


图 5.47 网络状况查看器

## 5.3 HTTP 调试工具: Fiddler

Fiddler 是一款由微软公司开发的 HTTP 调试工具。虽然 Firebug 已经提供了 HTTP 调试的功能，但

是 Firebug 只能在 Firefox 下更好地使用。当需要在 IE 系列浏览器下进行 HTTP 调试时, 则可以使用 Fiddler。下面向读者介绍 Fiddler 的下载安装和使用。

### 5.3.1 Fiddler 的下载和安装

Fiddler 可以在网站 <http://www.fiddlertool.com/Fiddler2/version.asp> 免费下载, 如图 5.48 所示。

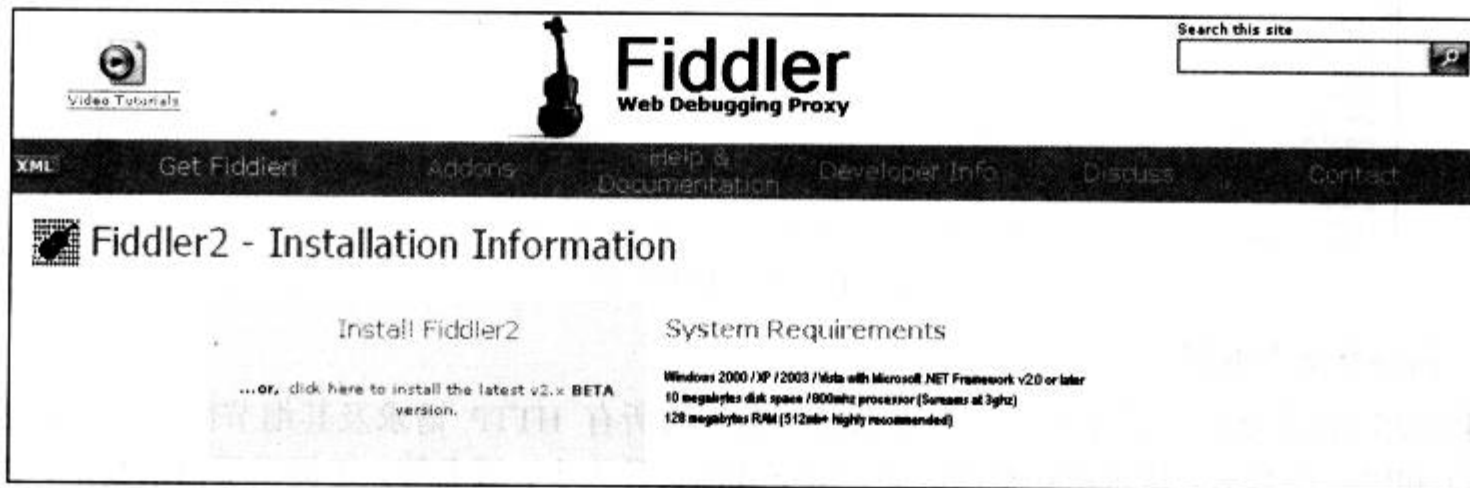


图 5.48 下载 Fiddler

单击 Install Fiddler2 链接开始下载。下载完成后, 得到一个名为 Fiddler2 Setup.exe 的安装文件, 运行安装即可。

**注意:** Fiddler 的系统要求为 Windows 2000/XP/2003/Vista with Microsoft .NET Framework v2.0 or later 10 megabytes disk space/800mhz processor (Screams at 3ghz) 128 megabytes RAM (512mb+highly recommended)。

### 5.3.2 使用 Fiddler

Fiddler 的使用非常简单。只需要将 Fiddler 开启, Fiddler 就会自动记录 IE 浏览器所有 HTTP 请求的信息, 如图 5.49 所示。

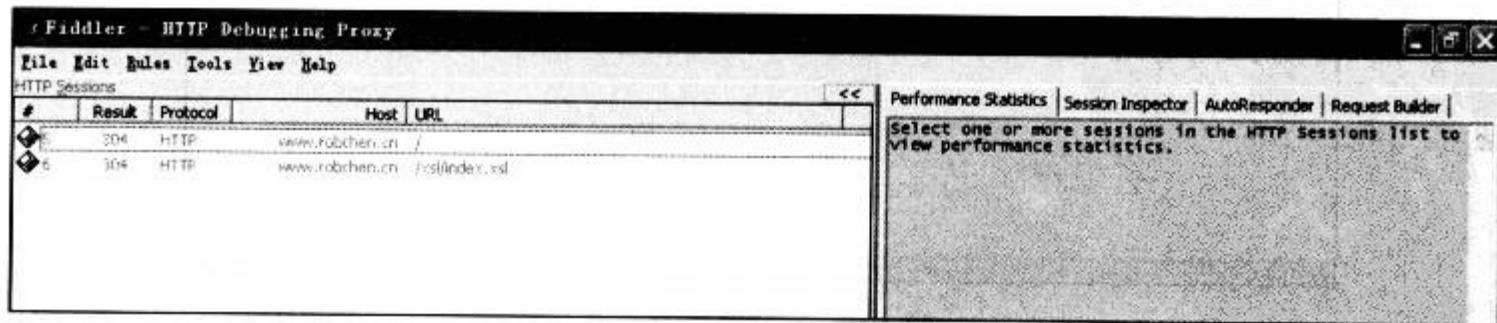


图 5.49 使用 Fiddler

界面左边部分是所有 HTTP 请求的记录。需要查看某条请求的详细信息, 只需要单击该条记录, 则记录的详细信息就会被显示在界面右边的区域中, 如图 5.50 所示。

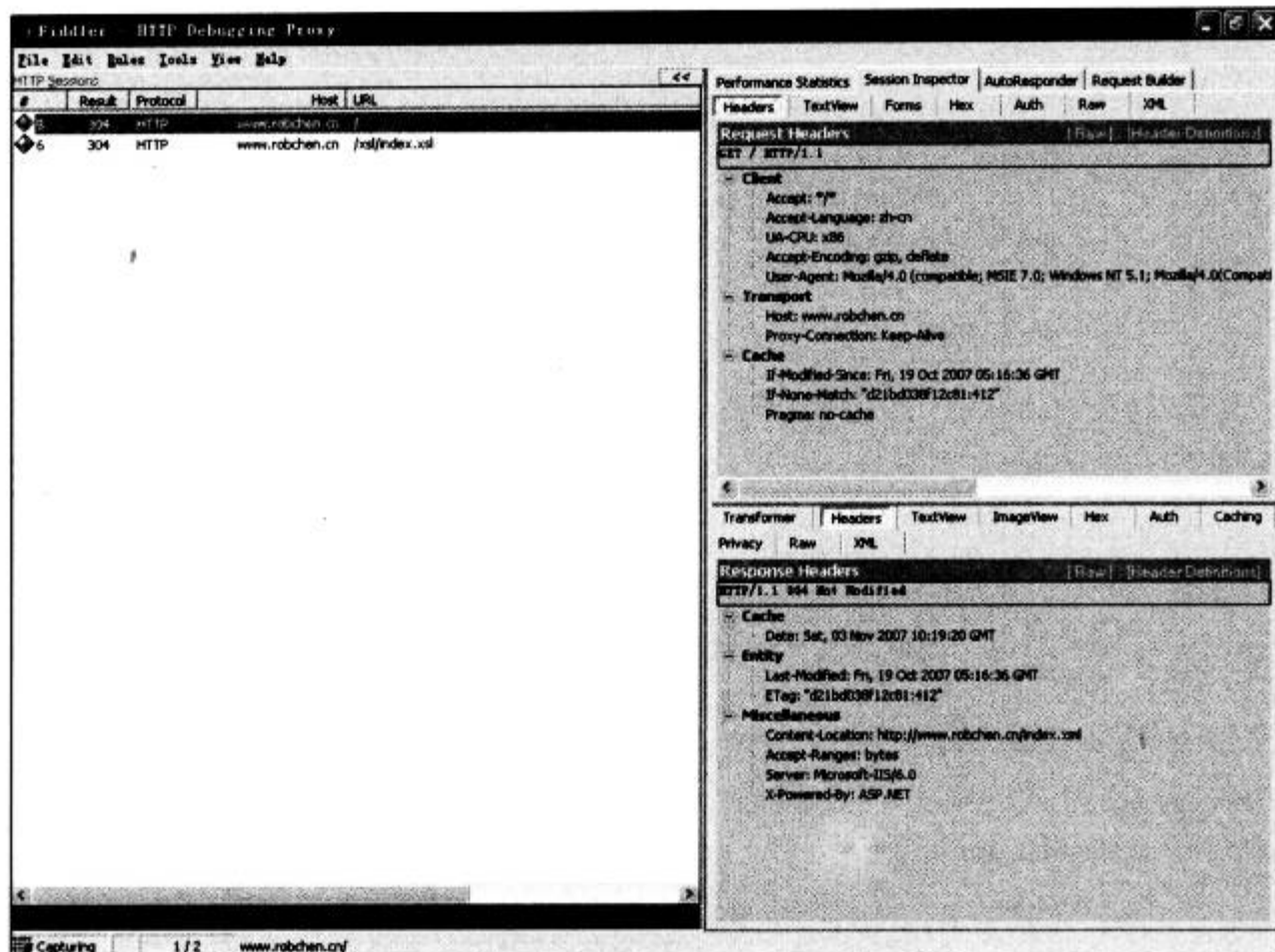


图 5.50 查看 HTTP 请求

## 5.4 小 结

本章向读者介绍了开发 Ajax 应用程序需要使用的工具，包括专业的 JavaScript 开发工具：Aptana 的界面介绍和使用方法；优秀的 Web 浏览器 Firefox，以及其错误控制台的使用；还有调试插件 Firebug 的详细介绍。最后介绍了在 IE 下使用的 HTTP 调试工具 Fiddler。







# 第 3 篇



## Ajax 应用技术分析篇

第6章 Hello World! 分析

第7章 完善的Ajax应用程序: Ajax留言本

第8章 调试技巧

第9章 常见问题

# 第3章

Ajax 应用案例

DESIGN

第6章 Hello World! 介绍

第7章 完善的Ajax应用案例：Ajax留言板

第8章 进阶知识

第9章 常见问题

# 第 6 章

## Hello World! 分析

- » XMLHttpRequest 对象详解
- » 搭建基本的 Ajax 开发框架
- » 小结

本书的前面几章向读者介绍开发 Ajax 应用程序所需要具备的基础知识和开发中经常用到的一些工具。从本章开始，正式向读者介绍 Ajax 的相关知识。在第 1 章中向读者展示了一个简单的 Ajax 应用程序实例：Hello World!，本章主要对该程序的各个组成部分进行分析。并将结合实例重点介绍 XMLHttpRequest 对象的属性和方法，并介绍如何搭建基本的 Ajax 应用框架。最后，再给 Hello World! 程序添加一些新的功能。

## 6.1 XMLHttpRequest 对象详解

回顾第 1 章中的 Ajax 示例程序 Hello World! 的内容。

```
try
{
    var xmlhttp = new XMLHttpRequest();
}
catch(e)
{
    var xmlhttp = new ActiveXObject('Microsoft.XMLHTTP');
}
xmlhttp.open('GET','hello_world.php',true);
xmlhttp.onreadystatechange = function()
{
    if(xmlhttp.readyState == 4 && xmlhttp.status == 200)
    {
        alert(xmlhttp.responseText);
    }
}
xmlhttp.send();
```

可以看到，所有的操作都是围绕 xmlhttp 这个对象展开的，而 xmlhttp 是 XMLHttpRequest 对象的一个实例。

XMLHttpRequest 对象是当今所有 Ajax 和 Web 2.0 应用程序的技术基础。尽管各大软件厂商和开源社团都推出了各种 Ajax 开发框架，以此来简化 XMLHttpRequest 对象的使用，但是详细了解 XMLHttpRequest 的使用方法，对于学习 Ajax 是一个必要的过程。

下面开始介绍 XMLHttpRequest 对象所有的属性和方法。

### 6.1.1 初始化请求

使用 XMLHttpRequest 的 open 方法来初始化一个请求。回顾 Hello World! 中 open 方法的使用，如下所示。

```
xmlhttp.open('GET','service/hello_world.php',true);
```

这里给 open 方法传递了 3 个参数：'GET'、'service/hello\_world.php' 和 true。它们的作用分别介绍如下。

- ☐ 'GET': 定义了请求的方法为 GET 方法。
- ☐ 'service/hello\_world.php': 定义了请求的目标地址。
- ☐ true: 定义了请求为异步请求。



```
oXMLHttpRequest.open(strMethod, strUrl, boolAsync, strUser, strPassword);
```

完整的参数如表 6-1 所示。

表 6-1 open 方法参数列表

参 数 名	说 明
strMethod	字符串型。HTTP请求的方法，例如POST、GET、PUT、HEAD、DELETE等，大小写不敏感
strUrl	字符串型。请求的URL地址，可以为绝对地址，也可以为相对地址
bool Async	布尔型。指定此请求是否为异步方式，默认为true。如果是异步方式，则当状态改变时会调用onreadystatechange属性指定的回调函数
strUser	字符串型。当服务器需要身份验证时，在此处指定用户名。可选参数
strPassword	字符串型。验证信息中的密码部分。如果用户名为空，则此值将被忽略。可选参数

6.1.2 设置请求的 HTTP 头信息

XMLHttpRequest 对象提供了 setRequestHeader 方法，可以用来设置请求的 HTTP 头信息：

```
oXMLHttpRequest.setRequestHeader(strHeader, strValue);
```

参数列表如表 6-2 所示。

表 6-2 setRequestHeader 方法参数列表

参 数 名	说 明
strHeader	字符串型。头名称
strValue	字符串型。值

常见用法介绍如下。

- ❑ 用 POST 方法提交请求时，设置编码类型：

```
oXMLHttpRequest.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
```

- ❑ 提交 COOKIE：

```
oXMLHttpRequest.setRequestHeader('COOKIE','cookieName=cookieValue');
```

- ❑ 提交 XML：

```
oXMLHttpRequest.setRequestHeader('Content-Type', 'text/xml');
```

备注：如果存在已经命名的 HTTP 头，则会被新的定义覆盖。此方法必须在 open 方法后调用。

现在修改 Hello World!的代码来测试 setRequestHeader 方法。程序首先通过 setRequestHeader 来发送一个 Cookie 值，然后在服务端获取后输出，再在客户端显示出来。将 Hello World!中 JavaScript 的内容进行适当修改，如下所示。



```
try
{
    var xmlhttp = new XMLHttpRequest();
}
catch(e)
{
    var xmlhttp = new ActiveXObject('Microsoft.XMLHTTP');
}
xmlhttp.open('GET','hello_world.php',true);
xmlhttp.setRequestHeader('COOKIE','author=Robin Chen');
xmlhttp.onreadystatechange = function()
{
    if(xmlhttp.readyState == 4 && xmlhttp.status == 200)
    {
        alert(xmlhttp.responseText);
    }
}
xmlhttp.send();
```

然后，修改 hello\_world.php 中的代码，如下所示。

```
<?
    echo $_COOKIE['author'];
?>
```

运行结果如图 6.1 所示。

对话框显示的正是程序所提交的 COOKIE: author 的内容。

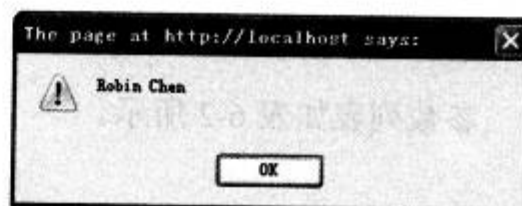


图 6.1 运行结果

### 6.1.3 发送请求

在 XMLHttpRequest 对象被初始化之后，可以调用 send 方法将请求发送到指定的 HTTP 服务器。

oXMLHttpRequest.send(varBody);

send 方法的参数说明如表 6-3 所示。

表 6-3 send 方法参数列表

参 数 名	说 明
varBody	通过请求发送的数据，variant 型，可以是字符串、DOM 树，或者其他任意数据流

如果请求是同步请求，此方法将会等待请求完成或者超时后才会返回，请求过程中页面程序将会中断执行，处于“假死”状态，请求返回后再继续执行。如果请求是异步请求，则立即返回，页面程序不会中断。

如果发送的数据为字符串型，则回应的数据被编码为 utf-8，可以按需要设置一个包含 charset 的文档类型头。如果发送的数据为 XML DOM Object，则回应的数据将被编码为在 XML 文档中声明的编码

类型。如果 XML 文档中并没有声明编码类型, 则使用默认的 utf-8。

当使用 GET 方法提交请求, 或者没有需要发送的数据时, 可以 `send(null)` 或直接省略参数 `send()`。

为了让读者明白同步请求和异步请求的差别, 看下面的例子。前台页面 `async_test.html` 的代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>同步、异步请求测试</title>
<style type="text/css">
input {margin-right:20px;}
</style>
<script type="text/javascript">
//异步请求
function asyncRequest(){
    try{
        var xmlhttp = new XMLHttpRequest();
    }catch(e){
        var xmlhttp = new ActiveXObject('Microsoft.XMLHTTP');
    }
    xmlhttp.open('GET','async_test.php',true);
    xmlhttp.onreadystatechange = function(){
        if(xmlhttp.readyState == 4 && xmlhttp.status == 200){
            alert(xmlhttp.responseText);
        }
    }
    xmlhttp.send();
}
//同步请求
function syncRequest(){
    try{
        var xmlhttp = new XMLHttpRequest();
    }catch(e){
        var xmlhttp = new ActiveXObject('Microsoft.XMLHTTP');
    }
    xmlhttp.open('GET','async_test.php',false);
    xmlhttp.send();
    alert(xmlhttp.responseText);
}
</script>
</head>

<body>
<input type="button" onclick="syncRequest();" value="发送同步请求" />
<input type="button" onclick="asyncRequest();" value="发送异步请求" />
</body>
</html>
```

为了更好地体现出同步请求和异步请求的差别，此例特别在服务端进行耗时的大运算量计算，来加大请求发出到返回的时间间隔。后台页面 `async_test.php` 的代码如下所示。

```
<?
    $i = 0;
    while($i < 10000000){
        $i ++;
    }
    echo $i;
?>
```

程序界面如图 6.2 所示。如图 6.3 所示，单击“发送同步请求”按钮后，浏览器进入“假死”状态，用户不能再进行任何操作，直到请求的结果返回才可以继续操作。而单击“发送异步请求”按钮，则不会影响用户其他的操作。返回结果，如图 6.4 所示。



图 6.2 同/异步请求例程界面



图 6.3 浏览器进入“假死”状态

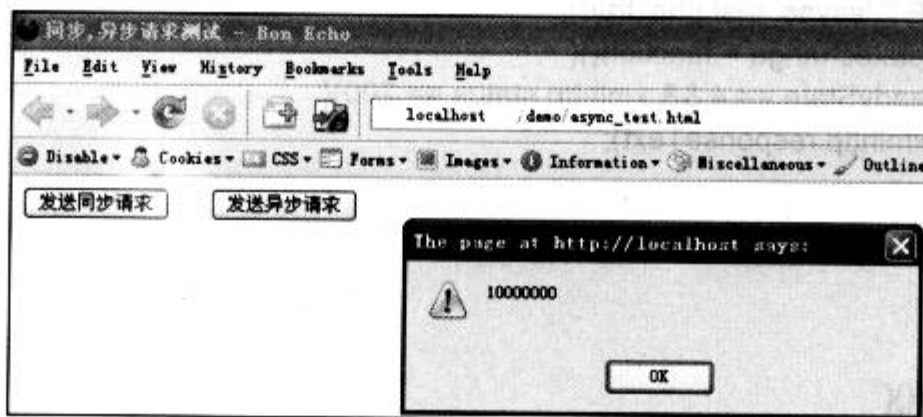


图 6.4 返回结果

显而易见，异步请求提供了更好的用户体验。

#### 6.1.4 获取请求的当前状态

一个 `XMLHttpRequest` 对象，在其生命周期中有 5 种状态，分别介绍如下。

- ❑ 0（未初始化）：对象已经创建，但是未调用 `open` 方法初始化。
- ❑ 1（初始化）：对象已经初始化，但未调用 `send` 方法。
- ❑ 2（发送数据）：`send` 方法已经被调用，但是 HTTP 状态和 HTTP 头未知。
- ❑ 3（数据传送中）：已经开始接收数据。但由于响应数据和 HTTP 头信息不全，这时尝试获取数据会出现错误。
- ❑ 4（完成）：数据接收完毕。

通过访问该对象的 `readyState` 的属性可以知道对象当前处于哪种状态。现在将 `HelloWorld.html` 内容进行修改，代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Hello World</title>
<script type="text/javascript">
window.onload = function()
{
    try
    {
        var xmlhttp = new XMLHttpRequest();
    }
    catch(e)
    {
        var xmlhttp = new ActiveXObject('Microsoft.XMLHTTP');
    }
    xmlhttp.open('GET','hello_world.php',true);
    xmlhttp.onreadystatechange = function()
    {
        document.body.innerHTML += '<div>readyState:' + xmlhttp.readyState + '</div>';
        if(xmlhttp.readyState == 4 && xmlhttp.status == 200)
        {
            document.body.innerHTML += '<div>responseText:' + xmlhttp.responseText + '</div>';
        }
    }
    xmlhttp.send();
}
</script>
</head>

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

通过 Web 服务器访问该页面，效果如图 6.5 所示。



图 6.5 readyState

### 6.1.5 指定请求状态改变时的事件处理句柄

当 XMLHttpRequest 对象的 readyState 发生变化时，会触发一个叫 readystatechange 的事件。通过将处理函数赋给 XMLHttpRequest 对象的 onreadystatechange 属性，可以为该事件指定事件处理函数。在 6.1.4 小节的示例中就是声明了一个匿名函数作为 readystatechange 的处理函数，并在该函数内完成了对 XMLHttpRequest 对象请求状态的判断和输出，代码如下所示。

```
xmlhttp.onreadystatechange = function()

    document.body.innerHTML += '<div>readyState:' + xmlhttp.readyState + '</div>';
    if(xmlhttp.readyState == 4 && xmlhttp.status == 200)
    {
        document.body.innerHTML += '<div>responseText:' + xmlhttp.responseText + '</div>';
    }
}
```

### 6.1.6 返回当前请求的 HTTP 状态码

在请求完成后，可以通过访问 XMLHttpRequest 对象的 status 属性来获取当前 HTTP 请求的状态，以此来判断请求是否成功。status 是一个只读的整型数值，其取值范围如表 6-4 所示。

表 6-4 HTTP 状态码

值	说 明
100	Continue
101	Switching Protocols
200	OK
201	Created
202	Accepted
203	Non-Authoritative Information
204	No Content
205	Reset Content
206	Partial Content
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect
400	Bad Request
401	Unauthorized



```

    }
    catch(e)
    {
        var xmlhttp = new ActiveXObject('Microsoft.XMLHTTP');
    }
    xmlhttp.open('GET','header.php',true);
    xmlhttp.onreadystatechange = function()
    {
        if(xmlhttp.readyState == 4 && xmlhttp.status == 200)
        {
            document.body.innerHTML += '<div>Header - Author:' + xmlhttp.getResponseHeader('Author')
+ '</div>';
            document.body.innerHTML += '<div>Header - WebSite:' +
xmlhttp.getResponseHeader('WebSite') + '</div>';
        }
    }
    xmlhttp.send();
}
</script>
</head>

<body>
</body>
</html>

```

后台页面 header.php 的代码如下所示。

```

<?
    header("Author:Robin Chen");
    header("WebSite:www.robchen.cn");
    echo "Hello World!";
?>

```

程序中，在后台的 header.php 中输出了两个自定义的头信息：Author 和 WebSite，然后在前台的 getResponseHeader.html 中获取并输出。程序运行结果如图 6.6 所示。

```

Header - Author:Robin Chen
Header - WebSite:www.robchen.cn

```

图 6.6 getResponseHeader

### 6.1.8 获取返回信息的所有 HTTP 头

使用 getResponseHeader 方法可以获取单个 HTTP 头信息。如果需要一次性获取所有的 HTTP 头信息，则可以使用 getAllResponseHeaders 方法。下面的示例演示了 getAllResponseHeaders 的用法，前台页面 getAllResponseHeaders.html 的代码如下所示。

```

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

```

```
<title>getAllResponseHeaders demo</title>
<script type="text/javascript">
window.onload = function()
{
    try
    {
        var xmlhttp = new XMLHttpRequest();
    }
    catch(e)
    {
        var xmlhttp = new ActiveXObject('Microsoft.XMLHTTP');
    }
    xmlhttp.open('GET','header.php',true);
    xmlhttp.onreadystatechange = function()
    {
        if(xmlhttp.readyState == 4 && xmlhttp.status == 200)
        {
            window.alert(xmlhttp.getAllResponseHeaders());
        }
    }
    xmlhttp.send();
}
</script>
</head>

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

后台页面 header.php 的代码如下所示。

```
<?
    header("Author:Robin Chen");
    header("WebSite:www.robchen.cn");
    echo "Hello World!";
?>
```

运行结果如图 6.7 所示。



图 6.7 getAllResponseHeaders

### 6.1.9 取得返回的数据

在 XMLHttpRequest 对象完成一次 HTTP 请求后，可以通过访问其 responseText 或 responseXML 属性来获取返回的数据。responseText 将返回的数据作为字符串格式返回，其使用方法在之前的很多示例中已经向读者展示过。responseXML 将返回的数据格式化为 XML 文档返回，它要求服务端输出的是一个有效的 XML 文档。下面的示例演示了 responseXML 的使用方法，前台页面 responseXML.html 的代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>responseXML demo</title>
<script type="text/javascript">
window.onload = function()
{
    try
    {
        var xmlhttp = new XMLHttpRequest();
    }
    catch(e)
    {
        var xmlhttp = new ActiveXObject('Microsoft.XMLHTTP');
    }
    xmlhttp.open('GET','message.xml',true);
    xmlhttp.onreadystatechange = function()
    {
        if(xmlhttp.readyState == 4 && xmlhttp.status == 200)
        {
            parseXMLData(xmlhttp.responseXML);
        }
    }
    xmlhttp.send();
}
function parseXMLData(data)
{
    var childs = data.childNodes;
    if(childs.length > 0)
    {
        document.body.innerHTML += '<div>' + data.nodeName + '</div>';
        for(var i = 0; i < childs.length; i++)
        {
            parseXMLData(childs[i]);
        }
    }
    else
    {
        document.body.innerHTML += '<div>' + data.nodeName + ':' + data.nodeValue + '</div>';
    }
}
</script>
</head>

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

后台页面 message.xml 的代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<list>
  <item>
    <name>Robin</name>
    <date>2007-11-5</date>
    <message>Hello World</message>
  </item>
  <item>
    <name>Lily</name>
    <date>2007-11-5</date>
    <message>Ajax</message>
  </item>
</list>
```

程序中通过 XMLHttpRequest 去请求 message.xml, 并将返回的数据格式化为 XML 文档对象, 并传递给 parseXMLData 解析后输出在页面中, 其运行结果如图 6.8 所示。

```
#document
list
#text:
item
#text:
name
#text:Robin
#text:
date
#text:2007-11-5
#text:
message
#text:Hello World
#text:
item
#text:
name
#text:Lily
#text:
date
#text:2007-11-5
#text:
message
#text:Ajax
#text:
#text:
```

图 6.8 responseXML

### 6.1.10 取消当前请求

如果需要取消一个正在进行中的请求, 可以调用 XMLHttpRequest 对象的 abort 方法, 该调用的语法如下所示。

```
oXMLHttpRequest.abort();
```

调用 abort 方法后, XMLHttpRequest 对象重新回到未初始化状态。

## 6.2 搭建基本的 Ajax 开发框架

介绍完 XMLHttpRequest 对象的作用和各个成员后, 接着向读者介绍如何基于 XMLHttpRequest 搭建一个基本的 Ajax 应用程序开发框架。这个基本的 Ajax 应用程序开发框架应该包含 XMLHttpRequest 对象的创建、提供回调功能以增强对业务逻辑的支持以及兼容性相关的处理。

### 6.2.1 创建 XMLHttpRequest 对象

由于在 IE 核心的浏览器中, IE6 及以下版本对 XMLHttpRequest 对象的支持是通过 ActiveXObject 提供的。而在 IE7 和其他非 IE 核心的浏览器中, XMLHttpRequest 对象是作为 window 对象的属性而原生支持的。当同样是通过 ActiveXObject 提供 XMLHttpRequest 的支持时, 也会存在版本差异。这样在创建 XMLHttpRequest 对象时, 就会遇到兼容性问题。可能一个程序在某个浏览器中能够正常地运行, 但是换到另外一种浏览器中, 或者版本低一些的浏览器中, 就无法运行了。于是, 这个框架首先需要

对 XMLHttpRequest 对象的创建进行封装, 以此来解决兼容性问题。一个封装好的创建 XMLHttpRequest 对象的函数如下所示。

```
function getTransport()
{
    var versions = [
        function()
        {
            return new XMLHttpRequest();
        },
        function()
        {
            new ActiveXObject('Microsoft.XMLHTTP');
        },
        function()
        {
            new ActiveXObject('Microsoft.XMLHTTP');
        }
    ];
    var request;
    for(var i = 0; i < versions.length; i++)
    {
        var lambda = versions[i];
        try
        {
            request = lambda();
            break;
        }
        catch(e){}
    }
    return request;
}
```

函数中, 首先用一个数组 versions 保存了 3 个匿名函数。这些匿名函数提供了各个不同版本的 XMLHttpRequest 的创建方法。然后用一个 for 循环语句配合 try...catch 语句来获取执行这些匿名函数, 并获得第一个成功执行的函数的返回值, 然后跳出循环。最后返回 XMLHttpRequest 对象。如果所有的匿名函数都没有被成功执行, 则会返回 undefined。

### 6.2.2 发送请求和回调函数

在对 XMLHttpRequest 对象的创建进行封装后, 还需要对请求的发送和回调函数进行处理, 以此来提供对业务逻辑的支持。现在, 增加一个名为 ajaxRequest 的函数来完成这些功能。

```
/*
 * 函数名: ajaxRequest
 * 功能: 根据用户指定的 URL、方法、参数、HTTP 头, 以及回调函数, 自动创建 XMLHttpRequest 对象并发送请求。
```



```

* 参数介绍:
*      url: 请求的 URL 地址, 字符串型
*      options: 参数集合, 对象类型, 其成员均为可选, 成员为:
*          method: 发送请求的方法, 可以为 GET 或 POST, 字符串型
*          parameters: 需要发送的数据, 字符串型, 其形式为"a=1&b=2&c=3"
*          headers: 需要发送的 HTTP 头信息, 对象类型。其每一个属性为一个头信息。
属性名为头信息的名称, 属性值为头信息的内容
*      onLoading: 在请求开始时执行的函数
*      onComplete: 在请求完成时执行的函数
*      onSuccess: 在请求成功时执行的函数
*      onFailure: 在请求失败时执行的函数
*/
function ajaxRequest(url,options)
{
    var request = getTransport();                                //创建 XMLHttpRequest 对象
    if(typeof request == 'undefined')                            //如果 request 为 undefined, 则
        抛出异常, 说明当前浏览器不支持 XMLHttpRequest,并退出函数
    {
        throw new Error('Your browser does not support XMLHttpRequest');
        return;
    }
    var url = url;
    var method = (options.method || 'POST').toUpperCase();      //获取提交方式, 默认为 POST
    if(method != 'GET' && method != 'POST')
    {
        method = 'POST';
    }
    var parameters = options.parameters || null;                //需要提交的参数, 默认为 null
    var headers = options.headers || {};                        //需要发送的 HTTP 头信息, 是一个对象, 其成员包含了头信息的名称和值的信息
    var onLoadingEventHandler = options.onLoading || function(); //在请求开始时执行的函数, 由用户指定, 默认为一个空函数
    var onCompleteEventHandler = options.onComplete || function(); //在请求完成时执行的函数, 由用户指定, 默认为一个空函数
    var onSuccessEventHandler = options.onSuccess || function(); //在请求成功时执行的函数, 由用户指定, 默认为一个空函数
    var onFailureEventHandler = options.onFailure || function(); //在请求失败时执行的函数, 由用户指定, 默认为一个空函数
    if(method == 'GET' && parameters != null)                    //如果提交方式被指定为 GET,
        则将 parameters 的内容拼接到 URL 中, 并将 parameters 设置为 null
    {
        if(url.indexOf('?') > -1)
        {
            url += '&' + parameters;
        }
        else
        {
            url += '?' + parameters;
        }
    }
}

```

```

        parameters = null;
    }
    request.open(method,url,true); //初始化 XMLHttpRequest 对象
    request.setRequestHeader('contentType','application/x-www-form-urlencoded');
    for(var name in headers) //设置由用户指定的 HTTP 头信息
    {
        request.setRequestHeader(name,headers[name]);
    }
    request.onreadystatechange = function() //注册 XMLHttpRequest 对象的
readystatechange 事件处理函数
    {
        if(request.readyState == 1) //当 readyState 等于 1 时,表示请求开
始,将当前 XMLHttpRequest 对象作为其参数调用 onLoadingEventHandler 函数
        {
            onLoadingEventHandler(request);
        }
        if(request.readyState == 4) //当 readyState 等于 4 时,表示请求完
成,将当前 XMLHttpRequest 对象作为其参数调用 onCompleteEventHandler 函数
        {
            onCompleteEventHandler(request);
            if(request.status && request.status >= 200 && request.status < 300)
            {
                onSuccessEventHandler(request); //当 HTTP 状态码大于等于 200 小于
300 时,表示请求成功,这时将当前 XMLHttpRequest 对象作为其参数调用 onSuccessEventHandler 函数
            }
            else
            {
                onFailureEventHandler(request); //否则表示请求失败,这时将当前
XMLHttpRequest 对象作为其参数调用 onFailureEventHandler 函数
            }
        }
    }
    request.send(parameters); //发送请求
}

```

关于函数及其组成代码的功能说明,参照代码中的注释。

### 6.2.3 一个封装好的基本 Ajax 应用程序开发框架

现在将两个函数的代码合并到一个 JavaScript 文件中,并将其命名为 ajaxRequest.js。这样,在以后需要用到 Ajax 功能的页面中,只需要通过<script>标签引入这个文件即可。完整的 ajaxRequest.js 文件代码如下所示。

```

/*
 * 函数名: getTransport
 * 功能: 根据不同的浏览器采用不同的方法自动创建 XMLHttpRequest 对象并返回,如果浏览器不支持
XMLHttpRequest,则返回 undefined
 */

```

```

function getTransport()
{
    var versions = [
        function()
        {
            return new XMLHttpRequest();
        },
        function()
        {
            new ActiveXObject('Microsoft.XMLHTTP');
        },
        function()
        {
            new ActiveXObject('Microsoft.XMLHTTP');
        }
    ];
    var request;
    for(var i = 0; i < versions.length; i++)
    {
        var lambda = versions[i];
        try
        {
            request = lambda();
            break;
        }
        catch(e){}
    }
    return request;
}

/*
* 函数名: ajaxRequest
* 功能: 根据用户指定的 URL、方法、参数、HTTP 头, 以及回调函数, 自动创建 XMLHttpRequest 对象并发送请求。
* 参数介绍:
*     url: 请求的 URL 地址, 字符串型
*     options: 参数集合, 对象类型, 其成员均为可选, 成员为:
*         method: 发送请求的方法, 可以为 GET 或 POST, 字符串型
*         parameters: 需要发送的数据, 字符串型, 其形式为 "a=1&b=2&c=3"
*         headers: 需要发送的 HTTP 头信息, 对象类型。其每一个属性为一个头信息。
* 属性名为头信息的名称, 属性值为头信息的内容
*     onLoading: 在请求开始时执行的函数
*     onComplete: 在请求完成时执行的函数
*     onSuccess: 在请求成功时执行的函数
*     onFailure: 在请求失败时指定的函数
*/
function ajaxRequest(url,options)
{
    var request = getTransport();

```

//创建 XMLHttpRequest 对象

```

    if(typeof request == 'undefined')                //如果 request 为 undefined, 则抛出异常, 说明当前浏览器不支持 XMLHttpRequest, 并退出函数
    {
        throw new Error("Your browser does not support XMLHttpRequest");
        return;
    }
    var url = url;
    var method = (options.method || 'POST').toUpperCase();    //获取提交方式, 默认为 POST
    if(method != 'GET' && method != 'POST')
    {
        method = 'POST';
    }
    var parameters = options.parameters || null;                //需要提交的参数, 默认为 null
    var headers = options.headers || {};                        //需要发送的 HTTP 头信息, 是一个对象, 其成员包含了头信息的名称和值的信息
    var onLoadingEventHandler = options.onLoading || function();    //在请求开始时执行的函数, 由用户指定, 默认为一个空函数
    var onCompleteEventHandler = options.onComplete || function();    //在请求完成时执行的函数, 由用户指定, 默认为一个空函数
    var onSuccessEventHandler = options.onSuccess || function();    //在请求成功时执行的函数, 由用户指定, 默认为一个空函数
    var onFailureEventHandler = options.onFailure || function();    //在请求失败时执行的函数, 由用户指定, 默认为一个空函数
    if(method == 'GET' && parameters != null)                //如果提交方式被指定为 GET, 则将 parameters 的内容拼接到 URL 中, 并将 parameters 设置为 null
    {
        if(url.indexOf('?') > -1)
        {
            url += '&' + parameters;
        }
        else
        {
            url += '?' + parameters;
        }
        parameters = null;
    }
    request.open(method, url, true);                //初始化 XMLHttpRequest 对象
    request.setRequestHeader('contentType', 'application/x-www-form-urlencoded');
    for(var name in headers)                //设置由用户指定的 HTTP 头信息
    {
        request.setRequestHeader(name, headers[name]);
    }
    request.onreadystatechange = function()                //注册 XMLHttpRequest 对象的 readystatechange 事件处理函数
    {
        if(request.readyState == 1)                //当 readyState 等于 1 时, 表示请求开始, 将当前 XMLHttpRequest 对象作为其参数调用 onLoadingEventHandler 函数
        {
            onLoadingEventHandler(request);
        }
    }

```

```

    }
    if(request.readyState == 4) //当 readyState 等于 4 时, 表示请求完
成, 将当前 XMLHttpRequest 对象作为其参数调用 onCompleteEventHandler 函数
    {
        onCompleteEventHandler(request);
        if(request.status && request.status >= 200 && request.status < 300)
        {
            onSuccessEventHandler(request); //当 HTTP 状态码大于等于 200 小于
300 时, 表示请求成功, 这时将当前 XMLHttpRequest 对象作为其参数调用 onSuccessEventHandler 函数
        }
        else
        {
            onFailureEventHandler(request); //否则表示请求失败, 这时将当前
XMLHttpRequest 对象作为其参数调用 onFailureEventHandler 函数
        }
    }
}
request.send(parameters); //发送请求
}

```

现在来编写一个例程以测试 ajaxRequest.js 文件的功能。首先编写前台页面 demo.html, 在这个页面中, 通过<script>标签引入 ajaxRequest.js 文件, 并对相关参数进行定义, 然后在页面加载完成后, 调用 ajaxRequest 函数发送一个请求, 其代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>demo</title>
<script type="text/javascript" src="ajaxRequest.js"></script>
<script type="text/javascript">
var options = {
    method: 'GET',
    parameters: 'Name=Robin',
    headers:{
        Sex: 'male',
        RequestBy: 'ajaxRequest'
    },
    onLoading: function()
    {
        document.body.innerHTML += '<div>Now loading...</div>';
    },
    onComplete: function()
    {
        document.body.innerHTML += '<div>Loading complete.</div>';
    },
    onSuccess: function()
    {

```



```
document.body.innerHTML += '<div>Request success.</div>';
},
onFailure: function()
{
    document.body.innerHTML += '<div>Request failure.</div>';
}
}
window.onload = function()
{
    ajaxRequest('demo.php',options);
}
</script>
</head>
```

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

然后编写后台被请求的页面 demo.php。这个页面非常简单，其代码如下所示。

```
<?
    echo 'ok';
?>
```

然后将两个页面放到 Web 服务器的虚拟目录中，并通过浏览器访问 demo.html，其运行结果如图 6.9 所示。

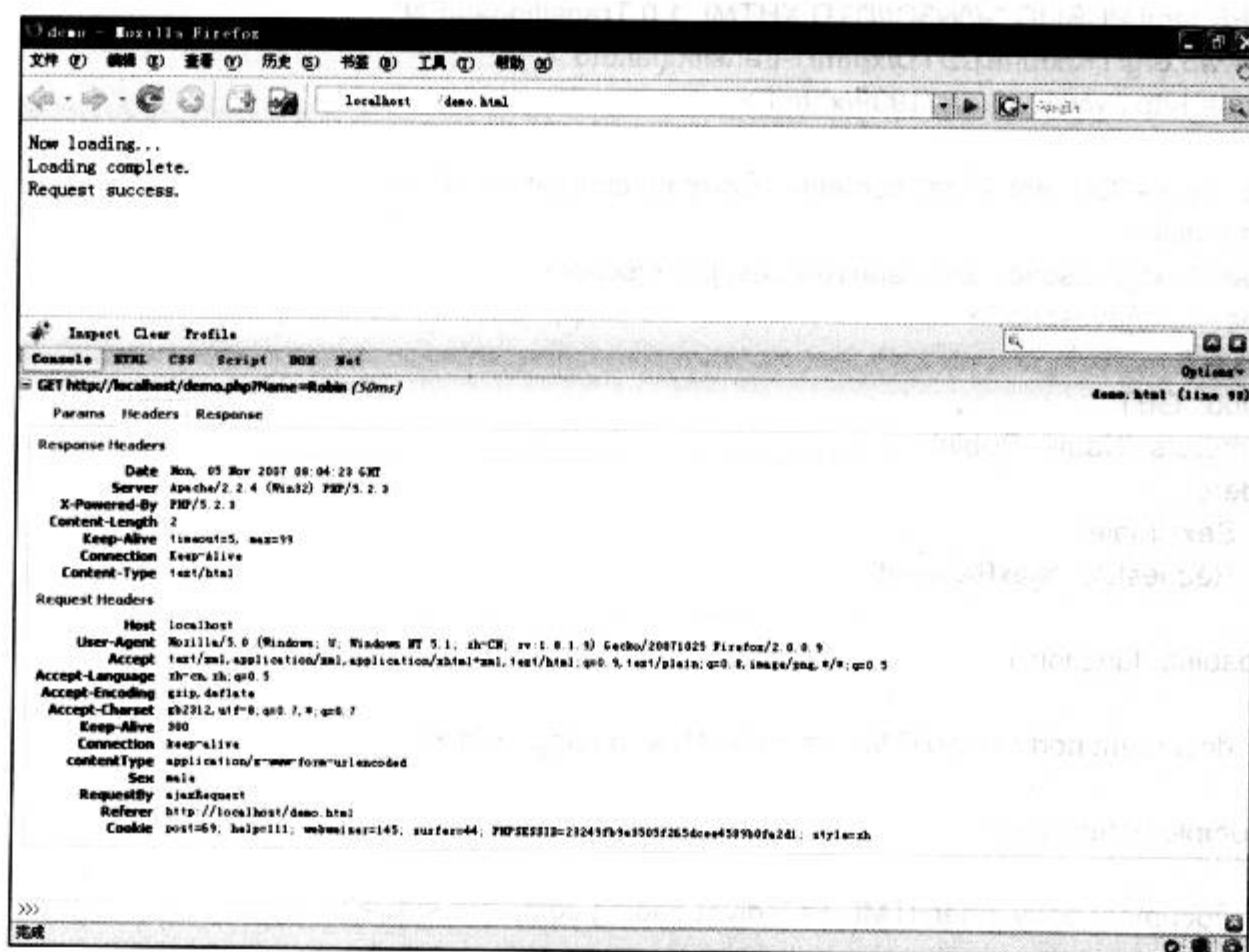


图 6.9 运行结果

通过 Firebug 的控制台可以看到, 通过 options 定义的参数 “Name=Robin” 被连接在 URL 的后面, 用 GET 方式提交了请求。同时定义的 HTTP 头信息 “Sex: male” 和 “RequestBy: ajaxRequest” 都被发送了出去。而定义的回调函数 onLoading、onComplete 和 onSuccess, 都在相应的时候被调用执行了, 所以页面上出现 3 条记录。而由于请求是成功的, 所以 onFailure 回调函数并没有被执行。ajaxRequest.js 成功地完成了作为 Ajax 基本开发框架的任务。

## 6.3 小 结

本章通过对第 1 章中的简单 Ajax 应用程序: Hello World! 的各组成部分的分析, 向读者详细讲解了 XMLHttpRequest 对象的知识。其中包括 XMLHttpRequest 对象的属性和方法, 以及同步、异步的概念。然后针对处理浏览器兼容性需要, 以及出于对业务逻辑支持的考虑, 对 XMLHttpRequest 对象的创建和请求的发送以及其状态的处理进行了封装, 来建立一个基本的 Ajax 应用程序开发框架, 并通过实例演示了其用法, 测试了其功能。

通过 `findup` 的返回值可以看到，通过 options 定义的参数 “Name=Robin” 被连接在 URL 的后面，用 GET 方式提交了请求。同时返回的 HTTP 头信息 “Sex: male” 和 “RedeemBy: ajaxRedeem” 都被发走了。而返回的响应数据 `onComplete` 和 `onSuccess`，则在相应的时候被调用执行了。所以页面上出现 3 条信息。由于请求是成功的，所以 `onFailure` 回调函数并没有被执行。 `ajaxRedeem` 是 jQuery 的默认方法，用于提交基本的 Ajax 请求，并返回一个 `jqXHR` 对象，该对象是 `XMLHttpRequest` 的包装器。该对象是 `XMLHttpRequest` 的包装器，并返回一个 `jqXHR` 对象。该对象是 `XMLHttpRequest` 的包装器，并返回一个 `jqXHR` 对象。

第 3 小節

本章节主要介绍：本章中的简单 Ajax 应用原理；Hello World 所对应的各个部分，向读者展示如何开发 Ajax 应用。本章中，我们将介绍 Ajax 应用的基本原理，以及如何在 Ajax 应用中实现 Ajax 应用。本章中，我们将介绍 Ajax 应用的基本原理，以及如何在 Ajax 应用中实现 Ajax 应用。本章中，我们将介绍 Ajax 应用的基本原理，以及如何在 Ajax 应用中实现 Ajax 应用。

# 第 7 章

## 完善的 Ajax 应用程序：Ajax 留言本

- » 留言本的需求
- » 留言本的基本设计
- » 留言本的实现
- » 留言本的功能测试
- » 小结

第 6 章向读者详细介绍了构建 Ajax 应用程序的核心：XMLHttpRequest 对象的相关属性和方法，并介绍了如何搭建一个基本的 Ajax 应用程序开发框架。在第 1 章中已经向读者展示了一个简单的 Ajax 程序：Hello World!。在本章中，笔者将和读者一起利用这个 Ajax 开发框架来搭建一个更加完善的 Ajax 应用程序：Ajax 留言本。



## 7.1 留言本的需求

在开始开发留言本前，需要先确定留言本的需求。留言本与传统的留言本需求基本一致，所不同的只是采用 Ajax 交互方式来代替传统交互方式。

在访客打开留言本时，显示所有的历史留言记录。留言本提供表单让访客可以添加新的留言。由于采用 Ajax 技术，前台页面不需要是一个动态页面，只使用一个静态的 HTML 页面即可。历史留言信息不需要预先加载，而是在访客打开页面后再去读取并无刷新地显示在页面上。访客添加新留言时，采用 Ajax 提交到后台的程序处理，并在提交成功后无刷新地将新留言显示在页面上的留言列表中。留言本的用例图如图 7.1 所示。

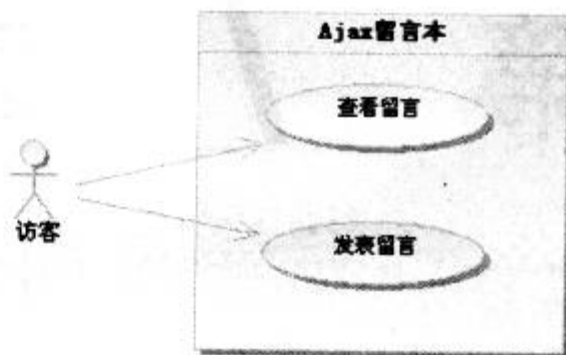


图 7.1 用例图

## 7.2 留言本的基本设计

在确认留言本的需求后，现在来做一些基本的设计，包括留言本的系统环境、基本数据模型和操作、数据库设计、前后台功能模块的划分以及文件结构和文件清单。

### 7.2.1 系统环境

留言本系统在如下环境下进行搭建。

- ☐ Web 服务器: Apache2.2.4。
- ☐ 服务端语言: PHP5.2.3。
- ☐ 数据库: Mysql5.0。

### 7.2.2 留言的数据和操作

首先，需要确认留言所包含的数据和针对需要对留言进行的操作。一条常规留言的数据应该包含留言人的名字 (name)、留言内容 (message) 和留言时间 (postdate)，系统可以读取所有留言以及插入一条新的留言。其 UML 表示如图 7.2 所示。

name、message 和 postdate 都是 Message 的属性。而 Message 还包含两个方法：静态方法 readAll 和实例方法 insert。readAll 方法提供了读取所有留言的操作。insert 方法提供了添加新留言的操作。

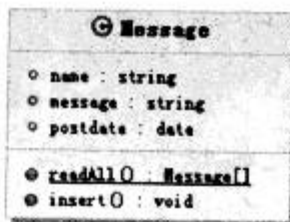


图 7.2 Message



注意: 对属性、静态方法、实例方法等 OOP (面向对象编程) 概念不熟悉的读者, 可以参阅本书的第 12 章中对 OOP 的介绍。

### 7.2.3 数据库设计

在确定 Message 所包含的数据后, 现在来对系统的数据库进行设计。数据库命名为 guestbook, 其中包含一个表 message, 设计如图 7.3 所示。

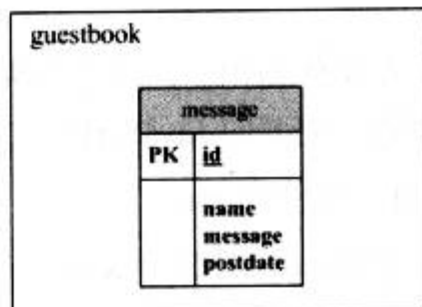


图 7.3 数据库设计

### 7.2.4 后台功能模块

后台的功能模块需要包含以下几个部分:

- ☐ 数据库操作模块。
- ☐ 留言板逻辑处理模块。
- ☐ 接口模块, 提供给前台脚本调用并输出信息。

各模块之间的依赖关系如图 7.4 所示。

### 7.2.5 前台脚本功能模块

然后需要确定一下前台脚本所需要具备的功能。按照功能的需要, 前台脚本应该包含以下主要功能模块:

- ☐ 基本 Ajax 功能模块。
- ☐ 读取留言功能模块。
- ☐ 发送留言功能模块。
- ☐ 显示留言基本模块。
- ☐ 留言数据特殊字符过滤模块。

各模块之间的依赖关系如图 7.5 所示。



图 7.4 后台功能模块

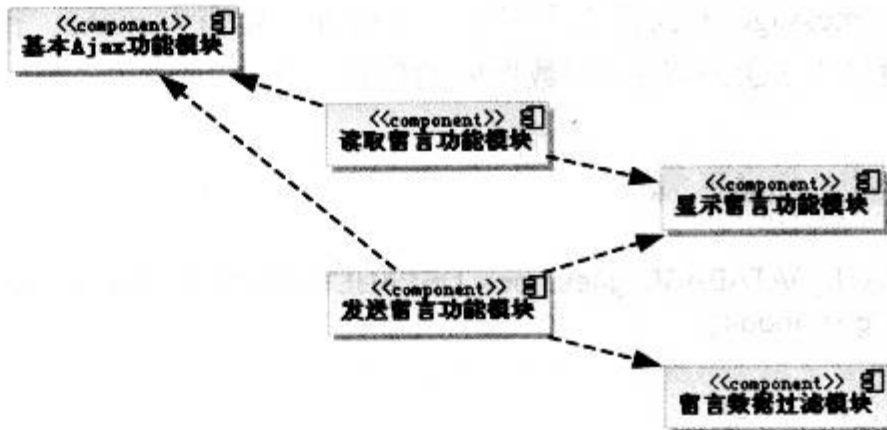


图 7.5 前台脚本功能模块

## 7.2.6 系统文件结构和文件清单

在文件结构上，用文件夹 ajaxGuestBook 作为根目录，其下包含 script、php、css 3 个子目录，分别放置所需要的 JavaScript 文件、PHP 文件和 CSS 样式表文件。根目录下详细的文件清单如下。

### 1. 文件夹 css

style.css: 样式表文件。

### 2. 文件夹 script

□ ajaxRequest.js: 基本 Ajax 功能模块。

□ guestbook.js: 留言本相关功能模块。

### 3. 文件夹 php

□ db.class.php: 数据库操作模块。

□ message.class.php: 留言逻辑处理模块。

□ service.php: 接口模块。

### 4. index.html: 留言本前台页面

## 7.3 留言本的实现

在做完留言本的系统设计后，现在开始进行具体的代码实现。首先根据数据库设计来创建数据库，然后对留言本的前台界面进行设计和制作，接着完成后台和前台的各个功能模块，最后再将这些模块和页面整合在一起。

### 7.3.1 创建数据库

根据 7.2 节中的数据库设计，需要创建一个名为 guestbook 的数据库。这个数据库只有一个表 message。message 表包含 4 个字段：主键 id、留言人 name、留言内容 message 和留言时间 postdate。可以使用以下 SQL 代码进行数据库的创建工作。

```
--
-- 数据库: 'guestbook'
--
CREATE DATABASE 'guestbook' DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
USE 'guestbook';
-----
--
-- 表的结构 'message'
--
CREATE TABLE 'message' (
  'id' int(11) NOT NULL auto_increment,
```

```
'name' varchar(50) NOT NULL,
'message' text NOT NULL,
'postdate' date NOT NULL,
PRIMARY KEY ('id')
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;
```

### 7.3.2 完成前台界面: index.html

创建好数据库后,现在来搭建留言本的前台界面。前台界面由页面 index.html 和样式表文件 style.css 组成。其中, index.html 的代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Ajax Guest Book</title>
<link type="text/css" rel="stylesheet" href="css/style.css" />
</head>

<body>
<h1>Guest Book<span id="loadingMsg" style="display:none;" class="msgBox">(loading...)</span></h1>
<ul id="msgList">
</ul>
<form name="fmMsg" id="fmMsg" action="?" method="post">
<h2>Message<span id="submitMsg" style="display:none;" class="msgBox">(loading...)</span></h2>
<label for="txtName">name</label>
<input name="txtName" type="text" id="txtName" value="guest" onfocus="this.select();" />
<label for="txtContent">Message</label>
<textarea name="txtContent" rows="4" id="txtContent" onfocus="this.select();">something to
say...</textarea>
<input type="button" value="Click here to submit your message!" id="btnSubmit" disabled="disabled" />
</form>
</body>
</html>
```

### 7.3.3 完成前台界面: 样式表

样式表文件 style.css 的代码如下所示。

```
@charset "utf-8";
/* CSS Document */
*{
    margin:0;
    padding:0;
}
html {
```

```
        background-color:#eee;
        height:100%;
    }
    body {
        padding:15px;
        font-size:11px;
        width:500px;
        background-color:#fff;
        height:100%;
        font-family:Tahoma;
        border-left:20px solid #ccc;
    }
    ul {
        list-style:none;
        border-top:1px solid #999;
        height:350px;
        overflow-x:auto;
        overflow-y:scroll;
    }
    span {
        font-weight:bold;
        font-size:12px;
    }
    span.date {
        margin-left:10px;
        font-weight:normal;
        font-size:11px;
    }
    li {
        border-bottom:1px dashed #666;
        line-height:20px;
    }
    form {
        margin-top:10px;
        border-top:1px solid #999;
    }
    label {
        display:block;
        line-height:20px;
        font-weight:bold;
        cursor:pointer;
        background-color:#999;
        color:#fff;
        margin:3px 0;
        padding-left:5px;
        width:100%;
    }
    #txtName , #txtContent {
        width:100%;
```

```

        font-size:11px;
    }
    #btnSubmit {
        display:block;
        margin-top:3px;
        border:1px solid #666;
        padding:2px 5px;
        width:100%;
    }
    .msgBox {
        font-size:11px;
    }
}

```

留言本的界面效果如图 7.6 所示。

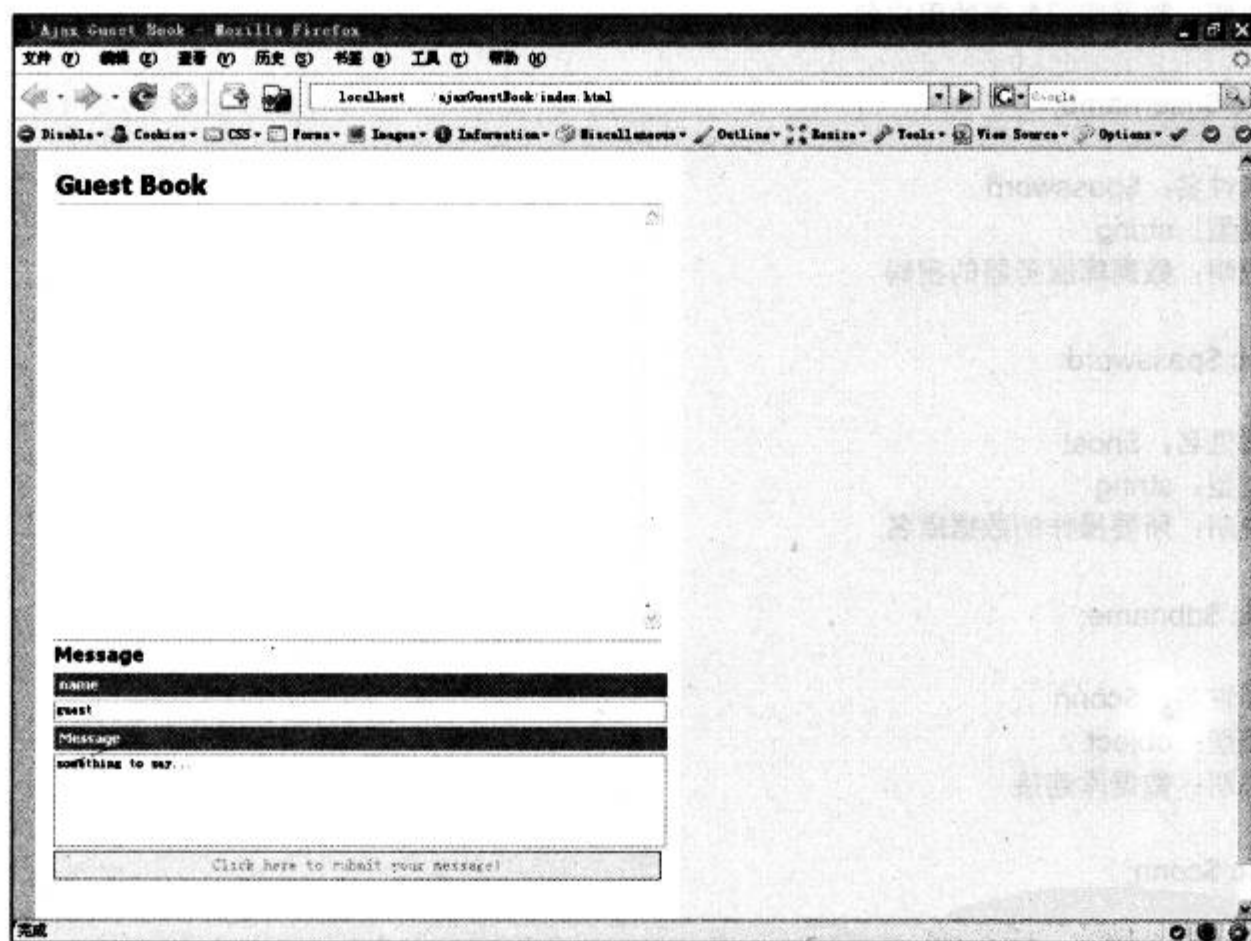


图 7.6 留言本界面效果

### 7.3.4 完成后台功能模块: 数据库操作模块

搭建好前台界面后, 现在开始完成后台的各个功能模块。

数据库操作模块提供了数据库相关操作的封装, 包括数据库连接的打开和关闭, 读取数据和插入数据的功能。将这个功能模块的代码单独放到一个 PHP 文件中, 并命名为 `db.class.php`, 其代码如下所示。

```

<?
/*

```



```
* 类名: DB
* 说明: 数据库操作类
*/
class DB
{
    /*
    * 属性名: $host
    * 类型: string
    * 说明: 数据库服务器的地址
    */
    public $host;
    /*
    * 属性名: $username
    * 类型: string
    * 说明: 数据库服务器的用户名
    */
    public $username;
    /*
    * 属性名: $password
    * 类型: string
    * 说明: 数据库服务器的密码
    */
    public $password;
    /*
    * 属性名: $db
    * 类型: string
    * 说明: 所要操作的数据库名
    */
    public $dbname;
    /*
    * 属性名: $conn
    * 类型: object
    * 说明: 数据库连接
    */
    public $conn;
    /*
    * 方法名: DB()
    * 说明: 构造函数
    * 参数列表:
    *     参数名: $host
    *         说明: 数据库服务器地址
    *         类型: string
    *         默认值: 'localhost'
    *     参数名: $username
    *         说明: 数据库服务器的用户名
    *         类型: string
    *         默认值: 'root'
    *     参数名: $password
    *         说明: 数据库服务器的密码
    */
}
```

```

*          类型: string
*          默认值: '123'
*      参数名: $dbname
*          说明: 需要操作的数据库
*          类型: string
*          默认值: 'guestbook'
* 返回值: 无
*/
public function DB($host='localhost',$username='root',$password='123',$dbname='guestbook')
{
    $this->host = $host;
    $this->username = $username;
    $this->password = $password;
    $this->dbname = $dbname;
}
/*
* 方法名: open()
* 说明: 打开一个数据库连接
* 参数列表: 无
* 返回值: 无
*/
private function open()
{
    $this->conn = mysql_connect($this->host,$this->username,$this->password);
    mysql_select_db($this->dbname);
}
/*
* 方法名: close()
* 说明: 关闭一个数据库连接
* 参数列表: 无
* 返回值: 无
*/
private function close()
{
    mysql_close($this->conn);
}
/*
* 方法名: getObjListBySql()
* 说明: 根据查询语句返回一个对象数组
* 参数列表:
*      参数名: $sql
*      类型: string
*      说明: 需要执行的查询语句
*      默认值: 无
* 返回值: 对象数组
*/
public function getObjListBySql($sql)
{
    $this->open();

```

```

        $rs = mysql_query($sql,$this->conn);
        $objList = array();
        while($obj = mysql_fetch_object($rs))
        {
            if($obj)
            {
                $objList[] = $obj;
            }
        }
        $this->close();
        return $objList;
    }
    /*
    * 方法名: insertData()
    * 说明: 插入一条新记录
    * 参数列表:
    *      参数名: $table
    *      类型: string
    *      说明: 目标表
    *      默认值: 无
    *      参数名: $columns
    *      类型: 数组
    *      说明: 需要插入的字段集合
    *      默认值: 空数组
    *      参数名: $values
    *      类型: 数组
    *      说明: 需要插入的字段值集合
    *      默认值: 空数组
    * 返回值: 对象数组
    */
    public function insertData($table,$columns=array(),$values=array())
    {
        $sql = 'insert into '.$table.'(';
        for($i = 0; $i < sizeof($columns);$i++)
        {
            $sql .= $columns[$i];
            if($i < sizeof($columns) - 1)
            {
                $sql .= ',';
            }
        }
        $sql .= ') values (';
        for($i = 0; $i < sizeof($values);$i++)
        {
            $sql .= "".$values[$i]."";
            if($i < sizeof($values) - 1)
            {
                $sql .= ',';
            }
        }
    }

```

```

    }
    $sql .= ' ';
    $this->open();
    mysql_query($sql,$this->conn);
    $id = mysql_insert_id($this->conn);
    $this->close();
    return $id;
}
}
?>

```

程序中将留言本程序所使用的数据库服务器的地址、用户名、密码和数据库名设置为 DB 类的构造函数的各个参数的默认值,这样可以简化一些编码的工作。

### 7.3.5 完成后台功能模块: 留言本逻辑处理模块

留言本逻辑处理模块主要提供留言对象的创建、留言的读取以及插入新留言的功能。这个模块依赖于数据库操作模块实现。将这个模块的代码放入一个单独的 PHP 文件,并命名为 message.class.php,其代码如下所示。

```

<?
require_once('db.class.php');    //引入 db.class.php
/*
 * 类名: Message
 * 说明: 留言类
 */
class Message
{
    /*
     * 属性名: $name
     * 类型: string
     * 说明: 留言人的姓名
     */
    public $name;
    /*
     * 属性名: $message
     * 类型: string
     * 说明: 留言的内容
     */
    public $message;
    /*
     * 属性名: $postdate
     * 类型: string
     * 说明: 留言的时间
     */
    public $postdate;
    /*
     * 方法名: Message()

```

```
* 说明: 构造函数
* 参数列表:
*     参数名: $name
*     说明: 留言人的姓名
*     类型: string
*     默认值: 无
*     参数名: $message
*     说明: 留言的内容
*     类型: string
*     默认值: 无
*     参数名: $postdate
*     说明: 留言的时间
*     类型: datetime
*     默认值: 无
* 返回值: 无
*/
public function Message($name,$message,$postdate)
{
    $this->name = $name;
    $this->message = $message;
    $this->postdate = $postdate;
}
/*
* 方法名: readAll()
* 说明: 读取所有的留言信息
* 参数列表: 无
* 返回值: 包含留言数据的数组
*/
public static function readAll()
{
    $db = new DB();
    $messages = $db->getObjListBySql('select * from message');
    return $messages;
}
/*
* 方法名: insert()
* 说明: 插入新留言
* 参数列表: 无
* 返回值: 无
*/
public function insert()
{
    $db = new DB();

    $db->insertData('message',array('name','message','postdate'),array($this->name,$this->message,$this->postdate));
}
}
?>
```



### 7.3.6 完成后台功能模块: 接口模块

接口模块相应前台脚本程序的请求, 并按照请求进行读取留言或插入新留言的操作, 其依赖于留言本逻辑处理模块。将其代码单独放置在一个 PHP 文件中, 并命名为 service.php, 其代码如下所示。

```
<?
require_once('message.class.php');           //引入 message.class.php
$action = $_REQUEST['action'];               //获取提交的 action 值
if($action == 'getall')                      //如果 action 值为 getall, 则
{
    $messages = Message::readAll();          //调用留言本逻辑处理模块来读取所有留言信息
    echo json_encode($messages);             //输出留言信息的 JSON 格式表达
}
else if($action == 'addnew')                 //如果 action 值为 addnew, 则
{
    $message = new Message($_REQUEST['name'],$_REQUEST['message'],date('Y-m-d'));
    $message->insert();                       //创建一个 Message 实例, 并插入到数据库
    echo 'ok';                               //输出 ok
}
else
{
    echo 'no action';                        //输出 no action
}
?>
```

### 7.3.7 完成前台功能模块: 基本 Ajax 功能模块

后台的各个功能模块完成后, 开始完成前台的功能模块。

基本 Ajax 功能模块采用第 6 章向读者介绍的 Ajax 基本开发框架来完成。将框架的代码单独放入一个 JavaScript 文件, 并命名为 ajaxRequest.js, 其代码如下所示。

```
/*
 * 函数名: getTransport
 * 功能: 根据不同的浏览器采用不同的方法自动创建 XMLHttpRequest 对象并返回, 如果浏览器不支持
XMLHttpRequest, 则返回 undefined
 */
function getTransport()
{
    var versions = [
        function()
        {
            return new XMLHttpRequest();
        },
        function()
        {
            return new ActiveXObject('Microsoft.XMLHTTP');
        }
    ];
    for (var i = 0; i < versions.length; i++)
    {
        if (versions[i]() != null)
        {
            return versions[i]();
        }
    }
    return undefined;
}
```

```

    }
    var parameters = options.parameters || null;           //需要提交的参数, 默认为 null
    var headers = options.headers || {};                  //需要发送的 HTTP 头信息, 是一个对象, 其成员包含了头信息的名称和值的信息
    var onLoadingEventHandler = options.onLoading || function(); //在请求开始时执行的函数, 由用户指定, 默认为一个空函数
    var onCompleteEventHandler = options.onComplete || function(); //在请求完成时执行的函数, 由用户指定, 默认为一个空函数
    var onSuccessEventHandler = options.onSuccess || function(); //在请求成功时执行的函数, 由用户指定, 默认为一个空函数
    var onFailureEventHandler = options.onFailure || function(); //在请求失败时执行的函数, 由用户指定, 默认为一个空函数
    if(method == 'GET' && parameters != null)              //如果提交方式被指定为 GET,
    则将 parameters 的内容拼接到 URL 中, 并将 parameters 设置为 null
    {
        if(url.indexOf('?') > -1)
        {
            url += '&' + parameters;
        }
        else
        {
            url += '?' + parameters;
        }
        parameters = null;
    }
    request.open(method,url,true);                          //初始化 XMLHttpRequest 对象
    request.setRequestHeader('contentType','application/x-www-form-urlencoded');
    for(var name in headers)                                //设置由用户指定的 HTTP 头信息
    {
        request.setRequestHeader(name,headers[name]);
    }
    request.onreadystatechange = function()                  //注册 XMLHttpRequest 对象的
    readystatechange 事件处理函数
    {
        if(request.readyState == 1)                        //当 readyState 等于 1 时, 表示请求开始, 将当前 XMLHttpRequest 对象作为其参数调用 onLoadingEventHandler 函数
        {
            onLoadingEventHandler(request);
        }
        if(request.readyState == 4)                        //当 readyState 等于 4 时, 表示请求完成, 将当前 XMLHttpRequest 对象作为其参数调用 onCompleteEventHandler 函数
        {
            onCompleteEventHandler(request);
            if(request.status && request.status >= 200 && request.status < 300)
            {
                onSuccessEventHandler(request);              //当 HTTP 状态码大于等于 200 小于 300 时, 表示请求成功, 这时将当前 XMLHttpRequest 对象作为其参数调用 onSuccessEventHandler 函数
            }
            else
            {

```

```

        onFailureEventHandler(request); //否则表示请求失败，这时将当前 XMLHttpRequest 对
象作为其参数调用 onFailureEventHandler 函数
    }
}
request.send(parameters); //发送请求
}

```

### 7.3.8 完成前台功能模块：读取和发送留言

其他模块包括读取留言模块、发送留言模块、显示留言模块以及留言数据过滤模块。由于这些模块本身与留言本页面耦合度较高，而且每个模块的代码都不多，所以将它们集中放置到一个 JavaScript 文件中，并命名为 guestbook.js，其代码如下所示。

```

var url = 'php/service.php'; //设置请求的地址
/*
 * 留言本初始化程序
 */
function init()
{
    document.getElementById('btnSubmit').onclick = submitMessage; //注册按钮 btnSubmit 的 click 事
件处理函数为 submitMessage 函数
    readMessages(); //读取所有历史留言信息
}
/*
 * 读取留言功能模块
 */
function readMessages()
{
    /*
     * 设置 Ajax 请求的相关参数和回调函数
     */
    var options = {
        method: 'GET',
        parameters: 'action=getall',
        onLoading: function()
        {
            document.getElementById('loadingMsg').style.display = ""; //请求过程中显示读取留言的进
            度提示
        },
        onSuccess: function(x)
        {
            var resBack = x.responseText;
            try
            {
                var messages = eval('(' + resBack + ')'); //将返回的 JSON 字符串转换成
                JavaScript 对象
            }
            for(var i = 0; i < messages.length; i++)
            {

```

```

        displayMessage(messages[i]); //显示留言
    }
    document.getElementById('loadingMsg').style.display = 'none'; //留言信息读取成功后隐藏
读取留言的进度提示
    document.getElementById('btnSubmit').disabled = false; //留言信息读取成功后启用
提交按钮
    doScroll(); //将留言列表滚动到最低端
    }
    catch(e){}
    },
    onFailure:function()
    {
        document.getElementById('loadingMsg').style.display = 'none'; //请求失败时隐藏读取留言
的进度提示
        alert('Request failure.');//弹出对话框提示请求失败
    }
    }
    ajaxRequest(url,options); //发送请求
}
/*
* 显示留言功能模块
* 函数 displayMessage 接受一个包含留言数据的对象作为参数, 该对象有 name、message、postdate 属性,
分别对应留言信息的数据字段
* 通过解析该对象的数据, 创建一条表示留言记录的 li 元素添加到页面的留言列表中
*/
function displayMessage(data)
{
    var name = data.name;
    var date = data.postdate;
    var message = data.message;
    var span = document.createElement('span');
    var nameText = document.createTextNode(name);
    span.appendChild(nameText);
    var dateSpan = document.createElement('span');
    var dateText = document.createTextNode(date);
    dateSpan.appendChild(dateText);
    dateSpan.className = 'date';
    var p = document.createElement('p');
    var contentText = document.createTextNode(message);
    p.appendChild(contentText);
    var li = document.createElement('li');
    li.appendChild(span);
    li.appendChild(dateSpan);
    li.appendChild(p);
    document.getElementById('msgList').appendChild(li);
}
/*
* 发送留言功能模块
*/
function submitMessage()
{

```

```

/*
 * 获取访客填写的姓名和留言内容
 */
var name = document.getElementById('txtName').value;
var message = document.getElementById('txtContent').value;
/*
 * 设置 Ajax 请求的相关参数和回调函数
 */
var options = {
    method: 'GET',
    parameters: 'action=addnew&name=' + escapeHTML(name) + '&message=' +
escapeHTML(message),
    onLoading: function()
    {
        document.getElementById('submitMsg').style.display = ''; //显示发送留言的进度提示
        document.getElementById('btnSubmit').disabled = true; //在请求过程中禁用提交按钮
    },
    onSuccess: function(x)
    {
        var resBack = x.responseText;
        /*
         * 如果请求返回的字符串数据是 ok, 则表示留言提交成功。这时候调用显示留言功能模块将刚才
提交的留言立即显示在留言列表的末端
         */
        if(resBack == 'ok')
        {
            var date = new Date();
            var postDate = date.getFullYear() + '-' + (date.getMonth() + 1) + '-' + date.getDate();
            var msg = {
                name: name,
                message: message,
                postdate: postDate
            }
            displayMessage(msg);
            doScroll();
        }
        /*
         * 否则弹出对话框提示提交失败
         */
        else
        {
            alert('Submit failure.');
```

```

        document.getElementById('submitMsg').style.display = 'none'; //隐藏提交留言的进度提示
        document.getElementById('btnSubmit').disabled = false; //启用提交按钮
    },
    onFailure: function()
    {
        document.getElementById('submitMsg').style.display = 'none'; //隐藏提交留言的进度提示
        document.getElementById('btnSubmit').disabled = true; //启用提交按钮
        alert('Request failure.');
```

```

        //请求失败时弹出对话框提示请求失败
    }
}
```



```

    }
  }
  ajaxRequest(url,options);           //发送请求
}
/*
 * 将留言列表滚动到最下端
 */
function doScroll()
{
    var height = document.getElementById('msgList').offsetHeight;
    var totalHeight = document.getElementById('msgList').scrollHeight;
    document.getElementById('msgList').scrollTop = totalHeight - height;
}
/*
 * 留言数据过滤模块
 * 将访客提交的内容中包含的<和>进行替换, 以防止留言信息中包含 HTML 标签而对页面进行破坏
 */
function escapeHTML(str)
{
    str = str.replace('<','&lt;');
    str = str.replace('>','&gt;');
    return str;
}
window.onload = init;                //页面加载完时初始化留言本程序

```

关于各模块代码功能的解释, 读者可以参见代码中的注释。

注意: 程序中使用了 JSON 作为传输历史留言记录的数据格式。本书将会在第 11 章中向读者介绍 JSON 的相关知识和使用方法。

### 7.3.9 整合留言本程序

完成所有的功能模块后, 现在对留言本程序进行整合, 主要的工作就是将 JavaScript 代码引入到前台的页面中去。将 index.html 页面的代码进行修改, 代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Ajax Guest Book</title>
<link type="text/css" rel="stylesheet" href="css/style.css" />
<script type="text/javascript" src="script/ajaxRequest.js"></script>
<script type="text/javascript" src="script/guestbook.js"></script>
</head>

<body>
    <h1>Guest Book<span id="loadingMsg" style="display:none;" class="msgBox">(loading...)</span></h1>
    <ul id="msgList">

```

```

</ul>
<form name="fmMsg" id="fmMsg" action="?" method="post">
  <h2>Message<span id="submitMsg" style="display:none;" class="msgBox">(loading...)</span></h2>
  <label for="txtName">name</label>
  <input name="txtName" type="text" id="txtName" value="guest" onfocus="this.select();" />
  <label for="txtContent">Message</label>
  <textarea name="txtContent" rows="4" id="txtContent" onfocus="this.select();">something to
say...</textarea>
  <input type="button" value="Click here to submit your message!" id="btnSubmit" disabled="disabled" />
</form>
</body>
</html>

```

## 7.4 留言本的功能测试

做完程序的整合后，就可以开始进行留言本的功能测试，以检测其功能是否符合之前所提出来的需求。

(1) 为了测试留言读取功能，可以使用下面的 SQL 语句预先在数据中添加一些留言信息。

```

INSERT INTO 'message' ('id', 'name', 'message', 'postdate') VALUES
(28, 'lily', 'I miss you,rob.', '2007-11-06'),
(29, 'Frank', 'Hey,what are you doing now?', '2007-11-06'),
(30, 'Susan', 'Do you like rock?', '2007-11-06');

```

(2) 然后将 ajaxGuestBook 放置到 Apache 的虚拟目录中，并通过浏览器访问留言本前台页面的 URL 地址，效果如图 7.7 所示。

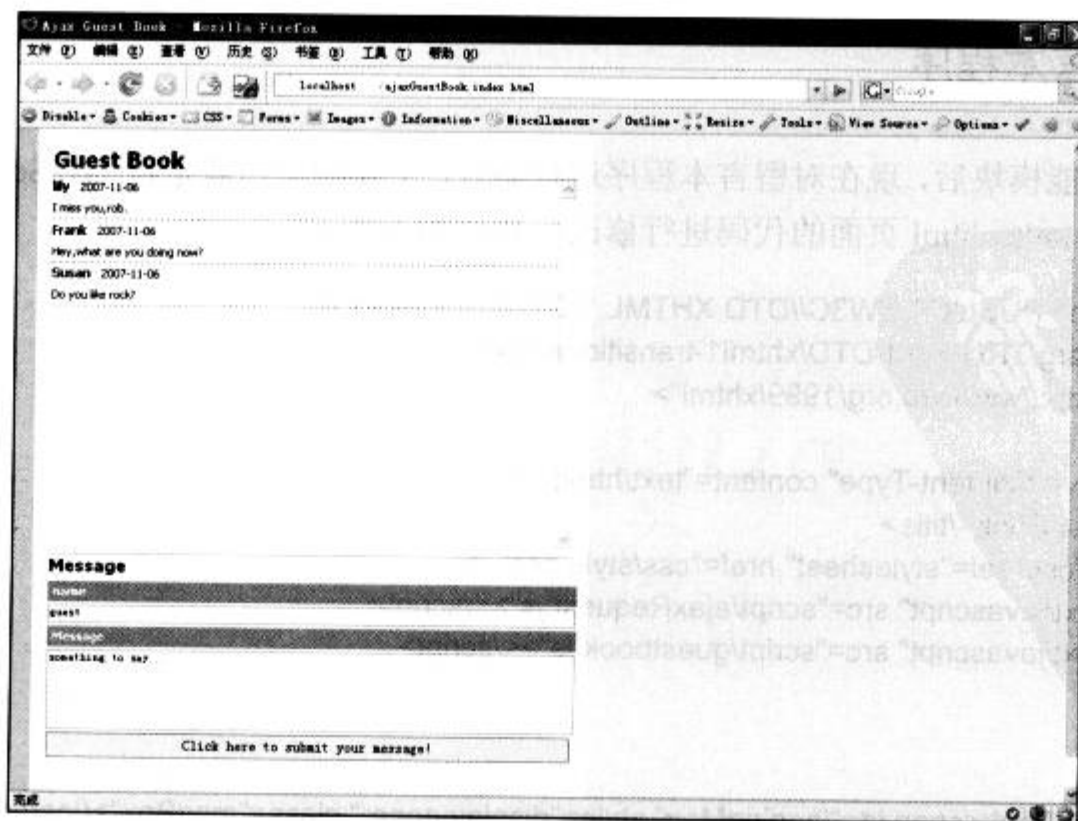


图 7.7 打开留言本

(3) 可以看到, 通过短暂的加载后, 数据库中的留言记录被完整地显示在了页面上的留言列表中。通过 Firebug 的控制台可以看到程序读取留言所发出的 Ajax 请求的详细信息, 如图 7.8~图 7.10 所示。

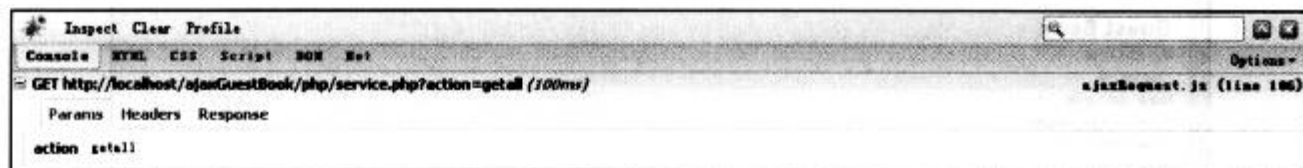


图 7.8 请求详细信息 1

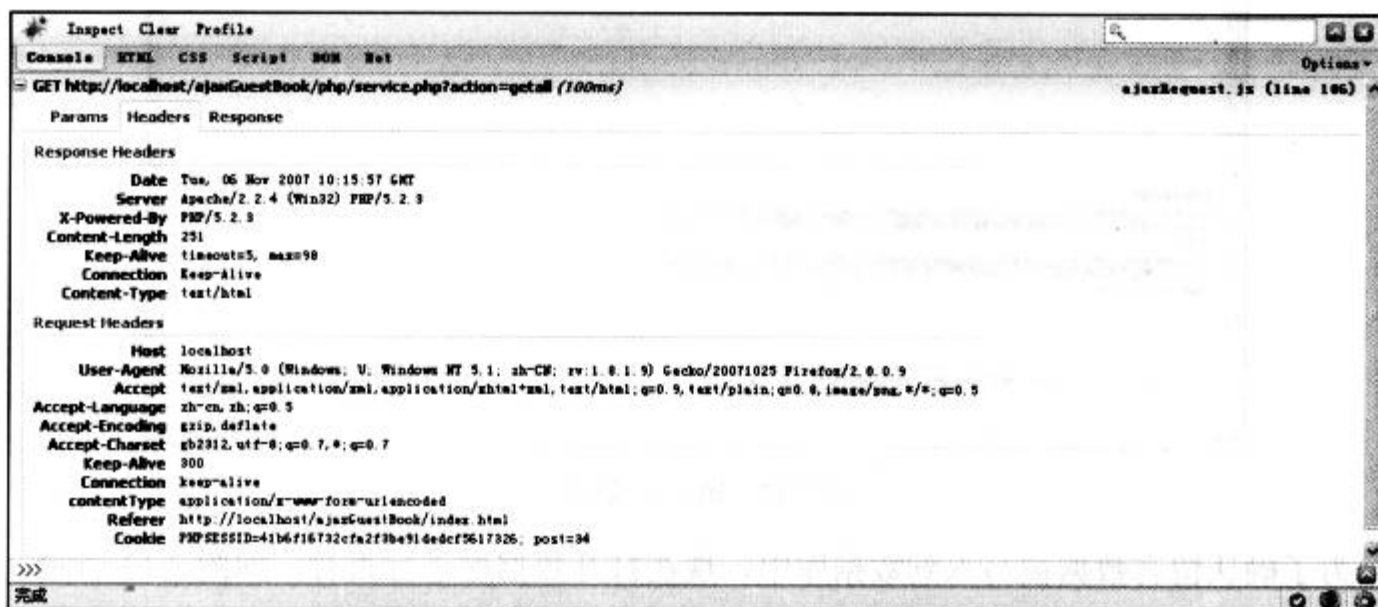


图 7.9 请求详细信息 2

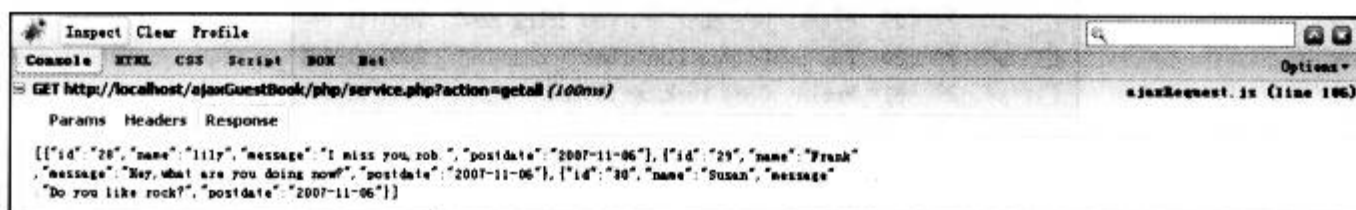


图 7.10 请求详细信息 3

(4) 然后再来测试发送留言的功能。填写 Message 表单, 并单击“提交”按钮, 效果如图 7.11 所示。

Message(loading...)

name

Robin

Message

Good luck, my friend.

Click here to submit your message!

图 7.11 提交留言

(5) 单击“提交”按钮后, Message 标题后面出现 Loading 字样提示程序正在处理中, 同时为了防止用户重复提交, 提交按钮被禁用。在短暂的等待后, 提交成功。Loading 字样消失, 按钮恢复原状, 页面中的留言列表显示了最新提交的留言信息, 如图 7.12 所示。

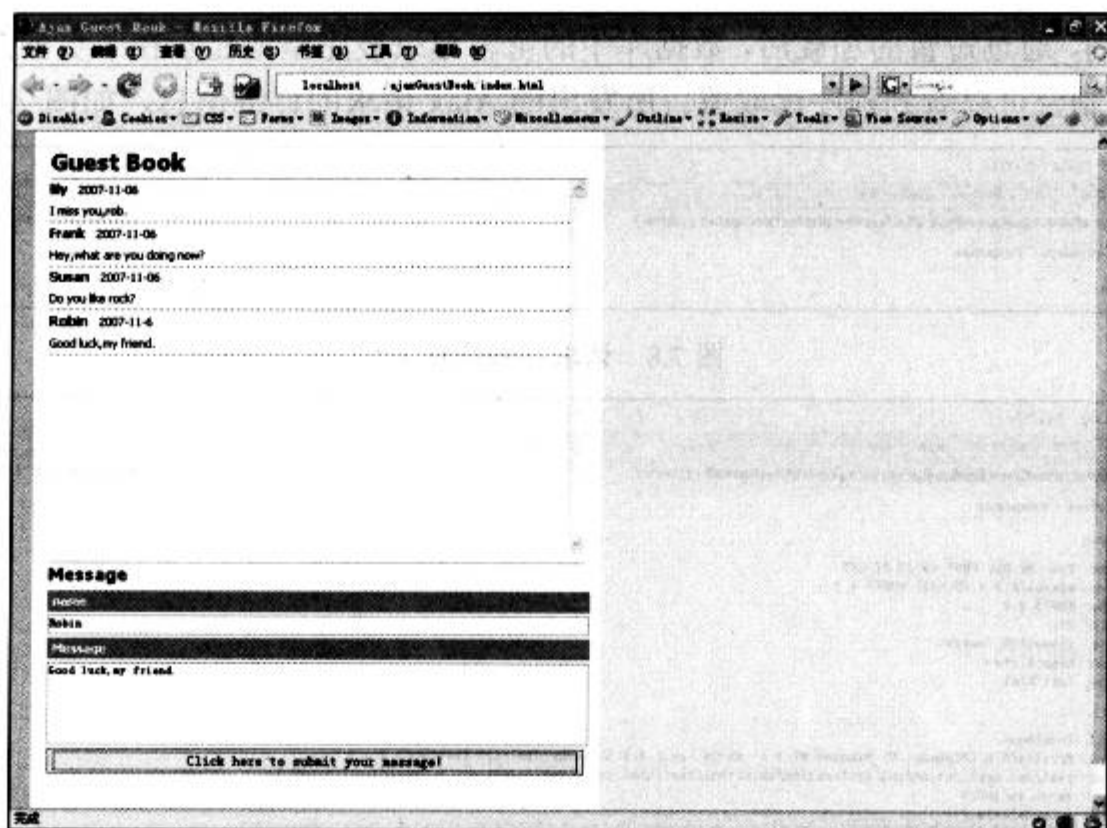


图 7.12 留言提交成功

(6) 为了确认留言数据被写入到数据库中，现在打开数据库进行查看，如图 7.13 所示。

			id	name	message	postdate
<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	28	lily	I miss you, rob.	2007-11-06
<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	29	Frank	Hey, what are you doing now?	2007-11-06
<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	30	Susan	Do you like rock?	2007-11-06
<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	31	Robin	Good luck, my friend.	2007-11-06

图 7.13 查看数据库

毫无疑问，留言数据被成功插入到数据库中。

## 7.5 小 结

本章笔者带领读者一起创建了一个更为复杂和完善的 Ajax 应用程序：Ajax 留言本。从留言本的需求确认，到系统设计，到代码实现和测试，使读者比较完整地经历了一个 Ajax 应用程序的开发流程。本章中所创建的留言本相对来说仍然是非常简单的 Ajax 应用程序，但是比较完整地体现了 Ajax 应用程序的特点。希望读者能够认真学习这个实例。



# 第 8 章

## 调试技巧

- ▶▶ 深入解析 Firebug 的调试功能
- ▶▶ 使用 Aptana 的集成调试功能
- ▶▶ 小结

理想的软件开发是不需要调试的，但在现实开发过程中，开发人员总是会会因为不同的原因而犯各种各样的错误，以致于给所构建的系统带来不同程度的危害。有些错误浅显而低级，例如用错了大小写，或者写错了变量名等；而有些错误是复杂的逻辑错误，这类错误往往隐藏较深，不容易找出错误的原因；另外一些错误可能是功能实现的方式有问题，导致程序有性能瓶颈等。可以说，软件开发就是不断编码和调试的过程。使用好的调试工具并掌握好的调试技巧能够加快软件开发进程，提高产品质量。本章将向读者详细介绍使用 Firebug 和 Aptana 调试 JavaScript 程序的一些技巧。



## 8.1 深入解析 Firebug 的调试功能

在本书的第 5 章中已经向读者介绍过 Firefox 浏览器的优秀调试插件 Firebug 的界面和基本功能。本节将向读者深入讲解如何利用 Firebug 的控制台输出各种自定义的信息、查看错误提示，如何利用命令行工具在页面上执行 JavaScript 代码，以及如何使用脚本查看器进行脚本的调试等。

### 8.1.1 检查常规错误

当 Firefox 遇到一个 JavaScript 错误时，Firebug 会在其控制台输出一个错误信息。这个错误信息包含错误的描述、发生错误的代码片断、包含该代码片断的函数或者方法以及事件对象信息。下面这个示例中，在测试按钮的事件处理函数中调用了不存在的函数，代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>console demo</title>
<script type="text/javascript">
function errorTest()
{
    notExistFunction();          //不存在的函数
}
</script>
</head>

<body>
<input type="button" value="test button" onclick="errorTest();" />
</body>
</html>
```

用 Firefox 打开页面，如图 8.1 所示。单击 test button 按钮，按钮的 click 事件处理函数 errorTest 会被调用，这时 errorTest 会调用一个不存在的函数 notExistFunction，从而引发一个错误。浏览器遇到脚本错误时，右下角 Firebug 的绿色小图标会变成红色以提示当前页面存在脚本错误，如图 8.2 所示。



图 8.1 示例程序界面



图 8.2 错误提示

红色图标后面的数字表示错误的个数。单击红色图标或者按 F12 键打开 Firebug 界面, 如图 8.3 所示。

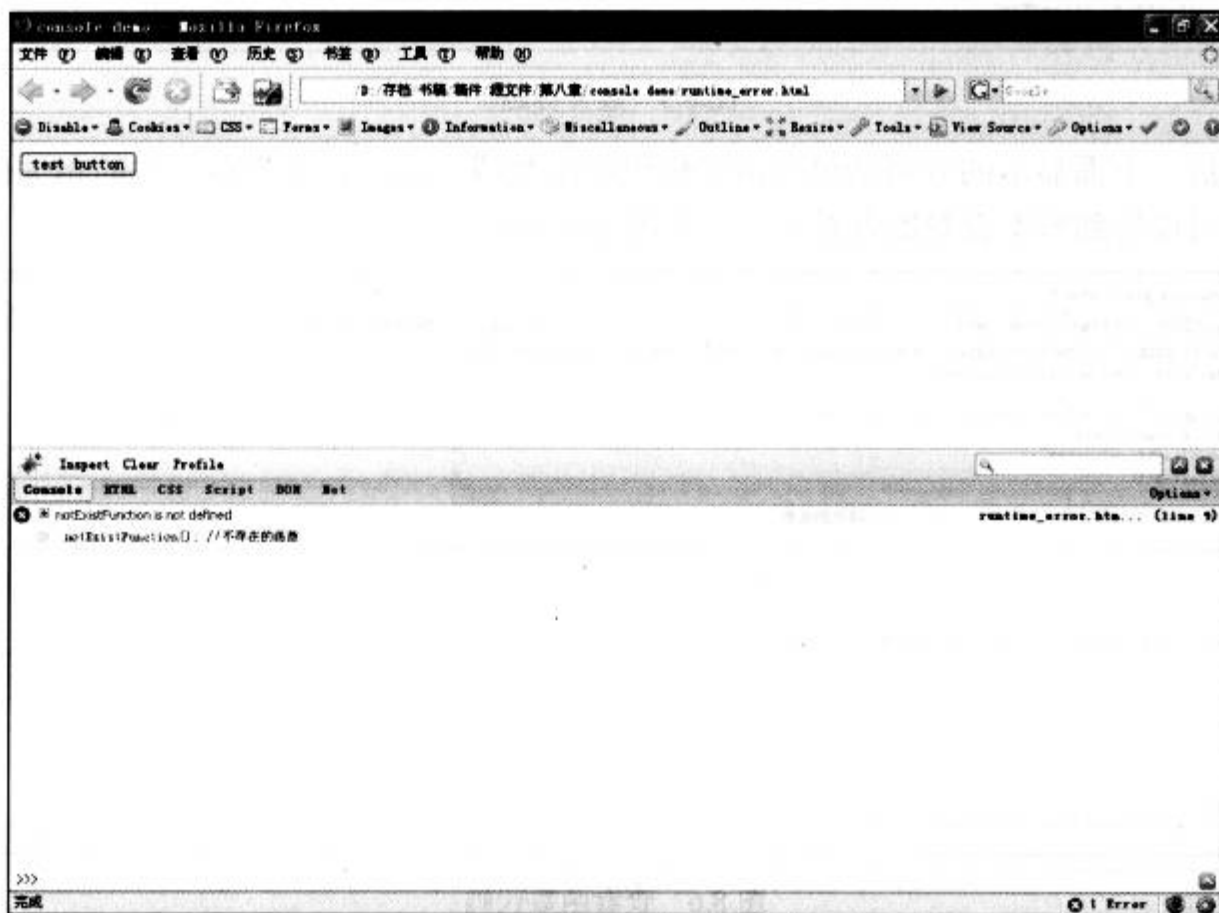


图 8.3 错误提示

错误被控制台用红色的字体, 配上红色的错误图标醒目地显现出来。错误的标题是对错误的描述, 这里 `notExistFunction is not defined` 指 `notExistFunction` 没有被定义。标题下面是发生错误的那一行代码。标题右边是发生错误的文件名和错误代码所在的行号。单击标题后面的文件说明或者标题下面的错误行代码提示, 都可以转到脚本查看器并完整地查看该文件的脚本代码, 如图 8.4 所示。

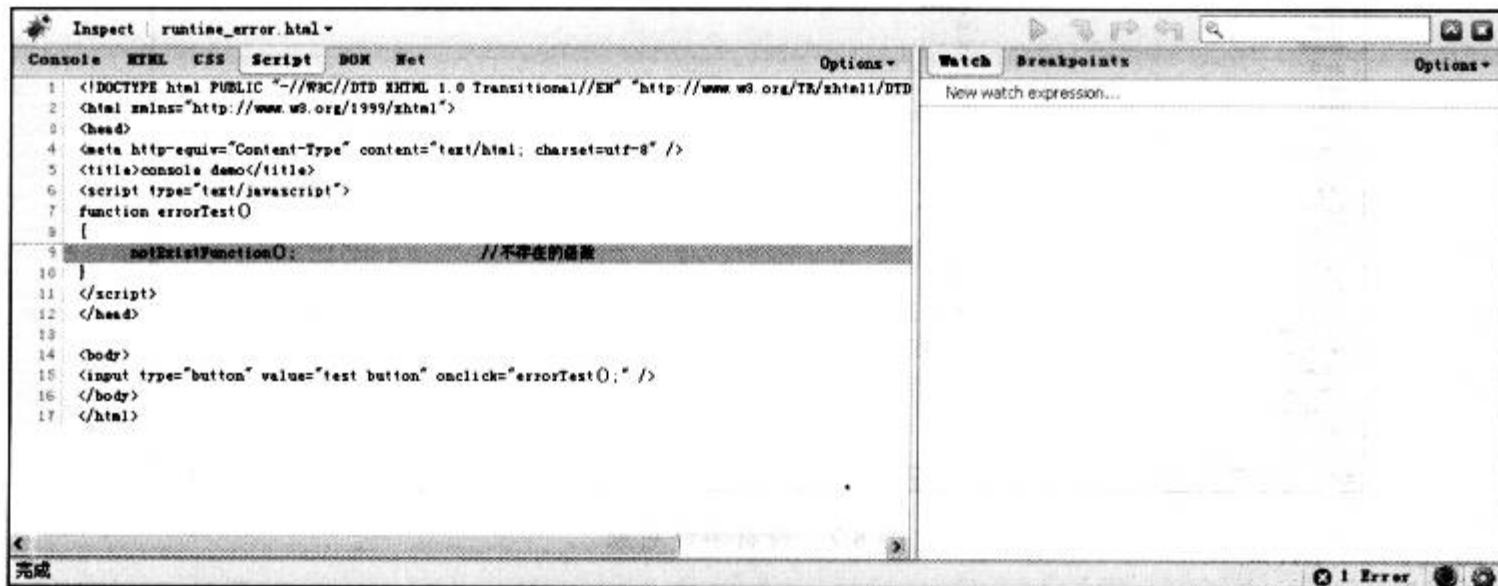


图 8.4 查看错误代码

标题前面有一个“+”号的小图标, 表示标题可以被展开, 单击图标展开标题, 如图 8.5 所示。

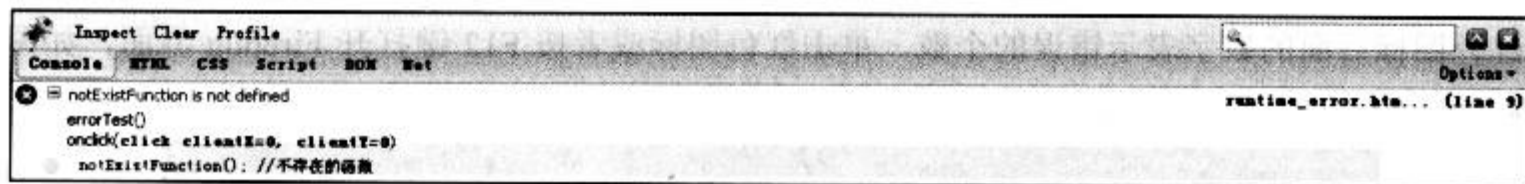


图 8.5 展开标题

标题展开后，下面显示的分别是包含错误代码的函数或方法，以及当前事件的相关信息。单击函数或方法名，可以转到脚本查看器查看代码，如图 8.6 所示。

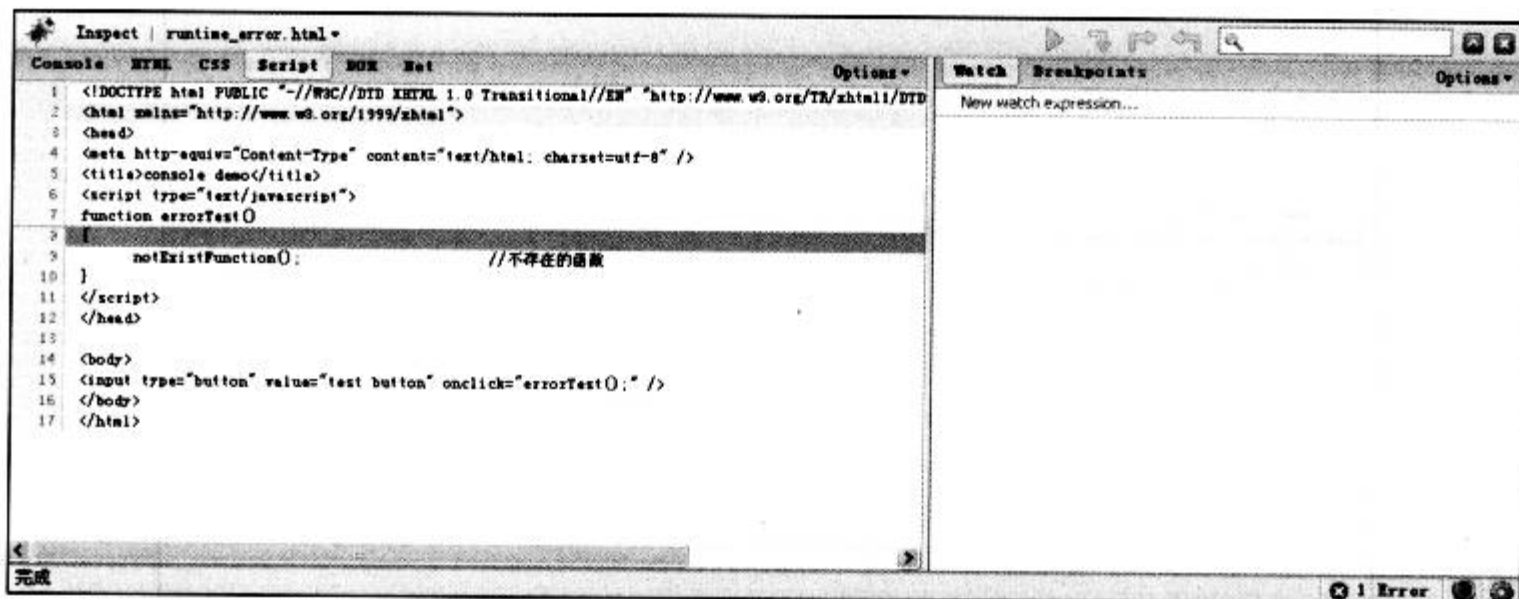


图 8.6 查看函数代码

单击事件的描述信息，可以转到 DOM 查看器查看事件对象的详细信息，如图 8.7 所示。

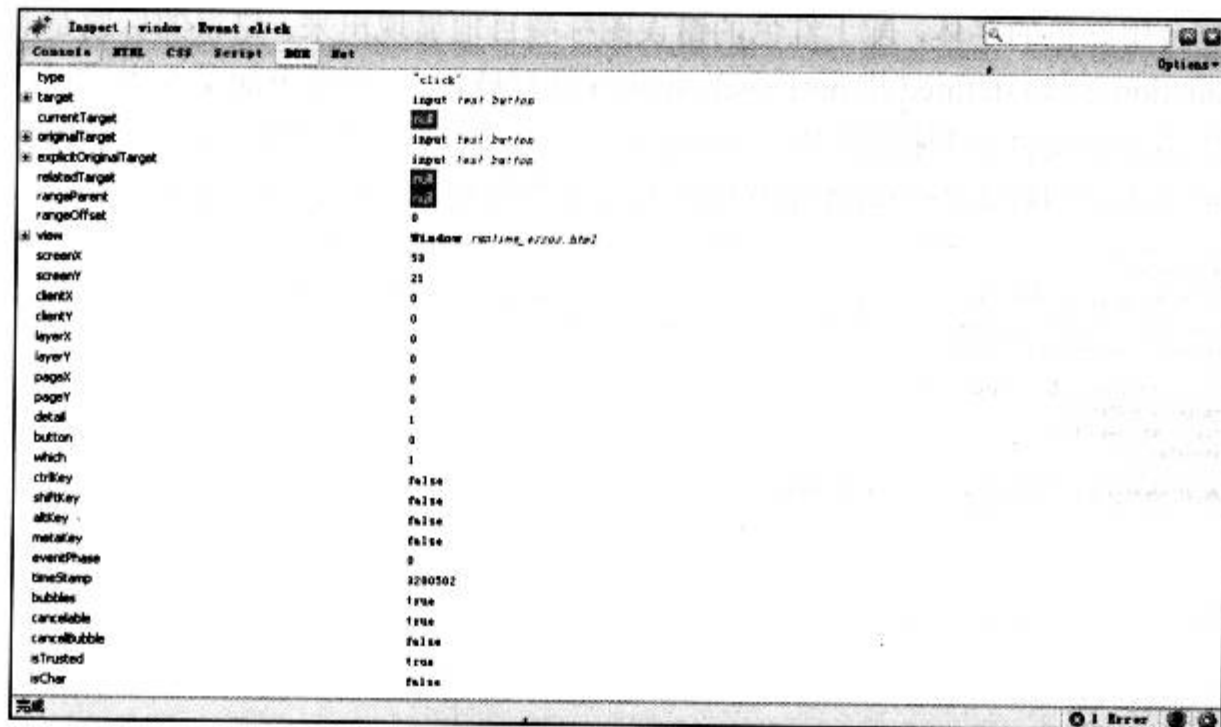


图 8.7 查看事件对象

当程序中出现错误时，通过查看 Firebug 控制台输出的错误信息可以让开发者快速定位分析并修复错误。

## 8.1.2 完善的 log 功能

在调试程序时，经常需要让程序在运行过程中输出一些信息，使得开发者可以实时掌握到程序运行的情况。Firebug 的控制台提供了完整的 log 功能，在第 5 章中读者已经见过 `console.log` 的使用，下面来向读者介绍所有的 log 语法。

### 1. console.log

`console.log` 提供了在控制台中输出信息的基本方法，其语法如下所示。

```
console.log(message1[,message2,...,messageN]);
```

在代码被执行时，其参数会被连接在一起输出到 Firebug 的控制台中。`console.log` 还支持 4 种占位符，如表 8-1 所示。

表 8-1 console.log 支持 4 种占位符

占 位 符	说 明
%s	字符串
%d,%i	整数
%f	浮点数
%o	对象

占位符只能在 `console.log` 的第一个参数中使用。当第一个参数中包含占位符时，程序会根据占位符的数量，取从第二个参数开始的足够数量的参数替换到对应的占位符所在的位置，剩下的其他参数则按照默认行为被连接到输出信息的末尾。下面的示例演示了 `console.log` 的用法，其代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>console.log demo</title>
<script type="text/javascript">
console.log('message start');
var number = 123.22;
var int = 55;
var string = '321';
var object = {name:'rob',sex:'mail'};
console.log('number = ',number);
console.log('number = %f,number);
console.log('int = %d,int);
console.log('string = %s,string);
console.log('object = %o,object);
console.log('number = %f , int = %d , string = %s , object = %o',number,int,string,object,'others...');
console.log('message end')
</script>
```

```
</head>
```

```
<body>
```

```
</body>
```

```
</html>
```

使用 Firefox 访问该示例页面，可以看到在 Firebug 的控制台中输出了指定的信息，如图 8.8 所示。

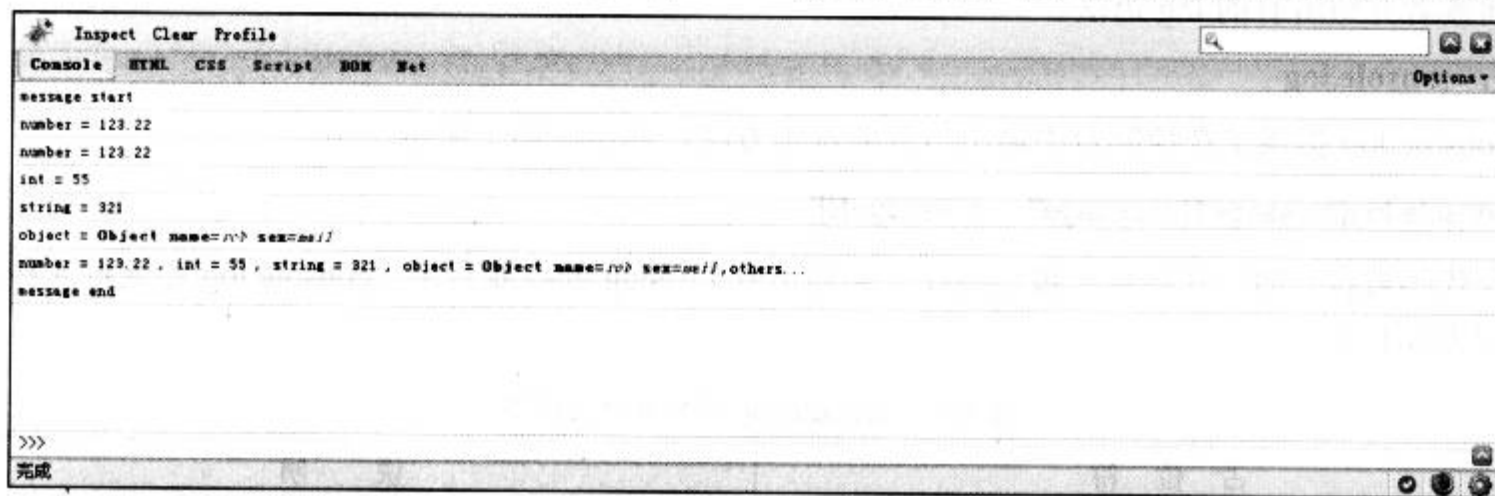


图 8.8 console.log 示例

## 2. console.debug

console.debug 与 console.log 一样可以接受多个参数，所不同的是通过 console.debug 输出的信息会被添加一个链接，单击链接时会根据参数的类型自动转换到其他视图。当参数为 DOM 对象或者 JavaScript 对象时，会转到 DOM 查看器显示对象的详细信息。当参数为 HTML 元素时，则会转到 HTML 查看器并定位到该元素节点上。如果参数为一个函数或者方法，则会转到脚本查看器并定位到函数或者方法的定义所在行。如果参数只是一个数字或者字符串等基本类型的值，则不会添加任何链接。比较特殊的是当参数为数组时，Firebug 会分析其中每个元素的类型并添加相应的链接。下面的示例显示了 console.debug 的用法，代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>console.debug demo</title>
<script type="text/javascript">
function func(a,b)
{
    return a + b;
}
window.onload = function()
{
    var object = {name:'rob',sex:'male'};
    var number = 12;
    var string = '123';
    var array = [1,2,3,object,number,string];
```



```
var node = document.getElementById('node');
console.debug(object);
console.debug(array);
console.debug(number);
console.debug(string);
console.debug(func);
console.debug(node);
console.debug(window);
console.debug(document);
console.debug(screen);
console.debug(navigator);
}
</script>
</head>

<body>
  <p id="node">console.debug demo</p>
</body>
</html>
```

程序运行后，在 Firebug 控制台输出的信息如图 8.9 所示。

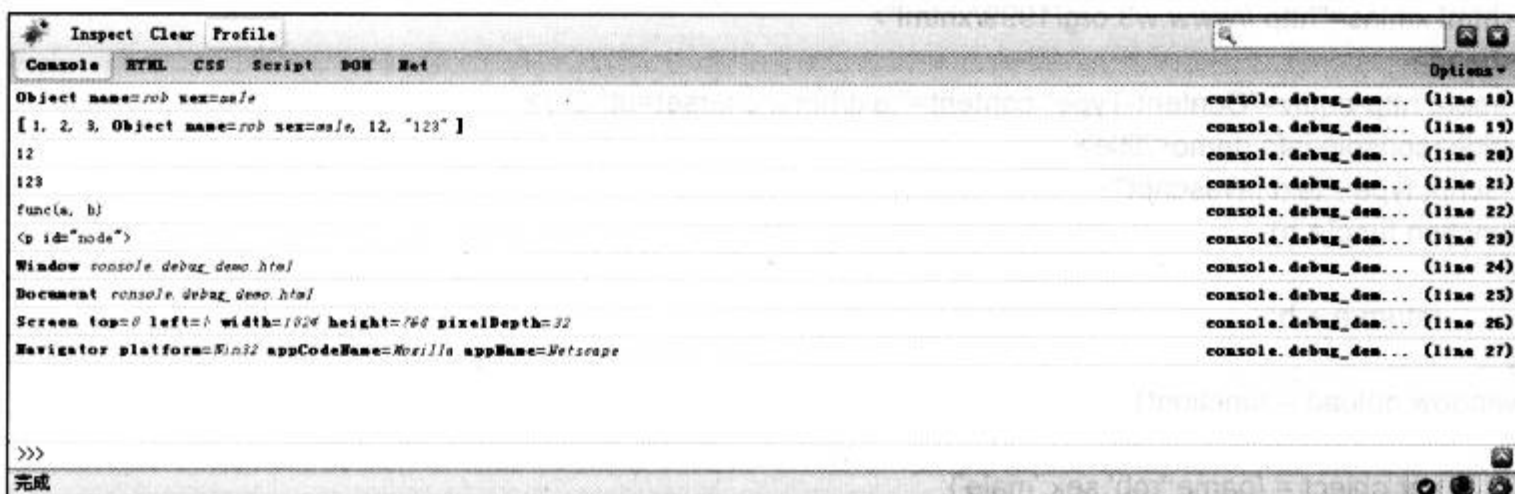


图 8.9 console.debug 示例

单击第一条信息的链接，Firebug 转到 DOM 查看器视图并显示了对象的详细信息，如图 8.10 所示。

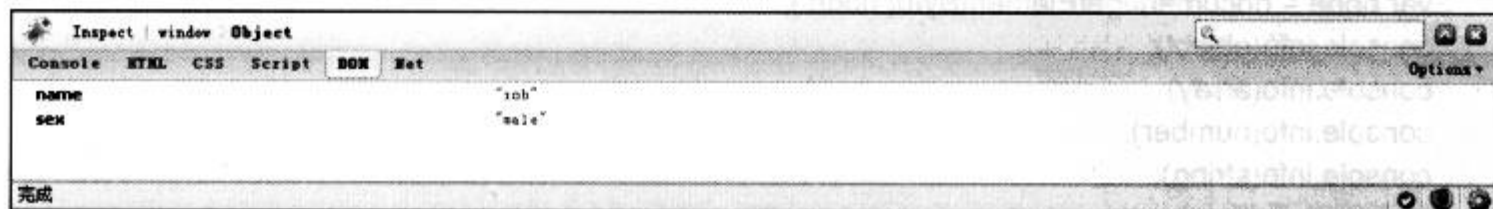


图 8.10 查看对象详细信息

单击 func(a,b)这条信息的链接，Firebug 转到脚本查看器视图，并定位到该函数定义所在的行，如图 8.11 所示。

单击<p id= "node">这条信息时，则会转到 HTML 查看器并定位到该元素节点上，如图 8.12 所示。

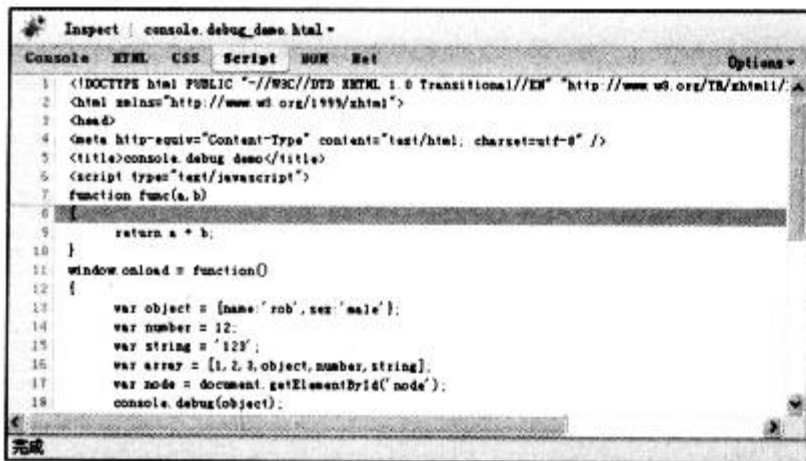


图 8.11 查看函数定义

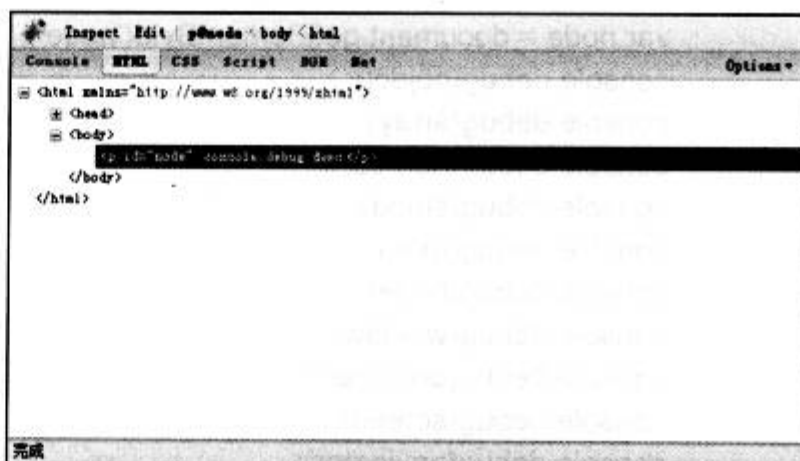


图 8.12 查看 HTML 元素

### 3. console.info

console.info 的功能与 console.debug 相同，所不同的是它会在输出到 Firebug 控制台的信息前面加上一个表示注意信息的小图标。例如，将上面示例中的 console.debug 替换成 console.info，代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>console.info demo</title>
<script type="text/javascript">
function func(a,b)
{
    return a + b;
}
window.onload = function()
{
    var object = {name:'rob',sex:'male'};
    var number = 12;
    var string = '123';
    var array = [1,2,3,object,number,string];
    var node = document.getElementById('node');
    console.info(object);
    console.info(array);
    console.info(number);
    console.info(string);
    console.info(func);
    console.info(node);
    console.info(window);
    console.info(document);
    console.info(screen);
    console.info(navigator);
}
</script>
</head>
```

```

<body>
  <p id="node">console.info demo</p>
</body>
</html>

```

其输出效果如图 8.13 所示。

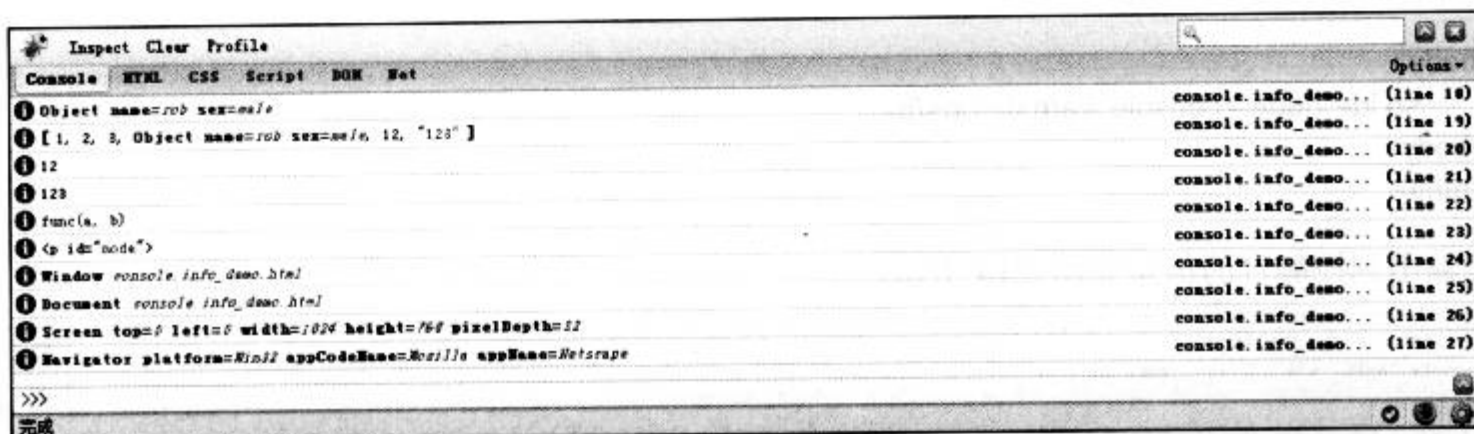


图 8.13 console.info 示例

#### 4. console.warn

console.warn 的功能与 console.debug 相同，所不同的是它会在输出到 Firebug 控制台的信息前面加上一个表示警告的图标，并将信息背景色设置为绿色。例如，将上面示例代码中的 console.info 全部修改为 console.warn，代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>console.warn demo</title>
<script type="text/javascript">
function func(a,b)
{
    return a + b;
}
window.onload = function()
{
    var object = {name:'rob',sex:'male'};
    var number = 12;
    var string = '123';
    var array = [1,2,3,object,number,string];
    var node = document.getElementById('node');
    console.warn(object);
    console.warn(array);
    console.warn(number);
    console.warn(string);
    console.warn(func);
    console.warn(node);

```

```

        console.warn(window);
        console.warn(document);
        console.warn(screen);
        console.warn(navigator);
    }
</script>
</head>

<body>
    <p id="node">console.warn demo</p>
</body>
</html>

```

其输出到控制台的信息如图 8.14 所示。

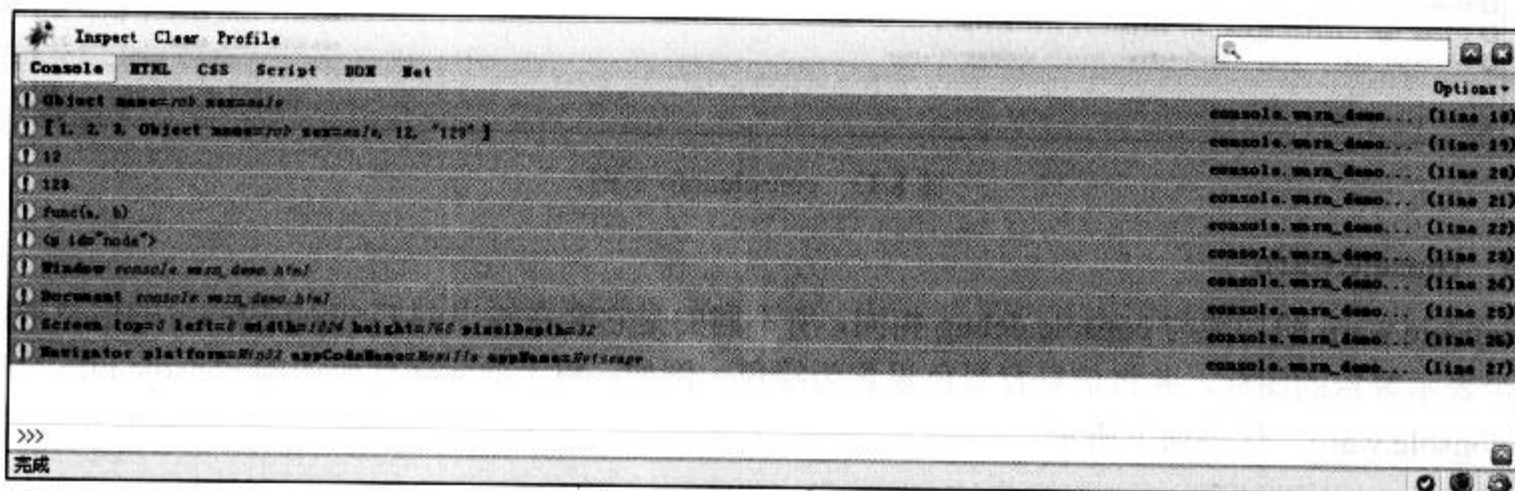


图 8.14 console.warn 示例

## 5. console.error

console.error 的功能与 console.debug 相同，所不同的是它会在输出到 Firebug 控制台的信息前面加上表示错误的图标，并在浏览器右下角 Firebug 的小图标中提示错误。将上面示例代码中的 console.warn 全部修改为 console.error，代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>console.error demo</title>
<script type="text/javascript">
function func(a,b)
{
    return a + b;
}
window.onload = function()
{
    var object = {name:'rob',sex:'male'};
    var number = 12;
    var string = '123';

```

```

var array = [1,2,3,object,number,string];
var node = document.getElementById('node');
console.error(object);
console.error(array);
console.error(number);
console.error(string);
console.error(func);
console.error(node);
console.error(window);
console.error(document);
console.error(screen);
console.error(navigator);
}
</script>
</head>

<body>
  <p id="node">console.error demo</p>
</body>
</html>

```

其输出效果如图 8.15 所示。

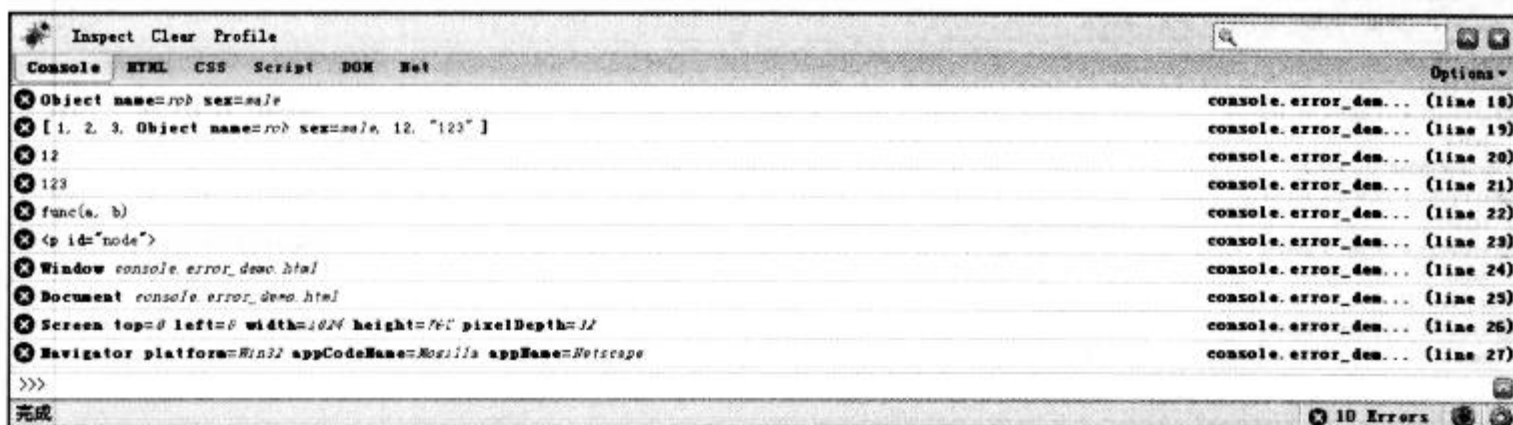


图 8.15 console.error 示例

## 6. console.assert

console.assert 可以进行断言，它接受一个表达式作为参数，当表达式的值为 true 时，不输出任何信息；当表达式的值为 false 时，输出一条表示断言失败的错误信息。console.assert 还可以接受更多的参数，以此作为断言失败时显示的信息。下面的示例使用 console.assert 来判断传入函数的参数的数据类型，代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>console.assert demo</title>
<script type="text/javascript">
function func(a,b)

```



```

{
    console.assert(typeof a == 'string','argument a is not a string');
    console.assert(typeof b == 'number','argument b is not a number');
    //do something...
}
console.log("func('123',1)");
func('123',1);
console.log("func(123,1)");
func(123,1);
console.log("func(new Object(),'123')");
func(new Object(),'123');
console.log("func('sss',[])");
func('sss',[]);
</script>
</head>

<body>
</body>
</html>

```

其输出到控制台的结果如图 8.16 所示。

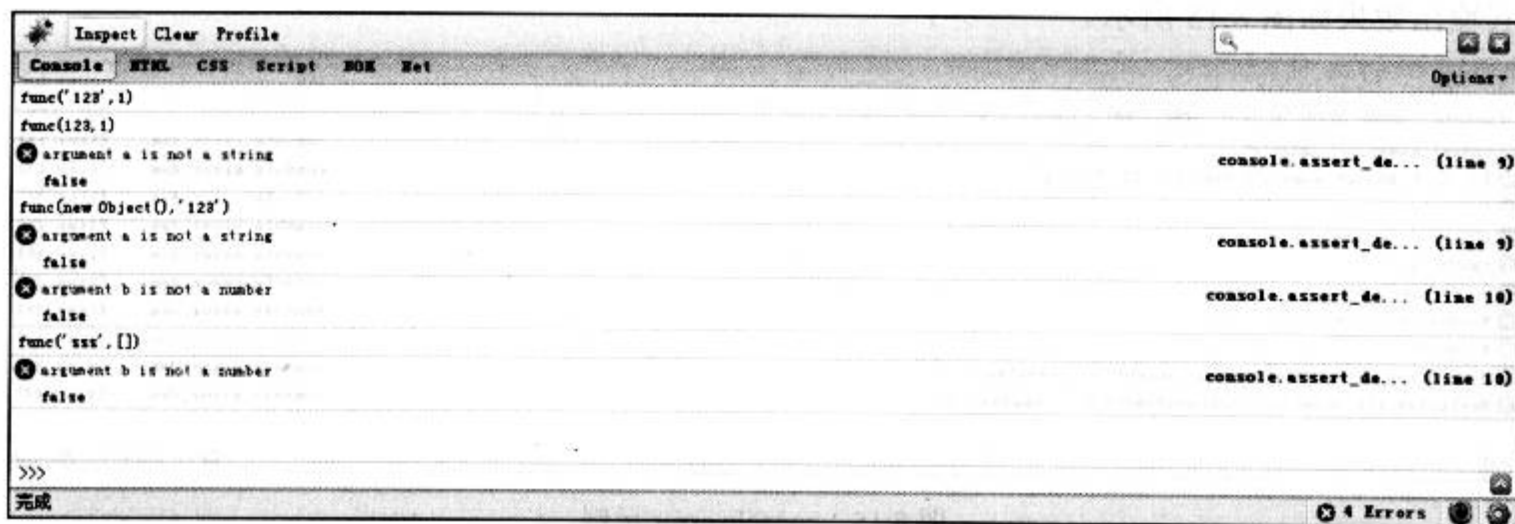


图 8.16 console.assert 示例

## 7. console.dir

console.dir 可以直接将对象或 HTML 元素的详细信息输出到 Firebug 中，就如同在 DOM 查看器中查看到的信息一样。下面的例子演示了 console.dir 的用法。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>console.dir demo</title>
<script type="text/javascript">
window.onload = function()
{

```

```
var object = {  
    a:'123',  
    b:44,  
    c:null,  
    d:function(){}  
}  
console.log('object information');  
console.dir(object);  
console.log('node information');  
console.dir(document.getElementById('node'));  
}  
</script>  
</head>  
  
<body>  
<p id="node">console.dir demo</p>  
</body>  
</html>
```

其在控制台中输出的内容如图 8.17 所示。

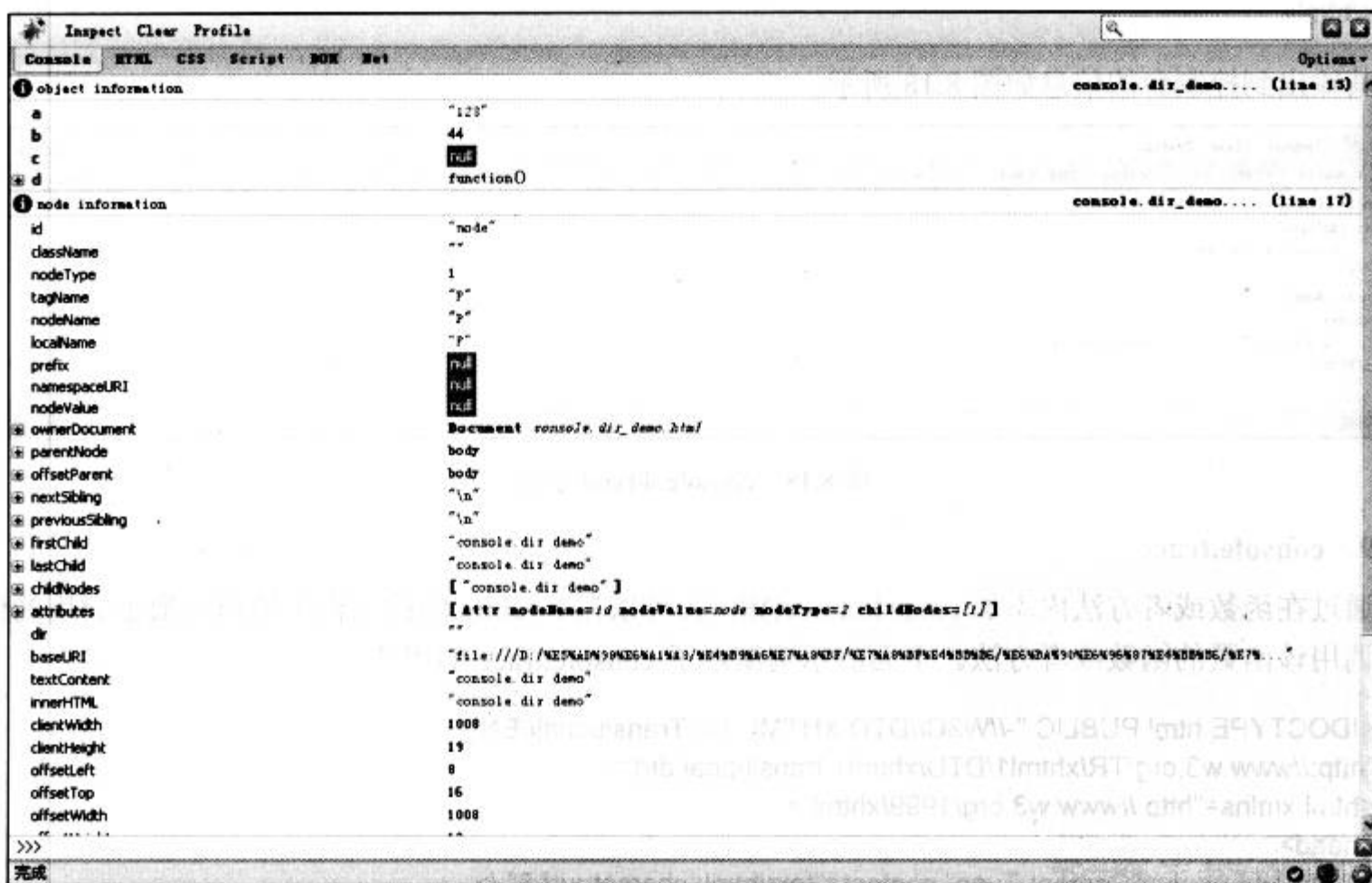


图 8.17 console.dir 示例

## 8. console.dirxml

`console.dirxml` 能够将 HTML 或 XML 节点元素的源代码输出到控制台中,就如同在 HTML 查看器中看到的效果一样,参考下面的示例代码。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>console.dirxml demo</title>
<script type="text/javascript">
window.onload = function()
{
    console.log('node: p');
    console.dirxml(document.getElementById('node'));
    console.log('node: body');
    console.dirxml(document.body);
}
</script>
</head>

<body>
<p id="node">console.dirxml demo</p>
</body>
</html>

```

其输出到控制台的信息如图 8.18 所示。

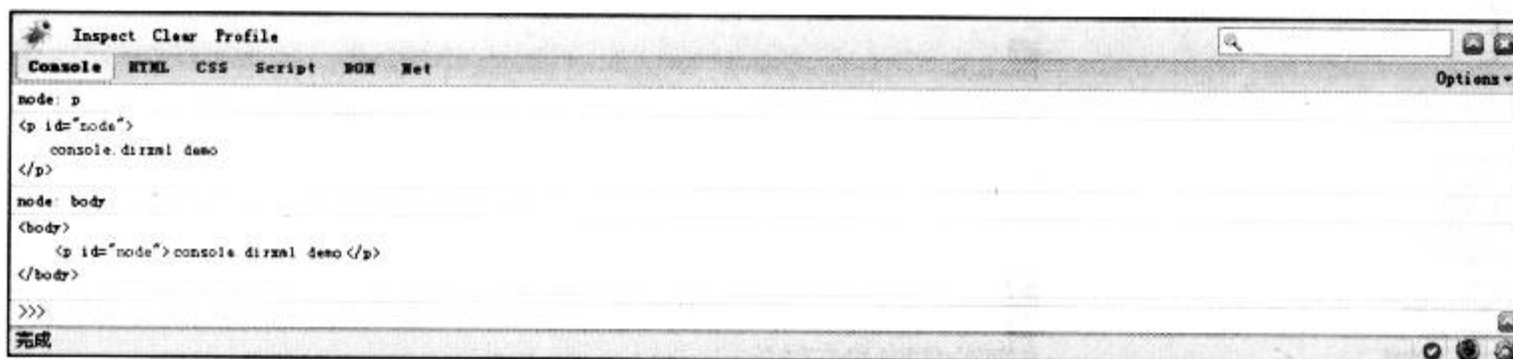


图 8.18 console.dirxml 示例

## 9. console.trace

通过在函数或者方法内添加 console.trace() 语句, 可以在 Firebug 的控制台中输出函数被调用的信息以及调用该函数的函数或者方法。下面的示例演示了 console.trace 的用法。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>console.trace demo</title>
<script type="text/javascript">
function func(a,b,c)
{
    console.trace();
}

```

```

function testA()
{
    func();
}
function testB()
{
    func(1,2,3);
}
var testC = function()
{
    func('parameter',{a:1,b:2,c:3});
}
testA();
testB();
testC();
(function()
{
    func('special');
})();
</script>
</head>

<body>
</body>
</html>

```

程序中先后通过4个函数来调用添加 trace 语句的函数 func, 其在控制台输出的信息如图 8.19 所示。

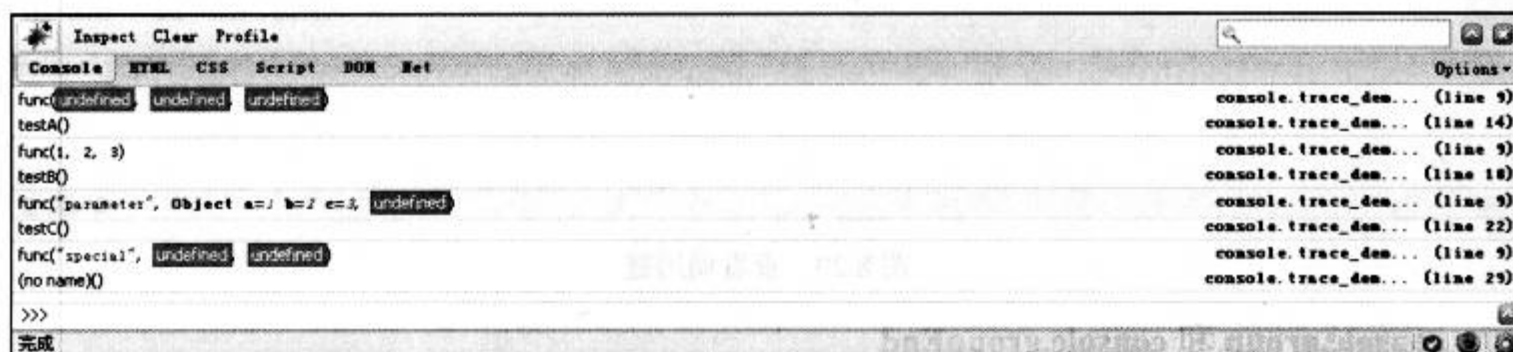


图 8.19 console.trace 示例

如果函数或者方法存在多层嵌套调用, 则 console.trace 会输出整个调用链的信息, 示例代码如下。

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>console.trace demo</title>
<script type="text/javascript">
function funcA()
{
    console.trace();
}

```

```
}  
function funcB()  
{  
    funcA();  
}  
function funcC()  
{  
    funcB();  
}  
function funcD()  
{  
    funcC();  
}  
funcD();  
</script>  
</head>  
  
<body>  
</body>  
</html>
```

在上面的程序代码中，funcA、funcB、funcC、funcD 存在嵌套调用关系。程序向 Firebug 控制台输出的信息如图 8.20 所示。

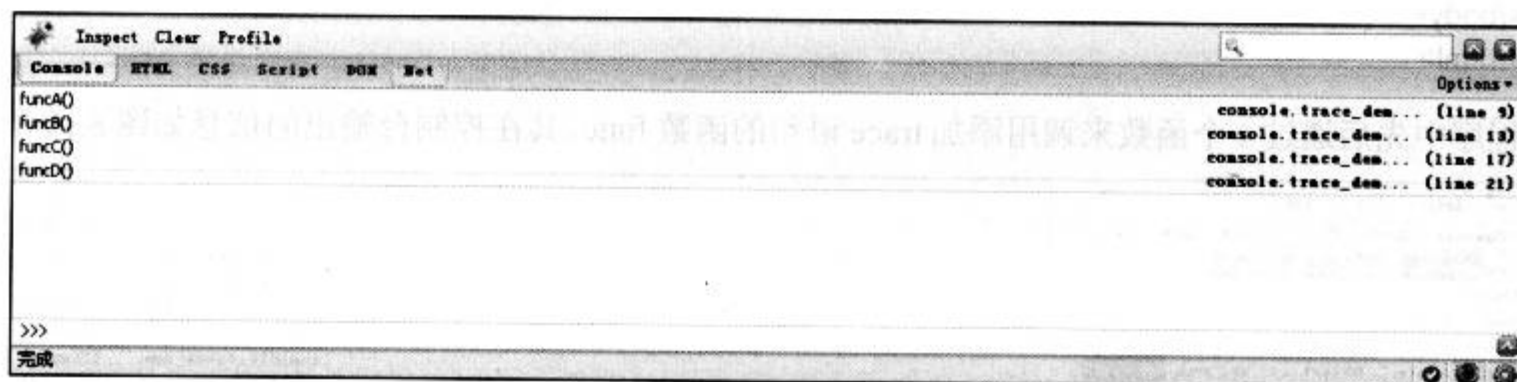


图 8.20 查看调用链

## 10. console.group 和 console.groupEnd

console.group 和 console.groupEnd 可以对输出的 log 信息进行分组。console.group 标识一个分组开始，其可以接受一个或者多个参数作为该分组的名称或者提示语，console.groupEnd 标识一个分组结束。下面的示例中，将一些 log 信息分为了两组，代码如下所示。

```
<!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>  
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  
<title>console.group demo</title>  
<script type="text/javascript">  
console.group('Group A');  
console.log('information');
```



```
console.info('information');
console.warn('information');
console.error('information');
console.groupEnd();
console.group('Group B');
console.log('information');
console.info('information');
console.warn('information');
console.error('information');
console.groupEnd();
</script>
</head>

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

分组的效果如图 8.21 所示。

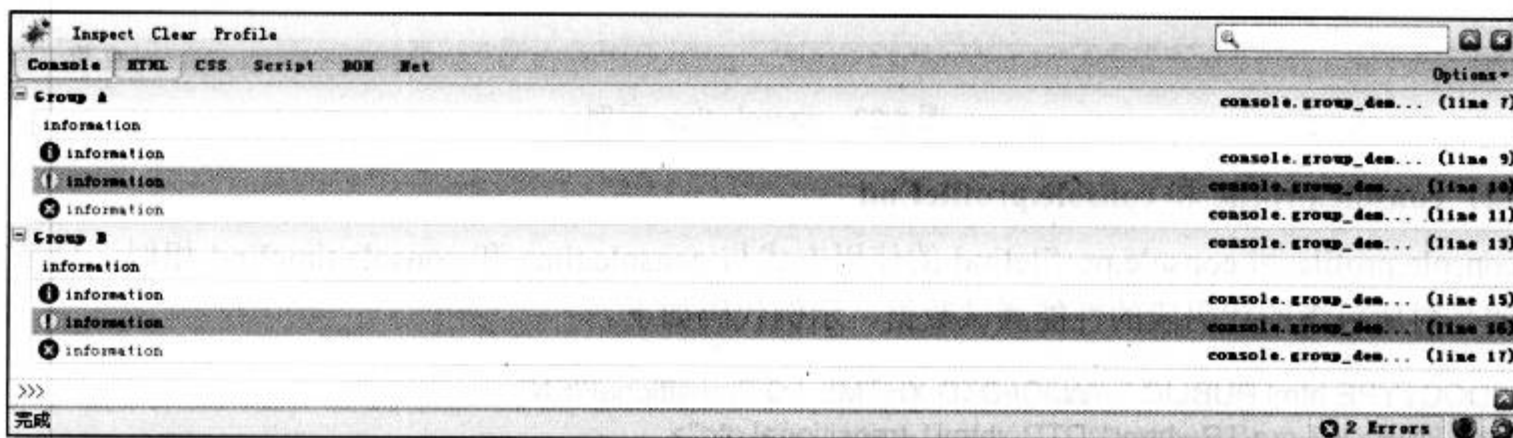


图 8.21 分组示例

## 11. console.time 和 console.timeEnd

通过将 `console.time` 和 `console.timeEnd` 添加到一段代码的开头和结尾,可以计算并输出执行这段代码所花费的时间,以毫秒为单位。`console.time` 接受一个字符串参数作为这个计算器的名字,在遇到一个以同样字符串作为参数的 `console.timeEnd` 时就会停止计时并输出时间耗费的信息。下面的示例输出执行一个循环体所花费的时间,代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>console.time demo</title>
<script type="text/javascript">
console.time('sum');
var sum = 0;
for(var i = 0 ; i < 1000; i ++ )
{
```

```
        sum += i;
    }
    console.timeEnd('sum');
</script>
</head>

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

其输出到 Firebug 控制台的信息如图 8.22 所示。

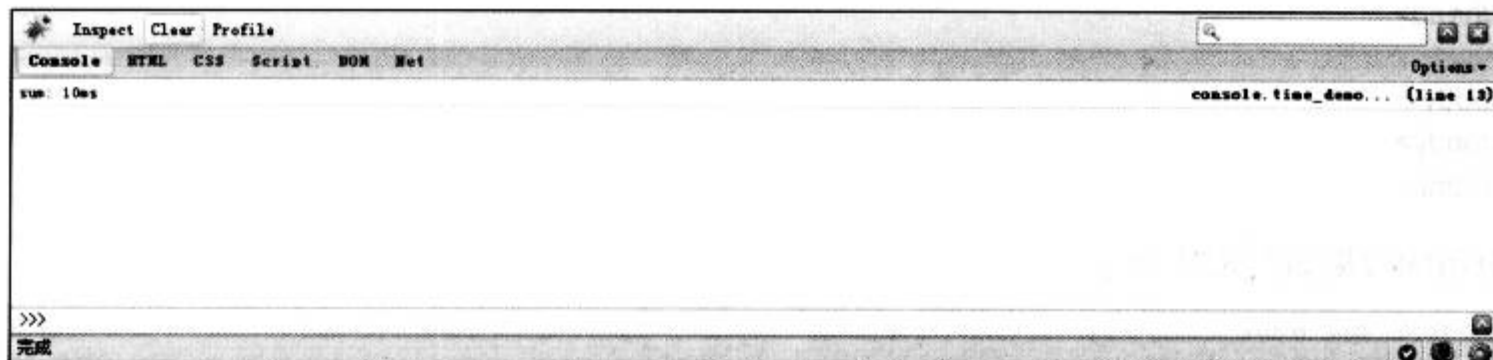


图 8.22 console.time 示例

## 12. console.profile 和 console.profileEnd

console.profile 和 console.profileEnd 的使用方式与 console.time 和 console.timeEnd 相似，只是它们输出的是对所包含的代码段的性能测试数据，示例代码如下。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>console.profile demo</title>
<script type="text/javascript">
function sum()
{
    var sum = 0;
    for(var i = 0 ; i < 10000; i++)
    {
        sum += i;
    }
    console.log(sum);
}
console.profile('sum');
sum();
sum();
sum();
sum();
console.profileEnd('sum');
</script>
```

```
</head>
```

```
<body>
```

```
</body>
```

```
</html>
```

其输出到控制台的信息如图 8.23 所示。

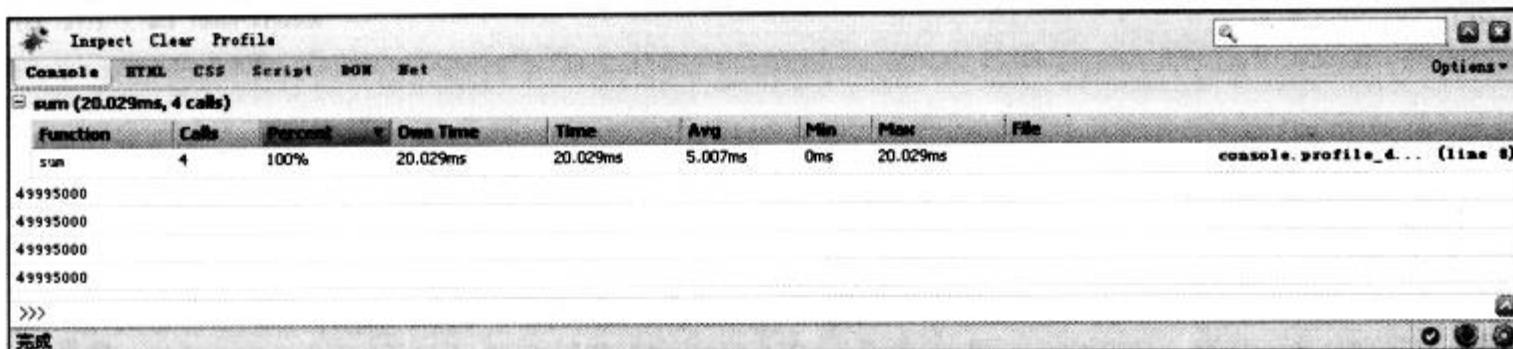


图 8.23 console.profile 示例

### 13. console.count

console.count 可以统计其自身被执行的次数，它接受一个字符串参数作为输出到控制台的信息的标题。一般将 console.count 放入到函数或者方法中来查看某个函数或者方法被调用的次数。下面的例子演示了 console.count 的用法。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>console.count demo</title>
<script type="text/javascript">
var num = 0;
function funcA()
{
    num ++;
    console.count('funcA');
    if(num % 2 == 0)
    {
        funcB();
    }
}
function funcB()
{
    console.count('funcB');
}
window.setInterval(funcA,500);
</script>
</head>

<body>
```

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

在上述程序代码中, `console.count` 被放置在了函数 `funcA` 和 `funcB` 中。`funcA` 每 500 毫秒执行一次, 每执行两次 `funcA` 执行一次 `funcB`。其输出到控制台的信息如图 8.24~图 8.26 所示。

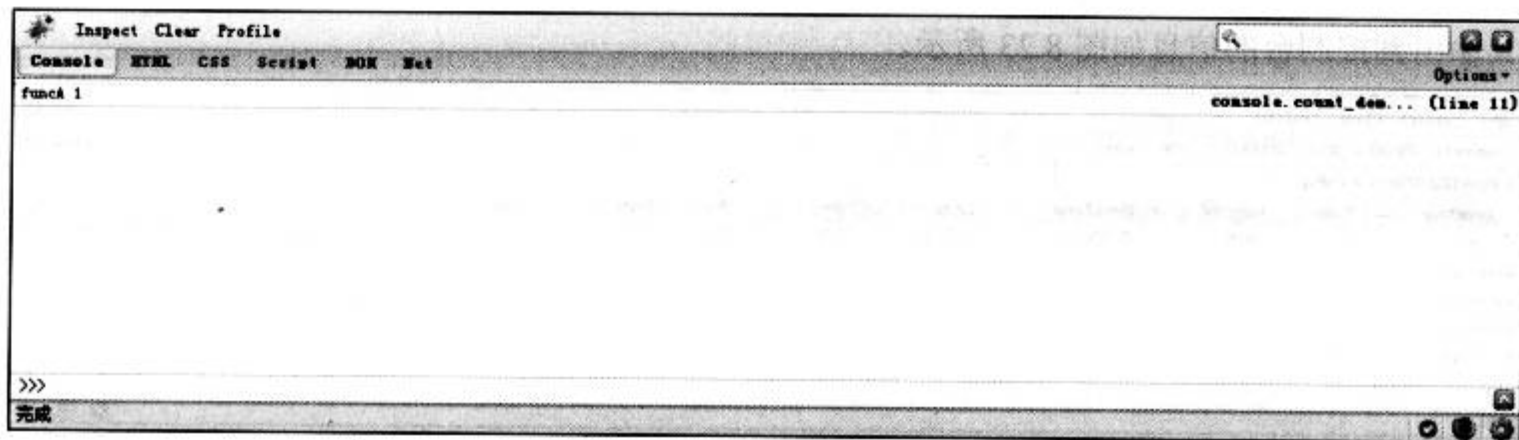


图 8.24 console.count 输出的信息 (一)

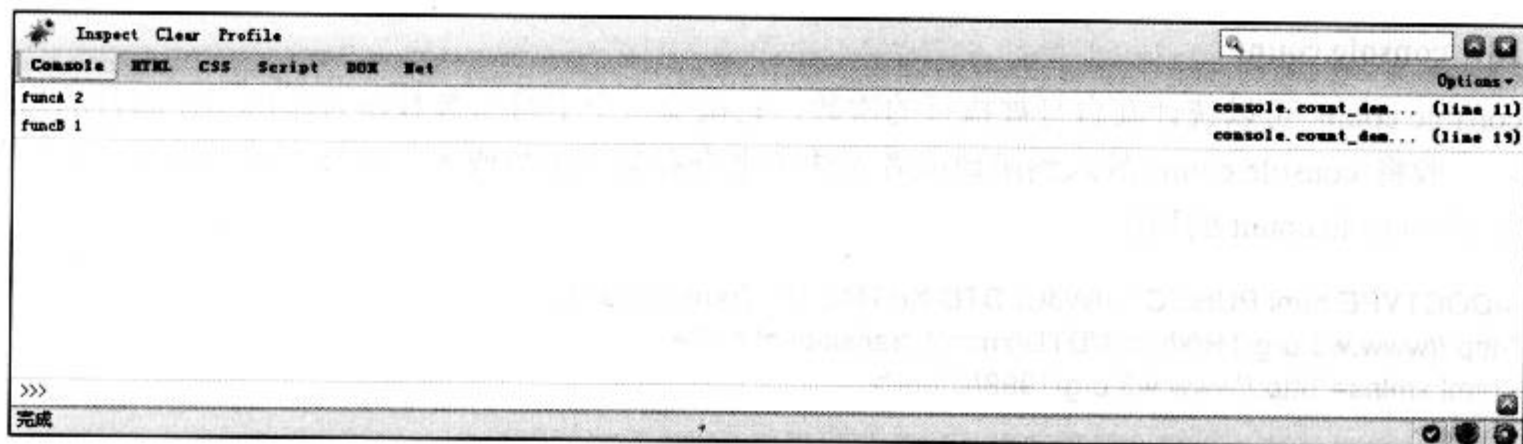


图 8.25 console.count 输出的信息 (二)

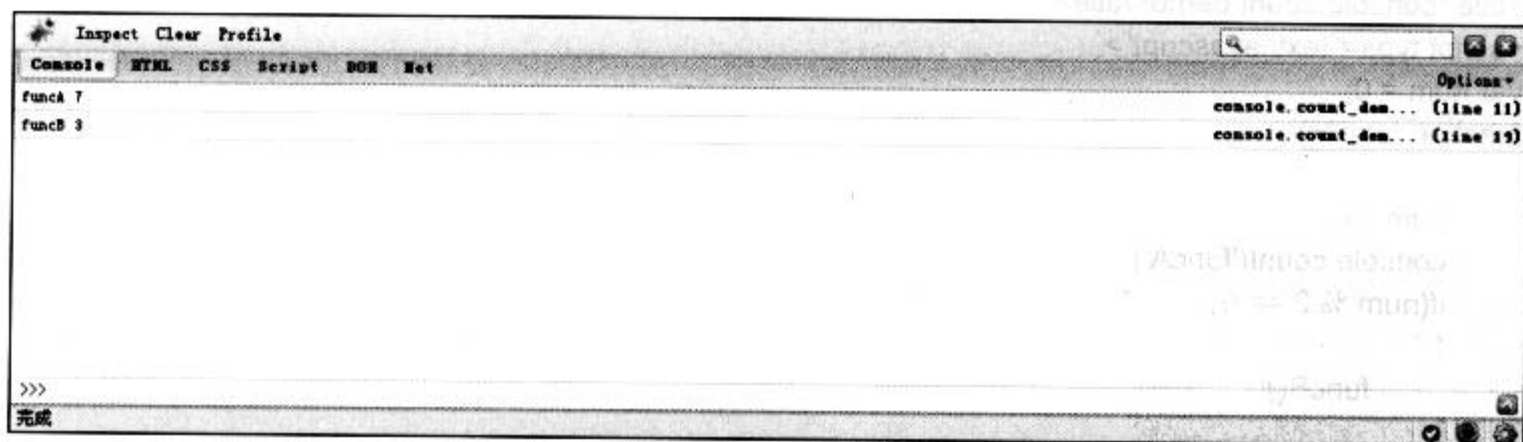


图 8.26 console.count 输出的信息 (三)

### 8.1.3 控制台的命令行功能

Firebug 的控制台还提供了命令行功能, 使得开发者可以直接通过浏览器在当前页面环境下运行 JavaScript 程序。如图 8.27 和图 8.28 所示, 直接在控制台下面的输入框中输入 JavaScript 代码, 然后按 Enter 键, 所输入的代码会显示在控制台中, 并立即执行该代码。

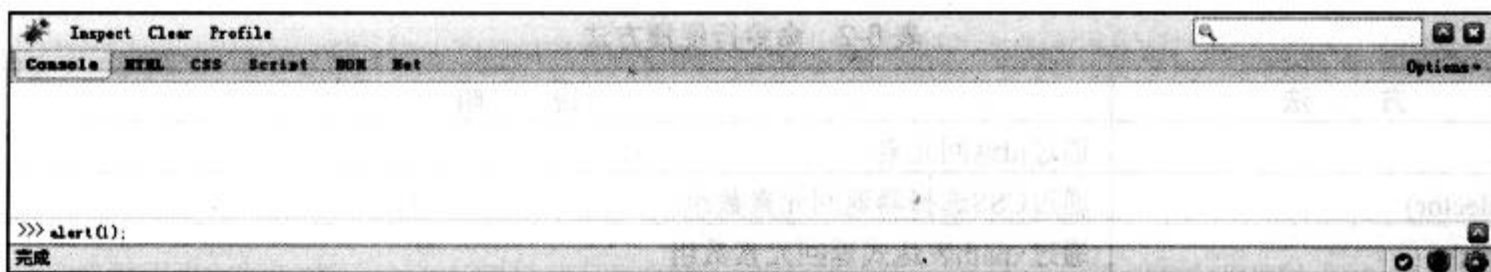


图 8.27 输入代码

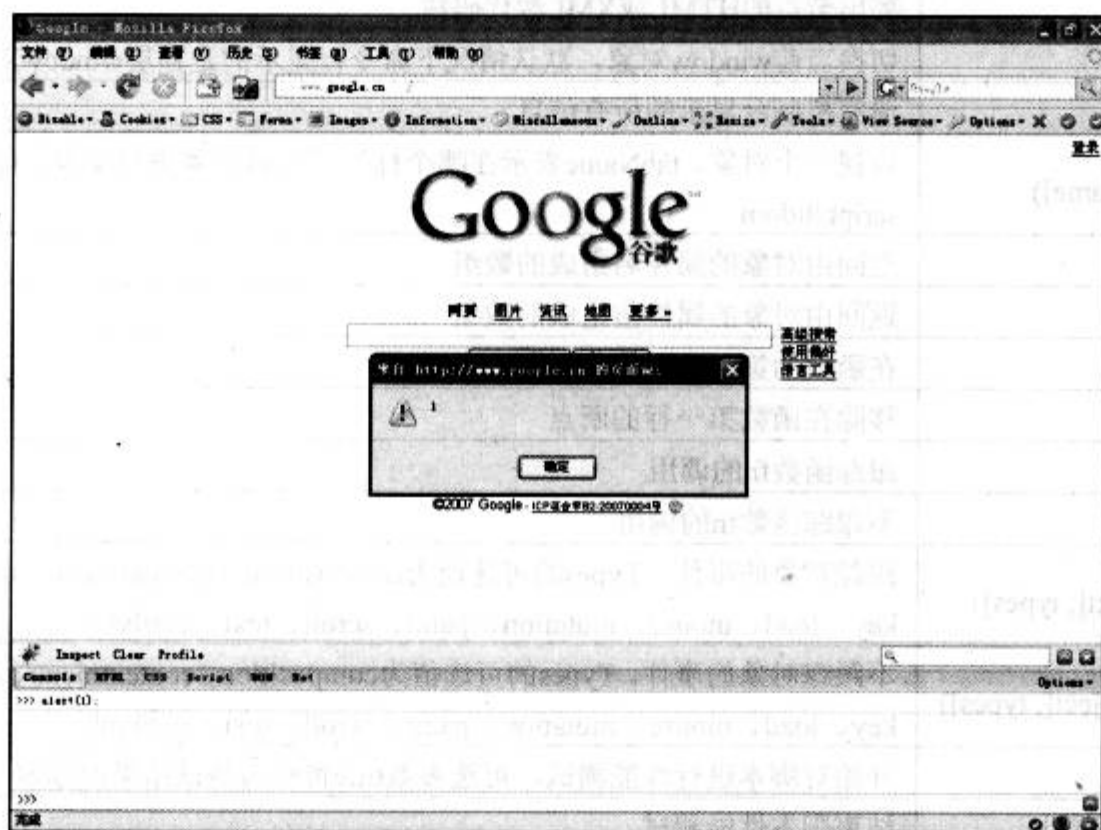


图 8.28 运行结果

单击命令行输入框右边的红色按钮可以将输入框放大。放大后的输入框会增加 run（运行）、clear（清除）和 copy（复制）3 个按钮，如图 8.29 所示。

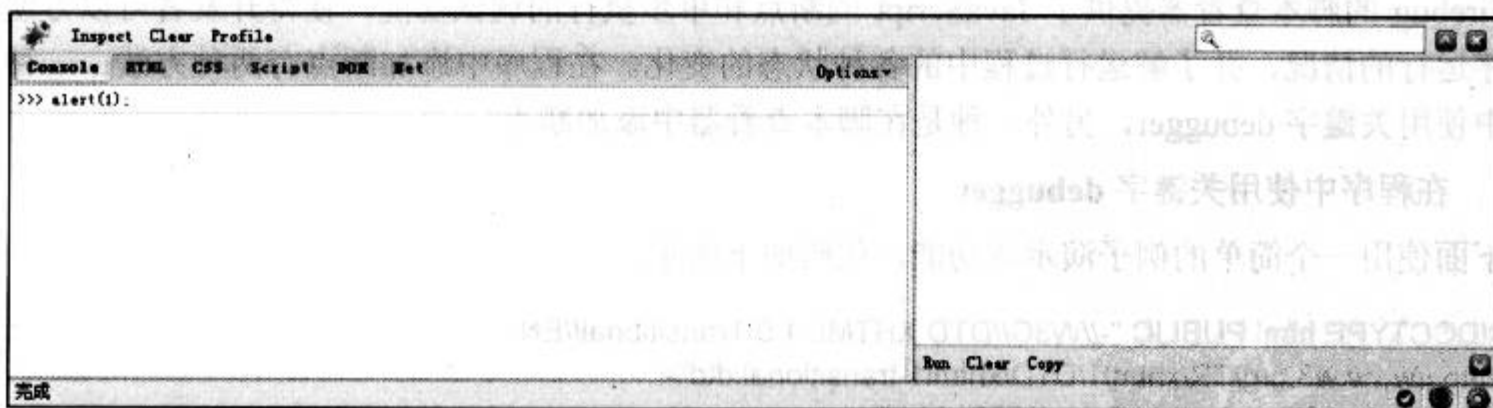


图 8.29 放大输入框

Firebug 还为命令行工具预先定义了一些便捷的方法，例如可以使用 `$(id)` 来代替 `document.getElementById(id)`，使用 `$$()` 来通过 CSS 选择器引用元素节点等。所有的这些便捷方法如表 8-2 所示。



表 8-2 命令行便捷方法

方 法	说 明
<code>\$(id)</code>	通过id返回元素
<code>\$\$(selector)</code>	通过CSS选择器返回元素数组
<code>\$x(xpath)</code>	通过xpath表达式返回元素数组
<code>dir(object)</code>	在控制台中列出对象的所有属性
<code>dirxml(node)</code>	列出节点的HTML或XML源代码树
<code>cd(window)</code>	切换当前window对象，默认情况下命令行显示的是顶层window对象
<code>clear()</code>	清空控制台显示的所有信息
<code>inspect(object[,tabName])</code>	监视一个对象。tabName表示在哪个标签页对该对象进行监视，可选值为html、css、script和dom
<code>keys(object)</code>	返回由对象的属性名组成的数组
<code>values(object)</code>	返回由对象的属性值组成的数组
<code>debug(fn)</code>	在函数的第一行增加一个断点
<code>undebug(fn)</code>	移除在函数第一行的断点
<code>monitor(fn)</code>	跟踪函数fn的调用
<code>unmonitor(fn)</code>	不跟踪函数fn的调用
<code>monitorEvents(object[, types])</code>	跟踪对象的事件。Types的可选值为composition、contextmenu、drag、focus、form、key、load、mouse、mutation、paint、scroll、text、ui和xul
<code>unmonitorEvents(object[, types])</code>	不跟踪对象的事件。Types的可选值为composition、contextmenu、drag、focus、form、key、load、mouse、mutation、paint、scroll、text、ui和xul
<code>profile([title])</code>	开始对脚本进行性能测试，可选参数title将作为测试结果的标题
<code>profileEnd()</code>	结束脚本性能测试

### 8.1.4 断点、单步执行和变量信息

Firebug 的脚本查看器提供了 JavaScript 的断点和单步执行的调试功能，使得开发者可以方便地跟踪程序运行的情况，并了解运行过程中的各种状态的变化。在程序中添加断点有两种方法：一种是在程序中使用关键字 `debugger`，另外一种是在脚本查看器中添加断点。

#### 1. 在程序中使用关键字 `debugger`

下面使用一个简单的例子演示该功能，代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>debugger demo</title>
<script type="text/javascript">
function debug()
{
    debugger;
```

```
    var a = 1;
    var b = add(a,1);
    alert(b);
}
function add(x,y)
{
    var sum = x + y;
    return sum;
}
</script>
</head>

<body>
<input type="button" value="test button" onclick="debug();" />
</body>
</html>
```

(1) 在上述程序代码中将函数 `debug` 注册成为测试按钮的 `click` 事件处理函数，并在函数的第一行使用了关键字 `debugger`。用 Firefox 打开示例页面，并单击 `test button` 按钮，发现并没有弹出对话框，而是停留在 `debugger` 这一行，如图 8.30 所示。

(2) 脚本查看器的脚本代码视图中用黄色的三角形图标标识了当前停留的行，右边的 Watch 窗口显示了当前函数作用域内的所有变量信息。由于变量的定义是存在于整个函数体内的，而初始化则要等到程序执行到初始化变量的代码，所以这时变量 `a` 和变量 `b` 的值都是 `undefined`。Watch 窗口上面部分有 4 个控制按钮，如图 8.31 所示。

段 •

图 8.30 断点



图 8.31 控制按钮

(3) 控制按钮的第一个是 `Continue` 按钮，单击该按钮可以使中断的程序恢复正常运行。第二个和第三个分别是 `Step Over` 按钮和 `Step Into` 按钮。这两个按钮都提供了单步执行的功能，所不同的是当遇到调用其他函数或者方法时，`Step Into` 会跳入该函数或者方法内而 `Step Over` 则不会。最后一个是 `Step Out` 按钮，单击该按钮会跳出当前的函数或者方法。

(4) 使用 `Step Over` 或者 `Step Into` 按钮使代码单步执行到第 11 行：`var b = add(a,1)`。这时变量 `a` 已经被初始化，在右侧 Watch 窗口中可以看到 `a` 的值已经被更新，如图 8.32 所示。

(5) 由于这一行代码调用了另外一个函数 `add`，所以如果单击 `Step Into` 按钮，则会跳入 `add` 函数体，如图 8.33 所示。

(6) 如果使用 `Step Over` 按钮，则不会跳入函数 `add` 内，如图 8.34 所示。

(7) 在使用 `Step Into` 按钮跳入 `add` 函数后，如果使用 `Step Out` 按钮，则会跳出 `add` 函数返回 `debug` 函数，如图 8.35 所示。

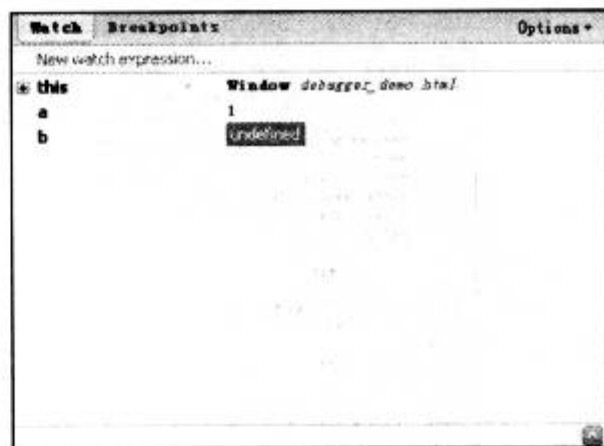


图 8.32 查看变量信息

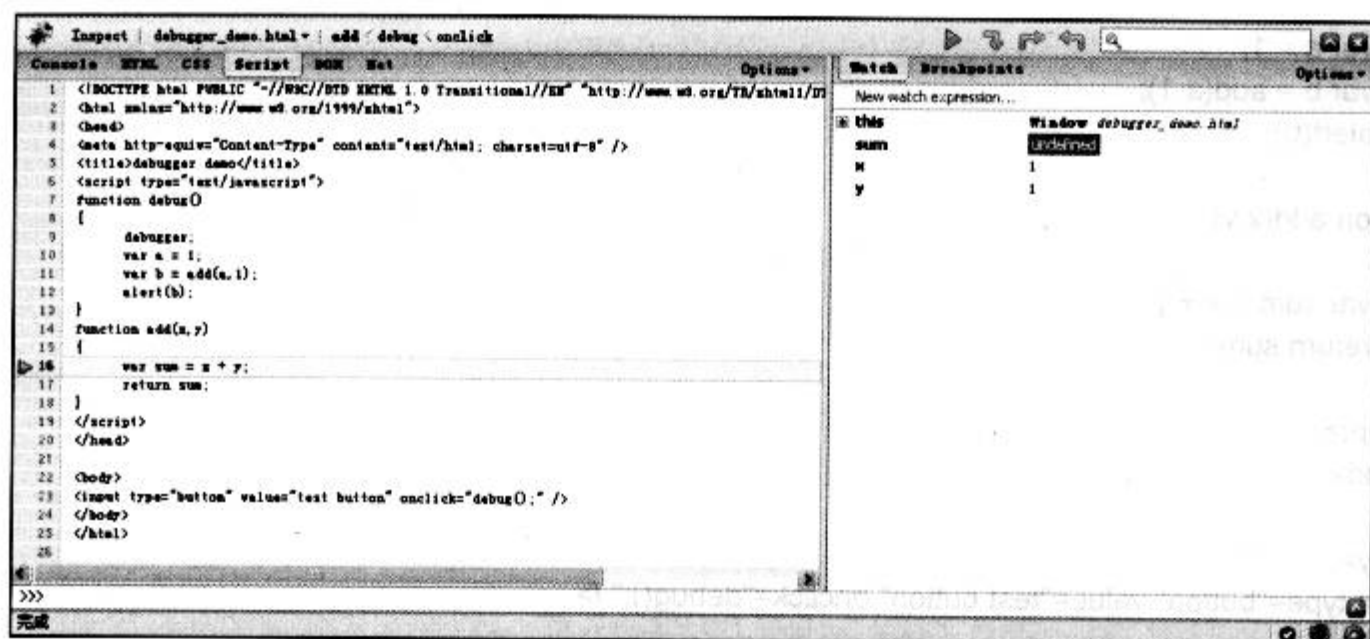


图 8.33 Step Into

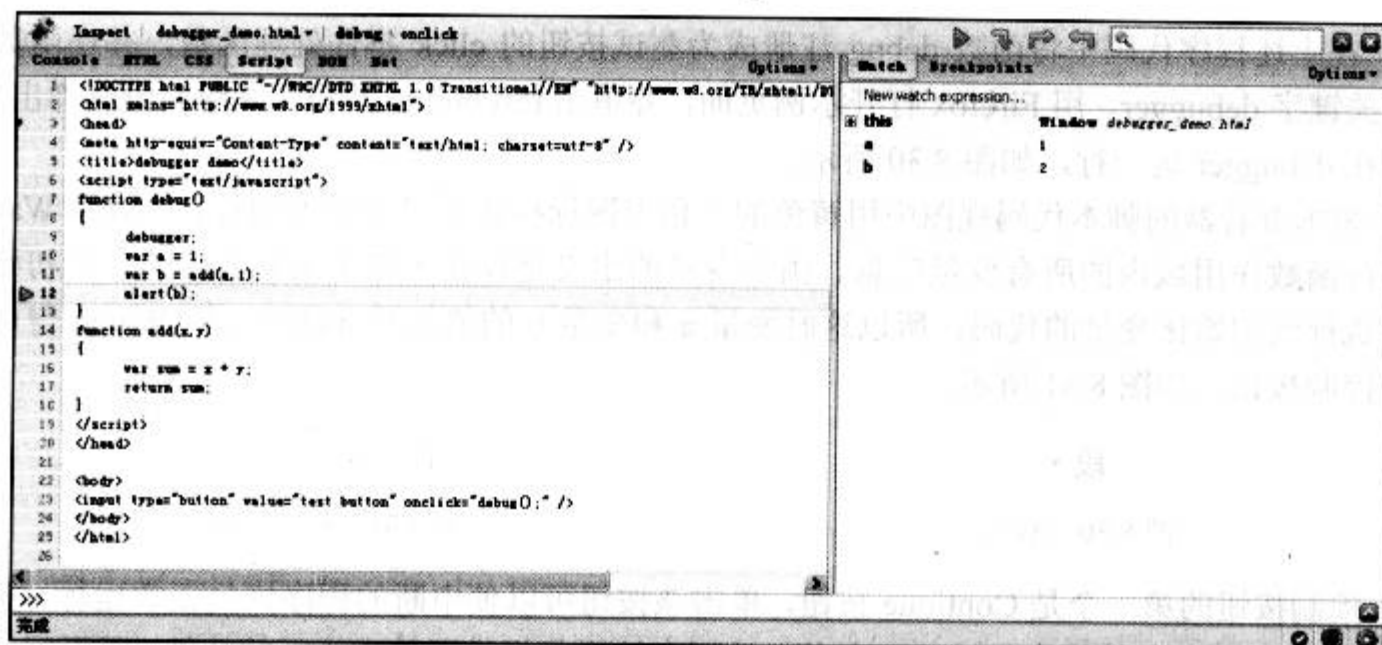


图 8.34 Step Over

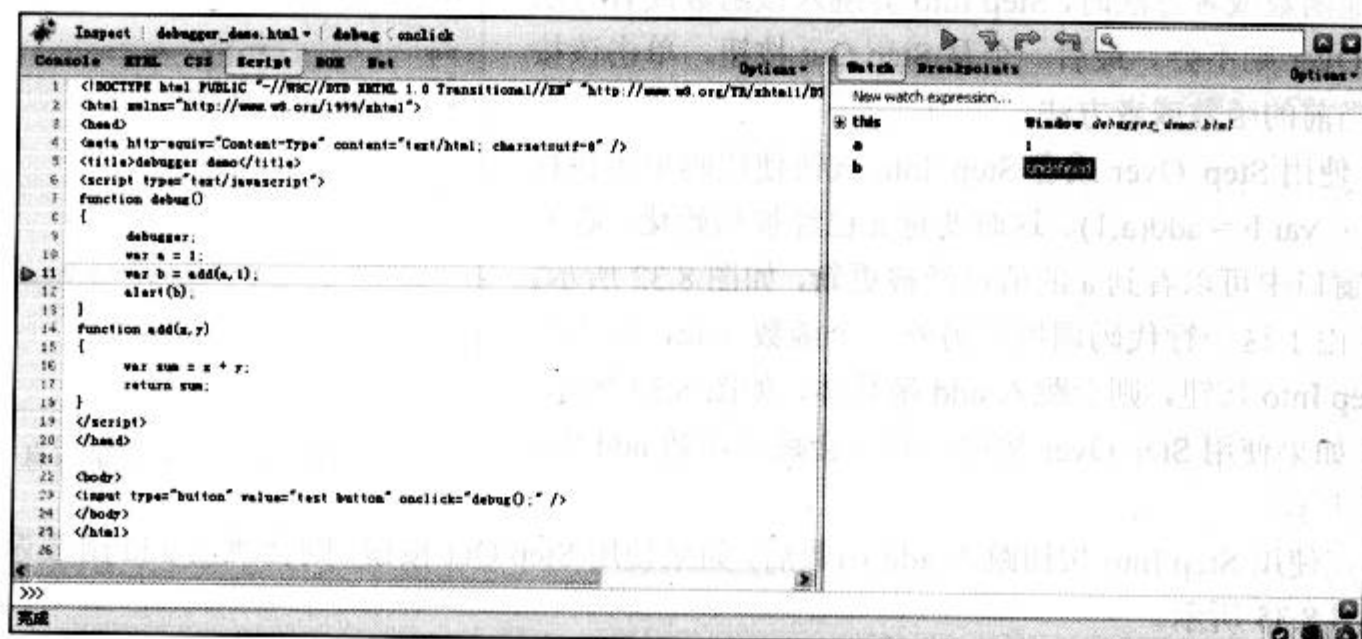


图 8.35 Step Out

## 2. 使用脚本查看器添加断点

使用脚本查看器添加断点非常简单,只需要在代码所在行的行号前面单击即可。在第12行添加一个断点,如图8.36所示。

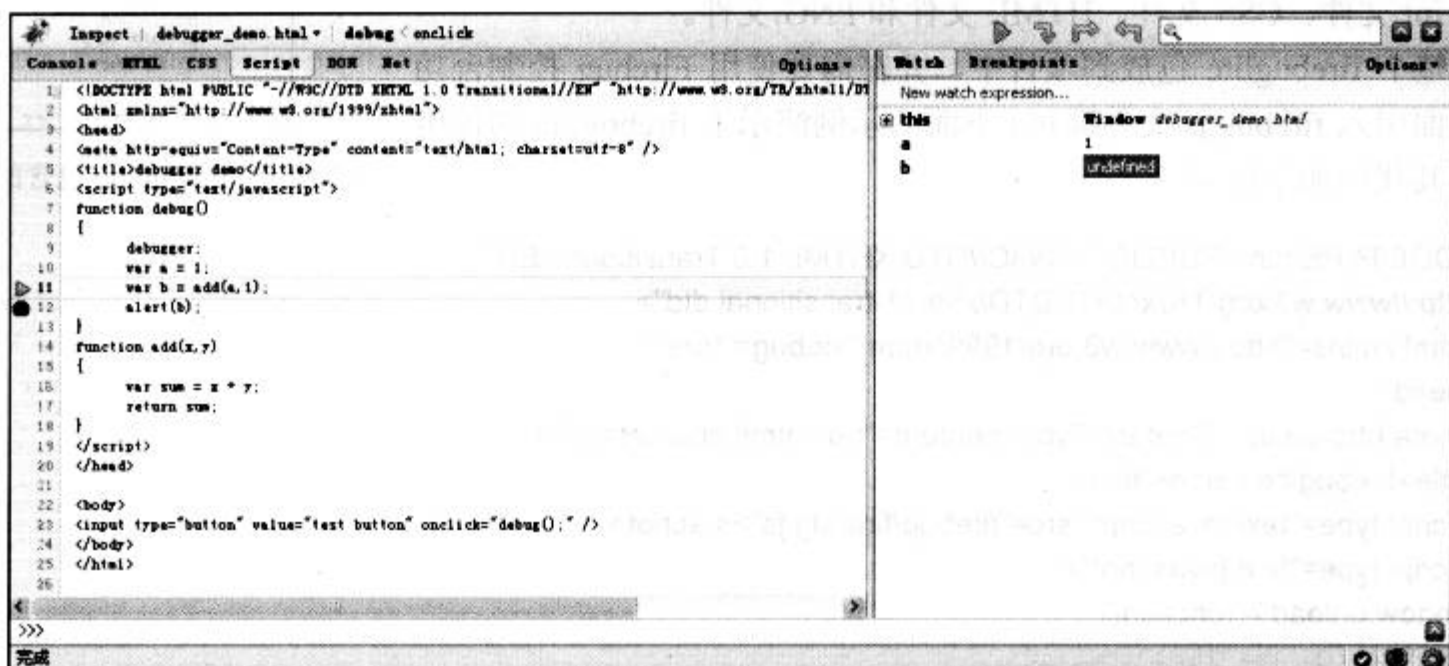


图 8.36 添加断点

单击 Continue 按钮使程序继续运行,程序在第12行被中断,如图8.37所示。

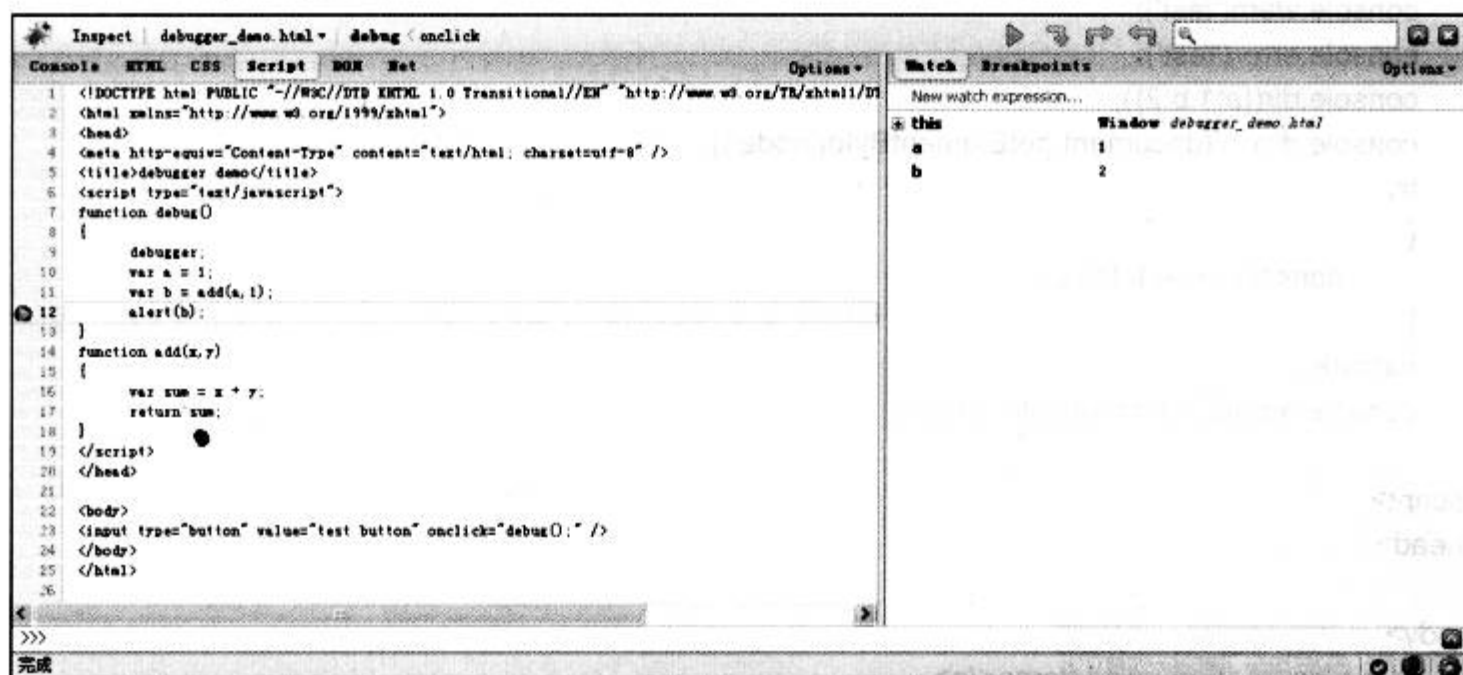


图 8.37 程序再次中断

灵活运用断点、单步执行和 Watch 窗口可以方便地对程序进行深入的跟踪调试,读者可以多进行一些尝试以便熟练掌握这些功能的使用。

### 8.1.5 在其他浏览器中使用 Firebug 的控制台

虽然 Firebug 插件只能在 Firefox 中使用,但是 Firebug 官方提供了一个 JavaScript 开发的组件,将



其包含到程序页面中就可以在其他浏览器环境中使用 Firebug 的控制台功能,官方将其称为 Firebuglite。Firebuglite 可以在 <http://www.getfirebug.com/Firebuglite/lite.html> 下载。其包含的文件如图 8.38 所示,主要有 JavaScript 文件、CSS 文件、HTML 文件和 PNG 文件。

将整个 firebuglite 包放到项目中,并在需要使用 Firebug 控制台功能的页面引入 firebug.js 文件即可。下面的示例演示了 firebuglite 的使用方法,其代码如下所示。



图 8.38 firebuglite 包含的文件

```
<!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" debug="true">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>firebuglite demo</title>
<script type="text/javascript" src="firebug/firebug.js"></script>
<script type="text/javascript">
window.onload = function()
{
    console.group('firebuglite group');
    console.log('test');
    console.info('test');
    console.warn('test');
    console.error('test');
    console.dir({a:1,b:2});
    console.dirxml(document.getElementById('node'));
    try
    {
        console.assert(1==2);
    }
    catch(e){}
    console.groupEnd('firebuglite group');
}
</script>
</head>

<body>
    <p id="node">firebuglite demo</p>
</body>
</html>
```

使用 IE 或者其他非 Firefox 浏览器访问该页面,显示效果如图 8.39 所示。

注意: 示例代码中 HTML 节点内的属性 debug="true"表示模式显示控制台面板。如果不添加该属性,则需要在页面打开后按 F12 键打开面板。



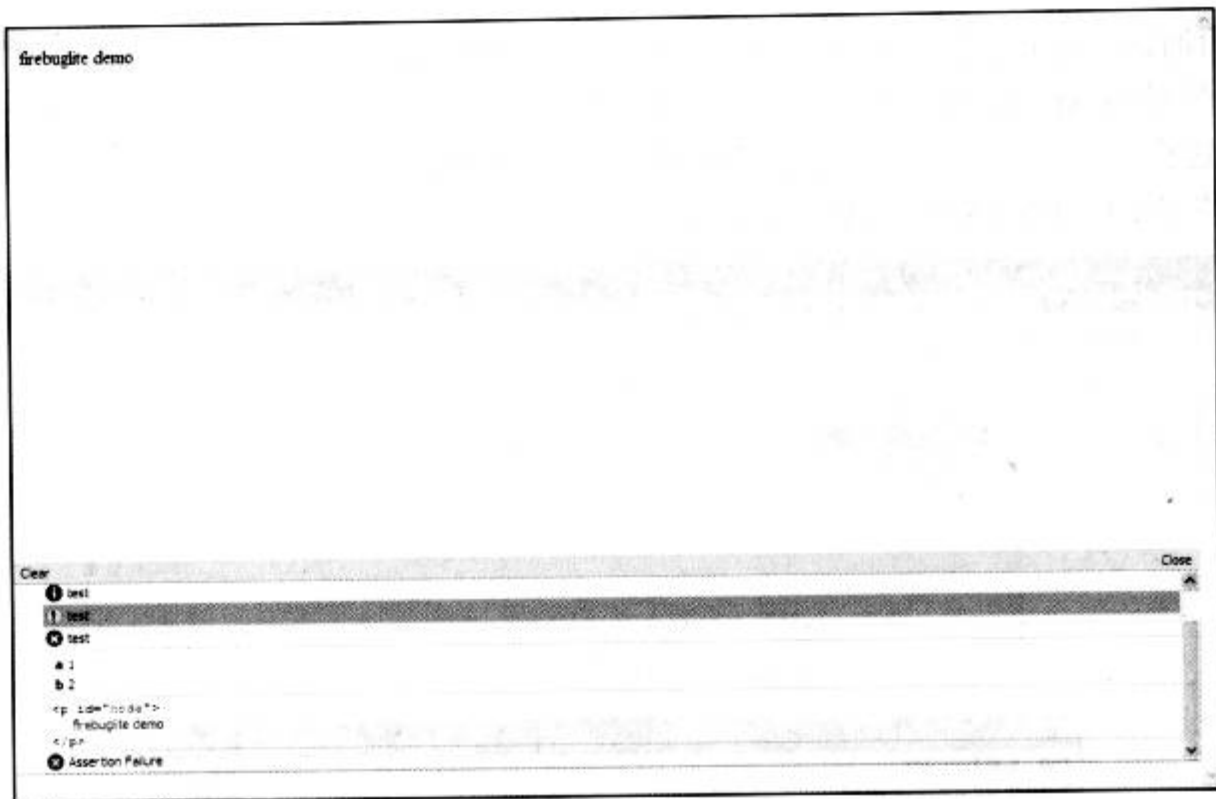


图 8.39 使用 IE 或者其他非 Firefox 浏览器访问页面的显示效果

### 8.1.6 屏蔽测试代码

在发布的项目中，不再需要显示各种各样的调试信息，除了将测试代码删除之外，还可以在页面头部加入下面的 JavaScript 代码以屏蔽测试代码的信息输出，代码如下所示。

```
var names = ["log", "debug", "info", "warn", "error", "assert", "dir", "dirxml",  
            "group", "groupEnd", "time", "timeEnd", "count", "trace", "profile", "profileEnd"];  
window.console = {};  
for (var i = 0; i < names.length; ++i)  
    window.console[names[i]] = function() {}
```

其原理是覆盖 console 对象的各种方法，将它们设置为空函数，这样测试代码就不会有任何行为发生。

## 8.2 使用 Aptana 的集成调试功能

Aptana 也在其 IDE 内集成了借助于浏览器实现的调试功能。普通版的 Aptana 是利用 Firefox 进行调试，而专业版的 Aptana 则提供了利用 IE 进行调试的功能。Aptana 的脚本调试功能与 Firebug 很相似，所不同的是其调试窗口是集成在 Aptana 内的，这样就给开发者提供了一定的便利。

### 8.2.1 配置集成调试环境

使用 Aptana 的调试功能需要在 Firefox 上安装一个叫做 Aptana Debugger Service 的插件，该插件在

启动 Firefox 的调试功能时会自动安装, 安装完成后会在 Firefox 浏览器右下角添加一个类似齿轮形状的 Aptana 的小图标, 如图 8.40 所示。



图 8.40 小图标

然后需要进行一些基本配置。首先通过快捷操作按钮中调试按钮的下拉列表菜单打开“调试”窗口, 如图 8.41 和图 8.42 所示。

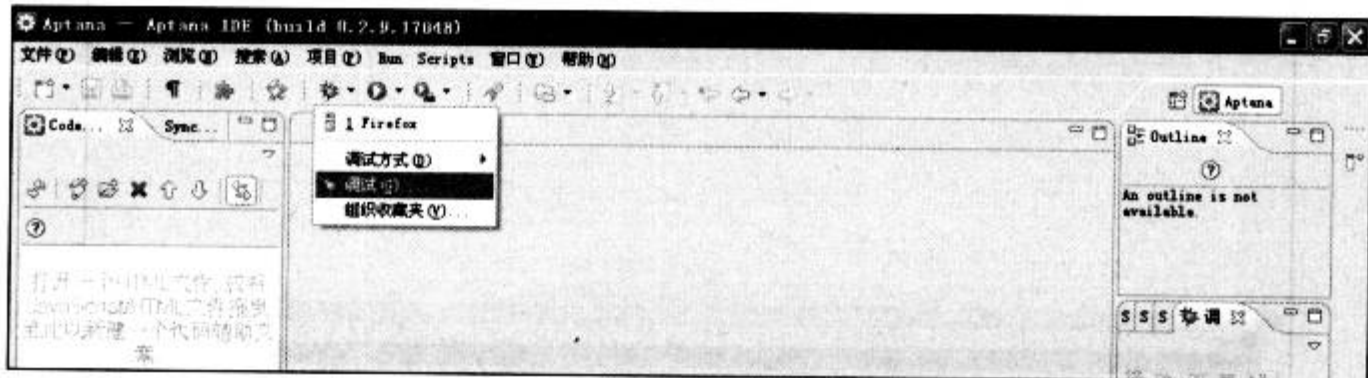


图 8.41 “调试”窗口 (一)

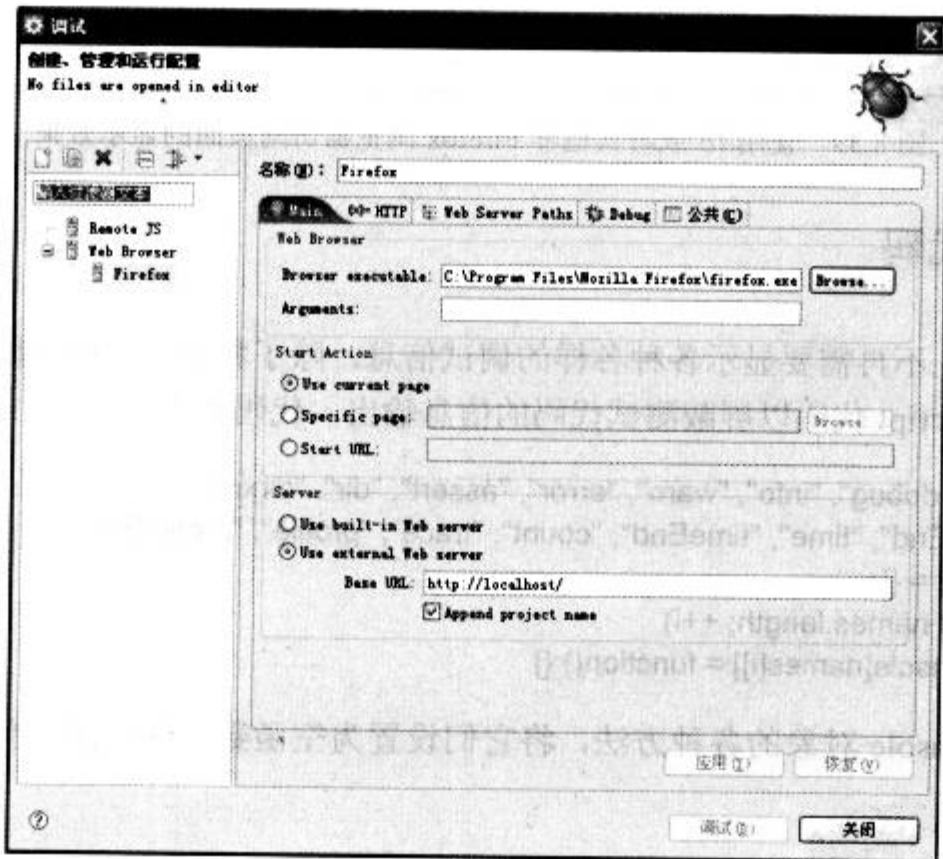


图 8.42 “调试”窗口 (二)

展开 Web Browser 节, 选择 Firefox 选项。首先在 Main 选项卡下面的第一项 Web Browser 下面的 Browser executable 进行设置, 将其设置为本地 firefox.exe 所在路径。然后将第三项 Server 选项切换到 Use external Wb server。Base URL 填写本地 Web 服务器的 URL 地址, 然后选中 Append project name (附加项目名称) 复选框。单击“应用”按钮并关闭。到此设置结束。

## 8.2.2 启动调试

配置好调试选项后, 就可以启动调试功能了。单击快捷操作按钮中的“调试”按钮, Aptana 会打

开一个新的 Firefox 进程来访问当前编辑的页面，并且将自身状态切换到调试视图。当第一次使用调试功能时，Aptana 会提示当前使用的功能关联到调试视图，需要将调试视图打开，询问用户是否打开该视图，如图 8.43 所示。

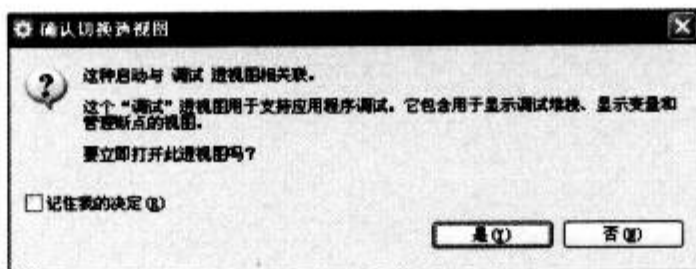


图 8.43 打开调试视图

选中“记住我的决定”复选框并单击“是”按钮，调试视图被打开，如图 8.44 所示。

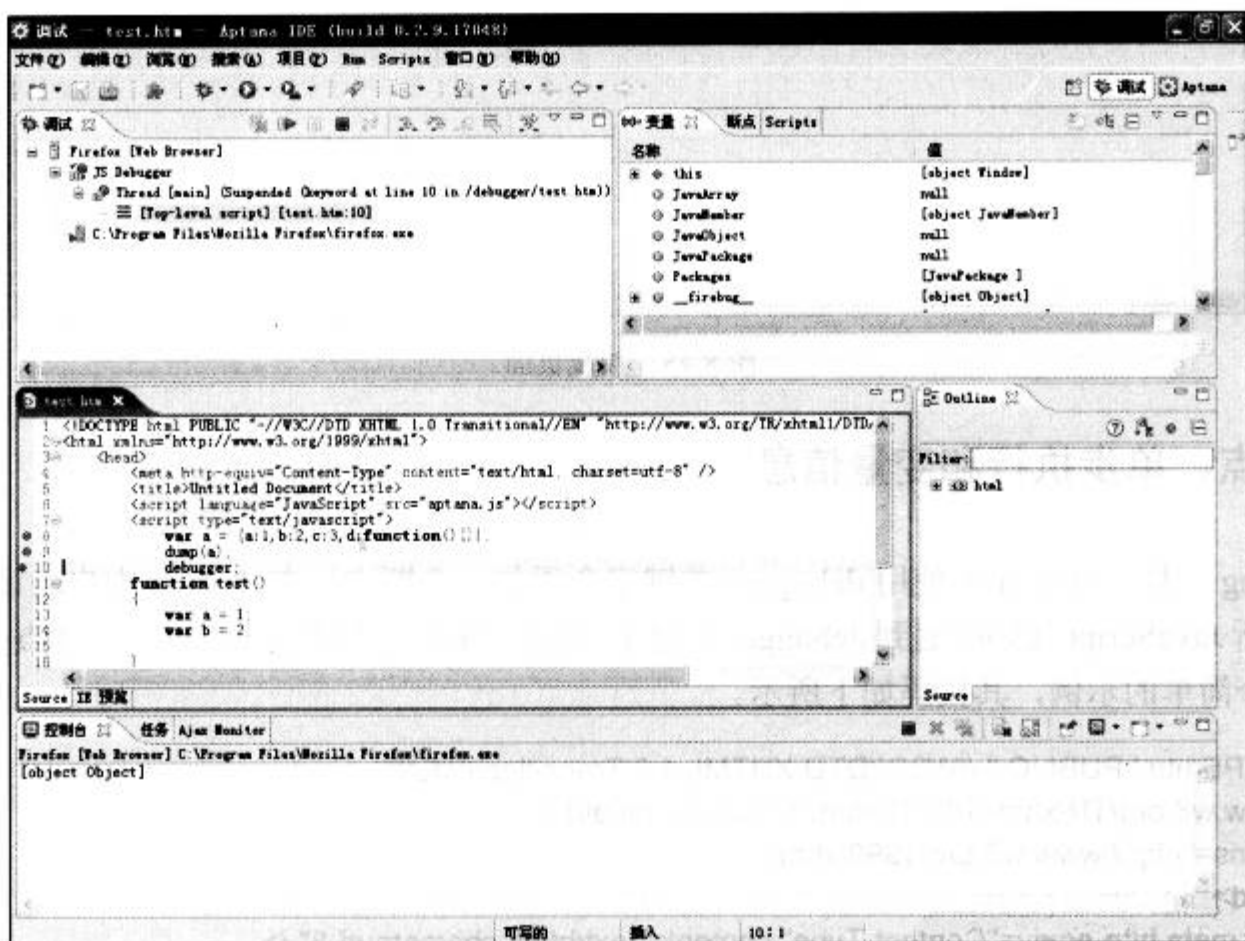


图 8.44 调试视图

可以看到，界面右上角的视图切换按钮新增加了一个调试按钮。主工作区域的位置下移了一部分，上面新增了调试区域。调试区域左边区域是控制区，显示了当前调试的进程信息，以及提供了一些控制按钮。右边区域则是变量信息查看面板，其作用与 Firebug 中的 Watch 窗口一样。

Aptana 切换到调试视图的同时，一个新的 Firefox 进程会被打开。在当前页面的 Firefox 中显示出来前，Firefox 显示的是 Aptana 的加载页面，如图 8.45 所示。

可以看到，这时 Firefox 右下角表示 Aptana Debugger Service 的小图标由灰色变成了亮色。



图 8.45 加载页面

### 8.2.3 断点、单步执行和变量信息

同 Firebug 一样，在程序中使用 `debugger` 关键字会添加一个断点。在 Aptana 中启动调试后，如果 Firefox 在解析 JavaScript 代码时遇到 `debugger` 关键字，则会中断程序的执行，并将窗口焦点交给 Aptana。下面来看一个简单的示例，其代码如下所示。

```
<!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>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>debugger demo</title>
    <script type="text/javascript">
      var a = 1;
      debugger;
      var b = 2;
      var c = add(a,b);
      alert(c);
      function add(x,y)
      {
        var z = x + y;
        return z;
      }
    </script>
```

```
</head>

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

使用 Aptana 启动调试, Aptana 打开一个新的 Firefox 进程浏览被调试的页面, 在 Firefox 解析页面中的 JavaScript 脚本时, 遇到 `debugger` 关键字, Aptana Debug Service 会中断页面程序的执行, 并切换到 Aptana, 同时定位到页面源代码中 `debugger` 这一行, 如图 8.46 所示。

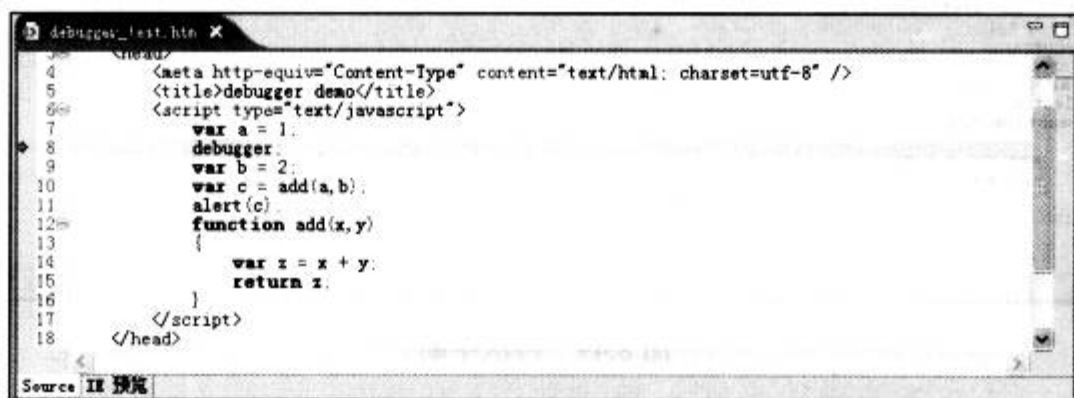


图 8.46 debugger 关键字

除了使用 `debugger` 关键字添加断点外, 还可以直接在 Aptana 中添加断点。在页面源代码的某行前面的灰色区域中双击鼠标左键, 或者右击该区域在弹出的快捷菜单中选择“切换断点”命令, 就可以为该行添加一个断点, 如图 8.47 所示。

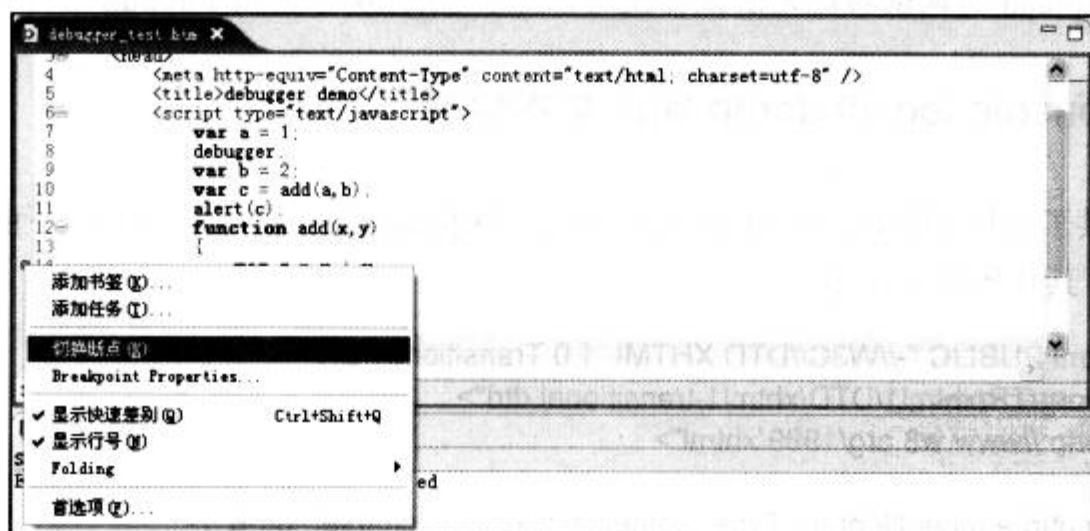


图 8.47 切换断点

同 Firebug 一样, Aptana 也提供了单步执行和查看变量信息的功能。第 14 行, 即 `add` 函数的第一行添加一个断点, 再次启动调试, 程序首先会在 `debugger` 关键字所在的行中断, 这时变量查看器中的内容如图 8.48 所示。

单击调试面板中的“继续”按钮, 程序运行到所设置的断点处再次被中断, 如图 8.49 所示。

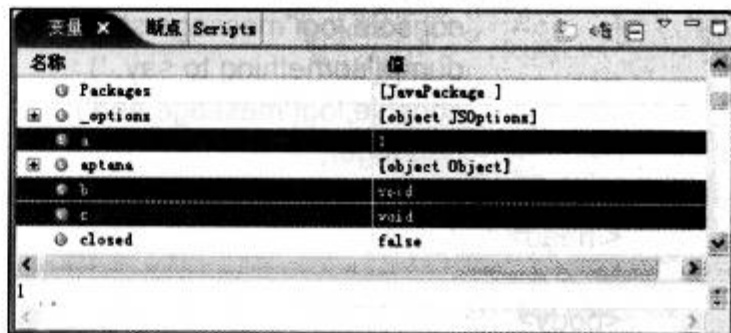


图 8.48 查看变量信息



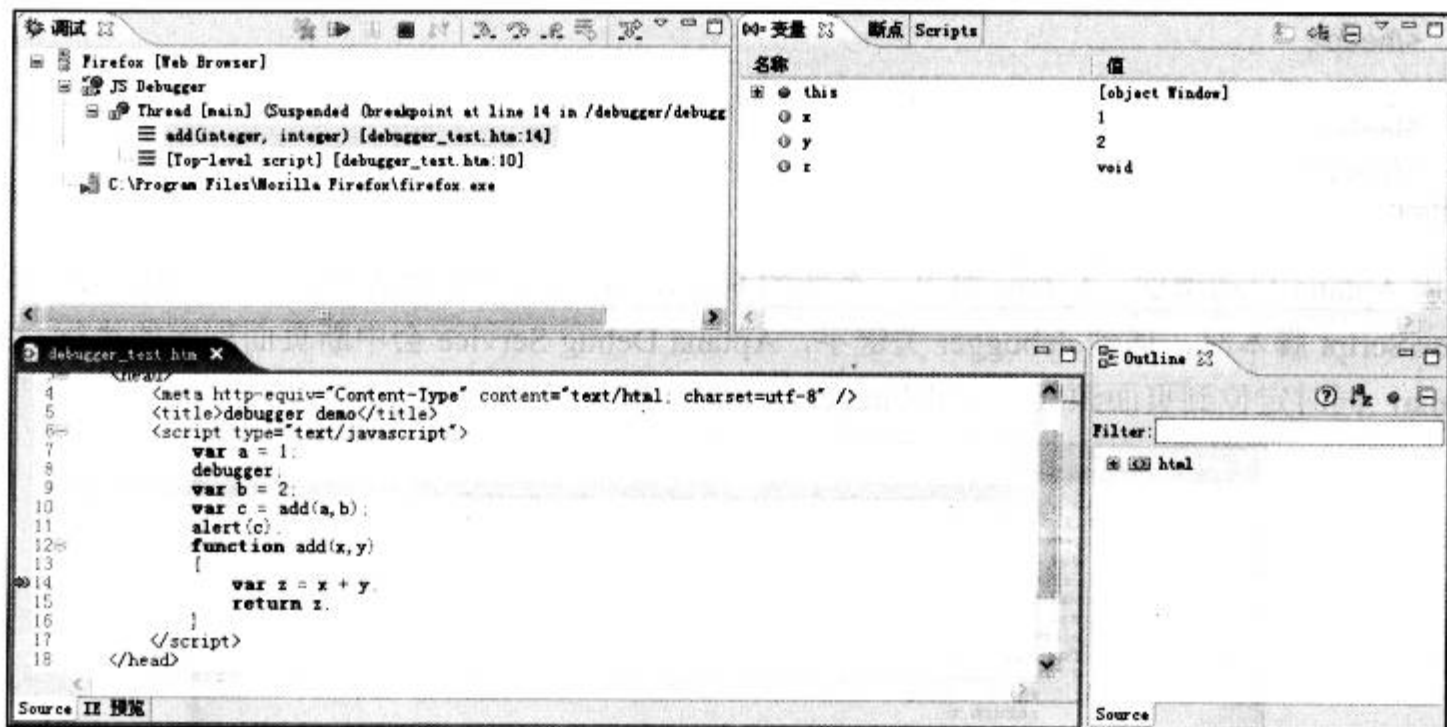


图 8.49 再次中断

调试面板中提供了 3 个分别与 Firebug 脚本查看器中的 Step Into(单步跳入)、Step Over(单步跳过)和 Step Out(单步返回)按钮功能一样的按钮,如图 8.50 所示。



图 8.50 调试按钮

这些按钮的用法参考前面介绍的 Firebug 的脚本调试器中对应按钮的用法,这里不再赘述。

注意:如果发现 Aptana 的调试功能无法正常使用,可在 Firefox 中禁用 Firebug 后再进行尝试。

## 8.2.4 使用 console.log 和 dump 输出文本信息

Aptana 也提供了向控制台输出信息的方法,就是使用 console.log 或者 dump。下面的示例演示了这两个方法的使用,其代码如下所示。

```
<!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>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>debugger log demo</title>
    <script type="text/javascript">
      console.log('message start');
      dump('something to say..')
      console.log('message end');
      debugger;
    </script>
  </head>

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

启动调试, 程序在 debugger 关键字处中断并返回 Aptana, 这时控制台中输出的信息如图 8.51 所示。

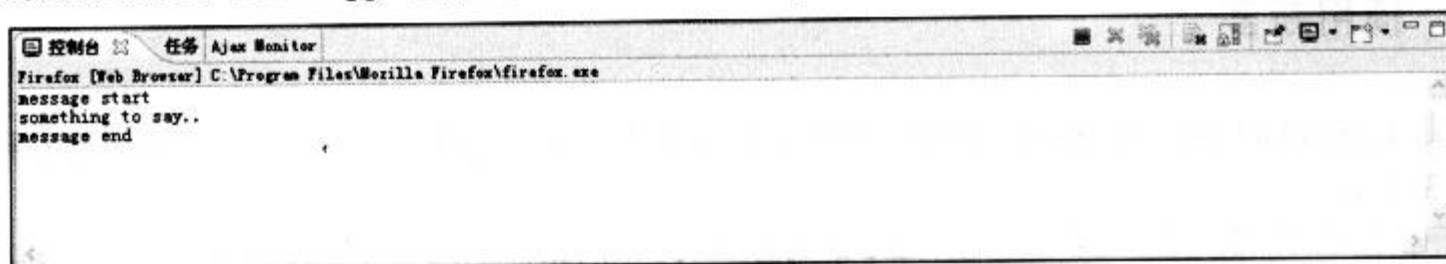


图 8.51 console.log 和 dump

### 8.2.5 使用 aptana.trace 输出调用堆栈信息

同 Firebug 的 console.trace 一样, Aptana 也提供了输出调用堆栈信息到控制台的方法, 就是使用 aptana.trace, 该方法可以接受一个字符串参数作为输出信息的标题。下面来看一个简单的示例, 代码如下所示。

```
<!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>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>aptana.trace demo</title>
    <script type="text/javascript">
      function funcA()
      {
        aptana.trace('funcA');
        debugger;
      }
      function funcB()
      {
        funcA();
      }
      funcB();
    </script>
  </head>

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

程序最终输出到 Aptana 控制台的信息如图 8.52 所示。

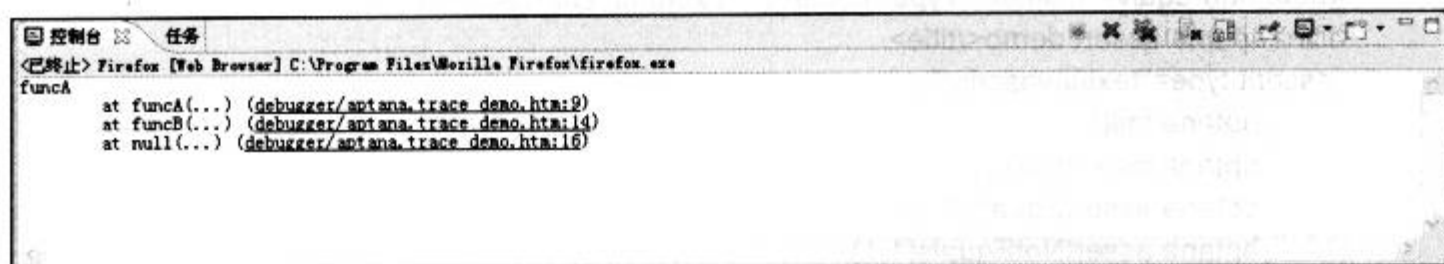


图 8.52 aptana.trace 示例

## 8.2.6 使用断言

比起 Firebug 的 `console.assert`, Aptana 提供了更为强大的断言功能, 其通过一组 Assert API 来实现, 如表 8-3 所示。

表 8-3 Aptana Assert API

方 法	说 明
<code>aptana.fail()</code>	总是返回断言失败
<code>aptana.assert(obj)</code>	断言obj为真或者有意义
<code>aptana.assertEquals(arg1,arg2)</code>	断言两个参数相等
<code>aptana.assertNotEquals(arg1,arg2)</code>	断言两个参数不等
<code>aptana.assertGreater(arg1,arg2)</code>	断言第一个参数比第二个参数大
<code>aptana.assertNotGreater(arg1,arg2)</code>	断言第一个参数不比第二个参数大
<code>aptana.assertLess(arg1,arg2)</code>	断言第一个参数比第二个参数小
<code>aptana.assertNotLess(arg1,arg2)</code>	断言第一个参数不比第二个参数小
<code>aptana.assertContains(arg1,arg2)</code>	断言第一个参数被第二个参数包含
<code>aptana.assertNotContains(arg1,arg2)</code>	断言第一个参数不被第二个参数包含
<code>aptana.assertTrue(expression)</code>	断言表达式为真
<code>aptana.assertFalse(expression)</code>	断言表达式为假
<code>aptana.assertNull(obj)</code>	断言对象为Null
<code>aptana.assertNotNull(obj)</code>	断言对象不为Null
<code>aptana.assertUndefined(var)</code>	断言变量未定义
<code>aptana.assertNotUndefined(var)</code>	断言变量已定义
<code>aptana.assertInstanceOf(arg1,arg2)</code>	断言第一个参数为第二个参数的一个实例
<code>aptana.assertNotInstanceOf(arg1,arg2)</code>	断言第一个参数不是第二个参数的一个实例
<code>aptana.assertTypeOf(obj,type)</code>	断言obj的类型是type
<code>aptana.assertNotTypeOf(obj,type)</code>	断言obj的类型不是type

下面的示例演示了这些方法的用法, 示例代码如下所示。

```
<!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>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>aptana.assert demo</title>
    <script type="text/javascript">
      aptana.fail();
      aptana.assert(null);
      aptana.assertEquals(1,2);
      aptana.assertNotEquals(1,1);
      aptana.assertGreater(1,2);
```

```

    aptana.assertNotGreater(1,2);
    aptana.assertLess(2,1);
    aptana.assertNotLess(2,1);
    aptana.assertContains("x",window);
    aptana.assertNotContains("document",window);
    aptana.assertTrue(false);
    aptana.assertFalse(true);
    aptana.assertNull(window);
    aptana.assertNotNull(null);
    aptana.assertUndefined(window);
    var x;
    aptana.assertNotUndefined(x);
    aptana.assertInstanceOf("str",Date);
    aptana.assertNotInstanceOf(new Date(),Date);
    aptana.assertTypeOf(1,"string");
    aptana.assertNotTypeOf(1,"number");
</script>
</head>

<body>
</body>
</html>

```

启动调试后,当某个断言失败时,程序的执行会被中断,并在 Firefox 中弹出一个对话框,如图 8.53 所示。

对话框包含 3 个按钮:Continue(继续)、Debug(调试)和 Terminate(终止)按钮。单击 Continue 按钮程序会忽略当前断言并继续执行,单击 Debug 按钮会转入 Aptana 进行调试,单击 Terminate 按钮会终止调试。



图 8.53 Assert 对话框

### 8.2.7 屏蔽调试代码

当调试完成后进行发布时,如果代码中仍然包含这些调试代码,用户在访问页面程序时就会发生错误,这时除了将测试代码删除外,还可以通过引入一段 JavaScript 代码来屏蔽这些测试代码,代码如下所示。

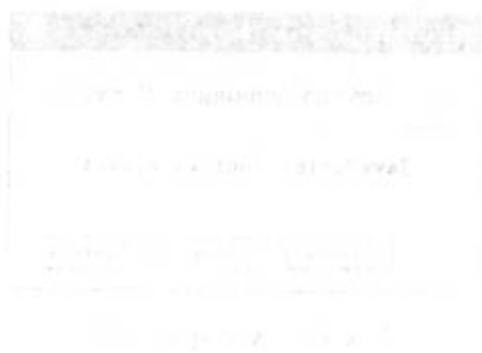
```

var names = ["log", "trace", "assert", "assertEquals", "assertNotEquals", "assertGreater", "assertNotGreater",
"assertLess", "assertNotLess", "assertContains", "assertNotContains", "assertTrue", "assertFalse", "assertNull", "
assertNotNull", "assertUndefined", "assertNotUndefined", "assertInstanceOf", "assertNotInstanceOf", "assertType
Of", "assertNotTypeOf"];
window.aptana = {};
for (var i = 0; i < names.length; ++i)
    window.console[names[i]] = function(){}
window.dump = function(){}
window.console = window.aptana;

```

### 8.3 小 结

本章向读者介绍了调试 Ajax 应用程序的一些技巧。首先介绍了如何利用 Firebug 的控制台来输出各种类型的日志信息, 以及介绍了命令行的使用, 然后介绍了利用 Firebug 脚本查看器进行断点调试和查看变量信息。接着向读者介绍了如何配置和使用 Aptana 的集成调试功能, 包括如何使用断点调试, 如何向控制台输出信息, 如何输出调用堆栈信息以及如何使用断言功能等。通过本章的学习, 同时结合第 5 章中对开发工具的介绍, 使读者能够使用调试工具对 Ajax 应用程序进行调试。



#### 8.2.7 屏蔽源代码

当网页完成加载后, 浏览器会尝试执行页面上的所有脚本。如果脚本中使用了未定义的变量或函数, 浏览器会抛出一个错误。为了防止这种情况, 可以在脚本的开始处添加一行代码, 将脚本中的变量和函数名称屏蔽掉。这样, 即使脚本中出现了未定义的变量或函数, 浏览器也不会抛出错误。

```

var names = ["log", "trace", "assert", "warn", "error", "info", "debug", "console.log", "console.trace", "console.assert", "console.warn", "console.error", "console.info", "console.debug"];
for (var i = 0; i < names.length; i++) {
    window.console[names[i]] = function() {
        window.debug = window.console;
    };
}

```



# 第 9 章

## 常见问题

- » 编码的处理
- » 控制缓存
- » 选择合适请求方式
- » 控制多个 Ajax 请求
- » Ajax 请求的安全性
- » 小结

第 8 章向读者介绍了调试 Ajax 应用程序的一些技巧。读者熟练掌握这些调试技巧可以解决在开发中遇到的大部分困难。但是仍然还会有这样一些问题：它们并不是真正意义上的语法或逻辑错误，它们多半由特定的环境所引起，并会妨碍应用程序的良好运行或者给应用程序留下隐患。解决这类问题需要依靠开发者的经验和智慧。本章将根据笔者的经验对这类问题进行总结，以帮助读者更加顺利地进行实际项目的开发。

## 9.1 编码的处理

编码就是用二进制的数字来对应表示字符集的字符，常用的编码有 gb2312 和 utf-8 两种。

最早发布的 gb2312 是简体中文字符集的中国国家标准，于 1981 年 5 月 1 日由中国国家标准总局发布，其适用于中国大陆、新加坡等地区，收录了 6700 多个汉字。gb2312 支持的汉字比较少，所以在 2000 年时又发布了 gb18030 编码，收录了 27484 个汉字，同时还收录了藏文、蒙文、维吾尔文等主要的少数民族文字。

提示：现在的中文 PC 平台一般都支持 gb18030 编码，但是很多嵌入式产品并不支持，例如手机、MP3 等。

utf-8 实际上是 Unicode 编码的一种传输方式。Unicode 也是一种字符编码方法，不过它是由国际组织设计，可以容纳全世界所有语言文字的编码方案。Unicode 的学名是 Universal Multiple-Octet Coded Character Set，简称 UCS。UCS 可以看作是 Unicode Character Set 的缩写。UCS 规定了怎么用多个字节表示各种文字，而专门传输这些编码则是由 UTF（UCS Transformation Format）规范规定的，其中包含 utf-7、utf-8、utf-16。

那么究竟该选择哪种编码来存储包含程序代码的文本文件呢？笔者推荐使用 utf-8。因为 Unicode 作为一种国际上通用的字符编码，使用 utf-8 在搭建国际化的应用程序时会有先天优势。试想一下，某位用户在访问其他语种的网站时，因为浏览器本身没有对表示该语种字符集的特有编码的支持，而导致显示的全部是乱码而不能正常访问，如果该网站使用 utf-8 编码，则访问不会受到任何影响。当需要在同一个页面上同时显示多个语种的字符时，这种优势表现得最为明显。

在 Ajax 应用程序的开发中，由于对编码的处理不当而导致乱码的情况很常见，下面对乱码发生的情况进行总结。

### 9.1.1 文件编码与声明编码

当程序页面的文件存储使用的编码与其在 meta 标签中所声明的编码不一致时，就可能发生乱码的问题。例如某页面存储使用的是 gb2312，而在 meta 标签中所声明的编码则是 utf-8，其代码如下所示。

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

  <body>
    <h1>编码测试</h1>
  </body>
</html>
```

使用浏览器访问该页面，页面上的中文字出现乱码，如图 9.1 所示。将页面文件的存储编码转换为 utf-8，或者将页面中声明的编码改为 gb2312，保持存储编码与声明编码一致，则可以解决乱码的问题，如图 9.2 所示。



图 9.1 出现乱码



图 9.2 显示正常

### 9.1.2 Ajax 请求乱码

用 Ajax 请求读取一个页面返回的数据时，responseText 中的中文有时会出现乱码。这是因为 XMLHttpRequest 默认会将返回的数据按照 UTF-8 进行解码，而如果服务器送出的数据并不是 UTF-8 编码的，这时就会出现乱码。下面来看一个发生乱码的例子，前台页面 demo.html 代码如下所示。

```
<!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>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>demo</title>
    <script type="text/javascript">
      function test()
      {
        try
        {
          var request = new XMLHttpRequest();
        }
        catch(e)
        {
          var request = new ActiveXObject("Microsoft.XMLHTTP");
        }
        request.open("GET","demo2.php",true);
        request.onreadystatechange = function()
        {
          if(request.readyState == 4 && request.status == 200)
          {
            document.getElementById("test").innerHTML = request.responseText;
          }
        }
        request.send();
      }
    </script>
  </head>
  <body>
    <div id="test">
    </div>
  </body>
</html>
```

```

</script>
</head>

<body>
<p id="test"></p>
  <input type="button" value="测试" onclick="test();" />
</body>
</html>

```

后台页面 demo2.php 代码如下所示。

```

<?php
  echo "Ajax 从入门到提高";
?>

```

示例中使用一个按钮来发起一个 Ajax 请求，以读取后台页面输出的数据，然后将数据显示在页面上，其界面如图 9.3 所示。单击“测试”按钮进行测试，由于后台页面使用 gb2312 编码，所以输出到前台后会产生乱码，如图 9.4 所示。

解决上述乱码问题有两种方法，第一种就是将后台页面转变为 utf-8 编码，第二种则是输出一个 HTTP 头信息来指明输出的数据所采用的编码，代码如下所示。

```
header("Content-type: text/html;charset=gb2312");
```

改变后台页面编码或者为输出数据制定编码后，再次单击“测试”按钮进行测试，中文被正常显示在页面上，如图 9.5 所示。



图 9.3 示例程序界面

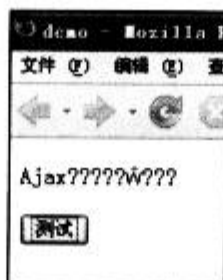


图 9.4 乱码



图 9.5 正常显示

### 9.1.3 发送数据乱码

当使用 Ajax 制造 HTTP 请求，并使用 POST 方法发送数据时，数据采用的是 utf-8 编码。如果接受数据的后台页面不是使用 utf-8 编码，则直接处理前台页面发送来的数据，此时会产生乱码。为了演示这个问题，在下面的示例程序中后台页面使用 gb2312 编码。前台页面通过一个输入框让用户填写数据并发送给后台页面，后台页面接收到数据后进行简单的字符串拼接处理，然后输出返回给前台页面并显示出来。前台页面代码如下所示。

```

<!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>

```



```

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>demo</title>
<script type="text/javascript">
    function test(data)
    {
        try
        {
            var request = new XMLHttpRequest();
        }
        catch(e)
        {
            var request = new ActiveXObject("Microsoft.XMLHTTP");
        }
        request.open("POST","demo3.php?",true);
        request.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
        request.onreadystatechange = function()
        {
            if(request.readyState == 4 && request.status == 200)
            {
                document.getElementById("test").innerHTML = request.responseText;
            }
        }
        request.send("data="+document.getElementById('txtData').value);
    }
</script>
</head>

<body>
<p id="test"></p>
    <input type="text" value="" id="txtData"/>
    <input type="button" value="测试" onclick="test();" />
</body>
</html>

```

后台页面 demo3.php 代码如下所示。

```

<?php
    header("Content-type: text/html; charset=gb2312");
    $data = '提交的数据是:'.$_REQUEST['data'];
    echo $data;
?>

```

访问前台页面，在页面中的输入框中输入“中文数据”并单击“测试”按钮提交，请求返回后页面上显示的信息出现乱码，如图 9.6 所示。

解决办法是在使用接收到的数据前，先对数据进行编码转换，将其转换为需要的编码类型即可。将后台页面的代码进行修改，如下所示。

```

<?php
    header("Content-type: text/html; charset=gb2312");
    $data = '提交的数据是:'.iconv('utf-8','gb2312',$_REQUEST['data']);

```



```
echo $data;
?>
```

然后再进行测试，一切正常，如图 9.7 所示。



图 9.6 发生乱码



图 9.7 显示正常

采用 POST 方式提交数据，所提交的数据一定会被使用 utf-8 进行编码。但是如果采用 GET 方式提交数据，如果页面使用的编码不是 utf-8，Firefox 等非 IE 核心的浏览器会对数据自动采用 utf-8 进行编码，但在 IE 下则不会，这样如果在后台进行简单的编码转换，始终不能兼顾这两种情况。这时就可以在前台页面使用 `encodeURIComponent` 方法对需要提交的数据进行强制转换，这样就可以保证提交的数据是采用 utf-8 编码的。`encodeURIComponent` 的语法如下所示。

```
Utf8Str = encodeURIComponent(str);
```

技巧：最好的避免乱码的方法仍然是所有页面和文件都采用 utf-8 一种编码。

## 9.2 控制缓存

在 IE 中使用 Ajax 时，用 GET 方式请求同一个 URL 地址，会发现在第一次请求后，后续的请求返回的数据都没有被更新。如果被请求的后台程序包含某种功能，例如更新数据库，开发者甚至会发现该功能只被执行了一次。下面这个示例采用一个 Ajax 轮询来获得后台 PHP 页面输出的不断更新的数据，代码如下所示。

```
<!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>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>cache demo</title>
    <script type="text/javascript">
      function test()
      {
        try
        {
          var request = new XMLHttpRequest();
        }
        catch(e)
        {
```

```

        var request = new ActiveXObject("Microsoft.XMLHTTP");
    }
    request.open("GET","cache.php",true);
    request.onreadystatechange = function()
    {
        if(request.readyState == 4 && request.status == 200)
        {
            document.getElementById("test").innerHTML = request.responseText;
            test();
        }
    }
    request.send();
}
</script>
</head>

<body>
<p id="test"></p>
<input type="button" value="测试" onclick="test();" />
</body>
</html>

```

后台 PHP 页面代码如下所示。

```

<?php
    echo time();
?>

```

使用 Firefox 等非 IE 核心的浏览器访问前台页面，并单击页面上的“测试”按钮，会看到页面上的数字会随着时间的变化而不断更新。但是如果使用 IE 访问并进行测试，则会发现第一次显示从后台读取的数字后，即使用户再次单击“测试”按钮也不会发生变化。这是为什么呢？其实这个问题是由 IE 的缓存引起的。当使用 GET 方式发送请求时，如果请求的 URL 没有变化，则会直接读取缓存中的数据，而不会真正去重新请求页面，这样就导致了请求只在第一次有效的现象。既然判断是否读取缓存的依据是 URL 有没有发生变化，那么可以利用这一点来巧妙地解决这个问题，方法就是给每次请求的 URL 加上一个随机的永不重复的参数，使用 UnixTime 可以满足要求。将上例代码中初始化 XMLHttpRequest 对象的代码进行修改，代码如下所示。

```
request.open("GET","cache.php?timeStamp="+ new Date().getTime(),true);
```

给请求的 URL 加上一个随机变化的 timeStamp 参数作为时间戳，这样每次请求的 URL 都不相同，就可以避免 IE 读取缓存。当然缓存也并不总是没有用处的，当确定后台返回的内容不会有更新时，可以合理地利用缓存来提交应用程序的性能。

### 9.3 选择合适的请求方式

Ajax 的初学者经常会犯的一个错误就是对 GET 和 POST 两种请求方式在使用上没有任何依据，而

是随心所欲地选择其中一种方式进行数据交互。可能读者对 GET 和 POST 两种方式在使用上并没有感觉到有很大差异,但是 HTTP 标准提供了与这两种不一样的方法,是为了达到不同的交互目的。

POST 方法用于创建一个资源,资源的内容将被编码到 HTTP 的主体中,往往被用作有副作用的请求,例如更新数据库等。GET 方法则用于没有任何副作用的请求,例如执行一次查询。而且由于 GET 方法是将查询参数附加在 URL 后面,如果 URL 过长,则可能会超过浏览器和服务端所支持的最大长度,从而产生错误。下面列举了选择 POST 或者 GET 方法的标准。

当下列任何条件成立时,都应该使用 POST 方法:

- ☐ 请求的结果有永久性的副作用。例如,向数据库中插入一行数据、更新某个文件等。
- ☐ 要发送的数据不是使用 7 位 ASCII 编码。
- ☐ 需要发送的数据超过 1024 字节。

当下列条件全部成立时,才可以使用 GET 方法:

- ☐ 请求本质上只是为了查找一个资源。
- ☐ 请求的结果没有任何永久性的副作用。
- ☐ 请求的数据小于 1024 字节。

## 9.4 控制多个 Ajax 请求

在一些需要使用大量 Ajax 请求的程序中,例如聊天室程序中,使用 Ajax 轮询来获得最新的发言信息,又例如 Google Suggest 中根据用户的输入不断发送请求等,如果对这些请求没有进行良好的控制,则可能对应用程序本身构成危害。

### 9.4.1 轮询模式

首先来看第一种情况。某个应用程序采用 Ajax 不断向某个后台页面进行请求,以实时更新界面。程序采用定时器定时向后台页面发起请求,但是对所发起的请求缺乏控制,这样就会有一定的几率发生界面更新错误的情况。因为虽然发起请求的时间有先后,但是发送请求到请求返回的这个时间间隔是不可预计的,受瞬时网络状况的影响,先发起的请求可能会比后发起的请求返回的慢,这样就会导致用旧的数据更新了已经使用新的数据更新过的界面,从而发生错误,如图 9.8 所示。

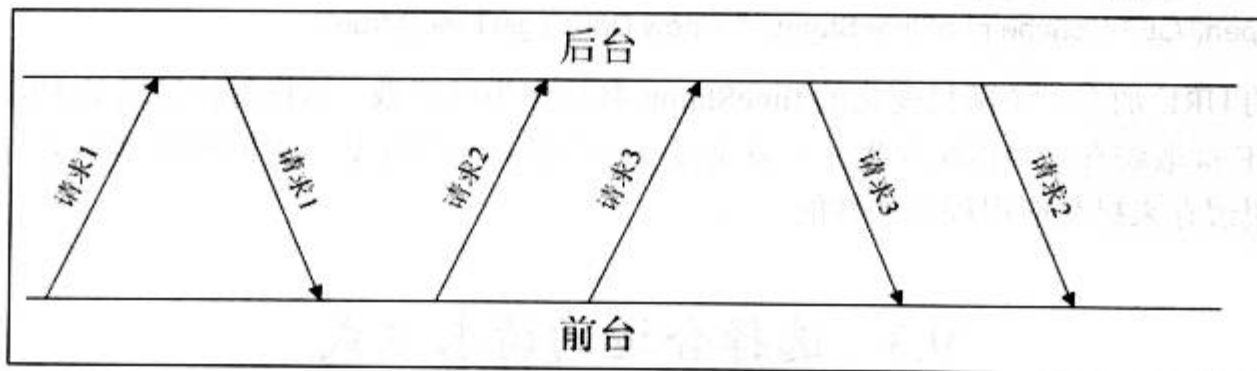


图 9.8 错误状态

一个控制良好的请求模型应该是当前一个请求返回后，才发起下一个请求，如图 9.9 所示。

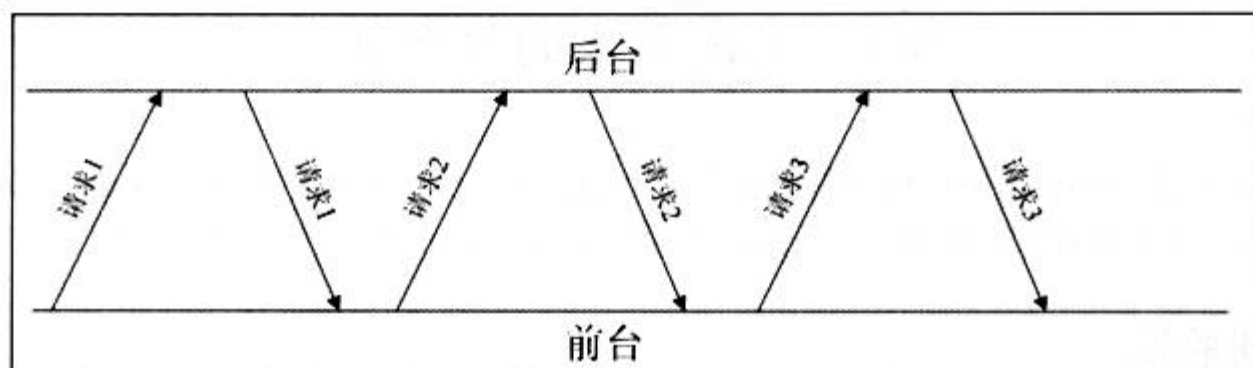


图 9.9 正确状态

其实现思路很简单，就是将下一次请求的发起放在上一个请求完成的回调函数中实现。

### 9.4.2 事件响应模式

第二种情况就是需要及时响应用户操作并发起请求来更新界面或者完成某项操作。一种糟糕的设计方式就是在每次用户操作的事件中去发起请求，这样往往会浪费更多的资源来做无意义的事情。例如在某个程序中需要用户使用方向键来选择一个项目，并实时根据选择的内容在界面上显示相关的数据，又或者像 Google Suggest 那样依靠用户的输入来返回相应的提示数据。此时如果在每次事件中都发起请求，就会发生资源浪费的情况。例如用户需要输入“ajax”并查看关于 ajax 的内容，但是当用户每次输入时都发出了请求，实际上最少有 4 次请求发生，而不是用户需要的一次。如果这种设计存在于访问量很大的系统中，则会给负责接收请求并处理数据的服务器带来巨大的额外负担。

良好的设计应该是等用户完成输入或者到达预期的位置后才发送必要的请求，但是开发者并不能预先知道用户需要的是什么，那么这似乎是无法实现了。其实虽然不能完美地直接实现，但是可以通过一种巧妙的方式来间接地达到或者接近预期的目的，这就是给每次请求设置延迟。在每次事件中，设置一个延迟来发送请求，在下一次事件中预先判断是否存在仍然处于延迟阶段、未被发送的请求，如果存在，则取消这个请求的发送，然后重新设置一个新的延迟发送的请求，延迟的时间间隔视需要而定。因为在一般情况下，用户进行某种操作时所发生的动作是连续的，直到操作效果达到预期的目标时才会中断，这样通过延迟机制，将延迟的时间设置为略大于用户每步操作的平均时间间隔，就可以巧妙地借助用户的这一行为习惯来避免不必要的性能开销。延迟机制的图形表示如图 9.10 所示。

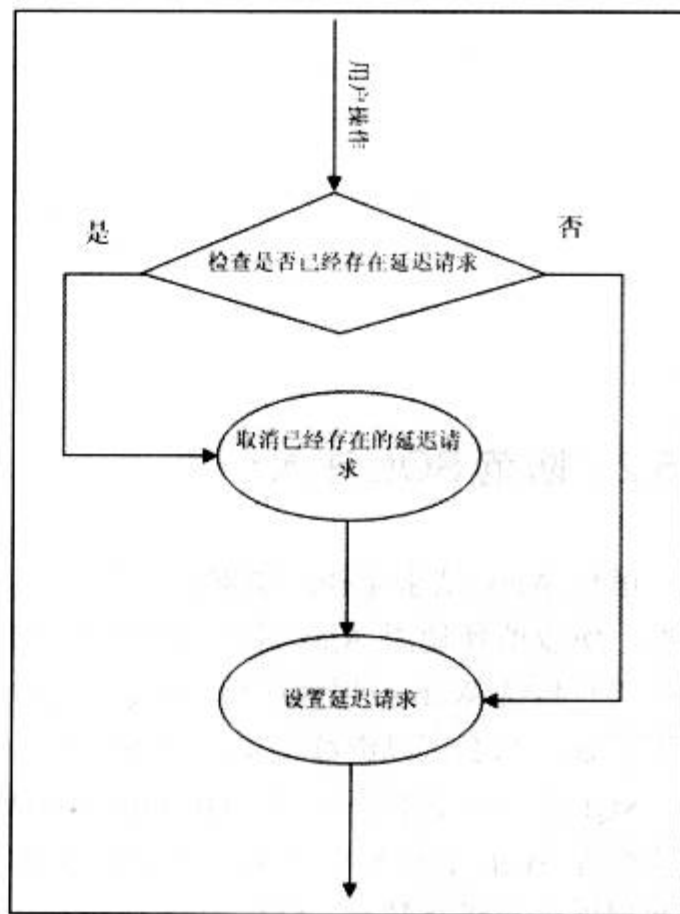


图 9.10 延迟机制



## 9.5 Ajax 请求的安全性

使用 Ajax 技术并不会给应用程序带来安全性上的加分, 反而如果开发者对 Ajax 的安全性没有足够的认识, 将会带来更多潜在的威胁。下面向读者介绍关于 Ajax 安全性的经验法则。

### 9.5.1 身份验证

首先是身份验证问题。在传统 Web 应用程序中, 当用户需要访问某些受保护的页面时, 就需要先对用户进行身份验证, 验证通常在被访问的页面上进行。而在 Ajax 应用程序中, 传统的页面会减少, 用户访问的只是少数几个页面, 而这些少量的页面通过 XMLHttpRequest 请求后台页面以获得丰富的数据或进行各种操作。这时身份验证不能仅仅在被用户访问的少数几个页面上进行, 而同时也需要在与这些页面发生数据交互的所有后台页面上进行, 如图 9.11 所示。

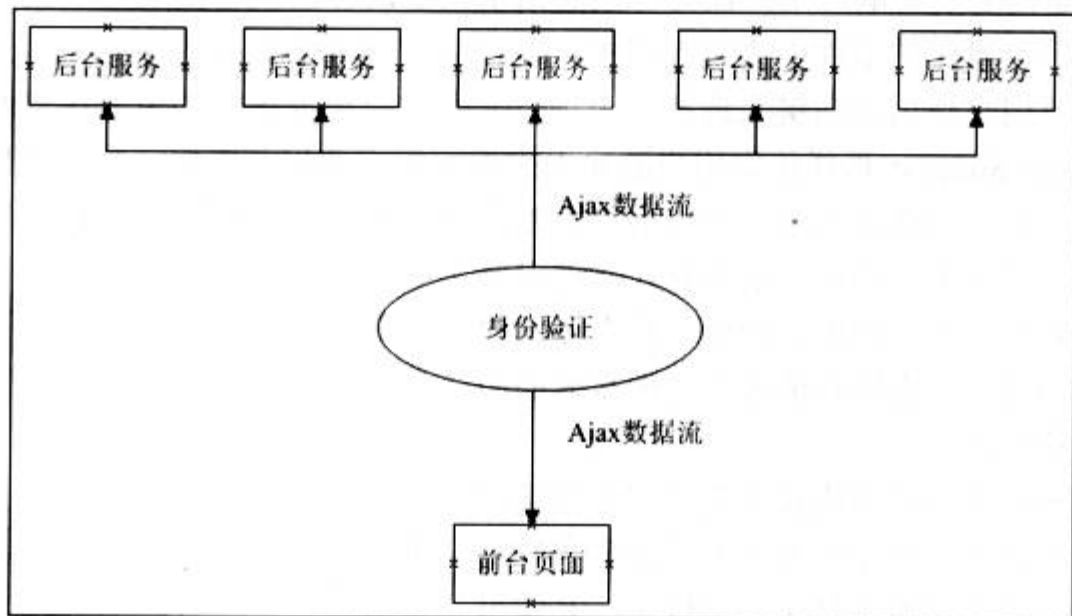


图 9.11 身份验证

### 9.5.2 防范 SQL 注入

虽然 Ajax 请求是在浏览器后台发生的, 用户不能直接控制, 但是因为 Ajax 技术总是基于 HTTP 模型, 所以即使使用 Ajax 来代替传统模式的交互, 在面对 SQL 注入攻击时仍然没有任何先天优势。所以所有传统 Web 应用程序使用的防范 SQL 注入的方法, Ajax 应用程序都需要。如果读者对 SQL 注入不了解, 那么可以继续浏览本节下面对 SQL 注入的介绍, 否则可以跳过本节余下的内容。

SQL 注入攻击源于英文 SQL Injection Attack, 一般而言, 是指通过恶意提交包含 SQL 脚本的数据来影响在 Web 后台程序中实际执行的 SQL 语句, 从而对数据库进行攻击, 或者通过返回的错误信息对数据库结果进行猜测。SQL 注入攻击利用的是 SQL 语法, 这样就使得这种攻击具有广泛性, 理论上说, 对所有基于 SQL 语言标准的数据库软件都是有效的, 包括 MS SQL Server、DB2、Sybase、MySQL、



Sybase 等。

防范 SQL 注入的一般方法是对用户提交的数据进行验证,以及对输出的错误信息进行必要的处理。对提交的数据进行验证包括数据类型的检查、数据长度的检查以及一些与 SQL 语句有关的敏感字符的过滤。对错误信息的处理表示当 SQL 语句发生错误时,不要直接将错误信息暴露给用户,而是输出一个替代的、不包含任何内部错误信息的友好错误提示,这样攻击者就不能依靠错误提示对数据库结构进行猜解。

说明:关于 SQL 注入攻击就简单地介绍到这里,有需要进一步了解的读者可以查阅相关的资料进行深入研究。

### 9.5.3 防范 JavaScript 注入

所谓 JavaScript 注入,就是在攻击者打开程序页面后,通过浏览器运行额外的 JavaScript 来改变页面中的数据,从而对应用程序进行攻击。通过在浏览器地址栏采用 javascript:伪协议的方式输入“JavaScript”,或者通过类似 Firebug 控制台的命令行的工具,都可以轻易执行任何 JavaScript 代码,通过 DOM 接口,理论上可以改变页面上的任何数据和内容。

一般情况下,这种攻击是毫无意义的,因为攻击者改变的只是页面在攻击者本地的一个副本。但是如果在一个高度依赖客户端脚本的应用程序中,某个关键的数据被修改了,而当这个数据被提交到后台程序时,且没有对其有效性进行足够的验证,则可能产生严重的后果。

例如某个电子商务系统,通过复杂的前台操作后生成了一个订单,然后将订单数据提交给后台进行后续操作,如进行支付然后自动发货。而如果攻击者在生成了一个巨额订单后,通过 JavaScript 注入修改了定单的金额,在提交给后台后,却没有对金额进行足够的验证,这样就会被攻击者欺骗。

防范 JavaScript 注入的方法,就是将关键的业务逻辑放在后台处理,这样就可以避免 JavaScript 注入的攻击,同时还需要对关键的业务数据建立有效的校验机制,以防止虚假数据的提交。

## 9.6 小 结

本章对 Ajax 应用程序开发中常见的一些问题进行了总结和分析。首先介绍了编码的相关知识,如何选择编码,以及如何应对乱码的问题。然后介绍了 IE 的缓存机制对 Ajax 应用程序的影响,并提供了解决方法。接着向读者讲解了 POST 和 GET 两种请求方式在使用上的区别,以及如何选择合适的请求方式。然后分轮询模式和事件响应模式两种情况,向读者介绍了如何控制多个 Ajax 请求。最后从身份验证、SQL 注入的防范以及 JavaScript 注入的防范 3 个方面分析了 Ajax 应用程序的安全性问题。

2. 数据库

数据库是存储数据的仓库，它按照一定的数据模型组织、存储和管理数据。数据库系统由数据库、数据库管理系统（DBMS）和数据库用户组成。数据库用户包括数据库管理员（DBA）、应用程序员和最终用户。数据库管理员负责数据库的创建、维护和管理。应用程序员负责开发数据库应用程序。最终用户负责使用数据库应用程序。数据库系统的主要功能包括：数据定义、数据操作、数据控制、数据查询和数据维护。

数据库系统的发展经历了三个阶段：第一阶段是人工管理阶段，第二阶段是文件系统阶段，第三阶段是数据库系统阶段。数据库系统阶段的特点是：数据集中、数据共享、数据独立、数据安全和数据完整性。

## 2.2.3 数据库系统

数据库系统是由数据库、数据库管理系统（DBMS）和数据库用户组成的。数据库是存储数据的仓库，数据库管理系统是管理数据库的软件，数据库用户是使用数据库的人。数据库系统的主要功能包括：数据定义、数据操作、数据控制、数据查询和数据维护。数据库系统的发展经历了三个阶段：第一阶段是人工管理阶段，第二阶段是文件系统阶段，第三阶段是数据库系统阶段。数据库系统阶段的特点是：数据集中、数据共享、数据独立、数据安全和数据完整性。

## 2.3 数据库系统

数据库系统是存储和管理数据的系统，它由数据库、数据库管理系统（DBMS）和数据库用户组成。数据库是存储数据的仓库，数据库管理系统是管理数据库的软件，数据库用户是使用数据库的人。数据库系统的主要功能包括：数据定义、数据操作、数据控制、数据查询和数据维护。数据库系统的发展经历了三个阶段：第一阶段是人工管理阶段，第二阶段是文件系统阶段，第三阶段是数据库系统阶段。数据库系统阶段的特点是：数据集中、数据共享、数据独立、数据安全和数据完整性。

# 第4篇



## Ajax 应用进阶篇

- 第10章 数据的组织方式: XML
- 第11章 数据的组织方式: JSON
- 第12章 JavaScript面向对象编程 (OOP)
- 第13章 浏览器的兼容性问题
- 第14章 Prototype.js框架介绍
- 第15章 关于架构的思考
- 第16章 Ajax的缺陷及补救



# 第4章

AjAx 应用案例

- 第10章 Ajax 的简单应用
- 第11章 关于 Ajax 的若干问题
- 第12章 Prototype.js 框架介绍
- 第13章 跨浏览器兼容性问题
- 第14章 JavaScript 面向对象编程 (OOP)
- 第15章 数据组织方式: JSON
- 第16章 数据组织方式: XML

# 第10章

## 数据的组织方式：XML

- » XML 概述
- » XML 语法规范
- » XML 命名空间
- » XML Schema
- » XML DOM
- » 强大的检索工具：XPath
- » 格式化 XML 工具：XSL
- » 在客户端格式化 XML
- » 跨浏览器的 XML 开发框架：zXML
- » 应用实例：Ajax 文章列表程序 (XML)
- » 小结

Ajax 中的 x 就是 XML 的意思。XML 是英文 Extensible Markup Language 的缩写，即可扩展标识语言的意思。XML 在 Ajax 应用程序中扮演着重要角色，它承担着数据的组织工作。在使用 Ajax 技术进行复杂的数据交互时，往往先将数据封装成 XML 格式，再进行发送。本章将对 XML 及其相关的技术做全面的介绍。



## 10.1 XML 概述

XML 是 Extensible Markup Language 的简称,即可扩展标记语言。很多人将 XML 当作 HTML 的简单扩展,实际上这是一种误解。XML 是从 SGML 和 HTML 发展而来的。

SGML 是指通用标识语言标准 (Standard Generalized Markup Language),它是国际上定义电子文件结构和描述内容的标准,是一种非常复杂的文档结构。同 XML 相比,SGML 定义的功能很强大,但是它不适用于 Web 数据的描述,而且 SGML 软件的价格非常地昂贵。

读者比较熟悉的 HTML,即超文本标记语言 (HyperText Markup Language),它的优点是比较适合 Web 页面的开发。但是它有一个比较大的缺点是标记较少,只有固定的标记集,缺少 SGML 的柔性和适应度,不能支持特定领域的标记语言。例如,Web 开发者很难在 Web 页面上表示数学公式、化学分子式或者乐谱等。

XML 继承了 SGML 和 HTML 的优点,并消除了各自的一些缺点。XML 仍被认为是一种 SGML 语言,它比 SGML 要简单,但是能实现 SGML 的大部分功能。XML 具有以下特点:

- XML 是一种元标记语言,元标记指开发者可以根据自己的需要定义自己的标记,任何满足 XML 命名规则的名称都可以用作标记,这就为不同的应用程序打开了大门。
- XML 是一种语义/结构化语言,它描述了文档的结构和语义。
- XML 是一种通用的数据格式,其表示的数据是独立于任何平台的。它提供的是一种规范,利用这种规范,任意平台上的任意应用程序之间都可以进行通信。

一个基本的 XML 文档范例如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<notes>
  <note level="1">
    <title>Ajax</title>
    <author>Robin</author>
    <date>2007-11-15</date>
    <content>He talk about Ajax.</content>
  </note>
  <note level="2">
    <title>XML</title>
    <author>Robin</author>
    <date>2007-11-16</date>
    <content/>
  </note>
</notes>
```

## 10.2 XML 语法规范

XML 是一种元标记语言,它允许开发者使用自定义标签,给开发者提供了极大的自由发挥空间,

但是 XML 本身却有着非常严格的语法规则。开发者必须熟悉这些规范,才能正确使用 XML。下面开始向读者一一介绍这些语法规则。

### 10.2.1 XML 声明

XML 文档的第一行是 XML 的声明。在 10.1 节 XML 概述中的基本 XML 范例,其声明如下所示。

```
<?xml version="1.0" encoding="utf-8">
```

XML 声明中, version 属性定义了该文档所遵循的 XML 标准的版本,在这个例子中是 1.0 版本的标准, encoding 属性声明了当前 XML 文档所使用的字符编码格式。

### 10.2.2 根节点

范例中的第二行是 XML 的根节点,如下所示。

```
<notes>
```

每个 XML 文档必须有且只能有一个根节点,其他所有的节点都是该节点的子孙节点。一般而言,根节点在命名上概括了文档的数据内容。

### 10.2.3 开始和结束标记

在 XML 文档中,所有的元素都必须有结束标记。在 HTML 中一些元素是可以没有结束标记的,例如 p 标签,但是 XML 中元素必须有与其开始标记相对应的结束标记。例如前文 XML 范例中的一个 note 节点,代码如下所示。

```
<title>Ajax</title>
<author>Robin</author>
<date>2007-11-15</date>
<content>He talk about Ajax.</content>
```

其中 title 元素、author 元素和 date 元素都有相应的结束标记。

当一个 XML 元素没有文本节点时,可以采取自封闭语法对该元素节点进行关闭,如下所示。

```
<content/>
```

注意: XML 声明不属于 XML 元素,所以不需要结束标记。

### 10.2.4 属性

XML 同 HTML 一样,其元素可以包含属性。XML 的属性以名/值的形式成对出现。例如前文 XML 范例中 note 节点的 level 属性,如下所示。

```
<note level="1">
```

XML 语法规规范要求 XML 的属性值必须使用引号 ( " ") 来包含, 否则就会被当作是错误的。

### 10.2.5 合理地嵌套包含

XML 不允许元素之间交叉包含, 例如下面的包含方式就是错误的。

```
<name>Robin<sex>male</name></sex>
```

正确的写法如下所示。

```
<name>Robin</name>
```

```
<sex>male</sex>
```

### 10.2.6 大小写敏感性

XML 是大小写敏感的, 所有的 XML 组成部分, 包括标签名、属性名以及值等都受大小写的影响。例如 <note> 与 <Note> 是不同的。一个 XML 新手常犯的错误就是标签的开始和关闭所使用的大小写不一致而导致错误发生, 如下所示。

```
<content>Something wrong..</Content>
```

content 元素并没有正确的被关闭, 因为大小写的关系, </Content> 与 <content> 不能正确匹配。首字母大写的 Content 在这里被当作了另外一个元素, 而文档中找不到其开始标记。正确的写法如下所示。

```
<content>Something wrong..</content>
```

### 10.2.7 空白被保留

在 XML 文档中, 空白部分并不会被解析器自动删除, 而是被完整地保留下来。例如, 有下面这样一句话。

```
"Hello      I'm Robin."
```

在 HTML 中, 多余的空格会被去掉, 这句话会被显示为 “Hello I’m Robin.”。但是在 XML 中, 所有的空白部分会被当作数据的一部分被完整地保留。

### 10.2.8 XML 的注释

XML 的注释采用和 HTML 一样的格式, 即使用 <!-- 标识注释的开始, 使用 --> 标识注释的结束。下面是一个在 XML 中使用注释的示例, 代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<!--
```

```
This document contains some notes' data.
```

```
-->
```

```
<notes>
```

```
<!-- a note record -->
<note level="1">
  <!-- the note's title -->
  <title>Ajax</title>
  <!-- the note's title -->
  <author>Robin</author>
  <!-- the note's post date -->
  <date>2007-11-15</date>
  <!-- the note's content -->
  <content>He talk about Ajax.</content>
</note>
<note level="2">
  <title>XML</title>
  <author>Robin</author>
  <date>2007-11-16</date>
  <content/>
</note>
</notes>
```

### 10.2.9 元素的命名

虽然 XML 允许开发者自己为 XML 元素命名,但是仍然需要开发者遵守下面这些命名规则。

- ❑ 元素的名字可以包含字母、数字和其他字符,但是不能以数字或标点符号开头。
- ❑ 元素的名字不能以 XML (以及其所有的大小写组合模式,包括 Xml、XmI 等) 开头。
- ❑ 元素的名字不能包含空格。

例如,下面这些命名都是不合法的。

```
<1item>
<+item>
<xmlitem>
<item one>
```

下面这些命名都是合法的。

```
<item>
<item1>
<item_one>
```

除了必须要遵守上面提到的这些规则外,为了让 XML 的数据能够更好地被理解,还应该遵守下面这些建议。

- ❑ 元素的名字应该具有可读性。例如<name>、<sex>、<age>等。
- ❑ 元素的名字不要太长,尽量简单易读。例如<the\_name\_of\_the\_person>就显得过于罗嗦,直接<name>即可。
- ❑ 如果数据是从某个数据库中得到的,那么尽量保持元素的名字与数据表中字段的名字一致,这样可以方便数据交换。
- ❑ 虽然非英文的字符也可以用在元素的名字中,并且完全合法,例如<名字>、<性别>、<年龄>等,但是某些软件并不能很好地支持这种命名,所以尽量使用英文字母来对元素进行命名。

### 10.2.10 转义字符

当在 XML 中使用类似“<”的字符时，会引起 XML 的解析错误，因为解析器会认为这是一个新的元素的开始。例如在下面的例子中，需要使用“<”来表示一个条件关系。

```
<condition>x < y</condition>
```

为了避免这样的情况出现，需要将“<”进行转义，代码如下所示。

```
<condition>x &lt; y</condition>
```

表 10-1 列举了 5 个转义字符。

表 10-1 XML 转义字符

字 符	转 义 字 符	说 明
<	&lt;	小于号
>	&gt;	大于号
&	&amp;	和
'	&apos;	单引号
"	&quot;	双引号

注意：除了“<”和“>”外，其他都不是强制要求转义的，但为了减少出错，使用转义字符是一种好习惯。

### 10.2.11 CDATA 部件

当某个节点的数据包含大量的需要转义的字符时，可以使用 CDATA 部件来避免逐一转义的麻烦。CDATA 部件以“<![CDATA[”开始，以“]]>”结束，其包含的所有文本都会被当作普通文本处理，所以特殊符号都会被忽略掉。其使用方法如下所示。

```
<scriptCode>
<![CDATA[
If(a > b)
{
    alert(b);
}
else if(a < b && a != 0)
{
    alert(a);
}
]]>
</scriptCode>
```

注意：CDATA 部件之间不能再包含 CDATA 部件（不能嵌套）。如果 CDATA 部件包含了字符“]]>”或者“<![CDATA[”，将有可能出错。同样要注意在字符串“]]>”之间没有空格或者换行符。



## 10.3 XML 命名空间

在使用 XML 描述数据时,有时会遇到元素命名冲突的问题。例如,在一个 XML 片段中,使用一个名为 bag 的元素来表示一个包里装的所有的东西,代码如下所示。

```
<bag>
  <handset>1</handset>
  <watch>1</watch>
  <flashlight>1</flashlight>
</bag>
```

在另外一个 XML 片段中,使用一个名为 bag 的元素来表示一种包的各种属性,代码如下所示。

```
<bag>
  <brand>LV</brand>
  <price>8200</price>
</bag>
```

当上面这两个片段在一起使用时,就会发生命名冲突。因为两个片段都包含了 bag 元素,却又有着不同的定义和内容。当命名发生冲突时,就会给数据的解析带来障碍。当然可以通过修改元素名来解决命名的冲突问题,但这样一来,原先使用这个 XML 片断的程序就可能无法工作了。

XML 为了解决这个问题,提供了命名空间的机制。通过给元素指定一个前缀,并将其与某个 URI (统一资源定位符) 相关联,从而对同名的元素进行区分。命名空间的声明如下所示。

**<前缀名:element xmlns:前缀名="URI 地址">**

前缀名可以是除了以 x、m、13 个字母序列的任何形式开头的字符串以外,任意 XML 允许的字符或者字符串。而 URL 本身并不会被 XML 解析器所使用,所以只需要提供一个唯一的不重复的 URL 地址即可。声明好命名空间后,在需要与该命名空间相关联的元素开始标签中使用定义好的前缀即可。例如将上面两段 XML 使用命名空间组合在一起,代码如下所示。

```
<c:bag xmlns:c="http://www.robchen.cn/container">
  <c:handset>1</c:handset>
  <c:watch>1</c:watch>
  <c:flashlight>1</c:flashlight>
</c:bag>
<p:bag xmlns:p="http://www.robchen.cn/product">
  <p:brand>LV</p:brand>
  <p:price>8200</p:price>
</p:bag>
```

这样通过给不同的 bag 元素指定不同的命名空间,成功解决了命名冲突的问题。

命名空间有作用范围的概念,一个被声明的命名空间只能作用于当前元素和其子孙元素上。例如在下面的例子中,第一个 bag 元素使用前缀 k 就是错误的,代码如下所示。

```
<c:bag>
  <c:handset>1</c:handset>
  <c:watch>1</c:watch>
  <c:flashlight>1</c:flashlight>
</c:bag>
<c:bag xmlns:c="http://www.robchen.cn/container">
  <c:handset>1</c:handset>
  <c:watch>1</c:watch>
  <c:flashlight>1</c:flashlight>
</c:bag>
```

在一个元素中可以声明多个命名空间以应付复杂的应用情况，如下例所示。

```
<c:company xmlns:c="http://www.robchen.cn/company" xmlns:m="http://www.robchen.cn/manager">
  <c:name>Microsoft</c:name>
  <c:telephone>8888888</c:telephone>
  <m:manager>
    <m:name>Bill Gates</m:name>
    <m:telephone>88888888</m:telephone>
  </m:manager>
</c:company>
```

还有一种使用 XML 命名空间的简单方法，就是给元素设置默认的命名空间。设置默认的命名空间不需要使用前缀，元素和其子元素也不需要使用前缀。默认命名空间被设置后，当前元素及其子孙元素都会默认与该命名空间进行关联，除非重新对某个子孙元素使用了另外的命名空间。一个使用默认命名空间的例子如下所示。

```
<company xmlns="http://www.robchen.cn/company">
  <name>Microsoft</name>
  <telephone>8888888</telephone>
  <m:manager xmlns:m="http://www.robchen.cn/manager">
    <m:name>Bill Gates</m:name>
    <m:telephone>88888888</m:telephone>
  </m:manager>
</company>
```

## 10.4 XML Schema

由于 XML 文档可以使用自定义的标记，这样就导致 XML 文档没有固定的标记和格式。XML 的这种特性在给开发者带来便利的同时，也给 XML 文档的有效性验证带来了困难。因为程序在处理一个 XML 时并不能保证该 XML 有其需要的足够数据，例如某个表示产品的<product>元素不一定总是包含一个表示其价格信息的<price>子元素。或者某个重要的元素，其值并不是需要的类型，例如表示价格的<price>元素却被赋予了一个不相关的字符串值。这就给 XML 文档的处理带来了许多不确定性，数据的缺失或者类型的错误等问题必须要等到程序处理 XML 中途才能被发现。

一种更好的方式是对不同的 XML 文档的结构和数据进行约束，处理程序通过获得文档的约束细

节, 可以在解析 XML 文档中的数据前就能对 XML 文档的有效性进行验证。W3C 推荐的约束方式是使用 XML Schema。一个 XML Schema 本身就是一个符合 XML 语法的文档, 其按照特定的规范描述了被约束的 XML 文档所需要包含的元素、元素所具有的属性、元素出现的顺序和次数、元素值的类型等各种细节。

### 10.4.1 基本示例

假设在某个文章系统中, 一个提供数据输出的程序从数据库中提取需要的数据, 然后输出一个表示文章列表信息的 XML 文档, 代码如下所示。

```
<?xml version="1.0" encoding="utf-8" ?>
<notes xmlns="http://www.robchen.cn/news" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.robchen.cn/news newsSchema.xsd">
  <note>
    <title>HTML</title>
    <summary>Learning HTML</summary>
    <author>
      <name>Susan Smith</name>
      <sex>female</sex>
    </author>
    <tags>HTML CSS</tags>
    <hit date="2007-11-17">267</hit>
    <url>http://www.robchen.cn/news/HTML.html</url>
  </note>
  <note postDate="2007-11-17">
    <title>Ajax</title>
    <summary>Learning Ajax</summary>
    <author>
      <sex>male</sex>
      <name>Robin Chen</name>
      <age>22</age>
    </author>
    <tags>Ajax JavaScript XML JSON</tags>
    <hit date="2007-11-17">854</hit>
    <url>http://www.robchen.cn/news/JSON.html</url>
  </note>
  <note>
    <title>XML Schema</title>
    <summary>Something About the XML Schema</summary>
    <comment>Nice</comment>
    <author>
      <name>Robin Chen</name>
      <age>22</age>
    </author>
    <url>http://www.robchen.cn/news/XMLSchema.htm</url>
  </note>
</notes>
```

系统需要对输出的 XML 文档进行下列约束：

- ❑ 根节点 `notes` 下包含 0 个或多个 `note` 元素。
- ❑ `note` 元素有一个可选的属性 `postDate`，其类型为日期类型，如果没有提供，则默认值为 2007-11-17。
- ❑ `note` 元素的第一个子元素必须为表示文章标题的 `title` 元素，`title` 元素不包含任何属性和子元素，其值必须为字符串类型。
- ❑ `note` 元素的第二个子元素必须为表示文章内容简介的 `summary` 元素，该元素不包含任何属性和子元素，其值必须为字符串类型。
- ❑ `note` 元素的下一个元素为表示文章评论的 `comment` 元素，该元素为可选项，但出现次数不能超过 1 次。如果出现，则其值必须为字符串类型。
- ❑ 在 `comment` 元素之后，必须有一个或者多个表示该文章作者的 `author` 元素。
- ❑ `author` 元素必须有一个表示作者姓名的 `name` 子元素，其值为字符串型。另外还可以有两个可选的子元素：表示作者性别的 `sex` 元素和标识作者年龄的 `age` 元素。如果出现 `sex` 元素，则只能从 `male` 和 `female` 中取值。如果出现 `age` 元素，则取值范围只能是 20~80 之间的整数（包含 20 和 80）。`name`、`sex` 和 `age` 3 个元素的先后顺序不做限制。
- ❑ `author` 元素之后，有一个表示文章标签的 `tags` 元素，该元素为可选项，最多只能出现 1 次。该元素的值为一个以空格分隔的字符串列表，例如 `HTML XML Ajax`，最多只能包含 10 个列表项。
- ❑ `tags` 元素之后，有一个表示文章单击数的 `hit` 元素，该元素为可选项，最多出现一次。该属性的取值范围为大于等于 0 的整数，默认值为 0。
- ❑ `hit` 元素之后必须有一个表示文章地址的 `url` 元素，其值为字符串型，但必须以 `http://www.robche.cn/news/` 开头，而且最后的文件名必须是 `html` 或 `htm`。

一个描述了上列约束内容的 XML Schema 文档如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="XMLSchema1" targetNamespace="http://www.robchen.cn/news"
  elementFormDefault="qualified" xmlns="http://www.robchen.cn/news"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="notes" type="Notes">
    </xs:element>
    <xs:element name="comment" type="xs:string">
      </xs:element>
    <xs:complexType name="Notes">
      <xs:sequence>
        <xs:element name="note" type="Note" minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="Note">
      <xs:sequence>
        <xs:element name="title" type="xs:string" />
        <xs:element name="summary" type="xs:string" />
        <xs:element ref="comment" minOccurs="0" />
        <xs:element name="author" type="AuthorType" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:schema>
```



## 10.4.2 定义元素

XML Schema 最基本的就是定义一个 XML 元素，其与受该 XML Schema 约束的 XML 文档中的同名元素相关联。定义一个元素很简单，例如上例 XML 中对 comment 元素进行定义，如下所示。

```
<xs:element name="comment" type="xs:string">
  </xs:element>
```

xs 是命名空间。元素 element 用来定义一个可能在 XML 中出现的元素。name 属性指定了该元素的名称，type 属性则指定了该元素的类型。如果被约束的 XML 文档中存在名为 comment 的元素，则在 XML Schema 中定义的该元素的约束规则会被利用来检查该元素。

## 10.4.3 简单类型

元素如果包含其他子元素或者属性，则称之为复合类型；元素如果仅仅只包含数字、字符串或其他数据等，但不包含任何子元素或者属性，则称之为简单类型。XML Schema 内置了一部分标准的简单类型提供给开发者使用，更多的简单类型以及复合类型，则需要开发者自己定义。例如上面例子中的 string，就是内置的简单类型，表示字符串类型。表 10-2 列举常用的内置简单类型。

表 10-2 常用内置类型

简单类型	示 例	备 注
string	Ajax and XML	
normalizedString	Ajax and XML	在string的基础上将换行符、回车和制表符转换成空格
token	Ajax and XML	在normalizedString基础上去除首位的空格并合并连续的空格
byte	-1,120	-255~255之间的所有整数
unsignedByte	0,126	0~255之间的所有整数
integer	-1,13762,0,-311	
positiveInteger	1,13762	正整数
negativeInteger	-1,-311	负整数
nonNegativeInteger	0,1,13762	非负整数
nonPositiveInteger	0,-1,-311	非正整数
int	-4,1245432	
unsignedInt	4,122312	无符号整型
long	-1, 12678967123123	
unsignedLong	0, 12678967123123	无符号长整型
short	-1,12678	
unsignedShort	0,12678	
decimal	-1.23,0,123.1	
float	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN	等同于单精度32位浮点数，其中NaN表示“不是一个数字”



续表

简单类型	示 例	备 注
double	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN	等同于双精度64位浮点数
boolean	true, false, 1, 0	
time	13:20:00.000, 13:20:00.000-05:00	
dateTime	2007-11-15T13:20:00.000-08:00	这个时间表示的含义是: 2007年11月15日美东标准时间下午1:20, 注意后面的-08:00表示这个时间比格林尼治时间早8个小时
duration	P1Y2M3DT10H30M12.3S	这表示经过了1年2个月3天又10个小时30分钟12.3秒
date	2007-11-16	
gMonth	--11--	表示11月
gYear	2007	表示2007年
gYearMonth	2007-11	表示2007年11月, 但不关心具体是几号
gDay	---16	表示16号
gMonthDay	--11-16	表示11月16号
anyURI	http://www.robchen.cn/, http://www.robchen.cn/example	任意URI地址
language	zh-cn,en,fr	XML1.0中定义的合法语言代码

注意: 更完整的类型可参考 W3C 的说明: <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html>。

XML Schema 还允许对现存的简单类型进行重新约束, 从而引出新的简单类型, 新类型的取值范围是现有类型的子集。例如, 对表示文章作者年龄的 age 元素约束是其取值为 20~80 之间的整数, 这样该元素的类型就需要在简单类型 integer 上进行重新约束。首先使用 simpleType 元素对自定义简单类型进行声明, 如下所示。

```
<xs:simpleType name="AgeType">
  </xs:simpleType>
```

这里定义了一个名为 AgeType 的简单类型。然后需要添加对该类型的约束, 通过 restriction 子元素来实现, 如下所示。

```
<xs:simpleType name="AgeType">
  <xs:restriction base="xs:integer">
    <xs:maxInclusive value="80" />
    <xs:minInclusive value="20" />
  </xs:restriction>
</xs:simpleType>
```

restriction 子元素的 base 属性指定了新类型所基于的原始类型, 这里是 integer。然后通过 restriction 的两个子元素 minInclusive 和 maxInclusive 分别制定了该类型的值, 取值范围最小为 20, 最大为 80。定义好一个新类型后, 可以在定义元素时将 type 属性指定为该类型的名字来引用该类型对元素进行约束, 如下所示。

```
<xs:element name="age" type="AgeType" />
```

则在实际 XML 文档中, 所对应的 age 元素的有效取值范围被约束为了 20~88 之间的整数。

XML Schema 为基于不同类型的简单类型提供了丰富的约束规则。例如, 约束类型为 string 的文章地址 url 元素时, 使用了 pattern 元素。pattern 元素允许指定一个正则表达式来对目标元素的值进行约束, 如下所示。

```
<xs:simpleType name="UrlType">
  <xs:restriction base="xs:string">
    <xs:pattern value="http://www.robchen.cn/news/.*\.(html|htm)"></xs:pattern>
  </xs:restriction>
</xs:simpleType>
```

又例如在定义约束表示作者性别的 sex 元素时, 使用以下代码。

```
<xs:simpleType name="SexType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="male" />
    <xs:enumeration value="female" />
  </xs:restriction>
</xs:simpleType>
```

enumeration 元素提供了目标元素取值范围的一个列表, 即 sex 元素的值只能是 male 和 female 其中之一。更多的约束规则可以参考 W3C 的官方文档 <http://www.w3.org/XML/Schema#dev>。

#### 10.4.4 复合类型

复合类型使用 complexType 元素进行定义, 如下所示。

```
<xs:complexType name="Note">
  <xs:sequence>
    <xs:element name="title" type="xs:string" />
    <xs:element name="summary" type="xs:string" />
    <xs:element ref="comment" minOccurs="0" />
    <xs:element name="author" type="AuthorType" maxOccurs="unbounded" />
    <xs:element name="tags" type="TagsType" minOccurs="0" />
    <xs:element name="hit" type="HitType" default="0" minOccurs="0" />
    <xs:element name="url" type="UrlType" />
  </xs:sequence>
  <xs:attribute name="postDate" default="2007-11-17" type="xs:date" />
</xs:complexType>
```

上面的示例定义了一个名为 Note 的复合类型。示例中 complexType 元素包含一个名为 sequence 的子元素, 该元素相当于一个分组, 其作用是指定复合元素中可能出现的子元素及子元素出现的顺序。例子中 sequence 元素下定义了 7 个子元素, 当这些子元素在 XML 文档中出现时, 必须按照其定义的顺序排列。sequence 元素中定义的子元素可以通过 minOccurs 和 maxOccurs 属性来指定其出现的最小次数和最大次数, 默认值均为 1。如果不限制某个子元素的最大出现次数, 则可以将 maxOccurs 属性

值设置为 unbounded。

除了 sequence 外, XML Schema 还提供了另外一种声明子元素的方法, 就是使用 all 元素。使用该元素声明的子元素在实际 XML 文档中的出现顺序将不受约束, 但是所有的子元素最多只能出现 1 次, 例如对表示作者的 author 元素的子元素的声明。

```
<xs:complexType name="AuthorType">
  <xs:all>
    <xs:element name="name" type="xs:string" minOccurs="1" />
    <xs:element name="sex" type="SexType" minOccurs="0" />
    <xs:element name="age" type="AgeType" minOccurs="0" />
  </xs:all>
</xs:complexType>
```

使用复合类型和使用简单类型一样, 只需要将 element 元素的 type 属性指定为相应复合类型的名称即可, 如下所示。

```
<xs:element name="notes" type="Notes">
</xs:element>
```

### 10.4.5 定义属性

当需要为一个元素定义某个属性时, 可以使用 attribute 元素, 例如在 Note 类型的定义中, 包含了对 postDate 属性的定义。

```
<xs:attribute name="postDate" default="2007-11-17" type="xs:date" />
```

其中 name 属性指定了属性的名称, type 指定了元素的类型, default 定义了该属性的默认值 (关于默认值的知识在 10.4.6 小节向读者介绍)。由于属性本身不能包含任何其他元素, 所以属性只能被指定为简单类型。同时属性不能单独存在, 其定义必须存放在复合类型的定义中。

对于属性而言, 它只能出现一次或者根本不出现。所以定义属性的出现约束的语法与元素有所不同, 其使用一个 use 属性来对属性是否出现进行约束, 如下所示。

```
<xs:attribute name="postDate" default="2007-11-17" type="xs:date" use="optional" />
```

use 属性的可选值包含 required、optional 及 prohibited 3 个, 其分别表示属性必须出现、可以出现和不能出现在元素中, 如果 use 被省略, 则默认为 optional。

### 10.4.6 默认值

属性和元素的默认值都通过 default 属性来声明, 但是在使用上有一些区别。声明一个属性的默认值, 如下所示。

```
<xs:attribute name="postDate" default="2007-11-17" type="xs:date" />
```

声明一个属性的默认值后, 如果该属性在 XML 文档中出现了, 则属性值以文档中指定的值为准。如果属性没有出现在 XML 文档中, 则属性值按照 XML Schema 中指定的默认值为准。如下面的 XML

文档片断。

```
<note>
  <title>HTML</title>
  <summary>Learning HTML</summary>
  <author>
    <name>Susan Smith</name>
    <sex>female</sex>
  </author>
  <tags>HTML CSS</tags>
  <hit date="2007-11-17">267</hit>
  <url>http://www.robchen.cn/news/HTML.html</url>
</note>
```

即使 `note` 元素中并没有出现 `postDate` 属性, 但是解析器仍然可以通过 XML Schema 得到一个 `postDate` 的属性值, 为 2007-11-17。

注意: 仅当属性声明的 `use` 属性为 `optional`, 即元素为可选项时, 才能声明默认值, 否则会发生错误。

使用 `default` 属性为元素指定一个默认值时, 其行为与元素的默认值不一样。如果元素在 XML 文档中出现且带有自身的内容, 元素的值就是文档中该元素的内容。如果元素没有内容, 那么解析器就认为这个元素的值等于声明中 `default` 属性的值。然而, 如果元素在 XML 文档中并不出现, 解析器则根本不认为该元素出现。

### 10.4.7 约束特殊值

当需要约束属性或元素的值必须为某个特殊值时, 可以使用 `fixed` 属性。当在某个属性或元素定义中使用 `fixed` 属性指定了一个值, 则该属性出现时, 其值必须为 `fixed` 属性所指定的值。例如在定义一种表示水果的元素中, 需要约束一个名为 `type` 的属性, 如果出现则值一定为 `fruit`, 其 XML Schema 代码如下所示。

```
<xs:attribute name="type" type="xs:string" fixed="fruit"/>
```

注意: `fixed` 属性和 `default` 属性是互斥的, 不能同时使用, 否则会发生错误。

### 10.4.8 列表类型

当某个元素的值为一个列表时, 例如示例中表示标签名的 `tags` 元素, 其值的样式如下所示。

```
<tags>Ajax JavaScript XML JSON</tags>
```

这时可以使用一种比较特殊的简单类型, 叫做列表类型。列表类型使用 `simpleType` 的子元素 `list` 来定义, 如下所示。

```
<xs:simpleType name="TagType">
  <xs:list itemType="xs:token">
  </xs:list>
```



```

</xs:simpleType>
<xs:simpleType name="TagsType">
  <xs:restriction base="TagType">
    <xs:minLength value="0" />
    <xs:maxLength value="10" />
  </xs:restriction>
</xs:simpleType>

```

XML Schema 代码中首先定义了一个基于 token 类型的列表类型 TagType, 然后对 TagType 重新约束并引出了新的列表简单类型 TagsType。TagsType 通过 minLength 和 maxLength 子元素约束了列表项的元素数目最小为 0 个, 最大为 10 个。

#### 10.4.9 联合类型

当某个简单类型的元素, 其元素值可能为多种类型时, 就需要使用联合类型对其进行约束。一个联合类型包含了多个简单类型 (包括列表类型), 受其约束的元素的值可以是这些简单类型中的其中任意一种类型。使用 simpleType 的子元素 union 声明一个联合类型, 如下所示。

```

<xs:simpleType name="TelephoneType">
  <xs:union memberTypes="xs:string xs:integer"/>
</xs:simpleType>
<xs:element name="telephone" type="TelephoneType"/>

```

其中通过 union 元素的 memberTypes 指定了该联合类型所支持的类型集合, 这里包括 string 和 integer, 所以下面这些 XML 片段都是有效的。

```

<telephone>13887654321</telephone>
<telephone>010-87654321</telephone>
<telephone>+86-010-87654321-888</telephone>

```

#### 10.4.10 匿名类型定义

在前面的示例中, 定义元素类型都是单独定义一个具有名称的类型元素, 然后在定义元素时将该名称赋给 type 属性以应用该类型对元素进行约束。这样的好处是提高了类型的复用性, 但是如果某个类型为某个元素所特有时, 这样就显得多此一举了。XML Schema 提供了一种匿名类型定义的方法, 就是将 simpleType 或 complexType 元素省略掉 name 属性然后直接放在元素定义中, 同时省略掉 element 元素的 type 属性。将基本示例中的 XML Schema 文件全部用匿名类型定义, 改写后代码如下所示。

```

<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="XMLSchema1" targetNamespace="http://www.robchen.cn/news"
  elementFormDefault="qualified" xmlns="http://www.robchen.cn/news"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="notes">
    <xs:complexType>
      <xs:sequence>

```



```

<xs:element name="note" minOccurs="0" maxOccurs="unbounded" >
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string" />
      <xs:element name="summary" type="xs:string" />
      <xs:element ref="comment" minOccurs="0" />
      <xs:element name="author" maxOccurs="unbounded">
        <xs:complexType>
          <xs:all>
            <xs:element name="name" type="xs:string" minOccurs="1" />
            <xs:element name="sex" minOccurs="0">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="male" />
                  <xs:enumeration value="female" />
                </xs:restriction>
              </xs:simpleType>
            </xs:element>
            <xs:element name="age" minOccurs="0">
              <xs:simpleType>
                <xs:restriction base="xs:integer">
                  <xs:maxInclusive value="80" />
                  <xs:minInclusive value="20" />
                </xs:restriction>
              </xs:simpleType>
            </xs:element>
          </xs:all>
        </xs:complexType>
      </xs:element>
      <xs:element name="tags" minOccurs="0">
        <xs:simpleType>
          <xs:restriction base="TagType">
            <xs:minLength value="0" />
            <xs:maxLength value="10" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="hit" default="0" minOccurs="0">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:nonNegativeInteger">
              <xs:attribute name="date" type="xs:date" use="required" />
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:element name="url">
        <xs:simpleType>
          <xs:restriction base="xs:string">

```

```

        <xs:pattern value="http://www.robchen.cn/news/.*\.(html|htm)"></xs:pattern>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
</xs:sequence>
  <xs:attribute name="postDate" default="2007-11-17" type="xs:date" />
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="comment" type="xs:string">
</xs:element>
<xs:simpleType name="TagType">
  <xs:list itemType="xs:token">
  </xs:list>
</xs:simpleType>
</xs:schema>

```

#### 10.4.11 简单的复合类型

一些 XML 元素既有简单类型元素的特点,但又不是简单元素,例如表示文章单击数的 hit 元素。

```
<hit date="2007-11-17">854</hit>
```

该元素和简单类型元素一样包含简单类型值,但是又拥有一个 date 属性。简单类型是不能拥有属性的,所以这里需要使用一个基于简单类型的复合类型来定义这个元素,使用 complexType 的 simpleContent 子元素来实现,如下所示。

```

<xs:complexType name="HitType">
  <xs:simpleContent>
    <xs:extension base="xs:nonNegativeInteger">
      <xs:attribute name="date" type="xs:date" use="required" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

```

simpleContent 元素中包含一个 extension 元素,通过其 base 属性指定所约束的元素值的类型,然后再声明一个 date 类型的属性,并通过 use 属性设置为必须出现。到这里,就定义好了一个简单的复合类型。

#### 10.4.12 混合内容

前面向读者介绍的所有约束形式,都没有包含类似于下面这种混合内容的表现形式。

```

<letter>
  <salutation>Dear Mr.<name>John Smith</name>.</salutation>,Your order of <quantity>1</quantity>
  <productName>Computer</productName> shipped from our warehouse on

```

```
<shipDate>2007-11-18</shipDate>. ....
</letter>
```

字符数据并没有仅仅被包含在最深处的元素中，而是与元素的子元素混合在一起。当使用 XML Schema 来定义这个类型的元素时，只需要将复合类型 `complexType` 元素的属性 `mixed` 设置为 `true` 即可，如下所示。

```
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="salutation">
        <xs:complexType mixed="true">
          <xs:sequence>
            <xs:element name="name" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="quantity" type="xs:positiveInteger"/>
      <xs:element name="productName" type="xs:string"/>
      <xs:element name="shipDate" type="xs:date" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

#### 10.4.13 任意类型

XML Schema 提供了一种特殊的类型：`anyType`，当使用该类型来定义某个元素时，表示对该元素值的类型不作任何限制，如下所示。

```
<xs:element name="comment" type="xs:anyType"/>
```

其实，如果不声明 `element` 元素的 `type` 属性，其 `type` 默认值就为 `anyType`，如下所示。

```
<xs:element name="comment"/>
```

#### 10.4.14 分组和引用

XML Schema 提供了引用机制来简化 XML Schema 的开发，回顾基本示例中的示例代码，在复合类型 `Note` 的定义中有如下代码。

```
<xs:element ref="comment" minOccurs="0" />
```

在这个元素定义并没有声明元素的名称和类型，而是使用 `ref` 属性来引用了一个已经定义好的元素，如下所示。

```
<xs:element name="comment" type="xs:string">
  </xs:element>
```

通过 ref 引用, 来继承一个已经定义好的元素的相关约束。引用不但可以作用于元素, 还可以作用于属性, 如下所示。

```
<xs:attribute name="postDate" type="xs:date">
</xs:attribute>
<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string"></xs:element>
    </xs:sequence>
    <xs:attribute ref="postDate"></xs:attribute>
  </xs:complexType>
</xs:element>
```

通过 ref 所引用的元素或者属性, 必须是全局定义的, 即该元素或者属性的定义不是包含在复合类型定义中, 而是单独出现的。

当需要一次性引用多个元素或者属性时, 可以先使用 group 和 attributeGroup 将多个元素或者属性的定义进行分组, 然后对组进行引用, 如下所示。

```
<xs:group name="NoteElementsGroup">
  <xs:sequence>
    <xs:element name="title" type="xs:string"></xs:element>
    <xs:element name="summary" type="xs:string"></xs:element>
    <xs:element name="comment" type="xs:string"></xs:element>
  </xs:sequence>
</xs:group>
<xs:attributeGroup name="NoteAttributesGroup">
  <xs:attribute name="postDate" type="xs:date" use="required"></xs:attribute>
  <xs:attribute name="other" default="nothing"></xs:attribute>
</xs:attributeGroup>

<xs:complexType name="Note">
  <xs:sequence>
    <xs:group ref="NoteElementsGroup"></xs:group>
    <xs:element name="author" type="AuthorType"></xs:element>
  </xs:sequence>
  <xs:attributeGroup ref="NoteAttributesGroup"></xs:attributeGroup>
</xs:complexType>
```

group 与 attributeGroup 都必须全局定义。attributeGroup 与 group 不同的是不能使用 all、sequence 等约束其包含元素出现顺序的相关方法。

### 10.4.15 命名冲突

在 XML Schema 中, 如果给两个对象定义了同样的名字, 则可能会产生命名冲突。一般而言, 两个对象越接近, 冲突的可能性越大。下面列出了可能发生命名冲突的一些情况。

- ❑ 如果两个对象都是自定义类型，当定义的名字一样时，就会发生命名冲突。
- ❑ 如果两个对象是类型和元素或者类型和属性，则不会发生命名冲突。
- ❑ 如果两个对象是不同类型的非全局元素，例如有两个不同的复合类型的定义中都包含一个名为 item 的元素，此时不会发生命名冲突。
- ❑ 如果两个对象都是类型，一个是内置类型，一个是自定义类型，则即使名字一样也不会发生命名冲突。

#### 10.4.16 关联 XML 与 XML Schema

当定义好一个 XML Schema 后，需要将其与受约束的 XML 文档进行关联。回顾基本示例中 XML 文档和 XML Schema 文档的根节点中的各项属性声明，如下所示。

XML 文档根节点如下所示。

```
<notes xmlns="http://www.robchen.cn/news" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.robchen.cn/news newsSchema.xsd">
```

XML Schema 文档根节点如下所示。

```
<xs:schema id="XMLSchema1" targetNamespace="http://www.robchen.cn/news"
elementFormDefault="qualified" xmlns="http://www.robchen.cn/news"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

XML 文档在根节点中声明了默认的命名空间，以及一个指向 <http://www.w3.org/2001/XMLSchema-instance> 的命名空间 xsi，然后通过 schemaLocation 声明了与该文档关联的 XML Schema 文档的作用命名空间和地址，分别为 <http://www.robchen.cn/news> 和 newsSchema.xsd。这里的 <http://www.robchen.cn/news> 与 XML Schema 文档根节点中的 targetNamespace 属性相对应。XML Schema 地址的声明可以是本地路径、相对路径或者 URI 地址。

将 XML 与 XML Schema 关联后，就可以使用相关的工具或程序对 XML 进行验证，例如可以使用 dotNet Framework 中 System.Xml.Schema 命名空间下的 XmlValidatingReader 类对 XML 进行有效性验证。

```
try
{
    XmlDocument doc = new XmlDocument();
    XmlTextReader rd = new XmlTextReader(xmlfilepath);
    XmlValidatingReader reader = new XmlValidatingReader(rd);
    doc.Load(reader);
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

当验证失败时，会抛出一个包含验证错误信息的异常。



## 10.5 XML DOM

XML DOM 即 XML 文档对象模型, 其定义了 XML 文档对象的各种属性以及提供了针对各种操作的支持。在第 4 章中已经向读者介绍过了 DOM 的基本结构、节点的类型和引用以及一些常规操作。本节将针对 XML DOM 的一些特性进行补充说明。

### 10.5.1 创建 XML DOM 对象

不同的语言和环境下创建 XML DOM 对象的方式有所不同, 本节主要向读者介绍在浏览器环境中使用 JavaScript 创建 XML DOM 对象。由于 IE 和其他非 IE 核心的浏览器在 XML DOM 的实现上有所不同, 所以要分开处理。首先来看 IE 下 XMLDOM 的创建, IE 对 XML DOM 的支持是通过 ActiveXObject 实现的, 如下所示。

```
var xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
```

因 ActiveXObject 对象所依赖的 MSXML 库的版本会对对象创建工作带来影响, 所以与 XMLHttpRequest 对象的创建类似, IE 下一个完整的支持多个版本的 XML DOM 对象创建方法如下所示。

```
var versions = ["MSXML2.DOMDocument.5.0", "MSXML2.DOMDocument.4.0", "MSXML2.DOMDocument.3.0",  
"MSXML2.DOMDocument", "Microsoft.XmlDom"];  
for(var i = 0; i < versions.length; i++)  
{  
    try  
    {  
        var xmlDoc = new ActiveXObject(versions[i]);  
        break;  
    }  
    catch(e){}  
}
```

在非 IE 核心的浏览器中创建 XML DOM 对象, 则使用 implementation 对象的 createDocument 方法来实现, 如下所示。

```
var xmlDoc = document.implementation.createDocument("", null);
```

该方法接受 3 个参数, 第一个参数为默认命名空间, 第二个参数为根元素的标签名, 第三个参数为文档类型对象 (总为 null)。

### 10.5.2 加载 XML 文档

定义好一个 XML DOM 对象后, 可以使用该对象的 load 方法来加载一个已经存在的文档, 其语法如下所示。

```
xmlDoc.load("news.xml");
```

load 方法的参数指明了需要加载的 XML 文档的路径和文件名。

### 1. 同步加载

XML 文档的加载与 XMLHttpRequest 对象的请求一样，有同步和异步的概念，通过 DOM 对象的 async 属性可以读取和设置，默认值为 true，即异步加载。可以通过手工设置将其更改为同步加载，如下所示。

```
xmlDoc.async = false;
```

使用同步加载，在 load 语句之后就可以马上使用所加载的 XML 数据，但是在加载过程中浏览器会停止响应。现在来看一个同步加载的示例，先创建一个名为 book.xml 的 XML 文件，代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<books>
  <book>
    <name>Study AJAX with me</name>
    <author>Robin Chen</author>
    <price>$20.00</price>
  </book>
  <book>
    <name>JavaScript Guide</name>
    <author>Robin Chen</author>
    <price>$14.00</price>
  </book>
</books>
```

然后创建一个包含 JavaScript 脚本的 HTML 页面，来读取这个 XML，并使用对话框输出 XML 文件的内容，代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>XML DOM demo</title>
<script type="text/javascript">
window.onload = function()
{
  try
  {
    if(window.ActiveXObject) //根据浏览器是否支持 ActiveXObject 来判断是否是 IE
    {
      var versions = ["MSXML2.DOMDocument.5.0",
"MSXML2.DOMDocument.4.0","MSXML2.DOMDocument.3.0",
"MSXML2.DOMDocument","Microsoft.XmlDom"];
      //定义不同版本的 MSXML 名称
      for(var i = 0; i < versions.length; i++)
      {
        try
        {
```

```

        var xmlDoc = new ActiveXObject(versions[i]);           //创建 XML DOM 对象
        break;
    }
    catch(e){}
}
}
else //如果不是 IE, 则使用 createDocument 方法来创建
{
    var xmlDoc = document.implementation.createDocument("",null); //创建 XML DOM 对象
}
if(typeof xmlDoc == 'undefined') //如果上述方法都没有创建成功, 则 xmlDoc 未定义
{
    throw new Error("XML DOM Is not supported by your browser!"); //此时抛出异常, 提示浏览器不支持 XML DOM
}
xmlDoc.async = false; //设置加载模式为同步加载
xmlDoc.load('book.xml'); //加载 book.xml 文件
alert(parseXML(xmlDoc)); //将 xmlDoc 交给 parseXML 解析, 然后弹出对话框显示返回结果
}
catch(e)
{
    alert(e.message);
}
}
/*
 * 将 XML DOM 对象所包含的 XML 数据转化成字符串类型
 */
function parseXML(doc)
{
    /*
     * 如果是 IE, 则使用根节点的 xml 属性输出字符串
     */
    if(window.ActiveXObject)
    {
        return doc.documentElement.xml;
    }
    /*
     * 如果不是 IE, 则使用 XMLSerializer 对象的 serializeToString 方法输出字符串
     */
    else
    {
        var oSerializer = new XMLSerializer();
        return oSerializer.serializeToString(doc,'text/xml');
    }
}
</script>
</head>

<body>
</body>
</html>

```

运行结果如图 10.1 所示。

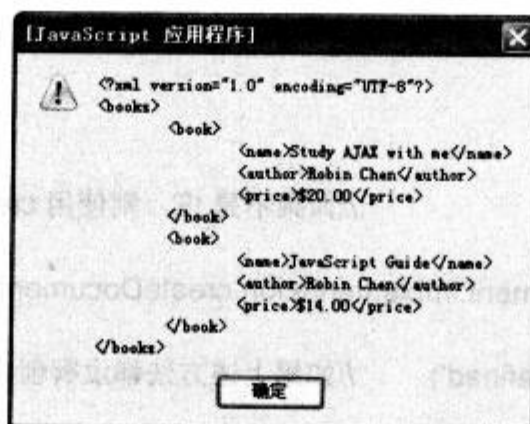


图 10.1 同步加载 XML 文件

## 2. 异步加载

异步加载的概念与 XMLHttpRequest 的异步请求一样，但是在 IE 核心和非 IE 核心的浏览器下的具体实现方式有所不同。在 IE 下，XML DOM 对象提供了类似 XMLHttpRequest 对象的 readyState 事件和 onreadystatechange 属性来控制完成状态。但是在非 IE 核心的浏览器中并没有提供 readyState 和 onreadystatechange，而是提供了一个 onload 事件，在文档被加载完成时触发。将上面同步加载的例子修改为异步加载，代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>XML DOM demo</title>
<script type="text/javascript">
window.onload = function()
{
    try
    {
        if(window.ActiveXObject) //根据浏览器是否支持 ActiveXObject 来判断是否是 IE
        {
            var versions = ["MSXML2.DOMDocument.5.0",
"MSXML2.DOMDocument.4.0", "MSXML2.DOMDocument.3.0",
"MSXML2.DOMDocument", "Microsoft.XmlDom"];
            //定义不同版本的 MSXML 名称

            for(var i = 0; i < versions.length; i++)
            {
                try
                {
                    var xmlDoc = new ActiveXObject(versions[i]); //创建 XML DOM 对象
                    xmlDoc.onreadystatechange = function() //为 xmlDoc 的 readystatechange 事件
                    {
                        if(xmlDoc.readyState == 4) //当完成状态为 4 时，数据加载完成
                        {
                            //...
                        }
                    }
                }
            }
        }
    }
}
</script>
</html>
```

注册一个匿名的处理函数

```

        alert(parseXML(xmlDoc));    //将 xmlDoc 交给 parseXML 解析, 然后弹出
    }
    }
    break;
}
catch(e){}
}
}
else    //如果不是 IE, 则使用 createDocument 方法来创建
{
    var xmlDoc = document.implementation.createDocument("", null);    //创建 XML DOM 对象
    xmlDoc.onload = function()    //为 xmlDoc 的 load 事件注册一个匿名的处理函数
    {
        alert(parseXML(xmlDoc));    //将 xmlDoc 交给 parseXML 解析, 然后弹出对话框显示返回
    }
}
if(typeof xmlDoc == 'undefined')    //如果上述方法都没有创建成功, 则 xmlDoc 未定义
{
    throw new Error('XML DOM is not supported by your browser!');    //此时抛出异常, 提示浏览器不支持 XML DOM
}
xmlDoc.load('book.xml');    //加载 book.xml 文件
}
catch(e)
{
    alert(e.message);
}
}
/*
 * 将 XML DOM 对象所包含的 XML 数据转化成字符串类型
 */
function parseXML(doc)
{
    /*
     * 如果是 IE, 则使用根节点的 xml 属性输出字符串
     */
    if(window.ActiveXObject)
    {
        return doc.documentElement.xml;
    }
    /*
     * 如果不是 IE, 则使用 XMLSerializer 对象的 serializeToString 方法输出字符串
     */
    else
    {
        var oSerializer = new XMLSerializer();
        return oSerializer.serializeToString(doc, 'text/xml');
    }
}

```

对话框显示返回结果

结果



```

    }
  }
</script>
</head>

<body>
</body>
</html>

```

### 3. 加载失败

当加载一个 XML 文档失败时, 在 IE 下会将错误信息写入到 XML DOM 对象的 `parseError` 属性中, 在非 IE 核心的浏览器中, 则是将错误信息作为节点写入到 XML DOM 对象中, 代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>XML DOM demo</title>
<script type="text/javascript">
window.onload = function()
{
    try
    {
        if(window.ActiveXObject)                //根据浏览器是否支持 ActiveXObject 来判断是否是 IE
        {
            var versions = ["MSXML2.DOMDocument.5.0",
"MSXML2.DOMDocument.4.0", "MSXML2.DOMDocument.3.0",
"MSXML2.DOMDocument", "Microsoft.XmlDom"];
            //定义不同版本的 MSXML 名称
            for(var i = 0; i < versions.length; i++)
            {
                try
                {
                    var xmlDoc = new ActiveXObject(versions[i]);    //创建 XML DOM 对象
                    break;
                }
                catch(e){}
            }
        }
        else                                //如果不是 IE, 则使用 createDocument 方法来创建
        {
            var xmlDoc = document.implementation.createDocument("", null);    //创建 XML DOM 对象
        }
        if(typeof xmlDoc == 'undefined')    //如果上述方法都没有创建成功, 则 xmlDoc 未定义
        {
            throw new Error("XML DOM Is not supported by your browser!");    //此时抛出异常, 提示浏览器不支持 XML DOM
        }
        xmlDoc.async = false;
    }
}

```

```

xmlDoc.load("book123.xml");           //加载一个不存在的 book.xml 文件以引发加载错误
if(window.ActiveXObject)
{
    if(xmlDoc.parseError != 0)         //判断是否发生错误
    {
        var oError = xmlDoc.parseError;           //输出错误信息
        alert("An error occurred:\n 错误代码: "
            + oError.errorCode + "\n"
            + "行数: " + oError.line + "\n"
            + "列数: " + oError.linepos + "\n"
            + "原因: " + oError.reason);
        return;
    }
}
else
{
    alert(parseXML(xmlDoc));           //输出 XML 内容
}
}
catch(e)
{
    alert(e.message);
}
}
/*
 * 将 XML DOM 对象所包含的 XML 数据转化成字符串类型
 */
function parseXML(doc)
{
    /*
     * 如果是 IE, 则使用根节点的 xml 属性输出字符串
     */
    if(window.ActiveXObject)
    {
        return doc.documentElement.xml;
    }
    /*
     * 如果不是 IE, 则使用 XMLSerializer 对象的 serializeToString 方法输出字符串
     */
    else
    {
        var oSerializer = new XMLSerializer();
        return oSerializer.serializeToString(doc,'text/xml');
    }
}
</script>
</head>

<body>
</body>
</html>

```

程序中使用 `xmlDoc` 对象去 `load` 一个不存在的 XML 文档，以此引发错误，然后根据浏览器的不同采用不同的方式输出错误信息，在 IE 下输出的错误信息如图 10.2 所示。

在 Firefox 下输出的错误信息如图 10.3 所示。

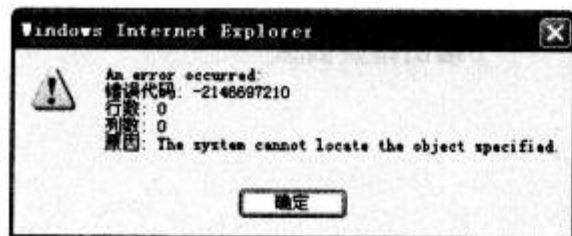


图 10.2 IE 下的错误信息

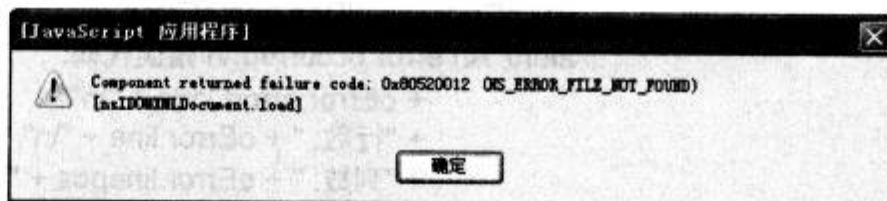


图 10.3 Firefox 下的错误信息

### 10.5.3 加载 XML 片段

除了载入整个文档外，XML DOM 还允许载入一个符合 XML 语法规则的字符串片段。在 IE 下，直接使用 `loadXML` 方法即可，而在非 IE 核心的浏览器下，则需要使用 `DOMParser` 对象来辅助加载，代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>XML DOM demo</title>
<script type="text/javascript">
window.onload = function()
{
    try
    {
        if(window.ActiveXObject) //根据浏览器是否支持 ActiveXObject 来判断是否是 IE
        {
            var versions = ["MSXML2.DOMDocument.5.0",
"MSXML2.DOMDocument.4.0","MSXML2.DOMDocument.3.0",
"MSXML2.DOMDocument","Microsoft.XmlDom"]; //定义不同版本的 MSXML 名称
            for(var i = 0; i < versions.length; i++)
            {
                try
                {
                    var xmlDoc = new ActiveXObject(versions[i]); //创建 XML DOM 对象
                    break;
                }
                catch(e){}
            }
        }
        else //如果不是 IE，则使用 createDocument 方法来创建
        {
            var xmlDoc = document.implementation.createDocument("",null); //创建 XML DOM 对象
        }
    }
}
```

```

        if(typeof xmlDoc == 'undefined')           //如果上述方法都没有创建成功, 则 xmlDoc 未定义
        {
            throw new Error('XML DOM Is not supported by your browser!');    //此时抛出异常, 提示浏览器不支持 XML DOM
        }
        xmlDoc.async = false;
        var xmlString = '<books>'
            + '<book>'
            + '<name>Study AJAX with me</name>'
            + '<author>Robin Chen</author>'
            + '<price>$20.00</price>'
            + '</book>'
            + '<book>'
            + '<name>JavaScript Guide</name>'
            + '<author>Robin Chen</author>'
            + '<price>$14.00</price>'
            + '</book>'
            + '</books>';

        try
        {
            xmlDoc.loadXML(xmlString);           //加载一个 xml 片断
        }
        catch(e)
        {
            var oParser = new DOMParser();       //定义 DOMParser 对象
            xmlDoc = oParser.parseFromString(xmlString,'text/xml');    //加载 XML 片断
        }
        alert(parseXML(xmlDoc));                 //输出 xml 内容
    }
    catch(e)
    {
        alert(e.message);
    }
}
/*
 * 将 XML DOM 对象所包含的 XML 数据转化成字符串类型
 */
function parseXML(doc)
{
    /*
     * 如果是 IE, 则使用根节点的 xml 属性输出字符串
     */
    if(window.ActiveXObject)
    {
        return doc.documentElement.xml;
    }
    /*
     * 如果不是 IE, 则使用 XMLSerializer 对象的 serializeToString 方法输出字符串
     */

```

```
else
{
    var oSerializer = new XMLSerializer();
    return oSerializer.serializeToString(doc,'text/xml');
}
}
</script>
</head>

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

程序在 IE 和 Firefox 下运行结果一样, 如图 10.4 所示。

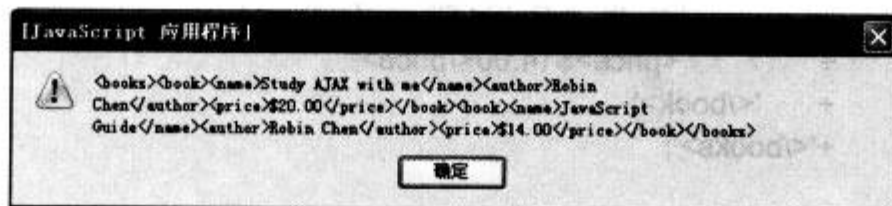


图 10.4 运行结果

#### 10.5.4 取得 XML 内容

回顾之前示例中输出 XML 内容的 parseXML 函数, 其内容如下所示。

```
function parseXML(doc)
{
    /*
    * 如果是 IE, 则使用根节点的 xml 属性输出字符串
    */
    if(window.ActiveXObject)
    {
        return doc.documentElement.xml;
    }
    /*
    * 如果不是 IE, 则使用 XMLSerializer 对象的 serializeToString 方法输出字符串
    */
    else
    {
        var oSerializer = new XMLSerializer();
        return oSerializer.serializeToString(doc,'text/xml');
    }
}
```

其工作原理是这样的: 在 IE 中的 XML DOM 中, 每个 DOM 对象都有一个 xml 属性, 其返回一个包含了当前元素和其所有子孙元素内容的字符串表示。而在非 IE 核心的浏览器中, 同样的功能则需要使用 XMLSerializer 对象的 serializeToString 方法来实现。



## 10.6 强大的检索工具: XPath

要获得 DOM 元素的引用,除了在第 4 章中向读者介绍的一些基本方法外,还可以使用 XPath。XPath 是一种在 XML 文档中查找信息的语言,通过元素和属性进行导航。XPath 被设计供 XSLT、XPoint 以及其他 XML 解析器使用,并于 1999 年 11 月 16 日成为 W3C 标准。熟练掌握 XPath 能够让开发者对 XML 的操作得心应手。本节将结合实例对 XPath 的语法和使用方法做详细介绍。

### 10.6.1 基本示例

在介绍 XPath 的语法前,先来看一个 XPath 的应用示例。有一个表示图书列表信息的 books.xml 文档,代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<books>
  <book category="sport">
    <name lang="en">Play Basketball with me</name>
    <author>Jack Robinson</author>
    <year>2007</year>
    <price>30.00</price>
  </book>
  <book category="it">
    <name lang="cn">Ajax 从入门到精通</name>
    <author>Robin Chen</author>
    <year>2007</year>
    <price>60.00</price>
  </book>
  <book category="cooking">
    <name lang="en">Nice Cookie</name>
    <author>Susan Smith</author>
    <year>2007</year>
    <price>15.00</price>
  </book>
  <book category="sport">
    <name lang="en">Running like Forest Gump</name>
    <author>Jim Jackson</author>
    <year>2007</year>
    <price>35.00</price>
  </book>
  <book category="cooking">
    <name lang="en">The Best Chicken</name>
    <author>Susan Smith</author>
    <year>2006</year>
    <price>20.00</price>
  </book>
</books>
```

该文档的根元素为 `books`，其包含多个 `book` 元素。每个 `book` 元素表示一个图书信息。`book` 元素的 `category` 属性表示该图书是属于哪个种类，例如 `sport`（运动）、`it`（信息技术）、`cooking`（烹饪）等。`book` 元素包含 4 个子元素，分别是 `name`（书名）、`author`（作者）、`year`（出版年份）和 `price`（价格）。其中 `name` 包含 `lang` 属性，表示该图书所使用的语言。

在第 4 章中已经向读者介绍过通过标签名引用元素（`getElementsByName`）、根据已有元素和元素关系引用元素（`parentNode`、`firstChild` 等）。当需要对元素数据进行查找，例如需要得到所有价格大于或者等于 30.00 的书籍，这时就需要对所有的 `book` 元素进行遍历然后对 `price` 元素的值进行判断，以此来获取符合要求的元素，代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>XPath simple demo</title>
<script type="text/javascript">
window.onload = function()
{
    try
    {
        if(window.ActiveXObject)           //根据浏览器是否支持 ActiveXObject 来判断是否是 IE
        {
            var versions = ["MSXML2.DOMDocument.5.0",
"MSXML2.DOMDocument.4.0","MSXML2.DOMDocument.3.0",
"MSXML2.DOMDocument","Microsoft.XmlDom"];           //定义不同版本的 MSXML 名称
            for(var i = 0; i < versions.length; i++)
            {
                try
                {
                    var xmlDoc = new ActiveXObject(versions[i]);           //创建 XML DOM 对象
                    break;
                }
                catch(e){}
            }
        }
        else                               //如果不是 IE，则使用 createDocument 方法来创建
        {
            var xmlDoc = document.implementation.createDocument("",null);           //创建 XML DOM 对象
        }
        if(typeof xmlDoc == 'undefined')           //如果上述方法都没有创建成功，则 xmlDoc 未定义
        {
            throw new Error("XML DOM Is not supported by your browser!");           //此时抛出异常，提示浏览器不支持 XML DOM
        }
        xmlDoc.async = false;           //设置加载模式为同步加载
        xmlDoc.load("books.xml");           //加载 books.xml 文件
        var xmlString = "";           //初始化 xml 输出字符串
```

```

var books = xmlDoc.documentElement.getElementsByTagName("book"); //取得所有 book 元素
for(var i = 0; i < books.length; i++) //遍历 book 元素
{ //取得 price 值
    var price = books[i].getElementsByTagName("price")[0].firstChild.nodeValue;
    if(parseFloat(price) >= 30) //使用 parseFloat 将 price 转化成 float 类型并与 30 比较
    {
        xmlString += parseXML(books[i]) + '\n'; //当价格大于 30 时, 将节点数据转化成字符串并拼接到 xmlString 中
    }
}
alert(xmlString); //输出 xml 字符串
}
catch(e)
{
    alert(e.message);
}
}
/*
 * 将 XML DOM 对象所包含的 XML 数据转化成字符串类型
 */
function parseXML(doc)
{
    /*
     * 如果是 IE, 则使用节点的 xml 属性输出字符串
     */
    if(window.ActiveXObject)
    {
        return doc.xml; //这里 doc 本身是节点, 所以直接使用 xml 属性
    }
    /*
     * 如果不是 IE, 则使用 XMLSerializer 对象的 serializeToString 方法输出字符串
     */
    else
    {
        var oSerializer = new XMLSerializer();
        return oSerializer.serializeToString(doc, 'text/xml');
    }
}
</script>
</head>

<body>
</body>
</html>

```

程序遍历所有的 book 元素, 将符合要求的元素提交给 parseXML 函数处理, 并将返回的字符串拼接到变量 xmlString 中, 最后使用对话框输出 xmlString 的内容, 来显示符合要求的所有 book 元素。其运行结果如图 10.5 所示。

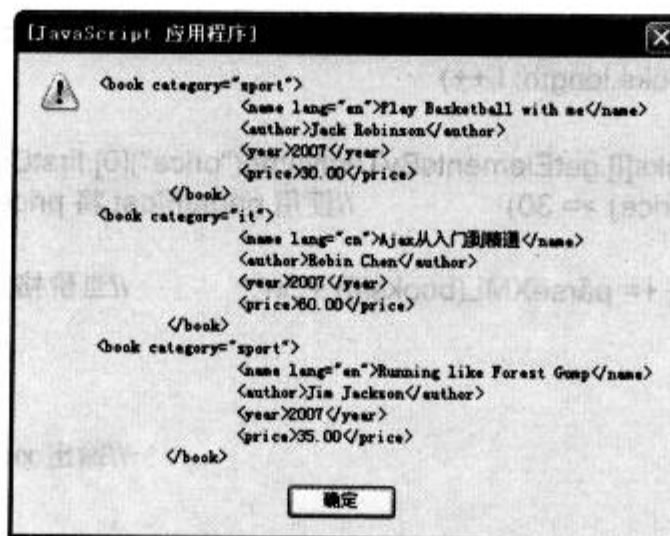


图 10.5 使用遍历查找

这种查找数据的方式在 XML 文档所包含的数据量比较小时,可以获得不错的效率,但当数据量很大,或者数据结构很复杂时,就不太适用了。下面再来看看如何使用 XPath 来实现同样的功能,代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>XPath simple demo</title>
<script type="text/javascript">
if( document.implementation.hasFeature("XPath", "3.0") )
{
XMLDocument.prototype.selectNodes = function(cXPathString, xNode)
{
    if( !xNode ) { xNode = this; }
    var oNSResolver = this.createNSResolver(this.documentElement)
    var altems = this.evaluate(cXPathString, xNode, oNSResolver,
        XPathResult.ORDERED_NODE_SNAPSHOT_TYPE, null)
    var aResult = [];
    for( var i = 0; i < altems.snapshotLength; i++)
    {
        aResult[i] = altems.snapshotItem(i);
    }
    return aResult;
}
}
window.onload = function()
{
    try
    {
        if(window.ActiveXObject) //根据浏览器是否支持 ActiveXObject 来判断是否是 IE
        {
            var versions = ["MSXML2.DOMDocument.5.0",
```

```

"MSXML2.DOMDocument.4.0","MSXML2.DOMDocument.3.0",
"MSXML2.DOMDocument","Microsoft.XmlDom"]; //定义不同版本的 MSXML 名称
    for(var i = 0; i < versions.length; i++)
    {
        try
        {
            var xmlDoc = new ActiveXObject(versions[i]); //创建 XML DOM 对象
            break;
        }
        catch(e){}
    }
}
else //如果不是 IE, 则使用 createDocument 方法来创建
{
    var xmlDoc = document.implementation.createDocument("",null); //创建 XML DOM 对象
}
if(typeof xmlDoc == 'undefined') //如果上述方法都没有创建成功, 则 xmlDoc 未定义
{
    throw new Error('XML DOM Is not supported by your browser!'); //此时抛出异常, 提示浏览器不支持 XML DOM
}
xmlDoc.async = false; //设置加载模式为同步加载
xmlDoc.load('books.xml'); //加载 books.xml 文件
//console.log(xmlDoc);
var xmlString = ""; //初始化 xml 输出字符串
var books = xmlDoc.selectNodes("//book[price>=30]"); //取得所有 price 大于 30 的 book 元素
for(var i = 0; i < books.length; i++) //遍历得到的 book 元素
{
    xmlString += parseXML(books[i]) + "\n"; //拼接字符串
}
alert(xmlString); //输出 xml 字符串
}
catch(e)
{
    alert(e.message);
}
}
/*
* 将 XML DOM 对象所包含的 XML 数据转化成字符串类型
*/
function parseXML(doc)
{
    /*
    * 如果是 IE, 则使用节点的 xml 属性输出字符串
    */
    if(window.ActiveXObject)
    {
        return doc.xml; //这里 doc 本身是节点, 所以直接使用 xml 属性
    }
}

```



```

/*
 * 如果不是 IE，则使用 XMLSerializer 对象的 serializeToString 方法输出字符串
 */
else
{
    var oSerializer = new XMLSerializer();
    return oSerializer.serializeToString(doc,'text/xml');
}
}
</script>
</head>

<body>
</body>
</html>

```

运行结果与遍历法一样，如图 10.6 所示。

程序中使用 `xmlDoc.selectNodes("//book[price>=30]")` 一句就完成了对所有价格大于 30 的 book 节点的搜索，其中 `"//book[price>=30]"` 就是标准的 XPath 表达式，表示选取所有 price 子元素的节点值大于 30 的 book 元素。selectNodes 方法接受一个 XPath 表达式，并返回符合表达式的所有节点的集合。selectNodes 被 IE 支持，但是不被 Firefox 等非 IE 核心的浏览器支持。Firefox 下实现同等功能的方式不一样，为了使用同一方法简化编码，在上面的示例中采用了原型扩展的方法来让 Firefox 也支持 selectNodes 方法，代码如下所示。

```

if( document.implementation.hasFeature("XPath", "3.0") )
{
    XMLDocument.prototype.selectNodes = function(cXPathString, xNode)
    {
        if( !xNode ) { xNode = this; }
        var oNSResolver = this.createNSResolver(this.documentElement)
        var altems = this.evaluate(cXPathString, xNode, oNSResolver,
                                   XPathResult.ORDERED_NODE_SNAPSHOT_TYPE, null)
        var aResult = [];
        for( var i = 0; i < altems.snapshotLength; i++)
        {
            aResult[i] = altems.snapshotItem(i);
        }
        return aResult;
    }
}

```

关于原型的相关知识，本书将在第 12 章：JavaScript 面向对象编程（OOP）中向读者介绍。

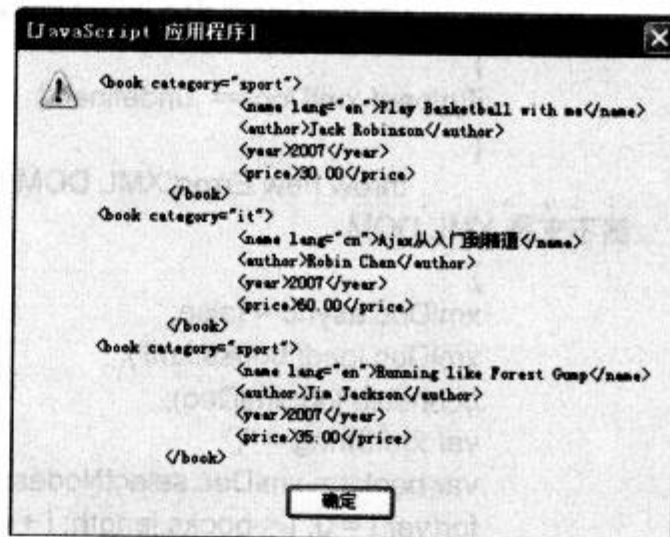


图 10.6 使用 XPath 选取元素

还有一个可以使用 XPath 表达式的方法是 `selectSingleNode`，它接受一个 XPath 表达式，返回符合该表达式的第一个元素。例如，需要找出图书列表中第一本语言为 `cn`（中文）的图书信息，代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>XPath simple demo</title>
<script type="text/javascript">
if( document.implementation.hasFeature("XPath", "3.0") )
{
XMLDocument.prototype.selectNodes = function(cXPathString, xNode)
{
    if( !xNode ) { xNode = this; }
    var oNSResolver = this.createNSResolver(this.documentElement)
    var aItems = this.evaluate(cXPathString, xNode, oNSResolver,
        XPathResult.ORDERED_NODE_SNAPSHOT_TYPE, null)
    var aResult = [];
    for( var i = 0; i < aItems.snapshotLength; i++)
    {
        aResult[i] = aItems.snapshotItem(i);
    }
    return aResult;
}
XMLDocument.prototype.selectSingleNode = function(cXPathString, xNode)
{
    if( !xNode ) { xNode = this; }
    var xItems = this.selectNodes(cXPathString, xNode);
    if( xItems.length > 0 )
    {
        return xItems[0];
    }
    else
    {
        return null;
    }
}
}
window.onload = function()
{
    try
    {
        if(window.ActiveXObject) //根据浏览器是否支持 ActiveXObject 来判断是否是 IE
        {
            var versions = ["MSXML2.DOMDocument.5.0",
                "MSXML2.DOMDocument.4.0", "MSXML2.DOMDocument.3.0",
                "MSXML2.DOMDocument", "Microsoft.XmlDom"]; //定义不同版本的 MSXML 名称
```

```

        for(var i = 0; i < versions.length; i++)
        {
            try
            {
                var xmlDoc = new ActiveXObject(versions[i]);    //创建 XML DOM 对象
                break;
            }
            catch(e){}
        }
    }
    else    //如果不是 IE，则使用 createDocument 方法来创建
    {
        var xmlDoc = document.implementation.createDocument("",null);    //创建 XML DOM 对象
    }
    if(typeof xmlDoc == 'undefined')    //如果上述方法都没有创建成功，则 xmlDoc 未定义
    {
        throw new Error("XML DOM is not supported by your browser!");    //此时抛出异常，提示浏览器不支持 XML DOM
    }
    xmlDoc.async = false;    //设置加载模式为同步加载
    xmlDoc.load('books.xml');    //加载 books.xml 文件
    var book = xmlDoc.selectSingleNode("//book[name[@lang='cn']]");    //取得第一个 name 子元素的 lang 属性为'cn'的 book 元素
    var xmlString = parseXML(book);    //转化成字符串
    alert(xmlString);    //输出 xml 字符串
}
catch(e)
{
    alert(e.message);
}
}
/*
 * 将 XML DOM 对象所包含的 XML 数据转化成字符串类型
 */
function parseXML(doc)
{
    /*
     * 如果是 IE，则使用节点的 xml 属性输出字符串
     */
    if(window.ActiveXObject)
    {
        return doc.xml;    //这里 doc 本身是节点，所以直接使用 xml 属性
    }
    /*
     * 如果不是 IE，则使用 XMLSerializer 对象的 serializeToString 方法输出字符串
     */
    else
    {
        var oSerializer = new XMLSerializer();
    }
}

```

```

        return oSerializer.serializeToString(doc,'text/xml');
    }
}
</script>
</head>

<body>
</body>
</html>

```

程序运行结果如图 10.7 所示。

程序中传递给 `selectSingleNode` 方法的 XPath 表达式是 `"//book[name[@lang='cn']]"`, 表示选取所有子元素 `name` 的属性 `lang` 的值为 `cn` 的 `book` 元素。`selectSingleNode` 也不被 Firefox 支持, 但同样可以使用原型扩展来解决这个问题。程序中使用如下代码来让 Firefox 支持 `selectSingleNode` 方法。

```

if( document.implementation.hasFeature("XPath", "3.0") )
{
    XMLDocument.prototype.selectNodes = function(cXPathString, xNode)
    {
        if( !xNode ) { xNode = this; }
        var oNSResolver = this.createNSResolver(this.documentElement)
        var altems = this.evaluate(cXPathString, xNode, oNSResolver,
                                   XPathResult.ORDERED_NODE_SNAPSHOT_TYPE, null)
        var aResult = [];
        for( var i = 0; i < altems.snapshotLength; i++)
        {
            aResult[i] = altems.snapshotItem(i);
        }
        return aResult;
    }
    XMLDocument.prototype.selectSingleNode = function(cXPathString, xNode)
    {
        if( !xNode ) { xNode = this; }
        var xltems = this.selectNodes(cXPathString, xNode);
        if( xltems.length > 0 )
        {
            return xltems[0];
        }
        else
        {
            return null;
        }
    }
}

```

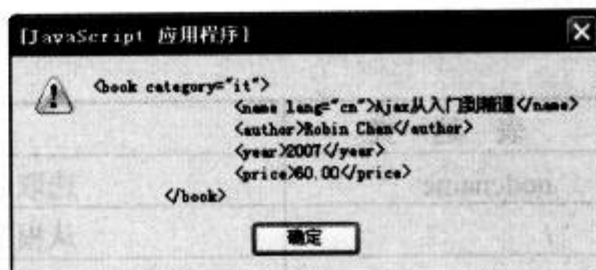


图 10.7 选择第一个语言为 cn 的图书

在对 XPath 的使用有初步了解后, 10.6.2 小节开始向读者介绍 XPath 的语法。

## 10.6.2 选取节点

XPath 是使用路径表达式在 XML 文档中选取节点的，其类似于文件系统的路径描述方法。例如在文件系统中，需要描述 books 文件夹下的 Ajax 文件夹，可以使用“books/Ajax”。在 XPath 中，如果需要描述 books 节点下的 Ajax 节点，也是使用“books/Ajax”。表 10-3 列出了最常用的路径表达式，如下所示。

表 10-3 最常用的路径表达式

表 达 式	说 明
nodename	选取名为nodename的节点
/	从根节点选取
//	从匹配选择的当前节点选择文档中的节点，而不考虑它们所在的位置
.	选取当前节点
..	选取当前节点的父节点
@attributename	选取名为attributename的属性

下面是一些使用这些路径表达式的例子。

books	//选取 books 元素的子节点
/books	//选取 books 根元素
/books/book/name	//选取 books 根元素下的 book 子元素的 name 子元素
book/name	//选取所有属于 book 元素的名字子元素
//book	//选取文档中任何位置的所有 book 子元素
book//name	//选取 book 元素下的所有 name 元素，而不管 name 元素出现在什么位置
name/@lang	//选取 name 元素的 lang 属性
//@lang	//选取所有的 lang 属性

## 10.6.3 谓语句

谓语句（Predicates）被嵌在方括号（[]）中，被用来查找符合某一特定条件的节点。下面是一些使用谓语句的示例。

/books/book[1]	//选取 books 根元素下的第一个 book 元素
/books/book[last()]	//选取 books 根元素下的最后一个 book 元素
/books/book[last()-1]	//选取 books 根元素下的倒数第二个 book 元素
/books/book[position()<4]	//选取 books 根元素下的前 3 个 book 元素
name[@lang]	//选取拥有 lang 属性的 name 元素
name[@lang="cn"]	//选取 lang 属性值为 cn 的 name 元素
book[name[@lang="cn"]]	//选取子元素 name 的属性 lang 的属性值为 cn 的 book 元素
book[price>30]	//选取子元素 price 的值大于 30 的 book 元素
book[price>30]/name	//选取子元素 price 的值大于 30 的 book 元素的子元素 name

注意：IE 的早期版本会将[0]当作第一个节点来执行，这个错误后来在 IE6 SP2 中被纠正。



10.6.4 通配符

XPath 提供了 3 种通配符，以让开发者可以选择未知的元素，如表 10-4 所示。

表 10-4 通配符

通 配 符	说 明
*	匹配任何元素节点
@*	匹配任何属性节点
node()	匹配任何节点

一些使用通配符的例子如下所示。

/books/*	//选取根元素 books 下的所有元素节点
//node()	//选取所有节点
//book[@*]	//选取所有拥有属性的 book 元素

10.6.5 使用多个路径

XPath 提供了一种在一个表达式中使用多个路径选择元素的方法，就是使用 “|” 符号来分隔多个路径。它与正则表达式中的选择符号功能类似，其作用是只要元素符合多个路径中的一个，就将元素作为结果集中的一项返回。一些使用 “|” 的例子如下所示。

//name //author	//选取所有的 name 元素和 author 元素
//book/name //book/author	//选取所有 book 元素下的 name 元素和 author 元素
//book[price<30] //book[price>60]	//选取所有 price 子元素的值小于 30 或者大于 60 的 book 元素

10.6.6 坐标轴

当需要获得相对于当前节点的节点集时，可以使用 XPath 的坐标轴。例如要获得当前节点的所有父节点中名为 book 的节点，可以使用如下 XPath 表达式。

parent::book

其中符号::前面的 parent 是轴名，后面是要选取的元素名。所有 XPath 所支持的轴如表 10-5 所示。

表 10-5 XPath 轴

轴 名	说 明
ancestor	选取当前节点的所有祖先元素
ancestor-or-self	选取当前节点和节点的所有祖先元素
attribute	选取当前节点的所有属性
child	选取当前节点的所有子元素
descendant	选取当前节点的所有子孙元素

续表

轴 名	说 明
descendant-of-self	选取当前节点和节点的所有子孙元素
following	选取文档中当前节点的结束标签之后的所有节点
following-sibling	选取当前节点之后的所有兄弟节点
namespace	选取当前节点的所有命名空间节点
parent	选取当前节点的所有父节点
preceding	选取文档中当前节点开始标签之前的所有节点
preceding-sibling	选取当前节点之前的所有兄弟节点
self	选取当前节点

### 10.6.7 运算符

在基本示例中为了选取所有 price 值大于 30 的 book 元素，使用了如下 XPath 语句。

```
//book[price>=30]
```

其中在谓语中使用的“>=”就是 XPath 的运算符之一，其与作用于 JavaScript 表达式的比较运算符“>=”一样。表 10-6 列举出了所有可以在 XPath 中使用的运算符。

表 10-6 XPath 运算符

运 算 符	说 明	示 例
+	加法	//book[price+10=100]
-	减法	//book[price-10=100]
*	乘法	//book[price*10=100]
div	除法	//book[price div 10=100]
=	等于	//book[price=30]
!=	不等于	//book[price!=30]
<	小于	//book[price<30]
<=	小于等于	//book[price<=30]
>	大于	//book[price>30]
>=	大于等于	//book[price>=30]
or	逻辑或	//book[price<30 or price>50]
and	逻辑与	//book[price>30 and price < 50]
mod	模运算	//book[price mod 10 = 2]

### 10.6.8 路径表达式的步语法

每个路径表达式都由一个或者多个步（step）组成，如下所示。

```
parent::books/book[price>30]
```

这个表达式由两步组成，第一步是 parent::books，第二步是 book[price>30]。路径表达式每步的组

成结构如下所示。

轴名称::节点[谓语]

轴名称和谓语在不需要时可以省略。

### 10.6.9 XPath 函数

XPath 内置了大量的函数来提供更为灵活和强大的功能。在 10.6.3 谓语一节中演示的 `position()` 和 `last()` 都属于 XPath 的内置函数。表 10-7 列出了常用的 XPath 函数。

表 10-7 常用的 XPath 函数

函 数	说 明
<code>count</code>	返回节点集中的节点数
<code>id</code>	根据唯一的ID选择元素
<code>last</code>	表示当前节点集的最后一个节点的位置
<code>position</code>	返回节点在父级的索引
<code>concat</code>	连接两个字符串
<code>contains</code>	如果第一个参数字符串包含第二个参数字符串, 则返回true, 否则返回false
<code>normalize-space</code>	返回去除了空白的参数字符串
<code>starts-with</code>	如果第一个参数字符串以第二个参数字符串开头, 则返回true, 否则返回false
<code>substring</code>	返回第一个参数字符串中从第二个参数指定的位置开始, 第三个参数指定的长度的字符串
<code>substring-after</code>	返回第一个参数字符串中第一次出现第二个参数字符串之后的子字符串
<code>substring-before</code>	返回第一个参数字符串中第一次出现第二个参数字符串之前的子字符串
<code>translate</code>	返回第一个参数字符串将所出现的第二个参数字符串指定的内容替换为第三个参数字符串所指定的内容后的字符串
<code>boolean</code>	将参数转换为布尔值
<code>true</code>	返回true
<code>false</code>	返回false
<code>lang</code>	如果当前节点中的xml:lang属性与参数字符串相同, 则返回true
<code>not</code>	如果参数为true, 则返回false。如果参数为false, 则返回true
<code>ceiling</code>	返回不小于参数的最小整数
<code>floor</code>	返回不大于参数的最大整数
<code>number</code>	将参数转换为数字
<code>round</code>	将参数取整并返回整数值
<code>sum</code>	返回节点集中所有节点值的总和。在计算前节点值会被转换成数字型

## 10.7 格式化 XML 工具: XSL

HTML 使用有限的固定标签, 所以浏览器可以预先对每个标签定义样式。使用浏览器打开 HTML 文档时, 浏览器就可以按照预定的样式来格式化文档。但是 XML 并没有固定的标签, 这样浏览器就

没有一个标注来自动格式化 XML 文档。为了控制 XML 的显示，就需要建立一种机制来为浏览器指示如何格式化 XML 文档，这时候就需要使用 XSL。XSL 是英文 eXtensible Stylesheet Language 的缩写，是用来格式化显示 XML 文档的样式语言。本节将向读者介绍 XSL 的组成结构、常用元素和使用方法等知识。

### 10.7.1 基本示例

在正式开始介绍 XSL 的知识前，先来看一个使用 XSL 格式化 XML 输出的示例，让读者对 XSL 有个直观的认识。XML 文档使用在 10.6 节 XPath 中使用过的 books.xml，并做一些适当的修改，代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="books.xsl"?>
<books>
  <book category="sport">
    <name lang="en">Play Basketball with me</name>
    <author>Jack Robinson</author>
    <year>2007</year>
    <price>30.00</price>
  </book>
  <book category="it">
    <name lang="cn">Ajax 从入门到精通</name>
    <author>Robin Chen</author>
    <year>2007</year>
    <price>60.00</price>
  </book>
  <book category="cooking">
    <name lang="en">Nice Cookie</name>
    <author>Susan Smith</author>
    <year>2007</year>
    <price>15.00</price>
  </book>
  <book category="sport">
    <name lang="en">Running like Forest Gump</name>
    <author>Jim Jackson</author>
    <year>2007</year>
    <price>35.00</price>
  </book>
  <book category="cooking">
    <name lang="en">The Best Chicken</name>
    <author>Susan Smith</author>
    <year>2006</year>
    <price>20.00</price>
  </book>
</books>
```

在 XML 文档的第二行加入 xsl 样式表的声明，并指定 xsl 文件的路径为 books.xsl。books.xsl 文件

的代码如下所示。

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="utf-8" doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"/>
<xsl:template match="/">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<title>Book List</title>
</head>

<body style="font-family:Tahoma;">
<h1>Book List</h1>
<xsl:element name="form">
  <xsl:attribute name="name">order</xsl:attribute>
  <xsl:attribute name="id">order</xsl:attribute>
  <xsl:attribute name="action">http://www.robchen.cn</xsl:attribute>
  <xsl:attribute name="method">post</xsl:attribute>
  <xsl:attribute name="target">_blank</xsl:attribute>
  <xsl:apply-templates select="books"></xsl:apply-templates>
  <xsl:element name="label">
    <xsl:attribute name="for">quantity</xsl:attribute>
    Quantity
  </xsl:element>
  <xsl:element name="input">
    <xsl:attribute name="type">text</xsl:attribute>
    <xsl:attribute name="name">quantity</xsl:attribute>
    <xsl:attribute name="id">quantity</xsl:attribute>
    <xsl:attribute name="value">1</xsl:attribute>
    <xsl:attribute name="size">2</xsl:attribute>
    <xsl:attribute name="maxlength">2</xsl:attribute>
    <xsl:attribute name="onchange"><![CDATA[if(!(/AD/.test(this.value)))this.value=1;]]></xsl:attribute>
  </xsl:element>
  <xsl:element name="input">
    <xsl:attribute name="type">submit</xsl:attribute>
    <xsl:attribute name="value">submit</xsl:attribute>
  </xsl:element>
</xsl:element>
</body>
</html>
</xsl:template>
<xsl:template match="books">
  <ul style="list-style:none;">
    <xsl:apply-templates select="book"></xsl:apply-templates>
  </ul>
</xsl:template>
<xsl:template match="book">
```



```

</li>
<div>
  <xsl:element name="input">
    <xsl:attribute name="type">radio</xsl:attribute>
    <xsl:attribute name="name">book</xsl:attribute>
    <xsl:attribute name="value"><xsl:value-of select="name"/></xsl:attribute>
    <xsl:attribute name="id">book<xsl:value-of select="position()"/></xsl:attribute>
    <xsl:if test="position()=1">
      <xsl:attribute name="checked">checked</xsl:attribute>
    </xsl:if>
  </xsl:element>
  <xsl:element name="label">
    <xsl:attribute name="for">book<xsl:value-of select="position()"/></xsl:attribute>
    <strong>&lt;&lt;<xsl:value-of select="name"></xsl:value-of>&gt;&gt;</strong>(<xsl:value-of
select="year"/></xsl:value-of select="name/@lang"/>)
  </xsl:element>
</div>
</li>
</ul>
<li>Category:<xsl:value-of select="@category"/></li>
<li>Author:<xsl:value-of select="author"></xsl:value-of></li>
<li>Price:$<xsl:value-of select="price"></xsl:value-of></li>
</ul>
</li>
</xsl:template>
</xsl:stylesheet>

```

然后在浏览器中打开 books.xml 文件，其显示的样式如图 10.8 所示。如果去掉 books.xml 中第二行的样式表声明，则在浏览器中只会显示默认的树形图，如图 10.9 所示。

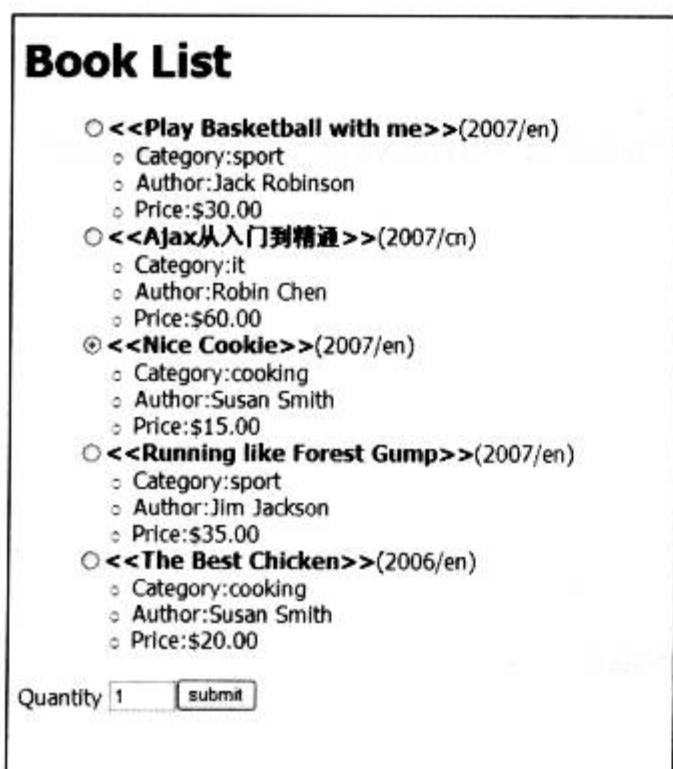


图 10.8 显示图书列表



图 10.9 树形图

### 10.7.2 XSL 声明

XSL 是一个符合 XML 语法规则的文档。如上例所示, 声明一个 XSL 文档的方法是将 `stylesheet` 元素作为 XSL 文档的根元素, 代码如下所示。

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">...</xsl:stylesheet>
```

`version` 属性声明了其使用的版本, `xsl` 是命名空间。

除了使用 `stylesheet` 元素外, 还可以使用 `transform` 元素作为根元素, 其作用与 `stylesheet` 一样, 代码如下所示。

```
<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">...</xsl:transform>
```

### 10.7.3 使用模板

XSL 的基本元素是 `template` (模板), 每个 XML 文档都由一个或者多个 `template` 组成。每个 `template` 都针对特定的元素定义了一组输出的内容。模板的定义使用 `template` 元素, 代码如下所示。

```
<xsl:template match="xpath">...</xsl:template>
```

`match` 属性指定了该模板所匹配的元素, 其属性值是一个 XPath 表达式。当 `match` 属性值为 `"/` 时表示该模板是根模板。根模板会被解析器当作整个 XSL 文档的入口进行解析, 代码如下所示。

```
<xsl:template match="/">...</xsl:template>
```

定义好模板后, 可以使用 `apply-templates` 来调用定义好的模板, 代码如下所示。

```
<xsl:apply-templates select="books"></xsl:apply-templates>
```

其中 `select` 属性值是一个 XPath 表达式, 指定了要选取的元素。当为所选择的元素定义了相应的模板时, 则会应用模板的定义进行格式化。如果没有找到对应的模板, 则会输出元素的字符串表示。例如上例中 `select` 属性值是 `books`, 表示选取当前节点下的所有 `books` 元素。而在文档中对 `books` 元素定义了一个模板, 其内容如下所示。

```
<xsl:template match="books">
  <ul style="list-style:none;">
    <xsl:apply-templates select="book"></xsl:apply-templates>
  </ul>
</xsl:template>
```

所以 `books` 元素会被替换成模板中指定的内容。而在 `books` 元素对应的模板中又对其子元素 `book` 应用了模板。文档中定义了匹配 `book` 元素的模板, 内容如下所示。

```
<xsl:template match="book">
  <li>
    <div>
      <xsl:element name="input">
```

```

    <xsl:attribute name="type">radio</xsl:attribute>
    <xsl:attribute name="name">book</xsl:attribute>
    <xsl:attribute name="value"><xsl:value-of select="name"/></xsl:attribute>
    <xsl:attribute name="id">book<xsl:value-of select="position()"/></xsl:attribute>
    <xsl:if test="position()=1">
        <xsl:attribute name="checked">checked</xsl:attribute>
    </xsl:if>
</xsl:element>
<xsl:element name="label">
    <xsl:attribute name="for">book<xsl:value-of select="position()"/></xsl:attribute>
    <strong>&lt;&lt;<xsl:value-of select="name"/></xsl:value-of>&gt;&gt;</strong>(<xsl:value-of
select="year"/></xsl:value-of select="name/@lang"/>)
</xsl:element>
</div>
</ul>
<li>Category:<xsl:value-of select="@category"/></li>
<li>Author:<xsl:value-of select="author"/></li>
<li>Price:$<xsl:value-of select="price"/></li>
</ul>
</li>
</xsl:template>

```

所以每个 book 元素又会被该模板内所定义的内容替换。这就是 XSL 格式化 XML 的过程。最终输出的 html 文档如下所示。

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Book List</title>
</head>
<body style="font-family:Tahoma;">
<h1>Book List</h1>
<form name="order" id="order" action="http://www.robchen.cn" method="post" target="_blank">
<ul style="list-style:none;">
<li>
<div>
<input type="radio" name="book" value="Play Basketball with me" id="book1" checked><label
for="book1"><strong>&lt;&lt;Play Basketball with me&gt;&gt;</strong>(2007/en)
</label>

</div>
<ul>
<li>Category:sport</li>
<li>Author:Jack Robinson</li>
<li>Price:$30.00</li>
</ul>
</li>
<li>

```

```

<div>
<input type="radio" name="book" value="Ajax&#20174;&#20837;&#38376;&#21040;&#31934;&#36890;"
id="book2"><label for="book2"><strong>&lt;&lt;Ajax 从入门到精通&gt;&gt;</strong>(2007/cn)
    </label>
</div>
<ul>
<li>Category:it</li>
<li>Author:Robin Chen</li>
<li>Price:$60.00</li>
</ul>
</li>
<li>
<div>
<input type="radio" name="book" value="Nice Cookie" id="book3"><label for="book3"><strong>&lt;&lt;Nice
Cookie&gt;&gt;</strong>(2007/en)
    </label>
</div>
<ul>
<li>Category:cooking</li>
<li>Author:Susan Smith</li>
<li>Price:$15.00</li>
</ul>
</li>
<li>
<div>
<input type="radio" name="book" value="Running like Forest Gump" id="book4"><label
for="book4"><strong>&lt;&lt;Running like Forest Gump&gt;&gt;</strong>(2007/en)
    </label>
</div>
<ul>
<li>Category:sport</li>
<li>Author:Jim Jackson</li>
<li>Price:$35.00</li>
</ul>
</li>
<li>
<div>
<input type="radio" name="book" value="The Best Chicken" id="book5"><label for="book5"><strong>&lt;&lt;The
Best Chicken&gt;&gt;</strong>(2006/en)
    </label>
</div>
<ul>
<li>Category:cooking</li>
<li>Author:Susan Smith</li>
<li>Price:$20.00</li>

```

```

</ul>
</li>
</ul>
<label for="quantity">
  Quantity
</label><input type="text" name="quantity" id="quantity" value="1" size="2" maxlength="2"
onchange="if(!(/D/.test(this.value)))this.value=1;"/><input type="submit" value="submit"/>
</form>
</body>
</html>

```

#### 10.7.4 取得数据

定义了一个模板，只是告诉了 XSL 遇到和模板匹配的元素，即使用该模板的内容对元素进行格式化，但是具体如何格式化呢？首先要做的就是读取 XML 的数据。在 XSL 中使用 value-of 元素来得到 XML 文档的数据，代码如下所示。

```
<xsl:value-of select="name"/>
```

与 template 元素类似，value-of 元素也有一个 select 属性。该属性值可以是一个 XPath 表达式，XPath 函数，或者一个基本类型的值。当属性值为 XPath 表达式时，解析器会查找与表达式匹配的元素，并输出该元素的值。当属性值为 XPath 函数时，将输出该函数的返回值。当属性值为一个基本类型的值时，则会直接输出该值。基本示例中多次用到了 value-of 元素，代码如下所示。

```

<xsl:value-of select="position()"/>
<xsl:value-of select="@category"/>
<xsl:value-of select="author"></xsl:value-of>
<xsl:value-of select="price"></xsl:value-of>

```

value-of 元素还有另外一个属性 disable-output-escaping，属性值可以为 yes 或者 no。该属性如果值为 yes 时，则一些特殊字符，例如“<”等，会按照原样输出。如果该属性的值为 no，则特殊字符会被转换成为相应的实体，如“<”转换成“&lt;”。disable-output-escaping 如果没有被提供，则默认为 no。

#### 10.7.5 使用 for-each 元素代替模板

在某些时候开发者并不会需要定义一个模板，这时候可以使用 for-each 元素来代替。for-each 元素包含了一组格式化的规则，这些规则将会一一被应用到通过 for-each 元素的 select 属性所选定的节点上。例如，基本示例中对 books 元素定义的模板内容如下所示。

```

<xsl:template match="books">
  <ul style="list-style:none;">
    <xsl:apply-templates select="book"></xsl:apply-templates>
  </ul>
</xsl:template>
<xsl:template match="book">
  <li>

```



```

<div>
  <xsl:element name="input">
    <xsl:attribute name="type">radio</xsl:attribute>
    <xsl:attribute name="name">book</xsl:attribute>
    <xsl:attribute name="value"><xsl:value-of select="name"/></xsl:attribute>
    <xsl:attribute name="id">book<xsl:value-of select="position()"/></xsl:attribute>
    <xsl:if test="position()=1">
      <xsl:attribute name="checked">checked</xsl:attribute>
    </xsl:if>
  </xsl:element>
  <xsl:element name="label">
    <xsl:attribute name="for">book<xsl:value-of select="position()"/></xsl:attribute>
    <strong>&lt;&lt;<xsl:value-of select="name"></xsl:value-of>&gt;&gt;</strong>(<xsl:value-of
select="year"/></xsl:value-of select="name/@lang"/>)
  </xsl:element>
</div>
<ul>
  <li>Category:<xsl:value-of select="@category"/></li>
  <li>Author:<xsl:value-of select="author"></xsl:value-of></li>
  <li>Price:$<xsl:value-of select="price"></xsl:value-of></li>
</ul>
</li>
</xsl:template>

```

使用 for-each 元素改写 books 模板，代码如下所示。

```

<xsl:template match="books">
  <ul style="list-style:none;">
    <xsl:for-each select="book">
      <li>
        <div>
          <xsl:element name="input">
            <xsl:attribute name="type">radio</xsl:attribute>
            <xsl:attribute name="name">book</xsl:attribute>
            <xsl:attribute name="value"><xsl:value-of select="name"/></xsl:attribute>
            <xsl:attribute name="id">book<xsl:value-of select="position()"/></xsl:attribute>
            <xsl:if test="position()=1">
              <xsl:attribute name="checked">checked</xsl:attribute>
            </xsl:if>
          </xsl:element>
          <xsl:element name="label">
            <xsl:attribute name="for">book<xsl:value-of select="position()"/></xsl:attribute>
            <strong>&lt;&lt;<xsl:value-of
select="name"></xsl:value-of>&gt;&gt;</strong>(<xsl:value-of select="year"/></xsl:value-of
select="name/@lang"/>)
          </xsl:element>
        </div>
      <ul>
        <li>Category:<xsl:value-of select="@category"/></li>
        <li>Author:<xsl:value-of select="author"></xsl:value-of></li>
        <li>Price:$<xsl:value-of select="price"></xsl:value-of></li>
      </ul>
    </li>
  </ul>

```

```

        </ul>
    </li>
</xsl:for-each>
</ul>
</xsl:template>

```

### 10.7.6 使用 sort 元素进行排序

当需要对 for-each 或 apply-templates 元素输出的内容进行排序时，可以使用 sort 元素。sort 元素使用时是作为 for-each 元素或 apply-templates 元素的子元素存在的，代码如下所示。

```

<xsl:for-each select="book">
    <xsl:sort select="name" order="ascending">
    ...
    </xsl:sort>
</xsl:for-each>

```

sort 元素可选的属性如表 10-8 所示。

表 10-8 sort 元素属性列表

属 性	属 性 值	说 明
select	XPath表达式	指定根据哪一个元素来排序
lang	语言代码	指定用哪一种语言来排序，例如en、de、fr等
data-type	text、number	指定用来排序的数据类型。默认为text
order	accending、descending	排序方式。accending为升序、descending为降序。默认为升序
case-order	upper-first、lower-list	指定是按大写字母先排序还是按小写字母先排序

下面来修改基本示例中的 XSL 代码，以让图书信息在页面上显示时，按照其价格从低到高排列。将 books 模板的定义修改为如下所示。

```

<xsl:template match="books">
    <ul style="list-style:none;">
        <xsl:apply-templates select="book">
            <xsl:sort select="price" data-type="number"
order="ascending"></xsl:sort>
        </xsl:apply-templates>
    </ul>
</xsl:template>

```

然后使用浏览器访问 books.xml 文档，效果如图 10.10 所示。



图 10.10 排序

### 10.7.7 流程控制

XSL 提供了流程控制功能, 通过 if、choose、when 和 otherwise 元素来实现。

#### 1. if 元素

if 元素通过 test 属性指定了一个条件, 当该条件成立时则执行 if 元素所包含的内容, 否则跳过该内容。在基本示例的 XSL 文档中, 在输出每个图书的标题前面的单选按钮时, 通过判断当前 book 元素的索引来确定是否选定该单选按钮, 代码如下所示。

```
<xsl:if test="position()=1">
  <xsl:attribute name="checked">checked</xsl:attribute>
</xsl:if>
```

当需要在 test 属性值的条件表达式中使用 “<” 和 “>” 等包含特殊符号的运算符时, 需要对特殊符号进行转义。例如将 position()=1 改成 position()>1, 则需要写成下面这种形式。

```
<xsl:if test="position()>1">
  <xsl:attribute name="checked">checked</xsl:attribute>
</xsl:if>
```

#### 2. choose、when 和 otherwise

choose、when 和 otherwise 与 JavaScript 中的 switch、case 和 default 结构相似, 可以用来表达多条件选择。一个 choose 元素中可以包含多个 when 元素和一个 otherwise 元素, when 元素与 if 元素一样, 使用一个名为 test 的属性来指定条件。遇到一个 choose 元素时, 解析器会按照 when 元素的出现顺序进行分析, 如果 when 元素的 test 属性指定的条件成立, 则执行 when 元素所包含的内容。如果没有一个 when 元素中的内容被执行, 则会执行 otherwise 元素中的内容。现在修改基本示例的 XSL 文档中 book 模板的内容, 来改变输出的图书信息样式, 第一个图书信息会被加上灰色的背景色, 最后一个会被加上红色的背景色, 而其他的图书信息则会被加上灰色的边框, 其代码如下所示。

```
<xsl:template match="book">
  <li>
    <xsl:choose>
      <xsl:when test="position()=1">
        <xsl:attribute name="style">background-color:#eee;</xsl:attribute>
      </xsl:when>
      <xsl:when test="position()=last()">
        <xsl:attribute name="style">background-color:#f00;</xsl:attribute>
      </xsl:when>
      <xsl:otherwise>
        <xsl:attribute name="style">border:1px solid #eee;</xsl:attribute>
      </xsl:otherwise>
    </xsl:choose>
    <div>
      <xsl:element name="input">
        <xsl:attribute name="type">radio</xsl:attribute>
```

```

<xsl:attribute name="name">book</xsl:attribute>
<xsl:attribute name="value"><xsl:value-of select="name"/></xsl:attribute>
<xsl:attribute name="id">book<xsl:value-of select="position()"/></xsl:attribute>
<xsl:if test="position()=1">
  <xsl:attribute name="checked">checked</xsl:attribute>
</xsl:if>
</xsl:element>
<xsl:element name="label">
  <xsl:attribute name="for">book<xsl:value-of select="position()"/></xsl:attribute>
  <strong>&lt;&lt;<xsl:value-of select="name"></xsl:value-of>&gt;&gt;</strong>(<xsl:value-of
select="year"/></xsl:value-of select="name/@lang"/>)
</xsl:element>
</div>
<ul>
  <li>Category:<xsl:value-of select="@category"/></li>
  <li>Author:<xsl:value-of select="author"></xsl:value-of></li>
  <li>Price:$<xsl:value-of select="price"></xsl:value-of></li>
</ul>
</li>
</xsl:template>

```

其显示效果如图 10.11 所示。

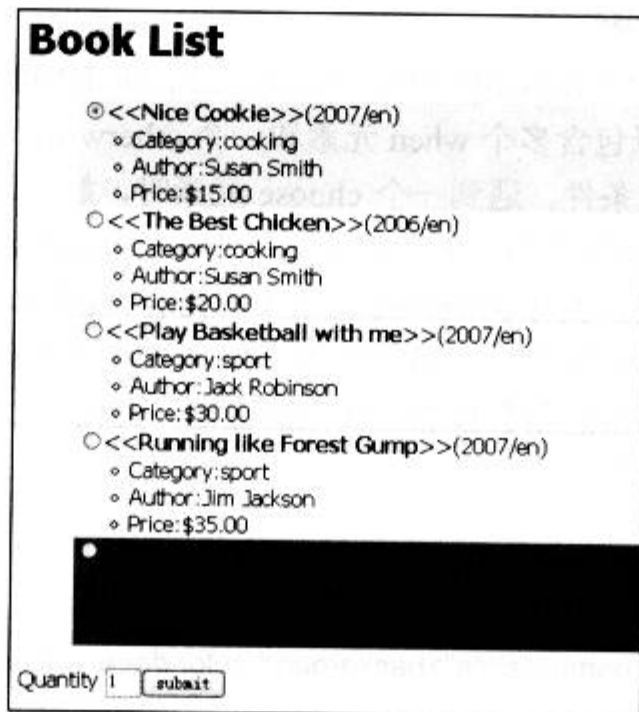


图 10.11 多条件控制

### 10.7.8 创建元素和属性

XSL 允许开发者按照需要创建新的元素和属性。创建元素使用 `element` 元素,其通过一个名为 `name` 的属性指定了所创建的元素名,而 `element` 元素的值则会被作为新创建的元素值。创建属性使用 `attribute` 属性,其通过一个名为 `name` 的属性指定了所创建的属性名,`attribute` 元素的值则会被作为新创建的属性的值。例如在基本示例中,通过 `element` 元素创建了表单元素,代码如下所示。

```

<xsl:element name="form">
  <xsl:attribute name="name">order</xsl:attribute>
  <xsl:attribute name="id">order</xsl:attribute>
  <xsl:attribute name="action">http://www.robchen.cn</xsl:attribute>
  <xsl:attribute name="method">post</xsl:attribute>
  <xsl:attribute name="target">_blank</xsl:attribute>
  <xsl:apply-templates select="books"></xsl:apply-templates>
  <xsl:element name="label">
    <xsl:attribute name="for">quantity</xsl:attribute>
    Quantity
  </xsl:element>
  <xsl:element name="input">
    <xsl:attribute name="type">text</xsl:attribute>
    <xsl:attribute name="name">quantity</xsl:attribute>
    <xsl:attribute name="id">quantity</xsl:attribute>
    <xsl:attribute name="value">1</xsl:attribute>
    <xsl:attribute name="size">2</xsl:attribute>
    <xsl:attribute name="maxlength">2</xsl:attribute>
    <xsl:attribute name="onchange"><![CDATA[if(!(/&D/.test(this.value)))this.value=1;]]></xsl:attribute>
  </xsl:element>
  <xsl:element name="input">
    <xsl:attribute name="type">submit</xsl:attribute>
    <xsl:attribute name="value">submit</xsl:attribute>
  </xsl:element>
</xsl:element>

```

### 10.7.9 指定输出格式

在基本示例的 XSL 文档中，使用了如下代码。

```

<xsl:output method="html" encoding="utf-8" doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"/>

```

这里通过 output 元素定义文档输出的格式。output 属性是一个顶级元素，其必须作为 stylesheet 或 transform 元素的子元素出现，按照惯例一般将其放在文档其他内容的上面。表 10-9 列举了 output 元素的可用属性及属性值。

表 10-9 output 元素属性列表

属 性	属 性 值	说 明
method	xml、html、text、name	定义了输出的格式，默认为xml。但如果根模板的第一个元素是html则默认为html
version	string	设置版本
encoding	string	设置输出所使用的字符集编码
omit-xml-declaration	yes、no	如果为yes，则XML的声明(<?xml...?>)在输出中会被忽略，否则就加入。默认为no
doctype-public	string	设置输出的public DOCTYPE值



续表

属 性	属 性 值	说 明
doctype-system	string	设置输出的system DOCTYPE值
indent	yes、no	当为yes时要根据结构缩进，为no时不缩进。默认是yes
media-type	string	定义输出的MIME类型，默认是text/xml

XSL 就介绍到这里，如果需要了解更多内容，可以查看 W3C 官方网站上的说明，网址为 <http://www.w3.org/Style/XSL/>。

## 10.8 在客户端格式化 XML

在之前的示例中，都是在 XML 文档中指定一个 XSL 文档来格式化 XML 的输出。当在客户端程序中得到一个包含数据的 XML 文档，需要用它所包含的数据来更新界面时，也可以使用 XSL 来格式化 XML。下面这个例子与 10.7 节中的基本示例所达到的效果一样，只是格式化是在客户端的脚本中完成的。首先需要修改 books.xml 文档来让它输出一个 HTML 片段而不是整个 HTML 文档，代码如下所示。

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<h1>Book List</h1>
<xsl:element name="form">
  <xsl:attribute name="name">order</xsl:attribute>
  <xsl:attribute name="id">order</xsl:attribute>
  <xsl:attribute name="action">http://www.robchen.cn</xsl:attribute>
  <xsl:attribute name="method">post</xsl:attribute>
  <xsl:attribute name="target">_blank</xsl:attribute>
  <xsl:apply-templates select="books"></xsl:apply-templates>
  <xsl:element name="label">
    <xsl:attribute name="for">quantity</xsl:attribute>
    Quantity
  </xsl:element>
  <xsl:element name="input">
    <xsl:attribute name="type">text</xsl:attribute>
    <xsl:attribute name="name">quantity</xsl:attribute>
    <xsl:attribute name="id">quantity</xsl:attribute>
    <xsl:attribute name="value">1</xsl:attribute>
    <xsl:attribute name="size">2</xsl:attribute>
    <xsl:attribute name="maxlength">2</xsl:attribute>
    <xsl:attribute name="onchange"><![CDATA[if(!(/D/.test(this.value)))this.value=1;]]></xsl:attribute>
  </xsl:element>
  <xsl:element name="input">
    <xsl:attribute name="type">submit</xsl:attribute>
    <xsl:attribute name="value">submit</xsl:attribute>
  </xsl:element>

```

```

</xsl:element>
</xsl:template>
<xsl:template match="books">
  <ul style="list-style:none;">
    <xsl:apply-templates select="book"></xsl:apply-templates>
  </ul>
</xsl:template>
<xsl:template match="book">
  <li>
    <div>
      <xsl:element name="input">
        <xsl:attribute name="type">radio</xsl:attribute>
        <xsl:attribute name="name">book</xsl:attribute>
        <xsl:attribute name="value"><xsl:value-of select="name"/></xsl:attribute>
        <xsl:attribute name="id">book<xsl:value-of select="position()"/></xsl:attribute>
        <xsl:if test="position()=1">
          <xsl:attribute name="checked">checked</xsl:attribute>
        </xsl:if>
      </xsl:element>
      <xsl:element name="label">
        <xsl:attribute name="for">book<xsl:value-of select="position()"/></xsl:attribute>
        <strong>&lt;&lt;<xsl:value-of select="name"></xsl:value-of>&gt;&gt;</strong>(<xsl:value-of
select="year"/></xsl:value-of select="name/@lang"/>)
      </xsl:element>
    </div>
    <ul>
      <li>Category:<xsl:value-of select="@category"/></li>
      <li>Author:<xsl:value-of select="author"></xsl:value-of></li>
      <li>Price:$<xsl:value-of select="price"></xsl:value-of></li>
    </ul>
  </li>
</xsl:template>
</xsl:stylesheet>

```

然后来编写示例页面 `demo.html`。在这个页面中提供一个按钮,当用户单击这个按钮时,页面中的 JavaScript 程序会读取 `books.xml` 和 `books.xsl`,并使用 `books.xsl` 将 `books.xml` 所包含的数据进行格式化,然后将格式化的结果显示在页面上。`demo.html` 页面代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>transformNode demo</title>
<script type="text/javascript">
function createXMLDOMObject()
{
  if(window.ActiveXObject)                //根据浏览器是否支持 ActiveXObject 来判断是否是 IE
  {

```

```

        var versions = ["MSXML2.DOMDocument.5.0",
            "MSXML2.DOMDocument.4.0", "MSXML2.DOMDocument.3.0",
            "MSXML2.DOMDocument", "Microsoft.XmlDom"];
        for(var i = 0; i < versions.length; i++)
        {
            try
            {
                var xmlDoc = new ActiveXObject(versions[i]);
                break;
            }
            catch(e){}
        }
    }
    else
    {
        //如果不是 IE, 则使用 createDocument 方法来创建
        var xmlDoc = document.implementation.createDocument("", null); //创建 XML DOM 对象
    }
    if(typeof xmlDoc == 'undefined')
    {
        //如果上述方法都没有创建成功, 则 xmlDoc 未定义
        throw new Error('XML DOM Is not supported by your browser!'); //此时抛出异常, 提示浏览器不支持 XML DOM
    }
    return xmlDoc;
}
//返回所创建的对象

function parseXML(doc)
{
    /*
    * 如果是 IE, 则使用根节点的 xml 属性输出字符串
    */
    if(window.ActiveXObject)
    {
        return doc.documentElement.xml;
    }
    /*
    * 如果不是 IE, 则使用 XMLSerializer 对象的 serializeToString 方法输出字符串
    */
    else
    {
        var oSerializer = new XMLSerializer();
        return oSerializer.serializeToString(doc, 'text/xml');
    }
}
/*
* 利用指定的 XSL 格式化 XML
* 在 IE 下使用 transformNode
* 在 FF 下使用 XSLTProcessor
*/
function transformToText(oXml, oXslt)
{
    if (typeof XSLTProcessor != "undefined") { //如果存在 XSLTProcessor, 则使用该对象进行格式化工作

```

```

    var oProcessor = new XSLTProcessor();           //实例化 XSLTProcessor
    oProcessor.importStylesheet(oXslt);             //加载 XSL 文档

    var oResultDom = oProcessor.transformToDocument(oXml); //格式化 XML 并返回一个 document
对象
    var sResult = parseXML(oResultDom);             //将 document 对象转换成字符串表示

    if (sResult.indexOf("<transformiix:result") > -1) { //截取需要的内容
        sResult = sResult.substring(sResult.indexOf(">") + 1,
                                    sResult.lastIndexOf("<"));
    }

    return sResult;                                //返回得到的字符串
} else if (window.ActiveXObject) {
    return oXml.transformNode(oXslt);              //如果是 IE, 则使用 transformNode 完成格式化工作
} else {
    throw new Error("浏览器不支持 XSL.");
}
};
function loadXMLData()
{
    try
    {
        var xml = createXMLDOMObject();             //加载 books.xml
        xml.async = false;
        xml.load('books.xml');
        var xsl = createXMLDOMObject();             //加载 books.xsl
        xsl.async = false;
        xsl.load('books.xsl');
        var htmlString = transformToText(xml,xsl);   //格式化 books.xml
        document.getElementById('main').innerHTML = htmlString; //将格式化后的内容添加到 div#main 中
    }
    catch(e)
    {
        alert(e.message);
    }
}
</script>
</head>

<body>
    <div id="main"></div>
    <input type="button" onclick="loadXMLData();" value="load" />
</body>
</html>

```

demo.html 的初始界面如图 10.12 所示。单击 load 按钮, 页面中的 JavaScript 会读取 books.xml 文档并将其格式化输出到页面上, 如图 10.13 所示。

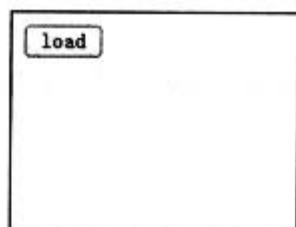


图 10.12 初始界面

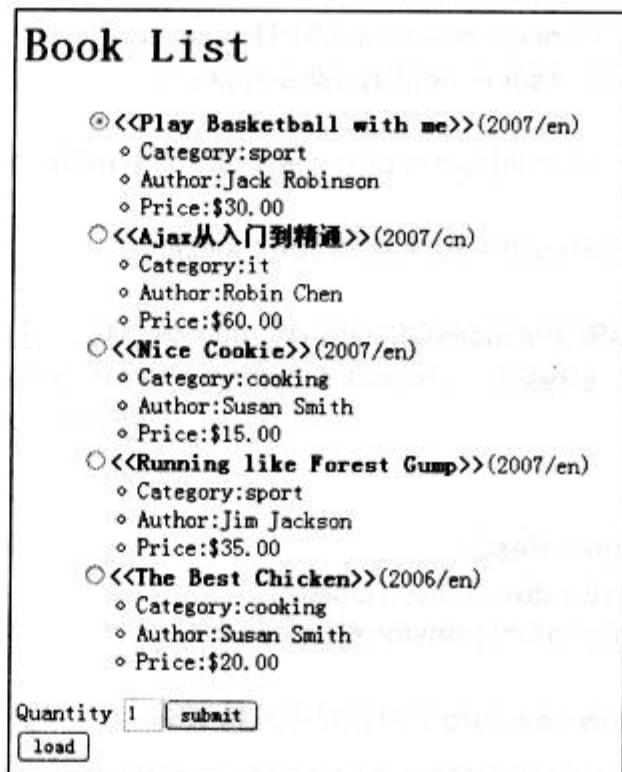


图 10.13 显示数据

程序中使用 `transformToText` 函数来完成 XML 的格式化，其接受两个参数，第一个为需要格式化的 XML 文档对象，第二个需要使用 XSL 文档对象。该函数针对浏览器的兼容性做了封装处理。在 IE 中使用 XSL 格式化 XML 很简单，只需要调用 `transformNode` 方法即可，代码如下所示。

```
return oXml.transformNode(oXslt);
```

而在非 IE 核心的浏览器，例如 Firefox 中，则会麻烦一些，需要使用 `XSLTProcessor` 对象，代码如下所示。

```
var oProcessor = new XSLTProcessor();           //实例化 XSLTProcessor
oProcessor.importStylesheet(oXslt);             //加载 XSL 文档

var oResultDom = oProcessor.transformToDocument(oXml); //格式化 XML 并返回一个 document 对象
var sResult = parseXML(oResultDom);              //将 document 对象转换成字符串表示

if (sResult.indexOf("<transformiix:result") > -1) { //截取需要的内容
    sResult = sResult.substring(sResult.indexOf(">") + 1,
                                sResult.lastIndexOf("<"));
}

return sResult;                                 //返回得到的字符串
```

## 10.9 跨浏览器的 XML 开发框架：zXML

由于不同的浏览器核心对 XML 的支持不同，导致在开发过程中需要针对不同的浏览器核心来编写大量的兼容代码，降低了开发效率。一个好的方法就是将这些差异处理进行总结和封装，提供一个统



一的方法在开发中使用, 以此来提高开发效率。由 Nicholas C. Zakas 开发的 zXML 就出色地完成了这个工作。zXML 框架将 XML 处理过程中主要的兼容性问题都进行了封装, 并提供了统一的兼容地方法。zXML 的核心代码很简洁, 如下所示。

```

/*-----
 * JavaScript zXml Library
 * Version 1.0.2
 * by Nicholas C. Zakas, http://www.nczonline.net/
 * Copyright (c) 2004-2006 Nicholas C. Zakas. All Rights Reserved.
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU Lesser General Public License as published by
 * the Free Software Foundation; either version 2.1 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *-----
 */

var zXml /*:Object*/ = {
    useActiveX: (typeof ActiveXObject != "undefined"),
    useDom: document.implementation && document.implementation.createDocument,
    useXmlHttp: (typeof XMLHttpRequest != "undefined")
};

zXml.ARR_XMLHTTP_VERS /*:Array*/ = ["MSXML2.XmlHttp.6.0", "MSXML2.XmlHttp.3.0"];

zXml.ARR_DOM_VERS /*:Array*/ = ["MSXML2.DOMDocument.6.0", "MSXML2.DOMDocument.3.0"];

/**
 * Static class for handling XMLHttp creation.
 * @class
 */
function zXmlHttp() {
}

/**
 * Creates an XMLHttp object.
 * @return An XMLHttp object.
 */
zXmlHttp.createRequest = function ()/*:XMLHttp*/ {

```

```

//if it natively supports XMLHttpRequest object
if (zXml.useXmlHttp) {
    return new XMLHttpRequest();
} else if (zXml.useActiveX) { //IE < 7.0 = use ActiveX

    if (!zXml.XMLHTTP_VER) {
        for (var i=0; i < zXml.ARR_XMLHTTP_VERS.length; i++) {
            try {
                new ActiveXObject(zXml.ARR_XMLHTTP_VERS[i]);
                zXml.XMLHTTP_VER = zXml.ARR_XMLHTTP_VERS[i];
                break;
            } catch (oError) {
            }
        }
    }

    if (zXml.XMLHTTP_VER) {
        return new ActiveXObject(zXml.XMLHTTP_VER);
    } else {
        throw new Error("Could not create XML HTTP Request.");
    }
} else {
    throw new Error("Your browser doesn't support an XML HTTP Request.");
}

};

/**
 * Indicates if XMLHttpRequest is available.
 * @return True if XMLHttpRequest is available, false if not.
 */
zXmlHttp.isSupported = function ()/*:Boolean*/ {
    return zXml.useXmlHttp || zXml.useActiveX;
};

/**
 * Static class for handling XML DOM creation.
 * @class
 */
function zXmlDom() {

}

/**
 * Creates an XML DOM document.
 * @return An XML DOM document.
 */

```

```

zXmlDom.createDocument = function () /*:XMLDocument*/{

    if (zXml.useDom) {

        var oXmlDom = document.implementation.createDocument("", "", null);

        oXmlDom.parseError = {
            valueOf: function () { return this.errorCode; },
            toString: function () { return this.errorCode.toString(); }
        };

        oXmlDom._initError_();

        oXmlDom.addEventListener("load", function () {
            this._checkForErrors_();
            this._changeReadyState_(4);
        }, false);

        return oXmlDom;

    } else if (zXml.useActiveX) {
        if (!zXml.DOM_VER) {
            for (var i=0; i < zXml.ARR_DOM_VERS.length; i++) {
                try {
                    new ActiveXObject(zXml.ARR_DOM_VERS[i]);
                    zXml.DOM_VER = zXml.ARR_DOM_VERS[i];
                    break;
                } catch (oError) {
                }
            }
        }

        if (zXml.DOM_VER) {
            return new ActiveXObject(zXml.DOM_VER);
        } else {
            throw new Error("Could not create XML DOM document.");
        }
    } else {
        throw new Error("Your browser doesn't support an XML DOM document.");
    }

};

/**
 * Indicates if an XML DOM is available.
 * @return True if XML DOM is available, false if not.
 */
zXmlDom.isSupported = function () /*:Boolean*/ {
    return zXml.useDom || zXml.useActiveX;
}

```

```

};

//Code to make Mozilla DOM documents act more like MS DOM documents.
var oMozDocument = null;
if (typeof XMLHttpRequest != "undefined") {
    oMozDocument = XMLHttpRequest;
} else if (typeof Document != "undefined") {
    oMozDocument = Document;
}

if (oMozDocument && !window.opera) {

    oMozDocument.prototype.readyState = 0;
    oMozDocument.prototype.onreadystatechange = null;

    oMozDocument.prototype._changeReadyState_ = function (iReadyState) {
        this.readyState = iReadyState;

        if (typeof this.onreadystatechange == "function") {
            this.onreadystatechange();
        }
    };

    oMozDocument.prototype._initError_ = function () {
        this.parseError.errorCode = 0;
        this.parseError.filepos = -1;
        this.parseError.line = -1;
        this.parseError.linepos = -1;
        this.parseError.reason = null;
        this.parseError.srcText = null;
        this.parseError.url = null;
    };

    oMozDocument.prototype._checkForErrors_ = function () {

        if (this.documentElement.tagName == "parsererror") {

            var reError = />([\s\S]*?)Location:([\s\S]*?)Line Number (\d+), Column (\d+);<sourceText>([\s\S]*?)
            (?;\s*\s*)/;

            reError.test(this.xml);

            this.parseError.errorCode = -999999;
            this.parseError.reason = RegExp.$1;
            this.parseError.url = RegExp.$2;
            this.parseError.line = parseInt(RegExp.$3);
            this.parseError.linepos = parseInt(RegExp.$4);
            this.parseError.srcText = RegExp.$5;
        }
    }
}

```

```

};

oMozDocument.prototype.loadXML = function (sXml) {

    this._initError_();

    this._changeReadyState_(1);

    var oParser = new DOMParser();
    var oXmlDom = oParser.parseFromString(sXml, "text/xml");

    while (this.firstChild) {
        this.removeChild(this.firstChild);
    }

    for (var i=0; i < oXmlDom.childNodes.length; i++) {
        var oNewNode = this.importNode(oXmlDom.childNodes[i], true);
        this.appendChild(oNewNode);
    }

    this._checkForErrors_();

    this._changeReadyState_(4);

};

oMozDocument.prototype._load_ = oMozDocument.prototype.load;

oMozDocument.prototype.load = function (sURL) {
    this._initError_();
    this._changeReadyState_(1);
    this._load_(sURL);
};

Node.prototype._defineGetter_("xml", function () {
    var oSerializer = new XMLSerializer();
    return oSerializer.serializeToString(this, "text/xml");
});

Node.prototype._defineGetter_("text", function () {
    var sText = "";
    for (var i = 0; i < this.childNodes.length; i++) {
        if (this.childNodes[i].hasChildNodes()) {
            sText += this.childNodes[i].text;
        } else {
            sText += this.childNodes[i].nodeValue;
        }
    }
    return sText;
};

```



```

    });

}

/**
 * Static class for handling XSLT transformations.
 * @class
 */
function zXslt() {

}

/**
 * Transforms an XML DOM to text using an XSLT DOM.
 * @param oXml The XML DOM to transform.
 * @param oXslt The XSLT DOM to use for the transformation.
 * @return The transformed version of the string.
 */
zXslt.transformToText = function (oXml /*:XMLDocument*/, oXslt /*:XMLDocument*/):String* {
    if (typeof XSLTProcessor != "undefined") {
        var oProcessor = new XSLTProcessor();
        oProcessor.importStylesheet(oXslt);

        var oResultDom = oProcessor.transformToDocument(oXml);
        var sResult = oResultDom.xml;

        if (sResult.indexOf("<transformix:result") > -1) {
            sResult = sResult.substring(sResult.indexOf(">") + 1,
                                      sResult.lastIndexOf("<"));
        }

        return sResult;
    } else if (zXml.useActiveX) {
        return oXml.transformNode(oXslt);
    } else {
        throw new Error("No XSLT engine found.");
    }
};

/**
 * Static class for handling XPath evaluation.
 * @class
 */
function zXPath() {

}

/**
 * Selects the first node matching a given XPath expression.

```

```

* @param oRefNode The node from which to evaluate the expression.
* @param sXPath The XPath expression.
* @param oXmlNs An object containing the namespaces used in the expression. Optional.
* @return An XML node matching the expression or null if no matches found.
*/
zXPath.selectNodes = function (oRefNode /*:Node*/, sXPath /*:String*/, oXmlNs /*:Object*/) {
    if (typeof XPathEvaluator != "undefined") {

        oXmlNs = oXmlNs || {};

        var nsResolver = function (sPrefix) {
            return oXmlNs[sPrefix];
        };

        var oEvaluator = new XPathEvaluator();
        var oResult = oEvaluator.evaluate(sXPath, oRefNode, nsResolver,
                                          XPathResult.ORDERED_NODE_ITERATOR_TYPE,
                                          null);

        var aNodes = new Array;

        if (oResult != null) {
            var oElement = oResult.iterateNext();
            while(oElement) {
                aNodes.push(oElement);
                oElement = oResult.iterateNext();
            }
        }

        return aNodes;

    } else if (zXml.useActiveX) {
        if (oXmlNs) {
            var sXmlNs = "";
            for (var sProp in oXmlNs) {
                sXmlNs += "xmlns:" + sProp + "=\"" + oXmlNs[sProp] + "\" ";
            }
            oRefNode.ownerDocument.setProperty("SelectionNamespaces", sXmlNs);
        }
        return oRefNode.selectNodes(sXPath);
    } else {
        throw new Error("No XPath engine found.");
    }
};

/**
* Selects the first node matching a given XPath expression.
* @param oRefNode The node from which to evaluate the expression.

```

```

* @param sXPath The XPath expression.
* @param oXmlNs An object containing the namespaces used in the expression.
* @return An XML node matching the expression or null if no matches found.
*/
zXPath.selectSingleNode = function (oRefNode /*:Node*/, sXPath /*:String*/, oXmlNs /*:Object*/) {
    if (typeof XPathEvaluator != "undefined") {

        oXmlNs = oXmlNs || {};

        var nsResolver = function (sPrefix) {
            return oXmlNs[sPrefix];
        };

        var oEvaluator = new XPathEvaluator();
        var oResult = oEvaluator.evaluate(sXPath, oRefNode, nsResolver,
            XPathResult.FIRST_ORDERED_NODE_TYPE, null);

        if (oResult != null) {
            return oResult.singleNodeValue;
        } else {
            return null;
        }
    } else if (zXml.useActiveX) {
        if (oXmlNs) {
            var sXmlNs = "";
            for (var sProp in oXmlNs) {
                sXmlNs += "xmlns:" + sProp + "=\"" + oXmlNs[sProp] + "\" ";
            }
            oRefNode.ownerDocument.setProperty("SelectionNamespaces", sXmlNs);
        }
        return oRefNode.selectSingleNode(sXPath);
    } else {
        throw new Error("No XPath engine found.")
    }
};

/**
 * General purpose XML serializer.
 * @class
 */
function zXMLSerializer() {

}

/**
 * Serializes the given XML node into an XML string.
 * @param oNode The XML node to serialize.

```

```

* @return An XML string.
*/
zXMLSerializer.prototype.serializeToString = function (oNode /*:Node*/):String* {

    var sXml = "";

    switch (oNode.nodeType) {
        case 1: //element
            sXml = "<" + oNode.tagName;

            for (var i=0; i < oNode.attributes.length; i++) {
                sXml += " " + oNode.attributes[i].name + "=\"" + oNode.attributes[i].value + "\"";
            }

            sXml += ">";

            for (var i=0; i < oNode.childNodes.length; i++){
                sXml += this.serializeToString(oNode.childNodes[i]);
            }

            sXml += "</" + oNode.tagName + ">";
            break;

        case 3: //text node
            sXml = oNode.nodeValue;
            break;
        case 4: //cdata
            sXml = "<![CDATA[" + oNode.nodeValue + "]]>";
            break;
        case 7: //processing instruction
            sXml = "<?" + oNode.nodevalue + "?>";
            break;
        case 8: //comment
            sXml = "<!--" + oNode.nodevalue + "-->";
            break;
        case 9: //document
            for (var i=0; i < oNode.childNodes.length; i++){
                sXml += this.serializeToString(oNode.childNodes[i]);
            }
            break;
    }

    return sXml;
};

```

zXML 的使用非常简单方便。首先来看如何使用 zXML 创建 XML DOM 对象。利用 zXML 创建 XML DOM 对象的方法如下所示。

```
var xmlDoc = zXmlDom.createDocument()
```

zXML 对 XML 文档的异步加载进行了封装, 在 IE 和非 IE 核心的浏览器中, 都统一使用 onreadystatechange 和 readyState 来控制, 代码如下所示。

```
var xmlDoc = zXmlDom.createDocument();
xmlDoc.onreadystatechange = function()
{
    if(xmlDoc.readyState == 1)
    {
        alert('开始加载. ');
    }
    if(xmlDoc.readyState == 4)
    {
        alert('加载完成. ');
    }
}
```

zXML 通过原型扩展使开发者可以在非 IE 核心的浏览器下使用 loadXML 方法及 xml 属性, 代码如下所示。

```
var xmlDoc = zXmlDom.createDocument();
xmlDoc.loadXML("<books><book><title>AJAX</title><author>Robin</author></book></books>");
alert(book.xml);
```

zXML 对 selectNodes 和 selectSingleNode 方法进行了封装, 并提供了统一的调用方式, 代码如下所示。

```
var books = zXPath.selectNodes(xmlDoc, "//books");
var book = zXPath.selectSingleNode(xmlDoc, "/books/book");
```

zXML 对 XML 的格式化进行了封装, 并提供了统一的调用方法, 代码如下所示。

```
var text = zXslt.transformToText(xmlDoc, xslDoc);
```

zXML 还提供了一个名为 zXMLSerializer 的类, 该类提供了一个名为 serializeToString 的实例方法, 可以将节点转化为相应的字符串表示, 代码如下所示。

```
var serializer = new zXMLSerializer();
var str = serializer(xmlNode);
```

除此之外, zXML 还提供了对创建 XMLHttpRequest 对象的兼容性封装, 代码如下所示。

```
var xmlhttp = zXmlHttp.createRequest();
```

下面使用 zXML 和异步加载来重写 10.8 节中格式化 XML 的例子, 代码如下所示。

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



```

<script type="text/javascript" src="zxml.js"></script><!-- 引入 zXML 框架 -->
<script type="text/javascript">
var xml = zXmlDom.createDocument();           //创建 XMLDOM 对象
var xsl = zXmlDom.createDocument();           //创建 XMLDOM 对象
xml.onreadystatechange = function()           //为 xml 注册 onreadystatechange 事件处理程序
{
    /*
    * 当 readyState 为 1 时弹出对话框提示开始加载
    */
    if(xml.readyState == 1)
    {
        alert('开始加载 books.xml.');
```

```
xml.load("books.xml");           //加载 books.xml
xsl.load("books.xsl");           //加载 books.xsl
}
/*
 * 当 books.xml 和 books.xsl 都加载完成后, 使用 books.xsl 格式化 books.xml
 * 并将格式化后的内容输出到页面中
 */
function output()
{
    if(xml.readyState == 4 && xsl.readyState == 4)
    {
        document.getElementById('main').innerHTML = zXslt.transformToText(xml,xsl);
    }
}
</script>
</head>

<body>
    <div id="main"></div>
    <input type="button" value="load" onclick="loadXML();" />
</body>
</html>
```

示例程序的初始界面如图 10.14 所示。



图 10.14 初始界面

单击页面上的 load 按钮, 程序开始加载 books.xml 和 books.xsl, 并在加载完成后将 books.xml 格式化输出到页面上。其过程如图 10.15~图 10.21 所示。

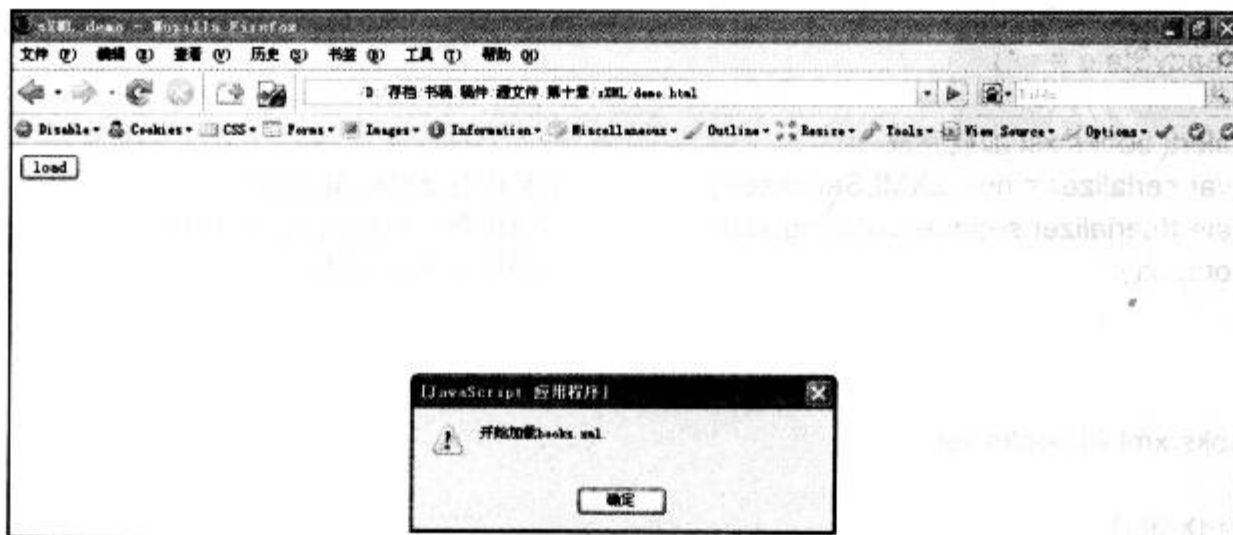


图 10.15 开始加载 books.xml

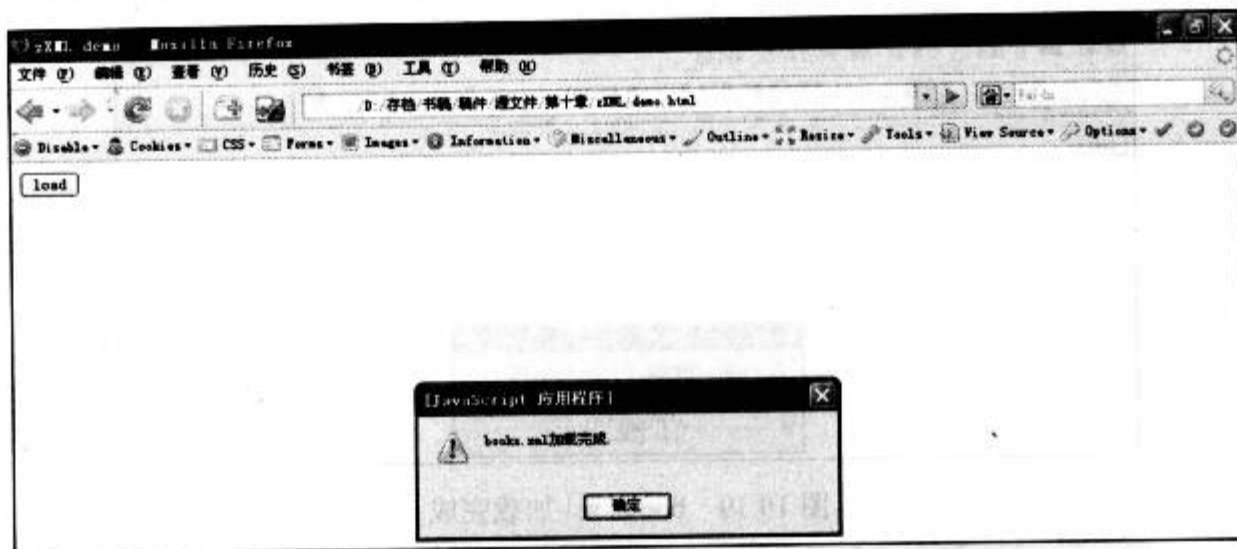


图 10.16 books.xml 加载完成

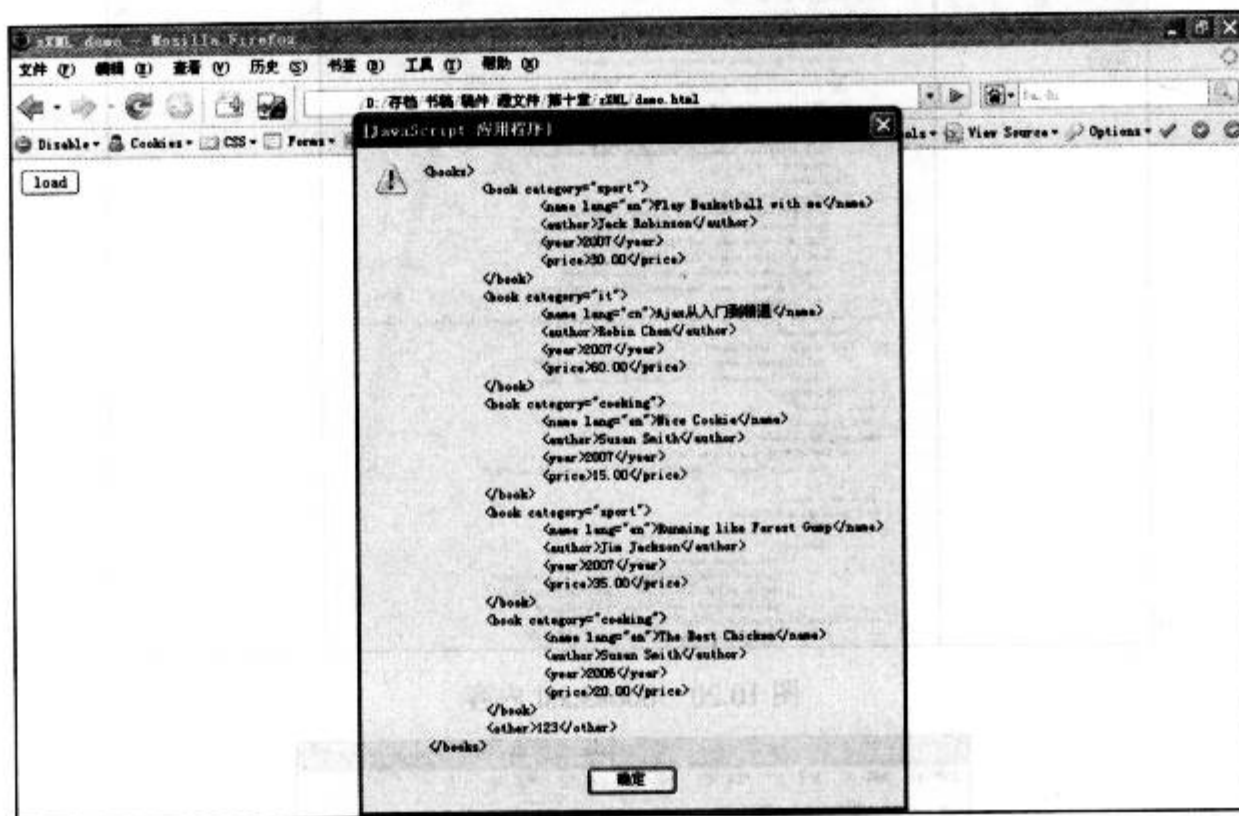


图 10.17 books.xml 内容

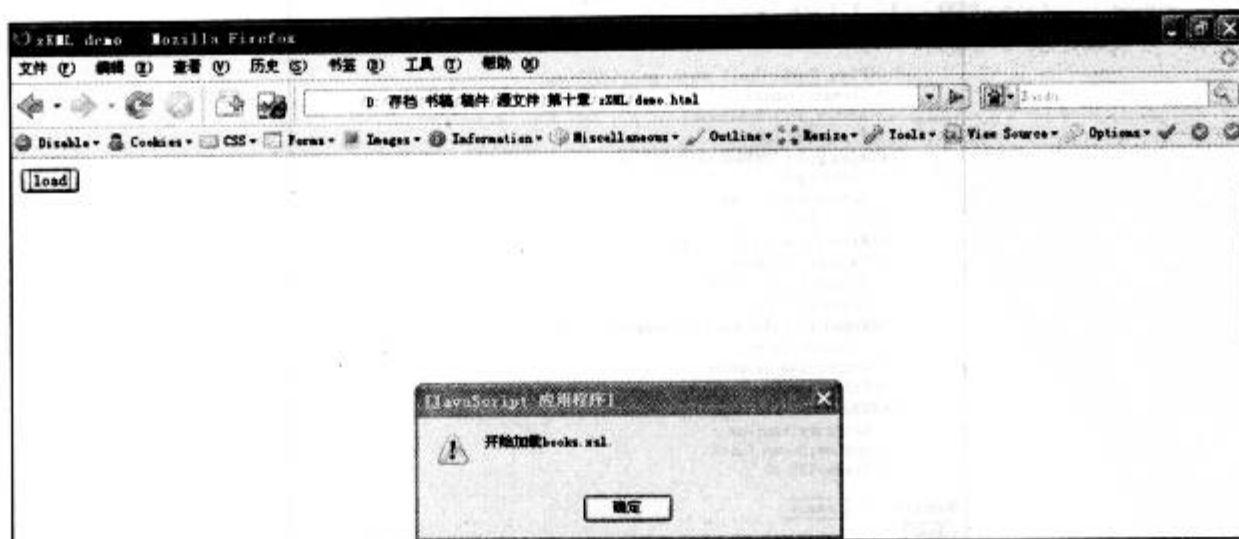


图 10.18 开始加载 books.xml



## 10.10 应用实例: Ajax 文章列表程序 (XML)

在介绍完关于 XML 各方面的知识后,现在回到 Ajax 这个主题上来。本节将使用本章中介绍的 XML 知识,并结合之前章节中向读者介绍的 Ajax 技巧,来完成一个常见的 Web 应用: Ajax 文章列表程序,也可以称之为简单的新闻系统。但是在这里只是完成前台的一部分,不提供后台的操作。下面首先来确定这个文章列表程序的需求。

### 10.10.1 确认需求

一个典型的文章列表程序提供用户根据文章分类来查看文章信息的功能。在程序界面上应该提供一个分类列表作为导航功能。当用户选择某个分类时,加载该类别下的文章数据并更新显示在界面上。当用户切换分类时,则重新加载新类别下的文章数据并更新显示在界面上。当文章数目过多时,为了防止加载时间过长,并且导致页面被撑得很大而影响浏览,需要提供分页显示的功能。并且用户可以自己选择每页显示的记录条数。数据方面,文章分类应该有分类名,文章应该包含标题、作者、发布时间、所属分类以及文章内容。文章系统的用例图如图 10.22 所示。

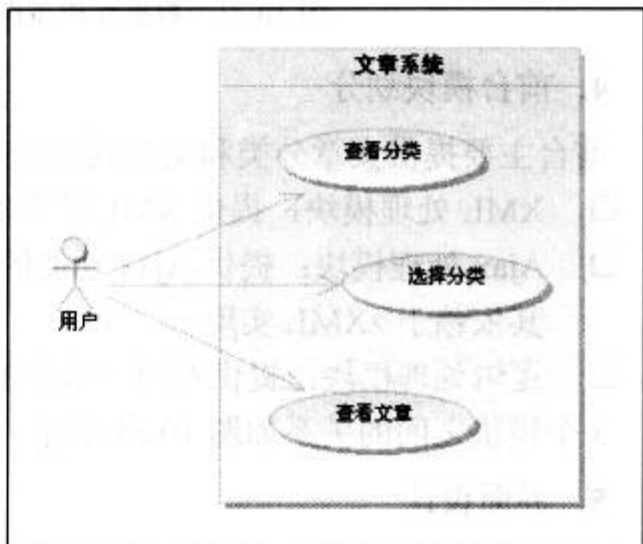


图 10.22 用例图

### 10.10.2 系统设计

在确定需求后,再来对系统进行设计。

#### 1. 系统环境

文章系统在如下环境下进行搭建。

- Web 服务器: Apache2.2.4。
- 服务端语言: PHP5.2.3。
- 数据库: Mysql5.0。

#### 2. 数据库设计

然后按照需求来做数据库的设计。将数据库命名为 article, 按照实体划分, 数据库由两个表组成: category 和 article。数据库的模型如图 10.23 所示。

#### 3. 后台模块划分

后台主要提供从数据库中查询数据并组织成 XML 输出, 这里简单地划分为两个模块。



- ❑ 数据处理模块：负责读取数据库中的数据并返回相应的 XML 表示。
- ❑ 接口模块：负责接收前台发送的请求，处理参数，调用数据处理模块读取数据，并返回操作结果。

两个模块之间的关系如图 10.24 所示。

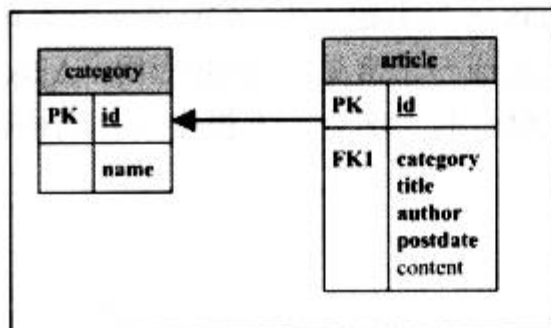


图 10.23 数据库模型图



图 10.24 后台模块

#### 4. 前台模块划分

前台主要提供文章分类和文章列表的查询和现实功能，按照功能将其划分为 3 个模块。

- ❑ XML 处理模块：提供 XML 的兼容性封装，由 zXML 完成。
- ❑ Ajax 处理模块：提供 Ajax 功能的底层支持，通过修改第 6 章中使用的 ajaxRequest.js 来完成。其依赖于 zXML 实现。
- ❑ 逻辑处理模块：提供对用户事件的处理以及完成界面的更新工作。

3 个模块之间的关系如图 10.25 所示。

#### 5. 界面设计

系统界面主要采用左右两栏结构，左栏为分类导航，右栏为文章列表，如图 10.26 所示。

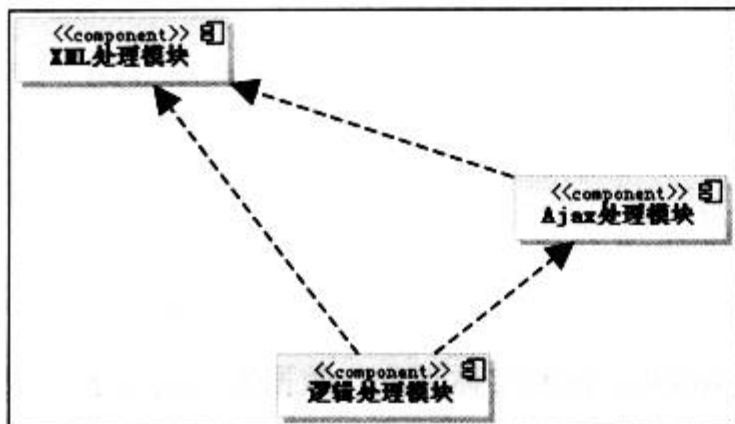


图 10.25 前台模块

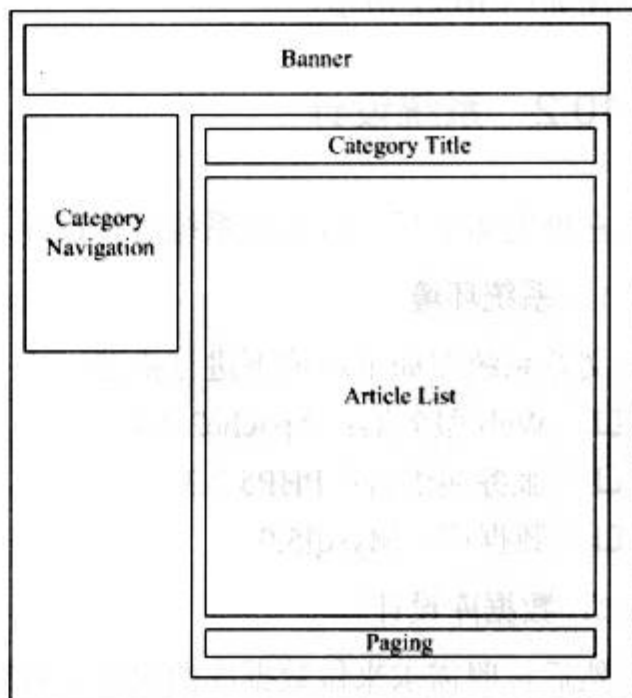


图 10.26 界面设计

## 6. 文件结构和清单

系统所包含的文件及文件结构, 如图 10.27 所示。

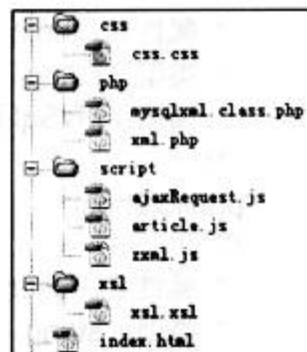


图 10.27 文件结构和清单

### 10.10.3 系统实现: 创建数据库

在做完系统设计后, 就可以着手开始进行系统实现了。

首先按照数据库模型来创建数据库, 将数据库命名为 `article`, 使用如下 SQL 脚本创建数据库及所需要的表和字段。

```

--
-- 数据库: 'article'
--
CREATE DATABASE 'article' DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
USE 'article';
-----

--
-- 表的结构 'article'
--
CREATE TABLE 'article' (
  'id' int(10) NOT NULL auto_increment,
  'category' int(10) NOT NULL default '1',
  'title' varchar(250) NOT NULL,
  'author' varchar(50) NOT NULL,
  'postdate' date NOT NULL,
  'content' text NOT NULL,
  PRIMARY KEY ('id')
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1;
-----

--
-- 表的结构 'category'
--
CREATE TABLE 'category' (
  'id' int(10) NOT NULL,
  'name' varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

### 10.10.4 系统实现: 完成后台模块

后台数据处理模块 `mysqlxml.class.php` 代码如下所示。

```

<?php
/*
 * 类名: MySqlXML
 */
class MySqlXML {
  /*

```

```
* 属性名: host
* 类型: string
* 描述: MySql 服务器
*/
public $host;
/*
* 属性名: user
* 类型: string
* 描述: MySql 用户名
*/
public $user;
/*
* 属性名: password
* 类型: string
* 描述: MySql 密码
*/
public $password;
/*
* 属性名: db
* 类型: string
* 描述: MySql 数据库名
*/
public $db;
/*
* 构造函数
*/
public function MySqlXML($host, $user, $password, $db)
{
    $this->host = $host;
    $this->user = $user;
    $this->password = $password;
    $this->db = $db;
}
/*
* 方法名: get_xml()
* 返回值: xml string
* 参数列表:
*     参数名: $table
*     类型: string
*     描述: 数据表名
*
*     参数名: $condition
*     类型: string
*     描述: 查询条件
*     默认值: ""
*
*     参数名: $pagesize
*     类型: int
*     描述: 每页返回的记录数
```

```

*      默认值: 0
*
*      参数名: $page
*      类型: int
*      描述: 当前页数
*      默认值: 0
*
*/
public function get_xml($table,$condition="", $pagesize=0,$page=0)
{
    $connection = mysql_connect($this->host, $this->user, $this->password, $this->db);
    mysql_select_db($this->db);
    $sql_string = "select count(*) as num from ".$table;
    if(!empty($condition))
    {
        $sql_string .= " where ".$condition;
    }
    $result = mysql_query($sql_string);
    while($row = @mysql_fetch_array($result))
    {
        $count = $row["num"];
    }
    $sql_string = "select * from ".$table;
    if(!empty($condition))
    {
        $sql_string .= " where ".$condition;
    }
    if($pagesize > 0)
    {
        $sql_string .= " LIMIT " . ($page*$pagesize).",".$pagesize;
    }
    $result = mysql_query($sql_string);
    $doc = new DOMDocument("1.0","utf-8");
    $doc->appendChild($doc->createElement("root"));
    $count_node = $doc->createElement("count");
    $count_node_text = $doc->createTextNode($count);
    $count_node->appendChild($count_node_text);
    $doc->documentElement->appendChild($count_node);
    while($row = mysql_fetch_array($result, MYSQL_ASSOC))
    {
        $num_fields = mysql_num_fields($result);
        $row_element = $doc->createElement(mysql_field_table($result, 0));
        $doc_root = $doc->documentElement;
        $row_element = $doc_root->appendChild($row_element);
        for ($i = 0; $i < $num_fields; $i++)
        {
            $field_name = mysql_field_name($result, $i);
            $col_element = $doc->createElement($field_name);
            $col_element = $row_element->appendChild($col_element);

```

```

        $text_node = $doc->createTextNode($row[$field_name]);
        $col_element->appendChild($text_node);
    }
}
mysql_free_result($result);
mysql_close($connection);
return $doc->saveXML();
}
}
?>

```

接口模块 xml.php 代码如下所示。

```

<?php
include("mysqlxml.class.php"); //引入 mysqlxml.class.php
header("Content-Type:text/xml;charset=utf-8"); //设置 Content-Type, 指定输出类型为
text/xml, 字符集编码为 utf-8
$mysqlxml = new MySqlXML("localhost","root","123","article"); //实例化 MySqlXML
/*
 * 如果 action 为 article, 则输出 article 相关的数据
 */
if($_REQUEST['action'] == 'article')
{
    /*
     * 如果没有指定 page 值, 则默认为 0
     */
    if(!is_numeric($_REQUEST["page"]))
    {
        $page = 0;
    }
    else
    {
        $page = $_REQUEST["page"];
    }
    /*
     * 如果没有指定 pagesize, 则默认为 5
     */
    if(!is_numeric($_REQUEST["pagesize"]))
    {
        $pagesize = 5;
    }
    else
    {
        $pagesize = $_REQUEST["pagesize"];
    }
    /*
     * 根据提交的 category 值来指定查询条件
     */
    if(!is_numeric($_REQUEST["category"]) || $_REQUEST["category"] <= 0)
    {

```



```

        $condition = "";
    }
    else
    {
        $condition = "category=".$_REQUEST["category"];
    }
    $xml = $mysqlxml->get_xml("article",$condition,$pagesize,$page);    //获取 xml
}
/*
 * 如果 action 值不为 article, 则默认输出 category 相关的数据
 */
else
{
    $xml = $mysqlxml->get_xml("category");    //获取 xml
}
echo $xml;    //输出 xml
?>

```

### 10.10.5 系统实现: 完成前台界面

前台界面由 index.html 和样式表 css.css 组成。index.html 的代码如下所示。

```

<!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>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <title>Article</title>
        <link type="text/css" rel="stylesheet" href="css/css.css" />
    </head>

    <body>
        <!-- Banner 开始 -->
        <h1>AJAX Article System</h1>
        <!-- Banner 结束 -->
        <!-- 分类导航开始 -->
        <div id="category">
            <ul>
                <li>loading..</li>
            </ul>
        </div>
        <!-- 分类导航结束 -->
        <div id="main">
            <h2>
                <!-- 页面尺寸选择 开始 -->
                <label for="pageSize" id="pageSizeSelector">
                    Page Size:
                    <select id="pageSize">
                        <option value="1">1</option>

```

```

        <option value="2">2</option>
        <option value="3">3</option>
        <option value="4">4</option>
        <option value="5" selected="selected">5</option>
        <option value="6">6</option>
        <option value="7">7</option>
        <option value="8">8</option>
        <option value="9">9</option>
        <option value="10">10</option>
    </select>
</label>
<!-- 页面尺寸选择 结束 -->
<!-- 分类标题 开始 -->
Category &raquo; <span id="currentCategory"></span>
<!-- 分类标题 结束 -->
</h2>
<!-- 文章列表开始 -->
<div id="list"></div>
<!-- 文章列表结束 -->
<!-- 分页开始 -->
<div id="paging">
</div>
<!-- 分页结束 -->
</div>
</body>
</html>

```

样式表 css.css 代码如下所示。

```

@charset "utf-8";
/* 总体样式定义 开始*/
*{
    margin:0;
    padding:0;
}
body {
    font-family:Tahoma,"宋体";
    font-size:12px;
    padding:10px;
}
/* 总体样式定义 结束*/
/* 一级标题样式(banner) 开始*/
h1 {
    display:block;
    background-color:#666;
    line-height:30px;
    padding:5px;
    clear:both;
    color:#fff;
    width:760px;

```

```

        font-size:24px;
        text-align:center;
    }
    /* 一级标题样式(banner) 开始*/
    /* 分类列表样式 开始 */
    #category {
        float:left;
        width:120px;
        margin:10px 20px 0 0;
        background-color:#666;
    }
    /* 列表样式 开始 */
    #category ul {
        padding:10px 0;
    }
    #category li {
        line-height:15px;
        color:#fff;
        list-style:none;
    }
    /* 分类链接按钮样式 开始 */
    #category li a {
        color:#fff;
        display:block;
        border-bottom:1px solid #fff;
        text-decoration:none;
        line-height:20px;
        padding-left:5px;
    }
    /* 分类链接按钮 鼠标滑过样式 开始 */
    #category li a:hover {
        font-weight:bold;
        background-color:#999;
    }
    /* 分类链接按钮 鼠标滑过样式 结束 */
    /* 分类链接按钮 当前选中项 开始 */
    #category li a.current {
        font-weight:bold;
        background-color:#999;
    }
    /* 分类链接按钮 当前选中项 结束 */
    /* 分类链接按钮样式 结束 */
    /* 列表样式 结束 */
    /* 分类列表样式 结束 */
    /* 主内容区样式 开始 */
    #main {
        margin-top:10px;
        width:608px;
        float:left;

```

```

        border: 1px solid #666;
        padding: 10px;
    }
    #main span {
        margin-right: 10px;
        font-size: 10px;
    }
    h2 {
        border-bottom: 1px solid #666;
        margin-bottom: 5px;
        float: left;
        font-size: 10px;
    }
    #categoryName {
        float: left;
    }
    #pageSizeSelector {
        float: right;
    }
    #list {
        clear: both;
        border-bottom: 1px solid #666;
        padding-bottom: 10px;
    }
    #list li {
        list-style: none;
    }
    h3 {
        margin-top: 10px;
    }
    /* 分页样式区 开始 */
    #paging {
        float: right;
    }
    /* 分页按钮样式 开始 */
    #paging a {
        margin-right: 5px;
        text-decoration: underline;
        color: #000;
    }
    /* 分页按钮 鼠标滑过样式 开始 */
    #paging a:hover {
        color: #f00;
    }
    /* 分页按钮 鼠标滑过样式 结束 */
    /* 分页按钮 当前选中项样式 开始 */
    #paging a.current {
        font-weight: bold;

```

```

text-decoration:none;
}
/* 分页按钮 当前选中项样式结束 */
/* 分页按钮样式 结束 */
/* 分页样式区 结束 */
/* 主内容区样式 结束 */

```

界面效果如图 10.28 所示。

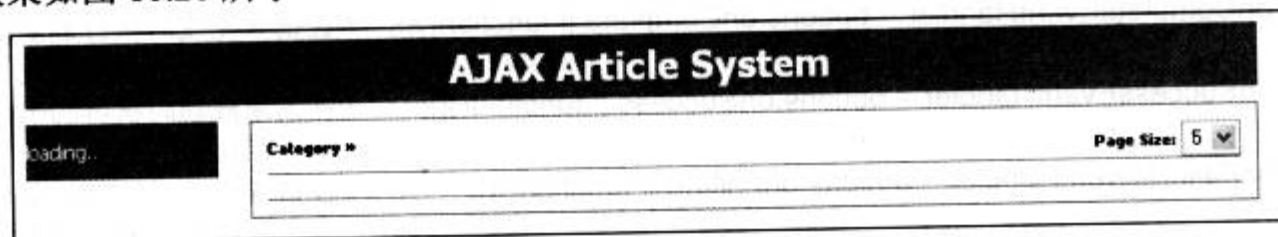


图 10.28 界面效果图

### 10.10.6 系统实现：完成前台模块

XML 处理模块由 zXML 完成，这里省略详细代码。Ajax 处理模块 ajaxRequest.js 代码如下所示。

```

/*
 * 函数名: ajaxRequest
 * 功能: 根据用户指定的 URL、方法、参数、HTTP 头，以及回调函数，自动创建 XMLHttpRequest 对象并发送
请求。
 * 参数介绍:
 *      url: 请求的 URL 地址，字符串型
 *      options: 参数集合，对象类型，其成员均为可选，成员为:
 *          method: 发送请求的方法，可以为 GET 或 POST，字符串型
 *          parameters: 需要发送的数据，字符串型，其形式为"a=1&b=2&c=3"
 *          headers: 需要发送的 HTTP 头信息，对象类型。其每一个属性为一个头信息
属性名为头信息的名称，属性值为头信息的内容
 *          onLoading: 在请求开始时执行的函数
 *          onComplete: 在请求完成时执行的函数
 *          onSuccess: 在请求成功时执行的函数
 *          onFailure: 在请求失败时执行的函数
 */
function ajaxRequest(url,options)
{
    var request = zXmlHttp.createRequest();
    if(typeof request == 'undefined')
        //创建 XMLHttpRequest 对象
        //如果 request 为 undefined，则抛出异常，
        //说明当前浏览器不支持 XMLHttpRequest，
        //并退出函数

    {
        throw new Error("Your browser does not support XMLHttpRequest");
        return;
    }
    var url = url;
    var method = (options.method || 'POST').toUpperCase();
    if(method != 'GET' && method != 'POST')
        //获取提交方式，默认为 POST

```



```

{
    method = 'POST';
}
var parameters = options.parameters || null;           //需要提交的参数, 默认为 null
var headers = options.headers || {};                  //需要发送的 HTTP 头信息, 是一个对象, 其成员包含
                                                    //了头信息的名称和值的信息
var onLoadingEventHandler = options.onLoading || function(){}; //在请求开始时执行的函数, 由用户指
                                                    //定, 默认为一个空函数
var onCompleteEventHandler = options.onComplete || function(){}; //在请求完成时执行的函数, 由用户指
                                                    //定, 默认为一个空函数
var onSuccessEventHandler = options.onSuccess || function(){}; //在请求成功时执行的函数, 由用户指
                                                    //定, 默认为一个空函数
var onFailureEventHandler = options.onFailure || function(){}; //在请求失败时执行的函数, 由用户指
                                                    //定, 默认为一个空函数
if(method == 'GET' && parameters != null)             //如果提交方式被指定为 GET, 则将 parameters 的内
                                                    //容拼接到 URL 中, 并将 parameters 设置为 null
{
    if(url.indexOf('?') > -1)
    {
        url += '&' + parameters;
    }
    else
    {
        url += '?' + parameters;
    }
    parameters = null;
}
request.open(method,url,true);                        //初始化 XMLHttpRequest 对象
request.setRequestHeader('contentType','application/x-www-form-urlencoded');
for(var name in headers)                             //设置由用户指定的 HTTP 头信息
{
    request.setRequestHeader(name,headers[name]);
}
request.onreadystatechange = function() //注册 XMLHttpRequest 对象的 readystatechange 事件处理函数
{
    if(request.readyState == 1)                 //当 readyState 等于 1 时, 表示请求开始, 将当前 XMLHttpRequest
                                                    //对象作为其参数调用 onLoadingEventHandler 函数
    {
        onLoadingEventHandler(request);
    }
    if(request.readyState == 4)                 //当 readyState 等于 4 时, 表示请求完成, 将当前 XMLHttpRequest
                                                    //对象作为其参数调用 onCompleteEventHandler 函数
    {
        onCompleteEventHandler(request);
        if(request.status && request.status >= 200 && request.status < 300)
        {
            onSuccessEventHandler(request); //当 HTTP 状态码大于等于 200 小于 300 时, 表示请求
                                                    //成功, 这时将当前 XMLHttpRequest 对象作为其参数
                                                    //调用 onSuccessEventHandler 函数
        }
    }
    else

```

```

        {
            onFailureEventHandler(request); //否则表示请求失败, 这时将当前
XMLHttpRequest 对象作为其参数调用 onFailureEventHandler 函数
        }
    }
}
request.send(parameters); //发送请求
return request;
}

```

逻辑处理模块 article.js 代码如下所示。

```

var url = 'php/xml.php'; //初始化后台数据接口地址
var currentCategory = 0; //初始化分类为 0
var currentPage = 0; //初始化页为 0
var currentPageSize = 5; //初始化页面尺寸为 5
var requestManager = null; //初始化请求管理器
var xsl = zXmlDom.createDocument(); //创建 xsl document object
/*
 * 注册 xsl 的 readystatechange 事件处理函数, 在 xsl 加载完后初始化文章列表程序
 */
xsl.onreadystatechange = function()
{
    if(xsl.readyState == 4)
    {
        init(); //调用初始化程序
    }
}
xsl.load('xsl/xsl.xsl'); //加载 xsl/xsl.xsl 文档
/*
 * 函数名: init()
 * 功能说明:
 *     初始化文章列表程序
 *     使用 Ajax 读取分类列表并使用 xsl 格式化分类列表后, 更新在界面上指定的位置。
 *     在更新完分类列表后, 将分类列表中第一个分类作为当前分类, 并读取该分类下的文章信息。
 */
function init()
{
    ajaxRequest(url,{ //发起一个 Ajax 请求
        onComplete:function(x) //注册在请求完成时被调用的函数
        {
            var xml = zXmlDom.createDocument(); //创建 xml dom 对象
            xml.loadXML(x.responseText); //加载返回的 xml 片段
            var category = zXPath.selectSingleNode(xml,'//category/id').text; //读取第一个分类
            /*
             * 等待页面中需要操作的 HTML 结构加载完后, 更新分类列表
             */
            (function()
            {
                if(document.getElementById('category'))

```

```

*      参数说明: 数据所在的页数
*
*      参数名: size
*      参数类型: int
*      参数说明: 每页包含的记录数
*
*      参数名: isRefreshPaging
*      参数类型: bool
*      参数说明: 读取数据完成后是否同时更新分页按钮,为 true 时更新
* 功能说明:
*      根据给定的参数读取文章信息并更新界面
*/
function showArticles(categoryId,page,size,isRefreshPaging)
{
    /*
    * 如果当前存在未完成的请求, 则取消该请求
    */
    if(requestManager != null)
    {
        requestManager.abort();           //取消已经存在的请求
        requestManager = null;
    }
    /*
    * 根据给定的参数重新发起请求以读取文章数据。
    */
    requestManager = ajaxRequest(url,{
        method:'get',
        parameters:'action=article&category=' + categoryId + '&page=' + page + '&pagesize=' + size,
        /*
        * 在加载过程中, 界面上显示"loading.."字样。
        */
        onLoading:function()
        {
            document.getElementById('list').innerHTML = 'loading..';
        },
        /*
        * 加载完成后更新界面, 以及按需要更新分页按钮。
        */
        onComplete:function(x)
        {
            var xml = zXmlDom.createDocument();           //创建 xml dom 对象
            xml.loadXML(x.responseText);                 //加载返回的 xml 片段
            var count = zXPath.selectSingleNode(xml,'/root/count').text; //读取总记录数
            var items = zXPath.selectNodes(xml,'/article').length;        //读取当前返回的记录数
            if(isRefreshPaging)buildPaging(count);        //如果 isRefreshPaging 被设置为 true, 则更新分页按钮
            /*
            * 如果总记录数和当前返回的记录数都大于 0,则将 xml 数据使用 xsl 格式化后更新到界面中。否
            则在界面上显示"no record"(没有记录)。
            */

```

```

        if(count > 0 && items > 0)
        {
            document.getElementById('list').innerHTML = zXslt.transformToText(xml,xsl);
        }
        else
        {
            document.getElementById('list').innerHTML = 'No Record!';
        }
        requestManager = null;
    }
});
}
/*
 * 函数名: buildPaging()
 * 参数列表:
 *     参数名: count
 *     参数类型: int
 *     参数说明: 总记录数
 * 功能说明:
 *     根据总记录数自动计算总页数, 并更新界面上的分页按钮
 */
function buildPaging(count)
{
    document.getElementById('paging').innerHTML = ""; //清空 div#paging 中的内容
    var count = count;
    var size = currentPageSize;
    var pageCount = Math.ceil(count/size); //计算总页数
    currentPage = 0; //设置当前页为 0
    /*
     * 循环创建所有的分页按钮
     */
    for(var i = 0; i < pageCount; i++)
    {
        var pageLink = document.createElement('a'); //创建 a 元素
        if(i == 0)pageLink.className = 'current'; //如果当前为第一个元素, 则将 a 元素的 css 类设置为 current
        pageLink.id = 'pageLink' + i; //设置 a 元素的 id
        pageLink.href = '#'; //设置 a 元素的链接
        pageLink.innerHTML = i + 1; //设置 a 元素的文本内容
        /*
         * 注册分页按钮的 click 事件处理程序
         * 在 click 事件被触发时, 切换分页按钮的样式, 并重新获取数据更新界面
         */
        pageLink.onclick = function()
        {
            if(currentPage == parseInt(this.innerHTML) - 1)return; //如果单击的是当前页, 则不发生任何行为
            document.getElementById('pageLink' + currentPage).className = ""; //切换分页按钮样式
            currentPage = parseInt(this.innerHTML) - 1; //设置当前页
            document.getElementById('pageLink' + currentPage).className = 'current'; //切换分页按钮样式
            showArticles(currentCategory,currentPage,currentPageSize,false); //重新获取新页的数据
        }
    }
}

```

并更新界面

```

        return false; //阻止 a 元素的默认 click 行为
    }
    document.getElementById('paging').appendChild(pageLink); //将分页按钮加入到 div#paging 中
}
}
/*
 * 函数名: changePageSize()
 * 功能说明:
 *     根据页面上选择的页面尺寸大小来更新界面
 */
function changePageSize()
{
    var selector = document.getElementById('pageSize');
    currentPageSize = selector.options[selector.selectedIndex].value; //取得选择的页面尺寸值
    changeCategory(currentCategory,true); //更新界面数据
}

```

### 10.10.7 系统实现: 编写 xsl 样式表

xsl 样式表 xsl.xsl 代码如下所示。

```

<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="utf-8"/>
<!-- 根模板 开始 -->
<xsl:template match="/">
    <xsl:apply-templates select="root"></xsl:apply-templates>
</xsl:template>
<!-- 根模板 结束 -->
<!-- root 元素模板 开始 -->
<xsl:template match="root">
    <ul>
        <xsl:apply-templates></xsl:apply-templates>
    </ul>
</xsl:template>
<!-- root 元素模板 结束 -->
<!-- category 元素模板 开始 -->
<xsl:template match="/root/category">
    <li>
        <xsl:element name="a">
            <xsl:attribute name="id">category<xsl:value-of select="id"></xsl:value-of></xsl:attribute>
            <xsl:attribute name="href">#</xsl:attribute>
            <xsl:attribute name="onclick">changeCategory(<xsl:value-of select="id"></xsl:value-of>);return
false;</xsl:attribute>
            <xsl:attribute name="title">category:<xsl:value-of select="name"></xsl:value-of></xsl:attribute>
            <xsl:value-of select="name"></xsl:value-of>
        </xsl:element>
    </li>
</xsl:template>

```



```

<!-- category 元素模板 结束 -->
<!-- article 元素模板 开始 -->
<xsl:template match="/root/article">
    <li>
        <h3><xsl:value-of select="title"></xsl:value-of></h3>
        <span><xsl:value-of select="author"></xsl:value-of></span>
        <span><xsl:value-of select="postdate"></xsl:value-of></span>
        <p><xsl:value-of select="content"></xsl:value-of></p>
    </li>
</xsl:template>
<!-- article 元素模板 结束 -->
<!-- count 元素模板 开始 -->
<xsl:template match="/root/count"></xsl:template>
<!-- count 元素模板 结束 -->
</xsl:stylesheet>

```

### 10.10.8 整合系统

将前台各模块引入 index.html，并对 index.html 作适当的修改，代码如下所示。

```

<!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>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <title>Article</title>
        <link type="text/css" rel="stylesheet" href="css/css.css" />
        <script type="text/javascript" src="script/zxml.js"></script><!--引入 zXML 框架-->
        <script type="text/javascript" src="script/ajaxRequest.js"></script><!--引入 ajaxRequest.js-->
        <script type="text/javascript" src="script/article.js"></script><!--引入 article.js-->
    </head>

    <body>
        <!-- Banner 开始 -->
        <h1>AJAX Article System</h1>
        <!-- Banner 结束 -->
        <!-- 分类导航开始 -->
        <div id="category">
            <ul>
                <li>loading..</li>
            </ul>
        </div>
        <!-- 分类导航结束 -->
        <div id="main">
            <h2>
                <!-- 页面尺寸选择 开始 -->
                <label for="pageSize" id="pageSizeSelector">
                    Page Size:
                    <select id="pageSize" onchange="changePageSize();">

```

```

        <option value="1">1</option>
        <option value="2">2</option>
        <option value="3">3</option>
        <option value="4">4</option>
        <option value="5" selected="selected">5</option>
        <option value="6">6</option>
        <option value="7">7</option>
        <option value="8">8</option>
        <option value="9">9</option>
        <option value="10">10</option>
    </select>
</label>
<!-- 页面尺寸选择 结束 -->
<!-- 分类标题 开始 -->
Category &raquo; <span id="currentCategory"></span>
<!-- 分类标题 结束 -->
</h2>
<!-- 文章列表开始 -->
<div id="list"></div>
<!-- 文章列表结束 -->
<!-- 分页开始 -->
<div id="paging">
</div>
<!-- 分页结束 -->
</div>
</body>
</html>

```

### 10.10.9 系统测试

在完成系统的实现后,现在可以开始对系统进行测试。在测试前,先向数据库中添加一些分类和文章的数据,代码如下所示。

```

--
-- 插入分类数据 `category`
--
INSERT INTO `category` (`id`, `name`) VALUES
(1, 'AJAX'),
(2, 'HTML'),
(3, 'JavaScript'),
(4, 'PHP'),
(5, 'ASP.NET'),
(6, 'JSP');
--
-- 插入文章数据 `article`
--
INSERT INTO `article` (`id`, `category`, `title`, `author`, `postdate`, `content`) VALUES
(1, 1, 'Learn Ajax', 'Robin Chen', '2007-11-25', 'Something about the book..'),
(2, 1, 'HTML Study', 'Robin Chen', '2007-11-25', 'Learning HTML with me.'),

```

- (3, 1, 'How to use the XMLDocument', 'Robin Chen', '2007-11-25', 'How to use the XMLDocument in PHP? Follow me ,step by step..'),
- (4, 3, 'javascript doesnt stop', 'stephan.gerlach', '2007-11-27', 'Ok I am trying to write a script which works great except that it doesnt stop loading.\r\n\r\nUsually I don"t just post all my code but in this case I just can"t find the problem and I hope someone of you guys does.\r\n\r\nOn the actual page i have a little code like this...'),
- (5, 3, 'show hide form field help - newbie', 'Steve Field', '2007-11-27', 'I am in need of help for a form that I have written. I want to hide certain form field items and only display them based on what the user selects from a drop down list. What is the best way to do this.\r\n\r\nMy form is located at the following address [http://gomavs.unomaha.edu/booster\\_cl...orms/index.php](http://gomavs.unomaha.edu/booster_cl...orms/index.php)\r\n\r\nI do not want the fields for T-ShirtSize, CapSize, and GolfShirtSize to display initally.\r\n\r\nIf the user selects double - I want T-ShirtSize to display. If the user selects triple, I want T-ShirtSize and CapSize to display. If the user selects Home Run, I want only GolfShirtSize to display.\r\n\r\nAny insight you can give me would greatly help. I have been trying to get this to work for about a week now by using various methods I have found on the internet, but I have yet to be successful. Thanks for your help.').
- (6, 3, 'Change Option List Selected Based on ID', 'nnhubbard', '2007-11-27', 'I am wondering if it is possible to have the default item in a selection list, be based on what text is in another field or inside another div.\r\n\r\nFor example...'),
- (7, 3, 'How to get cross domain-access to objects', 'bfrordorf', '2007-11-26', 'Hi friends\r\n\r\nHow can I work with jScript Objects on a foreign WEB page (cross domain-access) ? Problem I am working on different WEB servers containing data bases. I have designed a framed page in order to transfer form-input information from a source page in frame[1] to a target page in frame[2]. The source page displays information and contains hidden input fields with ID"s. The target page contains an WEB-form with input fields containing id"s.\r\n\r\nThe JScript for this is for instance...'),
- (8, 3, '1 input Date = 2 different Date formats', 'adalby', '2007-11-27', 'Hello I need to be able to select from date drop downs and have a second date formatted differently populate a hidden field. I have this script below that works very closely this way however it just copies that same formatted date into a hidden field. The month is the true problem...\r\n\r\nThe two formats that I need are:\r\n1. November 26, 2007\r\n2. 11/26/2007'),
- (9, 3, 'Events triggering on closing specific tab.', 'nighthawk', '2007-11-25', 'Just a quick question and can be a very high level answer, no detail required hehe\r\n\r\nI have a quote page, which when a client receives a quote if he doesn"t hit "save" but just closes the window then it triggers an event that sends an email out to him telling him the reference number so he can access it again.\r\n\r\nNow with tabbed browsing I"m being told (by the programmers) the goalposts have changed somewhat.\r\n\r\nI want to do the same thing if the client closes his Tab in a tabbed browser.\r\n\r\nI"m being told that if the client has 10 tabs open and he closes the browser(thus closing all 10 tabs) the event (email being sent out) will trigger 10 times.\r\n\r\nI"m not a hardcore coder but I would of thought that you could target the specific tab your application was appearing on and just trigger the event for that tab closing?!?\r\n\r\nHopefully that made sense, basically I need to know if I"m being told the classic \*rub the chin it"ll cost you\* IT equivalent of a plumbers excuse'),
- (10, 1, 'Ajax error', 'cooldyood', '2007-11-26', 'Please excuse the length of this post. I"m getting an error and am totally stuck! On submitting the login form from my website, the getFormLogin() function below is called and I get the error "There was a problem with the request". I must mention that postlogin.php is in a different directory than other files.\r\n\r\nSince formating was automatically removed from the code, I"ve attached it with formatting preserved for easier reading.\r\n\r\nAny help will be greatly appreciated!'),
- (11, 2, 'HTML editor you guys use', 'jjbarnone', '2007-11-23', 'Hello everyone,\r\n\r\nI"m curious which HTML editor you use for handwriting HTML/CSS/JS?\r\n\r\nI"ve been using a pretty good one for a few years now and like it, but I"m always tinkering with other ones and wonder what other people use.\r\n\r\njjj'),
- (12, 2, 'Append to an HTML file?', 'Jaxeed', '2007-11-15', 'Hey everyone!\r\n\r\nIntroduction:\r\n\r\nOn one of my most recent assignments I've run into a problem I so far haven't found a solution for. The website can ONLY use HTML, CSS and JavaScript, though needs a function I've never used in simple HTML before. However, I would think this would be a function that very well might exist for HTML or perhaps JavaScript, though for all I know it might not. So here it goes...\r\n\r\nCase:\r\n\r\nIs it possible to append content into an HTML page from a different HTML page, without using frames or iframes? ...');)

然后将系统所有文件部署到 Apache 下, 用浏览器访问 `index.html` 的地址。页面打开后, 程序首先会读取 `xsl` 文档并缓存在本地, 然后读取分类信息, 并使用 `xsl` 格式化后更新显示在页面左边的分类导航区内, 接着会将第一个分类设置为当前分类, 并读取该分类下的文章数据显示在界面上, 如图 10.29 所示。

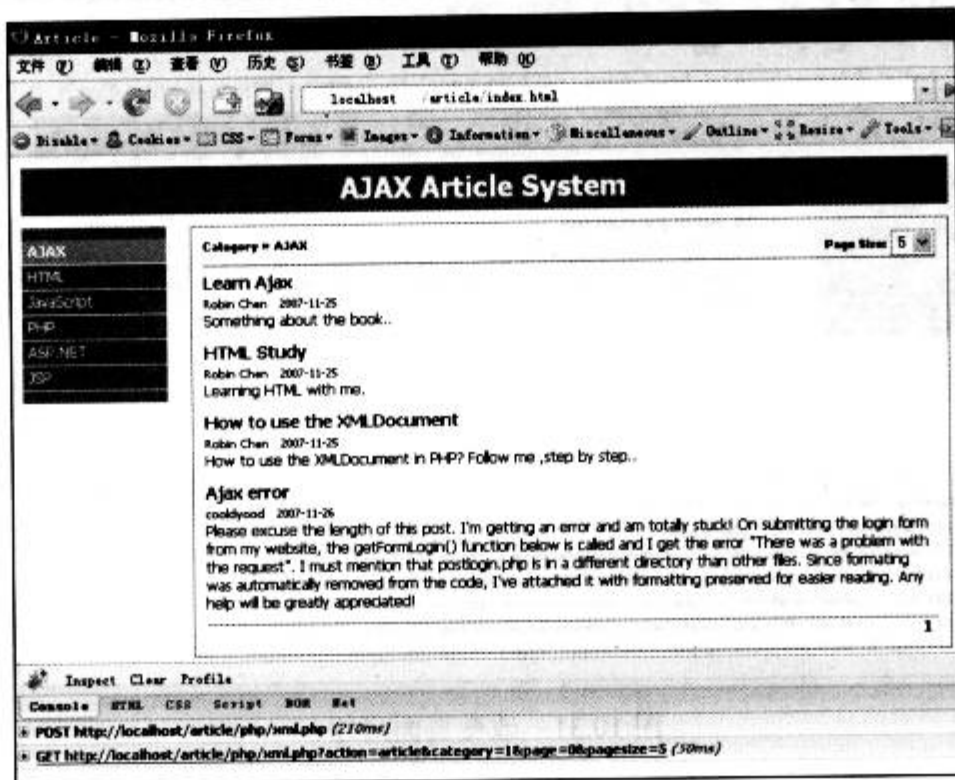


图 10.29 测试系统

在左边的分类导航中, 当前分类使用了不同的背景色和文字颜色以进行区分。单击分类 HTML 查看 HTML 分类下的数据, 如图 10.30 所示。

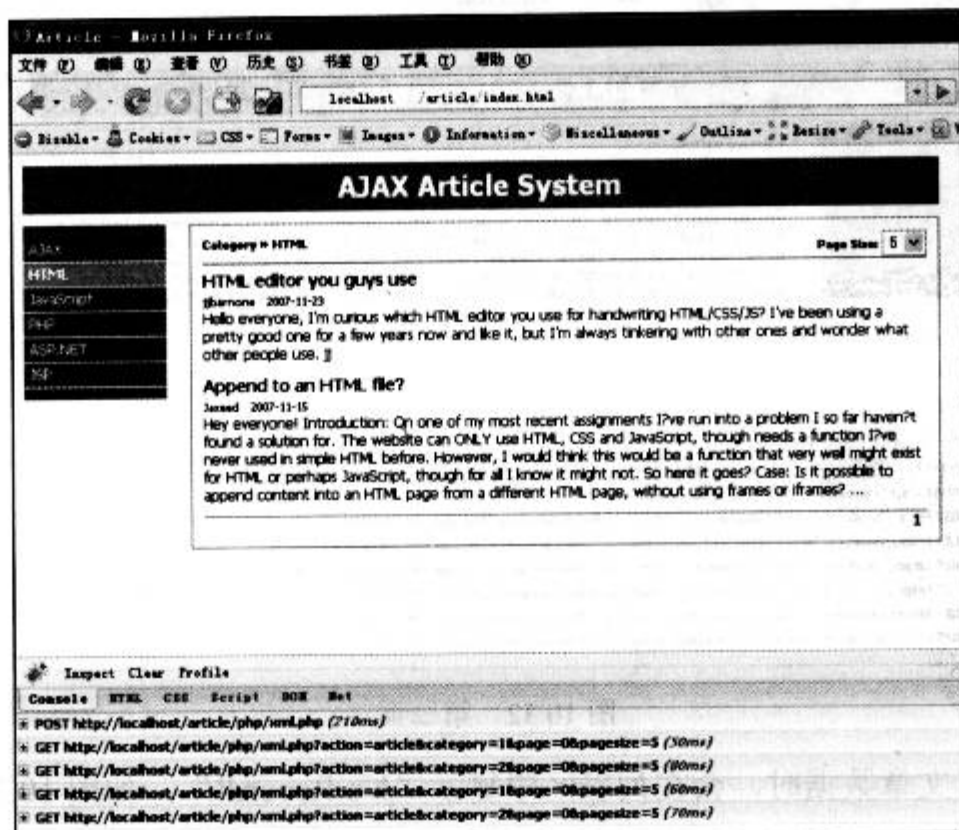


图 10.30 HTML 分类



页面右边的文章列表中的数据被更新了，显示的是 HTML 分类下的文章信息。文章列表右上角是一个选择文章列表每页记录数的下拉列表框，默认为 5 条。将下拉列表框选定为 1，文章列表被更新，每页只显示一条记录，在列表下面提供了分页按钮，如图 10.31 所示。

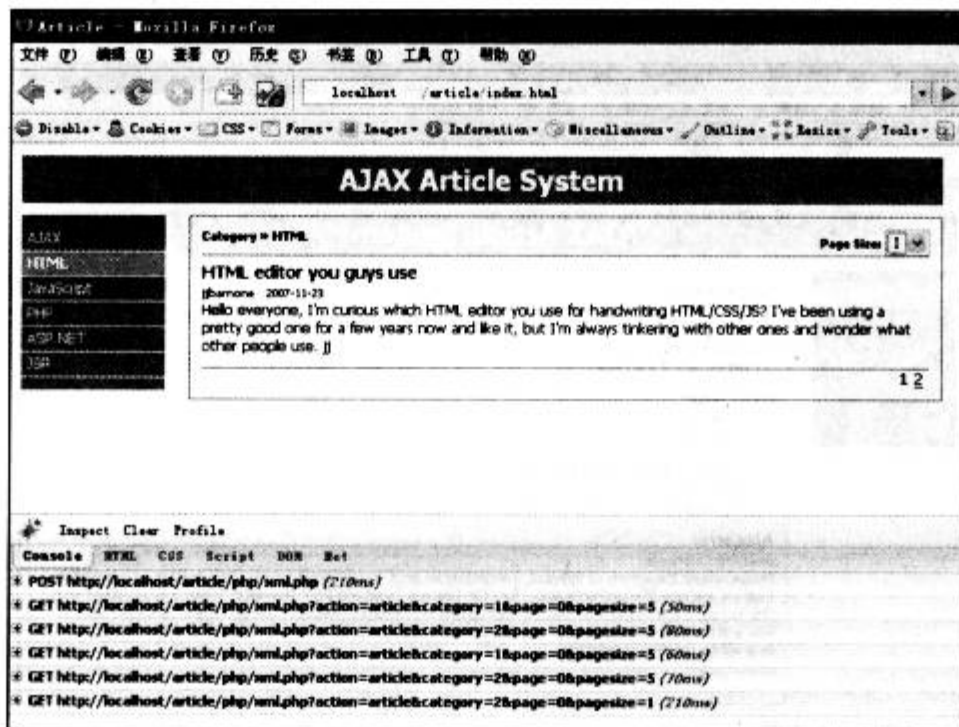


图 10.31 改变页面尺寸

分页按钮中，当前页面被加粗显示并去掉了下划线以此进行区分。单击表示第二页的链接按钮，列表再次被更新，显示的是第二页的数据，同时分页按钮的样式进行了切换，如图 10.32 所示。

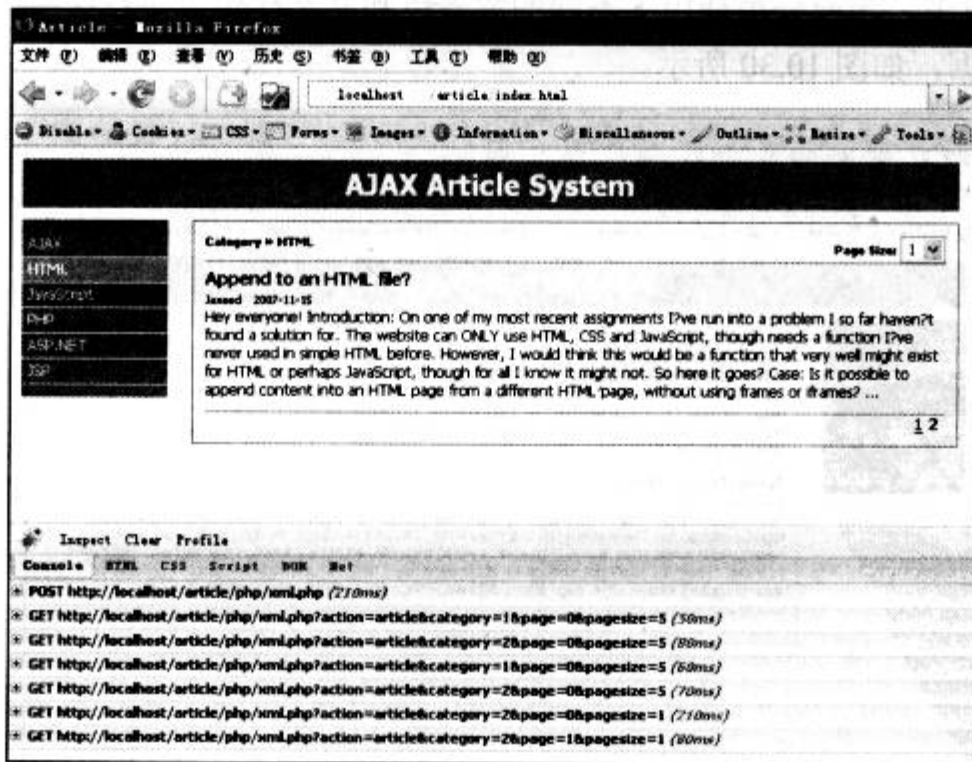


图 10.32 第二页

当某个分类下没有文章数据时，会在列表区中显示 “No Record!”，如图 10.33 所示。



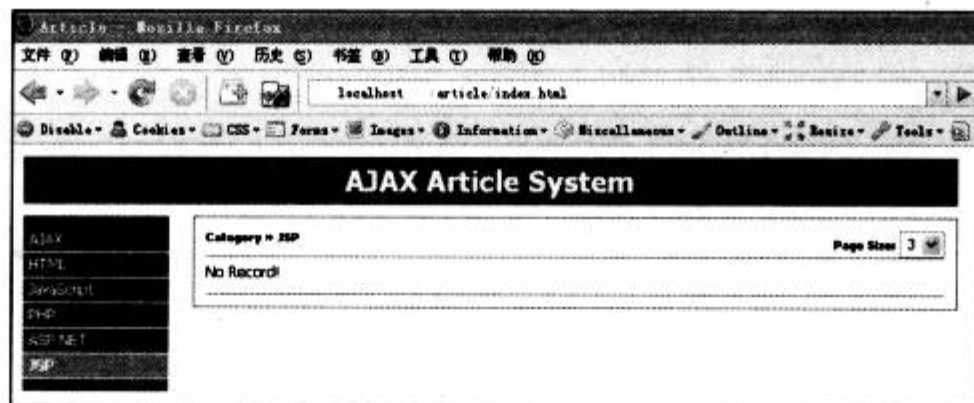


图 10.33 没有数据

## 10.11 小 结

本章向读者详细介绍了有关 XML 的各方面知识。首先向读者介绍了 XML 的基本知识, 包括 XML 的语法规则和命名空间。然后向读者介绍了 XML 的有效性验证工具: XML Schema。本章还在基于前面章节介绍的 DOM (文档对象模型) 基础上, 针对 XML DOM 进行了补充说明, 包括 XML DOM 对象的创建和加载。然后介绍了如何使用 XPath 来获取 XML 的节点, 以及基于 XPath 的 XML 格式化工具: XSL。之后介绍了如何在客户端格式化 XML, 并针对在不同浏览器环境下处理 XML 的兼容性问题, 向读者介绍了 zXML 框架的使用方法。最后基于本章所介绍的知识, 创建了一个使用 XML 封装和传输数据的 Ajax 应用: Ajax 文章列表程序。



图 10.11 网页显示

## 10.11 小结

本章介绍了 JMX 的各个方面。首先，我们介绍了 JMX 的各个方面，包括 JMX 的各个方面。然后，我们介绍了 JMX 的各个方面，包括 JMX 的各个方面。最后，我们介绍了 JMX 的各个方面，包括 JMX 的各个方面。

本章介绍了 JMX 的各个方面。首先，我们介绍了 JMX 的各个方面，包括 JMX 的各个方面。然后，我们介绍了 JMX 的各个方面，包括 JMX 的各个方面。最后，我们介绍了 JMX 的各个方面，包括 JMX 的各个方面。

本章介绍了 JMX 的各个方面。首先，我们介绍了 JMX 的各个方面，包括 JMX 的各个方面。然后，我们介绍了 JMX 的各个方面，包括 JMX 的各个方面。最后，我们介绍了 JMX 的各个方面，包括 JMX 的各个方面。

# 第 11 章

## 数据的组织方式：JSON

- » JSON 的语法结构
- » JSON 的语言支持
- » JSON 的优点和不足
- » 将 XML 转换为 JSON
- » 小结

JSON 是英文 JavaScript Object Notation 的简写，即 JavaScript 对象符号的意思。顾名思义，JSON 使用类似于 JavaScript 对象直接量的语法来描述数据，它是一种轻量级的数据交换方式，易于阅读和编写，也易于机器的解析和生成。JSON 基于 JavaScript 语言模型实现，但是又完全独立于语言本身。它保留了 C 语言（C、C++、C#、Java、JavaScript、Perl、Python 等）家族的一些习惯。这些特性使得 JSON 成为理想的数据交换语言。

在第 10 章中向读者介绍了 XML 的相关知识, XML 以其成熟性和广泛的支持成为了 Ajax 应用中数据传输的代表方式。但是随着 Ajax 技术的不断发展, Ajax 应用的不断深入, 一种新的数据组织方式被提出并迅速得到了广泛的使用和赞誉, 它就是 JSON。

一个描述某个图书数据的 JSON 的示例代码如下所示。

```
{
  "book":{
    "name":"Ajax",
    "publishDate":"2007-11-28",
    "author":{
      "name":"Robin Chen",
      "sex":"male"
    },
    "price":88.00,
    "tags":["Ajax","HTML","JavaScript","XMLHttpRequest"]
  }
}
```

与其等价的 XML 代码如下所示。

```
<book>
  <name>Ajax</name>
  <publishDate>2007-11-28</publishDate>
  <author>
    <name>Robin Chen</name>
    <sex>male</sex>
  </author>
  <price>88.00</price>
  <tags>Ajax</tags>
  <tags>HTML</tags>
  <tags>JavaScript</tags>
  <tags>XMLHttpRequest</tags>
</book>
```

## 11.1 JSON 的语法结构

JSON 的语法结构非常简单, 熟悉 JavaScript 就能够编写 JSON。尽管如此, 本节还是要对 JSON 的语法结构做详细介绍, 以加深读者对 JSON 的理解。

### 11.1.1 JSON 的基本结构

JSON 有两种基本结构: 名/值对的集合和值的有序列表。

名/值对的集合在不同语言中被理解为对象(object)、记录(record)、结构(struct)、字典(dictionary)、哈希表(hash table)、有键列表(keyed list)或者关联数据(associative array)。一个名/值对的示例如下所示。



```
"name": "Ajax"
```

每个名/值对称为一个元素，元素名和元素值中间用“:”号隔开，前面的是元素名，后面的是元素值。多个名/值对组合在一起就形成了 JSON 的基本结构，如下所示。

```
{  
  "name": "Ajax",  
  "publishDate": "2007-11-28"  
}
```

说明：使用符号“{”和“}”将多个名/值对包含起来形成一个集合，名/值对之间用符号“,”分隔。

JSON 的另外一种形式：值的有序列表，在大部分语言中，都被理解为数组（array）。一个值的有序列表如下所示。

```
["Ajax", "HTML", "JavaScript", "XMLHttpRequest"]
```

### 11.1.2 JSON 中值的类型

JSON 的两种基本结构：名/值对的集合和值的有序列表都离不开值。JSON 中值的类型有 6 种：对象、数组、数字、字符串、布尔值和 null，如图 11.1 所示。

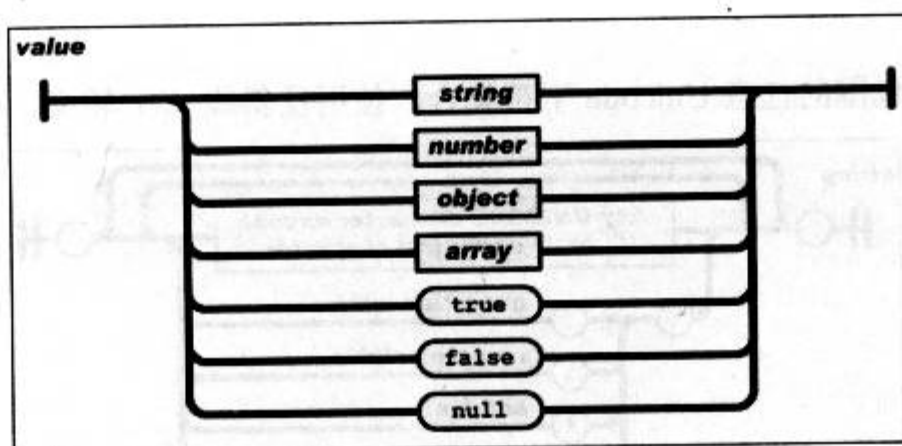


图 11.1 JSON 值

#### 1. 对象

一个对象就是一个无序的名/值对集合。一个对象以“{”开始、以“}”结束。每个名称后面跟一个“:”符号；名/值对之间使用“,”分隔，如图 11.2 所示。

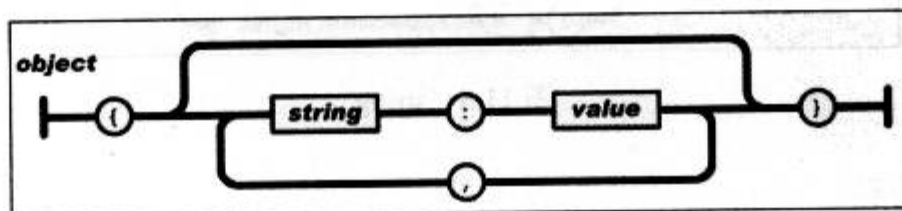


图 11.2 object

#### 2. 数组

数组是值（value）的有序集合。一个数组以“[”符号开始，以“]”符号结束，值之间使用“,”分隔，如图 11.3 所示。



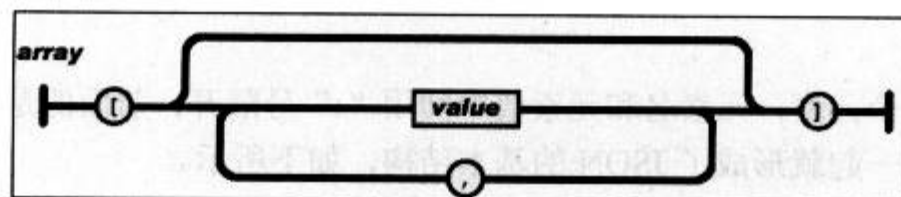


图 11.3 array

### 3. 数字

JSON 中的数字类型与 JavaScript 一样，但是不使用八进制和十六进制格式，如图 11.4 所示。

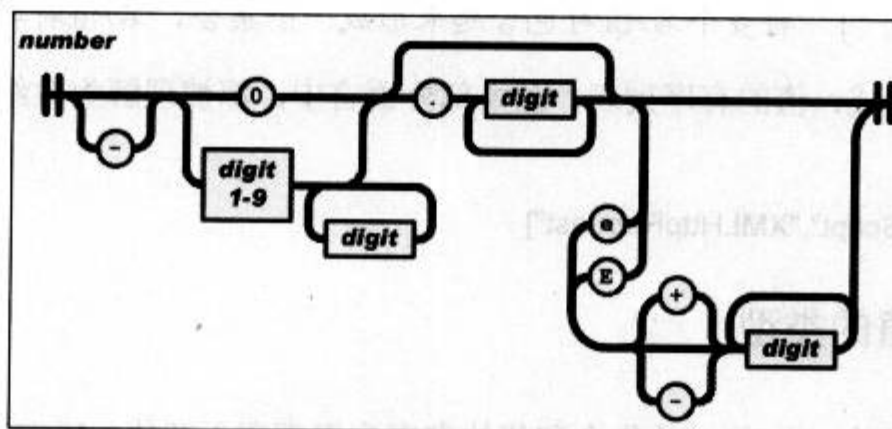


图 11.4 number

### 4. 字符串

字符串是由双引号包围的任意 Unicode 字符集合，使用反斜线 (\) 转义，如图 11.5 所示。

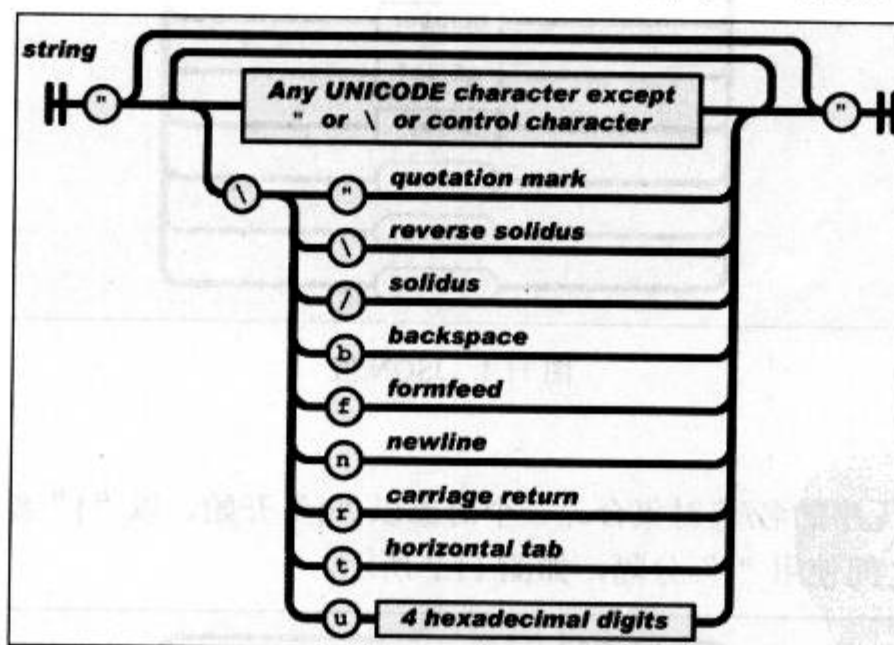


图 11.5 string

### 5. 布尔值

布尔值包含 true（真）和 false（假），代码如下所示。

```

{
  "isRoot":true,
  "hasChildNodes":false
}
  
```

## 6. null

null 表示某个值不存在或没有实际意义, 代码如下所示。

```
{
  "name": "apple",
  "price": "4.22",
  "parent": null
}
```

## 11.2 JSON 的语言支持

JSON 的语言支持非常广泛, 基本上所有主流的开发语言都能够对 JSON 有效地支持。虽然 JSON 诞生于 Web 开发领域, 但是 JSON 的语言支持已经超出了 Web 开发领域的范畴。支持 JSON 的开发语言包括 C、C++、C#、JavaScript、Java、PHP、Perl、Python、ActionScript 等。本节将向读者介绍在十分流行的 4 种开发语言——JavaScript、PHP、C# 和 Java 中使用 JSON 的方法。

### 11.2.1 在 JavaScript 中读取 JSON 数据

JSON 是基于 JavaScript 发展来的, 所以 JavaScript 原生支持 JSON。在 JavaScript 中使用 JSON 非常简单而且高效。

在 JavaScript 中读取 JSON 数据的关键是 eval() 方法的使用。eval() 方法接受一个字符串作为参数, 它会将该字符串作为 JavaScript 语句, 并调用 JavaScript 解析器来运行该语句, 然后将语句运行的结果当作返回值返回。由于 JSON 的基本结构实际上就是 JavaScript 中的对象和数据, 所以将 JSON 格式的字符串作为参数传递给 eval() 方法时, 返回的就是包含了相应数据的对象或者数据, 代码如下所示。

```
var jsonData = eval('(' + jsonString + ');');
```

下面是一个解析 JSON 数据的示例, 代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>read JSON data</title>
<script type="text/javascript">
function readJSON()
{
  var jsonString = document.getElementById('json').value;
  try
  {
    var data = eval('(' + jsonString + ');');
    console.dir(data);
  }
}
```

```
        catch(e)
        {
            console.dir(e);
        }
    }
</script>
</head>

<body>
<textarea cols="100" rows="15" id="json" style="display:block;"></textarea>
<input type="button" value="读取 JSON 数据" onclick="readJSON();" />
</body>
</html>
```

示例程序界面如图 11.6 所示。

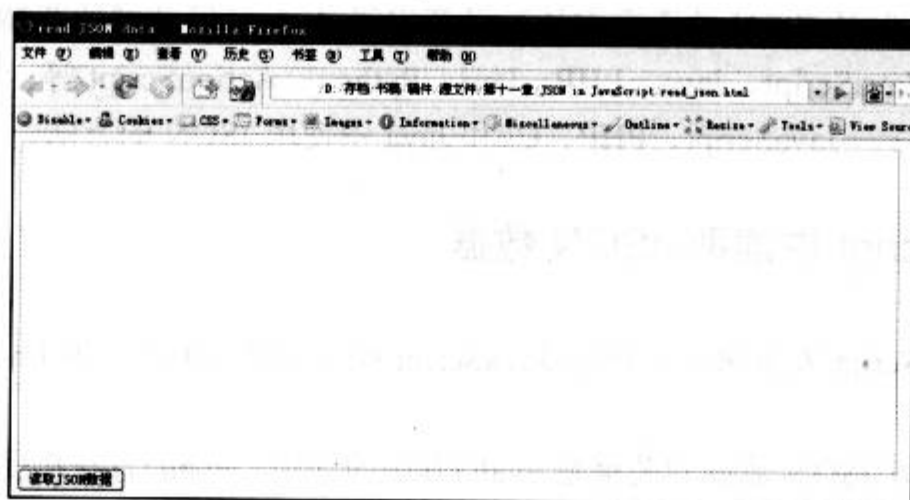


图 11.6 示例程序界面

程序提供了一个输入框来输入 JSON 格式的字符串，然后单击输入框下面的“读取 JSON 数据”按钮来将 JSON 数据读取到 JavaScript 变量中，接着使用 `console.dir` 语句将变量的信息输出到 Firebug 控制台。现在输入一段符合 JSON 格式的字符串，并单击“读取 JSON 数据”按钮，然后打开 Firebug 控制台，如图 11.7 所示。

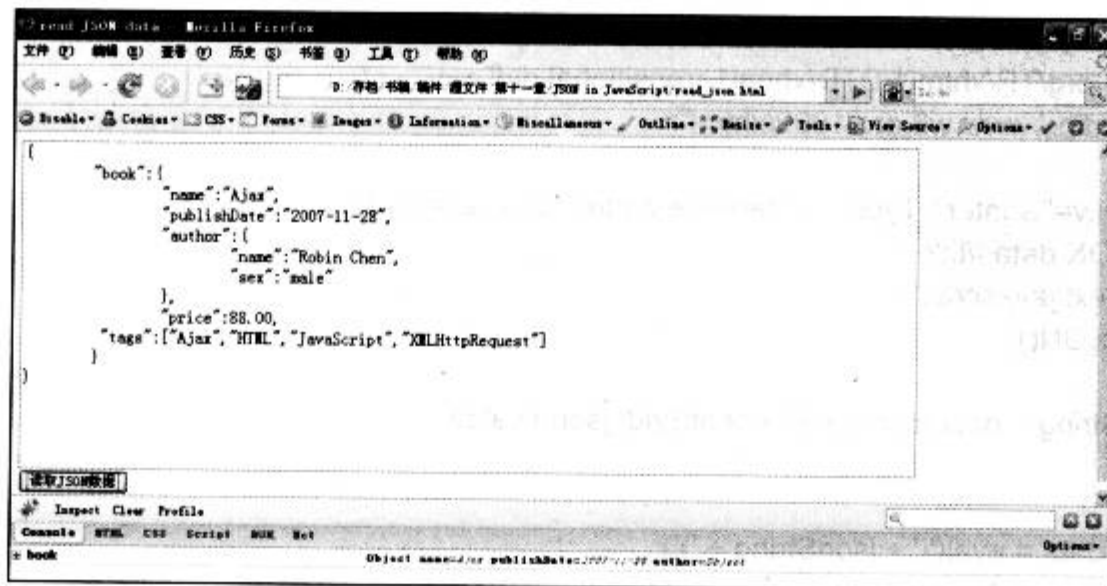


图 11.7 读取 JSON 数据

可以看到 Firebug 控制台中输出了一条名为 `book` 的记录, 其后面的描述信息显示 `book` 的类型是 `object`。展开 `book` 记录以查看其详细数据, 如图 11.8 所示。

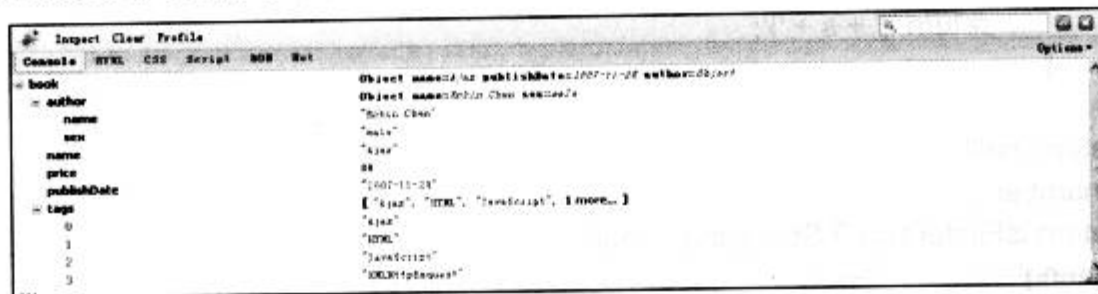


图 11.8 查看 `book` 对象信息

将 JSON 数据使用 `eval()` 方法解析并保存到变量中后, 可以很方便地读取数据, 例如上例中需要读取 `price` (价格) 信息, 直接使用 `book.price` 即可得到; 需要读取 `author` 的 `name` 属性, 使用 `book.author.name` 即可; 读取第一个 `tags` 的数据, 使用 `book.tags[0]` 即可。

### 11.2.2 在 JavaScript 中输出 JSON 数据

JavaScript 中的对象 (对象和数组) 都可以很方便地转化成 JSON 格式的字符串, 以提供给程序进一步处理, 例如传输给后端处理程序等。这里封装了一段 JavaScript 脚本来完成这个功能, 代码如下所示。

```
Array.prototype.__array = '__array';
var JSON = {
  stringify: function (arg) {
    var c, i, l, s = "", v;
    switch (typeof arg) {
      case 'object':
        if (arg) {
          if (arg.__array == '__array') {
            for (i = 0; i < arg.length; ++i) {
              v = this.stringify(arg[i]);
              if (s) {
                s += ',';
              }
              s += v;
            }
            return '[' + s + ']';
          } else if (arg instanceof Date) {
            return arg.toLocaleString();
          } else if (typeof arg.toString != 'undefined') {
            for (i in arg) {
              v = arg[i];
              if (typeof v != 'undefined' && typeof v != 'function') {
                v = this.stringify(v);
                if (s) {
                  s += ',';
                }
              }
            }
          }
        }
      default:
        s = arg;
    }
    return s;
  }
}
```

```

        s += this.stringify(i) + ':' + v;
    }
    }
    return '{' + s + '}';
}
}
return 'null';
case 'number':
    return isFinite(arg) ? String(arg) : 'null';
case 'string':
    l = arg.length;
    s = '';
    for (i = 0; i < l; i += 1) {
        c = arg.charAt(i);
        if (c >= ' ') {
            if (c == '\\' || c == '"') {
                s += '\\';
            }
            s += c;
        } else {
            switch (c) {
                case '\b':
                    s += '\\b';
                    break;
                case '\f':
                    s += '\\f';
                    break;
                case '\n':
                    s += '\\n';
                    break;
                case '\r':
                    s += '\\r';
                    break;
                case '\t':
                    s += '\\t';
                    break;
                default:
                    c = c.charCodeAt();
                    s += '\\u00' + Math.floor(c / 16).toString(16) +
                        (c % 16).toString(16);
            }
        }
    }
    return s + '';
case 'boolean':
    return String(arg);
default:
    return 'null';
}
},

```



```

parse: function (text) {
    var at = 0;
    var ch = '';

    function error(m) {
        throw {
            name: 'JSONError',
            message: m,
            at: at - 1,
            text: text
        };
    }

    function next() {
        ch = text.charAt(at);
        at += 1;
        return ch;
    }

    function white() {
        while (ch != " " && ch <= ' ') {
            next();
        }
    }

    function str() {
        var i, s = "", t, u;
        if (ch == '"') {
            while (next()) {
                if (ch == '"') {
                    next();
                    return s;
                }
                else if (ch == '\\') {
                    switch (next()) {
                        case 'b':
                            s += '\b';
                            break;
                        case 'f':
                            s += '\f';
                            break;
                        case 'n':
                            s += '\n';
                            break;
                        case 'r':
                            s += '\r';
                            break;
                        case 't':
                            s += '\t';
                            break;
                        case 'u':

```

```

        u = 0;
        for (i = 0; i < 4; i += 1) {
            t = parseInt(next(), 16);
            if (!isFinite(t)) {
                break outer;
            }
            u = u * 16 + t;
        }
        s += String.fromCharCode(u);
        break;
    default:
        s += ch;
    }
} else {
    s += ch;
}
}
}
error("Bad string");
}
function arr() {
    var a = [];
    if (ch == '[') {
        next();
        white();
        if (ch == ']') {
            next();
            return a;
        }
        while (ch) {
            a.push(val());
            white();
            if (ch == ']') {
                next();
                return a;
            } else if (ch != ',') {
                break;
            }
        }
        next();
        white();
    }
}
error("Bad array");
}
function obj() {
    var k, o = {};

    if (ch == '{') {
        next();
        white();
        if (ch == '}') {

```

```

        next();
        return o;
    }
    while (ch) {
        k = str();
        white();
        if (ch != ':') {
            break;
        }
        next();
        o[k] = val();
        white();
        if (ch == '}') {
            next();
            return o;
        } else if (ch != ',') {
            break;
        }
        next();
        white();
    }
    error("Bad object");
}

function num() {
    var n = "", v;
    if (ch == '-') {
        n = '-';
        next();
    }
    while (ch >= '0' && ch <= '9') {
        n += ch;
        next();
    }
    if (ch == '.') {
        n += '.';
        while (next() && ch >= '0' && ch <= '9') {
            n += ch;
        }
    }
    if (ch == 'e' || ch == 'E') {
        n += 'e';
        next();
        if (ch == '-' || ch == '+') {
            n += ch;
            next();
        }
        while (ch >= '0' && ch <= '9') {
            n += ch;
            next();
        }
    }
}

```

```

    }
    v = +n;
    if (!isFinite(v)) {
        error("Bad number");
    } else {
        return v;
    }
}
function word() {
    switch (ch) {
        case 'r':
            if (next() == 'r' && next() == 'u' && next() == 'e') {
                next();
                return true;
            }
            break;
        case 'f':
            if (next() == 'a' && next() == 'l' && next() == 's' &&
                next() == 'e') {
                next();
                return false;
            }
            break;
        case 'n':
            if (next() == 'u' && next() == 'l' && next() == 'l') {
                next();
                return null;
            }
            break;
    }
    error("Syntax error");
}
function val() {
    white();
    switch (ch) {
        case '[':
            return obj();
        case '[':
            return arr();
        case '"':
            return str();
        case '-':
            return num();
        default:
            return ch >= '0' && ch <= '9' ? num() : word();
    }
}
return val();
};

```

脚本中定义了名为 JSON 的对象,并实现了 stringify 方法来完成从 JavaScript 对象到 JSON 格式字符串的转换。该方法的原理就是遍历 JavaScript 对象的所有属性并根据其数据类型来输出不同格式的数据。现在来测试该方法:首先将上面的脚本保存为一个 js 文件,并命名为 JSON.js,然后编写测试代码。测试代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>output JSON data</title>
<script type="text/javascript" src="JSON.js"></script><!--引入 JSON.js-->
<script type="text/javascript">
/*
 * 创建一个名为 book 的对象,其拥有 name、publishDate、author、price 和 tags 属性
 */
var book = new Object();
book.name = 'AJAX';
book.publishDate = new Date(2007,11,28);
/*
 * author 属性本身也是对象, 包含 name 和 sex 属性
 */
book.author = new Object();
book.author.name = 'Robin Chen';
book.author.sex = 'male';
book.price = 88;
/*
 * tags 属性是一个数组
 */
book.tags = ["Ajax","HTML","JavaScript","XMLHttpRequest"];
/*
 * 调用 JSON.stringify()方法将 book 对象转换成对应的 JSON 数组, 并使用对话框输出
 */
var jsonString = JSON.stringify(book);
alert(jsonString);
</script>
</head>

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

运行测试代码,其输出结果如图 11.9 所示。

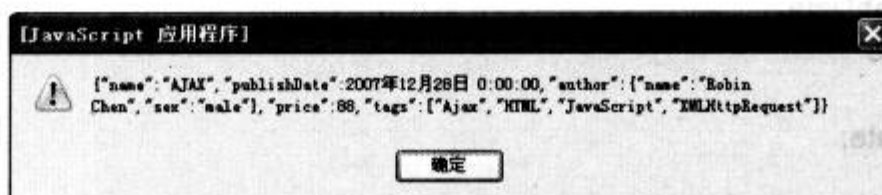


图 11.9 输出 JSON



### 11.2.3 在 PHP 中使用 JSON

PHP 从 5.2 版本以来内建了对 JSON 的支持,通过 `json_encode()`和 `json_decode()`这两个方法来实现。为了向读者演示这两个方法的使用,先来编写一段 PHP 程序,代码如下所示。

```
<?
/*
 * 类名: Author
 */
class Author
{
    /*
     * 属性名: name
     * 类型: string
     */
    public $name;
    /*
     * 属性名: sex
     * 类型: string
     */
    public $sex;
    /*
     * 构造函数
     */
    public function Author($name,$sex)
    {
        $this->name = $name;
        $this->sex = $sex;
    }
}
/*
 * 类名: Book
 */
class Book
{
    /*
     * 属性名: name
     * 类型: string
     */
    public $name;
    /*
     * 属性名: publishDate
     * 类型: datetime
     */
    public $publishDate;
    /*
     * 属性名: author
```

```

    * 类型: Author
    */
    public $author;
    /*
    * 属性名: price
    * 类型: float
    */
    public $price;
    /*
    * 属性名: tags
    * 类型: array
    */
    public $tags;
    /*
    * 构造函数
    */
    public function Book($name,$publishDate,$author,$price,$tags)
    {
        $this->name = $name;
        $this->publishDate = $publishDate;
        $this->author = $author;
        $this->price = $price;
        $this->tags = $tags;
    }
}
/*
* 创建 Book 实例并测试 json_encode()方法
*/
$author = new Author('Robin Chen','male');           //创建 Author 实例
$tags = array("Ajax","HTML","JavaScript","XMLHttpRequest"); //创建 tags
$book = new Book('Ajax',date('Y-m-d'),$author,88,$tags); //创建 Book 实例
$json = json_encode($book);                          //返回表示 book 数据的 JSON 字符串
echo $json;                                           //输出 JSON
?>

```

PHP 程序中定义了两个类: Author 和 Book, 实例化 Book 后使用 json\_encode 方法获取与该实例对应的 JSON 字符串, 并输出。输出的内容如图 11.10 所示。

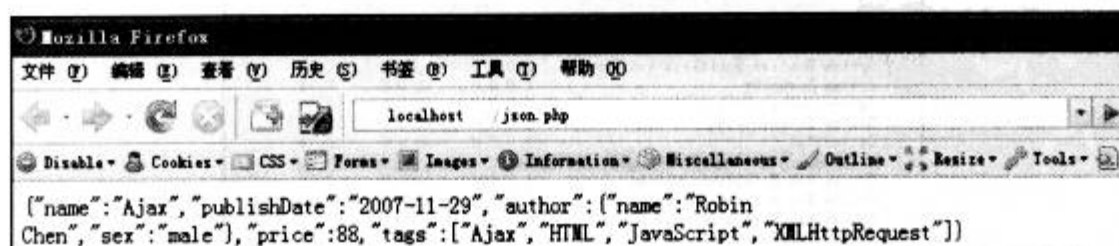


图 11.10 json\_encode()

json\_decode()与 json\_encode()相反, 它可以将 JSON 数据转换为 PHP 对象, 代码如下所示。

```

<?
/*
* 创建 JSON 数据

```

```

*/
$json = '{"name":"Ajax",'.
        '"publishDate":"2007-11-29",'.
        '"author":{"name":"Robin Chen","sex":"male"},'.
        '"price":88,'.
        '"tags":["Ajax","HTML","JavaScript","XMLHttpRequest"]}';
/*
 * 使用 json_decode()方法将 JSON 数据转化为对应的 PHP 数据类型, 这里为 PHP 对象
 */
$book = json_decode($json);
/*
 * 输出$book 对象的详细信息
 */
var_dump($book);
?>

```

其输出的结果如图 11.11 所示。

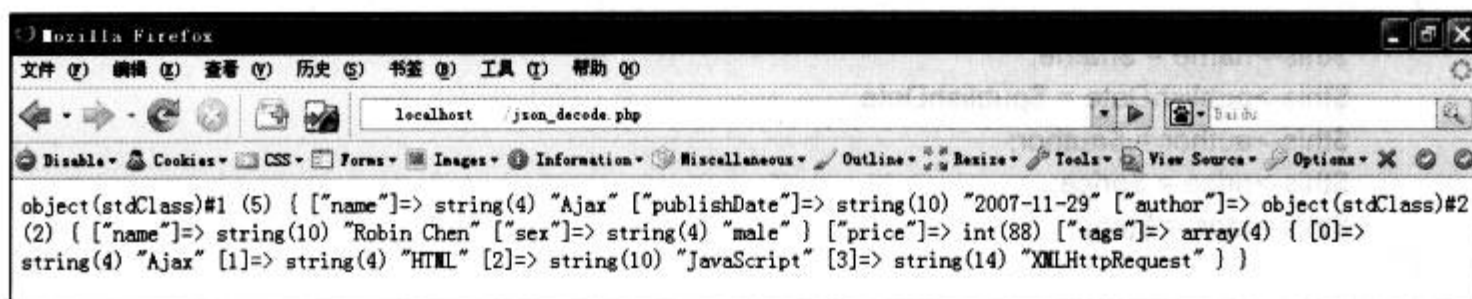


图 11.11 json\_decode()

#### 11.2.4 在 C#中输出 JSON 数据

Microsoft 在 2007 年初发布的 ASP.NET AJAX 1.0 中, 提供了对 JSON 的支持。ASP.NET AJAX 1.0 是 ASP.NET 2.0 的一个扩展, 其可以在 <http://ajax.asp.net> 免费下载和安装。ASP.NET AJAX 1.0 对 JSON 的支持是通过 System.Web.Script.Serialization 命名空间下的 JavaScriptSerializer 类实现的。安装了 ASP.NET AJAX 1.0 后, 需要在工程中添加对 System.Web.Extensions 的引用, 如图 11.12 所示。

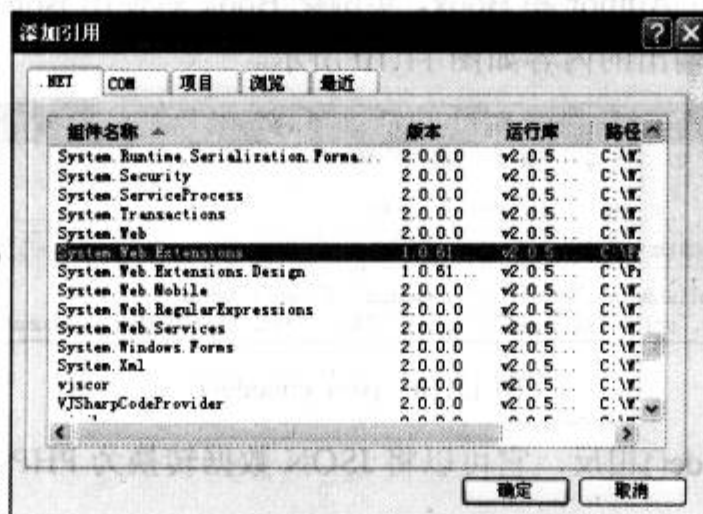


图 11.12 引入 System.Web.Extensions

在引入 System.Web.Extensions 后, 就可以使用 System.Web.Script.Serialization.JavaScriptSerializer 类了。下面通过一个示例来说明如何使用这个类以完成 JSON 数据的输出。示例的 Default.aspx 文件代码如下所示。

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!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 runat="server">
    <title>JSON demo</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label ID="lblJson" runat="server" Text=""></asp:Label>
        </div>
    </form>
</body>
</html>
```

Default.aspx.cs 文件代码如下所示。

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Web.Script.Serialization;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Author author = new Author("Robin Chen", Sex.male);           //创建作者实例
        Book book = new Book("Ajax", DateTime.Now, author, new string[] { "Ajax", "HTML", "JavaScript", "XML",
"JSON" });                  //创建图书实例
        JavaScriptSerializer serializer = new JavaScriptSerializer();    //创建 JavaScriptSerializer 实例
        string json = serializer.Serialize(book);    //使用 Serialize 方法将 book 实例转化成 JSON 格式的字符串
        lblJson.Text = json;    //将 JSON 字符串显示到页面上的 LABEL 中
    }
}

/// <summary>
/// 性别
```

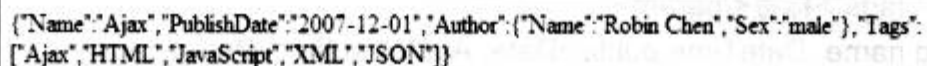
```
/// </summary>
public enum Sex
{
    /// <summary>
    /// 男
    /// </summary>
    male,
    /// <summary>
    /// 女
    /// </summary>
    female
}
/// <summary>
/// 作者
/// </summary>
public class Author
{
    /// <summary>
    /// 名字
    /// </summary>
    private string _name;
    /// <summary>
    /// 性别
    /// </summary>
    private Sex _sex;
    /// <summary>
    /// 构造函数
    /// </summary>
    /// <param name="name">名字</param>
    /// <param name="sex">性别</param>
    public Author(string name, Sex sex)
    {
        _name = name;
        _sex = sex;
    }
    /// <summary>
    /// 构造函数
    /// </summary>
    public Author() {}
    /// <summary>
    /// 名字
    /// </summary>
    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }
    /// <summary>
    /// 性别
```



```
/// </summary>
public string Sex
{
    get { return Enum.GetName(_sex.GetType(), _sex); }
    set { Enum.Parse(_sex.GetType(), value); }
}
}
/// <summary>
/// 图书
/// </summary>
public class Book
{
    /// <summary>
    /// 书名
    /// </summary>
    private string _name;
    /// <summary>
    /// 出版日期
    /// </summary>
    private DateTime _publishDate;
    /// <summary>
    /// 作者
    /// </summary>
    private Author _author;
    /// <summary>
    /// 标签
    /// </summary>
    private string[] _tags;
    /// <summary>
    /// 构造函数
    /// </summary>
    /// <param name="name">书名</param>
    /// <param name="publishDate">出版日期</param>
    /// <param name="author">作者</param>
    /// <param name="tags">标签</param>
    public Book(string name, DateTime publishDate, Author author, string[] tags)
    {
        _name = name;
        _publishDate = publishDate;
        _author = author;
        _tags = tags;
    }
    /// <summary>
    /// 构造函数
    /// </summary>
    public Book() { }
    /// <summary>
    /// 书名
    /// </summary>
```

```
public string Name
{
    get { return _name; }
    set { _name = value; }
}
/// <summary>
/// 出版日期
/// </summary>
public string PublishDate
{
    get { return _publishDate.ToShortDateString(); }
    set { _publishDate = DateTime.Parse(value); }
}
/// <summary>
/// 作者
/// </summary>
public Author Author
{
    get { return _author; }
    set { _author = value; }
}
/// <summary>
/// 标签
/// </summary>
public string[] Tags
{
    get { return _tags; }
    set { _tags = value; }
}
```

示例程序中声明了 Author 和 Book 两个类,并在 Page\_Load 事件处理程序中取得了 Book 的一个实例 book,然后实例化了 JavaScriptSerializer,并调用 Serialize 方法将 book 转化为对应的 JSON 数据。最后将 JSON 数据写入到界面上的 Label 控件中,效果如图 11.13 所示。



```
{\"Name\": \"Ajax\", \"PublishDate\": \"2007-12-01\", \"Author\": {\"Name\": \"Robin Chen\", \"Sex\": \"male\"}, \"Tags\": [\"Ajax\", \"HTML\", \"JavaScript\", \"XML\", \"JSON\"]}
```

图 11.13 JSON 数据

### 11.2.5 在 C# 中输出带类型说明的 JSON 数据

JavaScriptSerializer 类的构造函数还有一个带有一个参数的重载,可以为其指定转换的规则,其语法如下所示。

```
new JavaScriptSerializer(JavascriptTypeResolver resolver);
```

参数的类型为 JavaScriptTypeResolver, JavaScriptTypeResolver 提供了一个实现自定义类型解析器的抽象类。首先需要为自定义类型创建一个类型解析器,从 JavaScriptTypeResolver 继承,代码如下

所示。

```
public class CustomTypeResolver : JavaScriptTypeResolver
{
    public override Type ResolveType(string id)
    {
        return Type.GetType(id);
    }
    public override string ResolveTypeId(Type type)
    {
        if (type == null)
        {
            throw new ArgumentNullException("type");
        }
        return type.Name;
    }
}
```

然后修改实例化 JavaScriptSerializer 的方法, 代码如下所示。

```
JavaScriptSerializer serializer = new JavaScriptSerializer(new CustomTypeResolver());
```

完整代码如下所示。

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Web.Script.Serialization;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Author author = new Author("Robin Chen", Sex.male); //创建作者实例
        Book book = new Book("Ajax", DateTime.Now, author, new string[] { "Ajax", "HTML", "JavaScript", "XML",
"JSON" }); //创建图书实例
        JavaScriptSerializer serializer = new JavaScriptSerializer(new CustomTypeResolver()); //创建 JavaScriptSerializer 实例
        string json = serializer.Serialize(book); //使用 Serialize 方法将 book 实例转化成 JSON 格式的字符串
        lblJson.Text = json; //将 JSON 字符串显示到页面上的 LABEL 中
    }
}
/// <summary>
/// 性别
/// </summary>
```

```
public enum Sex
{
    /// <summary>
    /// 男
    /// </summary>
    male,
    /// <summary>
    /// 女
    /// </summary>
    female
}
/// <summary>
/// 作者
/// </summary>
public class Author
{
    /// <summary>
    /// 名字
    /// </summary>
    private string _name;
    /// <summary>
    /// 性别
    /// </summary>
    private Sex _sex;
    /// <summary>
    /// 构造函数
    /// </summary>
    /// <param name="name">名字</param>
    /// <param name="sex">性别</param>
    public Author(string name, Sex sex)
    {
        _name = name;
        _sex = sex;
    }
    /// <summary>
    /// 构造函数
    /// </summary>
    public Author() { }
    /// <summary>
    /// 名字
    /// </summary>
    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }
    /// <summary>
    /// 性别
    /// </summary>
```

```

public string Sex
{
    get { return Enum.GetName(_sex.GetType(), _sex); }
    set { Enum.Parse(_sex.GetType(), value); }
}
}
/// <summary>
/// 图书
/// </summary>
public class Book
{
    /// <summary>
    /// 书名
    /// </summary>
    private string _name;
    /// <summary>
    /// 出版日期
    /// </summary>
    private DateTime _publishDate;
    /// <summary>
    /// 作者
    /// </summary>
    private Author _author;
    /// <summary>
    /// 标签
    /// </summary>
    private string[] _tags;
    /// <summary>
    /// 构造函数
    /// </summary>
    /// <param name="name">书名</param>
    /// <param name="publishDate">出版日期</param>
    /// <param name="author">作者</param>
    /// <param name="tags">标签</param>
    public Book(string name, DateTime publishDate, Author author, string[] tags)
    {
        _name = name;
        _publishDate = publishDate;
        _author = author;
        _tags = tags;
    }
    /// <summary>
    /// 构造函数
    /// </summary>
    public Book() { }
    /// <summary>
    /// 书名
    /// </summary>
    public string Name

```



```
{
    get { return _name; }
    set { _name = value; }
}
/// <summary>
/// 出版日期
/// </summary>
public string PublishDate
{
    get { return _publishDate.ToShortDateString(); }
    set { _publishDate = DateTime.Parse(value); }
}
/// <summary>
/// 作者
/// </summary>
public Author Author
{
    get { return _author; }
    set { _author = value; }
}
/// <summary>
/// 标签
/// </summary>
public string[] Tags
{
    get { return _tags; }
    set { _tags = value; }
}
}

public class CustomTypeResolver : JavaScriptTypeResolver
{
    public override Type ResolveType(string id)
    {
        return Type.GetType(id);
    }

    public override string ResolveTypeId(Type type)
    {
        if (type == null)
        {
            throw new ArgumentNullException("type");
        }

        return type.Name;
    }
}
```

运行结果如图 11.14 所示。

```
{ "__type": "Book", "Name": "Ajax", "PublishDate": "2007-12-01", "Author": { "__type": "Author", "Name": "Robin Chen", "Sex": "male" }, "Tags": [ "Ajax", "HTML", "JavaScript", "XML", "JSON" ] }
```

图 11.14 带类型的 JSON 数据

可以看到, 在输出的字符串中, 每个对象内部都增加了一个名为 “\_\_type” 的属性, 其值为该 C# 对象的类型。

### 11.2.6 在 C# 中读取 JSON 数据

JavaScriptSerializer 类提供了 Deserialize 方法, 可以将 JSON 格式的数据转换为 C# 对象, 其语法如下所示。

```
JavaScriptSerializer.Deserialize<Type>(string json)
```

其中 Type 为将要转化为的 C# 对象的类型, json 为 JSON 格式的字符串, 该方法的示例如下所示。

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Web.Script.Serialization;

public partial class Default2 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        //实例化 JavaScriptSerializer
        JavaScriptSerializer serializer = new JavaScriptSerializer(new CustomTypeResolver());
        //JSON 数据
        string json = "{ \"__type\": \"Book\", \"Name\": \"Ajax\", \"PublishDate\": \"2007-12-02\", \"Author\": { \"__type\": \"Author\", \"Name\": \"Robin Chen\", \"Sex\": \"male\" }, \"Tags\": [ \"Ajax\", \"HTML\", \"JavaScript\", \"XML\", \"JSON\" ] }";
        //将 JSON 数据转换为 Book 类型的对象
        Book book = serializer.Deserialize<Book>(json);
        /*
         * 输出 book 相关数据
         */
        lblJson.Text = "<br/>Book Name:" + book.Name;
        lblJson.Text += "<br/>Publish Date:" + book.PublishDate;
        lblJson.Text += "<br/>Author Name:" + book.Author.Name;
        lblJson.Text += "<br/>Author Sex:" + book.Author.Sex;
```

```

        StringArrayConverter converter = new StringArrayConverter();
        lblJson.Text += "<br/>Tags:" + converter.ConvertToString(book.Tags);
    }
}

/// <summary>
/// 性别
/// </summary>
public enum Sex
{
    /// <summary>
    /// 男
    /// </summary>
    male,
    /// <summary>
    /// 女
    /// </summary>
    female
}

/// <summary>
/// 作者
/// </summary>
public class Author
{
    /// <summary>
    /// 名字
    /// </summary>
    private string _name;
    /// <summary>
    /// 性别
    /// </summary>
    private Sex _sex;
    /// <summary>
    /// 构造函数
    /// </summary>
    /// <param name="name">名字</param>
    /// <param name="sex">性别</param>
    public Author(string name, Sex sex)
    {
        _name = name;
        _sex = sex;
    }
    /// <summary>
    /// 构造函数
    /// </summary>
    public Author() {}
    /// <summary>
    /// 名字

```

```

/// </summary>
public string Name
{
    get { return _name; }
    set { _name = value; }
}
/// <summary>
/// 性别
/// </summary>
public string Sex
{
    get { return Enum.GetName(_sex.GetType(), _sex); }
    set { Enum.Parse(_sex.GetType(), value); }
}
}
/// <summary>
/// 图书
/// </summary>
public class Book
{
    /// <summary>
    /// 书名
    /// </summary>
    private string _name;
    /// <summary>
    /// 出版日期
    /// </summary>
    private DateTime _publishDate;
    /// <summary>
    /// 作者
    /// </summary>
    private Author _author;
    /// <summary>
    /// 标签
    /// </summary>
    private string[] _tags;
    /// <summary>
    /// 构造函数
    /// </summary>
    /// <param name="name">书名</param>
    /// <param name="publishDate">出版日期</param>
    /// <param name="author">作者</param>
    /// <param name="tags">标签</param>
    public Book(string name, DateTime publishDate, Author author, string[] tags)
    {
        _name = name;
        _publishDate = publishDate;
        _author = author;
        _tags = tags;
    }
}

```

```

    }
    /// <summary>
    /// 构造函数
    /// </summary>
    public Book() { }
    /// <summary>
    /// 书名
    /// </summary>
    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }
    /// <summary>
    /// 出版日期
    /// </summary>
    public string PublishDate
    {
        get { return _publishDate.ToShortDateString(); }
        set { _publishDate = DateTime.Parse(value); }
    }
    /// <summary>
    /// 作者
    /// </summary>
    public Author Author
    {
        get { return _author; }
        set { _author = value; }
    }
    /// <summary>
    /// 标签
    /// </summary>
    public string[] Tags
    {
        get { return _tags; }
        set { _tags = value; }
    }
}

public class CustomTypeResolver : JavaScriptTypeResolver
{
    public override Type ResolveType(string id)
    {
        return Type.GetType(id);
    }

    public override string ResolveTypeId(Type type)
    {
        if (type == null)

```



```

{
    throw new ArgumentNullException("type");
}

return type.Name;
}
}

```

在上述程序代码中将一段 JSON 格式的数据转换为 Book 类型的对象,然后将该对象的相关数据输出到界面上的 Label 控件中,结果如图 11.15 所示。

```

Book Name:Ajax
Publish Date:2007-12-02
Author Name:Robin Chen
Author Sex:male
Tags:Ajax,HTML,JavaScript,XML,JSON

```

图 11.15 输出结果

### 11.2.7 更多语言支持

如果需查看关于 JSON 更多的语言支持信息,可登录 JSON 的官方网站 <http://www.json.org>。在该网站首页的下部提供了一个列表,详细地列举了所有支持 JSON 的语言以及相关资源,如图 11.16 所示。

<ul style="list-style-type: none"> <li>• ASP.</li> <li>• ActionScript: <ul style="list-style-type: none"> <li>◦ ActionScript1.</li> <li>◦ ActionScript2.</li> <li>◦ ActionScript3.</li> <li>◦ JSONConnector.</li> </ul> </li> <li>• C.</li> <li>◦ JSON_checker.</li> <li>◦ json-c.</li> <li>◦ M's JSON parser.</li> <li>◦ YAJL.</li> <li>• C++:</li> <li>◦ jsoncpp.</li> <li>◦ xoolib.</li> <li>◦ Jaula.</li> <li>◦ JOST.</li> <li>◦ JSON Spirit.</li> <li>• C#:</li> <li>◦ JSON_checker.</li> <li>◦ Jayrock.</li> <li>◦ Newtonsoft Json.NET.</li> <li>◦ JSONSharp.</li> <li>◦ LitJSON.</li> <li>◦ JSON for .NET.</li> <li>◦ JsonFx.</li> <li>• ColdFusion:</li> <li>◦ ColdFusion 8.</li> <li>◦ CFJSON.</li> <li>◦ toJSON.</li> <li>• D.</li> <li>• Delphi:</li> <li>◦ Delphi Web Utils.</li> <li>◦ JSON Delphi Library.</li> <li>◦ JSON Toolkit.</li> </ul>	<ul style="list-style-type: none"> <li>• E.</li> <li>• Erlang.</li> <li>• Haskell:</li> <li>◦ JSON.hs.</li> <li>◦ HaskellNet.</li> <li>• haXe.</li> <li>• Java:</li> <li>◦ org.json.</li> <li>◦ org.json.ms.</li> <li>◦ json-lib.</li> <li>◦ JSON Tools.</li> <li>◦ org.json.simple.</li> <li>◦ Stringtree.</li> <li>◦ SOJO.</li> <li>◦ VRaptor.</li> <li>◦ Restlet.</li> <li>◦ Jettison.</li> <li>◦ json-taglib.</li> <li>◦ FLEXJSON.</li> <li>◦ XStream.</li> <li>◦ JsonMarshaller.</li> <li>◦ Flexjson.</li> <li>◦ Jackson JSON Processor.</li> <li>• JavaScript.</li> <li>• Lasso.</li> <li>• Lisp:</li> <li>◦ Common Lisp.</li> <li>◦ Emacs Lisp.</li> <li>◦ Chicken Scheme.</li> <li>◦ NZScheme.</li> <li>• LotusScript.</li> <li>• Lua:</li> <li>◦ Json4Lua.</li> <li>◦ json.lua.</li> </ul>	<ul style="list-style-type: none"> <li>• Objective C:</li> <li>◦ BSJSONAdditions.</li> <li>◦ Cocoa JSON Framework.</li> <li>• Objective CAML.</li> <li>• OpenLaszlo.</li> <li>• Perl.</li> <li>• PHP:</li> <li>◦ PHP 5.2.</li> <li>◦ json.</li> <li>◦ Services_JSON.</li> <li>◦ Zend_JSON.</li> <li>◦ JSONRPC.</li> <li>◦ Solar_json.</li> <li>◦ SCA_SDO.</li> <li>◦ Comparison of php json libraries.</li> <li>• Pike:</li> <li>◦ Public.Parser.JSON.</li> <li>◦ Public.Parser.JSON2.</li> <li>• pl/sql.</li> <li>• PowerShell.</li> <li>• Prolog.</li> <li>• Python:</li> <li>◦ python-cjson.</li> <li>◦ simplejson.</li> <li>◦ python-json.</li> <li>◦ demjson.</li> <li>◦ pyparsing.</li> <li>◦ Choosing a Python JSON Translator.</li> <li>• R.</li> <li>• REALbasic.</li> <li>• Rebol.</li> <li>• Ruby.</li> <li>• Squeak.</li> </ul>
--	---	---

图 11.16 JSON 的语言支持

## 11.3 JSON 的优点和不足

JSON 是在 Ajax 应用程序中作为 XML 的代替品而被提出的。JSON 本身具有一些令人兴奋的优点,但是世界上没有完美的东西,同样它也存在一些不足之处。本节将向读者介绍 JSON 的优点和不足。

### 11.3.1 JSON 的优点

JSON 的第一个优点就是它的非冗长性。下面分别使用 JSON 和 XML 来表达同一条数据。

XML:

```
<books>
  <book>
    <name>Ajax</name>
    <publishDate>2007-12-03</publishDate>
    <author>
      <name>Robin Chen</name>
      <sex>male</sex>
    </author>
    <price>88.00</price>
    <tags>
      <tag>Ajax</tag>
      <tag>JavaScript</tag>
      <tag>XMLHttpRequest</tag>
      <tag>HTML</tag>
      <tag>DOM</tag>
      <tag>XML</tag>
      <tag>JSON</tag>
    </tags>
  </book>
  <book>
    <name>DHTML</name>
    <publishDate>2007-12-03</publishDate>
    <author>
      <name>Robin Chen</name>
      <sex>male</sex>
    </author>
    <price>44.00</price>
    <tags>
      <tag>Ajax</tag>
      <tag>JavaScript</tag>
      <tag>XMLHttpRequest</tag>
      <tag>HTML</tag>
      <tag>DOM</tag>
      <tag>XML</tag>
      <tag>JSON</tag>
    </tags>
  </book>
</books>
```

JSON:

```
{
  books:[
    {"name":"Ajax",
      "publishDate":"2007-12-03",
```

```
"author":{
  "name":"Robin Chen",
  "sex":"male"
},
"price":88.00,
tags:["Ajax","JavaScript","XMLHttpRequest","DOM","XML","JSON"],
{"name":"DHTML",
"publishDate":"2007-12-03",
"author":{
  "name":"Robin Chen",
  "sex":"male"
},
"price":44.00,
tags:["Ajax","JavaScript","XMLHttpRequest","DOM","XML","JSON"]}
]
```

可见,在表示同样的数据时,JSON 比 XML 更简洁,所使用的字节数更少。因为 XML 的语法规则规定了标签必须闭合,实际上闭合的标签本身对数据的描述是没有任何意义的。这使得在进行大数据量的传输时,JSON 具有明显的性能优势。

JSON 的第二个优点就是解析方便。JSON 使用的分隔符与 JavaScript 引擎对数据结构的内部表示一致,这样就可以方便地将 JSON 格式的数据转化为对象,且轻松地访问数据,而不需要像 XML 那样通过复杂的 DOM 来进行操作,并且规避了浏览器对 DOM 的实现差异问题,简化了开发并提高了执行效率。

JSON 的第三个优点就是因为其结构简单,使得开发者无论在服务端使用什么语言,都可以很容易地生成 JSON。

### 11.3.2 JSON 的不足

JSON 没有像 XML 那样的命名空间机制,这样在不同上下文的相同信息段混合在一起时,可能会给解析带来一定的麻烦。同时 JSON 的创建和验证过程跟 XML 相比会稍微麻烦一些,毕竟 XML 已经发展了近 10 年,相关的工具支持更加广泛和成熟。

## 11.4 将 XML 转换为 JSON

在一些应用中,可能会需要将已有的 XML 数据转换为 JSON 格式的数据输出,以提供新的功能。例如某个传统的 WebService 使用 XML 作为数据载体,但将其应用到 Ajax 风格的应用程序中时,希望其输出的数据为 JSON 格式以便于客户端的处理,这时就可以将 XML 数据转换为 JSON 数据。本节将对具体的转换方式进行一些介绍。此时,读者可能首先会想到编写一个解析器,将 XML 数据转换为对象然后解析生成 JSON 格式的数据。这样做没有任何问题,但是其实现依赖于编写该解析器的语言环境,不具有通用性。其实真正有效的方法是使用 XSL 对 XML 进行格式化,并输出 JSON 风格的文本。



```

    <xsl:otherwise>
      <xsl:call-template name="escape-quot-string">
        <xsl:with-param name="s" select="$s"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
<xsl:template name="escape-quot-string">
  <xsl:param name="s"/>
  <xsl:choose>
    <xsl:when test="contains($s,'&quot;')">
      <xsl:call-template name="encode-string">
        <xsl:with-param name="s" select="concat(substring-before($s,'&quot;'),'&quot;')"/>
      </xsl:call-template>
      <xsl:call-template name="escape-quot-string">
        <xsl:with-param name="s" select="substring-after($s,'&quot;')"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="encode-string">
        <xsl:with-param name="s" select="$s"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
<xsl:template name="encode-string">
  <xsl:param name="s"/>
  <xsl:choose>
    <xsl:when test="contains($s,'&#x9;')">
      <xsl:call-template name="encode-string">
        <xsl:with-param name="s"
select="concat(substring-before($s,'&#x9;'),'t',substring-after($s,'&#x9;'))"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:when test="contains($s,'&#xA;')">
      <xsl:call-template name="encode-string">
        <xsl:with-param name="s"
select="concat(substring-before($s,'&#xA;'),'n',substring-after($s,'&#xA;'))"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:when test="contains($s,'&#xD;')">
      <xsl:call-template name="encode-string">
        <xsl:with-param name="s"
select="concat(substring-before($s,'&#xD;'),'r',substring-after($s,'&#xD;'))"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise><xsl:value-of select="$s"/></xsl:otherwise>
  </xsl:choose>
</xsl:template>

```



```

<xsl:template match="text()[not(string(number())='NaN')]">
  <xsl:value-of select="."/>
</xsl:template>
<xsl:template match="text()[translate(.,'TRUE','true')='true']">true</xsl:template>
<xsl:template match="text()[translate(.,'FALSE','false')='false']">false</xsl:template>
<xsl:template match="*[count(child::node())=0]">
  <xsl:call-template name="escape-string">
    <xsl:with-param name="s" select="local-name()"/>
  </xsl:call-template>
  <xsl:text>:null</xsl:text>
  <xsl:if test="following-sibling::*">,</xsl:if>
</xsl:template>
<xsl:template match="*" name="base">
  <xsl:if test="not(preceding-sibling::*)"></xsl:if>
  <xsl:call-template name="escape-string">
    <xsl:with-param name="s" select="name()"/>
  </xsl:call-template>
  <xsl:text>:</xsl:text>
  <xsl:apply-templates select="child::node()"/>
  <xsl:if test="following-sibling::*">,</xsl:if>
  <xsl:if test="not(following-sibling::*)"></xsl:if>
</xsl:template>
<xsl:template match="*[count(../*[name(../*)=name(..)]=count(../*) and count(../*)>1]">
  <xsl:if test="not(preceding-sibling::*)"></xsl:if>
  <xsl:choose>
    <xsl:when test="not(child::node())">
      <xsl:text>null</xsl:text>
    </xsl:when>
    <xsl:otherwise>
      <xsl:apply-templates select="child::node()"/>
    </xsl:otherwise>
  </xsl:choose>
  <xsl:if test="following-sibling::*">,</xsl:if>
  <xsl:if test="not(following-sibling::*)"></xsl:if>
</xsl:template>
<xsl:template match="/">
  <xsl:apply-templates select="node()"/>
</xsl:template>
</xsl:stylesheet>

```

现在将一个 XML 文档与之关联，代码如下所示。

```

<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="xml2json.xslt"?>
<books>
  <book>
    <name>Ajax</name>
    <publishDate>2007-12-3</publishDate>
    <author>
      <name>Robin Chen</name>

```

```

        <sex>male</sex>
    </author>
    <price>88.00</price>
    <tags>
        <tag>Ajax</tag>
        <tag>JavaScript</tag>
        <tag>HTML</tag>
        <tag>XMLHttpRequest</tag>
        <tag>XML</tag>
        <tag>JSON</tag>
    </tags>
</book>
<book>
    <name>DHTML</name>
    <publishDate>2007-12-3</publishDate>
    <author>
        <name>Robin Chen</name>
        <sex>male</sex>
    </author>
    <price>44.00</price>
    <tags>
        <tag>Ajax</tag>
        <tag>JavaScript</tag>
        <tag>HTML</tag>
        <tag>XMLHttpRequest</tag>
        <tag>XML</tag>
        <tag>JSON</tag>
    </tags>
</book>
</books>

```

其输出的文本如下所示。

```

{"books":[{"name":"Ajax","publishDate":"2007-12-3","author":{"name":"Robin
Chen","sex":"male"},"price":88.00,"tags":["Ajax","JavaScript","HTML","XMLHttpRequest","XML","JSON"]}, {"name
":"DHTML","publishDate":"2007-12-3","author":{"name":"Robin
Chen","sex":"male"},"price":44.00,"tags":["Ajax","JavaScript","HTML","XMLHttpRequest","XML","JSON"]}]}

```

## 11.5 小 结

本章向读者介绍了 Ajax 程序中除了 XML 的另外一种数据组织方案: JSON。JSON 由于其结构简单、解析方便等特点迅速赢得了广大开发者的喜爱。本章向读者介绍了 JSON 的基本形式和结构,并介绍了在 JavaScript、PHP 和 C#中使用 JSON 的方法。然后分析了 JSON 的优点和缺点,并提供了一种与平台无关的、可复用的 XML 转换为 JSON 的方案。本章并没有给出应用 JSON 的 Ajax 应用实例,读者可以参考第 7 章中的留言本程序。第 12 章将向读者介绍更高级的技巧: JavaScript 面向对象编程(OOP)。

```
<books>
  <book>
    <book>
      <name>DHTML</name>
      <publishDate>2007-12-3</publishDate>
      <author>
        <name>Robin Chen</name>
        <sex>male</sex>
        <author>
          <price>44.00</price>
          <tags>
            <tag>Ajax</tag>
            <tag>JavaScript</tag>
            <tag>HTML</tag>
            <tag>XMLHttpRequest</tag>
            <tag>XML</tag>
            <tag>JSON</tag>
          </tags>
        </author>
      </book>
    </book>
  </books>
```

其输出的文本如下所示:

```
{ "books": [ { "name": "Ajax", "publishDate": "2007-12-3", "author": { "name": "Robin Chen", "sex": "male", "price": 88.00, "tags": [ "Ajax", "JavaScript", "HTML", "XMLHttpRequest", "XML", "JSON"] }, "publishDate": "2007-12-3", "author": { "name": "Robin Chen", "sex": "male", "price": 44.00, "tags": [ "Ajax", "JavaScript", "HTML", "XMLHttpRequest", "XML", "JSON"] } } ] }
```

## 11.2 小结

本章向读者介绍了 Ajax 程序中使用 XML 的另外一种数据组织方案: JSON。JSON 由于其结构简单、数据方便等特点迅速赢得了广大开发者的喜爱。本章向读者介绍了 JSON 的基本语法和结构,并介绍了在 JavaScript、PHP 和 C# 中使用 JSON 的方法。然后介绍了 JSON 的优点和缺点,并提供了使用 Ajax 应用 JSON 的完整方案。本章并给出了应用 JSON 的 Ajax 应用实例,读者可以参考第 7 章中的留言本程序。第 12 章将向读者介绍更高级的技巧: JavaScript 面向对象编程(OOP)。

# 第12章

## JavaScript 面向对象编程 ( OOP )

- ▶▶ JavaScript 中的类
- ▶▶ JavaScript 中的继承
- ▶▶ 更多技巧
- ▶▶ 使用 JSVM 进行代码组织
- ▶▶ 小结

面向对象编程 (Object Oriented Programming, OOP) 是一种计算机编程架构。OOP 的一项基本原则就是计算机程序是由多个能起到子程序作用的单元或对象组合而成的。OOP 达到了软件工程的 3 个重要目标：重用性、灵活性和扩展性。为了实现整体运算，每个对象都可以接收信息、处理数据和向其他对象发送信息。

大部分的 JavaScript 编写者都只把它作为简单的脚本语言，其实 JavaScript 本身就具有强大的面向对象的功能。当不使用强类型时，JavaScript 可以巧妙地实现面向对象的功能，包括封装 (Encapsulation)、多态 (Polymorphism) 和继承 (Inheritance)。



## 12.1 JavaScript 中的类

在 OOP（面向对象编程）中，类定义一类对象，描述对象的属性（数据）和方法（行为）。具有相同或相似性质的对象的抽象就是类。类的具体化就是对象，也可以说类的实例是对象。举个例子来说，“人”是一个类，每个“人”都有名字，“名字”是“人”的属性，每个人都可以行走，“行走”是“人”的方法（行为）。“张三”、“李四”则是具体的一个“人”，可以说“张三”、“李四”都是“人”这个类的实例，每个实例都拥有类所包含的属性和方法，但是可能具体实现又有所不同。例如“张三”、“李四”都有名字，但是一个是“张三”，一个是“李四”。“张三”、“李四”两个人都可以行走，但各自的行走速度又不一样。本节将详细地向读者介绍在 JavaScript 中创建和使用类。

### 12.1.1 创建类

在 JavaScript 中的类使用“function”来声明。例如要定义一个名为 People 的类，代码如下所示。

```
function People()
{
}
```

一个最简单的类就定义完成了，然后可以使用 new 关键字来取得一个 People 类的实例，如下所示。

```
var someone = new People();
```

可以使用 instanceof 运算符来检查实例与类的关系，如果某个对象是某个类的实例，则返回 true，否则返回 false，如下所示。

```
someone instanceof People    //返回 true
```

### 12.1.2 类的属性

12.1.1 小节创建的 People 类没有包含任何数据，显然这是不实用的。在一个对象中，相关的数据被封装在不同的属性中。类的属性就是对象属性的抽象。下面给 People 类添加一个表示名字的名称属性和一个表示性别的 sex 属性，如下所示。

```
function People(name,sex)
{
    this.name = name;
    this.sex = sex;
}
```

这里 function People(){...} 扮演的是 People 类的构造函数的角色，构造函数接受两个参数：name 和 sex，并在构造函数内通过 this 指针来给对象添加 name 和 sex 两个属性，属性值就是相对应的参数值。将 People 实例化，其代码如下所示。



```

function People(name,sex, deposit)
{
    this.name = name;
    this.sex = sex;
    var deposit = deposit;           //私有属性 deposit（存款）
    this.changeName = function(newName)
    {
        this.name = newName;
    }
    this.consume = function(money)   //方法 consume（消费）
    {
        if(deposit >= money)
        {
            deposit -= money;
        }
        else
        {
            throw new Error("没有足够存款");
        }
    }
}

```

在 People 类的构造函数中，定义了一个局部变量 deposit 来表示存款，由于受到其作用域的限制，这个变量只能在函数体内被访问，巧妙地达到了私有属性的效果。这里给 People 类新增了一个 consume（消费）方法，该方法接受一个参数 money 表示消费的金额。在方法内对 deposit 变量进行操作，如果消费金额小于或者等于存款，则将存款减去消费的金额；如果消费金额大于存款，则消费失败，抛出一个异常提示没有足够存款。现在实例化 People，代码如下所示。

```

var susan = new People("Susan","female",1000);
var name = susan.name;           //name 是公有属性，可以访问
var deposit = susan.deposit;      //deposit 是私有属性，访问失败
alert(deposit);                  //输出 undefined
try
{
    susan.consume(500);           //500 小于 deposit 属性的初始值 1000，消费成功，此时 deposit 的属性值被
    改变为 500
}
catch(e)
{
    alert(e.message);
}
try
{
    susan.consume(800);           //800 大于 deposit 属性的值 500，消费失败
}
catch(e)
{
    alert(e.message);            //提示“没有足够存款”
}

```

### 12.1.5 公有方法和私有方法

属性有公有和私有之分，方法也有公有和私有之分，其概念与公有及私有属性类似。通过 `this` 指针添加的方法都是公有方法，公有方法可以被外部调用，例如 `changeName` 方法。私有方法则只能在对象内部被调用，其对外是不可见的，这里同样通过作用域来实现。一个在 JavaScript 中定义私有方法的示例如下所示。

```
function People(name,sex,deposit)
{
    this.name = name;
    this.sex = sex;
    var deposit = deposit;           //私有属性 deposit（存款）
    this.thew = 1;                   //体力
    this.changeName = function(newName)
    {
        this.name = newName;
    }
    this.consume = function(money)   //方法 consume（消费）
    {
        if(deposit >= money)
        {
            deposit -= money;
        }
        else
        {
            throw new Error("没有足够存款");
        }
    }
    var _this = this;                //保存当前对象到变量_this 中供私有方法使用
    var digest = function(food)      //私有方法：digest（消化）
    {
        _this.thew ++;
    }
    this.eat = function(food)        //公有方法：eat（吃）
    {
        digest(food);
    }
}
```

`People` 类在之前的基础上增加了表示体力的共有属性 `thew`，和表示吃东西这个行为的 `eat` 方法。同时，在构造函数内部定义了一个局部变量 `digest`，其保存了一个匿名函数的引用。`digest` 在这里被作为私有方法，表示消化食物这个行为，在 `eat` 方法内被调用。由于在匿名函数内，`this` 指针的指向会发生变化，所以程序中使用了一个 `_this` 变量来保存对当前 `People` 对象的引用，并在私有方法 `digest` 中使用。现在实例化 `People` 类以测试私有方法 `digest`，其代码如下所示。

```
    this.eat = function(food)           //公有方法: eat (吃)
    {
        digest(food);
    }
}
People.prototype = {
    threw:1,
    changeName:function(newName)
    {
        this.name = newName;
    }
}
```

通过 prototype 定义的属性和方法都是公共方法, 在这里将 threw 属性和 changeName 方法的定义放到 prototype 中。下面实例化 People 类以测试 prototype 定义, 代码如下所示。

```
var someone = new People("Susan","female",1000);
someone.changeName("Lily");
alert(someone.name);           //输出 Lily
someone.eat("apple");
alert(someone.threw);          //输出 2
```

使用原型对象的最大好处就是, 开发者可以按照需要随时对类进行扩展, 而不需要改动原有的定义, 这也是动态语言的优点之一, 其代码如下所示。

```
People.prototype.shout = function(words)
{
    alert(words);
}
someone.shout("help");          //输出 help
```

在 12.2 节“JavaScript 中的继承”中, 将进一步介绍 prototype 的作用。

## 12.2 JavaScript 中的继承

继承是面向对象程序设计语言不同于其他语言最重要的特点。所谓继承, 就是子类自动共享父类数据结构和方法的机制, 这是类之间的一种关系。在定义和实现一个类时, 可以在一个已经存在的类的基础上进行, 把这个已经存在的类的内容作为自己的内容, 并加入若干新的内容。在软件开发中, 类的继承使得所建立的软件具有开放性、扩充性, 它简化了对象、类的创建工作, 增加了代码的可重用性。JavaScript 并没有对继承提供语法上的支持, 但是仍然可以巧妙地实现继承的机制。下面向读者详细介绍 JavaScript 中的继承。

### 12.2.1 对象冒充

对象冒充是指一个对象冒充另外一个对象来执行其他对象的方法, 或者说是一个对象将另外一个

对象的方法当作自己的方法来执行。JavaScript 提供了 `call` 和 `apply` 两个方法来实现这种机制。`call` 和 `apply` 方法所实现的功能是一样的，只是参数形式不同，其语法如下所示。

```
functionName.call(object,argument1,argument2,argument3,...);
functionName.apply(object,[argument1,argument2,...]);
```

`call` 和 `apply` 的第一个参数都是一个对象，在某个函数或者方法调用 `call` 或 `apply` 后，该函数会被当作 `call` 或 `apply` 方法第一个参数所指定的对象的方法来执行，而后面的参数则会作为实际参数在函数或者方法执行时传入到函数或者方法中。在 `call` 中，实际参数是直接以 `call` 方法的参数形式出现，而在 `apply` 中，则被放到了一个数组中。下面来看一个对象冒充的例子，代码如下所示。

```
function People(name)
{
    this.name = name;
}
People.prototype.getName = function()
{
    return this.name;
}

var jim = new People("Jim Jackson");
var rob = new People("Robin Chen");
alert(jim.getName());           //输出 Jim Jackson
alert(rob.getName());          //输出 Robin Chen
alert(jim.getName.call(rob));   //输出 Robin Chen
alert(jim.getName.apply(rob,[])); //输出 Robin Chen
alert(rob.getName.call(jim));   //输出 Jim Jackson
alert(rob.getName.apply(jim,[])); //输出 Jim Jackson
```

### 12.2.2 使用对象冒充实现继承

使用对象冒充机制可以巧妙地实现继承机制，其代码如下所示。

```
function Shape(name)
{
    var name = name;
    this.getName = function()
    {
        return name;
    }
}

function Circle(center,radius)
{
    Shape.call(this,"circle"); //采用对象冒充的机制来调用父类的构造函数
    this.center = center;
    this.radius = radius;
}
```

```

var circle = new Circle(10,30);
var name = circle.getName();
alert(name);                //返回 circle

```

程序中首先定义了一个 Shape 类，其包含一个私有属性 name 和一个公共方法 getName。然后定义了一个 Circle 类，在其构造函数中，使用对象冒充机制来调用 Shape 类的构造函数，这样使得 Circle 类中也拥有 Shape 类在其构造函数中所定义的内容，达到了继承的效果。

### 12.2.3 继承原型对象中的属性和方法

对象冒充实现了继承，但是仍然还有一些缺陷。在使用对象冒充机制来实现继承时，子类只是在构造函数中调用了父类的构造函数，但是没有对父类的原型对象作任何处理，这使得定义在父类原型对象中的属性和方法没有被继承。为了弥补这一缺陷，现在对继承方法加以改进，代码如下所示。

```

function Shape(name)
{
    var name = name;
    this.getName = function()
    {
        return name;
    }
}
Shape.prototype = {
    draw:function()
    {
        alert("drawing...");
    }
}
function Circle(center,radius)
{
    Shape.call(this,"circle");    //采用对象冒充的机制来调用父类的构造函数
    this.center = center;
    this.radius = radius;
}
/*
 * 复制 Shape 的原型对象中的成员到 Circle 原型对象中，但不覆盖同名属性/方法
 */
for(var memberName in Shape.prototype)
{
    var memberValue = Shape.prototype[memberName];
    if(!Circle.prototype[memberName])
    {
        Circle.prototype[memberName] = memberValue;
    }
}
var circle = new Circle(10,30);

```



```

var name = circle.getName();
alert(name);           //返回 circle
circle.draw();         //返回 "drawing..."

```

代码中遍历了基类 Shape 的原型对象的所有属性和方法，并将其复制到子类 Circle 的原型对象中，但是如果某个属性或方法已经存在于 Circle 的原型对象中，则不覆盖原有属性或方法。采用复制原型对象的方式成功地解决了对象冒充实现继承的缺陷问题。

### 12.2.4 封装继承方法

为了简化开发，现在将继承的方法进行封装，以方便使用，代码如下所示。

```

/*
 * 通过 Function 的原型对象添加一个 inherit 方法，该方法完成继承逻辑的封装
 * 接受 3 个参数：
 *     instance: 实例
 *     baseClass: 基类
 *     arguments: 传递给基类构造函数的参数
 */
Function.prototype.inherit = function(instance, baseClass, arguments)
{
    this._baseClass = baseClass;           //给子类添加一个 _baseClass 属性，保存对基类的引用
    baseClass.apply(instance, arguments);  //使用对象冒充来调用基类的构造函数

    /*
     * 复制基类原型对象中的属性和方法到子类的原型对象中
     * 如果存在同名的属性或方法，则跳过
     */
    for(var memberName in baseClass.prototype)
    {
        var memberValue = baseClass.prototype[memberName];
        if(!this.prototype[memberName])
        {
            this.prototype[memberName] = memberValue;
        }
    }
}

```

使用方法如下所示。

```

function Shape(name)
{
    var name = name;
    this.getName = function()
    {
        return name;
    }
}
Shape.prototype = {

```

```

        draw:function()
        {
            alert("drawing...");
        }
    }
    function Circle(center,radius)
    {
        Circle.inherit(this,Shape,["circle"]);    //继承 Shape 类
        this.center = center;
        this.radius = radius;
    }
    var circle = new Circle(10,30);
    var name = circle.getName();
    alert(name);    //返回 circle
    circle.draw();    //返回 "drawing..."

```

## 12.3 更多技巧

12.1 节和 12.2 节向读者介绍了在 JavaScript 中使用类和继承的方法, 本节将向读者介绍更多 JavaScript 面向对象编程的技巧, 包括属性的封装、多态、命名空间等。

### 12.3.1 属性的封装

在面向对象的语言中, 例如 C#, 经常会提供属性的存取器 (Getter、Setter), 以完成对对象属性的封装。在 JavaScript 中同样可以实现该操作, 代码如下所示。

```

function People(name)
{
    var name = name;
    this.getName = function()    //Getter
    {
        return name;
    }
    this.setName = function(value)    //Setter
    {
        name = value.replace(/^\s*|\s*$/,"");    //写入属性前, 先去值的首位空格
    }
}
var someone = new People("Robin Chen");
alert(someone.getName());    //输出 Robin Chen
someone.setName(" Jim Green ");
alert(someone.getName());    //输出 Jim Green

```

程序中 People 类的 name 属性以私有属性的形式存储在类中, 而提供给外部的属性访问接口则是 getName 和 setName 方法。这种封装方法使得程序可以在属性存取的过程中增加一些逻辑处理, 并且

即使类内部的实现改变了，但提供给外部的接口不变，不会影响到其他的程序，代码如下所示。

```
function People(firstName,lastName)
{
    var firstName = firstName;
    var lastName = lastName;
    this.getName = function()
    {
        return firstName + ' ' + lastName;           //使用空格连接 firstName 和 lastName 属性并返回
    }
    this.setName = function(value)
    {
        value = value.replace(/^\s*|\s*$/,"");        //去掉首位的空格
        value = value.split(' ');                    //将传入的值按空格分组
        firstName = value[0];                         //将第一组的值写到 firstName 属性中
        lastName = value[1];                         //将第二组的值写到 lastName 属性中
    }
}

var someone = new People("Robin","Chen");
alert(someone.getName());                          //输出 Robin Chen
someone.setName(" Jim Green ");
alert(someone.getName());                          //输出 Jim Green
```

### 12.3.2 实现多态

多态是指给同名的方法提供不同的实现能力，它使得程序可以不用关心方法的具体实现而仅仅依靠其名称来进行调用操作，代码如下所示。

```
/*
 * Bike 类，其实现了 drive 方法
 */
function Bike()
{
    this.drive = function()
    {
        //do something..
    }
}

/*
 * Car 类，其实现了 drive 方法
 */
function Car()
{
    this.drive = function()
    {
        //do something..
    }
}
```

```
/*
 * Bus 类, 其实现了 drive 方法
 */
function Bus()
{
    this.drive = function()
    {
        //do something..
    }
}
/*
 * drive 函数, 其接受一个对象作为参数并调用对象的 drive 方法
 * 但并不关心对象是什么类型以及其 drive 方法如何实现
 */
function drive(obj)
{
    obj.drive();
}
var bike = new Bike();      //Bike 实例
var car = new Car();        //Car 实例
var bus = new Bus();        //Bus 实例
/*
 * 多态性
 */
drive(bike);
drive(car);
drive(bus);
```

在上述程序代码中, drive 函数接受一个对象作为参数, 并调用该对象的 drive 方法, 但 drive 函数并不关心对象是什么类型, 也不关心其 drive 方法是如何实现的, 只要对象提供了 drive 方法, 程序就可以进行处理。而 Bike、Car、Bus 3 个类都实现了 drive 方法, 也就是说提供了 drive 方法的 3 种不同的实现。

技巧: JavaScript 以命名同名方法的形式简单地实现了多态性。

### 12.3.3 命名空间

命名空间是一个层次模型, 其作用是用来避免对象命名上的冲突, 以及将不同功能的类分类放置以方便使用。

在 JavaScript 中, 可以对对象重复定义而不会引起任何的错误, 新的定义会覆盖已有定义, 这个机制给语言带来了巨大灵活性的同时, 也给程序的扩展性和复用性带来了问题。例如, 在某个已有的程序中, 已经存在一个 Worker 类的定义, 代码如下所示。

```
function Worker(name,age,type)
{
    this.name = name;
    this.age = age;
```

```
    this.type = type;
}
```

而在对程序进行扩展时，需要引入一个他人编写好的类库，例如 `classes.js`，类库中同样存在一个名为 `Worker` 的类的定义，代码如下所示。

```
function Worker(idCard)
{
    this.idCard = idCard;
}
```

那么在引入 `classes.js` 这个类库后，新的 `Worker` 类定义会覆盖原有的定义，导致原来的程序不能正常工作，而更麻烦的是，这虽然不会引起任何 JavaScript 的异常，但将使程序的调试变得更加困难。为了避免这种情况发生，使用命名空间是最好的选择。

命名空间在 JavaScript 中的实现很简单，其以一个对象为基数，代码如下所示。

```
/*
 * 定义根命名空间 Sys
 */
if(typeof Sys == 'undefined')
{
    Sys = {};
}
else
{
    if(typeof Sys != 'object')
    {
        throw new Error('命名空间 Sys 类型错误');
    }
}
/*
 * 定义子命名空间 Utility
 */
Sys.Utility = {};
/*
 * 定义属于子命名空间 Utility 下的一个类：StringBuilder
 */
Sys.Utility.StringBuilder = function()
{
    //do Something...
}
```

将类组织在不同的命名空间中，这样即使遇到同名的类，但是其所属的命名空间不同，也不会发生冲突。

### 12.3.4 实现短类名

使用命名空间带来的好处是毫无疑问的，但是同样也带来一些麻烦。那就是如果命名空间嵌套得



## 12.4 使用 JSVM 进行代码组织

在面向对象的语言中,例如 C#, 提供了 `using` 关键字来引入某个命名空间的定义,从而可以使用该命名空间下的所有类。而 JavaScript 并没有这个机制,只能在页面中通过 `<script>` 标签来加载 js 文件,这样就会带来一些问题。

首先,如果按照面向对象编程的思想来组织 JavaScript 代码,代码会被分成不同的模块和类,分布到不同的命名空间中。为了维护方便,一般的惯例是每个类都放在一个单独的文件中。当页面程序的逻辑比较简单时,逐个加载很容易实现;但当程序很复杂,支持其所有功能所需要的类的数量也很庞大时,这样逐个通过 `<script>` 标签来加载会给开发上带来麻烦。试想一下,如果每个页面的头部有几十个上百个 `<script>` 标签,页面将变得十分臃肿,而且书写和维护这些 `<script>` 标签本身也是一项巨大的工程。

其次,在页面打开时加载和解析大量的 js 文件会给用户体验带来明显的影响。而且往往用户并不会用到程序的所有功能,这样实际上加载了大量无用的代码,无效占用了带宽和客户端的内存。但是通过 `<script>` 标签加载 js 文件只能事先将所有可能用到的 js 文件全部加载,而无法提供按需加载的功能。

为了解决上述问题,更好地做到 JavaScript 规范开发和代码复用,JSVM 应运而生。JSVM 是 JavaScript Virtual Machine 的简称,是一个 JavaScript 基础框架、sourceforge 的开源项目。JSVM 提供了完整的代码组织管理的解决方案:命名空间、类、类的按需加载、模块等。同时 JSVM 还提供了一些底层的支持,例如事件管理、IO 模型、异常处理等。JSVM 还有一些高级特性,例如 `smartLoader` 类的自动加载、`debugger` 调试模式等。

本节主要向读者介绍使用 JSVM 对 JavaScript 代码进行组织管理的基本方法,更多高级的特性读者可以访问网站 <http://www.jsvm.org> 进行了解。

### 12.4.1 下载和配置 JSVM

JSVM 可以在网站 <http://www.jsvm.org> 免费下载,当前版本为 2.06。下载的是一个压缩包,解压后的文件中包含文档、示例和 JSVM 的核心程序,如图 12.1 所示。

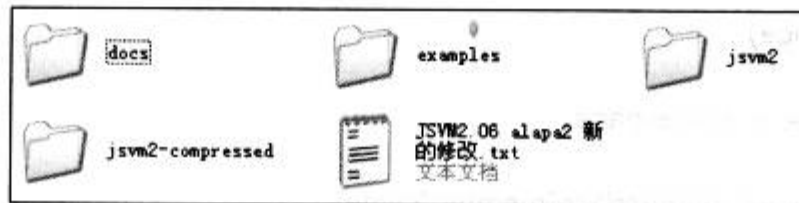


图 12.1 压缩包的内容

JSVM 的核心程序存放在 `jsvm2` 文件夹中。现在将 `jsvm2` 文件夹复制到需要使用 `jsvm` 的项目中。这里为了向读者演示 JSVM 的用法,预先在 Apache 的虚拟目录下建立一个文件夹 `jsvm`,其可以通过网站 <http://localhost/jsvm> 访问到。将 `jsvm2` 文件夹复制到 `jsvm` 目录中,如图 12.2 所示。

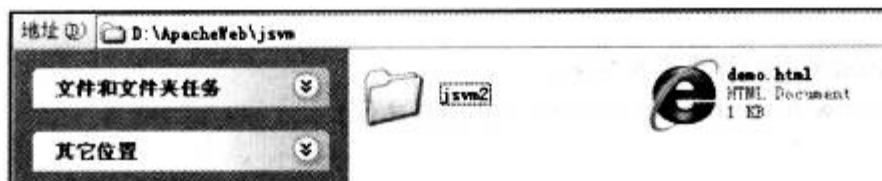


图 12.2 复制 jsvm2 到项目中

## 12.4.2 路径和文件名的约定

打开 jsvm2 文件夹，如图 12.3 所示。

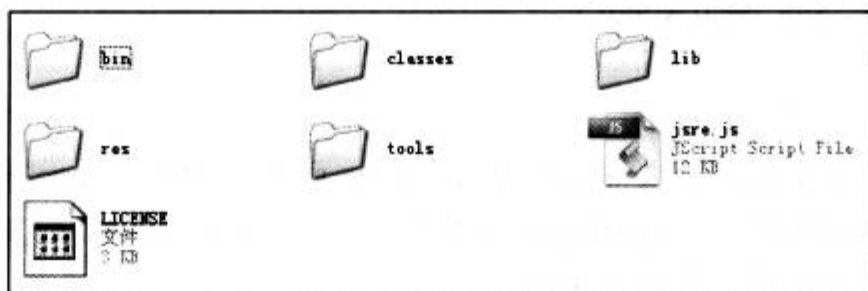


图 12.3 jsvm2 文件夹的结构

其中，bin 文件夹中所包含的是 JSVM 的核心代码；classes 文件夹用来存放所有的类；lib 文件夹用来存放打包好的类库；res 文件夹用来存放一些扩展插件；tools 则存放一些使用的工具，如打包工具等；jsre.js 这个文件是需要在使用 JSVM 的页面中引用的 js 文件；LICENSE 这个文件存放了授权信息。

classes 文件夹中存放了所有的类文件。类文件按照所属的不同 package（包/命名空间）进行分组，每个包就是一个文件夹，包的名字就是文件夹的名字。类文件存放在相应的包文件夹中，每个类就是一个单独的 js 文件，其命名约定为：类名.class.js。classes 文件夹下默认有 3 个包文件夹：example、js 和 org，如图 12.4 所示。

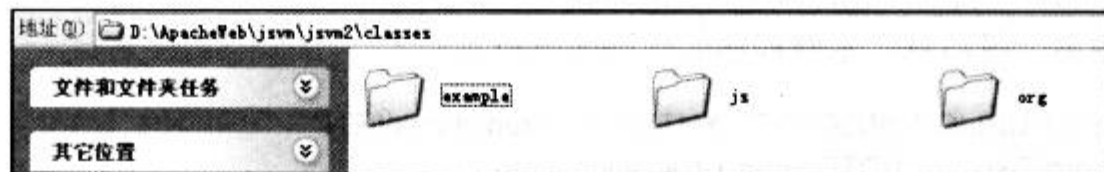


图 12.4 包文件夹

example 包内包含一个 HelloWorld 类，如图 12.5 所示。

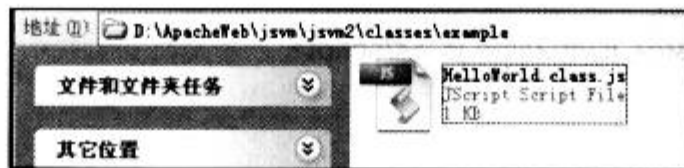


图 12.5 类文件

## 12.4.3 编写类

查看 HelloWorld.class.js 的内容，可以知道如何编写一个 JSVM 的类，其代码如下所示。

```

/**
 * 调用 jsvm2 提供的$package 函数定义包名
 * 作用相当于: if (!window.example) {window.example = {}}
 */
$package("example");
/**
 * 对 example.HelloWorld 进行定义
 */
example.HelloWorld = function (name) {
    this.name = name;
    this.say = function () {
        alert(this.name + " say: hello world!");
    }
}
}

```

其中类的定义读者已经了解, 需要注意的是\$package 函数的使用。\$package 函数由 JSVM 提供, 其作用是注册一个包(命名空间)。\$package 函数接受一个字符串参数, 该参数描述了所要注册的包(命名空间)的名字和位置结构, 代码如下所示。

```

$package("example");           //注册包: example
$package("example.level2");     //注册包: example.level2
$package("example.level2.level3"); //注册包: example.level3

```

#### 12.4.4 类的按需加载

为了向读者演示如何使用 JSVM 来加载存放在 classes 文件夹中的类, 这里编写了一个名为 demo.html 的示例程序, 与 jsvm2 文件夹同级。在示例程序中, 页面被打开时没有加载任何类文件。页面上提供了一个按钮, 当单击按钮时, 程序会加载位于 example 包下的 HelloWorld 类, 并取得该类的一个实例, 最后输出一段消息。示例程序的代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>demo</title>
<script type="text/javascript" src="jsvm2/jsre.js"></script>
<script type="text/javascript">
function execute()
{
    $import("example.HelloWorld");           //调用 jsvm2 提供的$import 函数加载“类”
    var obj = new example.HelloWorld("Robin"); //实例化 HelloWorld 类
    obj.say();                                //调用 say 方法输出信息
}
</script>
</head>

```

```

<body>
  <input type="button" value="测试" onclick="execute();" />
</body>
</html>

```

页面中通过<script>标签引入了 jsre.js 文件, 这是运行 JSVM 所必需的。页面被打开后, 界面如图 12.6 所示。

页面程序将函数 execute 注册成了“测试”按钮的 click 事件处理函数。在函数 execute 中, 使用了 JSVM 提供的\$import 函数来加载一个存放在 classes 文件夹中的类文件。\$import 函数接受一个字符串作为参数, 该参数描述了所要加载的类在包 (命名空间) 结构中的位置以及类的名字, 代码如下所示。

```
$import("example.HelloWorld");
```

在\$import 语句后面可以立即使用所加载的类。程序中取得了这个类的一个实例, 并调用 say 方法来输出信息。单击“测试”按钮, 效果如图 12.7 所示。

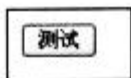


图 12.6 界面效果



图 12.7 单击“测试”按钮

使用\$import 函数加载类, 开发者无须关心 classes 文件夹与当前文件夹之间的路径关系, 只需要知道包 (命名空间) 的结构即可, 剩下的工作全部由 JSVM 来完成, 并且不需要在页面一开始就加载所有的 js 文件, 只是在用到相关功能时才使用\$import 加载, 真正做到了按需加载。

### 12.4.5 在类中引用其他类

在实际应用中, 类与类之间往往会有依赖关系, 某个类依赖于其他一个或者多个类实现的情况很常见。在类文件中, 如果当前的类需要依赖包中的类或者其他包中的类, 同样可以使用\$import 函数来引入这些类的定义。例如现在需要一个 People 类, 其有一个 name 属性, 有一个 sayHello 方法, sayHello 方法的实现依赖于 example 包中的 HelloWorld 类, People 类本身位于 entity 包内。首先在 classes 文件夹内建立 entity 文件夹, 然后在 entity 文件夹内建立 People.class.js 文件, 其实现代码如下所示。

```

/*
 * package: entity
 * class: People
 */

```

```

$import("example.HelloWorld");
$package("entity");
/*
 * People 类的定义
 */

```

```

//导入 example 包内的 HelloWorld 类的定义
//注册 entity 包

```



```

entity.People = function(name)
{
    this.name = name;
    /*
     * sayHello 方法依赖于 HelloWorld 类实现
     */
    this.sayHello = function()
    {
        var helloWorld = new example.HelloWorld(this.name); //以 name 属性值作为参数实例化 HelloWorld
        helloWorld.say(); //调用 say 方法
    }
}

```

现在修改 demo.html 的内容以测试 People 类，代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>demo</title>
<script type="text/javascript" src="jsvm2/jsre.js"></script>
<script type="text/javascript">
function execute()
{
    $import("entity.People"); //加载 People 类
    var rob = new entity.People("Robin"); //实例化 People
    rob.sayHello(); //调用 sayHello 方法
}
</script>
</head>

<body>
<input type="button" value="测试" onclick="execute();" />
</body>
</html>

```

单击页面上的“测试”按钮，运行结果如图 12.8 所示。



图 12.8 运行结果

## 12.5 小 结

本章向读者介绍了 JavaScript 面向对象编程的知识，包括类的创建、属性和方法、公有和私有成员的概念和实现、静态属性和静态方法以及类的原型对象，接着探讨了 JavaScript 中类的继承方式。然后向读者介绍了在 JavaScript 中实现属性的封装、多态、命名空间等技巧。最后介绍了如何使用 JSVM 来有效地对代码进行组织和管理。



# 第13章

## 跨浏览器的兼容性问题

- » 使用 getElementById 代替 idName
- » 表单元素的引用问题
- » 访问集合对象成员的问题
- » 读取自定义属性的问题
- » 常量的定义问题
- » input 元素的 type 属性读写问题
- » 模态窗口的问题
- » frame 的操作问题
- » innerText 的问题
- » 对父元素的引用问题
- » getElementsByName 的问题
- » outerText 的问题
- » outerHTML 的问题
- » 小结

在开发 Ajax 应用程序时, 开发者往往遇到程序代码在不同浏览器下不兼容的问题。在某个浏览器中能正常运行的程序, 到了另外一个浏览器下就完全无法工作。这是因为不同浏览器的内核不同, 而在 JavaScript 解析上存在差异。当前主流的浏览器主要基于两种内核: IE 和 Gecko。IE 内核的浏览器主要包括 IE 系列的各版本浏览器以及像腾讯 TT、傲游这类以 IE 内核为基础二次开发的浏览器。Gecko 内核的浏览器主要包括 Firefox、Netscape 等浏览器。本章将以 IE 和 Firefox 为例, 对浏览器的兼容性问题做一个汇编总结。

## 13.1 使用 getElementById 代替 idName

在 IE 中, 获得一个元素对象的引用, 可以直接使用该元素对象的 id, 以下代码在 IE 中可以正常运行。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title> getElementById </title>
<script type="text/javascript">
window.onload = function()
{
    alert(form.id);           //输出表单的 id 属性, 仅在 IE 下可以使用
}
</script>
</head>

<body>
<form name="form" id="form">
    <input type="text" name="txtInput" id="txtInput" />
    <textarea name="txtArea" id="txtArea"></textarea>
    <input type="radio" id="rdInput" name="rdInput" value="0" />
    <input type="checkbox" id="cbInput" name="cbInput" value="0" />
    <input type="button" id="btnInput" name="btnInput" value="Button" />
</form>
</body>
</html>
```

上述程序代码中直接使用了表单的 id, 即 form 来获取对表单的引用, 在 IE 下的运行结果如图 13.1 所示。

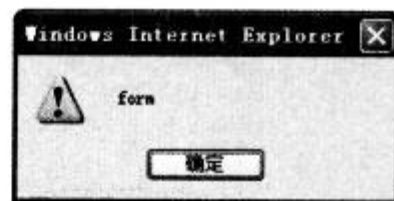


图 13.1 IE 下的运行结果

但是在 Firefox 下这样的操作不允许的。同样的代码在 Firefox 下的运行结果如图 13.2 所示。

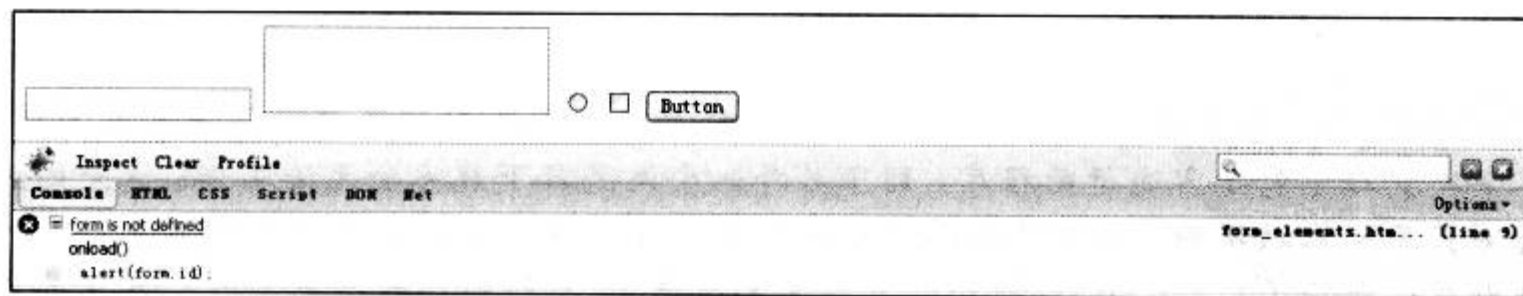


图 13.2 Firefox 下的运行结果

可见, 在 Firefox 下发生了错误, 提示 form 未定义。解决办法是统一使用 getElementById 方法来获得对元素对象的引用, 代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>getElementById</title>
<script type="text/javascript">
window.onload = function()
{
    alert(document.getElementById("form").id);           //兼容 IE 和 Firefox
}
</script>
</head>

<body>
<form name="form" id="form">
    <input type="text" name="txtInput" id="txtInput" />
    <textarea name="txtArea" id="txtArea"></textarea>
    <input type="radio" id="rdInput" name="rdInput" value="0" />
    <input type="checkbox" id="cbInput" name="cbInput" value="0" />
    <input type="button" id="btnInput" name="btnInput" value="Button" />
</form>
</body>
</html>

```

同样，在 IE 中使用 `eval("idName")` 来获得对元素对象引用的方法，在 Firefox 下也无法使用，应统一使用 `getElementById`。

## 13.2 表单元素的引用问题

在 IE 下，通过表单获得某个表单元素的引用，可以使用 `form.item("itemName")`，如下代码在 IE 中可以正常运行。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>form elements</title>
<script type="text/javascript">
window.onload = function()
{
    var form = document.getElementById("form");           //获得表单的引用
    var button = form.item("btnInput");                   //获得名为 btnInput 的表单元素的引用,只在 IE 下有效
    alert(button.value);                                   //输出表单元素的 value 属性值
}
</script>

```

```

</head>

<body>
<form name="form" id="form">
  <input type="text" name="txtInput" id="txtInput" />
  <textarea name="txtArea" id="txtArea"></textarea>
  <input type="radio" id="rdInput" name="rdInput" value="0" />
  <input type="checkbox" id="cbInput" name="cbInput" value="0" />
  <input type="button" id="btnInput" name="btnInput" value="Button" />
</form>
</body>
</html>

```

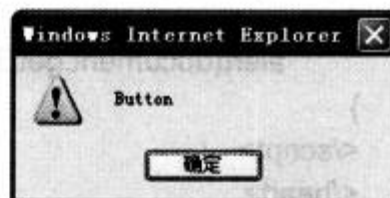


图 13.3 IE 下的运行结果

在 IE 下的运行结果如图 13.3 所示。

同样的代码在 Firefox 下的运行结果如图 13.4 所示。



图 13.4 Firefox 下的运行结果

程序在 Firefox 下发生了错误，提示 `form.item` 不是一个函数。解决办法是使用 `form.elements` 代替 `form.item`，代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>form elements</title>
<script type="text/javascript">
window.onload = function()
{
  var form = document.getElementById("form");      //获得表单的引用
  var button = form.elements["btnInput"];          //使用 elements 来获得对表单元素的引用，在 IE 和 FF 下都有效
  alert(button.value);                             //输出表单元素的 value 属性值
}
</script>
</head>

<body>
<form name="form" id="form">
  <input type="text" name="txtInput" id="txtInput" />
  <textarea name="txtArea" id="txtArea"></textarea>
  <input type="radio" id="rdInput" name="rdInput" value="0" />
  <input type="checkbox" id="cbInput" name="cbInput" value="0" />

```



```
<input type="button" id="btnInput" name="btnInput" value="Button" />
</form>
</body>
</html>
```

### 13.3 访问集合对象成员的问题

在 IE 中, 对集合对象的成员的访问, 可以使用圆括号加索引的方式。例如, 以下代码可以在 IE 中正常运行。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>aggregate</title>
<script type="text/javascript">
window.onload = function()
{
    var form = document.getElementById("form");           //获得表单的引用
    var elements = form.elements;                         //获取表单元素集合
    var firstElement = elements(0);                       //获取第一个元素的引用, 只在 IE 下有效
    alert(firstElement.id);                               //输出第一个元素的 id 属性值
}
</script>
</head>

<body>
<form name="form" id="form">
    <input type="text" name="txtInput" id="txtInput" />
    <textarea name="txtArea" id="txtArea"></textarea>
    <input type="radio" id="rdInput" name="rdInput" value="0" />
    <input type="checkbox" id="cbInput" name="cbInput" value="0" />
    <input type="button" id="btnInput" name="btnInput" value="Button" />
</form>
</body>
</html>
```

其在 IE 下的运行结果如图 13.5 所示。



图 13.5 IE 下的运行结果

同样的代码在 Firefox 下的运行结果如图 13.6 所示。





图 13.6 Firefox 下的运行结果

在 Firefox 下程序发生了错误，提示 `elements` 不是一个函数。解决办法是统一使用方括号加索引的方式来访问集合对象的成员，代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>aggregate</title>
<script type="text/javascript">
window.onload = function()
{
    var form = document.getElementById("form");           //获得表单的引用
    var elements = form.elements;                          //获取表单元素集合
    var firstElement = elements[0];                        //获取第一个元素的引用
    alert(firstElement.id);                                //输出第一个元素的 id 属性值
}
</script>
</head>

<body>
<form name="form" id="form">
    <input type="text" name="txtInput" id="txtInput" />
    <textarea name="txtArea" id="txtArea"></textarea>
    <input type="radio" id="rdInput" name="rdInput" value="0" />
    <input type="checkbox" id="cbInput" name="cbInput" value="0" />
    <input type="button" id="btnInput" name="btnInput" value="Button" />
</form>
</body>
</html>
```

## 13.4 读取自定义属性的问题

在一些应用场合，为了程序处理的需要，会给 HTML 元素添加一些自定义属性，代码如下所示。

```
<input type="text" customAttributeName="customAttributeValue" />
```

在读取自定义属性值时，IE 和 Firefox 同样存在差异。在 IE 中，自定义属性可以像操作对象属性一样直接访问；而在 Firefox 中，自定义属性只能通过 `getAttribute` 方法进行访问，代码如下所示。

```

<title>const</title>
<script type="text/javascript">
const PI = 3.1415926;
alert(PI);
</script>
</head>

<body>
</body>
</html>

```

其在 Firefox 下的运行结果如图 13.9 所示。在 IE 下则会报错，提示语法错误，如图 13.10 所示。

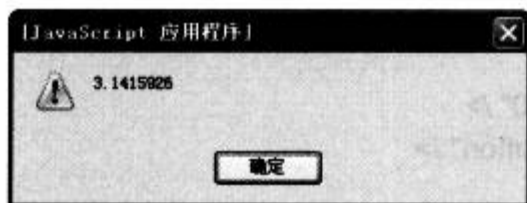


图 13.9 Firefox 下的运行结果

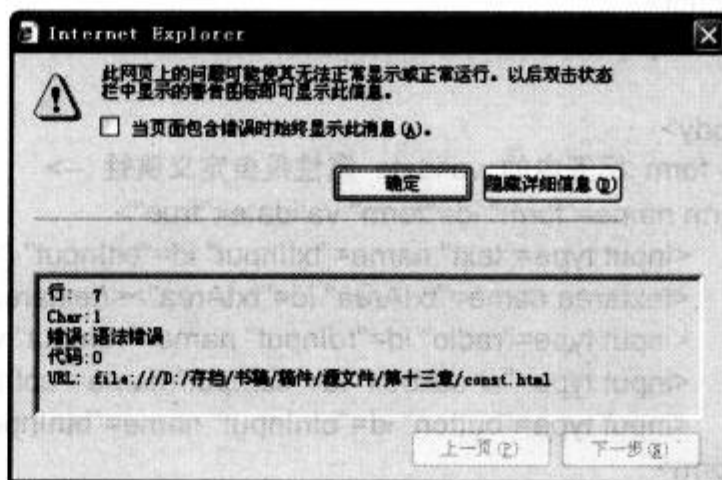


图 13.10 IE 下的运行结果

解决办法是统一使用 var 来定义，只是使用大小写来区分常量和变量，代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>const</title>
<script type="text/javascript">
var PI = 3.1415926;           //常量
var girth = 18;              //变量
</script>
</head>

<body>
</body>
</html>

```

## 13.6 input 元素的 type 属性读写问题

在 Firefox 下，input 元素的 type 属性是可读写的，这个特性可以动态改变 input 元素的形态，代码

如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>input type</title>
<script type="text/javascript">
function changeInputType()
{
    var input = document.getElementById("btnInput"); //获取表单元素的引用
    input.type = "text"; //将 input 的 type 属性设置为 text, 只在 FF 中有效
    delete input.onclick; //删除 click 事件的处理函数
}
</script>
</head>

<body>
<input type="button" id="btnInput" name="btnInput" value="Button" onclick="changeInputType();" />
</body>
</html>
```

在上述程序代码中给页面上的 Button 按钮的 click 事件注册了一个事件处理函数 `changeInputType`，在该函数中 Button 按钮的 `type` 属性值被改写为了 `text`，在 Firefox 下，Button 按钮的形态就会被改变为输入框的样式。在 Firefox 中打开页面，效果如图 13.11 所示。单击 Button 按钮后，效果如图 13.12 所示。

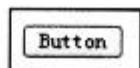


图 13.11 页面初始效果

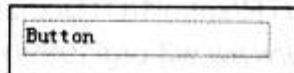


图 13.12 单击 Button 按钮后的效果

可见，按钮不见了，取而代之的是一个输入框。这看起来是一个不错的特性，但是在 IE 下却无法使用。所以除非程序确定不用在 IE 下面运行，否则不要使用这个特性来完成一些动态效果。

## 13.7 模态窗口的问题

在 IE 下，可以使用 `showModalDialog` 来创建模态窗口。原窗口可以给模态窗口传递参数，并接受模态窗口返回的值。模态窗口在关闭前不能失去焦点，且常常被用来制作复杂的对话框。一个在 IE 下使用模态窗口的例子如下所示。

主窗口 `parent.html` 的代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>parent window</title>
<script type="text/javascript">
/*
 * 构造一个对象，其被用来当作参数传递给模态窗口
 */
var paramObj = {
    name:'Robin'
}
/*
 * 创建模态窗口
 */
var returnValue = window.showModalDialog("modal.html",paramObj,"dialogWidth=200px;dialogHeight=100px");
/*
 * 输出返回值
 */
alert("返回值是:" + returnValue);
</script>
</head>

<body>
</body>
</html>

```

模态窗口 modal.html 的代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>modal window</title>
<script type="text/javascript">
var param = window.dialogArguments;           //接收主窗口传递来的参数
window.returnValue = param.name;              //将参数的 name 属性作为窗口的返回值
</script>
</head>

<body>
<h1>模态窗口</h1>
<input type="button" value="关闭" onclick="window.close();" />
</body>
</html>

```

在 IE 下打开 parent.html，出现如图 13.13 所示的模态窗口。单击模态窗口中的“关闭”按钮，如图 13.14 所示。



图 13.13 模态窗口



图 13.14 模态窗口的返回值

在某些场合，使用模态窗口会很方便，但是 Firefox 并不支持模态窗口。如果程序仅仅只是在 IE 下面运行，那么就要放弃模态窗口，使用 `window.open` 来代替。同样 Firefox 不支持的还有非模态窗口 `showModelessDialog`，也需要使用 `window.open` 来代替。

## 13.8 frame 的操作问题

在 IE 中，通过 `window.frameId` 或者 `window.frameName` 都可以获得对 frame 页面 window 对象的引用，代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>frame</title>
<script type="text/javascript">
window.onload = function()
{
    window.frameId.location = "http://www.csdn.net";           //通过 frameId 获得引用，只在 IE 中有效
    //window.frameName.location = "http://www.csdn.net";
}
</script>
</head>

<body>
<h1>Frame</h1>
<iframe id="frameId" name="frameName" src="about:blank" width="750" height="500" />
</body>
</html>
```

在 IE 中打开页面，效果如图 13.15 所示。

Firefox 并不支持通过 `frameId` 获得引用的方式。在 Firefox 下打开页面，程序出错，如图 13.16 所示。





图 13.15 使用 IE 打开页面

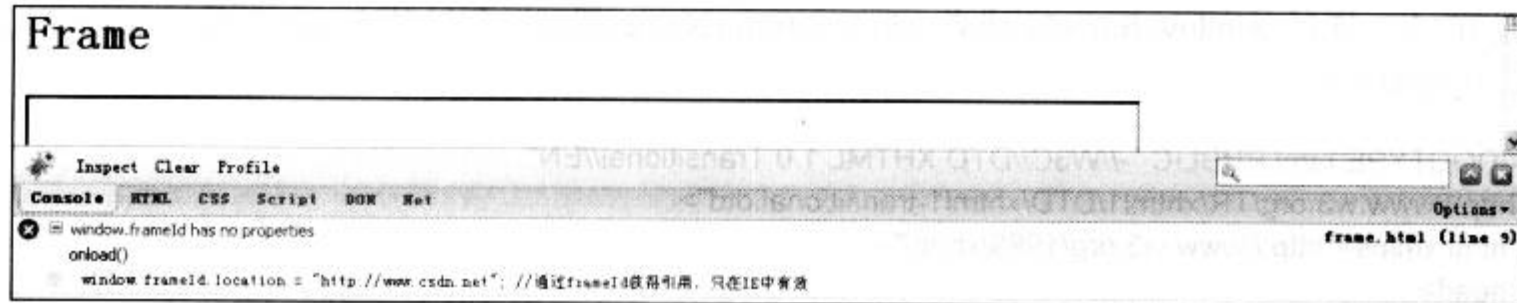


图 13.16 Firefox 下的运行结果

但是 Firefox 支持使用 `frameName` 获得引用, 所以解决办法是统一使用 `frameName`, 代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>frame</title>
<script type="text/javascript">
window.onload = function()
{
    window.frameName.location = "http://www.csdn.net";
}
</script>
</head>

<body>
<h1>Frame</h1>
<iframe id="frameId" name="frameName" src="about:blank" width="750" height="500" />
```

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

还有一个办法就是使用 frame 元素的 contentWindow 属性来获得对 frame 页面 window 对象的引用, 代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>frame</title>
<script type="text/javascript">
window.onload = function()
{
    document.getElementById('frameId').contentWindow.location = 'http://www.csdn.net';
}
</script>
</head>

<body>
<h1>Frame</h1>
<iframe id="frameId" name="frameName" src="about:blank" width="750" height="500" />
</body>
</html>
```

## 13.9 innerText 的问题

在 IE 下可以使用 innerText 属性来读取或设置一个元素内的文本值, 代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>innerText</title>
<script type="text/javascript">
window.onload = function()
{
    var container = document.getElementById('container');
    container.innerText = '内容已经被改变了';
    alert(container.innerText);
}
</script>
</head>

<body>
```

```
//获得对 div 的引用
//改变 innerText 值
//输出 innerText 值
```

```
<div id="containter">innerText demo</div>
</body>
</html>
```

其在 IE 下的运行效果如图 13.17 所示。可见，页面中 div 内的文本内容已经被改变了。但是 Firefox 并不支持 innerText 属性，同样的代码在 Firefox 下的运行效果如图 13.18 所示。



图 13.17 在 IE 下的运行效果



图 13.18 Firefox 下的运行效果

虽然输出的是最新的文本内容，但是页面上的文本内容并没有改变。这里 innerText 被当作一个新的属性被添加到 div 元素对象中，并没有达到预期的效果。其实 Firefox 提供了 textContent 属性，其作用与 IE 下的 innerText 属性一样。所以，在 Firefox 下代码可以写成下面这种形式。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>innerText</title>
<script type="text/javascript">
window.onload = function()
{
    var containter = document.getElementById('containter');
    containter.textContent = '内容已经被改变了';
    alert(containter.textContent);
}
</script>
</head>

<body>
<div id="containter">innerText demo</div>
</body>
</html>
```

//获得对 div 的引用  
//改变 innerText 值  
//输出 innerText 值



图 13.19 textContent

其在 Firefox 下的运行效果如图 13.19 所示。

这样，可以针对浏览器使用不同的方法来达到兼容的效果。问题虽然解决了，但是每次使用 innerText 功能时都需

要判断浏览器的种类,操作过于麻烦,有没有更简单的兼容办法呢?答案是肯定的,那就是使用属性构造器,让 Firefox 也支持 innerText,代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>innerText</title>
<script type="text/javascript">
/*
 * 使用属性构造器来扩展 Gecko Dom 中 HTMLElement 的属性,以让其支持 innerText,达到与 IE 兼容的效果
 */
if(!window.ActiveXObject)
{
    /*
     * 构造 HTMLElement 的 innerText 属性的读取器
     */
    HTMLElement.prototype._defineGetter_("innerText",function()
    {
        return this.textContent;
    });
    /*
     * 构造 HTMLElement 的 innerText 属性的设置器
     */
    HTMLElement.prototype._defineSetter_("innerText",function(value)
    {
        this.textContent = value;
    });
}
window.onload = function()
{
    var container = document.getElementById('container');
    container.innerText = '内容已经被改变了';
    alert(container.innerText);
}
</script>
</head>

<body>
<div id="container">innerText demo</div>
</body>
</html>
```

//获得对 div 的引用  
//改变 innerText 值  
//输出 innerText 值

当浏览器不是 IE 内核时,通过对 HTMLElement 原型的扩展,使非 IE 内核的浏览器也支持 innerText 属性,并且通过 \_defineGetter\_ 和 \_defineSetter\_ 指定该属性的读取和设置方式。这样,只要在程序中加入这段扩展的代码,就可以完成对 innerText 兼容性的处理,保证对外接口的一致性。

## 13.10 对父元素的引用问题

在 IE 中对父元素的引用可以使用 `parentElement` 和 `parentNode`，但是在 Firefox 中只能使用 `parentNode`，代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>parentNode</title>
<script type="text/javascript">
window.onload = function()
{
    var div = document.getElementById('test');
    alert("div.parentNode:" + div.parentNode + "\ndiv.parentElement:" + div.parentElement);
}
</script>
</head>

<body>
<div id="test"></div>
</body>
</html>
```

在 IE 下的运行结果如图 13.20 所示。而同样的代码在 Firefox 下的运行结果如图 13.21 所示。



图 13.20 IE 下的运行结果



图 13.21 Firefox 下的运行结果

可见，在 Firefox 下，`parentElement` 返回的是 `undefined`。所以，应统一使用 `parentNode` 来获取对父元素的引用，而尽量避免使用 `parentElement`。

## 13.11 `getElementsByName` 的问题

在 IE 中，如果给 `div` 添加 `name` 属性，然后通过 `getElementsByName` 来查找这些 `div`，则 `getElements-`



ByName 不能正常工作, 代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>getElementsByName</title>
<script type="text/javascript">
window.onload = function()
{
    var div = document.getElementsByName('test')[0];
    alert(div);
}
</script>
</head>

<body>
<div name="test"></div>
</body>
</html>
```

其在 IE 下的运行结果如图 13.22 所示。可见, 程序返回的是 `undefined`, `getElementsByName` 并没有获得对 `div` 的引用。而同样的代码在 Firefox 中则可以正常运行, 如图 13.23 所示。



图 13.22 IE 下的运行结果



图 13.23 Firefox 下的运行结果

所以, 要尽量避免这种使用方法。如果一定要通过 `name` 属性来查找 `div`, 可以参考下面的方法。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>getElementsByName</title>
<script type="text/javascript">
window.onload = function()
{
    var divs = [];
```

//声明一个空数组

```

var allDivs = document.getElementsByTagName('div');           //取得页面上所有的 div
/*
 * 遍历所有 div, 如果有 div 的 name 属性值等于 test, 则将其加入到数组 divs 中
 */
for(var i = 0; i < allDivs.length; i++)
{
    if(allDivs[i].name == 'test')
    {
        divs.push(allDivs[i]);
    }
}
alert(divs[0]);           //输出第一个符合条件的 div
}
</script>
</head>

<body>
<div name="test"></div>
</body>
</html>

```

其在 IE 下的运行结果如图 13.24 所示。



图 13.24 IE 下的运行结果

## 13.12 outerText 的问题

在 IE 下, HTML 元素有 `outerText` 属性, 其返回值与 `innerText` 属性一样, 但是如果改写元素的 `outerText` 属性, 则元素本身会被文本所替换。也就是说, 当改变一个元素的 `outerText` 属性时, 实际上是用一个文本节点替换了元素, 代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>outerText</title>
<script type="text/javascript">
window.onload = function()
{
    var div = document.getElementById('test');
    div.outerText = '文本值被改变, 原 div 已经被删除';
    alert(!document.getElementById('test'));
}
</script>
</head>

<body>
<div id="test">outerText demo</div>

```

```

</body>
</html>

```

程序在 IE 下的运行结果如图 13.25 所示。但是 Firefox 并不支持 `outerText` 属性, 同样的代码在 Firefox 下的运行结果如图 13.26 所示。



图 13.25 IE 下的运行结果



图 13.26 在 Firefox 下的运行结果

可见, 页面上的文本并没有发生改变, 而且页面上原有的 `div` 仍然存在。解决办法是, 同样适用圆形扩展来让 Firefox 增加对 `outerText` 属性的支持, 代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>outerText</title>
<script type="text/javascript">
/*
 * 如果是非 IE 内核的浏览器, 则扩展 HTMLElement 原型, 使其支持 outerText 属性
 */
if(!window.ActiveXObject)
{
    /*
     * 构造 outerText 属性的读取器
     */
    HTMLElement.prototype._defineGetter_("outerText",function()
    {
        return this.textContent; //返回元素的 textContent 属性
    });
    /*
     * 构造 outerText 属性的设置器
     */
    HTMLElement.prototype._defineSetter_("outerText",function(value)
    {
        var textNode = document.createTextNode(value); //创建文本节点
        this.parentNode.replaceChild(textNode,this); //使用文本节点替换原元素
    });
}

```

```

window.onload = function()
{
    var div = document.getElementById('test');
    div.outerText = '文本值被改变, 原 div 已经被删除';
    alert(!document.getElementById('test'));
}
</script>
</head>

<body>
<div id="test">outerText demo</div>
</body>
</html>

```

其在 Firefox 下的运行效果如图 13.27 所示。  
至此, 就成功解决了兼容性的问题。



图 13.27 Firefox 下的运行效果

### 13.13 outerHTML 的问题

在 IE 下, 读取元素的 outerHTML 属性, 可以得到包含元素及其子孙元素在内的所有 HTML 表示。如果修改元素的 outerHTML 属性, 则会使用一段新的 HTML 代码替换原有的元素。替换后, 原有的元素会从页面中被删除, 代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>outerHTML</title>
<script type="text/javascript">
window.onload = function()
{
    var div = document.getElementById('test');
    div.outerHTML = '<input type="text" />';
    alert(!document.getElementById('test'));
}
</script>
</head>

<body>
<div id="test">outerHTML demo</div>
</body>
</html>

```

程序在 IE 下的运行结果如图 13.28 所示。可见, div 被替换成了文本框, 并且输出的信息为 false, 表示原来的 div 已经不存在于页面中了。但是 Firefox 并不支持 outerHTML 属性, 程序在 Firefox 下的运行结果如图 13.29 所示。



图 13.28 IE 下的运行结果

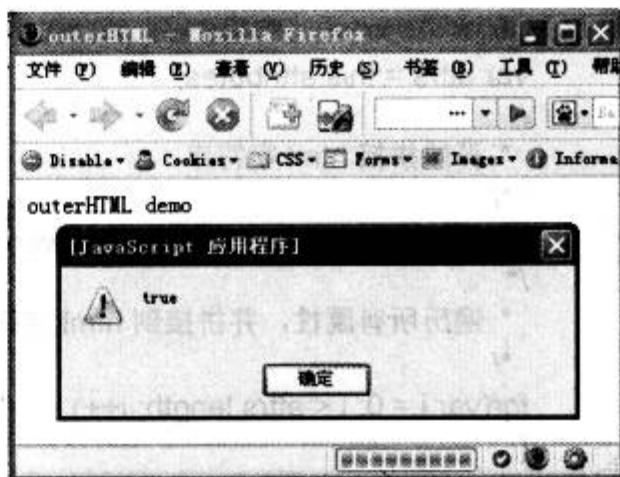


图 13.29 在 Firefox 下的运行结果

解决办法同样是通过圆形扩展来让 Firefox 增加对 outerHTML 属性的支持，代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>outerHTML</title>
<script type="text/javascript">
/*
 * 如果是非 IE 内核的浏览器，则扩展 HTMLElement 原型，使其支持 outerText 属性
 */
if(!window.ActiveXObject)
{
    /*
     * 构造 outerHTML 属性的设置器
     */
    HTMLElement.prototype._defineSetter_("outerHTML",function(sHTML)
    {
        /*
         * 将 HTML 片段转换为 DOM 元素
         */
        var range = this.ownerDocument.createRange();
        range.setStartBefore(this);
        var domFragment = range.createContextualFragment(sHTML);
        /*
         * 替换原有元素
         */
        this.parentNode.replaceChild(domFragment,this);
    });
    /*
     * 构造 outerHTML 属性的读取器
     */
    HTMLElement.prototype._defineGetter_("outerHTML",function()
    {
        var attr;
        /*
         * 取得元素的所有属性
         */
    });
}
```



```

    */
    var attrs = this.attributes;
    /*
    * 开始拼接 html 字符串
    */
    var str = "<" + this.tagName.toLowerCase();
    /*
    * 遍历所有属性，并拼接到 html 字符串中
    */
    for(var i = 0; i < attrs.length; i++)
    {
        attr = attrs[i];
        if(attr.specified)
        {
            str += " " + attr.name + "=" + attr.value + "";
        }
    }
    str += ">" + this.innerHTML + "</" + this.tagName.toLowerCase() + ">";
    return str;
});
}
window.onload = function()
{
    var div = document.getElementById('test');
    div.innerHTML = '<input type="text" />';
    alert(!document.getElementById('test'));
}
</script>
</head>

<body>
<div id="test">outerHTML demo</div>
</body>
</html>

```

其在 Firefox 下的运行结果如图 13.30 所示。

至此，就成功地解决了 outerHTML 属性的兼容问题。



图 13.30 Firefox 下的运行结果

## 13.14 小 结

本章对 Ajax 开发中可能遇到的浏览器兼容性问题进行了总结和分析，并辅以实例说明了解决办法。这些兼容性的问题包括：idName 的使用、表单元素的引用、集合对象成员的访问、自定义属性的读取、常量的定义、input 元素 type 属性的读写问题、模态窗口的使用、frame 元素的操作、父元素的引用、getElementsByName 的问题以及 innerText、outerText 和 outerHTML 在 Firefox 下的实现。除此之外，本书前面的章节也介绍了一些兼容性处理的方法，例如事件对象的兼容性处理、XML DOM 的兼容性处理等，在本章中就不再重复介绍了。

# 第 14 章

## Prototype.js 框架介绍

- ▶▶ Prototype.js 常用方法介绍
- ▶▶ 基于 Prototype.js 的类和继承
- ▶▶ Prototype.js 中的事件处理
- ▶▶ Prototype.js 的 Ajax 功能
- ▶▶ 基于 Prototype.js 的留言本程序
- ▶▶ 小结

随着 Ajax 技术的不断发展, 应用的不断深入, 逐渐涌现出了一批 JavaScript 开发框架。这些框架有些关注于 JavaScript 开发的底层支持, 提供兼容性的封装, 提供一些便捷有效的方法等; 有些则提供了更高层面的功能, 例如 UI、动画效果等。使用这些框架可以极大地简化 JavaScript 的开发工作, 加快项目进程。其中作为 JavaScript 的基本开发框架, Prototype.js 要最具知名度。

Prototype.js 是由 Sam Stephenson 编写的 JavaScript 和 Ajax 基本开发框架, 它是开源和免费的, 可以在网站 <http://www.prototypejs.org/> 下载并获得文档支持, 当前最新版本是 1.5.1。本章将向读者介绍 Prototype.js 的常用方法。

## 14.1 Prototype.js 常用方法介绍

Prototype.js 其实更像一个大类库，提供了大量方便 JavaScript 和 Ajax 开发的方法。熟悉这些方法可以极大地加快项目的开发进程。本节将结合实例，向读者介绍 Prototype.js 中最常用的一些方法。

### 14.1.1 使用\$方法代替 document.getElementById

Prototype.js 最具代表性的方法之一，就是\$方法。\$方法最直接的作用就是代替 document.getElementById 使用。在 JavaScript 程序中，经常需要根据元素 id 来取得对元素的引用，而 document.getElementById 的书写过于冗长，这时，就可以使用 Prototype.js 提供的\$方法来代替。例如，要取得页面上一个 id 为 container 的 div 元素，可以使用如下代码。

```
var divElement = $('id');           //返回 id 为 id 的元素对象
```

当然，\$方法的作用不仅仅如此，Prototype.js 给该方法增加一些使用的功能。首先，\$方法允许传入多个参数，并返回一个元素对象的数组，如下所示。

```
var elements = $('id1', 'id2', 'id3'); //返回由 id 为 id1、id2、id3 3 个元素组成的数组
```

其次，也是最重要的一点，就是 Prototype.js 给每个从\$方法获得的对象增加了很多实用的方法，例如，从\$方法获得一个 div 元素的引用后，可以直接调用其 hide 方法来隐藏这个 div 元素，如下所示。

```
var div = $('divId');           //获得 div 的引用  
div.hide();                     //隐藏 div 元素
```

注意：如果直接给\$方法传入元素对象的引用，返回的是被 Prototype.js 扩展后的元素对象。

### 14.1.2 使用\$\$方法获得元素引用

\$\$方法同样用于获得元素的引用，与\$不同的是，它接受一个 css 选择器作为参数，将所有符合该选择器的元素作为数组返回，如下所示。

```
$$("div");                       //返回页面上所有的 div  
$$("#content");                  //与$("content")作用基本一致，但返回的是一个数组  
$$("li.first");                  //返回所有 class 属性是 first 的 li 元素  
$$("#content a");                //返回 id 为 content 的元素下的所有 a 元素  
$$("#a[href='#]");               //返回所有 href 属性为#的 a 元素
```

### 14.1.3 根据 css 类名取得元素集合

Prototype.js 给 document 对象新增了一个 getElementsByClassName 的方法，通过指定一个 css 类名为参数，来获取当前页面中所有应用了该 css 类样式的元素集合。下面是一个使用该方法来改变页面上内容标题字体大小的例子，代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>getElementsByClassName</title>
<script type="text/javascript" src="prototype1.5.js"></script>
<style type="text/css">
.title {font-weight:bold;}
</style>
<script type="text/javascript">
function changeFontSize()
{
    var titles = document.getElementsByClassName('title');           //获取所有 class 属性为 title 的元素
    titles.each(function(title)                                       //调用数组的 each 方法来进行迭代, 该方法
    由 Prototype.js 提供
    {
        title.style.fontSize = '20pt';                               //设置元素字体大小
    });
}
</script>
</head>

<body>
<h1>getElementsByClassName demo</h1>
<p class="title">标题一</p>
<p>内容内容内容内容内容内容内容内容内容内容内容内容内容内容内容内容内容内容内容内容内容内容内容</p>
<p class="title">标题二</p>
<p>内容内容内容内容内容内容内容内容内容内容内容内容内容内容内容内容内容内容内容内容内容内容内容</p>
<input type="button" value="改变字体大小" onclick="changeFontSize()" />
</body>
</html>

```

打开页面后, 效果如图 14.1 所示。单击“改变字体大小”按钮, JavaScript 程序会通过 `getElementsByClassName` 方法获取所有 class 属性为 `title` 的元素, 并将其字体大小改变为 20pt, 如图 14.2 所示。

## getElementsByClassName demo

标题一

内容

标题二

内容

改变字体大小

图 14.1 界面效果

## getElementsByClassName demo

标题一

内容

标题二

内容

改变字体大小

图 14.2 改变字体大小



### 14.1.4 使用 Try.these 尝试运行多个函数

Prototype.js 提供了 Try.these 方法来尝试执行多个函数，其接受一个或者多个函数作为参数，并返回第一个成功运行的函数的返回值，其语法如下所示。

```
Try.these(func1[,func2[,func3...]]);
```

Try.these 方法可以用来替代多个 try...catch 语句，以简化编码。例如，可以使用 Try.these 方法来完成 XMLHttpRequest 对象的创建，代码如下所示。

```
var xmlhttprequest = Try.these(  
    function()  
    {  
        return new XMLHttpRequest();  
    },  
    function()  
    {  
        return new ActiveXObject("Microsoft.XMLHTTP");  
    }  
);
```

### 14.1.5 使用 \$F 方法来获得表单元素的值

Prototype.js 提供了 \$F 方法来获得表单元素的值，其接受一个字符串参数，表示需要取值的表单元素的 id 值，并返回该表单元素的值。一个使用 \$F 方法的示例如下所示。

```
<!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>  
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  
<title>$F demo</title>  
<script type="text/javascript" src="prototype1.5.js"></script>  
</head>  
  
<body>  
<input type="text" id="txtName" name="txtName" />  
<input type="button" onclick="alert($F('txtName'));" value="显示输入框值" />  
</body>  
</html>
```

页面效果如图 14.3 所示。在文本框中输入“Test”，并单击“显示输入框值”按钮，效果如图 14.4 所示。





图 14.3 界面效果

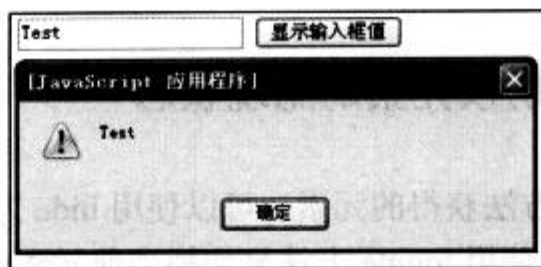


图 14.4 显示输入框的值

### 14.1.6 数组迭代的简化方法

Prototype.js 对 Array 对象的原型进行了扩展，为其增加了一个名为 `each` 的方法来完成对数组的迭代，其语法如下所示。

```
array.each(function(arrayItem,arrayIndex)
{
    //do something..
});
```

`each` 方法接受一个函数作为参数，该函数会在迭代的过程中被调用，`each` 方法在每次迭代中都会将当前的数组元素和索引值作为参数传入该函数。例如，需要将一个数字数组的每一项的值加 1，传统的做法如下所示。

```
var array = [1,2,3,4,5];
for(var i = 0,i = array.length; i < i; i++)
{
    array[i]++;
}
```

而使用 Prototype.js 提供的 `each` 方法，则只需要下面的代码就可以完成。

```
var array = [1,2,3,4,5];
array.each(function(item)
{
    item++;
});
```

### 14.1.7 设置元素的样式

使用 `$` 方法获得的元素，Prototype.js 会为其加上一些使用的方法，其中就包括 `setStyle` 方法。`setStyle` 方法可以用来设置元素的 CSS 样式，其接受一个对象作为参数，对象的每一个属性对应一个 CSS 样式属性，对象的属性名就是 CSS 样式的属性名，属性值就是 CSS 样式的属性值，代码如下所示。

```
var div = $("divId");
div.setStyle({
    cssFloat:"left",
    opacity:0.5
});
```

### 14.1.8 切换元素的隐现状态

通过\$方法获得的元素,可以使用 hide 方法来隐藏该元素,也可以使用 show 方法来显示隐藏的元素,还可以使用 toggle 方法来切换当前元素的隐现状态。其语法如下所示。

```
var div = $("divId");
div.hide();           //隐藏元素
div.show();           //显示元素
div.toggle();         //切换元素隐藏/现实的状态
```

一个使用这些方法的示例如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Element.hide,Element.show,Element.hide</title>
<script type="text/javascript" src="prototype1.5.js"></script>
<script type="text/javascript">
function hideDiv()
{
    $('test').hide();           //隐藏 DIV
}

function showDiv()
{
    $('test').show();           //显示 DIV
}

function toggleDiv()
{
    $('test').toggle();         //切换 DIV 的隐现状态
}
</script>
<style type="text/css">
#test {
    width:400px;
    height:200px;
    border:1px solid black;
    background-color:#ccc;
    margin-bottom:10px;
}
</style>
</head>

<body>
<div id="test">DIV</div>
<input type="button" onclick="hideDiv()" value="隐藏 DIV" />
<input type="button" onclick="showDiv()" value="显示 DIV" />
```

```
<input type="button" onclick="toggleDiv()" value="切换 DIV 的隐现状态" />
</body>
</html>
```

界面效果如图 14.5 所示。单击“隐藏 DIV”按钮，DIV 被隐藏，如图 14.6 所示。单击“显示 DIV”按钮，DIV 重新显示在页面上，如图 14.7 所示。

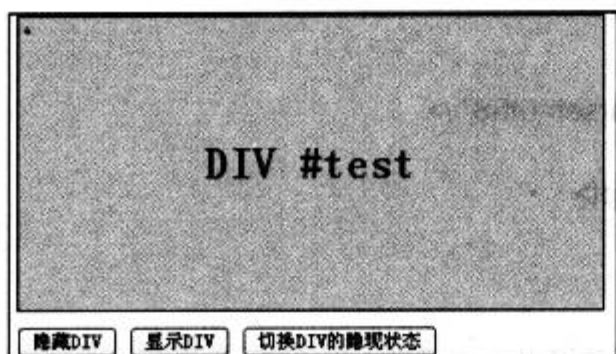


图 14.5 界面效果



图 14.6 隐藏 DIV



图 14.7 显示 DIV

使用 `toggle` 方法切换元素隐现状态时，如果元素当前被隐藏，则元素会被显示，如果元素当前是现实状态，则元素会被隐藏。单击“切换 DIV 的隐现状态”按钮，由于 DIV 当前是现实状态，则 DIV 会被隐藏掉，如图 14.8 所示。

再次单击“切换 DIV 的隐现状态”按钮，DIV 重新被显示出来，如图 14.9 所示。

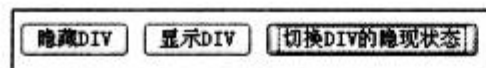


图 14.8 切换隐现状态（一）



图 14.9 切换隐现状态（二）

### 14.1.9 序列化单值

在 Ajax 应用程序中，经常会使用 `XMLHttpRequest` 来提交数据而不是使用传统的表单，但是仍然

会以表单为单位来组织界面。Prototype.js 提供了 `serialize` 方法来完成对表单值的序列化。使用 `$` 方法获取的表单元素，Prototype.js 会对其扩展，增加 `serialize` 方法来完成对其表单项的序列化输出。一个示例程序如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>form serialize</title>
<script type="text/javascript" src="prototype1.5.js"></script>
<script type="text/javascript">
function serializeForm()
{
    alert($('fmLogin').serialize());    //序列化表单值并输出
}
</script>
<style type="text/css">
label,input {
    display:block;
    margin-bottom:5px;
}
</style>
</head>

<body>
<form id="fmLogin" name="fmLogin">
    <label for="txtUsername">Username</label>
    <input type="text" id="txtUsername" name="txtUsername" />
    <label for="txtEmail">Email</label>
    <input type="text" id="txtEmail" name="txtEmail" />
    <input type="button" value="查看序列化值" onclick="serializeForm()" />
</form>
</body>
</html>
```

界面效果如图 14.10 所示。在文本框中输入一些数据，并单击“查看序列化值”按钮，效果如图 14.11 所示。

图 14.10 界面效果



图 14.11 序列化表单值

可见，输出的数据是表单的值经过序列化后的字符串表示：`txtUsername=Robin&txtEmail=robchen%40126.com`。

### 14.1.10 转换 HTML 标签

在一些 Ajax 程序中，例如留言本程序，经常需要对用户输入的值进行 HTML 编码处理，以防止输入的内容中包含 HTML 标签而破坏应用程序。Prototype.js 对 String 对象进行了扩展，并提供了 `escapeHTML` 方法来完成之一工作，其语法如下所示。

```
var str = htmlstring.escapeHTML();
```

一个使用 `escapeHTML` 的示例如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>escapeHTML</title>
<script type="text/javascript" src="prototype1.5.js"></script>
<script type="text/javascript">
function showEscapeHTMLResult()
{
    var value = $F('txtInput');           //取得文本域的值
    alert(value.escapeHTML());           //转换 html 标签并输出结果
}
</script>
<style type="text/css">
textarea {
    display: block;
    margin-bottom: 10px;
}
</style>
</head>

<body>
<textarea id="txtInput" name="txtInput" rows="10" cols="50"></textarea>
<input type="button" value="查看 escapeHTML 结果" onclick="showEscapeHTMLResult()" />
</body>
</html>
```

示例程序的界面效果如图 14.12 所示。

在文本域中输入一些包含 HTML 标签的内容，然后单击“查看 `escapeHTML` 结果”按钮，效果如图 14.13 所示。

可见，所有的 HTML 标签都被替换成了对应的实体。



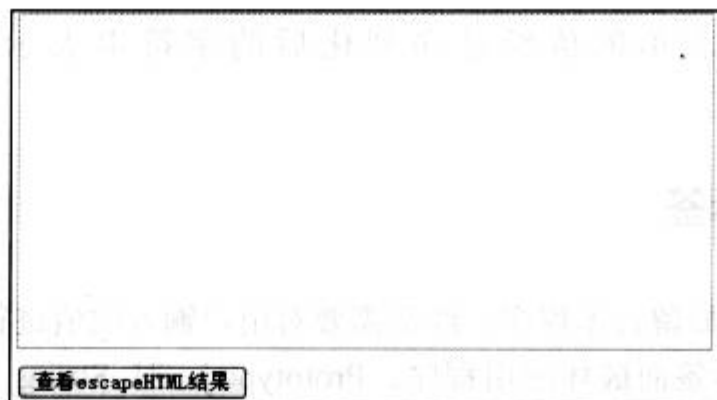


图 14.12 界面效果

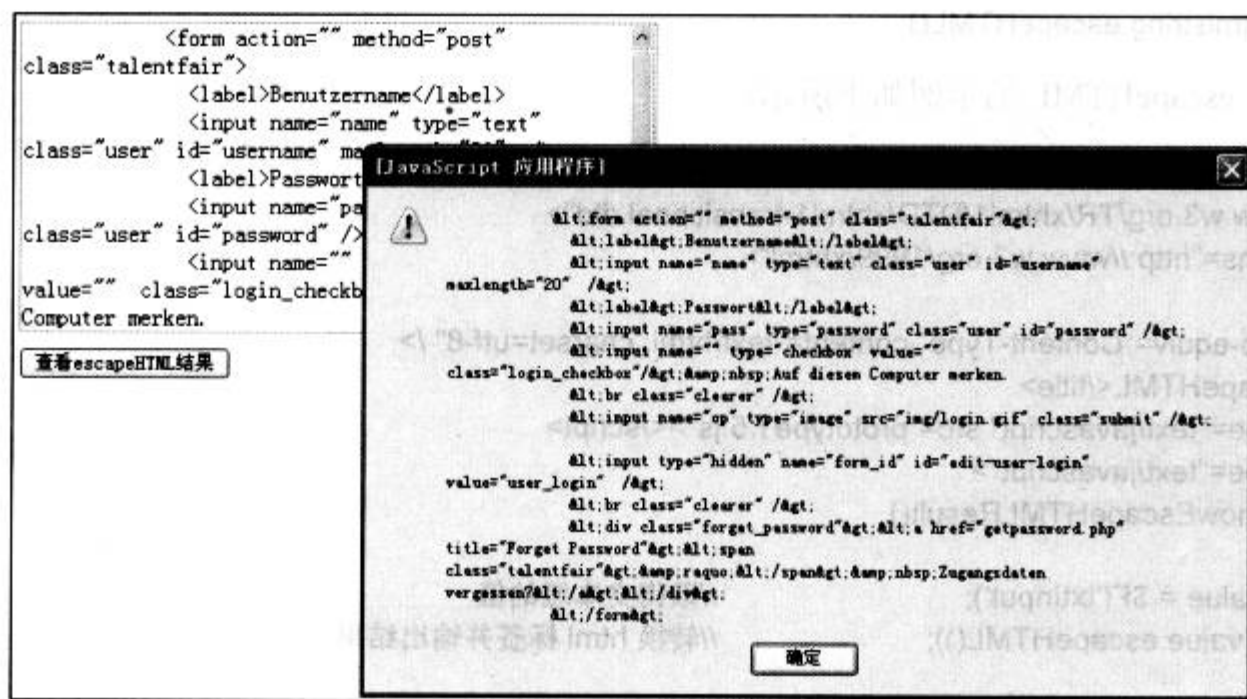


图 14.13 escapeHTML 结果

说明：与 escapeHTML 方法对应，Prototype.js 还提供了 unescapeHTML，其作用与 escapeHTML 相反，是将实体转换回 HTML 标签。

## 14.2 基于 Prototype.js 的类和继承

作为 JavaScript 的基础开发框架，Prototype.js 也提供了一套类和继承的体系。本节将向读者介绍如何在 Prototype.js 下创建类，以及实现类的继承。

### 14.2.1 使用 Class.create() 创建类

在 Prototype.js 中创建一个类很简单，只需要使用如下语法即可。

```
var class = Class.create();
```

Class 是 Prototype.js 提供的一个全局对象，在 Prototype.js 中对 Class 对象的定义如下所示。

```

var Class = {
  create: function() {
    return function() {
      this.initialize.apply(this, arguments);
    }
  }
}

```

这里实现得很巧妙。Class 包含一个 create 方法，该方法返回一个函数。返回的函数是作为构造函数存在的，其内部调用了自身的 initialize 方法来完成构造过程。所以用 Class.create() 创建的类的 prototype 原型对象中需要实现一个名为 initialize 的方法，其作用就相当于构造函数。一个完整的创建类的示例如下所示。

```

var person = Class.create();           //类的声明
person.prototype = {
  initialize: function(name, sex)      //构造函数
  {
    this.name = name;
    this.sex = sex;
  },
  anotherProperty: "something",        //属性
  anotherMethod: function()           //方法
  {
    //do something..
  }
}

```

### 14.2.2 Prototype.js 中的继承

Prototype.js 提供了 Object.extend 方法来复制对象的属性和方法，其实现代码如下所示。

```

Object.extend = function(destination, source) {
  for (var property in source) {
    destination[property] = source[property];
  }
  return destination;
}

```

这里可以借助这个方法来完成类的继承，代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Inherit</title>
<script type="text/javascript" src="prototype1.5.js"></script>
<script type="text/javascript">

```

```
var Person = Class.create();           //定义一个 Person 类
Person.prototype = {
  initialize:function(name,sex)        //Person 类的构造函数
  {
    this.name = name;
    this.sex = sex;
  },
  changeName:function(newName)         //Person 类的 changeName 方法
  {
    this.name = newName;
  }
}

var Woman = Class.create();            //声明一个 Woman 类
Object.extend(Woman.prototype,Person.prototype); //复制 Person 类的原型对象中的成员到 Woman 类的原型对象中
Woman.prototype.initialize = function(name) //重写 Woman 类的构造函数
{
  this.name = name;
  this.sex = "female";
}

var someone = new Woman("Susan");       //实例化 Woman
alert(someone.name);                    //输出 Susan
alert(someone.sex);                     //输出 female
someone.changeName("Lily");              //调用 changeName 方法来改变 name 属性
alert(someone.name);                    //输出 Lily
</script>
</head>

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

## 14.3 Prototype.js 中的事件处理

Prototype.js 提供了事件注册和注销的兼容性封装，并提供了事件句柄的管理功能。本节将向读者介绍如何使用 Prototype.js 来注册和注销事件处理函数。

### 14.3.1 注册事件处理函数

Prototype.js 提供了 Event.observe 来注册事件处理函数，其语法如下所示。

Event.observe(element, eventName, handler[, useCapture = false])

element 为发生事件的对象；eventName 为事件名称，是不带“on”前缀的，例如 click、keydown

等；handler 是事件处理函数；userCapture 是可选参数，表示是否在捕捉阶段处理事件，默认为 false。一个使用 Event.observe 方法的示例如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Event.observe</title>
<script type="text/javascript" src="prototype1.5.js"></script>
<script type="text/javascript">
window.onload = function()
{
    Event.observe($('btnTest'),'click',showMsg);    //使用 Event.observe 为#idTest 按钮的 click 事件注册
    一个事件处理函数
}

function showMsg()
{
    alert("事件处理函数已经被注册");
}
</script>
</head>

<body>
<input type="button" id="btnTest" name="btnTest" value="测试" />
</body>
</html>
```

打开示例程序，并单击“测试”按钮，效果如图 14.14 所示。

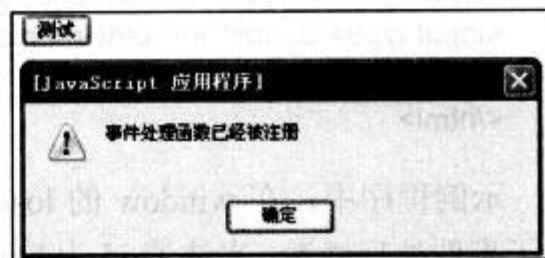


图 14.14 Event.observe 示例

### 14.3.2 注销事件处理函数

使用 Event.stopObserving 方法可以注销一个事件处理函数，其语法如下所示。

Event.stopObserving(element, eventName, handler[, useCapture = false])

element 为发生事件的对象；eventName 为事件名称，是不带“on”前缀的，例如 click、keydown 等；handler 是事件处理函数；userCapture 是可选参数，表示是否在捕捉阶段处理事件，默认为 false。一个使用 Event.stopObserving 方法的示例如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Event.observe</title>
<script type="text/javascript" src="prototype1.5.js"></script>
<script type="text/javascript">
```

```
window.onload = function()
{
    Event.observe($('btnTest'),'click',showMsg);           //使用 Event.observe 为#idTest 按钮的 click 事件注册
    一个事件处理函数
    Event.observe($('btnUnregist'),'click',unregist);      //使用 Event.observe 为#idUnregist 按钮的 click 事件
    注册一个事件处理函数
}

function showMsg()
{
    alert("事件处理函数已经被注册");
}

function unregist()
{
    Event.stopObserving($('btnTest'),'click',showMsg);
    alert("事件处理函数已经被注销");
}
</script>
</head>

<body>
<input type="button" id="btnTest" name="btnTest" value="测试" />
<input type="button" id="btnUnregist" name="btnUnregist" value="注销事件处理函数" />
</body>
</html>
```

示例程序中，在 window 的 load 事件中，为 id 为 btnUnregist 的按钮注册了事件处理函数，来注销 id 为 btnTest 的按钮的 click 事件的处理函数。打开示例程序，效果如图 14.15 所示。

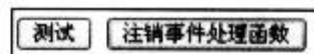


图 14.15 示例程序界面

单击“测试”按钮，触发了该按钮的 click 事件并调用注册的事件处理程序 showMsg，效果如图 14.16 所示。再单击“注销事件处理函数”按钮，来注销“测试”按钮的 click 事件处理函数，效果如图 14.17 所示。然后再次单击“测试”按钮，可以看到没有任何对话框出现了。

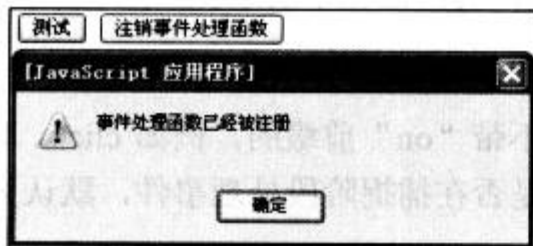


图 14.16 单击“测试”按钮

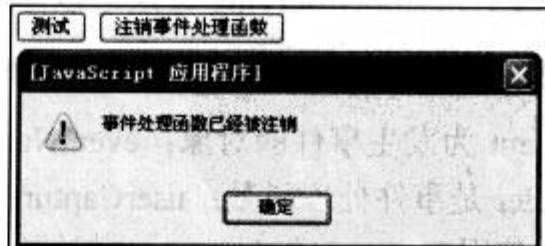


图 14.17 注销事件处理函数

## 14.4 Prototype.js 的 Ajax 功能

Prototype.js 提供了全局的 Ajax 对象来完成对 Ajax 相关功能的支持，并对表单元素进行了扩展以简化表单数据 Ajax 方式的提交。本节将向读者介绍 Prototype.js 的功能。



### 14.4.1 Ajax.Request 方法详解

Prototype.js 的 Ajax 功能最核心的是 Ajax.Request 类，其他提供 Ajax 功能的类都是基于这个类的扩展，本节将向读者详细讲解该类的使用方法。Ajax.Request 方法的语法如下所示。

```
new Ajax.Request(url[,options]);
```

其中 url 是要请求的地址；options 是一个可选参数，对象类型，可以指定请求的一些参数。一个完整参数的 Ajax.Request 类的使用示例如下所示。

```
new Ajax.Request("service.php",{
  asynchronous:true,           //是否使用异步请求，默认为 true
  contentType:"application/x-www-form-urlencoded", //指定 ContentType，默认为 application/x-www-
form-urlencoded
  encoding:"UTF-8",           //指定编码，默认为 UTF-8
  method:"post",              //请求方法，默认为 post
  parameters:" ",             //请求的参数，可以是一个序列化的字符串或者是一个对象
  postBody:null,              //post 方法提交的数据，如果没有被提供，则会将 parameters 的内容
  //当作提交的数据
  requestHeaders:null,         //请求的 HTTP 头信息，接受的数据格式可以为对象或者数组。如果是数组
  //时，则索引为奇数的元素为头名称，索引为偶数的元素为头值。如果是对象，则每一个名/值对应一个头信息
  onComplete:function(xmlhttprequest){...}, //在请求完成时被调用的回调函数，被调用时会将
  //当前 XMLHttpRequest 对象当作参数传入
  onException:function(xmlhttprequest,exception){...}, //在请求发生异常时被调用的回调函数，第一个参
  //数是当前请求对象，第二个参数是异常对象
  onFailure:function(xmlhttprequest){...}, //在请求完成，但 http 状态码不在 200~300 之间时被调用的
  //回调函数
  onInteractive:function(xmlhttprequest){...}, //在请求对象接收到部分返回的数据时被触发
  onLoaded:function(xmlhttprequest){...}, //在请求对象已经被设置，并且连接已经打开，准备发送数
  //据时触发
  onLoading:function(xmlhttprequest){...}, //在请求对象打开连接时被触发
  onSuccess:function(xmlhttprequest){...}, //当请求返回，而且 http 状态码在 200~300 之间时（不包含
  //300）时被触发
  onUninitialized:function(xmlhttprequest){...}, //在请求对象刚被创建时触发
  onXYZ:function(xmlhttprequest){...} //XYZ 表示 http 状态码，在返回的 http 状态码与指定的 http
  //状态码一致时被触发。例如 on200:function(){...}
});
```

下面来看一个使用 Ajax.Request 来完成 Ajax 请求的实例，代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Ajax.Request demo</title>
<script type="text/javascript" src="prototype1.5.js"></script>
<script type="text/javascript">
function viewSource()
{
```

```

/*
 * 实例化 Ajax.Request 类以发起一个 Ajax 请求
 */
new Ajax.Request('ajax_request.html',{
    onSuccess:function(xmlhttprequest)           //在请求成功时被调用
    {
        alert(xmlhttprequest.responseText);
    },
    onFailure:function()                         //在请求失败时被调用
    {
        alert("请求失败");
    }
});
}
</script>
</head>

<body>
<h1>Prototype.js</h1>
<h2>Ajax.Request Class demo</h2>
<input type="button" value="查看页面源代码" onclick="viewSource()" />
</body>
</html>

```

程序中使用 Ajax 请求本页，并使用对话框输出返回的文本内容，以达到查看页面 HTML 源代码的目的，其界面如图 14.18 所示。

单击“查看页面源代码”按钮，Ajax 请求被发出，在短暂的等待后请求成功返回并弹出对话框显示返回的数据，如图 14.19 所示。

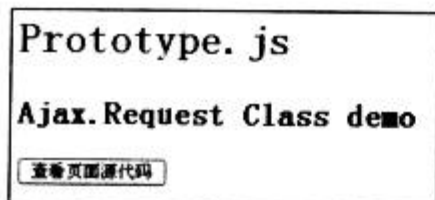


图 14.18 界面效果

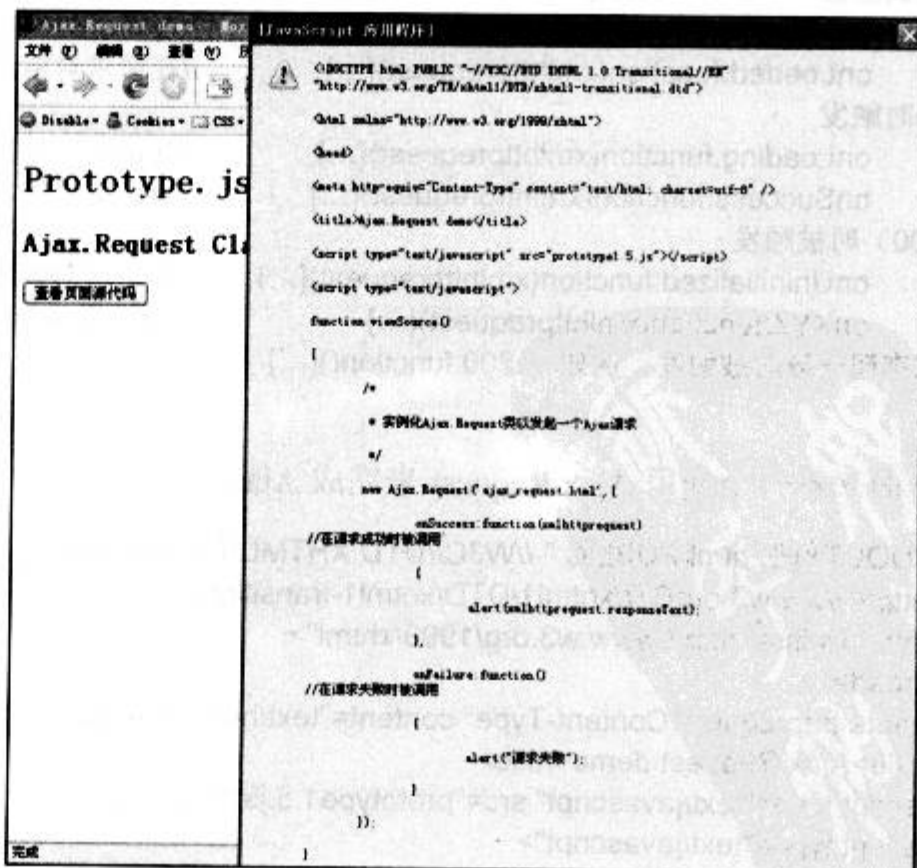


图 14.19 返回数据

### 14.4.2 用 Ajax.Updater 更新界面

Prototype.js 提供了 Ajax.Updater 类来完成根据 Ajax 请求返回的数据直接更新界面的功能。Ajax.Updater 类是基于 Ajax.Request 类的扩展，其语法如下所示。

```
new Ajax.Updater(container, url[, options]);
```

其中 container 是需要更新的元素对象，url 为请求的地址，options 是请求的相关参数，其结构与 Ajax.Request 的 options 参数一样。下面来看一个使用 Ajax.Updater 类的实例，该示例由 login.html 和 login.php 两个文件组成，login.html 代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Ajax.Updater demo</title>
<script type="text/javascript" src="prototype1.5.js"></script>
<script type="text/javascript">
function login()
{
    /*
     * 使用 Ajax.Updater 请求 login.php，使用#fmLogin 表单的序列化值作为参数，并更新#container 的内容
     */
    new Ajax.Updater($('container'), 'login.php', {
        parameters: $('fmLogin').serialize()
    });
}
</script>
<style type="text/css">
#container {
    margin:10px;
    padding:10px;
    border:1px solid #666;
    background-color:#eee;
    width:400px;
    height:200px;
}
label,input {
    display:block;
    margin-bottom:10px;
}
</style>
</head>

<body>
<div id="container">
```

```
<form name="fmLogin" id="fmLogin">
  <label for="txtUsername">昵称</label>
  <input type="text" name="txtUsername" id="txtUsername" />
  <input type="button" value="登录" onclick="login()" />
</form>
</div>
</body>
</html>
```

login.php 的代码如下所示。

```
<?
  header('Content-Type:text/html;charset=utf-8');
  echo '<h3>欢迎您, '.$_REQUEST['txtUsername'].'.</h3>';
?>
```

其界面效果如图 14.20 所示。在文本框中输入“Robin”，然后单击“登录”按钮，JavaScript 会使用 Prototype.js 提供的 Ajax.Updater，将文本框的值作为参数通过 Ajax 请求发送到 login.php 页面，并使用返回的数据更新界面，效果如图 14.21 所示。

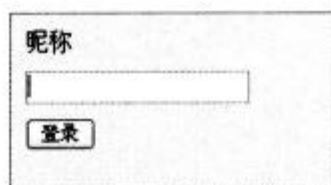


图 14.20 界面效果

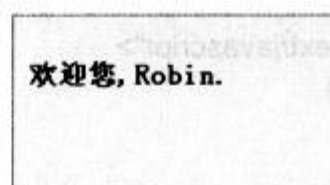


图 14.21 更新界面

### 14.4.3 用 Ajax.PeriodicalUpdater 定时更新界面

Prototype.js 提供了定时更新界面的工具：Ajax.PeriodicalUpdater，其语法与 Ajax.Updater 一致，如下所示。

```
new Ajax.PeriodicalUpdater(container, url[, options]);
```

所不同的是，这里的 options 对象会多出两个可选的属性。第一个是 frequency，表示更新频率，是一个以秒为单位的数字，例如 2、0.75 等。第二个是 decay，该参数是一个正整数值，其作用是用来监视每次请求返回的数据是否有变化，如果没有变化，则将下一次请求的时间间隔设置为 frequency\*decay 的值；如果有变化，则时间间隔又恢复原始的 frequency 指定的值。Ajax.PeriodicalUpdater 的用法可参考 14.4.2 小节中 Ajax.Updater 的示例。

### 14.4.4 使用 Form.request 方法无刷新提交表单

使用 \$ 方法获得表单元素的引用时，Prototype.js 会对表单元素进行扩展，增加一个名为 request 的方法，来实现将表单数据以 Ajax 方式提交的功能。request 方法是基于 Ajax.Request 实现的，其语法如下所示。





```

<body>
<div id="container">
  <form name="fmLogin" id="fmLogin" action="login.php" method="post">
    <label for="txtUsername">昵称</label>
    <input type="text" name="txtUsername" id="txtUsername" />
    <input type="button" value="登录" onclick="login()" />
  </form>
</div>
</body>
</html>

```

其效果与 14.4.2 小节中的示例一样。

## 14.5 基于 Prototype.js 的留言本程序

为了更好地体现出 Prototype.js 简化开发的特性，并更全面地演示 Prototype.js 的使用方法，本节将重写第 7 章留言本程序的前台脚本部分。由于在第 7 章中已经详细地说明了系统设计等要素，这里就不再重复，只给出实现代码，如下所示。

```

var url = 'php/service.php'; //设置请求的地址
/*
 * 留言本初始化程序
 */
function init()
{
  Event.observe($('btnSubmit'),'click',submitMessage); //注册按钮 btnSubmit 的 click 事件处理函数为
  submitMessage 函数
  readMessages(); //读取所有历史留言信息
}
/*
 * 读取留言功能模块
 */
function readMessages()
{
  /*
   * 设置 Ajax 请求的相关参数和回调函数
   */
  var options = {
    method:'GET',
    parameters:'action=getall',
    onLoading:function()
    {
      $('loadingMsg').show(); //请求过程中显示读取留言的进度提示
    },
    onSuccess:function(x)
    {
      var messages = x.responseText.evalJSON(); //将返回的 JSON 字符串转换成 JavaScript 对象
    }
  }
}

```

```

        messages.each(function(message)
        {
            displayMessage(message);           //显示留言
        });
        $('#loadingMsg').hide();               //留言信息读取成功后隐藏读取留言的进度提示
        $('#btnSubmit').disabled = false;      //留言信息读取成功后启用提交按钮
        doScroll();                            //将留言列表滚动到最低端
    },
    onFailure:function()
    {
        $('#loadingMsg').hide();               //请求失败时隐藏读取留言的进度提示
        alert('Request failure. ');           //弹出对话框提示请求失败
    }
}
new Ajax.Request(url,options);               //发送请求
}
/*
 * 显示留言功能模块
 * 函数 displayMessage 接受一个包含留言数据的对象作为参数，该对象有 name、message、postdate 属性分
别对应留言信息的数据字段
 * 通过解析该对象的数据，创建一条表示留言记录的 li 元素添加到页面的留言列表中
 */
function displayMessage(data)
{
    new Insertion.Bottom('msgList','<li><span>' + data.name + '</span><span>' + data.postdate +
    '</span><p>' + data.message + '</p></li>'); //使用 Prototype.js 提供的
    Insertion.Bottom 类将新留言的信息显示到留言列表的末尾
}
/*
 * 发送留言功能模块
 */
function submitMessage()
{
    /*
     * 获取访客填写的姓名和留言内容
     */
    var name = $('#txtName').unescapeHTML();
    var message = $('#txtContent').unescapeHTML();
    /*
     * 设置 Ajax 请求的相关参数和回调函数
     */
    var options = {
        method:'GET',
        parameters:'action=addnew&name=' + name + '&message=' + message,
        onLoading:function()
        {
            $('#submitMsg').show();           //显示发送留言的进度提示
            $('#btnSubmit').disabled = true;  //在请求过程中禁用提交按钮
        }
    },

```

```

    onSuccess:function(x)
    {
        var resBack = x.responseText;
        /*
        * 如果请求返回的字符串数据是 ok, 则表示留言提交成功。这时候调用显示留言功能模块将刚才
        提交的留言立即显示在留言列表的末端
        */
        if(resBack == 'ok')
        {
            var date = new Date();
            var postDate = date.getFullYear() + '-' + (date.getMonth() + 1) + '-' + date.getDate();
            var msg = {
                name:name,
                message:message,
                postdate:postDate
            }
            displayMessage(msg);
            doScroll();
        }
        /*
        * 否则弹出对话框提示提交失败
        */
        else
        {
            alert('Submit failure.');
```

```

            $('submitMsg').hide();

```

```

            //隐藏提交留言的进度提示

```

```

            $('btnSubmit').disabled = false;

```

```

            //启用提交按钮

```

```

        },

```

```

        onFailure:function()

```

```

        {

```

```

            $('submitMsg').hide();

```

```

            //隐藏提交留言的进度提示

```

```

            $('btnSubmit').disabled = true;

```

```

            //启用提交按钮

```

```

            alert('Request failure.');
```

```

            //请求失败时弹出对话框提示请求失败

```

```

        }

```

```

    }
    new Ajax.Request(url,options);

```

```

    //发送请求

```

```

}

```

```

/*

```

```

* 将留言列表滚动到最下端

```

```

*/

```

```

function doScroll()

```

```

{

```

```

    var height = $('msgList').offsetHeight;

```

```

    var totalHeight = $('msgList').scrollHeight;

```

```

    $('msgList').scrollTop = totalHeight - height;

```

```

}

```

```

Event.observe(window,'load',init);

```

```

//页面加载完时初始化留言本程序

```

## 14.6 小 结

本章向读者介绍了优秀的 JavaScript 和 Ajax 基础开发框架：Prototype.js，其中包括一些常用的方法，还有基于 Prototype.js 的类和继承，以及 Prototype.js 对事件处理的兼容性封装，Prototype.js 对 Ajax 的支持。最后使用 Prototype.js 重写了第 7 章中留言本程序的前台脚本的实现。由于篇幅有限，本章对 Prototype.js 的介绍就到这里，更多内容可参考本书的附录 C：Prototype 速查手册。

## 14.6 小结

本章向读者介绍了优秀的 JavaScript 和 Ajax 基础开发框架：Prototype.js，其中包括一些常用的方法，还有基于 Prototype.js 的类和继承，以及 Prototype.js 对事件处理封装封装，Prototype.js 对 Ajax 的支持，最后使用 Prototype.js 重写了第 1 章中留言本程序的前台脚本的实现。由于篇幅有限，本章对 Prototype.js 的介绍到此为止，更多内容可参考本书附带的《C: Prototype 速查手册》。



## 15.1 REST 架构模式

REST 是英文 Representational State Transfer 的缩写，意思是“表述性状态转移”，是由 Roy Thomas Fielding 博士在其博士论文《Architectural Styles and the Design of Network-based Software Architectures》中提出的一种基于网络的软件架构模式。REST 软件架构是当今世界上最成功的互联网超媒体分布式系统架构模式，它让人们真正理解了 HTTP 协议的本来面貌。当前它正成为网络服务的主流技术，并改变着网络软件开发的思维方式。

REST 软件架构是一个抽象的概念，是一种为了实现基于互联网的超媒体分布式系统的行为指南，使用任何技术都可以实现这种理念。实现 REST 软件架构，最著名的就是 HTTP 协议。通常将 REST 也写作为 REST/HTTP，在实际中往往将 REST 理解为基于 HTTP 的 REST 软件架构。

HTTP 是互联网上应用最广泛的计算机协议。HTTP 并不是一个简单的数据传输协议，而是具有丰富内涵的网络软件协议。它不仅仅能够对互联网的资源进行唯一性定义，并且能够包含对资源的处理信息，这也是 REST 软件架构中最重要的两个理念。

REST 架构模式可以降低开发的复杂度、提高系统的可伸缩性，它提出了一些设计概念和准则。

- ☐ 网络上的所有事物都被抽象为资源（Resource）。
- ☐ 每个资源对应一个唯一的资源标识（URI）。
- ☐ 通过通用的连接器接口对资源进行操作（HTTP）。
- ☐ 对资源的各种操作不会改变资源的标识。
- ☐ 所有的操作都是无状态的（Stateless）。

REST 之所以能够简化开发，是因为其引入的架构约束。HTTP 协议实际上除了 GET 和 POST 两种方法，还有 PUT 和 DELETE。使用 HTTP 作为通用连接器接口，HTTP 则把对每一个 URL 的操作限制在了这 4 个方法之内，实际上对应的就是 CURD（Create、Update、Read、Delete）操作。REST 之所以能够提高系统的可伸缩性，是因为它强制所有操作都是无状态的，这样就没有了上下文的约束，可以很方便地做到分布式、集群。另一方面，REST 强调将所有事物都抽象为资源，每个资源对应一个唯一的资源标识，对资源的各种操作都不会改变这个资源的标识，使得资源的数据可以被有效的缓存，从而减少或者完全避免一些操作，提高系统的负载能力。REST 对系统性能的提升来自于它对客户端和服务端任务的有效分配。服务端负责提供资源和操作资源的服务，而客户端则根据资源中的数据和表现来自己完成呈现，这样就减少了服务器的性能开销。

## 15.2 Web 应用程序的发展史

早期的大部分 Web 应用程序实际上都是遵守 REST 准则的，但是随着融入式 Web 程序的日益普及，Web 应用程序架构逐渐开始背离 REST 准则，这使得当前的 Web 应用程序丧失了 REST 架构所具有的大量优点。本节将简单介绍 Web 应用程序的发展史以分析这个变更过程，来让读者了解到当前融入式服务器端 Web 应用程序的一些问题。

### 15.2.1 提供静态文档的 Web 站点

Web 最初是作为研究人员共享文档和在文档之间创建简单的链接以加速知识和思想传播的一种手段，其内容就是一些静态的 HTML 文档，以及很多到其他静态文档的链接，如图 15.1 所示。

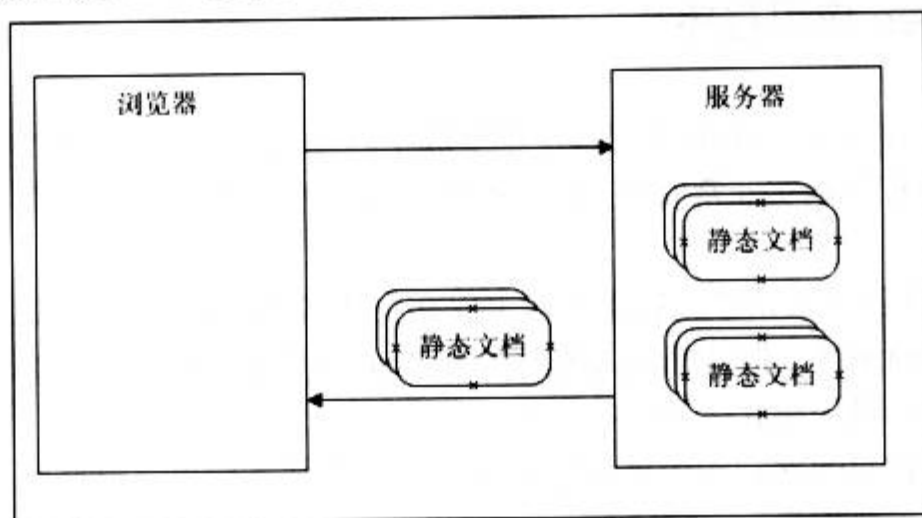


图 15.1 提供静态文档的 Web 站点

### 15.2.2 早期的动态 Web 应用程序

由于 URI 将资源的标识和资源的存储机制区分开来，这样使得 Web 开发人员可以创建一些程序，通过检查 URI 语法，来动态生成文档，将预定的界面元素和动态检索得到的数据合并在一起。尽管这些文档是动态生成的，但是它们的缓存特性与静态文档完全相同，如图 15.2 所示。

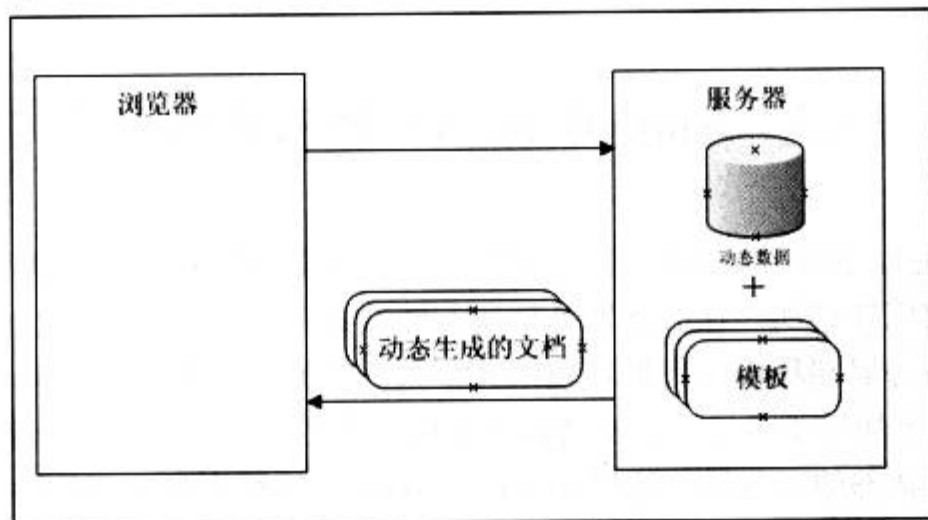


图 15.2 早期的动态 Web 程序

一个早期的动态 Web 应用程序的例子是一个电话簿程序。用户在表单中输入某人的名字，例如 Robin Chen，表单根据用户输入的名字生成一个 URI，例如 `http://www.example.com/telephonebook/Robin+Chen`。然后浏览器从服务器上请求这个 URI 的资源，服务器会检查这个 URI 并使用相应的数据来生成一个 Web 页面。最后服务器将这个页面回发到浏览器上并呈现给用户。

这种交互模式的一个重要特性，就是它是幂等 (idempotent) 的，也就是说除非底层的数据资源发

生了变化（例如 Robin Chen 修改了自己的电话号码），否则同一请求的结果总是相同的。这使得浏览器或者代理服务器都可以对该资源进行缓存，只要底层的数据资源没有发生变化，就可以从本地的缓存中检索数据，而不再需要从服务器上检索。这种方法提高了响应速度，并增加了系统的整体效率和可伸缩性。

### 15.2.3 融入式 Web 应用程序

Web 发展到今天，大部分的 Web 应用程序都变成了高度融入的、提供个性化内容的富应用程序模型。融入式的 Web 应用程序确实非常有用，但服务器端的融入式 Web 应用程序风格从根本上来讲是不符合 REST 架构模型的。

REST 的无状态操作约束准则禁止在服务器上保存会话状态，符合这一约束进行设计可以提供系统的可见性、可靠性和可伸缩性。但是融入式的服务端 Web 应用程序希望能够为单个用户提供大量个性化的内容，于是要在两种设计模式之间进行选择。

第一种设计模式是在每次客户端的请求中都包含大量的状态信息，这些状态信息完整地保留了上下文的内容，服务器是无状态的。

第二种设计模式表面看起来比较简单，而且当前大部分 Web 应用程序都采取这种模式，就是客户端在请求中只发送一个简单的用户标识，并在服务端为这个标识关联一个“会话”，在“会话”中保存用户的状态信息。第二种模式虽然实施起来更容易，但是它直接破坏了 REST 无状态操作的准则。

融入式服务端 Web 应用程序的另一个严重问题在于，它破坏了 REST 提供的高效的缓存机制。融入式服务端 Web 应用程序可以被看作一个活动实体，它会根据部分或者所有用户新的输入内容，以及新的后台数据而不断发生变化。由于服务器必须根据多个用户与应用程序的交互来生成界面，因此实际上无法两次生成相同的文档。这样，浏览器或者代理服务器就无法对服务器资源进行缓存。

## 15.3 Ajax 为 REST 带来新的契机

随着使用 Web 应用程序的用户越来越多，采用融入式服务端风格的 Web 应用程序，其系统提供服务所需要的资源也会相应地增加。让服务器处理所有的状态和相应的请求，这就需要容量更大的服务器或者集群服务器（服务器端环境在集群环境中并不太适用）。但如果将处理分布到客户端，那么每增加一个用户，实际上就相当于多了一台支持部分新的负载的电脑。如果将会话状态分布到客户端中，那么就可以得到一个无状态的服务器，这样就会使得 Web 应用程序具有可伸缩性。

既然有好的解决方案，为什么没有被广为采用呢？这是因为在 Ajax 出现以前，这种设计模式很难实现。每次用户访问一个新的页面时，应用程序状态就会被销毁。用户访问一个页面时，都要下载一组包含内容的文件，以及影响内容外观的样式，整个文档会被浏览器当作一个复杂的对象树。当用户访问一个新页面时，浏览器就会销毁当前的对象树，并为下一个页面创建一个新的对象树。

Ajax 的出现为 REST 带来了新的契机。Ajax 风格的应用程序可以让页面无刷新地呈现新的数据，这样页面的对象树就可以得到保留，使得客户端可以用来保存用户状态等应用程序资源。由于应用程序资源和数据资源的绑定转移到了客户端，因此这些应用程序可以享受世界上最好的东西：融入式 Web



应用程序中动态、个性化的用户体验，以及遵守 REST 准则的应用程序中简单、可伸缩的架构。

### 15.3.1 缓存 Ajax 程序引擎

这里将用于实现 Ajax 应用的 JavaScript 程序称之为 Ajax 程序引擎。一个设计良好的 Ajax 程序引擎可以不包含任何业务数据和个性化内容，而只是对业务逻辑的抽象。在典型的 Web 环境中，应用程序资源可能很长时间才会变化一次，所以负责隔离应用程序资源和数据资源的 Ajax 程序引擎是高度可缓存的。同时由于 Ajax 程序引擎只是一个或者几个文件，这样它也是可以被代理缓存的。经过良好设计的 Ajax 程序引擎符合 REST 架构准则，与服务端 Web 应用程序相比，它拥有显著的可伸缩性优势。

### 15.3.2 缓存 Ajax 数据

在 Ajax 风格的 Web 应用程序中，由于应用程序逻辑和状态都在客户端驻留并执行，这使得服务器不需要返回带有界面样式的混合内容，而只需要返回业务数据即可。同时，Ajax 风格的 Web 应用程序不需要刷新页面就可以获取新的数据，并且可以将不同类型的数据获取分开进行。这样就可以将特定于用户的个性化数据和非特定于用户的数据进行分开处理。非特定于用户的数据具有高度的可伸缩性，可以被高效地缓存。这样就不会因为要取得特定于用户的个性化数据被迫刷新页面，同时也重新获取了非特定于用户的数据，浪费了服务器的性能开销，从而有利于提高服务器的整体负载能力。

## 15.4 小 结

本章对 REST 架构进行了简要的介绍，包括其概念、设计准则以及其带来的好处。然后通过回顾 Web 应用程序的发展史，来分析当前融入式服务端 Web 应用程序因为违背 REST 架构准则而带来的问题。最后分析了 Ajax 技术为 REST 带来的新的契机，Ajax 风格的应用程序同时具备了 REST 架构的优势和融入式的用户体验。为了避免晦涩难懂，本章对相关概念的介绍并不深入，只是让读者对 REST/Ajax 有一个初步的认识，提供一个日后研究的方向。

应用中，使用Ajax技术，可以大大减少网络流量，提高用户体验。在Web应用中，使用Ajax技术，可以实现局部刷新，而不需要重新加载整个页面。这不仅可以提高页面的响应速度，还可以减少服务器的负担。此外，使用Ajax技术，还可以实现更复杂的交互功能，如拖拽、缩放等。总之，Ajax技术是现代Web应用开发中不可或缺的一部分。

## 12.3.1 Ajax 技术简介

这里我们主要介绍 Ajax 技术。Ajax 是 Asynchronous JavaScript and XML 的缩写，它是一种用于创建交互式网页的技术。它通过 XMLHttpRequest 对象，可以在后台向服务器发送请求，并接收返回的数据。这样，用户可以在不重新加载整个页面的情况下，更新部分数据。Ajax 技术的出现，极大地提高了网页的交互性和响应速度，成为现代 Web 应用开发的重要技术之一。

## 12.3.2 Ajax 数据

在 Ajax 应用中，数据通常以 JSON 格式返回。JSON 是一种轻量级的数据交换格式，易于阅读和编写。它使用文本表示结构化数据，格式简单且易于理解。在 Ajax 应用中，客户端通过 XMLHttpRequest 对象向服务器发送请求，服务器返回 JSON 数据。客户端然后解析这些数据，并更新页面内容。JSON 格式的数据通常包含键值对，键是字符串，值是各种数据类型，如字符串、数字、布尔值、数组等。这种格式使得数据交换更加灵活和高效。

## 12.4 小结

本章主要介绍了 REST 架构和 Ajax 技术。REST 是一种轻量级的架构风格，强调资源的统一和操作的原子性。它使用标准的 HTTP 方法（GET、POST、PUT、DELETE）来操作资源。Ajax 是一种用于创建交互式网页的技术，通过 XMLHttpRequest 对象实现异步通信。本章还介绍了如何使用 Ajax 技术从服务器获取数据，并更新页面内容。通过本章的学习，读者应该对 REST 架构和 Ajax 技术有了更深入的了解，并能够应用到实际开发中。



# 第16章

## Ajax 的缺陷及补救

- » 搜索引擎的收录问题
- » 前进和后退的问题
- » 小结

任何事物都有其两面性，Ajax 也不例外。Ajax 可以实现激动人心的应用，但同样其本身也存在一些缺陷。Ajax 主要的缺陷就是对搜索引擎的不友好，以及破坏了浏览器前进和后退的功能。这些缺陷从 Ajax 逐渐被人们所知开始就饱受诟病的。本章将向读者分析这些缺陷产生的原因，以及讨论解决办法。

## 16.1 搜索引擎的收录问题

面对搜索引擎友好的争议一直伴随着 Ajax 的成长，直至今天，仍有大量的用户因为 Ajax 对搜索引擎的不友好而放弃使用 Ajax。本节将向读者分析为什么说 Ajax 是对搜索引擎不友好的，并且讨论一些解决的办法。

### 16.1.1 问题产生的原因

首先来看搜索引擎的基本工作原理。搜索引擎依靠一种名为“网络爬虫”的程序，在万维网上进行爬行并将所访问的页面内容存储到搜索引擎的数据库中，以供使用者检索信息。“网络爬虫”通过识别页面上的链接到达网站的其他页面以及其他站点。页面与页面之间的相互链接使得整个网络成了一个蜘蛛网的结构，“网络爬虫”可以通过这个“蜘蛛网”到达任何地方。一个传统的基于超链接的 Web 程序如图 16.1 所示。

再来看 Ajax 风格的程序。Ajax 风格的 Web 程序，通过 JavaScript 和 XMLHttpRequest 从 WebService 获取数据并更新界面，并不依赖于传统的超链接，其结构如图 16.2 所示。

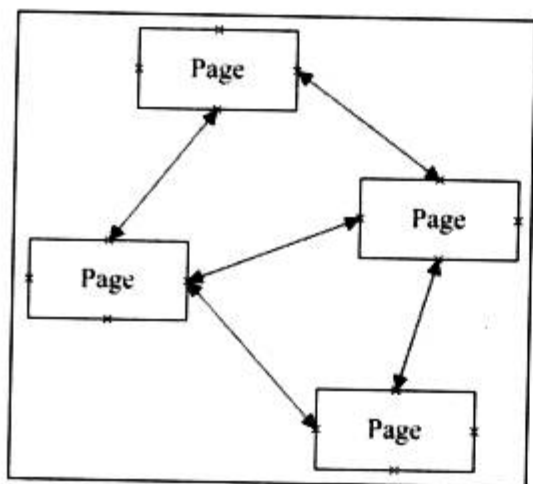


图 16.1 传统 Web 程序

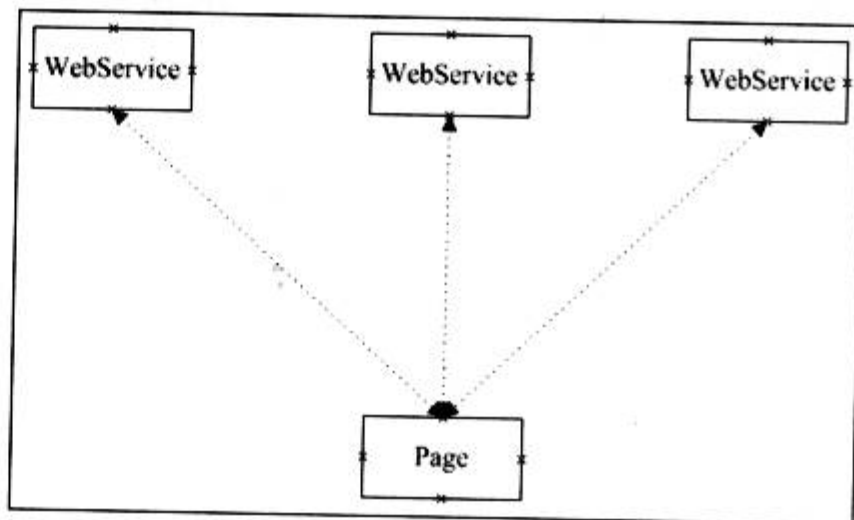


图 16.2 Ajax 风格的 Web 程序

这时候，问题就出现了。搜索引擎的“网络爬虫”只能识别 HTML 数据而不会执行页面上的 JavaScript，所以通过 XMLHttpRequest 完成的数据交互对于“网络爬虫”来说更加是透明的，仿佛从未存在过。搜索引擎只能识别 Ajax 程序的初始页面，并且不能通过该页面获取更多信息，从而使得使用 Ajax 技术的站点内所包含的数据不能被收录，不能被最终用户检索到。这对于信息展示类的网站是灾难性的。试想一下，如果某个公司的产品信息不能被搜索引擎收录，并被潜在的客户检索到，将会带来巨大的损失，特别是那些主要依靠互联网进行推广和销售的企业。

### 16.1.2 解决办法

在希望使用 Ajax 技术改善用户体验，而又要保持对搜索引擎友好的时候，可以按照搜索引擎的工

作原理，让页面没有 JavaScript 时依然可以正常访问到所有数据，这同样对没有 JavaScript 支持的浏览器用户也是友好的。在搜索引擎的“爬虫”爬行到站点时，可以找到充足的链接来对网站的页面进行索引，而普通用户访问站点时，则通过 JavaScript 程序来达到完全不一样的运行效果，如图 16.3 所示。

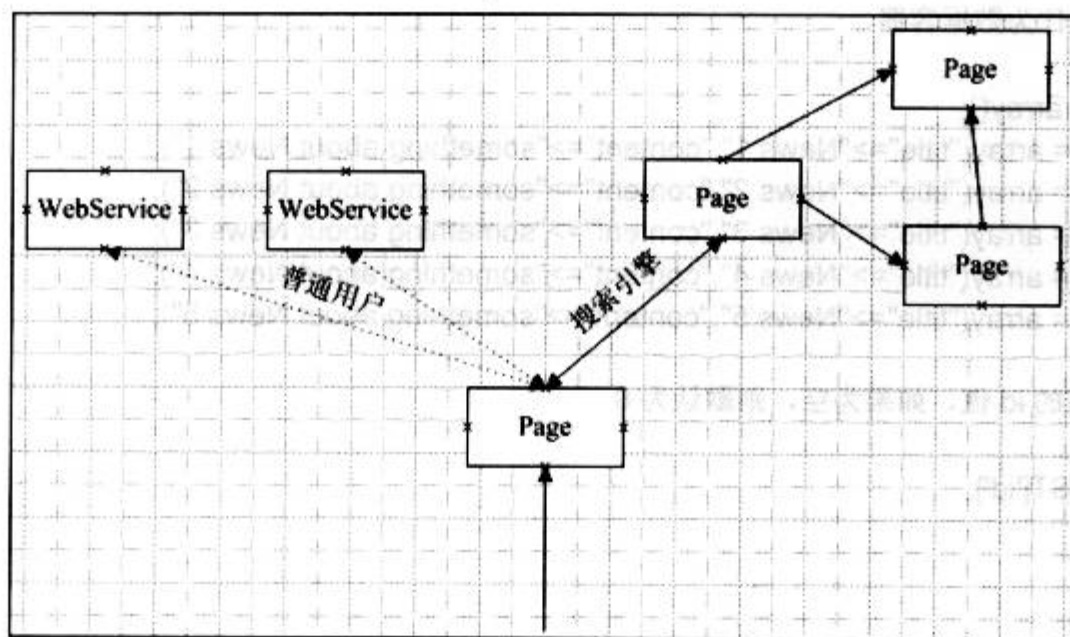


图 16.3 解决方案

接下来，会通过一个简单的新闻列表程序来向读者演示如何实现这种架构。这个示例程序的界面示意图如图 16.4 所示。

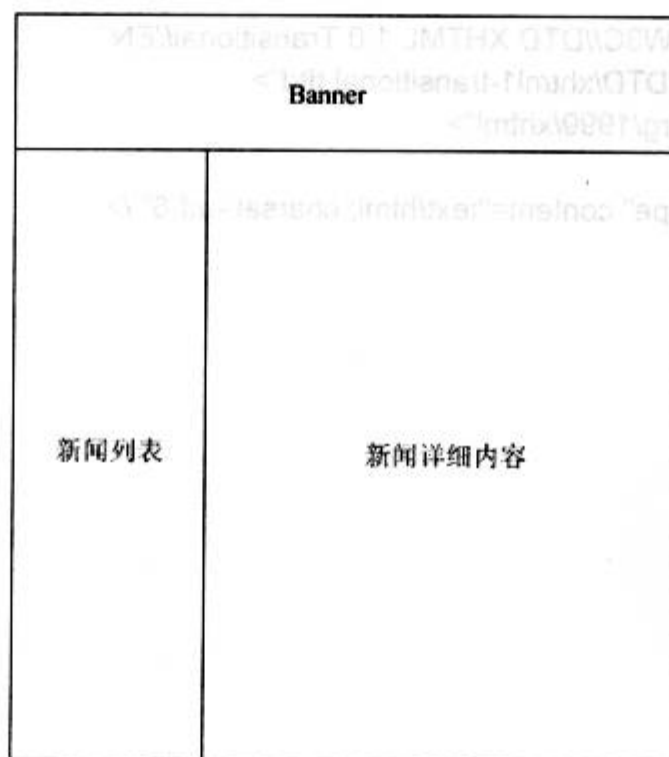


图 16.4 界面示意图

界面左边是一个新闻列表，单击某条新闻的链接，则会在界面右边的区域中显示新闻的详细内容。第一步，首先使用传统的做法来实现这个新闻列表程序。为了更简单地完成示例，这里就不使用任何数据库了，而是在服务端使用一个数组来保存少量的新闻数据，然后通过传递来的参数来取对应的新

闻数据。将这个新闻列表的程序页面命名为 news.php，其实现代码如下所示。

```
<?
/*
 * 使用一个数组作为数据容器
 */
$newsStorage = array();
$newsStorage[] = array("title"=>"News 1","content"=>"something about News 1");
$newsStorage[] = array("title"=>"News 2","content"=>"something about News 2");
$newsStorage[] = array("title"=>"News 3","content"=>"something about News 3");
$newsStorage[] = array("title"=>"News 4","content"=>"something about News 4");
$newsStorage[] = array("title"=>"News 5","content"=>"something about News 5");
/*
 * 取得传递过来的 id 值，如果为空，则默认为 0
 */
$id = $_REQUEST['id'];
if(empty($id))
{
    $id = 0;
}
/*
 * 取得索引为指定 id 的新闻数据
 */
$news = $newsStorage[$id];
?>
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>News</title>
<style type="text/css">
*{
    margin:0;
    padding:0;
}
body {
    font-size:11px;
    font-family:Tahoma;
}
h1 {
    height:80px;
    color:#ccc;
    background-color:#333;
    line-height:80px;
    text-align:center;
    font-size:24px;
}
h2 {
    font-size:18px;
```



```

        text-align:center;
        border-bottom:1px solid #ccc;
        margin-bottom:5px;
        height:30px;
        line-height:30px;
    }
    #list {
        float:left;
        clear:left;
        width:20%;
        text-align:center;
    }
    #list li {
        list-style:none;
        height:20px;
        margin-bottom:1px;
    }
    #list li a{
        display:block;
        background-color:#ccc;
        line-height:20px;
        color:#333;
        font-weight:bold;
        text-decoration:none;
    }
    #list li a:hover {
        background-color:#666;
        color:#eee;
    }
    #detail {
        float:left;
        margin-left:10px;
        width:79%;
    }
    #newsContent {
        text-indent:2em;
    }
</style>
</head>

<body>
<h1>News List System</h1>
<div id="main">
    <ul id="list">
        <? for($i = 0; $i < sizeof($newsStorage); $i ++)>
        {
            echo '<li><a href="news.php?id='.$i.'"
title="'.$newsStorage[$i]['title'].'">'.$newsStorage[$i]['title'].'</a></li>';
        }
    ?>

```



```

</ul>
<div id="detail">
    <h2 id="newsTitle"><?=$news["title"]?></h2>
    <p id="newsContent"><?=$news["content"]?></p>
</div>
</div>
</body>
</html>

```

将页面程序放在 Apache 的虚拟目录中，并通过浏览器访问，其效果如图 16.5 所示。

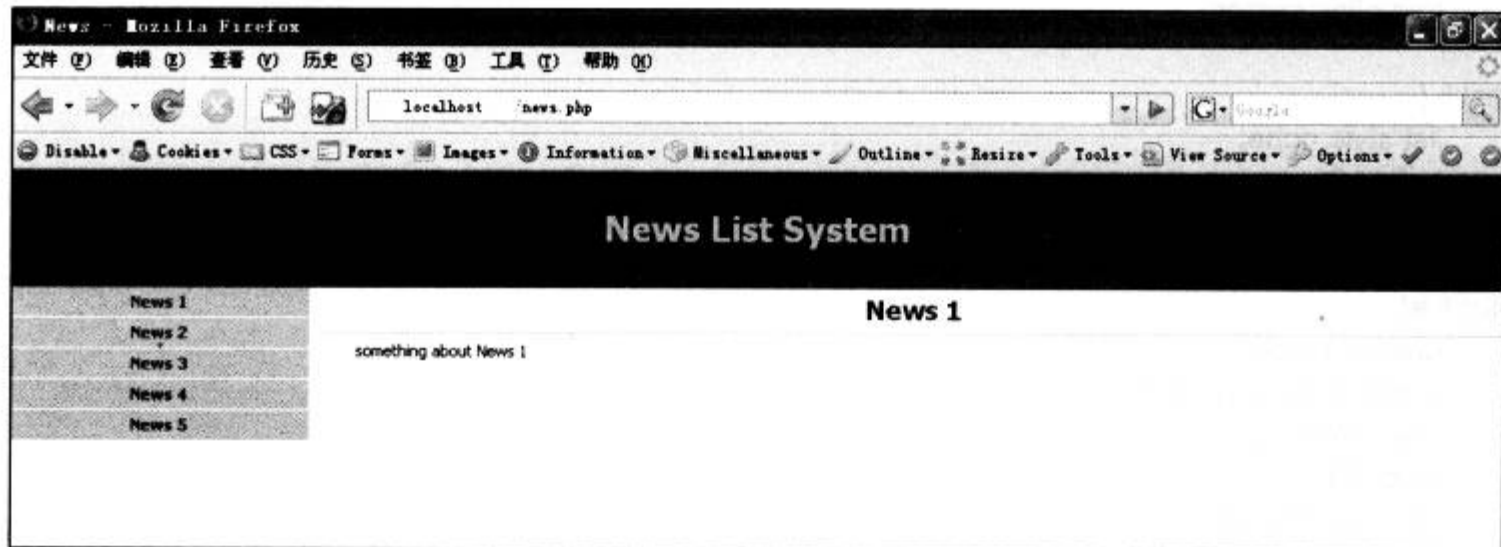


图 16.5 新闻列表程序

当前默认显示的是第一条新闻。单击左边列表中的链接，页面会刷新，并显示相应的新闻信息，如图 16.6 所示。

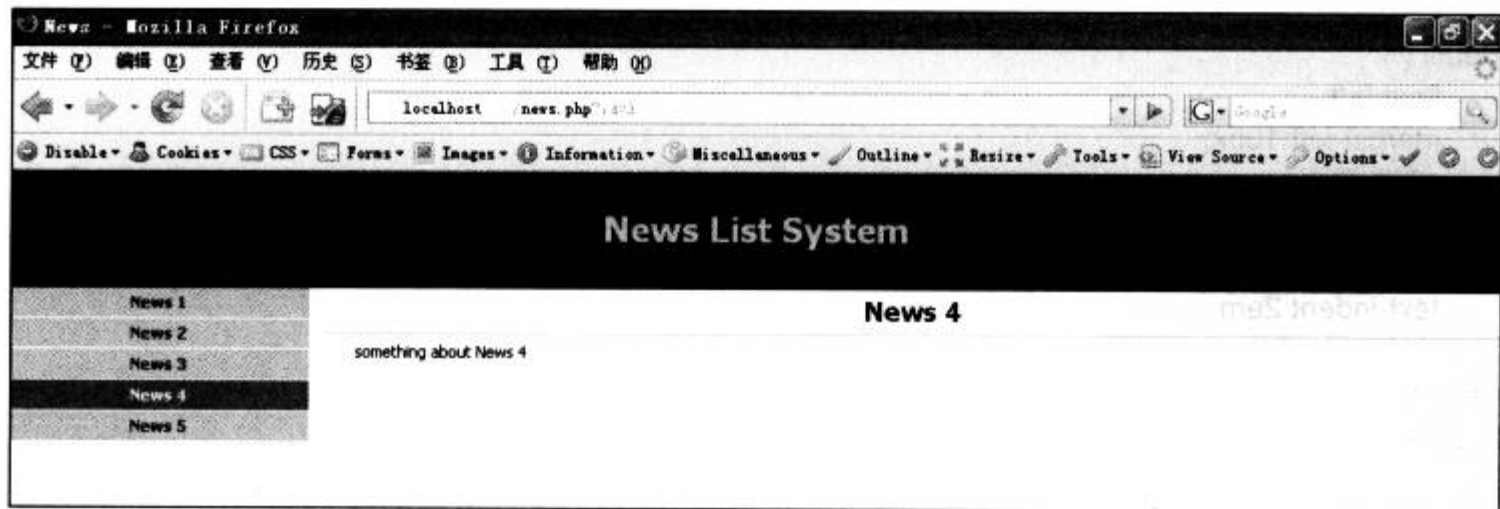


图 16.6 单击链接

这是一个传统的 Web 程序，其对搜索引擎是友好的，因为它提供了足够多的链接来让搜索引擎找到每一个新闻的数据页面。现在来对它加以改进，让它在对搜索引擎保持友好的基础上，对普通用户提供 Ajax 风格的用户体现，代码如下所示。

```

<?
/*
* 使用一个数组作为数据容器

```

```

*/
$newsStorage = array();
$newsStorage[] = array("title"=>"News 1","content"=>"something about News 1");
$newsStorage[] = array("title"=>"News 2","content"=>"something about News 2");
$newsStorage[] = array("title"=>"News 3","content"=>"something about News 3");
$newsStorage[] = array("title"=>"News 4","content"=>"something about News 4");
$newsStorage[] = array("title"=>"News 5","content"=>"something about News 5");
/*
 * 取得传递过来的 id 值, 如果为空, 则默认为 0
 */
$id = $_REQUEST['id'];
if(empty($id))
{
    $id = 0;
}
/*
 * 取得索引为指定 id 的新闻数据
 */
$news = $newsStorage[$id];
/*
 * 如果请求是由 xmlhttprequest 发起, 则只输出新闻数据的 json 表示
 */
if($_REQUEST['requestby'] == 'xmlhttprequest')
{
    echo json_encode($news);
    exit;
}
?>
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>News</title>
<script type="text/javascript" src="prototype1.5.js"></script>
<script type="text/javascript">
/*
 * 在 window 的 load 事件中初始化程序
 */
Event.observe(window,'load',function()
{
    /*
     * 取得所有#list 下的 a 元素
     */
    var links = $$("#list a");
    /*
     * 遍历所有取得的 a 元素并注册 click 事件处理函数
     */
    links.each(function(link)

```

```

{
    /*
    * 注册 click 事件处理函数，在处理函数中发起 Ajax 请求并更新界面
    */
    Event.observe(link,'click',function(evt)
    {
        /*
        * 根据 a 元素的 href 值发起请求并附加一个 requestby 参数，其值为 xmlhttprequest，表示当前
        请求由 xmlhttprequest 发起
        */
        new Ajax.Request(link.href,{
            method:'GET',
            parameters:'requestby=xmlhttprequest',
            /*
            * 在请求成功时处理返回的 JSON 数据并更新界面
            */
            onSuccess:function(x)
            {
                var news = eval('(' + x.responseText + ')');
                $('newsTitle').innerHTML = news.title;
                $('newsContent').innerHTML = news.content;
            }
        });
        /*
        * 阻止浏览器的默认行为以避免页面的跳转
        */
        Event.stop(evt);
    });
});
</script>
<style type="text/css">
*{
    margin:0;
    padding:0;
}
body {
    font-size:11px;
    font-family:Tahoma;
}
h1 {
    height:80px;
    color:#ccc;
    background-color:#333;
    line-height:80px;
    text-align:center;
    font-size:24px;
}
h2 {

```

```

    font-size:18px;
    text-align:center;
    border-bottom:1px solid #ccc;
    margin-bottom:5px;
    height:30px;
    line-height:30px;
}
#list {
    float:left;
    clear:left;
    width:20%;
    text-align:center;
}
#list li {
    list-style:none;
    height:20px;
    margin-bottom:1px;
}
#list li a{
    display:block;
    background-color:#ccc;
    line-height:20px;
    color:#333;
    font-weight:bold;
    text-decoration:none;
}
#list li a:hover {
    background-color:#666;
    color:#eee;
}
#detail {
    float:left;
    margin-left:10px;
    width:79%;
}
#newsContent {
    text-indent:2em;
}
</style>
</head>

<body>
<h1>News List System</h1>
<div id="main">
    <ul id="list">
        <? for($i = 0; $i < sizeof($newsStorage); $i ++)>
        {
            echo '<li><a href="news.php?id='.$i.'"
title="'.$newsStorage[$i]['title'].'">'.$newsStorage[$i]['title'].'</a></li>';
        }
    </ul>

```

```
?>
</ul>
<div id="detail">
    <h2 id="newsTitle"><?=$news["title"]?></h2>
    <p id="newsContent"><?=$news["content"]?></p>
</div>
</div>
</body>
</html>
```

(1) 首先修改了后台的 PHP 程序, 当请求中包含的 `requestby` 参数值为 `xmlhttprequest` 时, 只输出相应新闻数据的 JSON 表示。

(2) 然后, 为前台页面增加了 JavaScript 程序。这里引入了 `Prototype.js`, 在页面加载完的事件中为列表的每个 `a` 标签的 `click` 事件都注册了事件处理函数。

(3) 在处理函数中, 根据当前的 `a` 标签的 `href` 值进行 Ajax 请求, 并发送 `requestby` 参数, 其值为 `xmlhttprequest`, 这样返回的就只是 JSON 数据而不是页面的 HTML 代码, 然后利用 JSON 数据更新新闻详细内容区域的显示。再次访问该页, 如图 16.7 所示。

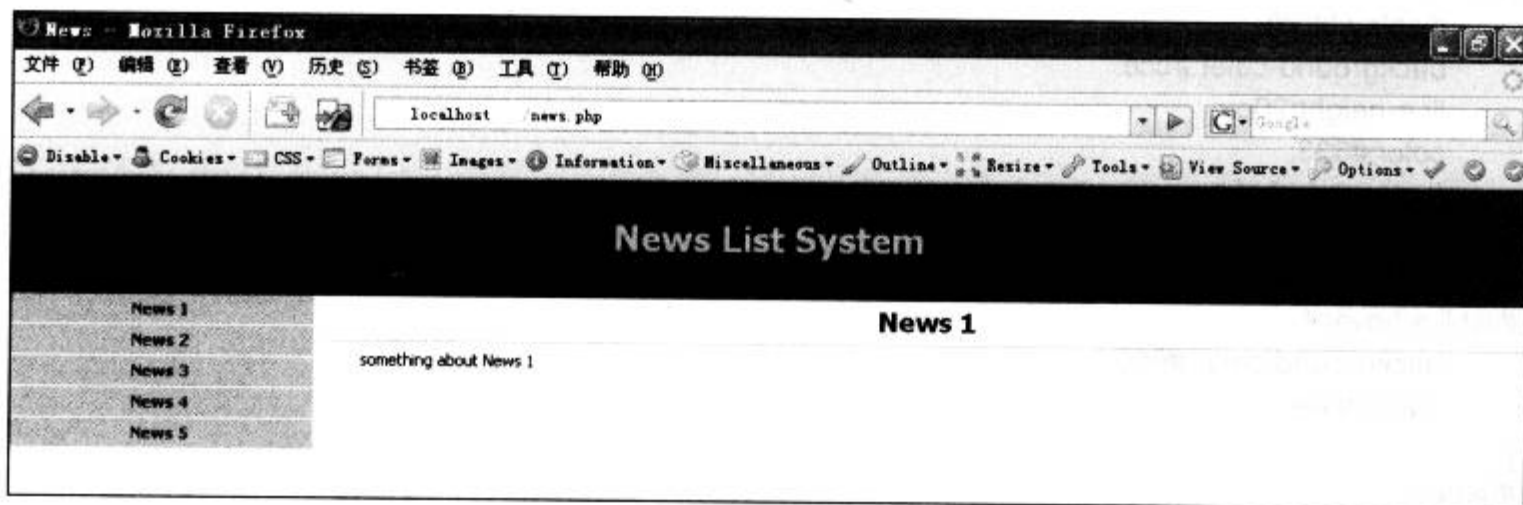


图 16.7 新闻列表

单击页面左边列表中的链接, 这时页面没有发生跳转, 但是右边区域的内容已经更新了, 如图 16.8 所示。

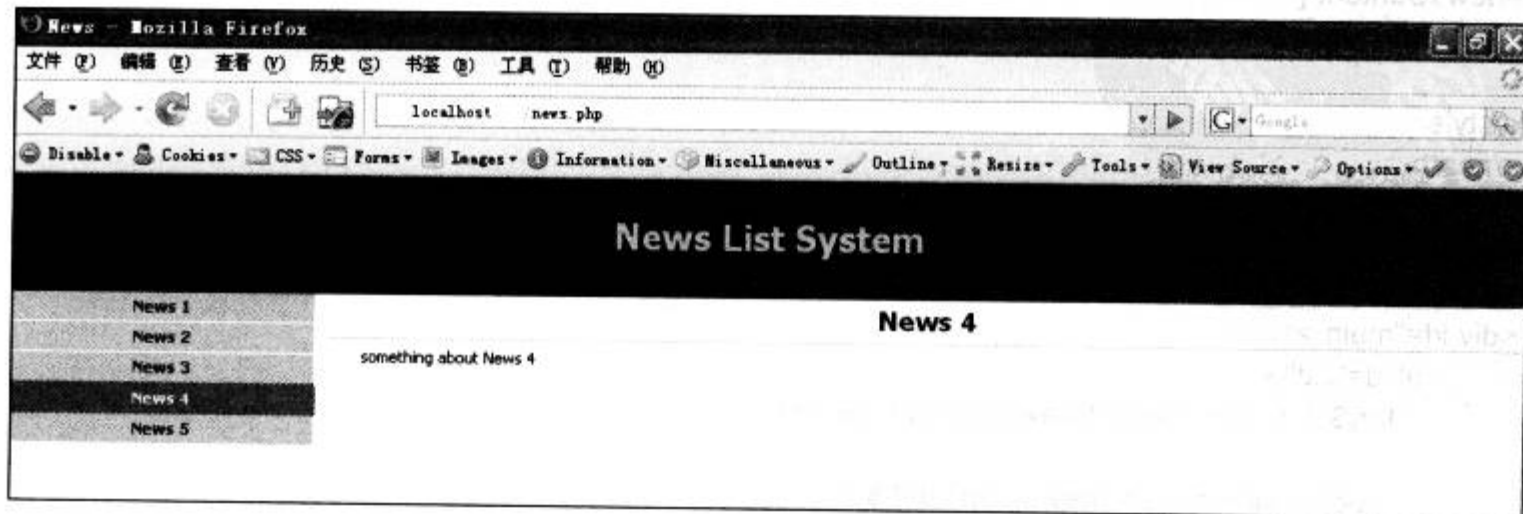


图 16.8 没有页面跳转



这样，就同时保持了对搜索引擎的友好和良好的用户体验。

## 16.2 前进和后退的问题

Ajax 另外一个饱受诟病的缺陷，就是其破坏了浏览器的前进和后退的功能，使得用户的操作习惯得不到延续，同时也给用户造成了不便。本节将向读者分析问题产生的原因，并探讨解决办法。

### 16.2.1 问题产生的原因

传统的基于超链接的 Web 应用程序，浏览器可以记录每次页面的跳转，然后提供浏览历史(history)前进和后退的功能，这样使得用户可以自由地在访问的页面链中切换当前页面并查找需要的信息。传统的浏览流程如图 16.9 所示。

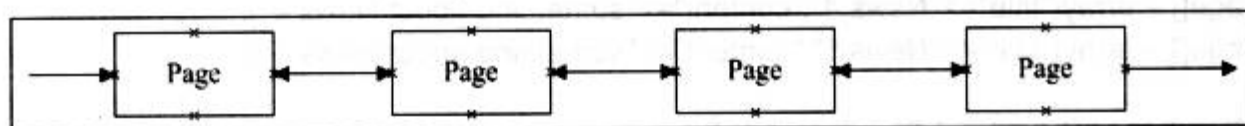


图 16.9 传统流程

但是 Ajax 风格的应用程序破坏了这一机制。由于 Ajax 风格的应用程序是没有页面刷新的，所有的数据交互都是依靠 XMLHttpRequest 来完成，然后通过 JavaScript 程序更新界面，这样浏览器就无法记录用户的操作历史并提供有效的前进和后退的功能。Ajax 下的浏览流程如图 16.10 所示。

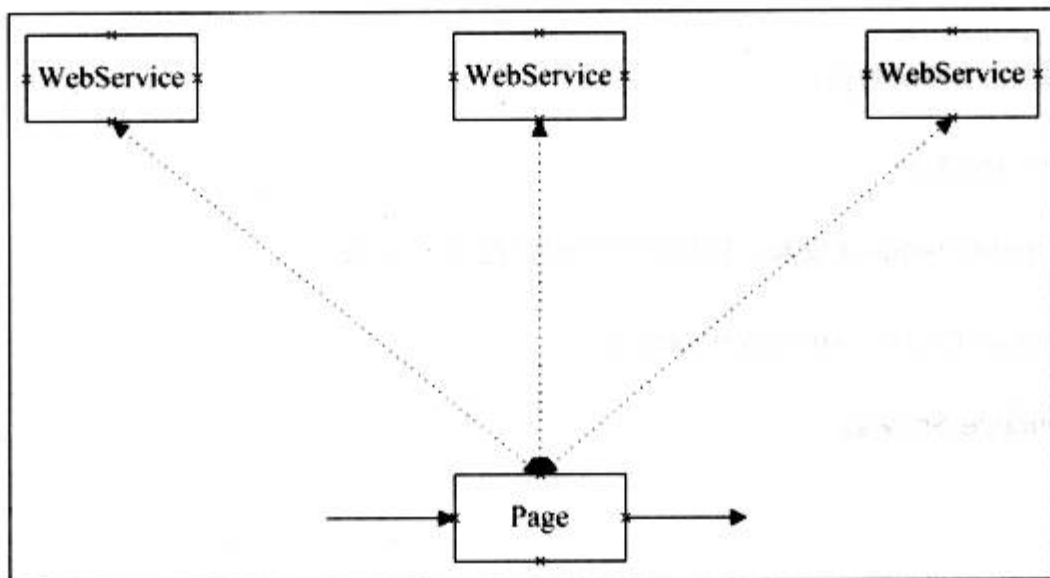


图 16.10 Ajax 下的浏览流程

从 16.2.2 小节开始，将向读者介绍如何在 Firefox 和在 IE 下来解决这个问题。

### 16.2.2 Firefox 下的解决办法

在开始介绍 Firefox 下的解决办法之前，首先需要让读者明白 URL 的中 Hash 的概念，例如下面这段 URL。

<http://www.robchen.cn/demo/selector/index.html#init>

其中“#”号后面的部分就称之为 Hash, 在浏览器中改变 Hash 值并不会导致页面跳转。并且在 Firefox 中, Hash 值的变化也会被浏览器记录进历史记录中。所以可以利用这一机制在 Firefox 下实现 Ajax 的前进和后退功能。

这里, 仍然以 16.1.2 小节中的新闻列表程序为例, 并做一些适当的修改来让其支持前进后退功能, 其实现代码如下所示。

```
<?
/*
 * 使用一个数组作为数据容器
 */
$newsStorage = array();
$newsStorage[] = array("title"=>"News 1","content"=>"something about News 1");
$newsStorage[] = array("title"=>"News 2","content"=>"something about News 2");
$newsStorage[] = array("title"=>"News 3","content"=>"something about News 3");
$newsStorage[] = array("title"=>"News 4","content"=>"something about News 4");
$newsStorage[] = array("title"=>"News 5","content"=>"something about News 5");
/*
 * 取得传递过来的 id 值, 如果为空, 则默认为 0
 */
$id = $_REQUEST['id'];
if(empty($id))
{
    $id = 0;
}
/*
 * 取得索引为指定 id 的新闻数据
 */
$news = $newsStorage[$id];
/*
 * 如果请求是由 xmlhttprequest 发起, 则只输出新闻数据的 json 表示
 */
if($_REQUEST['requestby'] == 'xmlhttprequest')
{
    echo json_encode($news);
    exit;
}
?>
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>News</title>
<script type="text/javascript" src="prototype1.5.js"></script>
<script type="text/javascript">
/*
 * 设置一个名为 currentId 的变量保存当前显示的新闻所对应的 id 值
```

```

    */
    var currentId = 0;
    /*
    * 在 window 的 load 事件中初始化程序
    */
    Event.observe(window,'load',function()
    {
        /*
        * 取得所有#list 下的 a 元素
        */
        var links = $$("#list a");
        /*
        * 遍历所有取得的 a 元素并注册 click 事件处理函数
        */
        links.each(function(link)
        {
            /*
            * 注册 click 事件处理函数，在处理函数中发起 Ajax 请求并更新界面
            */
            Event.observe(link,'click',function(evt)
            {
                /*
                * 读取链接中 href 属性所包含的 id 值，并写入到页面 hash 值中去
                */
                var currentId = link.href.replace(/^(.*\d+).*/,'$1');
                window.location.hash = 'id=' + currentId;
                /*
                * 根据 a 元素的 href 值发起请求并附加一个 requestby 参数，其值为 xmlhttprequest，表示当前
                请求由 xmlhttprequest 发起
                */
                new Ajax.Request(link.href,{
                    method:'GET',
                    parameters:'requestby=xmlhttprequest',
                    /*
                    * 在请求成功时处理返回的 JSON 数据并更新界面
                    */
                    onSuccess:function(x)
                    {
                        var news = eval('(' + x.responseText + ')');
                        $('newsTitle').innerHTML = news.title;
                        $('newsContent').innerHTML = news.content;
                    }
                });
                /*
                * 阻止浏览器的默认行为以避免页面的跳转
                */
                Event.stop(evt);
            });
        });
        /*
        * 设置一个定时器来定时执行 historyChange 函数，这个函数用来检查页面的 hash 值是否发生了变化，并

```

根据检查结果进一步进行处理

```
    */
    setInterval(historyChange,100);
});

function historyChange()
{
    /*
    * 取得当前 hash 值中的 id 数据, 如果 hash 值为空, 则 id 默认为 0
    */
    if(window.location.hash != "")
    {
        var id = window.location.hash.replace(/^.*(\d+).*$/, '$1');
    }
    else
    {
        var id = 0;
    }
    /*
    * 如果 id 与 currentId 相等, 则直接返回
    */
    if(id == currentId)
    {
        return;
    }
    /*
    * 否则就将 id 值赋予 currentId, 使用该 id 值更新界面
    */
    currentId = id;
    new Ajax.Request('news.php',{
        method:'GET',
        parameters:'id=' + id + '&requestby=xmlhttprequest',
        /*
        * 在请求成功时处理返回的 JSON 数据并更新界面
        */
        onSuccess:function(x)
        {
            var news = eval('(' + x.responseText + ')');
            $('newsTitle').innerHTML = news.title;
            $('newsContent').innerHTML = news.content;
        }
    });
}
</script>
<style type="text/css">
*{
    margin:0;
    padding:0;
}
body {
    font-size:11px;
```

```
font-family:Tahoma;
}
h1 {
height:80px;
color:#ccc;
background-color:#333;
line-height:80px;
text-align:center;
font-size:24px;
}
h2 {
font-size:18px;
text-align:center;
border-bottom:1px solid #ccc;
margin-bottom:5px;
height:30px;
line-height:30px;
}
#list {
float:left;
clear:left;
width:20%;
text-align:center;
}
#list li {
list-style:none;
height:20px;
margin-bottom:1px;
}
#list li a{
display:block;
background-color:#ccc;
line-height:20px;
color:#333;
font-weight:bold;
text-decoration:none;
}
#list li a:hover {
background-color:#666;
color:#eee;
}
#detail {
float:left;
margin-left:10px;
width:79%;
}
#newsContent {
text-indent:2em;
}
</style>
</head>
```



```
<body>
<h1>News List System</h1>
<div id="main">
  <ul id="list">
    <? for($i = 0; $i < sizeof($newsStorage); $i ++)>
    {
      echo '<li><a href="news.php?id='.$i.'"
title="'. $newsStorage[$i]['title'].'">'. $newsStorage[$i]['title'].'</a></li>';
    }
  ?>
</ul>
  <div id="detail">
    <h2 id="newsTitle"><?=$news["title"]?></h2>
    <p id="newsContent"><?=$news["content"]?></p>
  </div>
</div>
</body>
</html>
```

现在用 Firefox 访问该页面，如图 16.11 所示。

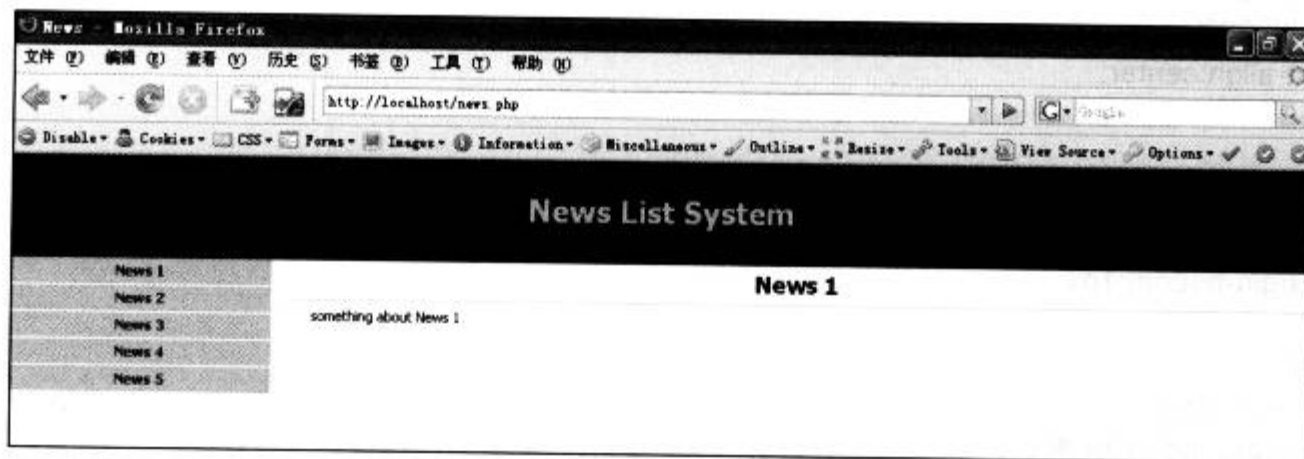


图 16.11 使用 Firefox 访问

此时 Firefox 的地址栏显示为 `http://localhost/news.php`。单击左边列表的其中一个链接，JavaScript 程序此时会读取链接的 `href` 值并使用其中包含的 `id` 值数据更新 Hash 值，然后更新界面，如图 16.12 所示。

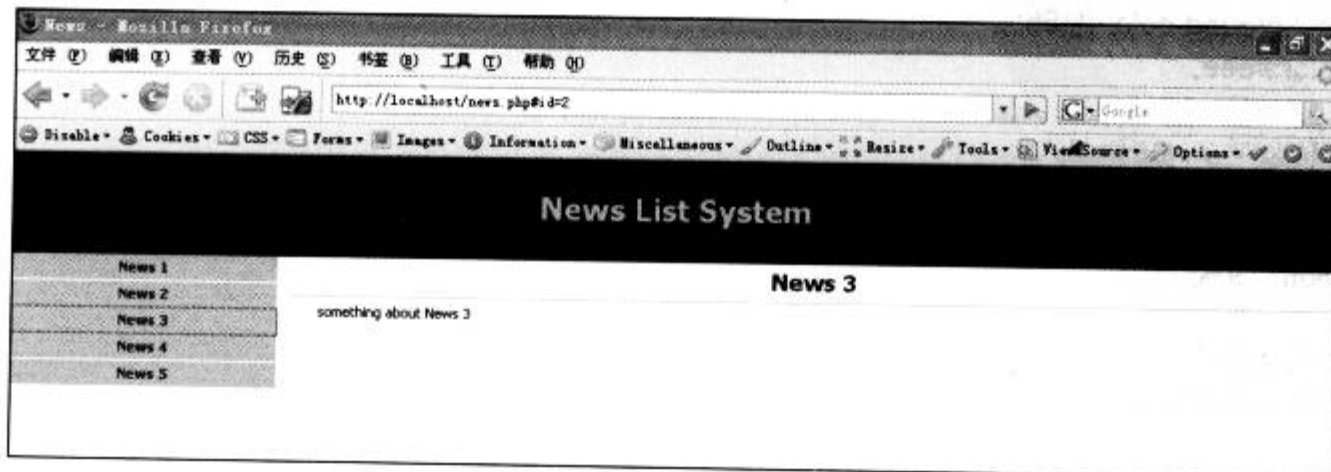


图 16.12 更新 Hash 值

可见, 此时 Firefox 的地址栏显示变为 `http://localhost/news.php?id=2`。然后, 单击 Firefox 的后退按钮, 这时页面的 Hash 值会发生变化, 从而触发执行 `historyChange` 函数, 如图 16.13 所示。

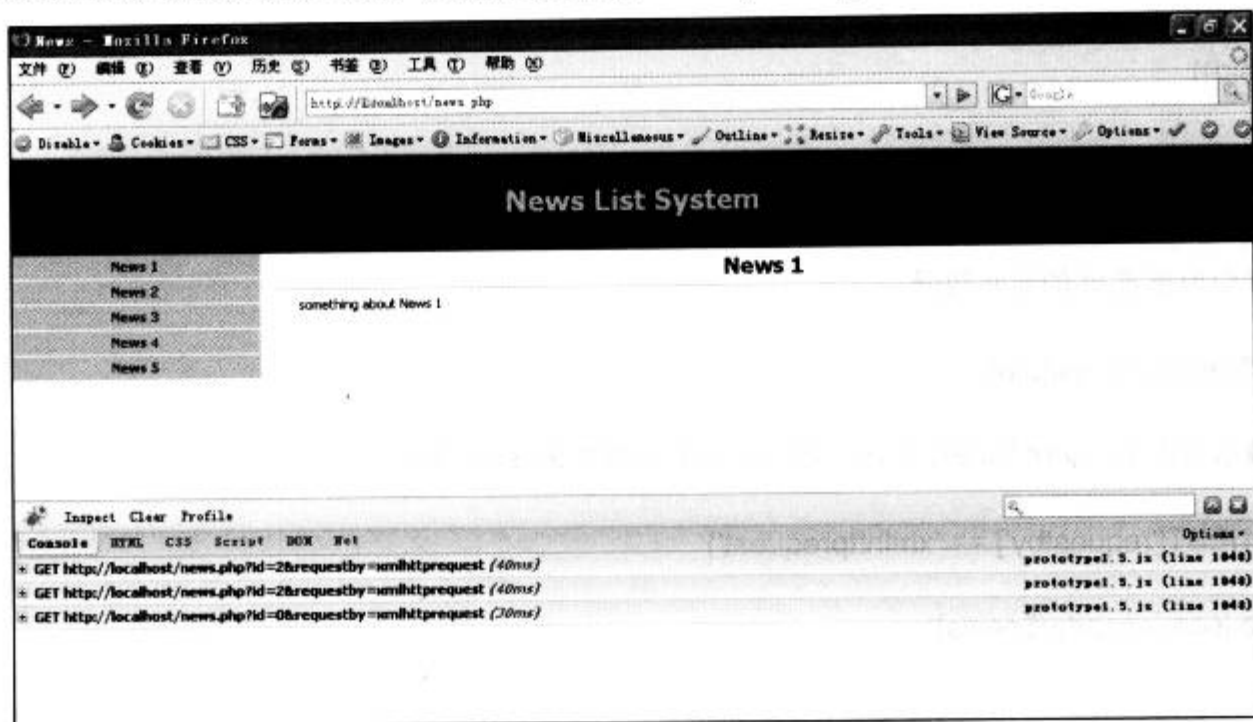


图 16.13 后退

可以看见, 地址栏重新变成了 `http://localhost/news.php`, 并且页面右边的新闻内容重新返回了“News 1”, 从 Firebug 的控制台也可以看到交互的过程。

### 16.2.3 在 IE 下的解决办法

16.2.2 小节中介绍的方法仅仅在 Firefox 及其他 Gecko 核心的浏览器中有效, 但在 IE 中无效。因为在 IE 中, 页面 Hash 值的变化并不会被浏览器记录到浏览历史中, 所以仍然无法使用前进和后退功能。那么在 IE 中如何解决这个问题呢? 答案是 `iframe`。

在 IE 中, 如果一个页面包含一个或者多个 `iframe` 元素, 若 `iframe` 中页面发生了跳转, 那么也会被浏览器记录到浏览历史中。这时, 如果单击浏览器的前进或者后退按钮, 主页面不会受到影响, 只会让 `iframe` 中的页面发生跳转。通过这个机制, 可以将上例中的 `id` 值交给 `iframe` 保存。在发生 Ajax 请求并更新了界面时, 改变 `iframe` 的地址; 在用户单击浏览器的前进或者后退按钮时, 从 `iframe` 中读取 `id` 值并用来更新界面, `news.php` 代码实现如下所示:

```
<?
/*
 * 使用一个数组作为数据容器
 */
$newsStorage = array();
$newsStorage[] = array("title"=>"News 1","content"=>"something about News 1");
$newsStorage[] = array("title"=>"News 2","content"=>"something about News 2");
$newsStorage[] = array("title"=>"News 3","content"=>"something about News 3");
$newsStorage[] = array("title"=>"News 4","content"=>"something about News 4");
$newsStorage[] = array("title"=>"News 5","content"=>"something about News 5");
```

```

/*
 * 取得传递过来的 id 值, 如果为空, 则默认为 0
 */
$id = $_REQUEST['id'];
if(empty($id))
{
    $id = 0;
}
/*
 * 取得索引为指定 id 的新闻数据
 */
$news = $newsStorage[$id];
/*
 * 如果请求是由 xmlhttprequest 发起, 则只输出新闻数据的 json 表示
 */
if($_REQUEST['requestby'] == 'xmlhttprequest')
{
    echo json_encode($news);
    exit;
}
?>
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>News</title>
<script type="text/javascript" src="prototype1.5.js"></script>
<script type="text/javascript">
/*
 * 在 window 的 load 事件中初始化程序
 */
Event.observe(window,'load',function()
{
    /*
     * 取得所有#list 下的 a 元素
     */
    var links = $$("#list a");
    /*
     * 遍历所有取得的 a 元素并注册 click 事件处理函数
     */
    links.each(function(link)
    {
        /*
         * 注册 click 事件处理函数, 在处理函数中发起 Ajax 请求并更新界面
         */
        Event.observe(link,'click',function(evt)
        {
            /*

```

```

        * 取得链接 href 值中包含的 id 值并写入到 iframe 的 src 中的 search 部分
        */
        var id = link.href.replace(/^.*(\d+).*/,'$1');
        $('#ifHistory').src = 'history.html?id=' + id;
        /*
        * 根据 a 元素的 href 值发起请求并附加一个 requestby 参数, 其值为 xmlhttprequest, 表示当前
        请求由 xmlhttprequest 发起
        */
        new Ajax.Request(link.href,{
            method:'GET',
            parameters:'requestby=xmlhttprequest',
            /*
            * 在请求成功时处理返回的 JSON 数据并更新界面
            */
            onSuccess:function(x)
            {
                var news = eval('(' + x.responseText + ')');
                $('#newsTitle').innerHTML = news.title;
                $('#newsContent').innerHTML = news.content;
            }
        });
        /*
        * 阻止浏览器的默认行为以避免页面的跳转
        */
        Event.stop(evt);
    });
});
/*
* 在 iframe 的 history.html 中被调用
*/
function historyChange(id)
{
    /*
    * 使用 id 值更新界面
    */
    new Ajax.Request('news.php',{
        method:'GET',
        parameters:'id=' + id + '&requestby=xmlhttprequest',
        /*
        * 在请求成功时处理返回的 JSON 数据并更新界面
        */
        onSuccess:function(x)
        {
            var news = eval('(' + x.responseText + ')');
            $('#newsTitle').innerHTML = news.title;
            $('#newsContent').innerHTML = news.content;
        }
    });
}

```

```
}
</script>
<style type="text/css">
*{
    margin:0;
    padding:0;
}
body {
    font-size:11px;
    font-family:Tahoma;
}
h1 {
    height:80px;
    color:#ccc;
    background-color:#333;
    line-height:80px;
    text-align:center;
    font-size:24px;
}
h2 {
    font-size:18px;
    text-align:center;
    border-bottom:1px solid #ccc;
    margin-bottom:5px;
    height:30px;
    line-height:30px;
}
#list {
    float:left;
    clear:left;
    width:20%;
    text-align:center;
}
#list li {
    list-style:none;
    height:20px;
    margin-bottom:1px;
}
#list li a{
    display:block;
    background-color:#ccc;
    line-height:20px;
    color:#333;
    font-weight:bold;
    text-decoration:none;
}
#list li a:hover {
    background-color:#666;
    color:#eee;
```



```
    if(id >= 0)
    {
        parent.historyChange(id);
    }
}
</script>
</head>

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

程序中，每次用户单击页面新闻列表页面左边列表中的链接时，程序都会读取该链接包含的 id 值信息并写入到 iframe 页面 url 的 search 中去，这时 iframe 页面发生了跳转，浏览器会记录到浏览历史中去。当单击前进或者后退按钮时，iframe 再次发生跳转，在 window 的 load 事件中，JavaScript 会读取当前 url 包含的 id 信息传递给父页面的 historyChange 函数来更新界面，从而达到前进和后退的效果。

(1) 使用 IE 访问 news.php，效果如图 16.14 所示。

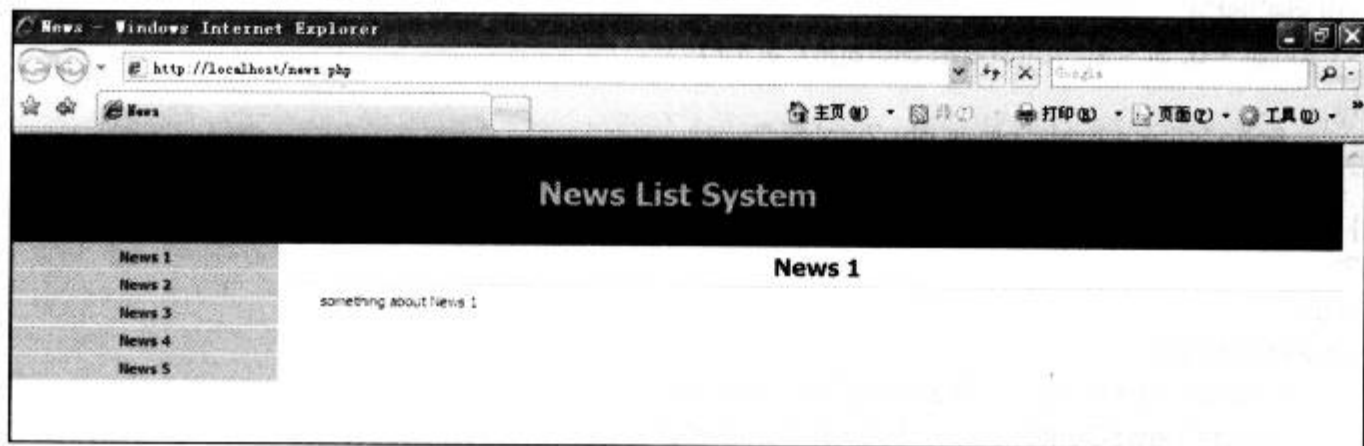


图 16.14 使用 IE 访问

(2) 单击左边列表的某个链接，如图 16.15 所示。

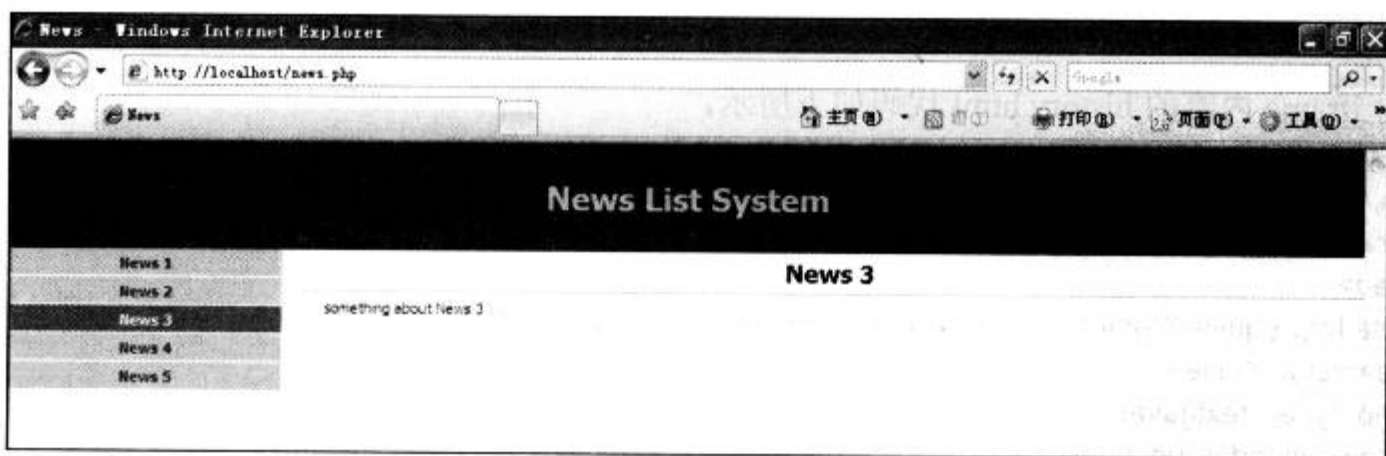


图 16.15 单击链接

(3) 可以看到，界面被更新后，浏览器的后退按钮也变亮了，表示其可用。单击“后退”按钮，如图 16.16 所示。

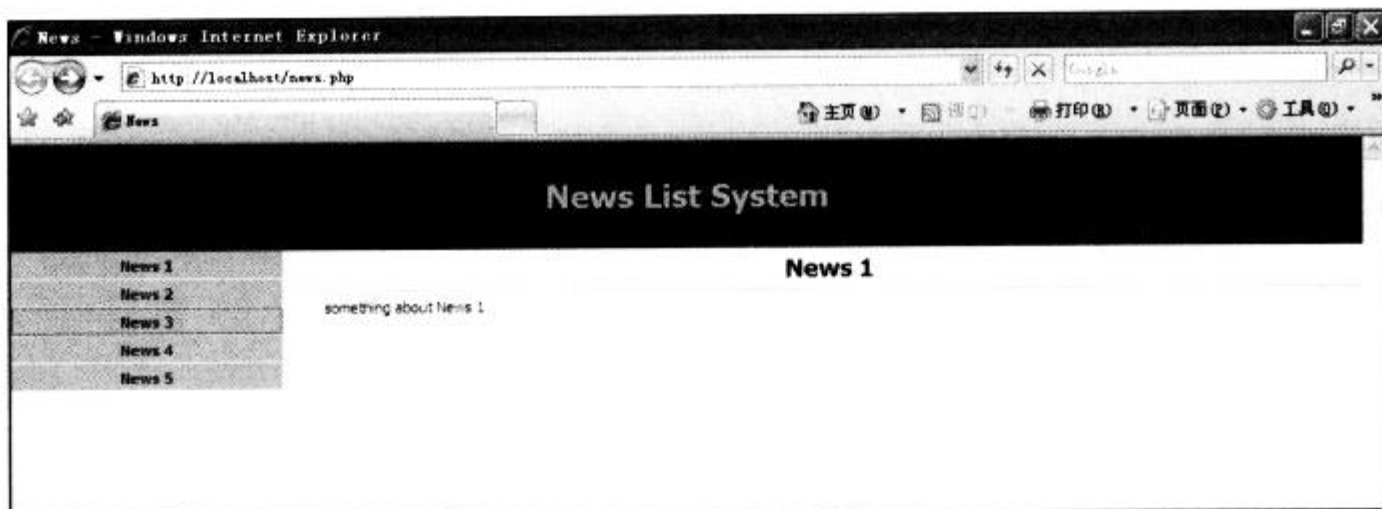


图 16.16 单击“后退”按钮

(4) 可见, 单击“后退”按钮后, 页面重新被更新并显示到了单击链接之间的状态。读者可以看到此时浏览器的前进按钮变亮了, 表示其可用。单击“前进”按钮, 如图 16.17 所示。

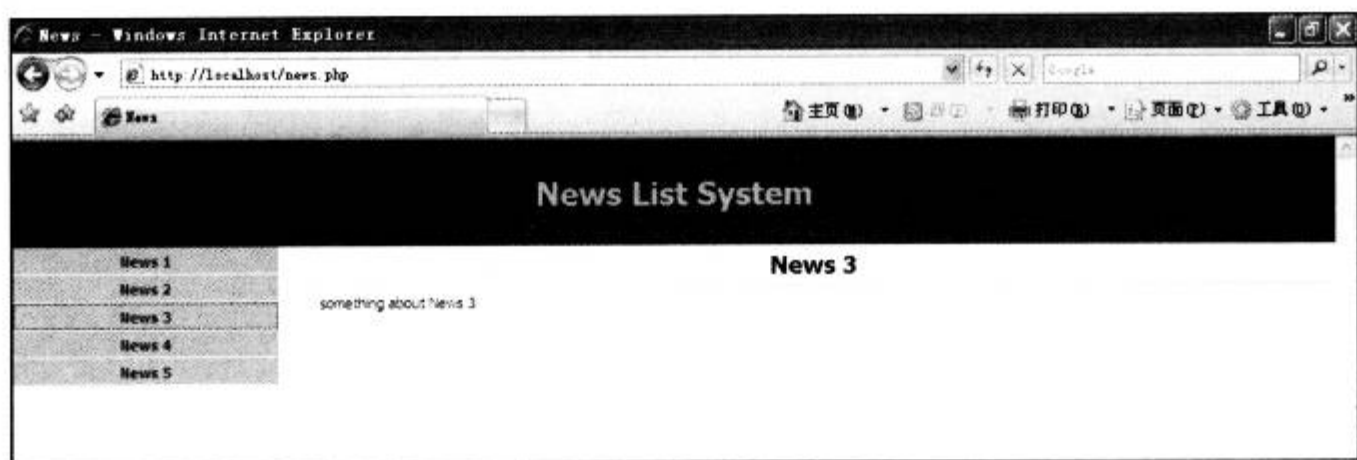


图 16.17 单击“前进”按钮

(5) 此时页面中显示的新闻数据再次回到了单击链接后的效果。

## 16.3 小 结

本章向读者介绍了 Ajax 的主要缺陷, 包括对搜索引擎的不友好, 以及破坏了浏览器的前进和后退功能。本章还详细地向读者分析了问题产生的原因, 并辅以实例说明了解决的办法。其中前进和后退问题要针对不同的浏览器核心来不同的对待。虽然本章提出了解决问题的方案, 但在实际引用过程中, 所面对的环境要比书中的示例复杂很多, 所以还需要读者具体问题具体分析, 而不可生搬硬套。



# 第 5 篇



## Ajax 实战篇

第17章 Ajax高级表单验证程序

第18章 Ajax动态联动菜单

第19章 Ajax聊天室



# 第 17 章

## Ajax 高级表单验证程序

- » 确定需求
- » 基本设计
- » 代码实现
- » 测试

在交互性的 Web 应用程序中，主要依靠各种表单来采集用户的数据。作为安全的、健壮的应用程序的一部分，对表单数据的有效性验证是必不可少的。表单验证程序是 JavaScript 最基本、也是最经典的应用。一个设计良好的表单验证程序，不仅为数据的有效性提供了保障，同时也能给用户带来良好的操作体验。本章将会和读者一起设计并完成一个功能强大的 Ajax 高级表单验证程序。



## 17.1 确定需求

在开始做系统设计之前，首先来确定需求。

- ☐ 提供对表单元素的绑定验证。
- ☐ 提供内建验证规则以支持常用的验证模式，例如邮件地址、电话号码等。
- ☐ 内建规则要能够方便地进行扩展。
- ☐ 提供自定义验证规则，包括自定义正则表达式模式、自定义验证函数等。
- ☐ 支持 Ajax 远程验证。
- ☐ 验证程序要提供足够的事件以提供对界面功能的支持。
- ☐ 验证程序要与界面结构耦合以提高复用度。

## 17.2 基本设计

在确定需求之后，现在开始做系统设计。按照需求，将验证程序的逻辑抽象出来，封装成类，利用 JSVM 进行组织和管理。验证程序由以下 5 个类组成。

- ☐ **EventManager**: 提供事件相关操作的兼容性封装。
- ☐ **Request**: 提供 Ajax 功能的底层支持。
- ☐ **Validator**: 验证器。提供内建验证规则、自定义验证规则、自定义验证函数等功能支持。
- ☐ **FormItemValidator**: 表单项验证器。绑定表单项元素，提供用户自定义验证事件，并自动取值，使用给定的验证器进行验证，对外提供验证的事件。
- ☐ **FormValidator**: 表单验证器。绑定表单元素，并根据其包含的表单项验证器对表单进行验证，对外提供验证事件。

类所包含的属性和方法以及类之间的关系如图 17.1 所示。

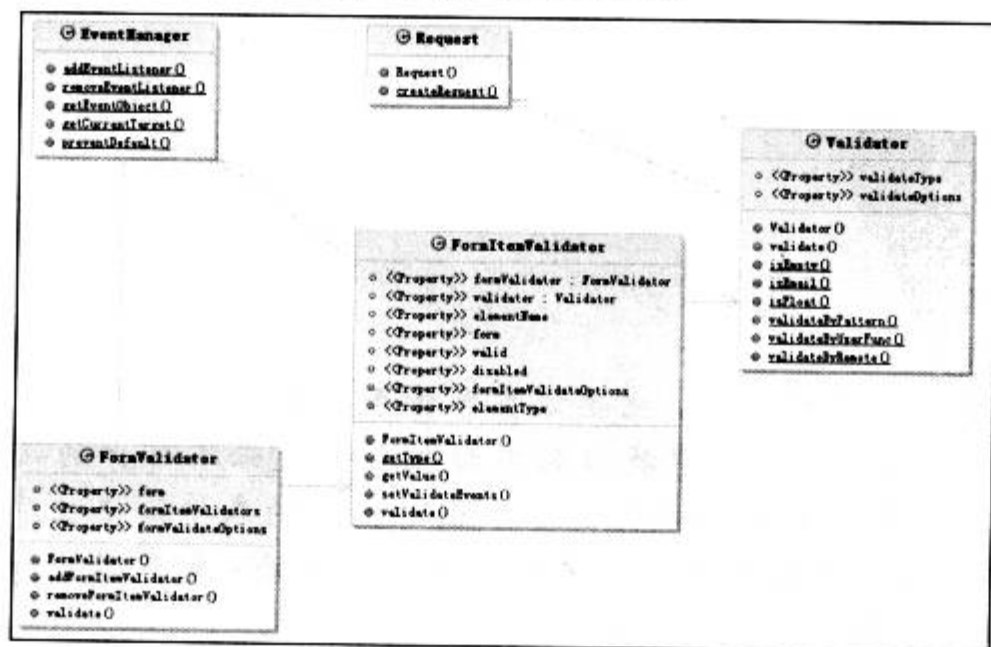


图 17.1 类图

所有类部署到 JSVM 下的 util 包中, 其中 Request 包部署到 util 下的 Ajax 包中, FormValidator、FormItemValidator 和 Validator 部署到 util 下的 validate 包中。包之间的关系如图 17.2 所示。

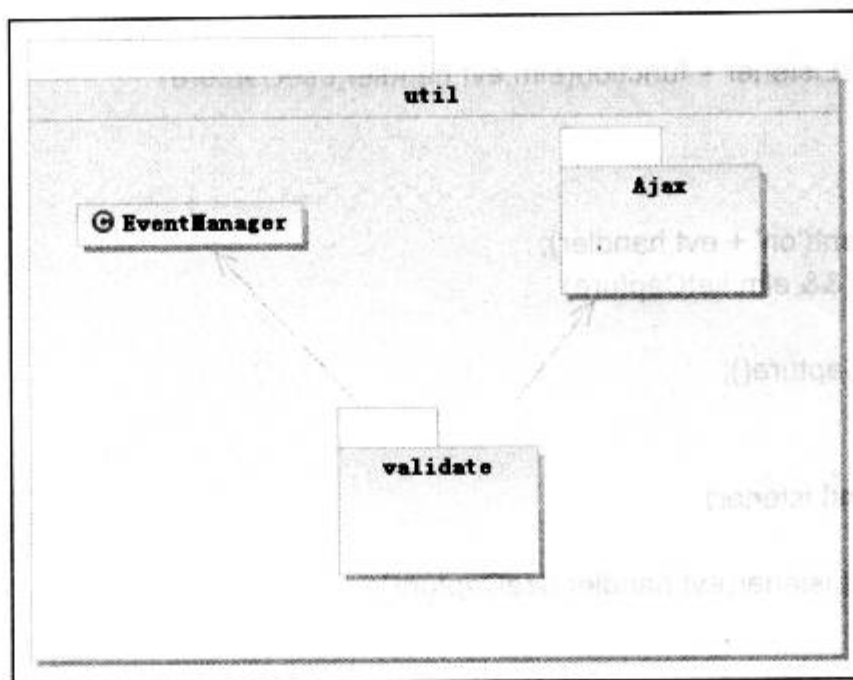


图 17.2 包

## 17.3 代码实现

现在开始对验证程序所包含的类进行具体的代码实现。在 JSVM 的 classes 文件夹下建立文件夹 util, 然后分别建立 Ajax 文件夹和 validate 文件夹。首先实现 EventManager 类。

### 17.3.1 实现 EventManager 类

在 util 文件夹下建立文件 EventManager.class.js, 该类的实现代码如下所示。

```

/*
 * @package util
 * @author: Robin Chen(robchen@126.com)
 * @class: EventManager
 * @description: 事件管理工具,提供对注册和注销事件, 以及事件对象相关处理的兼容性封装
 */
$package('util');

/*
 * @description: 构造函数
 */
var EventManager = util.EventManager = function(){}

/*
 * @description: 提供对事件注册的兼容性封装
 * @params:
 *   eml: HTMLElement, 元素对象

```

```

*      evt: 事件名
*      handler: 事件句柄
*      userCapture: 是否使用捕捉模式
*/
EventManager.addListener = function(elm,evt,handler,userCapture)
{
    if(elm.attachEvent)
    {
        elm.attachEvent('on' + evt,handler);
        if(userCapture && elm.setCapture)
        {
            elm.setCapture();
        }
    }
    else if(elm.addEventListener)
    {
        elm.addEventListener(evt,handler,userCapture);
    }
    else
    {
        throw new Error("浏览器不支持 attachEvent 或 addEventListener");
    }
}
/*
* @description: 提供对事件注销的兼容性封装
* @params:
*      eml: HTMLElement, 元素对象
*      evt: 事件名
*      handler: 事件句柄
*      userCapture: 是否使用捕捉模式
*/
EventManager.removeListener = function(elm,evt,handler,userCapture)
{
    if(elm.detachEvent)
    {
        elm.detachEvent('on' + evt,handler);
        if(userCapture && elm.releaseCapture)
        {
            elm.releaseCapture();
        }
    }
    else if(elm.removeEventListener)
    {
        elm.removeEventListener(evt,handler,userCapture);
    }
    else
    {
        throw new Error("浏览器不支持 detachEvent 或 removeEventListener");
    }
}
```

```

}
/*
 * @description: 取得事件对象
 * @params:
 *      e: EventObject, 在 Gecko 核心的浏览器下, 会传入事件对象
 * @return: EventObject, 事件对象
 */
EventManager.getEventObject = function(e)
{
    return e || event;
}
/*
 * @description: 取得事件作用对象
 * @params:
 *      e: EventObject, 事件对象
 * @return: HTMLElement, 事件作用对象
 */
EventManager.getCurrentTarget = function(e)
{
    var evt = EventManager.getEventObject(e);
    return evt.target || evt.srcElement;
}
/*
 * @description: 阻止浏览器默认行为
 * @params:
 *      e: EventObject, 事件对象
 */
EventManager.preventDefault = function(e)
{
    var evt = EventManager.getEventObject(e);
    if(evt.preventDefault)
    {
        evt.preventDefault();
    }
    else
    {
        evt.returnValue = false;
    }
}

```

### 17.3.2 实现 Request 类

在 Ajax 文件夹下建立 Request.class.js, 该类的实现代码如下所示。

```

/*
 * @packge: util
 * @author: Robin Chen(robchen@126.com)
 */
$package('util.Ajax');

```

```

/*
 * 功能: 根据用户指定的 URL、方法、参数、HTTP 头, 以及回调函数, 自动创建 XMLHttpRequest 对象并发送
请求。
 * 参数介绍:
 *         url: 请求的 URL 地址, 字符串型
 *         options: 参数集合, 对象类型, 其成员均为可选, 成员为:
 *                 method: 发送请求的方法, 可以为 GET 或 POST, 字符串型
 *                 parameters: 需要发送的数据, 字符串型, 其形式为 "a=1&b=2&c=3"
 *                 headers: 需要发送的 HTTP 头信息, 对象类型。其每一个属性为一个头信息。
属性名为头信息的名称, 属性值为头信息的内容
 *                 onLoading: 在请求开始时执行的函数
 *                 onComplete: 在请求完成时执行的函数
 *                 onSuccess: 在请求成功时执行的函数
 *                 onFailure: 在请求失败时执行的函数
 */
var Request = util.Ajax.Request = function(url, options)
{
    var request = Request.createRequest();
    if(typeof request == 'undefined')
    {
        //创建 XMLHttpRequest 对象
        //如果 request 为 undefined, 则抛出异
        //常, 说明当前浏览器不支持 XMLHttpRequest, 并退出函数
        throw new Error('浏览器不支持 XMLHttpRequest');
        return;
    }
    var url = url;
    var method = (options.method || 'POST').toUpperCase();
    if(method != 'GET' && method != 'POST')
    {
        //获取提交方式, 默认为 POST
        method = 'POST';
    }
    var parameters = options.parameters || null;
    var headers = options.headers || {};
    //需要提交的参数, 默认为 null
    //需要发送的 HTTP 头信息, 是一个对
    //象, 其成员包含了头信息的名称和值的信息
    var onLoadingEventHandler = options.onLoading || function();
    //在请求开始时执行的函数, 由用
    //户指定, 默认为一个空函数
    var onCompleteEventHandler = options.onComplete || function();
    //在请求完成时执行的函数, 由用
    //户指定, 默认为一个空函数
    var onSuccessEventHandler = options.onSuccess || function();
    //在请求成功时执行的函数, 由用
    //户指定, 默认为一个空函数
    var onFailureEventHandler = options.onFailure || function();
    //在请求失败时执行的函数, 由用
    //户指定, 默认为一个空函数
    if(method == 'GET' && parameters != null)
    {
        //如果提交方式被指定为 GET,
        //则将 parameters 的内容拼接到 URL 中, 并将 parameters 设置为 null
        {
            if(url.indexOf('?') > -1)
            {
                url += '&' + parameters;
            }
            else

```



```

    {
        url += '?' + parameters;
    }
    parameters = null;
}
request.open(method,url,true);           //初始化 XMLHttpRequest 对象
request.setRequestHeader('contentType','application/x-www-form-urlencoded');
for(var name in headers)                 //设置由用户指定的 HTTP 头信息
{
    request.setRequestHeader(name,headers[name]);
}
request.onreadystatechange = function()   //注册 XMLHttpRequest 对象的 readystatechange 事件
                                          处理函数
{
    if(request.readyState == 1)           //当 readyState 等于 1 时,表示请求开始,将当前 XMLHttpRequest
                                          对象作为其参数调用 onLoadingEventHandler 函数
    {
        onLoadingEventHandler(request);
    }
    if(request.readyState == 4)           //当 readyState 等于 4 时,表示请求完成,将当前 XMLHttpRequest
                                          对象作为其参数调用 onCompleteEventHandler 函数
    {
        onCompleteEventHandler(request);
        if(request.status && request.status >= 200 && request.status < 300)
        {
            onSuccessEventHandler(request);           //当 HTTP 状态码大于等于 200
小于 300 时,表示请求成功,这时将当前 XMLHttpRequest 对象作为其参数调用 onSuccessEventHandler 函数
        }
        else
        {
            onFailureEventHandler(request);           //否则表示请求失败,这时将当前
XMLHttpRequest 对象作为其参数调用 onFailureEventHandler 函数
        }
    }
}
request.send(parameters);                 //发送请求
return request;
}
/*
 * 根据不同的浏览器内核和 XMLHttpRequest 版本创建 XMLHttpRequest 实例
 */
Request.createRequest = function()
{
    var versions = [
        function()
        {
            return new XMLHttpRequest();
        },
        function()

```

```

    {
        return new ActiveXObject('Microsoft.XMLHTTP');
    },
    function()
    {
        return new ActiveXObject('MSXML2.XmlHttp.6.0');
    },
    function()
    {
        return new ActiveXObject('MSXML2.XmlHttp.3.0');
    }
];
var request;
for(var i = 0; i < versions.length; i++)
{
    var lambda = versions[i];
    try
    {
        request = lambda();
        break;
    }
    catch(e){}
}
return request;
}

```

### 17.3.3 实现 Validator 类

在 validate 文件夹下建立文件 Validator.class.js, 该类的实现代码如下所示。

```

/*
 * @package util.validate
 * @author: Robin Chen(robchen@126.com)
 * @class: Validator
 * @description: 验证器类, 实现验证规则与验证行为分离, 便于维护和扩展, 可以以多个 FormItemValidator 对
象公用一个验证器
 */
$package('util.validate');
/*
 * @description: 构造函数
 * @params:
 *      validateType: string, 验证规则, 参见 Validator.validate() 方法, 规则可扩展
 *      validateOptions: object, 验证相关选项, 一个完整的参数示例如下所示
 *          validateOption: {
 *              patternOptions: {
 *                  pattern: '.*', //使用 PATTERN'则时, 自定义模式验证的规则
 *                                //模式表达式的字符串表示
 *                  Global: false, //是否使用全局匹配模式
 *                  Multiline: false, //是否使用多行匹配模式
 *                  Ignore: false //是否使用忽略大小写的匹配模式

```

```

*      },
*      remoteOptions:{                                //使用 REMOTE 模式时, 相关参数配置
*          url:'xxx.php',                             //后台验证程序的地址
*          parameters:'a=1&b=2...',                   //验证请求的额外参数
*          method:'GET',                             //提交验证请求的方法 POST/GET
*          headers:{'xx':'xxx',...},                 //指定验证请求的 HTTP 头信息
*          onLoading:function(){...},                 //在请求开始时被调用的回调函数
*          userFunc:function(xmlhttprequest){...}     //在请求完成时被调用的回调函数, 需要返
回一个 bool 类型的值提供给验证器以识别是否验证成功
*      }
*  }
*/
var Validator = util.validate.Validator = function(validateType,validateOptions)
{
    this.validateType = validateType.toUpperCase();
    this.validateOptions = validateOptions;
}
/*
* @description: 根据指定的规则和参数对所给的值进行验证
* @params:
*     value: object, 需要验证的值
*     formItemValidator: object, 当前使用该验证器的 FormItemValidator 对象
*     autoContinue: bool, 是否自动继续验证, 用在 REMOTE 模式验证时, 在请求返回后自动完成剩下的验
证过程
* @return:
*     bool, 表示当前验证是否通过
*     LOADING, 表示当前远程验证请求已经发出还未返回
*/
Validator.prototype.validate = function(value,formItemValidator,autoContinue)
{
    switch(this.validateType)
    {
        case 'NOEMPTY':
            return !Validator.isEmpty(value || "");
        case 'EMPTY':
            return Validator.isEmpty(value || "");
        case 'EMAIL':
            return Validator.isEmail(value || "");
        case 'FLOAT':
            return Validator.isFloat(value || "");
        case 'PATTERN':
            return Validator.validateByPattern(value || "",this.validateOptions.patternOptions);
        case 'USERFUNC':
            return Validator.validateByUserFunc(value,this.validateOptions.userFunc);
        case 'REMOTE':
            return Validator.validateByRemote(value ||
            "",this.validateOptions.remoteOptions,formItemValidator,autoContinue);
        default:
            throw new Error("\\" + this.validateType + "\" 类型不能被处理");
    }
}

```

```

    }
}
/*
 * @description: 验证值是否为空
 * @params:
 *     value: object, 需要验证的值
 * @return: bool, 验证结果
 */
Validator.isEmpty = function(value)
{
    return /^s*$/ .test(value);
}
/*
 * @description: 验证值是否为 email 格式
 * @params:
 *     value: object, 需要验证的值
 * @return: bool, 验证结果
 */
Validator.isEmail = function(value)
{
    return /^w+([-+.]w+)*@w+([-+.]w+)*\lw+([-+.]w+)*$/ .test(value);
}
/*
 * @description: 验证值是否为浮点数
 * @params:
 *     value: object, 需要验证的值
 * @return: bool, 验证结果
 */
Validator.isFloat = function(value)
{
    return /^(-?\d+)(\.\d+)?$/ .test(value);
}
/*
 * @description: 根据自定义的模式规则对值进行验证
 * @params:
 *     value: object, 需要验证的值
 *     patternOptions: object, 模式规则选项, 完整的参数示例如下所示。
 *
 *         patternOptions = {
 *             pattern: '.*', //使用 PATTERN 规则时, 自定义模式验证的规则
 *             Global: false, //模式表达式的字符串表示
 *             Multiline: false, //是否使用全局匹配模式
 *             Ignore: false, //是否使用多行匹配模式
 *             //是否使用忽略大小写的匹配模式
 *         },
 *     formItemValidator: object, 当前的 FormItemValidator 对象
 * @return: bool, 验证结果
 */
Validator.validateByPattern = function(value, patternOptions, formItemValidator)
{
    var options = "";

```

```

    if(patternOptions)
    {
        if(patternOptions.Global)
        {
            options += 'g';
        }
        if(patternOptions.Multiline)
        {
            options += 'm';
        }
        if(patternOptions.Ignore)
        {
            options += 'i';
        }
    }
    var reg = new RegExp(patternOptions.pattern,options);
    return reg.test(value);
}
/*
 * @description: 使用自定义函数对值进行验证
 * @params:
 *     value: object, 需要验证的值
 *     userFunc: function, 用户自定义函数, 值会被作为参数传给这个函数, 函数返回一个 bool 值作为验证
结果, 一个自定义验证函数的示例如下所示
 *
 *     function userFunc(value)
 *     {
 *         return parseInt(value) > 1;    //判断值是否大于 1, 大于 1 则通过验证, 否则验证失败
 *     }
 * @return: bool,验证结果
 */
Validator.validateByUserFunc = function(value,userFunc)
{
    return userFunc(value);
}
/*
 * @description: 远程验证, 制造一个 Ajax 请求, 将值传送到后台处理程序, 并使用后台返回的数据作为验证
依据
 * @params:
 *     value: object, 需要验证的值
 *     remoteOptions: object, 远程验证相关选项, 一个完整的参数示例如下所示。
 *
 *         remoteOptions = {
 *             url:'xxx.php',                //使用 REMOTE 模式时, 相关参数配置
 *             parameters:'a=1&b=2...',      //后台验证程序的地址
 *             method:'GET',                 //验证请求的额外参数
 *             headers:{'xx':'xxx',...},     //提交验证请求的方法 POST/GET
 *             onLoading:function(){...},    //指定验证请求的 HTTP 头信息
 *             userFunc:function(xmlhttpreuest){...} //在请求开始时被调用的回调函数
 *
 *         }
 *
 *         //在请求完成时被调用的回调函数, 需要返
回一个 bool 类型的值提供给验证器以识别是否验证成功
 *
 *     }

```



```

*      formItemValidator: object, 当前 FormItemValidator 对象
*      authContinue: bool, 是否在远程验证结束时自动对表单进行验证
* @return: bool, 验证结果
*/
Validator.validateByRemote = function(value, remoteOptions, formItemValidator, autoContinue)
{
    $import('util.Ajax.Request');
    setTimeout(function(){
        var request = new util.Ajax.Request(remoteOptions.url, {
            method: remoteOptions.method || 'GET',
            parameters: (remoteOptions.parameters || '') + '&' + formItemValidator.elementName + '=' +
value,
            headers: remoteOptions.headers || {},
            onLoading: remoteOptions.onLoading || function() {},
            onComplete: function(xmlHttpRequest)
            {
                formItemValidator.valid = remoteOptions.userFunc(xmlHttpRequest);
                if(autoContinue)
                {
                    formItemValidator.formValidator.validate(true);
                }
            }
        });
    }, 200);
    return 'LOADING';
}

```

### 17.3.4 实现 FormItemValidator 类

在 validate 文件夹下建立 FormItemValidator.class.js, 该类的实现代码如下所示。

```

/*
* @package util.validate
* @author: Robin Chen(robchen@126.com)
* @class: FormItemValidator
* @description: 表单项验证
*/
$package('util.validate');

$import('util.validate.Validator');
$import('util.EventManager');
/*
* @description: 构造函数
* @params:
*      formValidator: object, 所依赖的 FormValidator 对象, 参见 util.validate.FormValidator
*      elementName: string, 表单元素的 name 属性
*      validator: object, 验证所使用的验证器, 指定了验证规则, 参见 util.validate.Validator
*      formItemValidateOptions: object, 验证相关选项, 一个完整的参数示例如下所示
*
*      formItemValidateOptions = {

```

```

*           events:['change','focus','blur','click'],           //需要将验证行为绑定到的事件
*           //在验证失败时执行的回调函数
*           //formItemValidator: 当前 FormItemValidator 对象
*           onError:function(formItemValidator)
*           {
*               //do something...
*           },
*           //在验证成功时执行的回调函数
*           //formItemValidator: 当前 FormItemValidator 对象
*           onCorrect:function(formItemValidator)
*           {
*               //do something...
*           },
*           disabled:false,                                     //是否禁用验证器, 默认为 false
*       }
*/

var FormItemValidator = util.validate.FormItemValidator =
function(formValidator,elementName,validator,formItemValidateOptions)
{
    this.formValidator = formValidator;
    this.validator = validator;
    this.elementName = elementName;
    this.form = this.formValidator.form;
    this.valid = false;
    this.disabled = false;
    this.formItemValidateOptions = formItemValidateOptions;
    var elements = this.form.elements[elementName];
    if(elements.length)
    {
        this.elements = elements;
    }
    else
    {
        this.elements = [elements];
    }
    if(!this.elements[0])
    {
        throw new Error("指定的表单元素不存在");
    }
    this.elementType = FormItemValidator.getType(this.elements[0]);
    if(this.elementType == 'select')
    {
        this.elements = [this.elements[0].parentNode];
    }
    this.setValidateEvents();
}
/*
* @description: 取得表单元素的类型

```

```
* @params:
*   element: HTMLFormElement, 表单元素对象
* @return: string, 表单元素的类型
*/
FormItemValidator.getType = function(element)
{
    if(element.tagName.toLowerCase() == 'input')
    {
        return element.type.toLowerCase();
    }
    else if(element.tagName.toLowerCase() == 'textarea')
    {
        return 'textarea';
    }
    else if(element.tagName.toLowerCase() == 'option')
    {
        return 'select';
    }
    else
    {
        throw new Error('给定的元素不能被处理。');
    }
}

FormItemValidator.prototype = {
    /*
    * @description: 取得当前表单项的值,根据表单项的不同类型采取不同的取值方法
    */
    getValue:function()
    {
        switch(this.elementType)
        {
            case 'text':
            case 'password':
            case 'textarea':
            case 'select':
                return this.elements[0].value;
            case 'radio':
                var elements = this.elements;
                for(var i = 0, l = elements.length; i < l; i++)
                {
                    if(elements[i].checked)
                    {
                        return elements[i].value;
                    }
                }
                return null;
            case 'checkbox':
                var elements = this.elements;
```

```

        var value = [];
        for(var i = 0, l = elements.length; i < l; i++)
        {
            if(elements[i].checked)
            {
                value.push(elements[i].value);
            }
        }
        if(value.length)
        {
            return value.join(',');
        }
        else
        {
            return null;
        }
        default:
            return null;
    }
},
/*
 * @description: 绑定验证相关的事件句柄，一般在构造函数中被调用
 */
setValidateEvents:function()
{
    if(!this.formItemValidateOptions.events || !this.formItemValidateOptions.events.length)
    {
        return;
    }
    var _this = this;
    var elements = this.elements;
    var events = this.formItemValidateOptions.events;
    var element, evt, useChangeEvent = false;
    for(var i = 0, l = elements.length; i < l; i++)
    {
        element = elements[i];
        for(var j = 0, n = events.length; j < n; j++)
        {
            evt = events[j];
            if(evt == 'change')
            {
                useChangeEvent = true;
                continue;
            }
        }
        util.EventManager.addListener(element, evt, function()
        {
            _this.validate();
        }, false);
    }
};

```

```

    }
    util.EventManager.addListener(element,'change',function()
    {
        _this.valid = false;
        if(useChangeEvent)
        {
            _this.validate();
        }
    },false
    );
}
},
/*
 * @description: 验证表单项
 * @params:
 *      autoContinue: bool, 提供给 Validator 对象使用,一般在远程验证时使用
 */
validate:function(autoContinue)
{
    if(this.disabled)
    {
        return;
    }
    var value = this.getValue();
    this.valid = this.validator.validate(value,this,autoContinue);
    var valid = this.valid;
    var formItemValidateOptions = this.formItemValidateOptions;
    if(valid == true)
    {
        if(formItemValidateOptions.onCorrect)
        {
            formItemValidateOptions.onCorrect(this);
        }
    }
    else if(valid == false)
    {
        if(formItemValidateOptions.onError)
        {
            formItemValidateOptions.onError(this);
        }
    }
    return valid;
}
}
}

```

### 17.3.5 实现 FormValidator 类

在 validate 文件夹下建立文件 FormValidator.class.js, 该类的实现代码如下所示。



```

/*
 * @package util.validate
 * @author: Robin Chen(robchen@126.com)
 * @class: FormValidator
 * @description: 表单验证类
 */
$package('util.validate');

$import('util.EventManager');
$import('util.validate.FormItemValidator');
/*
 * @description: 构造函数
 * @params:
 *      formElement: HTMLFormElement, 表单对象
 *      formValidateOptions: 表单验证相关选项, 完整的参数示例如下所示。
 *
 *      formValidateOptions = {
 *          stopSubmit:true,           //是否在验证失败时组织表单提交
 *          loadingNotice:"           //在遇到远程验证时的提示语
 *          onError:function(formValidator){...}, //在验证失败时被调用的回调函数, formValidator
为当前 FormValidator 对象
 *          onCorrect:function(formValidator){...}, //在验证成功时被调用的回调函数, formValidator
为当前 FormValidator 对象
 *          userNoticeFunc:function(msg){...} //用户自定义提示信息处理函数, msg 为提示信息
 *      }
 */
var FormValidator = util.validate.FormValidator = function(formElement,formValidateOptions)
{
    this.form = formElement;
    this.formItemValidators = {};
    var formValidateOptions = formValidateOptions || {};
    if(!formValidateOptions.userNoticeFunc)
    {
        formValidateOptions.userNoticeFunc = function(msg)
        {
            alert(msg);
        }
    }
    this.formValidateOptions = formValidateOptions;
    var _this = this;
    util.EventManager.addListener(this.form,'submit',function(e)
    {
        var result = _this.validate(true);
        if(!result)
        {
            if(typeof result != 'undefined' && formValidateOptions.onError)
            {
                formValidateOptions.onError(util.EventManager.getEventObject(e),__this);
            }
            if(formValidateOptions.stopSubmit)

```

```

        {
            util.EventManager.preventDefault(e);
        }
    }
    if(result && formValidateOptions.onCorrect)
    {
        formValidateOptions.onCorrect(util.EventManager.getEventObject(e),_this);
    }
    },false
);
}

```

```
FormValidator.prototype = {
```

```

    /*
    * @description: 添加表单项验证器
    * @params
    *     formItemValidator: object, FormItemValidator 对象
    */

```

```
addFormItemValidator:function(formItemValidator)
```

```

{
    formItemValidator.disabled = false;
    this.formItemValidators[formItemValidator.elementName] = formItemValidator;
},

```

```
removeFormItemValidator:function(obj)
```

```

{
    if(typeof obj == 'string')
    {
        this.formItemValidators[obj].disabled = true;
        delete this.formItemValidators[obj];
    }
    else
    {
        this.formItemValidators[obj.elementName].disabled = true;
        delete this.formItemValidators[obj.elementName];
    }
},

```

```
validate:function(isSubmitEvent)
```

```

{
    var formItemValidator;
    var formItemValidators = this.formItemValidators;
    var userNoticeFunc = this.formValidateOptions.userNoticeFunc;
    var loadingNotice = this.formValidateOptions.loadingNotice || '一个远程验证的结果还未返回，请稍后
再尝试提交。';

```

```

    var result;
    for(name in formItemValidators)
    {
        formItemValidator = formItemValidators[name];
        if(!formItemValidator.valid)
        {

```

```

        result = formItemValidator.validate(isSubmitEvent);
        if(result == 'LOADING')
        {
            userNoticeFunc/loadingNotice);
            return;
        }
        if(!result)
        {
            return false;
        }
    }
}
return true;
}
}
}

```

## 17.4 测 试

在完成代码实现以后，编写以下用例以进行测试，测试代码如下所示。  
index.html 代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>form-validator</title>
<script type="text/javascript" src="../../jsvm2/jsre.js"></script>
<script type="text/javascript">
window.onload = function(){
    $import('util.validate.FormValidator'); //引入 FormValidator
    /*
    * 创建消息显示函数
    */
    var showMsg = function(msg)
    {
        var msgbox = document.getElementById('msg');
        msgbox.innerHTML += '<div>' + msg + '</div>';
        msgbox.scrollTop = msgbox.scrollHeight;
    }
    /*
    * 创建 FormValidator 示例
    */
    val = new util.validate.FormValidator(document.getElementById('test'),{
        stopSubmit:true, //验证失败时阻止表单提交
    });
    /*
    * 在验证失败时输出信息
    */
}

```

```

        onError:function(e,obj)
        {
            showMsg('表单验证失败');
        },
        /*
        * 在验证成功时输出信息
        */
        onCorrect:function(e,obj)
        {
            showMsg('表单验证成功');
            util.EventManager.preventDefault(e);
        },
        /*
        * 用户自定义提示函数
        */
        userNoticeFunc:function(msg)
        {
            showMsg(msg);
        }
    });
    /*
    * 创建验证器，绑定 NOEMPTY 规则
    */
    var generalVal = new util.validate.Validator('NOEMPTY');
    /*
    * 创建验证器，绑定 EMAIL 规则
    */
    var emailVal = new util.validate.Validator('EMAIL');
    /*
    * 创建验证器，使用 REMOTE 模式进行远程 Ajax 验证
    */
    var remoteVal = new util.validate.Validator('REMOTE',{
        remoteOptions:{
            url:'validate.php',
            userFunc:function(x)
            {
                if(x.responseText == 1)
                {
                    return !showMsg('远程验证成功');
                }
                else
                {
                    return !!showMsg('远程验证失败');
                }
            }
        }
    });
    /*
    * 表单项验证参数
    */

```

```

var itemOptions = {
    events:['change'],
    onError:function(itemValidator)
    {
        showMsg(itemValidator.elementName + ' 验证失败');
    },
    onCorrect:function(itemValidator)
    {
        showMsg(itemValidator.elementName + ' 验证通过');
    }
}
/*
 * 为表单元素创建 FormItemValidator 实例
 */
var itemVal = new util.validate.FormItemValidator(val,'cb',generalVal,itemOptions);
var mailVal = new util.validate.FormItemValidator(val,'mail',emailVal,itemOptions);
var userVal = new util.validate.FormItemValidator(val,'username',remoteVal,itemOptions);
var cityVal = new util.validate.FormItemValidator(val,'city',generalVal,itemOptions);
var mutiVal = new util.validate.FormItemValidator(val,'language',generalVal,itemOptions);
/*
 * 将 FormItemValidator 实例加入到表单验证器中
 */
val.addFormItemValidator(itemVal);
val.addFormItemValidator(mailVal);
val.addFormItemValidator(userVal);
val.addFormItemValidator(cityVal);
val.addFormItemValidator(mutiVal);
}
</script>
<style type="text/css">
body {font-size:12px;}
input,select {display:block; margin-top:5px;}
#msg {
    position:absolute;
    width:454px;
    height:300px;
    overflow-y:scroll;
    z-index:1;
    left: 240px;
    top: 23px;
    background-color: #eee;
}
</style>
</head>

<body>
<form name="test" id="test">
    <label for="cb1">cb:</label>
    <input type="checkbox" name="cb" id="cb1" value="1" />
    <label for="cb2">cb:</label>

```



```

<input type="checkbox" name="cb" id="cb2" value="2" />
<label for="mail">email:</label>
<input type="text" name="mail" id="mail" value="robchen@126.com" />
<label for="username">username:</label>
<input type="text" name="username" id="username" value="xxx" />
<div id="msg"></div>
<label for="city">city:</label>
<select name="city" id="city">
  <option value="">Select</option>
  <option value="Wuhan">Wuhan</option>
  <option value="Tianjing">Tianjing</option>
</select>
<label for="language">language:</label>
<select name="language" id="lanaguage" multiple="multiple">
  <option value="en">en</option>
  <option value="fr">fr</option>
  <option value="de">de</option>
  <option value="jp">jp</option>
  <option value="co">co</option>
</select>
<input type="submit" id="btn" value="validate" />
</form>
</body>
</html>

```

validate.php 代码如下所示。

```

<?
header('Content-Type:text/html;charset=utf-8');
echo '1';
?>

```

将 JSVM 和示例程序都放到 Apache 的虚拟目录中，并使用浏览器访问示例程序，如图 17.3 所示。

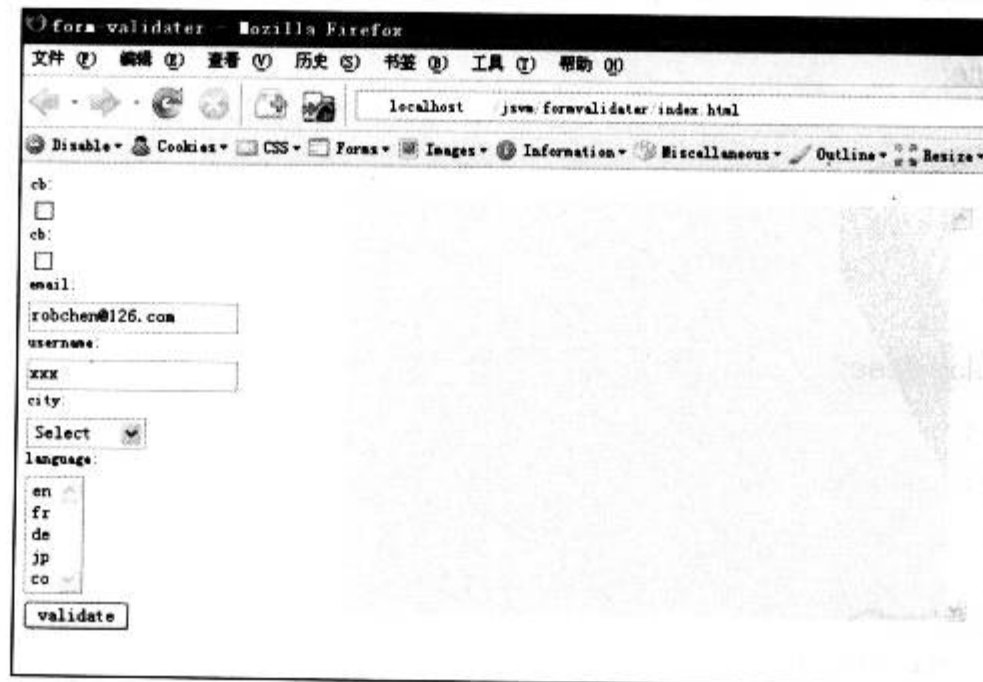


图 17.3 示例程序

对表单项进行操作, 或者单击 **validate** 按钮, 验证程序会自动按照指定的规则进行验证, 并将验证信息输出到右边的灰色区域, 如图 17.4 所示。

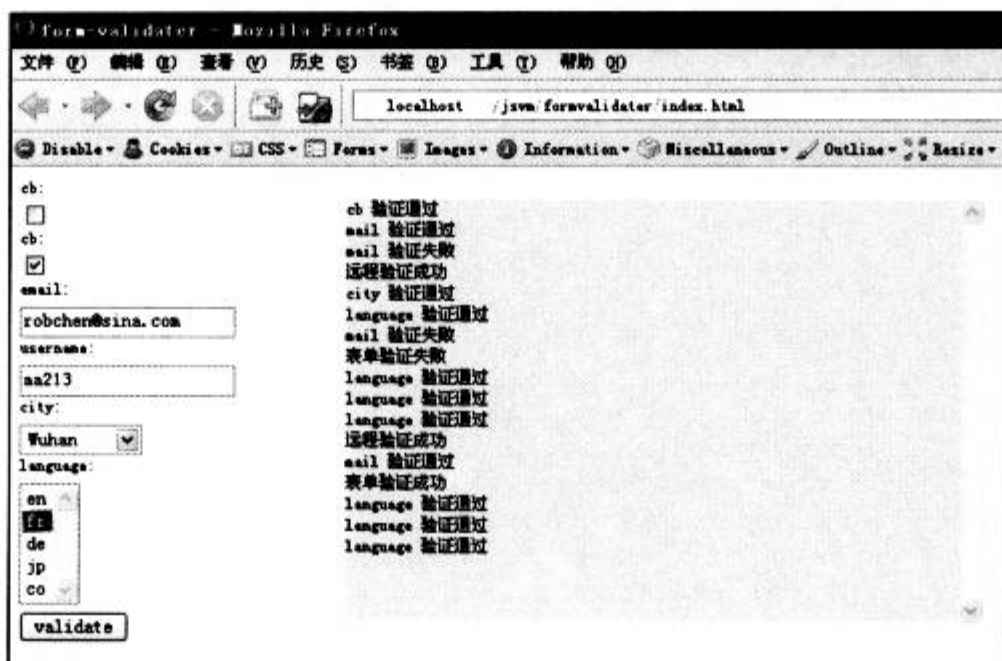


图 17.4 示例程序



# 第18章

## Ajax 动态联动菜单

- » 确定需求
- » 基本设计
- » 实例代码
- » 测试

联动菜单是指多个下拉框菜单组成一组菜单，下拉菜单之间有联动的关系。上级菜单的选中项改变时，下级菜单会根据上级菜单的选中项显示相应的选项内容。联动菜单在 Web 应用程序中应用非常广泛，例如用来呈现多级分类、显示省市级联等。传统的联动菜单都是使用 XML 或者数组保存选项数据，这就要求在页面加载时加载所有的选项数据。当选项数据较为庞大时，就会影响加载速度。本章将向读者介绍如何创建一个基于 Ajax 的无限级联动菜单。



## 18.1 确定需求

首先需要确定联动菜单程序的需求。

- ☐ 采用 Ajax 加载选项数据。
- ☐ 级联层数无限制。
- ☐ 对外提供事件支持。
- ☐ 支持一对一、一对多、多对一、多对多的关联模式。
- ☐ 可以动态修改关联关系。
- ☐ 支持 JSON 和 XML 两种数据格式。

## 18.2 基本设计

在确定需求后，现在对联动菜单程序进行基本设计。联动菜单基于 JSVM 搭建，使用本书第 17 章中已经创建的 Request 类提供 Ajax 底层功能，并使用 EventManager 类提供的事件兼容性封装。

将联动菜单的逻辑抽象出来封装成类，并命名为 SelectLoader，计划存放在 util 包下的 Ajax 包中。每个 SelectLoader 实例对应一个 select 元素。SelectLoader 类封装了用户事件响应和数据加载的功能，并通过添加子加载器来建立关联关系。在实例对应的 select 元素选中项发生改变时，自动调用所包含的子加载器实例对象进行数据加载。SelectLoader 类所包含的属性和方法如图 18.1 所示。

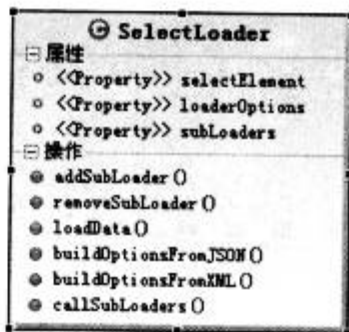


图 18.1 SelectLoader 类

## 18.3 实例代码

SelectLoader 类的代码如下所示。

```

/*
 * @package: util.Ajax
 * @author: Robin Chen(robchen@126.com)
 * @description: 无限级 Ajax 下拉菜单，支持一对一、一对多、多对一、多对多模式，支持 JSON 和 XML 数据
 */
$package('util.Ajax.SelectLoader');

$import('util.EventManager');
/*
 * @description: 构造函数
 * @params:
 *      selectElement: HTMLSelectElement, select 元素对象
  
```



```

*      loaderOptions: object, 相关选项, 参数示例如下所示。
*      loaderOptions = {
*          url:'xx.php',                      //webservice 地址
*          method:'GET',                      //请求方法
*          headers: {'headerName':'headerValue',...}, //请求头信息
*          parameters:'a=1&b=2&c=3...',        //请求的额外参数
*          onLoading:function(){...},          //在开始加载时调用的回调函数
*          onLoadingComplete:function(){...}    //在加载完成后调用的回调函数
*      }
*/
var SelectLoader = util.Ajax.SelectLoader = function(selectElement,loaderOptions)
{
    this.selectElement = selectElement;
    this.loaderOptions = loaderOptions;
    this.subLoaders = {};
    var _this = this;
    util.EventManager.addListener(selectElement,'change',function()
    {
        _this.callSubLoaders();
    },false
    );
}

SelectLoader.prototype = {
    /*
    * @description: 增加子加载器
    * @params:
    *     selectLoader: SelectLoader, SelectLoader 对象实例
    */
    addSubLoader:function(selectLoader)
    {
        this.subLoaders[selectLoader.selectElement.name] = selectLoader;
    },
    /*
    * @description: 移除子加载器
    * @params:
    *     selectLoader: SelectLoader, SelectLoader 对象实例
    */
    removeSubLoader:function(selectLoader)
    {
        delete this.subLoaders[selectLoader.selectElement.name];
    },
    /*
    * @description: 加载数据, 如果加载器包含子加载器, 子加载器也会被调用并加载数据
    * @params:
    *     parentSelectValue: object, 父级加载器的当前值
    */
    loadData:function(parentSelectValue)
    {
        $import('util.Ajax.Request');
        var loaderOptions = this.loaderOptions;

```

```

var url = loaderOptions.url;
var _this = this;
var requestOptions = {
    method:loaderOptions.method || 'GET',
    headers:loaderOptions.headers || {},
    parameters:(loaderOptions.parameters || "") + '&selectname=' + __this.selectElement.name +
    '&parentselectvalue=' + (parentSelectValue || ""),
    onSuccess:function(request)
    {
        var contentType = request.getResponseHeader('Content-Type');
        if(contentType.indexOf('text/xml') != -1)
        {
            _this.buildOptionsFromXML(request.responseXML,true);
        }
        else
        {
            _this.buildOptionsFromJSON(request.responseText,true);
        }
    }
};
if(loaderOptions.onLoading)
{
    loaderOptions.onLoading(this);
}
new util.Ajax.Request(url,requestOptions);
this.selectElement.disabled = true;
},
/*
 * @description: 根据 json 格式的数据构建选项
 * @params:
 *     jsonData: string, json 格式的字符串
 *     isRebuild: 是否重建所有选项, 如果为 true, 则会移除已经存在的所有选项
 */
buildOptionsFromJSON:function(jsonData,isRebuild)
{
    try
    {
        var data = eval('(' + jsonData + ')'), dataItem;
        var selectElement = this.selectElement;
        if(isRebuild)
        {
            selectElement.options.length = 0;
        }
        for(var i = 0, l = data.length; i < l; i++)
        {
            dataItem = data[i];
            selectElement.options[selectElement.options.length] = new
Option(dataItem.name,dataItem.value);
            if(dataItem.selected)
            {
                selectElement.options[selectElement.options.length - 1].selected = true;
            }
        }
    }
}

```

```

        }
    }
    if(this.loaderOptions.onLoadingComplete)
    {
        this.loaderOptions.onLoadingComplete(this);
    }
    this.callSubLoaders();
    this.selectElement.disabled = false;
}
catch(e)
{
    this.selectElement.disabled = false;
    throw new Error('构建项目失败, 数据不是指定的 JSON 格式');
}
},
/*
 * @description: 根据 xml 格式的数据构建选项
 * @params:
 *      xmlDoc: XML DOM, xml document
 *      isRebuild: 是否重建所有选项, 如果为 true, 则会移除已经存在的所有选项
 */
buildOptionsFromXML:function(xmlDoc,isRebuild)
{
    try
    {
        var data = xmlDoc.documentElement , dataItem;
        var selectElement = this.selectElement;
        if(isRebuild)
        {
            selectElement.options.length = 0;
        }
        for(var i = 0,l = data.childNodes.length; i < l; i++)
        {
            dataItem = data.childNodes[i];
            selectElement.options[selectElement.options.length] = new
Option(dataItem.firstChild.nodeValue,dataItem.getAttribute('value') || "");
            if(dataItem.getAttribute('selected'))
            {
                selectElement.options[selectElement.options.length - 1].selected = true;
            }
        }
        if(this.loaderOptions.onLoadingComplete)
        {
            this.loaderOptions.onLoadingComplete(this);
        }
        this.callSubLoaders();
        this.selectElement.disabled = false;
    }
    catch(e)
    {

```

```

        this.selectElement.disabled = false;
        throw new Error('构建项目失败, 数据不是指定的 XML 格式');
    }
},
/*
 * @description: 调用子加载器加载数据
 */
callSubLoaders:function()
{
    var subLoaders = this.subLoaders , subLoader , loaderName;
    var value = this.selectElement.value;
    for(loaderName in subLoaders)
    {
        subLoader = subLoaders[loaderName];
        subLoader.loadData(value);
    }
}
}

```

## 18.4 测 试

在实现 SelectLoader 类后, 现在编写测试用例对该类的功能进行测试。测试用例包含 3 个文件, 其中 index.html 代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>selector</title>
<script type="text/javascript" src="../jsvm2/jsre.js"></script>
<script type="text/javascript">
$import('util.Ajax.SelectLoader');
window.onload = function()
{
    /*
    * 获取 select 元素
    */
    var level1 = document.getElementById('level1');
    var level2 = document.getElementById('level2');
    var level3 = document.getElementById('level3');
    var level32 = document.getElementById('level3-2');
    /*
    * 设置数据加载选项, 针对 JSON 数据
    */
    var jsonLoaderOptions = {
        url:'json.php',
        onLoading:function(loader){

```

```

        setMsg('开始加载' + loader.selectElement.name + '的数据');
    },
    onLoadingComplete:function(loader)
    {
        setMsg(loader.selectElement.name + '的数据加载完成');
    }
}
/*
 * 信息显示函数，用于输出状态信息到界面上以方便测试
 */
var setMsg = function(msg)
{
    var msgbox = document.getElementById('msgBox');
    msgbox.innerHTML += '<div>' + msg + '</div>';
    msgbox.scrollTop = msgbox.scrollHeight;
}
/*
 * 创建加载器
 */
var loaderLevel1 = new util.Ajax.SelectLoader(level1,jsonLoaderOptions);
var loaderLevel2 = new util.Ajax.SelectLoader(level2,jsonLoaderOptions);
var loaderLevel3 = new util.Ajax.SelectLoader(level3,jsonLoaderOptions);
var loaderLevel32 = new util.Ajax.SelectLoader(level32,jsonLoaderOptions);
/*
 * 设置关联关系
 */
loaderLevel1.addSubLoader(loaderLevel2);
loaderLevel2.addSubLoader(loaderLevel3);
loaderLevel2.addSubLoader(loaderLevel32);
/*
 * 开始加载数据
 */
loaderLevel1.loadData();
/*
 * 获取 select 元素
 */
var xlevel1 = document.getElementById('xlevel1');
var xlevel2 = document.getElementById('xlevel2');
var xlevel3 = document.getElementById('xlevel3');
var xlevel32 = document.getElementById('xlevel3-2');
/*
 * 设置数据加载选项，针对 XML 数据
 */
var xmlLoaderOptions = {
    url:'xml.php',
    onLoading:function(loader){
        setMsg('开始加载' + loader.selectElement.name + '的数据');
    },
    onLoadingComplete:function(loader)
    {
        setMsg(loader.selectElement.name + '的数据加载完成');
    }
}

```



```

    }
}
/*
 * 创建加载器
 */
var xloaderLevel1 = new util.Ajax.SelectLoader(xlevel1,xmlLoaderOptions);
var xloaderLevel2 = new util.Ajax.SelectLoader(xlevel2,xmlLoaderOptions);
var xloaderLevel3 = new util.Ajax.SelectLoader(xlevel3,xmlLoaderOptions);
var xloaderLevel32 = new util.Ajax.SelectLoader(xlevel32,xmlLoaderOptions);
/*
 * 设置关联关系
 */
xloaderLevel1.addSubLoader(xloaderLevel2);
xloaderLevel2.addSubLoader(xloaderLevel3);
xloaderLevel2.addSubLoader(xloaderLevel32);
/*
 * 开始加载数据
 */
xloaderLevel1.loadData();
}
</script>
<style type="text/css">
body {font-size:12px;}
.special {display:block; margin-left:185px; margin-top:10px;}
#msgBox {
    position:absolute;
    width:333px;
    height:383px;
    z-index:1;
    left: 433px;
    top: 11px;
    overflow-y:scroll;
    background-color:#eee;
}
</style>
</head>

<body>
    <div id="msgBox"></div>
    <div>JSON:</div>
    <select id="level1" name='level1'>
    </select>
    <select id="level2" name='level2'>
    </select>
    <select id="level3" name='level3'>
    </select>
    <select id="level3-2" name='level3-2' class="special">
    </select>
    <div>XML:</div>
    <select id="xlevel1" name='xlevel1'>
    </select>

```

```

<select id="xlevel2" name='xlevel2'>
</select>
<select id="xlevel3" name='xlevel3'>
</select>
<select id="xlevel3-2" name='xlevel3-2' class="special">
</select>
</body>
</html>

```

json.php 用来输出 JSON 数据, 代码如下所示。

```

<?
header('Content-Type:text/json;charset=utf-8');

$data = array();

if($_REQUEST['selectname'] == 'level1')
{
    $data[] = array('name'=>'sport','value'=>'sport','selected'=>true);
    $data[] = array('name'=>'IT','value'=>'IT');
}
if($_REQUEST['selectname'] == 'level2')
{
    if($_REQUEST['parentselectvalue'] == 'sport')
    {
        $data[] = array('name'=>'football','value'=>'football','selected'=>true);
        $data[] = array('name'=>'basketball','value'=>'basketball');
    }
    if($_REQUEST['parentselectvalue'] == 'IT')
    {
        $data[] = array('name'=>'HTML','value'=>'HTML','selected'=>true);
        $data[] = array('name'=>'JavaScript','value'=>'JavaScript');
    }
}
if($_REQUEST['selectname'] == 'level3')
{
    if($_REQUEST['parentselectvalue'] == 'football')
    {
        $data[] = array('name'=>'football video','value'=>'football video','selected'=>true);
        $data[] = array('name'=>'football shoes','value'=>'football shoes');
    }
    if($_REQUEST['parentselectvalue'] == 'basketball')
    {
        $data[] = array('name'=>'basketball video','value'=>'basketball video','selected'=>true);
        $data[] = array('name'=>'basketball shoes','value'=>'basketball shoes');
    }
    if($_REQUEST['parentselectvalue'] == 'HTML')
    {
        $data[] = array('name'=>'HTML basic','value'=>'HTML basic','selected'=>true);
        $data[] = array('name'=>'HTML tags','value'=>'HTML tags');
    }
}

```

```

        if($_REQUEST['parentselectvalue'] == 'JavaScript')
        {
            $data[] = array('name'=>'JavaScript basic','value'=>'JavaScript basic','selected'=>true);
            $data[] = array('name'=>'JavaScript OOP','value'=>'JavaScript OOP');
        }
    }
    if($_REQUEST['selectname'] == 'level3-2')
    {
        if($_REQUEST['parentselectvalue'] == 'football')
        {
            $data[] = array('name'=>'Nike football','value'=>'Nike football','selected'=>true);
            $data[] = array('name'=>'Adidas football','value'=>'Adidas football');
        }
        if($_REQUEST['parentselectvalue'] == 'basketball')
        {
            $data[] = array('name'=>'Nike basketball','value'=>'Nike basketball','selected'=>true);
            $data[] = array('name'=>'Adidas basketball','value'=>'Nike basketball');
        }
        if($_REQUEST['parentselectvalue'] == 'HTML')
        {
            $data[] = array('name'=>'Oreily HTML books','value'=>'Oreily HTML books','selected'=>true);
            $data[] = array('name'=>'Broadway HTML books','value'=>'Broadway HTML books');
        }
        if($_REQUEST['parentselectvalue'] == 'JavaScript')
        {
            $data[] = array('name'=>'Oreily JavaScript books','value'=>'Oreily JavaScript books','selected'=>true);
            $data[] = array('name'=>'Broadway JavaScript books','value'=>'Broadway JavaScript books');
        }
    }

    echo json_encode($data);
    ?>

```

xml.php 用来输出 xml 数据，代码如下所示。

```

<?
header('Content-Type:text/xml;charset=utf-8');

$data = '<?xml version="1.0" encoding="utf-8"?>';
$data .= '<options>';

if($_REQUEST['selectname'] == 'xlevel1')
{
    $data .= '<option value="sport" selected="true">sport</option>';
    $data .= '<option value="IT" selected="true">IT</option>';
}
if($_REQUEST['selectname'] == 'xlevel2')
{
    if($_REQUEST['parentselectvalue'] == 'sport')
    {
        $data .= '<option value="football" selected="true">football</option>';
    }
}

```

```

        $data .= '<option value="basketball">basketball</option>';
    }
    if($_REQUEST['parentselectvalue'] == 'IT')
    {
        $data .= '<option value="HTML" selected="true">HTML</option>';
        $data .= '<option value="JavaScript">JavaScript</option>';
    }
}
if($_REQUEST['selectname'] == 'xlevel3')
{
    if($_REQUEST['parentselectvalue'] == 'football')
    {
        $data .= '<option value="football video" selected="true">football video</option>';
        $data .= '<option value="football shoes">football shoes</option>';
    }
    if($_REQUEST['parentselectvalue'] == 'basketball')
    {
        $data .= '<option value="basketball video" selected="true">basketball video</option>';
        $data .= '<option value="basketball shoes">basketball shoes</option>';
    }
    if($_REQUEST['parentselectvalue'] == 'HTML')
    {
        $data .= '<option value="HTML basic" selected="true">HTML basic</option>';
        $data .= '<option value="HTML tags">HTML tags</option>';
    }
    if($_REQUEST['parentselectvalue'] == 'JavaScript')
    {
        $data .= '<option value="JavaScript basic" selected="true">JavaScript basic</option>';
        $data .= '<option value="JavaScript OOP">JavaScript OOP</option>';
    }
}
if($_REQUEST['selectname'] == 'xlevel3-2')
{
    if($_REQUEST['parentselectvalue'] == 'football')
    {
        $data .= '<option value="Nike football" selected="true">Nike football</option>';
        $data .= '<option value="Adidas football">Adidas football</option>';
    }
    if($_REQUEST['parentselectvalue'] == 'basketball')
    {
        $data .= '<option value="Nike basketball" selected="true">Nike basketball</option>';
        $data .= '<option value="Adidas basketball">Adidas basketball</option>';
    }
    if($_REQUEST['parentselectvalue'] == 'HTML')
    {
        $data .= '<option value="Oreily HTML books" selected="true">Oreily HTML books</option>';
        $data .= '<option value="Broadway HTML books">Broadway HTML books</option>';
    }
    if($_REQUEST['parentselectvalue'] == 'JavaScript')
    {
        $data .= '<option value="Oreily JavaScript books" selected="true">Oreily JavaScript books</option>';
    }
}

```

```
$data .= '<option value="Broadway JavaScript books">Broadway JavaScript books</option>';  
}  
}  
$data .= '</options>';  
  
echo $data;  
?>
```

将测试用例与 JSVM 程序包放到 Apache 虚拟目录下，然后通过浏览器访问 index.html，如图 18.2 所示。

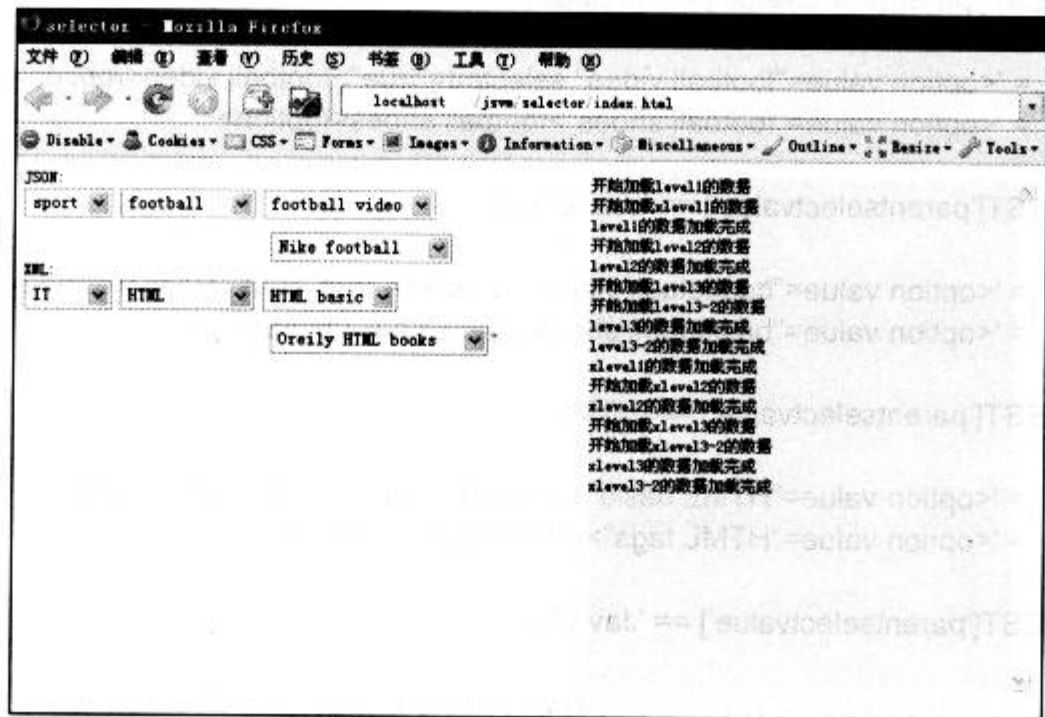


图 18.2 测试

打开页面后，数据自动加载，并在页面右边区域显示了状态信息。改变第一级菜单的选中项，其子孙级的菜单都会自动更新选项，如图 18.3 所示。

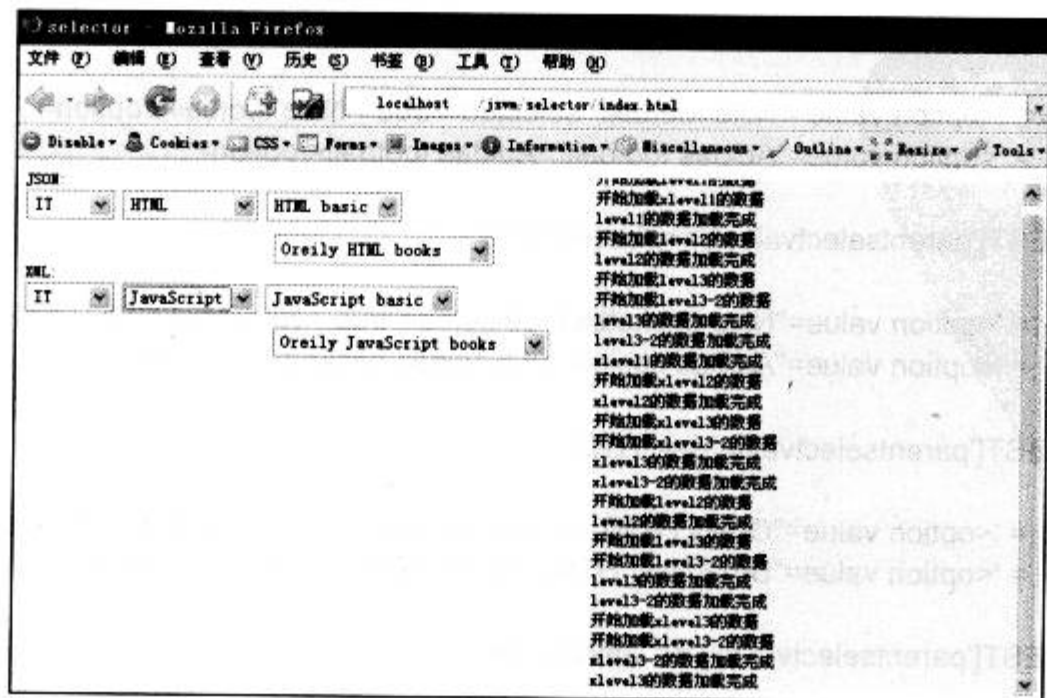


图 18.3 自动更新选项



# 第19章

## Ajax 聊天室

- » 确定需求
- » 基本设计
- » 实例代码
- » 测试

聊天室是经典的 Web 应用程序之一，其作用是给不同的用户提供一个交流的场所。用户在聊天室程序中可以发布信息，其他所有在聊天室中的用户都可以看到其发布的信息，就如同所有用户都在一个个真正室内进行聊天一样。本章将使用 Ajax 技术来实现无刷新的聊天室应用程序。

## 19.1 确定需求

首先需要确定聊天室的需求。

- ☐ 用户登录聊天室。
- ☐ 用户可以设置自己的名字（昵称）。
- ☐ 用户可以发布信息。
- ☐ 用户可以查看其他所有人发布的信息。
- ☐ 用户可以选择信息字体的颜色。
- ☐ 用户可以选择表情图标添加到信息中发布。

## 19.2 基本设计

在确定聊天室的需求后，下面对聊天室系统进行基本的系统设计，包括系统的体系结构、前台界面、前/后台功能模块、数据实体以及数据库的设计等。

### 19.2.1 系统结构

聊天室程序的系统结构如图 19.1 所示。

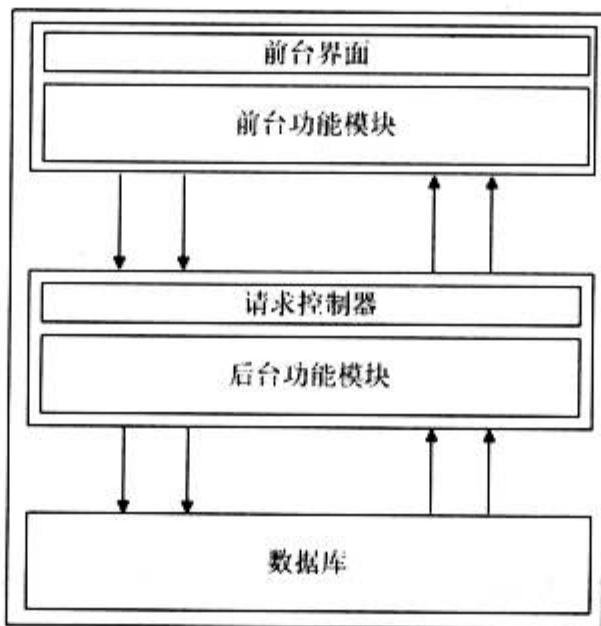


图 19.1 系统结构

其中，数据库负责存储用户和留言数据。程序的服务端程序包括请求控制器和后台功能模块两部分。请求控制器负责处理前台程序发送来的请求并调用后台功能模块进行处理，然后返回处理结果。后台功能模块提供了实体封装和逻辑处理。聊天室的前台部分包括前台界面和前台功能模块。前台界面提供了用户数据的输入输出接口。前台功能模块定义了实现聊天室各功能的脚本代码，其被用户事

件驱动，并与后台的请求控制器进行交互。

### 19.2.2 实体及数据库设计

按照需求，确定了用户及消息两个实体，其包含的属性及实体间的关系如图 19.2 所示。

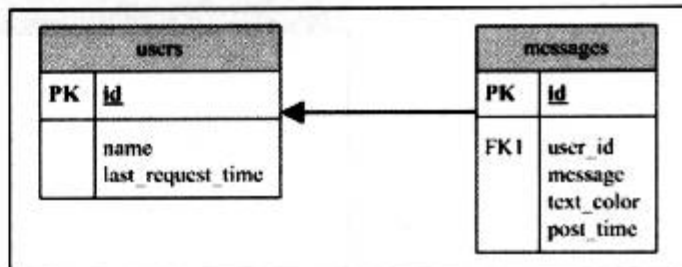


图 19.2 实体

### 19.2.3 后台功能模块

后台功能模块提供实体的封装和相关的逻辑处理，按照需求，该模块由数据库操作类（DB）、用户类（User）、消息类（Message）组成，其设计如图 19.3 所示。

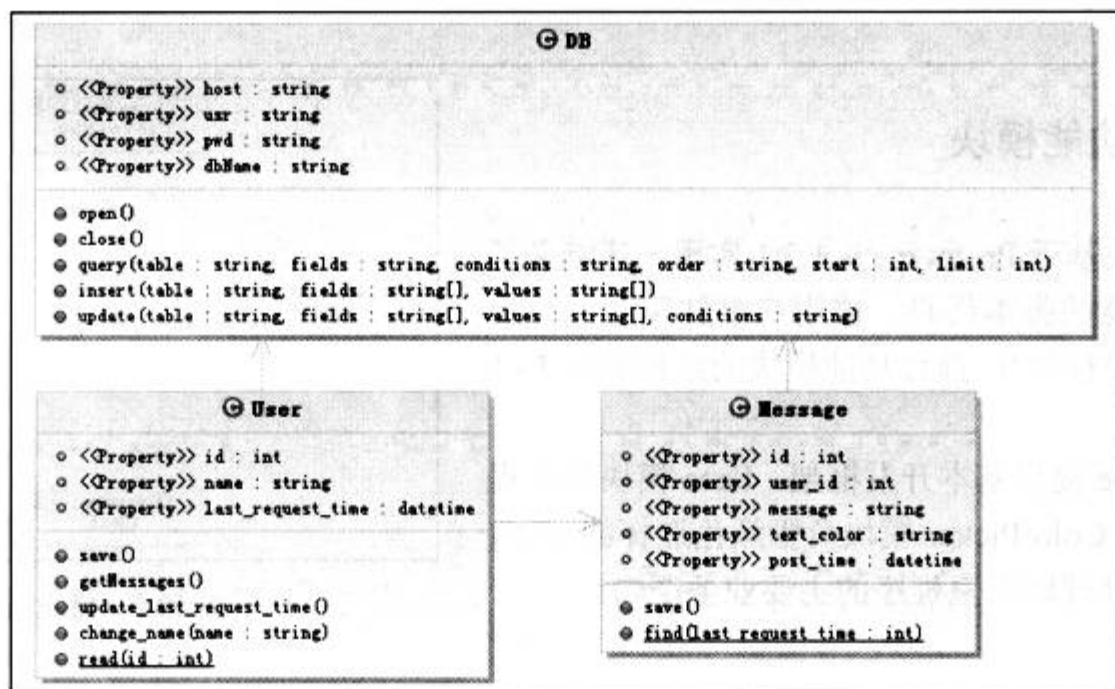


图 19.3 后台功能模块

### 19.2.4 请求控制器

请求控制器负责接收前台程序发送的请求并调用后台功能模块进行处理，其包含的逻辑分支如下所示。

- ☐ saveuser: 保存用户。
- ☐ savemessage: 保存消息。
- ☐ changename: 用户更改昵称。
- ☐ getmessages: 获取消息。

### 19.2.5 前台界面

前台界面的设计如图 19.4 所示。

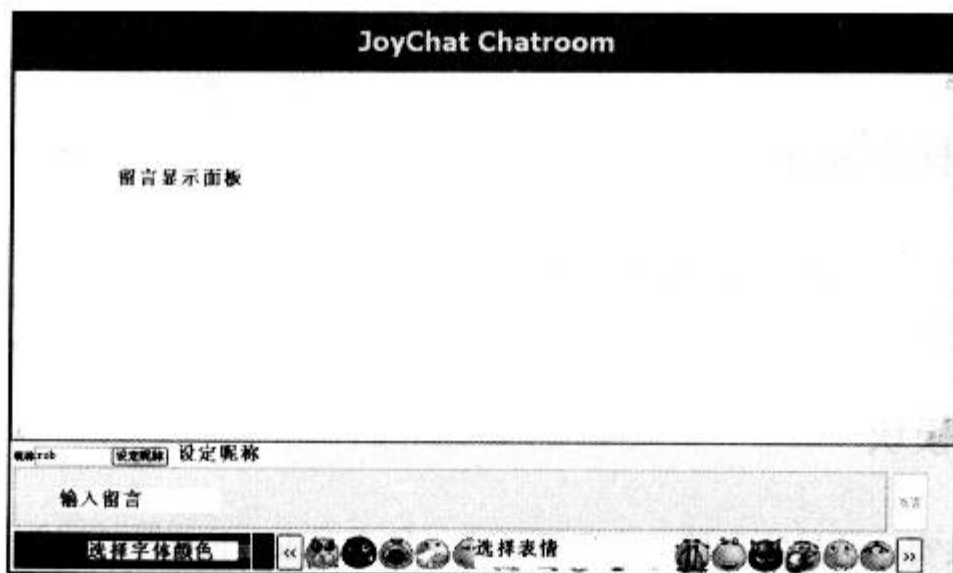


图 19.4 界面设计

### 19.2.6 前台功能模块

前台功能模块基于 Prototype.js 1.5.1 实现，其定义了实现聊天室各功能的脚本代码，被用户事件驱动，与后台的请求控制器进行交互。前台功能模块的结构如图 19.5 所示。

其中 Prototype 提供基本开发框架，Face 模块负责提供表情相关功能，ColorPicker 模块负责颜色选择的功能，Chat 模块则负责处理聊天室程序的主要业务逻辑。

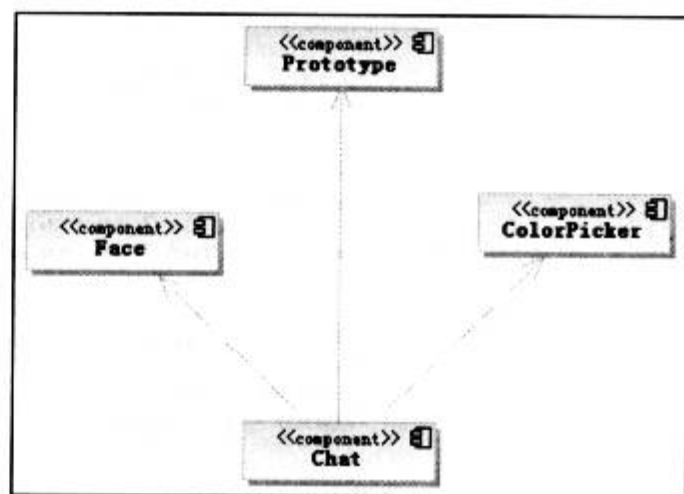


图 19.5 模块结构

#### 1. Face 模块

Face 模块由 Face 类组成，其设计如图 19.6 所示。

#### 2. ColorPicker 模块

ColorPicker 模块由 ColorBar 类和 ColorPicker 类组成。ColorPicker 类的设计如图 19.7 所示。ColorBar 类的设计如图 19.8 所示。



图 19.6 Face 类

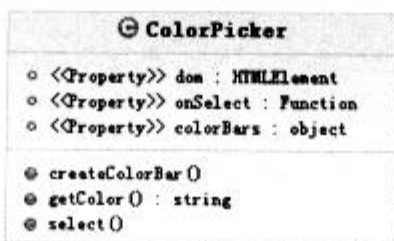


图 19.7 ColorPicker 类

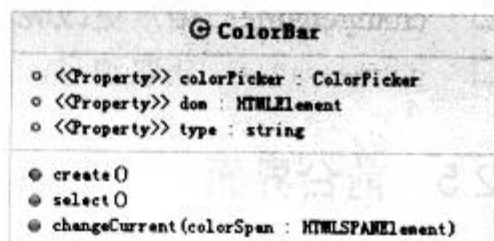


图 19.8 ColorBar 类



### 3. Chat 模块

Chat 模块由 Chat 对象、User 类和 Message 类组成。Chat 对象的设计如图 19.9 所示。

User 类的设计如图 19.10 所示。

Message 类的设计如图 19.11 所示。

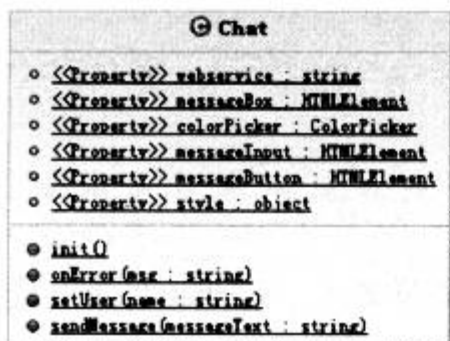


图 19.9 Chat 对象

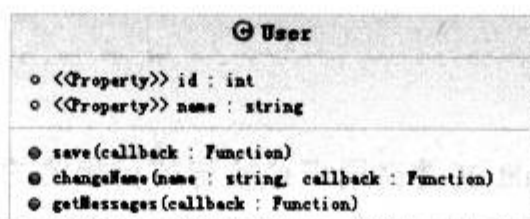


图 19.10 User 类

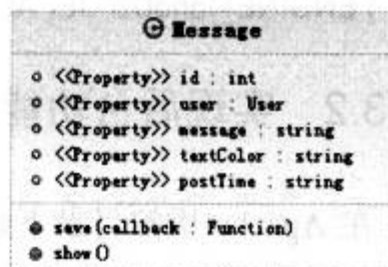


图 19.11 Message 类

## 19.3 实例代码

在完成基本设计后，现在开始对系统各个组成部分进行代码实现，包括数据库、前后台功能模块、界面等。

### 19.3.1 建立数据库

在 MySQL 中建立数据库 joychat，并创建对应的表，使用如下代码。

```

SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
--
-- 数据库: 'joychat'
--
CREATE DATABASE 'joychat' DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
USE 'joychat'
-----
--
-- 表的结构 'messages'
--
CREATE TABLE 'messages' (
  'id' int(10) NOT NULL auto_increment,
  'user_id' int(10) NOT NULL,
  'message' text NOT NULL,
  'text_color' varchar(10) NOT NULL,
  'post_time' datetime NOT NULL,
  PRIMARY KEY ('id')
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1;
--
--表的结构'users'
  
```



```
--
CREATE TABLE 'users' (
  'id' int(10) NOT NULL auto_increment,
  'name' varchar(50) NOT NULL,
  'last_request_time' datetime NOT NULL,
  PRIMARY KEY ('id')
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1;
```

### 19.3.2 实现后台功能模块

在 Apache 虚拟目录下建立 chat 文件夹用来存放与项目相关的程序资源文件。在 chat 目录下建立 service 目录用来存放后台相关的程序文件。在 service 目录下建立 db.class.php 文件存放 DB 类，其代码如下所示。

```
<?
/*
 * @class: DB
 * @description: 数据库操作类
 */
class DB
{
  /*
   * @property: $host
   * @type: string
   * @description: 数据库服务器
   */
  var $host;
  /*
   * @property: $usr
   * @type: string
   * @description: 数据库用户名
   */
  var $usr;
  /*
   * @property: $pwd
   * @type: string
   * @description: 数据库密码
   */
  var $pwd;
  /*
   * @property: $dbName
   * @type: string
   * @description: 数据库名
   */
  var $dbName;
  /*
   * @property: $con
```

```

* @type: mysql connection
* @description:数据库连接实例
*/
var $con;
/*
* @description:构造函数
* @param: $host,string,数据库服务器地址
* @param: $usr,string,数据库用户名
* @param: $pwd,string,数据库密码
* @param: $dbName,string,数据库名
*/
function DB($host='localhost',$usr='root',$pwd='123',$dbName='joychat')
{
    $this->host = $host;
    $this->usr = $usr;
    $this->pwd = $pwd;
    $this->dbName = $dbName;
}
/*
* @description:打开数据库连接
*/
function open()
{
    $this->con = mysql_connect($this->host,$this->usr,$this->pwd);
    mysql_select_db($this->dbName,$this->con);
}
/*
* @description:关闭数据库连接
*/
function close()
{
    mysql_close($this->con);
}
/*
* @description:查询
* @param: $table,string,表名
* @param: $fields,string,字段
* @param: $conditions,string,查询条件
* @param: $order,string,排序规则
* @param: $start,int,结果集起点
* @param: $limit,int,结果集数量
* @return: objects,结果对象集
*/
function query($table,$fields="",$conditions="",$order="",$start=0,$limit=0)
{
    $this->open();
    $sql = 'select '.$fields.' from '.$table;
    if(!empty($conditions))
    {

```

```

        $sql .= ' where '.$conditions;
    }
    if(!empty($order))
    {
        $sql .= ' order by '.$order;
    }
    if($start != 0)
    {
        $sql .= ' limit '.$start.','.$limit;
    }
    else if($limit != 0)
    {
        $sql .= ' limit '.$limit;
    }
    $rs = mysql_query($sql,$this->con);
    $objects = array();
    while($row = mysql_fetch_object($rs))
    {
        $objects[] = $row;
    }
    $this->close();
    return $objects;
}
/*
 * @description: 插入数据
 * @param: $table,string,表名
 * @param: $fields,array,字段
 * @param: $values,array,字段值
 */
function insert($table,$fields,$values)
{
    $this->open();
    for($i = 0; $i < sizeof($fields); $i++)
    {
        $fieldsString .= $fields[$i];
        $valuesString .= '"'.$values[$i].'"';
        if($i < sizeof($fields) - 1)
        {
            $fieldsString .= ',';
            $valuesString .= ',';
        }
    }
    $sql = 'insert into '.$table.' ('.$fieldsString.') values('.$valuesString.')';
    mysql_query($sql,$this->con);
    $insert_id = mysql_insert_id($this->con);
    $this->close();
    return $insert_id;
}
/*

```

```

    * @description:更新
    * @param: $table,string,表名
    * @param: $fields,array,字段
    * @param: $values,array,字段值
    * @param: $conditions,string,更新条件
    */
function update($table,$fields,$values,$conditions)
{
    $this->open();
    $sql = 'update '.$table.' set ';
    for($i = 0; $i < sizeof($fields); $i++)
    {
        $sql .= $fields[$i] . '=' . '"'.$values[$i].'"';
        if($i < sizeof($fields) - 1)
        {
            $sql .= ',';
        }
    }
    $sql .= 'where ' . $conditions;
    mysql_query($sql,$this->con);
    $this->close();
}
}

?>

```

建立 user.class.php 存放 User 类，其代码如下所示。

```

<?
/*
 * @class: User
 * @description: 用户类
 */
class User
{
    /*
     * @property: $id
     * @type: int
     * @description:用户 id
     */
    var $id;
    /*
     * @property: $name
     * @type: string
     * @description:用户昵称
     */
    var $name;
    /*
     * @property: $last_request_time
     * @type: datetime

```

```

    * @description:最后请求时间
    */
    var $last_request_time;
    /*
    * @description: 构造函数
    * @param: $id,int,用户 id
    * @param: $name,string,用户昵称
    * @param: $last_request_time,datetime,最后请求时间
    */
    function User($id=NULL,$name="", $last_request_time="")
    {
        $this->id = $id;
        $this->name = $name;
        $this->last_request_time = $last_request_time;
        if(empty($this->last_request_time))
        {
            $this->last_request_time = date("Y-m-d H:i:s");
        }
    }
    /*
    * @description: 保存用户数据
    */
    function save()
    {
        $db = new DB();
        $this->id =
        $db->insert('users',array('name','last_request_time'),array($this->name,$this->last_request_time));
    }
    /*
    * @description: 更新最后请求时间
    */
    function update_last_request_time()
    {
        $db = new DB();
        $last_request_time = date("Y-m-d H:i:s");
        $db->update('users',array('last_request_time'),array($last_request_time),'id='.$this->id);
        $this->last_request_time = $last_request_time;
    }
    /*
    * @description: 更改用户昵称
    * @param: $name,string,用户昵称
    */
    function change_name($name)
    {
        $db = new DB();
        $db->update('users',array('name'),array($name),'id='.$this->id);
        $this->name = $name;
    }
    /*

```



```

    * @description: 获取消息
    * @return: array,消息对象集合
    */
function getMessages()
{
    $messages = Message::find($this->last_request_time);
    $this->update_last_request_time();
    return $messages;
}
/*
    * @description: 读取指定 id 的用户数据
    * @param: $id,int,用户 id
    * @return: User,用户对象
    */
static function read($id)
{
    $db = new DB();
    $datas = $db->query('users','*', 'id='.$id);
    if(sizeof($datas) > 0)
    {
        $user_data = $datas[0];
        $user = new User($id,$user_data->name,$user_data->last_request_time);
        return $user;
    }
    else
    {
        return null;
    }
}
}
?>

```

建立 message.class.php 文件存放 Message 类，其代码如下所示。

```

<?
/*
    * @class: Message
    * @description: 消息类
    */
class Message
{
    /*
        * @property: $id
        * @type:int
        * @description:消息 id
        */
    var $id;
    /*
        * @property: $user
        * @type:User
    */
}

```

```

    * @description:发送消息的用户
    */
    var $user;
    /*
    * @property: $message
    * @type:string
    * @description:消息内容
    */
    var $message;
    /*
    * @property: $text_color
    * @type:string
    * @description:消息文字颜色
    */
    var $text_color;
    /*
    * @property: $post_time
    * @type:datetime
    * @description:消息发布时间
    */
    var $post_time;
    /*
    * @description: 构造函数
    * @param: $id,int,消息 id
    * @param: $user,User,发送消息的用户
    * @param: $message,string,消息内容
    * @param: $text_color,string,消息文字颜色
    * @param: $post_time,datetime,消息发送时间
    */
    function Message($id=NULL,$user=NULL,$message="",$text_color="",$post_time="")
    {
        $this->id = $id;
        $this->user = $user;
        $this->message = $message;
        $this->text_color = $text_color;
        $this->post_time = $post_time;
        if(empty($this->post_time))
        {
            $this->post_time = date('Y-m-d H:i:s');
        }
    }
    /*
    * @description:保存消息
    */
    function save()
    {
        $db = new DB();
        $this->id =
        $db->insert('messages',array('user_id','message','text_color','post_time'),array($this->user->id,$this->message,
        $this->text_color,$this->post_time));
    }

```

```

    }
    /*
    * @description:查找指定时间之后发送的消息
    * @param: $last_request_time,datetime,最后请求时间
    * @return: array,消息对象集合
    */
    static function find($last_request_time)
    {
        $db = new DB();
        $datas = $db->query('messages','id,user_id,message,text_color,post_time','post_time >=
\'".$last_request_time."\','post_time desc');
        $messages = array();
        for($i = 0; $i < sizeof($datas); $i++)
        {
            $data = $datas[$i];
            $user = User::read($data->user_id);
            $messages[] = new
Message($data->id,$user,$data->message,$data->text_color,$data->post_time);
        }
        return $messages;
    }
}
?>

```

### 19.3.3 请求控制器

在 service 目录下建立 index.php 文件存放请求控制器，代码如下所示。

```

<?
/*
* 自动加载所需的类
*/
function _autoload($class)
{
    include(ucfirst($class) . ".class.php");
}
/*
* 请求控制器
*/
switch($_REQUEST['action'])
{
    /*
    * 保存用户
    */
    case 'saveuser':
        $user = new User();
        $user->name = $_REQUEST['name'];
        $user->last_request_time = date('Y-m-d H:i:s');
        $user->save();

```

```

        $message = new Message();
        $message->user = $user;
        $message->message = ' 进入了聊天室';
        $message->text_color = 'red';
        $message->save();
        echo json_encode($user);
        break;
    /*
    * 保存消息
    */
    case 'savemessage':
        $message = new Message();
        $message->message = $_REQUEST['message'];
        $message->text_color = $_REQUEST['text_color'];
        $message->user = new User($_REQUEST['user_id']);
        $message->save();
        echo json_encode($message);
        break;
    /*
    * 更改昵称
    */
    case 'changename':
        $user = User::read($_REQUEST['id']);
        $old_name = $user->name;
        $user->change_name($_REQUEST['name']);
        $message = new Message();
        $message->user = $user;
        $message->message = $old_name.' 将名字更改为了:'.$user->name;
        $message->text_color = 'red';
        $message->save();
        echo 'ok';
        break;
    /*
    * 取得消息
    */
    case 'getmessages':
        $user = User::read($_REQUEST['id']);
        $messages = $user->getMessages();
        echo json_encode($messages);
        break;
    }
    ?>

```

### 19.3.4 界面 HTML 和 CSS 代码

在 chat 目录下建立 index.html 文件存放界面 HTML 代码，其代码如下所示。

```

<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Joy Chat</title>
<link type="text/css" href="css/css.css" rel="stylesheet" />
</head>

<body>
<h1>JoyChat Chatroom</h1>
<div id="messageBox">
</div>
<div id="toolbar">
    <label for="txtName">昵称</label><input type="text" id="txtName" name="txtName" size="10"
maxlength="10" value="请先设定昵称" onfocus="this.select();" /><input type="button" id="btnSetName"
name="btnSetName" value="设定昵称" />
    <div class="clear"></div>
    <textarea id="txtMessage" name="txtMessage"></textarea><input type="button" id="btnSubmit"
name="btnSubmit" value="发言" />
    <div class="clear"></div>
    <div id="colorBox"></div><div id="color"></div>
    <input type="button" id="btnPrevPage" name="btnPrevPage" value="<<" />
    <div id="emotBox"></div>
    <input type="button" id="btnNextPage" name="btnNextPage" value=">>" />
</div>
</body>
</html>

```

在 chat 目录下建立 css 目录存放相关的样式表文件。在 css 目录下建立 css.css 文件，代码如下所示。

```

* {
    margin:0;
    padding:0;
}
body {
    overflow:hidden;
    font-family:Tahoma,"宋体";
    font-size:11px;
}
h1 {
    text-align:center;
    height:60px;
    background-color:#666;
    color:#eee;
    line-height:60px;
    font-size:30px;
}
.clear {
    clear:both;
    height:1px;

```



```
}  
#messageBox {  
    height:400px;  
    border:2px solid #666;  
    overflow:scroll;  
}  
#messageBox div {  
    clear:both;  
}  
#messageBox div span {  
    line-height:20px;  
    float:left;  
    margin-right:5px;  
}  
span.name {  
    width:100px;  
    background-color:#eee;  
    text-align:center;  
    font-weight:bold;  
}  
span.time {  
    font-size:10px;  
}  
#toolbar {  
    height:150px;  
    background-color:#eee;  
    padding:5px 0 0 5px;  
}  
#txtMessage {  
    height:60px;  
    width:940px;  
    float:left;  
}  
#btnSubmit {  
    height:64px;  
    width:40px;  
    line-height:60px;  
    float:left;  
    margin-left:5px;  
}  
#colorBox {  
    margin-top:5px;  
    float:left;  
}  
#color {  
    float:left;  
    width:20px;  
    height:36px;  
    background-color:#000;  
    margin:5px 3px 0 3px;  
    border:1px solid #000;
```

```

        cursor:pointer;
    }
    #emotBox {
        float:left;
        min-width:640px;
    }
    #emotBox img,#messageBox img{
        width:40px;
        height:40px;
    }
    #emotBox img {
        margin-top:5px;
    }
    #emotBox img:hover {
        border:1px solid #666;
    }
    #btnPrevPage,#btnNextPage {
        margin-top:5px;
        float:left;
        height:40px;
        width:30px;
        text-align:center;
    }
    #btnClear {
    }
}

```

### 19.3.5 前台功能模块

在 chat 目录下建立 script 目录用于存放相关的 JavaScript 文件。

#### 1. Prototype 模块

将 prototype.js 复制到 script 目录下，代码略。

#### 2. Face 模块

在 chat 目录下建立 Face.js 文件，存放 Face 模块，代码如下所示。

```

/*
 * @Description:Face 模块
 */
/*
 * @class: Face
 * @description: Face 类
 */
var Face = Class.create();
Face.prototype = {
    /*
     * @description:构造函数
     * @param: name,string,表情名称
     * @param: facePic,string,表情图标地址
    */
}

```

```
* @param: onSelect,Function,在表情被选择时调用的回调函数
* @param: onLoad,Function,在表情图标加载完成时调用的回调函数
*/
initialize:function(name,facePic,onSelect,onLoaded)
{
    /*
    * 将数据写入到对应的属性中
    */
    this.name = name;
    this.facePic = facePic;
    this.onSelect = onSelect;
    this.onLoaded = onLoaded;
    /*
    * 创建 Image 对象保存表情图标
    */
    this.dom = new Image();
    if(onLoaded)
    {
        this.dom.onload = this.onLoaded.bindAsEventListener(this);
    }
    this.dom.src = this.facePic;
    $(this.dom).setStyle({
        'cursor':'pointer'
    });
    /*
    * 为图标对象注册 click 事件句柄
    */
    Event.observe(this.dom,'click',this.select.bind(this));
    /*
    * 将当前实例保存到静态属性 instances 中
    */
    Face.instances[this.name] = this;
},
/*
* @description:获取 HTML 表示
*/
getHTML:function()
{
    var html = '';
    return html;
},
/*
* @description:选择表情图标时被调用的方法
*/
select:function()
{
    if(this.onSelect)
    {
        this.onSelect(this);
    }
}
```

```

}
/*
 * @property:instance
 * @type:object
 * @description:静态属性, 保存所有 Face 实例对象
 */
Face.instances = {};
/*
 * @property:pageSize
 * @type:int
 * @description:静态属性, 每页显示的表情数
 */
Face.pageSize = 6;
/*
 * @property:currentPage
 * @type:int
 * @description:静态属性, 当前页数
 */
Face.currentPage = 0;
/*
 * @property:totalPage
 * @type:int
 * @description:静态属性, 总页数
 */
Face.totalPage = 0;
/*
 * @property:dom
 * @type:HTMLElement
 * @description:静态属性, 用于呈现表情图标的面页元素容器
 */
Face.dom = null;
/*
 * @description:静态方法, 显示所有表情图标到页面元素容器中
 */
Face.show = function()
{
    if(!Face.dom)
    {
        throw new Error('Face 还未绑定到容器对象');
    }
    /*
     * 清空容器内容
     */
    Face.dom.innerHTML = "";
    /*
     * 获取需要显示的 Face 对象并显示到容器中
     */
    var faces = Object.values(Face.instances);
    Face.totalPage = Math.ceil((faces.length) / Face.pageSize) - 1;
    if(Face.currentPage > Face.totalPage)
    {

```

```

        return;
    }
    var start = Face.currentPage * Face.pageSize;
    var end = start + Face.pageSize;
    var currentFaces = faces.slice(start, end);
    currentFaces.each(function(face) {
        Face.dom.appendChild(face.dom);
    });
}
/*
 * @description: 静态方法, 显示上一页的表情图标
 */
Face.prevPage = function()
{
    Face.currentPage--;
    if(Face.currentPage < 0)
    {
        Face.currentPage = 0;
    }
    Face.show();
}
/*
 * @description: 静态方法, 显示下一页的表情图标
 */
Face.nextPage = function()
{
    Face.currentPage++;
    if(Face.currentPage > Face.totalPage)
    {
        Face.currentPage = Face.totalPage;
    }
    Face.show();
}

```

### 3. ColorPicker 模块

在 script 目录下建立 ColorPicker.js 文件用于存放 ColorPicker 模块相关代码, 其代码如下所示。

```

/*
 * @description: ColorPicker 模块
 */
/*
 * @class: ColorPicker
 * @description: 拾色器类
 */
var ColorPicker = Class.create();
ColorPicker.prototype = {
    /*
     * @description: 构造函数
     * @param: dom, HTMLElement, 存放拾色器的页面元素容器
     * @param: onSelect, Function, 在用户选取颜色后调用的回调函数
    */

```



```

    */
    initialize:function(dom,onSelect)
    {
        /*
        * 填充数据到对应的属性中
        */
        this.dom = dom;
        this.onSelect = onSelect;
        this.colorBars = {};
        /*
        * 创建色条对象
        */
        this.createColorBar();
    },
    /*
    * @description:创建 R、G、B 3 个色条对象
    */
    createColorBar:function()
    {
        var colorBars = this.colorBars,dom = this.dom;
        colorBars['R'] = new ColorBar(this,document.createElement('div'),'R');
        colorBars['G'] = new ColorBar(this,document.createElement('div'),'G');
        colorBars['B'] = new ColorBar(this,document.createElement('div'),'B');
        dom.appendChild(colorBars['R'].dom);
        dom.appendChild(colorBars['G'].dom);
        dom.appendChild(colorBars['B'].dom);
    },
    /*
    * @description: 获取当前颜色代码
    */
    getColor:function()
    {
        var colorBars = this.colorBars;
        var color =
        [colorBars['R'].current.color['R'],colorBars['G'].current.color['G'],colorBars['B'].current.color['B']];
        return '#' + color.invoke('toColorPart').join("");
    },
    /*
    * @description: 在用户选择颜色时调用的方法
    */
    select:function()
    {
        if(this.onSelect)this.onSelect(this.getColor());
    }
}
/*
* @class: ColorBar
* @description: 色条类
*/
var ColorBar = Class.create();

```

```

ColorBar.prototype = {
  /*
   * @description: 构造函数
   * @param: colorPicker,ColorPicker,所属的 ColorPicker 对象
   * @param: dom,HTMLElement,用于存放色条的页面元素容器
   * @param: type,string,色条的类型, R、G、B 为 3 个可供选择的值
   */
  initialize:function(colorPicker,dm,type)
  {
    /*
     * 填充数据到对应的属性中
     */
    this.colorPicker = colorPicker;
    this.dom = dm;
    this.type = type.toUpperCase();
    /*
     * 创建色条
     */
    this.create();
  },
  /*
   * @description: 创建色条
   */
  create:function()
  {
    var colorObject = {'R':0,'G':0,'B':0},dm = this.dom,type = this.type,span,color;
    /*
     * 清空容器内容并设置其样式
     */
    dm.innerHTML = "";
    $(dm).setStyle({
      'display':'block',
      'float':'left',
      'clear':'both',
      'marginBottom':'1px'
    });
    var _this = this;
    /*
     * 创建每个颜色单位
     */
    $R(0,255).each(function(colorValue)
    {
      colorObject[type] = colorValue;
      span = document.createElement('span');
      color = '#' + [colorObject['R'],colorObject['G'],colorObject['B']].invoke('toColorPart').join("");
      $(span).setStyle({
        'backgroundColor':color,
        'width':'1px',
        'height':'12px',
        'float':'left',

```

```

        'cursor':'pointer'
    });
    span.color = {};
    Object.extend(span.color,colorObject);
    Event.observe(span,'mousedown',_this.select.bindAsEventListener(_this));
    if(colorValue == 0)
    {
        _this.changeCurrent(span);
    }
    dom.appendChild(span);
    });
},
/*
 * @description: 选择颜色
 * @param: evt,Event,事件对象
 */
select:function(evt)
{
    var colorSpan = Event.element(evt);
    this.changeCurrent(colorSpan);
    this.colorPicker.select();
},
/*
 * @description: 切换当前选中的颜色
 */
changeCurrent:function(colorSpan)
{
    if(this.current)
    {
        var colorObject = this.current.color;
        var color = '#' + [colorObject['R'],colorObject['G'],colorObject['B']].invoke('toColorPart').join("");
        $(this.current).setStyle({
            'backgroundColor':color
        });
    }
    this.current = colorSpan;
    $(colorSpan).setStyle({
        'backgroundColor':'#fff'
    });
}
}

```

#### 4. chat 模块

在 script 目录下建立 Chat.js 文件用于存放 chat 模块相关代码，其代码如下所示。

```

/*
 * @description: Chat 模块
 */
/*
 * @description: Chat 对象，主控制器

```

```

*/
var Chat = {
    /*
    * @property:webservice
    * @type:string
    * @description:webservice 地址
    */
    webservice:'service/',
    /*
    * @property:messageBox,
    * @type:HTMLElement
    * @description:用于显示消息的页面容器
    */
    messageBox:null,
    /*
    * @property:colorPicker
    * @type:ColorPicker
    * @description:拾色器对象
    */
    colorPicker:null,
    /*
    * @property:messageInput
    * @type:HTMLElement
    * @description:用于用户输入消息的表单控件
    */
    messageInput:null,
    /*
    * @property:messageButton
    * @type:HTMLElement
    * @description:用于用户发送消息的按钮对象
    */
    messageButton:null,
    /*
    * @property:style
    * @type:object
    * @description:消息的样式，对应 css 类名
    */
    style:{
        nameStyle:'name',           //昵称样式
        timeStyle:'time',           //时间样式
        messageStyle:'message'      //消息内容样式
    },
    /*
    * @description:在程序发生错误时，用于输出错误信息的方法
    * @param:msg,string,错误信息
    */
    onError:function(msg)
    {
        alert(msg);
    },
    /*

```

```

    * @description:初始化
    */
    init:function()
    {
        Chat.messageInput.disabled = false;
        Chat.messageButton.disabled = false;
        Chat.user.getMessages();
    },
    /*
    * @description:设置当前用户
    * @param:name,string,昵称
    */
    setUser:function(name)
    {
        if(!Chat.user)
        {
            /*
            * 如果用户不存在,则创建新用户并初始化 Chat
            */
            Chat.user = new User();
            Chat.user.name = name;
            Chat.user.save(Chat.init.bind(Chat));
        }
        else
        {
            /*
            * 如果用户已经存在,则修改用户的昵称
            */
            if(name != Chat.user.name)
            {
                Chat.user.changeName(name);
            }
        }
    },
    /*
    * @description:发送消息
    * @param:messageText,string,消息内容
    */
    sendMessage:function(messageText)
    {
        /*
        * 过滤 HTML 标签并替换表情标签为对应的 HTML 字符串
        */
        var messageText = messageText.escapeHTML(),faceName,face;
        while(faceName = messageText.replace(/^(\n|\r|.)*\[(e\d+)\](\n|\r|.)*$/im,'$2'))
        {
            if(/^e\d+$/i.test(faceName))
            {
                face = Face.instances[faceName];
                messageText = messageText.replace('['+faceName+']',face.getHTML());
            }
        }
    }
}

```



```

        else
        {
            break;
        }
    }
    /*
    * 创建 Message 对象实例并保存消息
    */
    var message = new Message();
    message.user = Chat.user;
    message.message = messageText;
    message.textColor = Chat.colorPicker.getColor();
    message.save(function()
    {
        Chat.messageInput.value = "";
    });
}
/*
* @class:User
* @description:用户类
*/
var User = Class.create();
User.prototype = {
    /*
    * @description:构造函数
    * @param:id,int,用户 id
    * @param:name,string,用户昵称
    */
    initialize:function(id,name)
    {
        this.id = id;
        this.name = name;
    },
    /*
    * @description:保存用户数据
    * @param:callback,Function,在保存成功后调用的回调函数
    */
    save:function(callback)
    {
        var _this = this;
        /*
        * 制造 Ajax 请求与后台请求控制器交互
        */
        new Ajax.Request(Chat.webservice,{
            method:'post',
            parameters:{
                'action':'saveuser',
                'name':_this.name
            },
            onSuccess:function(xmlhttprequest)

```

```

    {
        /*
         * 成功返回时填充 id 属性
         */
        var data = eval('(' + xmlhttprequest.responseText + ')');
        _this.id = data.id;
        if(callback)
        {
            callback(xmlhttprequest);
        }
    },
    onFailure:function()
    {
        /*
         * 输出错误信息
         */
        Chat.onError('用户保存失败');
    }
});

changeName:function(name,callback)
{
    var _this = this;
    new Ajax.Request(Chat.webservice,{
        method:'post',
        parameters:{
            'action':'changenname',
            'id':_this.id,
            'name':name
        },
        onSuccess:function(xmlhttprequest)
        {
            _this.name = name;
            if(callback)
            {
                callback(xmlhttprequest);
            }
        }
    });
},
/*
 * @description:获取消息
 * @param:callback,Function,在每次成功获取消息后被调用的回调函数
 */
getMessages:function(callback)
{
    var _this = this;
    /*
     * 制造 Ajax 请求与后台请求控制器交互以获取数据
     */
    new Ajax.Request(Chat.webservice,{

```

```

        method:'post',
        parameters:{
            'action':'getmessages',
            'id':_this.id
        },
        onSuccess:function(xmlhttprequest)
        {
            /*
             * 成功返回时利用返回的数据创建 Message 对象，并显示到页面中
             */
            var datas = eval('(' + xmlhttprequest.responseText + ')'),message,user;
            datas.each(function(data){
                user = new User(data.user.id,data.user.name);
                message = new
                Message(data.id,user,data.message,data.text_color,data.post_time);
                message.show();
            });
            setTimeout(function(){_this.getMessages(callback);},800);
        },
        onFailure:function()
        {
            /*
             * 输出错误信息
             */
            Chat.onError('信息读取失败');
        }
    });
}
}
/*
 * @class:Message
 * @description:消息类
 */
var Message = Class.create();
Message.prototype = {
    /*
     * @description:构造函数
     * @param:id,int,消息 id
     * @param:user,User,消息所属的用户对象
     * @param:message,string,消息内容
     * @param:textColor,string,文字颜色代码
     * @param:postTime,string,消息发布时间的字符串表示
     */
    initialize:function(id,user,message,textColor,postTime)
    {
        this.id = id;
        this.user = user;
        this.message = message;
        this.textColor = textColor;
    }
}

```

```

        this.postTime = postTime;
    },
    /*
    * @description:保存消息
    * @param:callback,Function,在消息保存成功时调用的回调函数
    */
    save:function(callback)
    {
        var _this = this;
        /*
        * 制造 Ajax 请求与后台请求控制器交互以保存消息数据
        */
        new Ajax.Request(Chat.webservice,{
            method:'post',
            parameters:{
                'action':'savemessage',
                'user_id':_this.user.id,
                'message':_this.message,
                'text_color':_this.text_color
            },
            onSuccess:function(xmlhttprequest)
            {
                /*
                * 成功返回时填充 id 和 postTime 属性
                */
                var data = eval('(' + xmlhttprequest.responseText + ')');
                _this.id = data.id;
                _this.postTime = data.post_time;
                if(callback)
                {
                    callback(xmlhttprequest);
                }
            },
            onFailure:function()
            {
                /*
                * 输出错误信息
                */
                Chat.onError('信息保存失败');
            }
        });
    },
    /*
    * @description:将消息显示到页面中
    */
    show:function()
    {
        /*
        * 创建用于显示消息的 HTML 对象并添加到页面消息容器中
        */
    }
}

```

```

var messageBox = $(Chat.messageBox);
var divMessage = document.createElement('div');
var nameSpan = document.createElement('span');
nameSpan.innerHTML = this.user.name;
nameSpan.className = Chat.style.nameStyle;
var timeSpan = document.createElement('span');
timeSpan.innerHTML = this.postTime;
timeSpan.className = Chat.style.timeStyle;
var messageSpan = document.createElement('span');
messageSpan.innerHTML = this.message;
messageSpan.className = Chat.style.messageStyle;
messageSpan.style.color = this.textColor;
divMessage.appendChild(nameSpan);
divMessage.appendChild(timeSpan);
divMessage.appendChild(messageSpan);
messageBox.appendChild(divMessage);
/*
 * 将滚动条滚动到最下面以便于查看最新显示的消息
 */
messageBox.scrollTop = messageBox.scrollHeight;
}
}

```

### 19.3.6 加入表情图标

在 chat 目录下建立 emoticons 目录，并将准备好的 72 个表情图标放到目录下，表情图标的命名为 e1~e72，如图 19.12 所示。

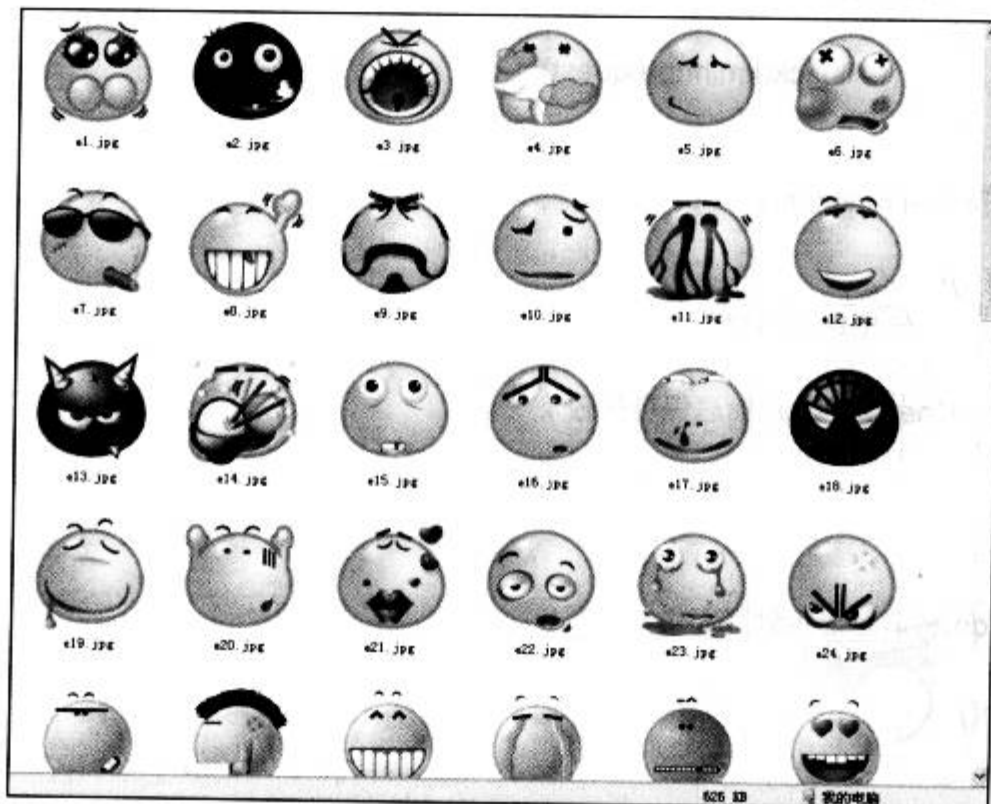


图 19.12 放置表情图标



### 19.3.7 整合程序

修改 index.html 的代码, 引入需要的 JavaScript 文件, 并初始化一些参数和注册事件句柄, 代码如下所示。

```
<!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>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Joy Chat</title>
<link type="text/css" href="css/css.css" rel="stylesheet" />
<script type="text/javascript" src="script/prototype.js"></script>
<script type="text/javascript" src="script/ColorPicker.js"></script>
<script type="text/javascript" src="script/Face.js"></script>
<script type="text/javascript" src="script/Chat.js"></script>
<script type="text/javascript">
/*
 * 创建 Face 对象,缓存表情图标
 */
parseInt(72).times(function(i){
    new Face('e' + (i + 1), 'emoicons/e' + (i + 1) + '.jpg', function(face){
        $('#txtMessage').value += '[' + face.name + '];
    });
});
/*
 * 初始化应用程序
 */
function init()
{
    /*
     * 禁用#txtMessage 和#btnSubmit
     */
    $('#txtMessage').disabled = true;
    $('#btnSubmit').disabled = true;
    /*
     * 创建拾色器并显示到#colorBox 中
     */
    var colorPicker = new ColorPicker($('#colorBox'), function(color){
        $('#color').setStyle({
            'backgroundColor': color
        });
    });
    /*
     * 注册#color 的 click 事件句柄,选择颜色
     */
    Event.observe($('#color'), 'click', function(){
```

```

        $('txtMessage').setStyle({
            'color': $('color').style.backgroundColor
        });
    });
    /*
    * 初始化 Chat 对象的相关参数
    */
    Chat.colorPicker = colorPicker;
    Chat.messageBox = $('messageBox');
    Chat.messageInput = $('txtMessage');
    Chat.messageButton = $('btnSubmit');
    /*
    * 初始化 Face 对象并显示表情图标到#emotBox 中
    */
    Face.dom = $('emotBox');
    Face.pageSize = 16;
    Face.show();
    /*
    * 使用#btnPrevPage 和#btnNextPage 作为表情图标翻页的控制按钮
    */
    $('btnPrevPage').onclick = Face.prevPage.bind(Face);
    $('btnNextPage').onclick = Face.nextPage.bind(Face);
    /*
    * 注册#btnSetName 的 click 事件句柄, 用于处理创建用户以及更改昵称
    */
    $('btnSetName').onclick = function()
    {
        var name = $F('txtName');
        Chat.setUser(name);
    }
    /*
    * 注册#btnSubmit 的 click 事件句柄, 用于发送消息
    */
    $('btnSubmit').onclick = function()
    {
        if($F('txtMessage') == "")
        {
            alert('请输入消息内容');
            return;
        }
        Chat.sendMessage($F('txtMessage'));
    }
    /*
    * 在当前页面加载完成后初始化聊天室应用程序
    */
    Event.observe(window, 'load', init);
</script>
</head>

```

```

<body>
<h1>JoyChat Chatroom</h1>
<div id="messageBox">
</div>
<div id="toolbar">
    <label for="txtName">昵称</label><input type="text" id="txtName" name="txtName" size="10"
maxlength="10" value="请先设定昵称" onfocus="this.select();" /><input type="button" id="btnSetName"
name="btnSetName" value="设定昵称" />
    <div class="clear"></div>
    <textarea id="txtMessage" name="txtMessage"></textarea><input type="button" id="btnSubmit"
name="btnSubmit" value="发言" />
    <div class="clear"></div>
    <div id="colorBox"></div><div id="color"></div>
    <input type="button" id="btnPrevPage" name="btnPrevPage" value="<<" />
    <div id="emotBox"></div>
    <input type="button" id="btnNextPage" name="btnNextPage" value=">>" />
</div>
</body>
</html>

```

## 19.4 测 试

在完成系统的代码实现后就可以进行聊天室程序的测试了。

(1) 使用浏览器访问 index.html 所在的 URL，界面效果如图 19.13 所示。

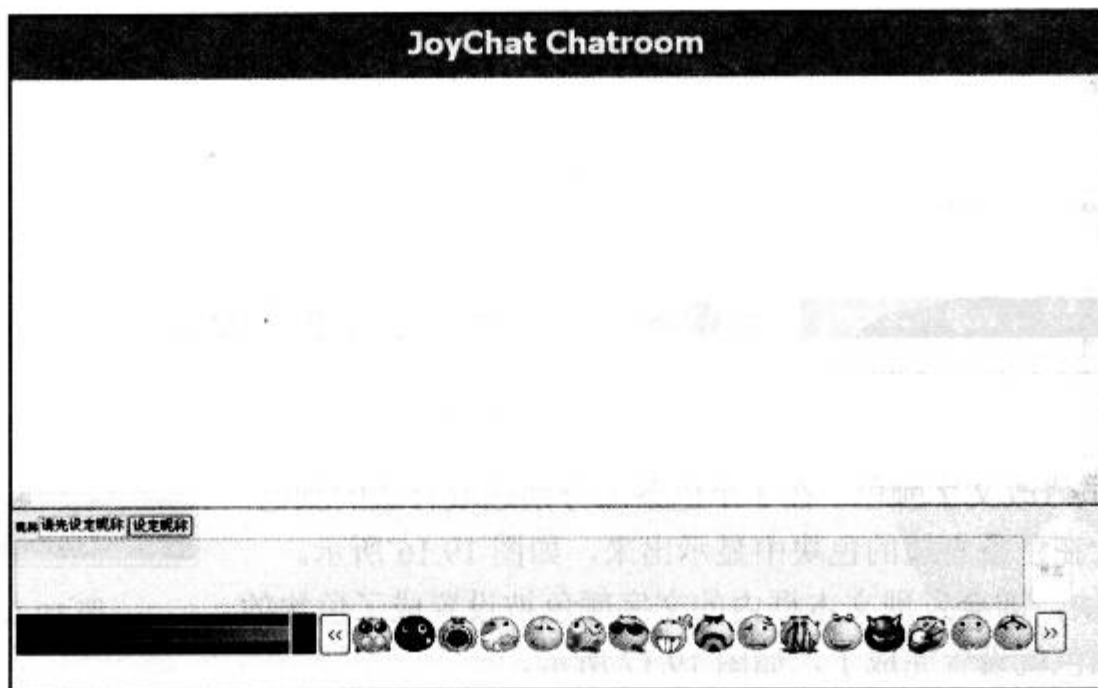


图 19.13 界面

(2) 此时消息输入框和“发言”按钮均为被禁用状态。首先设置昵称，在“昵称”文本框中输入 Rob 然后单击“设定昵称”按钮，如图 19.14 所示。

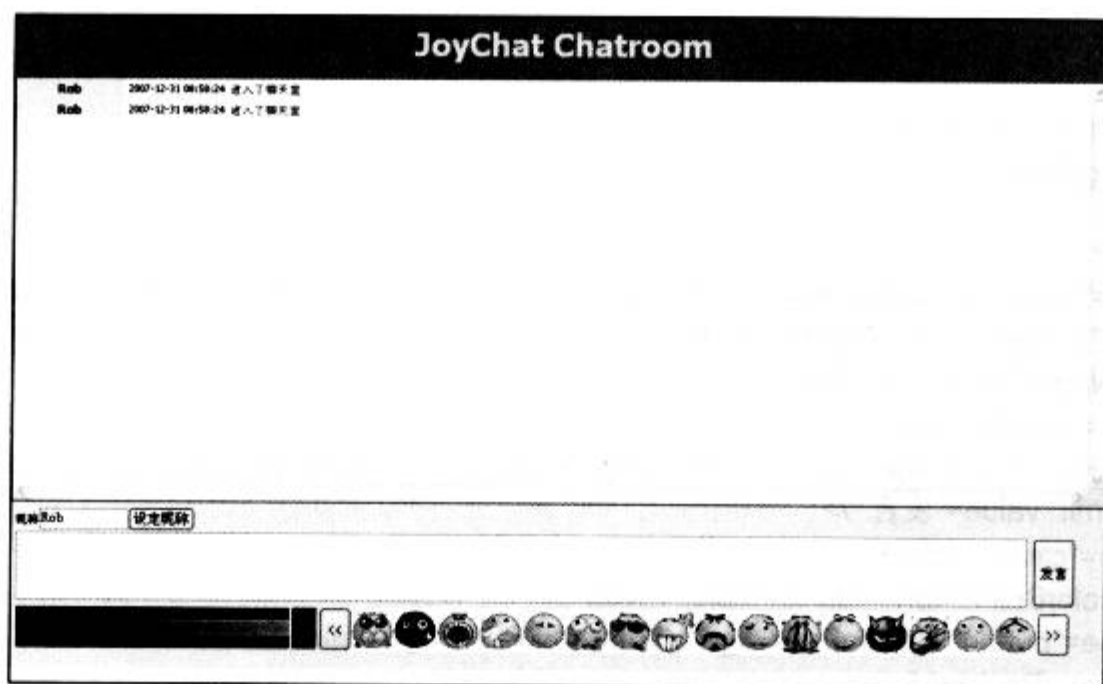


图 19.14 设置昵称

(3) 当第一次设置昵称后, 在消息显示区域会显示某人进入了聊天室, 并启用了页面上的消息输入框和“发言”按钮。在消息输入框中输入一些文字并单击“发言”按钮, 效果如图 19.15 所示。

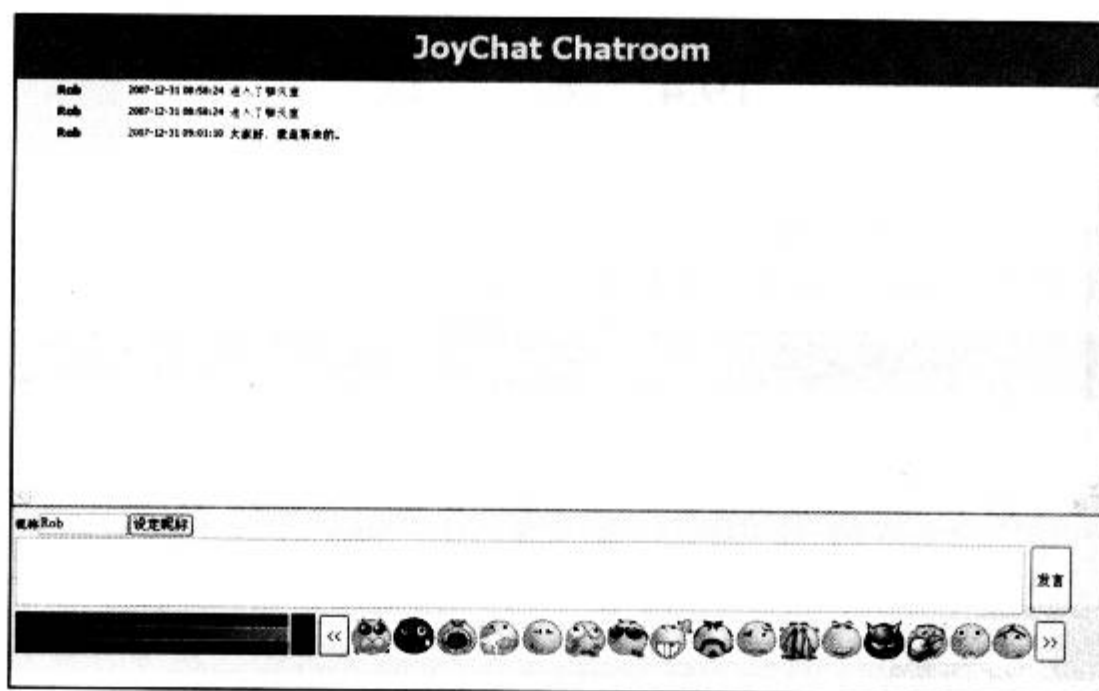


图 19.15 发送消息

(4) 下面尝试修改文字颜色。在 3 个色条上分别选取合适的颜色, 最终混合的颜色会在色条右边的色块中显示出来, 如图 19.16 所示。

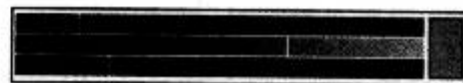


图 19.16 选择颜色

(5) 单击色块, 即会发现文本框内的文字颜色被设置成了色块的颜色, 此时文字颜色就设置完成了, 如图 19.17 所示。

(6) 在消息输入框中再次输入一些文字, 单击“发言”按钮, 这时显示在消息显示区域中的文字颜色被更改为设置的颜色, 如图 19.18 所示。

(7) 页面右下角是表情区域, 单击“<<”和“>>”按钮可以向前和向后翻页, 如图 19.19 所示。

(8) 单击表情图标, 对应的标签会被加入到消息输入框中, 如图 19.20 所示。

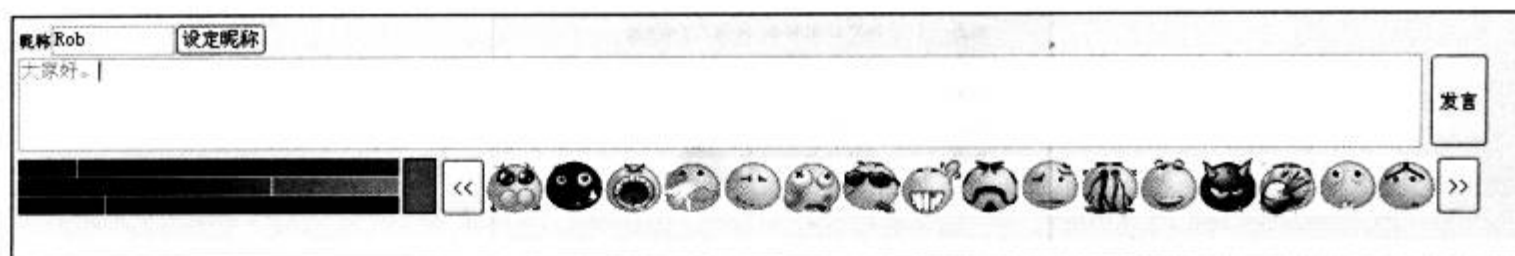


图 19.17 设置文字颜色

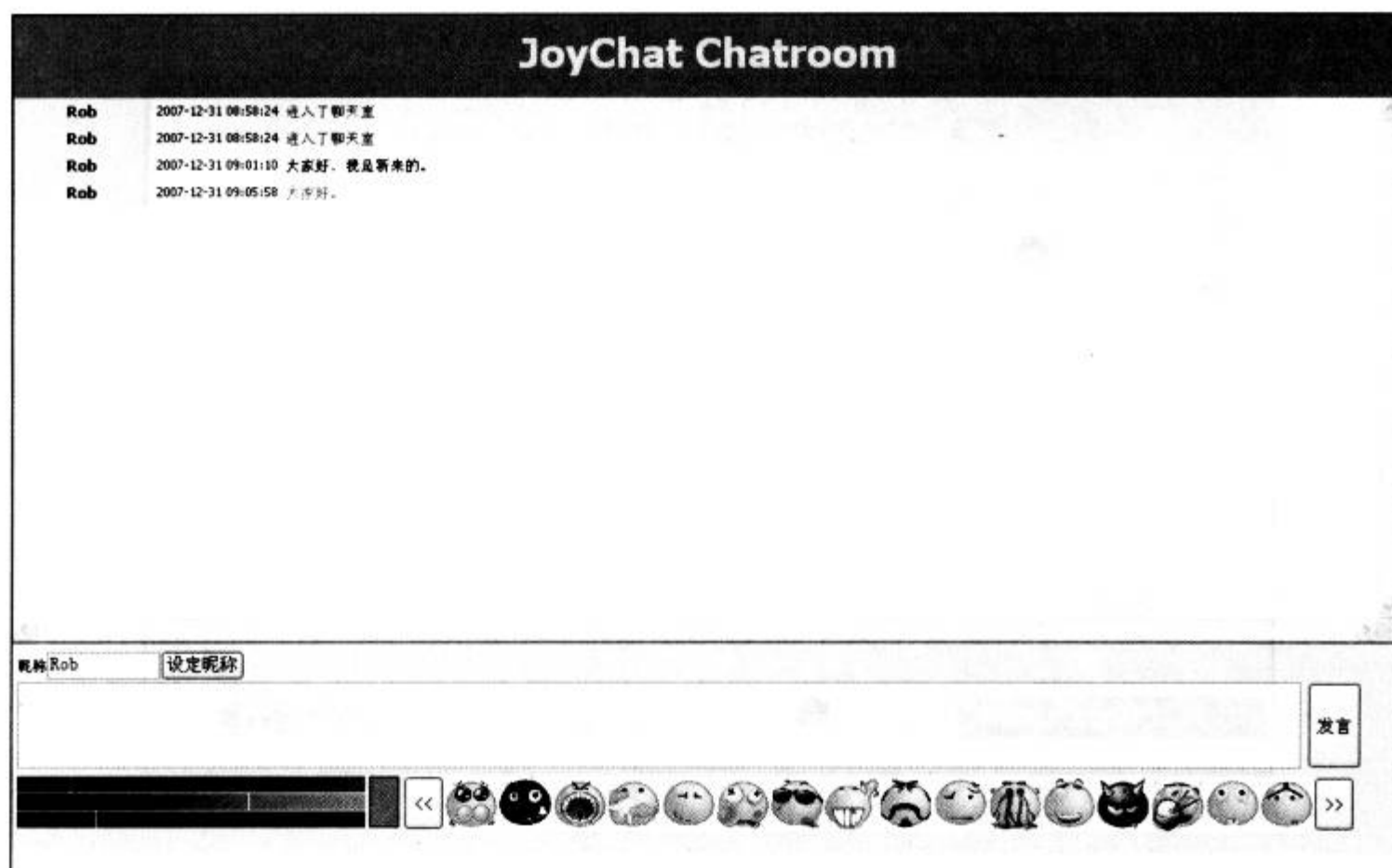


图 19.18 颜色设置效果

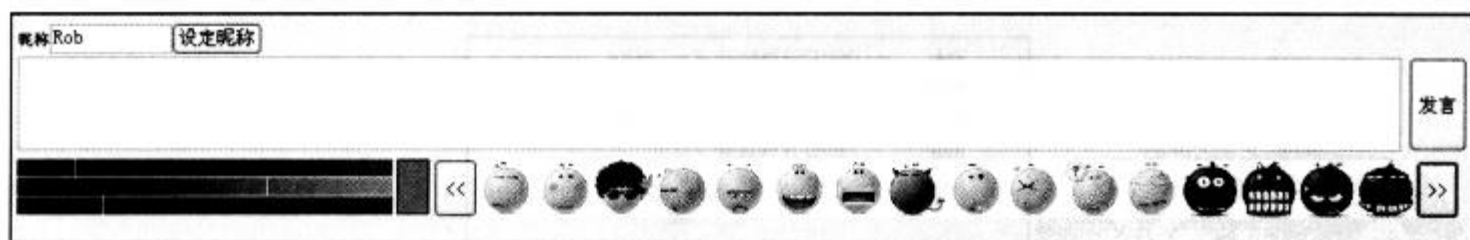


图 19.19 翻页

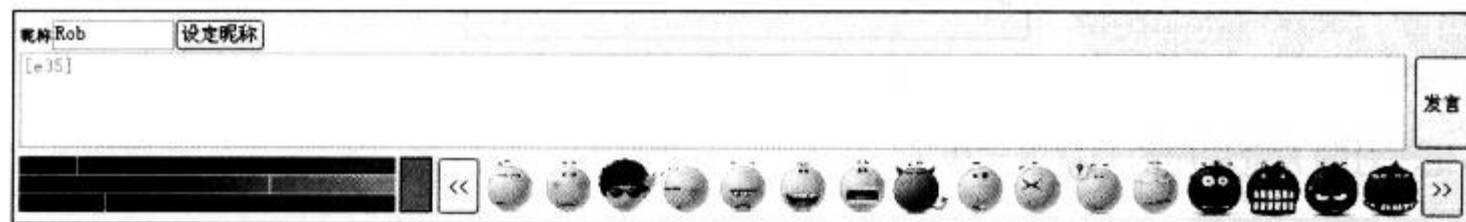


图 19.20 单击表情图标

(9) 单击“发言”按钮，发送的消息中包含了所选择的表情图标，如图 19.21 所示。





图 19.21 表情图标

(10) 修改昵称为 Robin，然后单击“设定昵称”按钮，如图 19.22 所示。

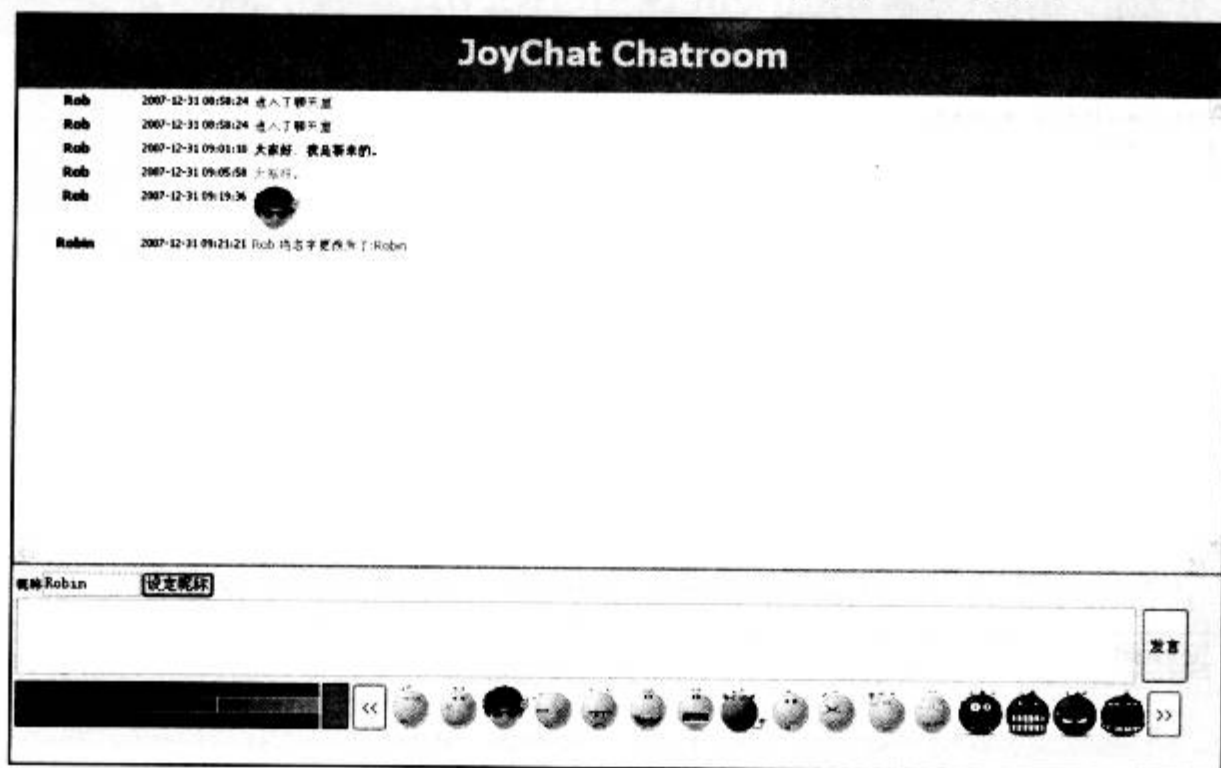


图 19.22 修改昵称

(11) 可见，在消息显示区域中显示了“Rob 将名字修改为了：Robin”的消息以通知其他用户。此时再次发送消息，在显示出来的消息内容中用户的名字已经变成了 Robin，如图 19.23 所示。



图 19.23 再次发送消息

至此，对聊天室程序的测试就完成了。

## 附录 A JavaScript 速查手册

### A

#### abs

**Math.abs(argument);**

返回变量的绝对值。

#### acos

**Math.acos(argument);**

返回变量的反余弦值。

#### action

**formElement.action**

表单提交的目标路径。

#### alert

**alert(argument);**

显示一个 JavaScript 警告对话框，显示 **argument** 的内容。

#### alinkColor

**document.alinkColor**

定义鼠标按下但未释放时的链接颜色。

#### anchor

**string.anchor(anchorName)**

创建并显示一个锚点。

## anchors

`document.anchors`

将文档中所有锚点作为数组返回。

## appCodeName

`navigator.appCodeName`

取得当前浏览器的代码名称。

## appName

`navigator.appName`

取得当前浏览器的应用程序名。

## appVersion

`navigator.appVersion`

取得当前浏览器的版本信息。

## asin

`Math.asin(argument)`

返回变量的反正弦值。

## atan

`Math.atan(argument)`

返回变量的反正切值。

## B

## back

`history.back()`

跳转到浏览器历史记录列表中的前一个 URL。

## bgColor

`document.bgColor`

设置文档的背景颜色。

## blur

`element.blur()`

从当前元素上去除焦点。

## C

## ceil

`Math.ceil(argument)`

返回最接近变量，但不小于变量的整数。

## charAt

`string.charAt(index)`

返回字符串中指定位置的字符。

## checked

`element.checked`

返回一个表示单选按钮或者复选框是否被选中的布尔值。

## clear

`document.clear()`

擦除窗口内容。。

## clearInterval

`clearInterval(interval)`

清除一个定时执行器。

## defaultSelected

**element.defaultSelected**

取得一个 option 元素的默认选中状态。

## defaultValue

**element.defaultValue**

取得一个表单元素的默认值。

## document

**document.property**

**document.method(parameters)**

当前文档对象。

## E

## E

**Math.E**

取得自然对数的底。

## elements

**form.elements**

返回表单中的表单元素集合。

## escape

**escape(argument)**

返回参数的 ASCII 码表示。

## eval

**eval(argument)**

将参数作为 JavaScript 语句运行并返回运行结果。



## exp

**Math.exp(argument)**

返回一个指数函数值。

## F

## fgColor

**document.fgColor**

设置文档的前景文本颜色。

## floor

**Math.floor(argument)**

返回最接近，但不小于参数的整数。

## focus

**element.focus()**

将键盘焦点移动到当前元素。

## forms

**document.forms**

返回表单内的所有表单元素集合。

## forward

**history.forward()**

跳转到浏览器历史记录列表中的下一个 URL。

## frames

**window.frames**

返回当前窗口内包含的所有框架元素集合。

## G

**getDate****Date.getDate()**

返回一个 1~31 的整数，表示一月中的第几天。

**getHours****Date.getHours()**

返回时间中的小时部分。

**getMinutes****Date.getMinutes()**

返回时间中的分钟部分。

**getMonth****Date.getMonth()**

返回一个 0~11 的整数，表示日期中的月份部分。

**getSeconds****Date.getSeconds()**

返回时间中的秒数部分。

**getTime****Date.getTime()**

返回时间的 UNIX 时间戳。

**getTimeZoneOffset****Date.getTimeZoneOffset()**

以分为单位，返回当前时间与 GMT 时间的时间差。

## getFullYear

**Date.getFullYear()**

返回日期中的年份部分。

## go

**history.go(argument)**

跳转到浏览器历史记录列表中指定位置的 URL。

# H

## hash

**document.location.hash**

返回一个 URL 中#后面的部分。

## history

**document.history**

浏览器的浏览记录列表，只读。

## host

**window.location.host**

返回一个 URL 中服务器名和端口组成的字符串。

## hostname

**window.location.hostname**

返回一个 URL 中的服务器名。

## href

**window.location.href**

取得或设置一个 URL 地址。

## I

**index**

`option.index`

返回一个 `option` 元素的下标值。

**indexOf**

`string.indexOf(substring, startingPoint)`

在字符串中从 `startingPoint` 位置开始查找指定的子字符串 `subString` 第一次出现的位置。`startingPoint` 默认为 0。

**isNaN**

`isNaN(argument)`

检查一个参数是否不是数字。

## L

**lastIndexOf**

`string.lastIndexOf(substring)`

与 `indexOf` 作用类似，但是是从字符串尾部从后向前查找子字符串。

**lastModified**

`document.lastModified`

返回表示当前文档最后修改日期的字符串，只读。

**length**

`array.length`

返回数组的长度。

`string.length`

返回字符串的长度。

## links

`document.links`

返回文档内包含的所有链接元素的集合。

## linkColor

`document.linkColor`

获取或设置文档中超链接的颜色。

## LN2

`Math.LN2`

表示 2 的自然对数的一个常数。

## LN10

`Math.LN10`

表示 10 的自然对数的一个常数。

## location

`window.location`

获取或设置窗口的 URL。

## log

`Math.log(argument)`

返回一个正数的自然对数（以 E 为底）。

## LOG2E

`Math.LOG2E`

返回 E 的以 2 为底的对数。



## O

## open

`document.open([MIMEType])`

打开一个文档输出流。

`window.open(url,name,options);`

打开一个新窗口。

## options

`select.options`

获取一个 `select` 元素所包含的所有 `option` 元素集合。

## P

## parent

`window.parent`

获取当前窗口的父窗口的 `window` 对象的引用。

## parse

`Date.parse(timeString)`

根据一个表示时间的字符串返回该时间对应的 UNIX 时间戳。

## parseFloat

`parseFloat(string)`

`parseFloat` 字符串接受一个字符串参数，如果字符串的第一个字符是一个正号、负号、十进制小数点、指数或者数字，它将返回一个浮点数；否则返回 `NaN`。

## parseInt

`parseInt(string[,radix])`

与 `parseFloat` 功能类似，其分析一个字符串，并根据指定的基数返回一个整数，基数默认为 10。

## pathname

**window.location.pathname**

返回 URL 中的路径部分。

## PI

**Math.PI**

返回圆周率的值。

## port

**window.location.port**

返回 URL 中的端口部分。

## pow

**Math.pow(argument1, argument2)**

返回底数的指数幂。

## prompt

**prompt(string)**

显示一个用来接收用户输入的对话框，返回用户输入的信息。

## protocol

**window.location.protocol**

返回 URL 中的协议部分。

## R

## random

**Math.random()**

返回一个大于 0 而小于 1 的随机浮点数。

## referrer

`document.referrer`

引导到当前文档的 URL 地址。

## reset

`form.reset()`

重置表单。

## round

`Math.round(argument)`

将一个数四舍五入。

## S

## search

`window.location.search`

返回 URL 中的查询字符串（位于?之后的部分）

## select

`element.select()`

选择一个表单元素的选项或输入区域。

## selected

`option.selected`

检查一个 option 元素是否被选中。

## selectedIndex

`select.selectedIndex`

返回一个 select 元素的选中项的索引值。

## self

`window.self`

获取对当前窗口的 `window` 对象的引用。

## setDate

`date.setDate(argument)`

设置日期的日期（一月中的第几天）部分。

## setHours

`date.setHours(argument)`

设置时间的小时部分。

## setInterval

`setInterval(function,interval)`

设置一个定时执行器，每隔 `interval` 毫秒就会调用一次 `function` 函数。

## setMinutes

`date.setMinutes(argument)`

设置时间的分钟部分。

## setMonth

`date.setMonth(argument)`

设置日期的月份部分。

## setSeconds

`date.setSeconds(argument)`

设置时间的秒数部分。

## setTime

`date.setTime(argument)`

设置日期的 UNIX 时间戳。

## setTimeout

**setTimeout(function,millionSeconds)**

设置一个超时执行器。在 **millionSeconds** 毫秒后调用 **function** 函数。

## setYear

**date.setYear(argument)**

设置一个日期的年份部分。

## sin

**Math.sin(argument)**

返回一个变量的正弦值。

## sqrt

**Math.sqrt(argument)**

返回一个变量的平方根。

## SQRT1\_2

**Math.SQRT1\_2**

返回 1/2 的平方根。

## SQRT2

**Math.SQRT2**

返回 2 的平方根。

## status

**window.status**

获取浏览器的状态栏信息。

## submit

**form.submit()**

提交表单。



## substring

`string.substring(index1,index2)`

根据两个下标，返回指定位置的子字符串。

## T

## tan

`Math.tan(argument)`

返回变量的正切值。

## target

`form.target`

`link.target`

设置表单提交或者打开链接的目标窗口。

## title

`document.title`

获取或设置文档的标题。

## toGMTString

`date.toGMTString()`

将日期对象转换为 GMT 格式的字符串表示。

## toLocaleString

`date.toLocaleString()`

使用本地约定将一个日期对象转换为字符串表示。

## toLowerCase

`string.toLowerCase()`

将字符串转换为小写形式。

## toUpperCase

`string.toUpperCase()`

将字符串转换为大写形式。

## U

## unescape

`unescape(string)`

返回一个基于其 ASCII 码的字符串。

## userAgent

`navigator.userAgent`

返回浏览器发送的 `user-agent` 头信息内容。

## V

## vlinkColor

`document.vlinkColor`

获取或设置被访问过的链接的颜色。

## W

## write

`document.write(string)`

向当前文档中输出内容。

## writeln

`document.writeln(string)`

与 `write` 方法作用一样，只是会在输出的内容后面增加一个换行符。

# 附录 B HTML DOM 速查手册

## B.1 Node

操作 Node 对象的相关属性和方法如表 B-1 所示。

表 B-1 Node 对象相关属性和方法

属性/方法	说 明
getElementsByTagName(tagName)	查找指定tagName的节点集合
parentNode	获得节点的父节点
firstChild	获得节点的第一个直接子节点
lastChild	获得节点的最后一个直接子节点
previousSibling	获得节点的前一个兄弟节点
nextSibling	获得节点的后一个兄弟节点
nodeName	节点名
nodeValue	节点值
nodeType	节点类型

Node 类型的详细信息如表 B-2 所示。

表 B-2 Node 类型

类 型	说 明
Element(1)	元素节点
Attribute(2)	属性节点
Text(3)	文本节点
Comment(8)	注释节点
Document(9)	文档节点

## B.2 Window

Window 对象的属性如表 B-3 所示。

表 B-3 Window 对象的属性

属 性	说 明
closed	窗口是否已经被关闭
document	见B.7节Document对象的说明

续表

属 性	说 明
history	见B.5节History对象的说明
length	设置或获取窗口中的frames数量
location	见B.6节Location对象的说明
name	设置或获取window对象的名称
opener	返回创建此窗口的window对象的引用
parent	返回当前窗口的父窗口对象的引用
self	返回当前窗口的引用
status	设置或获取窗口状态栏信息（Firefox不支持）
top	返回顶级窗口的引用

Window 对象的方法如表 B-4 所示。

表 B-4 Window 对象的方法

方 法	说 明
alert(string)	显示一个有确定（OK）按钮的对话框
blur()	使当前窗口失去焦点
clearInterval(interval)	清除一个定时执行器
clearTimeout(timeout)	清除一个超时执行器
close()	关闭当前窗口
confirm(string)	显示一个有确定（OK）按钮和取消（Cancel）按钮的确认对话框
focus()	使当前窗口获得焦点
moveBy(x,y)	将窗口移动到指定的距离
moveTo(x,y)	将窗口移动到指定的位置
open(url)	打开新窗口
print()	打印当前页面内容
prompt(string)	显示一个用于用户输入信息的对话框
resizeBy(width,height)	调整窗口大小
resizeTo(width,height)	将窗口调整到指定大小
scrollBy(distance)	将窗口滚动指定距离
scrollTo(int)	将窗口滚动到指定位置
setInterval(function,millionSeconds)	设置一个定时执行器
setTimeout(function, millionSeconds)	设置一个超时执行器

## B.3 Navigator

Navigator 对象的属性如表 B-5 所示。

表 B-5 Navigator 对象属性

属 性	说 明
appCodeName	浏览器的代码名称
appName	浏览器的应用程序名
appVersion	浏览器的版本号
cookieEnabled	浏览器是否开启了Cookie
platform	操作系统信息
userAgent	浏览器发送的user-agentHTTP头信息内容

## B.4 Screen

Screen 对象的属性如表 B-6 所示。

表 B-6 Screen 对象属性

属 性	说 明
availHeight	屏幕显示区域的高度（包括windows任务栏）
availWidth	屏幕显示区域的宽度（包括windows任务栏）
colorDepth	颜色深度
height	屏幕显示区域的高度
width	屏幕显示区域的宽度

## B.5 History

History 对象的属性如表 B-7 所示。

表 B-7 History 对象属性

属 性	说 明
length	浏览器历史记录列表的长度

History 对象的方法如表 B-8 所示。

表 B-8 History 对象方法

方 法	说 明
back()	加载历史记录中前一个URL
forward()	加载历史记录中后一个URL
go(offset)	加载历史记录中相对于当前记录指定位置的URL



## B.6 Location

Location 对象的属性如表 B-9 所示。

表 B-9 Location 对象属性

属 性	说 明
hash	设置或获取当前URL中#号后面的部分内容
host	设置或获取当前URL中主机名和端口号
hostname	设置或获取当前URL中的主机名
href	设置或获取当前URL
pathname	设置或获取当前URL中的路径部分
port	设置或获取当前URL中的端口号
protocol	设置或获取当前URL中的协议
search	设置或获取当前URL中的?号后面的部分内容

Location 对象的方法如表 B-10 所示。

表 B-10 Location 对象方法

方 法	说 明
assign(url)	加载一个新的文档
reload()	重新加载当前文档
replace(url)	使用一个新的文档替换当前文档

## B.7 Document

Document 对象的属性如表 B-11 所示。

表 B-11 Document 对象属性

属 性	说 明
body	引用body节点
cookie	cookie数据
domain	文档的域名
lastModified	文档最后修改时间
referrer	引导进入当前文档的URL
title	文档标题
URL	文档的URL
anchors	文档中所有的锚点集合
forms	文档中所有的表单集合

续表

属 性	说 明
images	文档中所有的图像集合
links	文档中所有的链接集合

Document 对象的方法如表 B-12 所示。

表 B-12 Document 对象方法

方 法	说 明
close()	关闭一个文档输出流
getElementById(id)	根据id获得元素的引用
getElementsByName(name)	根据name获得元素的引用，返回的是一个元素集合
getElementsByTagName(name)	根据tagName元素的引用，返回的是一个元素集合
open()	打开一个文档输出流
write(string)	输出文档内容
writeln(string)	输出一行文档内容

## B.8 Anchor

Anchor 对象的属性如表 B-13 所示。

表 B-13 Anchor 对象属性

属 性	说 明
accessKey	设置或获取链接的快捷键
charset	设置或获取链接资源的字符集
coords	设置或获取image-map中的项目
href	设置或获取链接
hreflang	设置或获取链接资源的语言代码
id	设置或获取链接的id
innerHTML	设置或获取链接的文本
name	设置或获取链接的name
rel	设置或获取当前文档和目标URL之间的关联
rev	设置或获取当前文档和目标URL之间的关联
shape	设置或获取图片链接的形状
tabIndex	设置或获取链接的tab顺序
target	设置或获取在哪里打开链接
type	设置或获取链接资源的MIME类型
className	设置或获取链接的CSS样式类
dir	设置或获取文本的方向
lang	设置或获取语言代码
title	设置或获取提示标题

Anchor 对象的方法如表 B-14 所示。

表 B-14 Anchor 对象方法

方 法	说 明
blur()	将焦点从链接上移除
focus()	使链接获得焦点

## B.9 Area

Area 对象的属性如表 B-15 所示。

表 B-15 Area 对象属性

属 性	说 明
accessKey	设置或获取区域的快捷键
alt	设置或获取一个替换文本
coords	设置或获取image-map中的项目
hash	设置或获取区域的URL中#号后面的部分
host	设置或获取区域的URL中的主机名和端口号
href	设置或获取image-map的URL地址
id	设置或获取链接的id
noHref	设置或获取当前区域是否是活动的
pathname	设置或获取区域的URL中的路径名
protocol	设置或获取区域的URL中的协议名
search	设置或获取区域的URL中?号后面的部分
shape	设置或获取图片链接的形状
tabIndex	设置或获取tab顺序
target	设置或获取在哪里打开链接
className	设置或获取CSS样式类
dir	设置或获取文本的方向
lang	设置或获取语言代码
title	设置或获取提示标题

## B.10 Base

Base 对象的属性如表 B-16 所示。

表 B-16 Base 对象属性

属 性	说 明
href	设置或获取页面所有链接的基本URL
id	设置或获取id
target	设置或获取页面所有链接的默认target属性

## B.11 Body

Body 对象的属性如表 B-17 所示。

表 B-17 Body 对象属性

属 性	说 明
className	设置或获取CSS样式类
dir	设置或获取文本的方向
id	设置或获取id
lang	设置或获取语言代码
title	设置或获取提示标题

## B.12 Button

Button 对象的属性如表 B-18 所示。

表 B-18 Button 对象属性

属 性	说 明
accessKey	设置或获取按钮的快捷键
disabled	设置或获取是否应该禁用按钮
form	返回包含按钮的表单的引用
name	设置或获取name
tabIndex	设置或获取按钮的tab顺序
type	返回按钮的类型
value	设置或获取显示在按钮上的文本
className	设置或获取CSS样式类
dir	设置或获取文本的方向
id	设置或获取id
lang	设置或获取语言代码
title	设置或获取提示标题

## B.13 Event

事件句柄如表 B-19 所示。

表 B-19 事件句柄

句 柄	事件触发时机
onabort	加载中的图片被阻止时
onblur	元素失去焦点时
onchange	一个控件的内容发生改变时
onclick	鼠标单击了一个对象时
ondblclick	鼠标双击了一个对象时
onerror	在文档或图片加载发生错误时
onfocus	元素获得焦点时
onkeydown	一个键盘键被按下时
onkeypress	一个键盘键被按下或保持按下时
onkeyup	一个键盘键被释放时
onload	一个页面或图片加载完成时
onmousedown	一个鼠标键被按下时
onmousemove	当鼠标移动了时
onmouseout	当鼠标移出对象时
onmouseover	当鼠标滑过对象时
onmouseup	当鼠标键被释放时
onreset	当重置按钮被按下时
onresize	当窗口大小改变时
onselect	当文本被选中时
onsubmit	当表单被提交时
onunload	当用户退出页面时

事件属性如表 B-20 所示。

表 B-20 事件属性

属 性	说 明
altKey	Alt键是否被按下
button	哪个鼠标键被按下
clientX	事件发生时鼠标位置的X坐标
clientY	事件发生时鼠标位置的Y坐标
ctrlKey	Ctrl键是否被按下
metaKey	META键是否被按下
screenX	事件发生时鼠标相对于屏幕的X坐标



续表

属 性	说 明
screenY	事件发生时鼠标相对于屏幕的Y坐标
shiftKey	Shift键是否被按下
type	事件名

## B.14 Form

Form 对象的属性如表 B-21 所示。

表 B-21 Form 对象属性

属 性	说 明
elements	获取表单中表单项的集合
action	设置或获取表单提交的路径
encrypt	设置或获取用来给表单内容编码的MIME类型
name	设置或获取name
length	返回表单中表单项的数量
method	设置或获取表单的提交方法
target	设置或获取表单提交的目标
className	设置或获取CSS样式类
dir	设置或获取文本的方向
id	设置或获取id
lang	设置或获取语言代码
title	设置或获取提示标题

Form 对象的方法如表 B-22 所示。

表 B-22 Form 对象方法

方 法	说 明
reset()	重置表单
submit()	提交表单

## B.15 Frame

Frame 对象的属性如表 B-23 所示。

表 B-23 Frame 对象属性

属 性	说 明
frameBorder	设置或获取是否显示frame边框
longDesc	设置或获取一个包含对frame的描述的URL地址
marginHeight	设置或获取frame的上下边距
marginWidth	设置或获取frame的左右边距
name	设置或获取frame的name
noResize	设置或获取frame是否可以被改变大小
src	设置或获取frame要显示的页面路径
className	设置或获取CSS样式类
dir	设置或获取文本的方向
id	设置或获取id
lang	设置或获取语言代码
title	设置或获取提示标题

## B.16 Frameset

Frameset 对象的属性如表 B-24 所示。

表 B-24 Frameset 对象属性

属 性	说 明
col	设置或获取框架中的列数
rows	设置或获取框架中的行数
className	设置或获取CSS样式类
dir	设置或获取文本的方向
id	设置或获取id
lang	设置或获取语言代码
title	设置或获取提示标题

## B.17 IFrame

Iframe 对象的属性如表 B-25 所示。

表 B-25 Iframe 对象属性

属 性	说 明
align	设置或获取iframe的对齐方式
frameBorder	设置或获取是否显示iframe边框
height	设置或获取iframe的高度
longDesc	设置或获取一个包含对iframe的描述的URL地址

续表

属 性	说 明
marginHeight	设置或获取iframe的上下边距
marginWidth	设置或获取iframe的左右边距
name	设置或获取iframe的name
src	设置或获取iframe要显示的页面路径
className	设置或获取CSS样式类
dir	设置或获取文本的方向
id	设置或获取id
lang	设置或获取语言代码
title	设置或获取提示标题

## B.18 Image

Image 对象的属性如表 B-26 所示。

表 B-26 Image 对象属性

属 性	说 明
align	设置或获取image的对齐方式
alt	设置或获取替换文本
border	设置或获取image的边框大小
complete	返回image是否加载完成
height	设置或获取image的高度
hspace	设置或获取image左右的空白区域
longDesc	设置或获取一个包含对iframe的描述的URL地址
isMap	返回image是否为一个服务端图片地图
userMap	设置或获取图片使用的客户端图片地图
vspace	设置或获取image上下的空白区域
width	设置或获取image的宽度
name	设置或获取image的name
src	设置或获取要显示的图片的URL
className	设置或获取CSS样式类
id	设置或获取id
title	设置或获取提示标题

## B.19 CheckBox

CheckBox 对象的属性如表 B-27 所示。

表 B-27 CheckBox 对象属性

属 性	说 明
accessKey	设置或获取checkbox的快捷键
Fchecked	设置或获取checkbox是否被选中
defaultChecked	获取checkbox的默认选中状态
disabled	设置或获取是否应该禁用checkbox
form	获取对包含checkbox的表单元素的引用
name	设置或获取checkbox的name
tabIndex	设置或获取checkbox的tab顺序
type	获取checkbox的类型
className	设置或获取CSS样式类
dir	设置或获取文本的方向
value	设置或获取checkbox的value
id	设置或获取id
lang	设置或获取语言代码
title	设置或获取提示标题

CheckBox 对象的方法如表 B-28 所示。

表 B-28 CheckBox 对象方法

方 法	说 明
blur()	移除checkbox的焦点
click()	模拟单击checkbox
focus()	使checkbox获得焦点

## B.20 FileUpload

FileUpload 对象的属性如表 B-29 所示。

表 B-29 FileUpload 对象属性

属 性	说 明
defaultValue	默认值
disabled	设置或获取是否应该禁用file控件
form	获取对包含file控件的表单元素的引用
name	设置或获取file控件的name
tabIndex	设置或获取file控件的tab顺序
type	获取file控件的类型
value	设置或获取file控件的value属性
className	设置或获取CSS样式类
dir	设置或获取文本的方向

续表

属 性	说 明
id	设置或获取id
lang	设置或获取语言代码
title	设置或获取提示标题

FileUpload 对象的方法如表 B-30 所示。

表 B-30 FileUpload 对象方法

方 法	说 明
blur()	移除file控件的焦点
focus()	使file控件获得焦点

## B.21 Hidden

Hidden 对象的属性如表 B-31 所示。

表 B-31 Hidden 对象属性

属 性	说 明
alt	设置或获取替换文本
form	获取对包含hidden控件的表单元素的引用
name	设置或获取hidden控件的name
type	获取hidden控件的类型
className	设置或获取CSS样式类
dir	设置或获取文本的方向
id	设置或获取id
lang	设置或获取语言代码
title	设置或获取提示标题

## B.22 Password

Password 对象的属性如表 B-32 所示。

表 B-32 Password 对象属性

属 性	说 明
accessKey	设置或获取password控件的快捷键
alt	设置或获取替换文本
defaultValue	设置或获取password控件的默认值
form	获取对包含password控件的表单元素的引用



续表

属 性	说 明
name	设置或获取password控件的name
disabled	设置或获取password控件是否需要被禁用
type	获取password控件的类型
maxLength	设置或获取password控件中最大的字符数
readOnly	设置或获取password控件是否是只读的
size	设置或获取password控件的尺寸
tabIndex	设置或获取password控件的tab顺序
value	设置或获取password控件的value属性
className	设置或获取CSS样式类
dir	设置或获取文本的方向
id	设置或获取id
lang	设置或获取语言代码
title	设置或获取提示标题

Password 对象的方法如表 B-33 所示。

表 B-33 Password 对象方法

方 法	说 明
blur()	移除password控件的焦点
focus()	使password控件获得焦点
select()	选中password控件的文本内容

## B.23 Radio

Radio 对象的属性如表 B-34 所示。

表 B-34 Radio 对象属性

属 性	说 明
accessKey	设置或获取radio控件的快捷键
alt	设置或获取替换文本
checked	设置或获取radio控件是否被选中
defaultChecked	获取radio控件的默认选中状态
form	获取对包含radio控件的表单元素的引用
name	设置或获取radio控件的name
disabled	设置或获取radio控件是否需要被禁用
type	获取radio控件的类型
tabIndex	设置或获取radio控件的tab顺序
value	设置或获取radio控件的value属性

续表

属 性	说 明
className	设置或获取CSS样式类
dir	设置或获取文本的方向
id	设置或获取id
lang	设置或获取语言代码
title	设置或获取提示标题

Radio 对象的方法如表 B-35 所示。

表 B-35 Radio 对象方法

方 法	说 明
blur()	移除radio控件的焦点
focus()	使radio控件获得焦点
click()	模拟单击radio控件

## B.24 Text

Text 对象的属性如表 B-36 所示。

表 B-36 Text 对象属性

属 性	说 明
accessKey	设置或获取text控件的快捷键
alt	设置或获取替换文本
defaultValue	设置或获取text控件的默认值
form	获取对包含text控件的表单元素的引用
name	设置或获取text控件的name
disabled	设置或获取text控件是否需要被禁用
type	获取text控件的类型
maxLength	设置或获取text控件中最大的字符数
readOnly	设置或获取text控件是否是只读的
size	设置或获取text控件的尺寸
tabIndex	设置或获取text控件的tab顺序
value	设置或获取text控件的value属性
className	设置或获取CSS样式类
dir	设置或获取文本的方向
id	设置或获取id
lang	设置或获取语言代码
title	设置或获取提示标题

Text 对象的方法如表 B-37 所示。

表 B-37 Text 对象方法

方 法	说 明
blur()	移除text控件的焦点
focus()	使text控件获得焦点
select()	选中text控件的文本内容

## B.25 Select

Select 对象的属性如表 B-38 所示。

表 B-38 Select 对象属性

属 性	说 明
options	获取select控件包含的所有option元素集合
form	获取对包含select控件的表单元素的引用
name	设置或获取select控件的name
disabled	设置或获取select控件是否需要被禁用
type	获取select控件的类型
multiple	设置或获取select控件是否可以被多选
size	设置或获取select控件可见的行数
length	获取select控件包含的option元素数量
tabIndex	设置或获取select控件的tab顺序
className	设置或获取CSS样式类
dir	设置或获取文本的方向
id	设置或获取id
lang	设置或获取语言代码
title	设置或获取提示标题

Select 对象的方法如表 B-39 所示。

表 B-39 Select 对象方法

方 法	说 明
blur()	移除select控件的焦点
focus()	使select控件获得焦点
add(option,before)	增加select控件包含的option元素
remove(index)	删除select控件包含的指定索引的option元素

## B.26 Table

Table 对象的属性如表 B-40 所示。

表 B-40 Table 对象属性

属 性	说 明
cells	获取表格包含的所有单元格的集合
rows	获取表格包含的所有行的集合
border	设置或获取表格的边框
caption	设置或获取表格的标题
cellPadding	设置或获取表格单元格的内容与边框的距离
cellSpacing	设置或获取表格单元格之间的距离
frame	设置或获取表格的外边框
rules	设置或获取表格的内边框
summary	设置或获取表格的摘要信息
className	设置或获取CSS样式类
width	设置或获取表格的宽度
dir	设置或获取文本的方向
id	设置或获取id
lang	设置或获取语言代码
title	设置或获取提示标题

Table 对象的方法如表 B-41 所示。

表 B-41 Table 对象方法

方 法	说 明
createCaption()	为表格创建一个标题元素
deleteCaption()	删除一个标题元素
deleteRow(index)	删除一行
insertRow(index)	插入一个新行

## B.27 TableRow

TableRow 对象的属性如表 B-42 所示。

表 B-42 TableRow 对象属性

属 性	说 明
cells	获取表格行包含的所有单元格的集合
align	设置或获取对齐方式

续表

属 性	说 明
id	设置或获取id
innerHTML	设置或获取包含的HTML内容
rowIndex	获取行的索引
vAlign	获取或设置垂直对齐方式

TableRow 对象的方法如表 B-43 所示。

表 B-43 TableRow 对象方法

方 法	说 明
deleteCell(index)	删除一个单元格
insertCell(index)	插入一个新单元格

## B.28 TableCell

TableCell 对象的属性如表 B-44 所示。

表 B-44 TableCell 对象属性

属 性	说 明
abbr	设置或获取单元格内容的简短版本
align	设置或获取对齐方式
id	设置或获取id
innerHTML	设置或获取包含的HTML内容
cellIndex	获取单元格的索引
vAlign	获取或设置垂直对齐方式
className	设置或获取CSS样式类
width	设置或获取单元格的宽度
dir	设置或获取文本的方向
id	设置或获取id
lang	设置或获取语言代码
title	设置或获取提示标题

## B.29 Textarea

Textarea 对象的属性如表 B-45 所示。



表 B-45 Textarea 对象属性

属 性	说 明
accessKey	设置或获取文本域的快捷键
cols	设置或获取文本域的列数
defaultValue	设置或获取文本域的默认值
form	获取对包含文本域的表单元素的引用
name	设置或获取文本域的name
disabled	设置或获取文本域是否需要被禁用
type	获取文本域的类型
readOnly	设置或获取文本域是否是只读的
tabIndex	设置或获取文本域的tab顺序
value	设置或获取文本域的value属性
rows	设置或获取文本域的行数
className	设置或获取CSS样式类
dir	设置或获取文本的方向
id	设置或获取id
lang	设置或获取语言代码
title	设置或获取提示标题

Textarea 对象的方法如表 B-46 所示。

表 B-46 Textarea 对象方法

方 法	说 明
blur()	移除文本域的焦点
focus()	使文本域获得焦点
select()	选中文本域的文本内容

# 附录 C Prototype 速查手册

## C.1 快捷方法

### C.1.1 \$

`$(id | element) -> HTMLElement`  
`$((id | element)...) -> [HTMLElement...]`

如果给定的参数是字符串，则返回在 DOM 树中 ID 与之匹配的元素。如果给定的参数是 DOM 元素，则返回 DOM 元素本身。返回的元素都会被 Prototype 进行扩展。如果给定了多个参数，则返回一个元素数组。

### C.1.2 \$\$

`$(cssRule...) -> [HTMLElement...]`

\$\$方法以任意数量的 CSS 选择器作为参数，返回一个符合其中一个选择器的元素数组。

### C.1.3 \$A

`$A(iterable) -> actualArray`

\$A 方法接受一个集合对象作为参数，返回等价的数组。

### C.1.4 \$F

`$F(element) -> value`

\$F 方法接受一个表单元素的 id 或者表单元素本身作为参数，返回该表单元素的值。

### C.1.5 \$H

`$H([obj]) -> EnumerableHash`

\$H 方法接受一个对象作为参数，返回一个包含了对象数据的 Hash 对象。

### C.1.6 \$R

`$R(start, end[, exclusive = false]) -> ObjectRange`

返回一个 `ObjectRange` 对象，是 `ObjectRange` 构造函数的别名。

### C.1.7 \$w

`$w(String) -> Array`

`$w` 方法将一个字符串分隔成数组，字符串中所有的空格都会被当作分隔符。

### C.1.8 getElementsByClassName

`document.getElementsByClassName(className[, element]) -> [HTMLElement...]`

`getElementsByClassName` 方法返回拥有制定 CSS 类的元素数组，第二个可选参数可以指定只在某个元素的子元素中下载。

### C.1.9 Try.these

`Try.these(Function...) -> firstOKResult`

`Try.these` 方法接受一组函数作为参数，并以第一成功运行的函数的返回值作为返回值。

## C.2 模 块

### C.2.1 Ajax

#### 1. Ajax.Request

`new Ajax.Request(url[, options])`

制造并发起一个 Ajax 请求。url 为请求的地址，options 为请求选项，参见 `Ajax.Options`。

#### 2. Ajax.Responders

`Ajax.Responders.register(responder)`

`Ajax.Responders.unregister(responder)`

`Ajax.Responders` 是一个储藏全局的基于 Prototype 的 Ajax 请求的事件监听器仓库。

#### 3. Ajax.Updater

`new Ajax.Updater(container, url[, options])`

制造一个 Ajax 请求并使用返回的 `responseText` 更新 `container` 的内容。`container` 是页面元素容器，url 是请求地址，options 是请求选项，参见 `Ajax.options`。

#### 4. Ajax.options

`Ajax.options` 提供了 Ajax 请求的所有核心参数和回调函数。核心参数如表 C-1 所示。

表 C-1 核心参数

选 项	默 认 值	说 明
asynchronous	true	是否使用异步请求
contentType	'application/x-www-form-urlencoded'	Content-Type头信息
encoding	'UTF-8'	请求所使用的字符集编码
method	'post'	请求所使用的方法
parameters	''	请求参数。可选的格式是序列化的字符串或者Hash对象
postBody	None	为post方法指定要发送的数据。如果没有指定值, 则使用parameters代替
requestHeaders	见说明	为请求指定HTTP头信息。默认指定了的头信息包含: X-Prototype-Version 为Prototype版本号; Accept为text/javascript, text/html, application/xml, text/xml, */*; Content-type为制定的contentType和encoding参数组成

所有回调函数如表 C-2 所示。

表 C-2 回调函数

选 项	说 明
onComplete	在请求返回时被调用, 当前XMLHttpRequest对象会被作为参数传递给回调函数
onException	在请求发生异常时被调用, 当前XMLHttpRequest对象和异常对象会被作为第一和第二个参数传递给回调函数
onFailure	当请求返回的HTTP状态码小于200或者大于等于300时被调用。如果一个onXYZ回调函数被定义, 则该回调函数会被忽略
onInteractive	在接受数据过程中被触发
onLoaded	在XMLHttpRequest对象被设置, 连接已经打开, 准备发送数据时触发
onLoading	在请求的连接打开后被触发
onSuccess	在请求发挥的HTTP状态码在200~300之间(包含200)时触发, 当前XMLHttpRequest对象会作为第一个参数传递给回调函数
onUninitialized	在XMLHttpRequest刚被创建时触发
onXYZ	XYZ为某个指定的3位数数字, 表示HTTP状态码, 例如on200、on303等。当请求返回的HTTP状态码与回调函数名字中指定的状态码一致时, 回调函数被调用

## 5. Ajax.PeriodicalUpdater

`new Ajax.PeriodicalUpdater(container, url[, options])`

周期性地制造 Ajax 请求并使用返回的 `responseText` 更新 `container` 的内容。`container` 为页面元素容器, `url` 为请求地址, `options` 为请求选项, 参见 `Ajax.options`。

## C.2.2 Array

### 1. clear

`clear() -> Array`

清空数组, 返回空数组。

## 2. clone

`clone() -> newArray`

返回一个数组的拷贝。

## 3. compact

`compact() -> newArray`

去掉数组中值为 `null` 或 `undefined` 的元素，并返回一个新的数组。

## 4. each

`each(iterator) -> Array`

迭代数组，每次会将数组元素传入给定的迭代器中执行。

## 5. first

`first() -> value`

返回数组的第一个元素。

## 6. flatten

`flatten() -> newArray`

将一个多维数组（JavaScript 中并没有真正的多位数组类型，这里指数组元素中包含数组的数组）转换为一维数组，返回一个新的数组。如果某个数组元素本身为数组，则它会被它自己包含的元素所替代。

## 7. from

`Array.from(iterable) -> actualArray`

拷贝一个已经存在的数组或者将一个集合对象转换为数组。

## 8. indexOf

`indexOf(value) -> position`

返回指定值在数组中的索引值，如果不存在于数组中，则返回-1。

## 9. last

`last() -> value`

返回数组最后一个元素。

## 10. reduce

`reduce() -> Array | singleValue`

只有一个元素的数组会直接返回这个元素。有多个元素的数组则返回数组本身。

## 11. reverse

`reverse([inline = true]) -> Array`

与原生的 `reverse` 方法作用一样。只是当 `inline` 参数被设置为 `false` 时，会返回一个新数组而不是改



变原有数组。

### 12. toJSON

`toJSON() -> String`

将数组转换为对应的 JSON 字符串。

### 13. uniq

`uniq() -> newArray`

去掉数组中重复的元素（如果元素是字符串，则区分大小写），返回一个新数组。

### 14. without

`without(value...) -> newArray`

去掉指定的一个或者多个元素，返回新数组。

## C.2.3 Class

### create

`create() -> Function`

返回一个函数，当被调用时，会调用其 `initialize` 方法。也就是返回了一个以 `initialize` 方法作为构造函数的类。

## C.2.4 Date

### toJSON

`toJSON() -> String`

将 Date 对象实例转换为对应的 JSON 字符串。

## C.2.5 Element

### 1. addClassName

`addClassName(element, className) -> HTMLElement`

给元素增加一个 CSS 样式类。即将某个 CSS 类应用到元素上。

### 2. addMethods

`addMethods([methods])`

为所有元素增加方法。methods 参数为一个 Hash 对象，其每一个成员的名字标识了方法名，成员的值方法体。

### 3. ancestors

ancestors(element) -> [HTMLElement...]

将元素的所有祖先元素作为数组返回。

### 4. classNames

classNames(element) -> [className...]

将元素的所有 CSS 类名作为数组返回。

### 5. descendantOf

descendantOf(element, ancestor) -> Boolean

检查 element 是否为 ancestor 的子孙元素，并返回一个布尔值标识检查结果。

### 6. descendants

descendants(element) -> [HTMLElement...]

将元素的所有子孙元素作为数组返回。

### 7. down

down(element[, cssRule][, index = 0]) -> HTMLElement | undefined

返回元素的第一个子孙元素，如果 cssRule 被指定，则在符合 cssRule 的子孙元素中查找，如果 index 被指定，则返回位置为 index 的子孙元素。

### 8. empty

empty(element) -> Boolean

检查元素内容是否为空。

### 9. getDimensions

getDimensions(element) -> {height: Number, width: Number}

返回一个标识了元素高（height）和宽（width）值的对象。

### 10. getHeight

getHeight(element) -> Number

返回元素的高。

### 11. getStyle

getStyle(element, property) -> String | null

返回元素指定的 CSS 样式属性值。

### 12. getWidth

getWidth(element) -> Number

返回元素的宽。

### 13. hasClassName

hasClassName(element, className) -> Boolean

检查元素是否应用了某个 CSS 类，返回一个标识检查结果的布尔值。

### 14. hide

hide(element) -> HTMLElement

隐藏元素。

### 15. immediateDescendants

immediateDescendants(element) -> [HTMLElement...]

将元素的所有直接子元素作为一个数组返回，子元素在数组中的排列顺序按照其在 DOM 树中出现的顺序。

### 16. next

next(element[, cssRule][, index = 0]) -> HTMLElement | undefined

返回元素的下一个兄弟元素。如果 cssRule 被指定，则在符合 cssRule 的兄弟元素中查找。如果 index 被指定，则返回指定位置的兄弟元素。

### 17. nextSiblings

nextSiblings(element) -> [HTMLElement...]

将位于 DOM 树中元素位置之后的所有兄弟元素作为数组返回。

### 18. observe

observe(element, eventName, handler[, useCapture = false]) -> HTMLElement

为元素注册事件句柄，等价于 Event.observe。

### 19. previous

previous(element[, cssRule][, index = 0]) -> HTMLElement | undefined

返回元素的上一个兄弟元素。如果 cssRule 被指定，则在符合 cssRule 的兄弟元素中查找。如果 index 被指定，则返回指定位置的兄弟元素。

### 20. previousSiblings

previousSiblings(element) -> [HTMLElement...]

将位于 DOM 树中元素位置之前的所有兄弟元素作为数组返回。

### 21. readAttribute

readAttribute(element, attribute) -> String | null

读取元素的某个属性值。

**22. remove**

`remove(element) -> HTMLElement`

将某个元素从 DOM 树中移除。

**23. removeClassName**

`removeClassName(element, className) -> HTMLElement`

移除元素的某个 CSS 类。

**24. replace**

`replace(element[, html]) -> HTMLElement`

使用给定的 HTML 字符串替换元素并返回包含新 HTML 内容的元素。

**25. scrollTo**

`scrollTo(element) -> HTMLElement`

滚动窗口使元素显示在视觉焦点上。

**26. setStyle**

`setStyle(element, styles) -> HTMLElement`

为元素设置 CSS 样式。styles 参数是一个 Hash 对象，其每一个成员的名字为 CSS 属性名，值为对应的 CSS 属性值。

**27. show**

`show(element) -> HTMLElement`

显示元素。

**28. siblings**

`siblings(element) -> [HTMLElement...]`

将元素的所有兄弟元素作为数组返回。

**29. stopObserving**

`stopObserving(element, eventName, handler) -> HTMLElement`

注销事件句柄，等价于 `Event.stopObserving`。

**30. toggle**

`toggle(element) -> HTMLElement`

切换元素的隐现状态。

**31. toggleClassName**

`toggleClassName(element, className) -> HTMLElement`

切换元素的某个 CSS 类的应用状态。

### 32. up

`up([cssRule][, index = 0]) -> HTMLElement | undefined`

返回元素的第一个祖先元素, 如果 `cssRule` 被指定, 则在符合 `cssRule` 的祖先元素中查找, 如果 `index` 被指定, 则返回位置为 `index` 的祖先元素。

### 33. update

`update(element[, newContent]) -> HTMLElement`

使用新的 HTML 内容更新元素内容。

### 34. visible

`visible(element) -> Boolean`

返回一个布尔值标识元素当前的隐现状态。

## C.2.6 Enumerable

### 1. all

`all([iterator = Prototype.K]) -> Boolean`

检查集合中所有的元素是否可以转换为 `true`。如果指定了迭代器 (`iterator`), 则会将集合中每个元素传递给迭代器执行, 并使用执行结果作为检查的依据。

### 2. any

`any([iterator = Prototype.K]) -> Boolean`

与 `all` 功能类似, 但如果集合中有一个元素可以转换为 `true`, 或者迭代器执行的结果中有一个为 `true`, 则返回 `true`。

### 3. collect

`collect(iterator) -> Array`

遍历集合元素, 并将每一个元素传递给迭代器执行, 然后将执行结果替换原集合元素。

### 4. detect

`detect(iterator) -> firstElement | undefined`

返回第一个传入迭代器执行结果为 `true` 的集合元素。

### 5. each

`each(iterator) -> Enumerable`

将集合中的所有元素传入迭代器执行。返回原集合对象。

### 6. entries

`entries() -> Array`

将集合转换为数组, 等价于 `toArray` 方法。



## 7. find

find(iterator) -> firstElement | undefined

等价于 detach。

## 8. findAll

findAll(iterator) -> Array

将所有被迭代器执行后返回 true 的集合元素作为数组返回。

## 9. grep

grep(regex[, iterator = Prototype.K]) -> Array

将所有可以被给定的正则表达式 (regex) 匹配的集合元素作为数组返回。如果迭代器 (iterator) 被提供, 则会使用迭代器执行的结果作为返回的数组元素。

## 10. include

include(object) -> Boolean

检查集合中是否包含某个元素。

## 11. inject

inject(accumulator, iterator) -> accumulatedValue

遍历集合, 并将 accumulator 和每个集合元素传入迭代器执行, 返回最终的 accumulator 值。

## 12. invoke

invoke(methodName[, arg...]) -> Array

遍历集合, 并执行每个集合元素的名为 methodName 的方法, 将所有运行结果作为数组返回。

## 13. map

map(iterator) -> Array

等价于 collect 方法。

## 14. max

max([iterator = Prototype.K]) -> maxValue

返回集合中最大的元素。如果迭代器被提供, 则返回通过迭代器执行后返回的值最大的元素。

## 15. member

member(object) -> Boolean

等价于 include 方法。

## 16. min

min([iterator = Prototype.K]) -> minValue

返回集合中最小的元素。如果迭代器被提供, 则返回通过迭代器执行后返回的值最小的元素。

### 17. pluck

pluck(propertyName) -> Array

将集合中所有元素的名为 propertyName 属性值作为数组返回。

### 18. reject

reject(iterator) -> Array

将集合中所有元素传入迭代器执行，并将所有执行结果为 false 的集合元素作为数组返回。

### 19. select

select(iterator) -> Array

等价于 findAll 方法。

### 20. sortBy

sortBy(iterator) -> Array

将集合中所有元素放入迭代器执行，并使用执行结果进行对集合进行排序，返回一个排序后的数组。

### 21. toArray

toArray()->Array

将集合转换为等价的数组。

## C.2.7 Event

### 1. element

Event.element(event) -> Element

获取当前发生事件的元素对象。

### 2. findElement

Event.findElement(event, tagName) -> Element

在发生事件的元素对象中查找第一个指定 tagName 的元素对象。

### 3. isLeftClick

Event.isLeftClick(event) -> Boolean

判断当前事件是否由鼠标左边点击触发。

### 4. observe

Event.observe(element, eventName, handler[, useCapture = false])

注册事件句柄。element 为元素对象，eventName 为事件名，handler 为事件句柄，userCapture 标识是否在事件的捕捉阶段进行处理。

### 5. pointerX

Event.pointerX(event) -> Number

返回事件发生时鼠标所在的 X 轴位置。

### 6. pointerY

Event.pointerY(event) -> Number

返回事件发生时鼠标所在的 Y 轴位置。

### 7. stop

Event.stop(event)

组织浏览器对事件的默认行为。

### 8. stopObserving

Event.stopObserving(element, eventName, handler[, useCapture = false])

注销事件句柄。element 为元素对象，eventName 为事件名，handler 为事件句柄，useCapture 标识是否在事件的捕捉阶段进行处理。

## C.2.8 Form

### 1. disable

disable(formElement) -> HTMLFormElement

禁用表单中所有的元素，返回表单元素。

### 2. enable

enable(formElement) -> HTMLFormElement

启用表单中所有的元素，返回表单元素。

### 3. findFirstElement

findFirstElement(formElement) -> HTMLElement

返回表单中第一个非隐藏的、非禁用的表单元素。

### 4. focusFirstElement

focusFirstElement(formElement) -> HTMLFormElement

将键盘输入焦点定位到表单的第一个元素上。

### 5. getElements

getElements(formElement) -> array

将表单中所有的元素作为数组返回。

## 6. `getInputs`

`getInputs(formElement [, type [, name]]) -> array`

将表单中所有的 Input 元素作为数组返回。可以指定元素的 `type` 属性和 `name` 属性。

## 7. `request`

`request([options]) -> new Ajax.Request`

使用 Ajax 请求发送表单中的数据。options 为请求选项，参见 `Ajax.options`。

## 8. `reset`

`reset(formElement) -> HTMLFormElement`

重置表单。

## 9. `serialize`

`serialize(formElement) -> string`

将表单数据序列化输出。

## C.2.9 Function

### 1. `bind`

`bind(thisObj[, arg...]) -> Function`

返回一个新函数，该函数被调用时，会将 `thisObj` 作为原函数的当前对象来调用原函数。`bind` 方法的第二个和以后的参数会被作为参数传递给原函数。

### 2. `bindAsEventListener`

`bindAsEventListener(thisObj[, arg...]) -> Function`

与 `bind` 方法作用一样，只是在调用原函数时，会将当前事件对象作为第一个参数传递给原函数。

## C.2.10 Hash

### 1. `each`

`each(iterator) -> Hash`

遍历 Hash 对象中的每一个元素，并将元素传入迭代器（`iterator`）执行。传入迭代器的元素是一个对象，其 `key` 属性保存了元素的名字，`value` 属性保存了元素的值。

### 2. `keys`

`keys() -> [String...]`

将 Hash 对象中所有元素的名字作为数组返回。

### 3. remove

`remove(key) -> value`

`remove(key1, key2...) -> Array`

移除 Hash 对象中指定名称的元素。

### 4. toQueryString

`toQueryString() -> String`

串联 Hash 对象中所有元素的名和值，返回一个查询字符串。

### 5. values

`values() -> Array`

将 Hash 对象中所有元素的值作为数组返回。

## C.2.11 Insertion

### 1. After

`new Insertion.After(element, html)`

将 html 内容插入到指定元素之后。

### 2. Before

`new Insertion.Before(element, html)`

将 html 内容插入到指定元素之前。

### 3. Bottom

`new Insertion.Bottom(element, html)`

将 html 内容作为指定元素的最后一个直接子元素插入到 DOM 树中。

### 4. Top

`new Insertion.Top(element, html)`

将 html 内容作为指定元素的第一个直接子元素插入到 DOM 树中。

## C.2.12 Number

### 1. times

`times(iterator) -> Number`

将 0 到当前数字减 1 的数字依次传递给迭代器执行。



## 2. toColorPart

toColorPart() -> String

将数字转换为对应的十六进制颜色代码。

## C.2.13 Object

### 1. clone

Object.clone(obj) -> Object

复制对象返回一个新的对象。

### 2. extend

Object.extend(dest, src) -> alteredDest

将 src 对象中的所有成员复制到 dest 对象中。

### 3. keys

Object.keys(obj) -> [String...]

将对象中所有成员的名字作为一个数组返回。

### 4. toJSON

toJSON()->String

将对象转换为对应的 JSON 字符串。

### 5. values

Object.values(obj) -> Array

将对象中所有成员的值作为一个数组返回。

## C.2.14 ObjectRange

### include

include(value) -> Boolean

检查 ObjectRange 对象中是否包含某个指定的值。

## C.2.15 PeriodicalExecuter

### stop

stop()

停止定时执行器。

## C.2.16 String

### 1. blank

blank() -> Boolean

检查字符串是否为空白，包括空字符串和空白字符串。

### 2. capitalize

capitalize() -> string

将字符串的第一个字符转换为大写，其余的字符转换为小写。

### 3. empty

empty() -> Boolean

检查字符串是否为空。

### 4. endsWith

endsWith(substring) -> Boolean

检查字符串是否以 substring 字符串结尾。

### 5. escapeHTML

escapeHTML() -> string

将字符串中包含的 HTML 符号转换为对应的实体。

### 6. evalJSON

evalJSON([sanitize]) -> object

将 JSON 格式的字符串转换为对应的对象。

### 7. evalScripts

evalScripts() -> [returnedValue...]

将字符串中所有<script>标签内的内容作为 JavaScript 语句执行，并将返回结果按照<script>标签出现的顺序组成数组返回。

### 8. extractScripts

extractScripts() -> [script...]

取出字符串中所有的<script>标签包含的字符串，作为数组返回。

### 9. gsub

gsub(pattern, replacement) -> string

使用 replacement 指定的字符串替换正则表达式 pattern 匹配的原字符串中的部分。如果 replacement 参数为一个函数，则会将匹配的结果作为参数传递给该函数，并使用函数的返回值来进行替换。

**10. include**

include(substring) -> Boolean

检查字符串中是否包含指定的 substring 子字符串。

**11. scan**

scan(pattern, iterator) -> string

将字符串中每个被 pattern 正则表达式匹配的字符串作为参数传递给 iterator 迭代器执行。

**12. startsWith**

startsWith(substring) -> Boolean

检查字符串是否由 substring 子字符串开头。

**13. strip**

strip() -> string

取出字符串开头和结尾的所有空格。

**14. stripScripts**

stripScripts() -> string

去除字符串中包含的所有<script>标签及其包含的内容。

**15. stripTags**

stripTags() -> string

去除字符串中所有的 HTML 标签。

**16. toArray**

toArray() -> [character...]

将数组中每个字符作为数组的一个元素返回一个数组。

**17. toJSON**

toJSON() -> String

将字符串转换为对应的 JSON 字符串。

## C.2.17 Template

**evaluate**

evaluate(object) -> String

使用给定的对象数组来格式化输出字符串。

[ G e n e r a l I n f o r m a t i o n ]

书名= A J A X从入门到精通

作者= 陈华编著

页数= 6 2 4

出版社= 北京市：清华大学出版社

出版日期= 2 0 0 8 . 0 9

S S号= 1 2 0 6 4 6 4 3

D X号= 0 0 0 0 0 6 6 1 0 6 4 3

U R L = h t t p : / / b o o k 2 . d u x i u . c o m / b o o k D e t a i l . j s p ? d

x N u m b e r = 0 0 0 0 0 6 6 1 0 6 4 3 & d = 6 3 5 5 4 2 0 B 7 8 2 D C 2 A 6 6 D A 2  
B 8 E D 0 4 9 1 3 D 3 5

准备篇

第1章 什么是A j a x

1.1 A j a x 概述

1.1.1 关键技术：X M . H t t p R e q u e s t

1.1.2 A j a x 名词的由来

1.2 经典案例1：搜索关键词建议系统（G o o g l e S u g g e s t）

1.3 经典案例2：优秀的电子邮件服务系统（G m a i l

1.3.1 注册G m a i l

1.3.2 G r m a i l 主界面

1.3.3 I n b o x（收件箱）工作区域

1.3.4 W e b C l i p（网络剪辑）

1.3.5 邮件操作区域

1.3.6 操作邮件

1.4 第一个A j a x 应用程序：H e l l o W o r l d！

1.5 小结

第2篇 A j a x 技术构成篇

第2章 J a v a S c r i p t：A j a x 的开发语言

2.1 J a v a S c r i p t 概述

2.1.1 对J a v a S c r i p t 的误解

2.1.2 J a v a 5 c r i p t 的版本

2.2 数据类型和值

2.2.1 数字

2.2.2 字符串

2.2.3 布尔值

2.2.4 函数

2.2.5 对象

2.2.6 数组

2.2.7 n u l l 值

2.2.8 u n d e f i n e d 值

2.3 J a v a S c r i p t 的变量

2.3.1 变量的类型

2.3.2 变量的声明

2.3.3 变量的作用域

2.3.4 基本类型和引用类型

2.4 表达式和运算符

2.4.1 表达式

2.4.2 算术运算符

2.4.3 相等运算符

2.4.4 关系运算符

2.4.5 赋值运算符

2.4.6 逻辑运算符

2.4.7 字符串运算符

2.4.8 其他运算符

2.5 语句

2.5.1 声明变量：v a r 语句



- 2.5.2 流程控制：i f 语句
- 2.5.3 流程控制：e l s e i f 语句
- 2.5.4 流程控制：s w i t c h 语句
- 2.5.5 循环：w h i l e 语句
- 2.5.6 循环：d o / w h i l e 语句
- 2.5.7 循环：f o r 语句
- 2.5.8 遍历：f o r / i n 语句
- 2.5.9 控制语句：b r e a k 语句
- 2.5.10 控制语句：c o n t i n u e 语句
- 2.5.11 函数语句：f u n c t i o n 语句
- 2.5.12 函数返回值：r e t u r n 语句
- 2.5.13 抛出异常：t h r o w 语句
- 2.5.14 异常处理：t r y / c a t c h / f i n a l l y 语句
- 2.5.15 空语句
- 2.6 J a v a S c r i p t 的函数
  - 2.6.1 函数的定义和调用
  - 2.6.2 实际参数列表：a r g u m e n t s 属性的使用
- 2.7 大小写敏感性
- 2.8 J a v a S c r i p t 的注释
- 2.9 J a v a S c r i p t 的保留字
- 第3章 J a v a S c r i p t 的常用对象
  - 3.1 保存多个数据元素的容器：A r r a y 对象
    - 3.1.1 数组的创建
    - 3.1.2 得到数组的长度
    - 3.1.3 添加、删除和替换数组元素
    - 3.1.4 得到数组片段
    - 3.1.5 反转数组
    - 3.1.6 将数组转换成字符串
    - 3.1.7 数组元素的排序
  - 3.2 字符串的处理：S t r i n g 对象
    - 3.2.1 获取字符串的长度
    - 3.2.2 字符串的截取
    - 3.2.3 字符串的替换
    - 3.2.4 大小写转换
    - 3.2.5 将字符串转换成数组
    - 3.2.6 拼接字符串的优化方法
  - 3.3 正则表达式：R e g E x p 对象
    - 3.3.1 创建正则表达式
    - 3.3.2 正则表达式的语法规则
    - 3.3.3 正则表达式的属性
    - 3.3.4 用于模式匹配的 S t r i n g 方法
    - 3.3.5 用于模式匹配的 R e g E x p 方法
    - 3.3.6 常用正则表达式
  - 3.4 日期和时间：D a t e 对象
    - 3.4.1 D a t e 对象的创建
    - 3.4.2 读取和设置日期及时间的各个部分
    - 3.4.3 日期和时间的换算
  - 3.5 复杂的数学运算：M a t h 对象
    - 3.5.1 小数的取整
    - 3.5.2 得到随机数

- 3 . 5 . 3 幂运算
    - 3 . 5 . 4 最大值和最小值
  - 3 . 6 操作浏览器窗口：w i n d o w 对象
    - 3 . 6 . 1 使用对话框
    - 3 . 6 . 2 改变窗口状态栏的信息
    - 3 . 6 . 3 延迟执行和定时执行
    - 3 . 6 . 4 U R L 的获取和跳转
    - 3 . 6 . 5 历史记录的前进和后退
    - 3 . 6 . 6 控制窗口的大小和位置
    - 3 . 6 . 7 打开和关闭窗口
    - 3 . 6 . 8 获得焦点和失去焦点
    - 3 . 6 . 9 取得用户显示器的信息
    - 3 . 6 . 1 0 取得用户浏览器的信息
  - 3 . 7 操作H T M L 文档：d o c u m e n t 对象
    - 3 . 7 . 1 文档的输出
    - 3 . 7 . 2 文档的标题
    - 3 . 7 . 3 文档的图像
    - 3 . 7 . 4 文档的超链接
    - 3 . 7 . 5 文档的表单
  - 3 . 8 应用实例
    - 3 . 8 . 1 在网页上显示自动更新的日期和时间
    - 3 . 8 . 2 一个简单的小游戏：L u c k y   S e v e n ( 幸运 7 )
  - 3 . 9 小结
- 第 4 章 D O M 文档对象模型介绍
- 4 . 1 基本概念
    - 4 . 1 . 1 树形结构
    - 4 . 1 . 2 结点的类型和组成
    - 4 . 1 . 3 结点之间的关系
  - 4 . 2 结点的引用
    - 4 . 2 . 1 根据 i d 属性引用结点
    - 4 . 2 . 2 根据 n a m e 属性引用结点
    - 4 . 2 . 3 根据标签名引用结点
    - 4 . 2 . 4 引用父结点
    - 4 . 2 . 5 引用子结点
    - 4 . 2 . 6 引用相邻的结点
  - 4 . 3 结点的操作
    - 4 . 3 . 1 创建元素结点
    - 4 . 3 . 2 创建文本结点
    - 4 . 3 . 3 添加结点
    - 4 . 3 . 4 插入子结点
    - 4 . 3 . 5 替换子结点
    - 4 . 3 . 6 复制结点
    - 4 . 3 . 7 删除子结点
    - 4 . 3 . 8 读取结点属性
    - 4 . 3 . 9 添加和修改属性结点
    - 4 . 3 . 1 0 删除属性结点
  - 4 . 4 控制元素的样式
    - 4 . 4 . 1 获取和设置元素的 c s s 类
    - 4 . 4 . 2 获取和设置元素样式
  - 4 . 5 事件处理

	4 . 5 . 1 事件模型和传播机制
	4 . 5 . 2 注册事件处理程序
	4 . 5 . 3 注销事件处理程序
	4 . 5 . 4 事件对象
	4 . 5 . 5 常用事件
4 . 6	应用实例
	4 . 6 . 1 悬浮的广告
	4 . 6 . 2 可拖动的层
4 . 7	小结
第5章	开发A j a x 应用程序需要使用的工具
5 . 1	开发工具：A p t a n a
	5 . 1 . 1 A p t a n a 的下载和安装
	5 . 1 . 2 A p t a n a 的界面介绍
	5 . 1 . 3 A p t a n a 的使用
	5 . 1 . 4 A p t a n a 的更新
	5 . 1 . 5 A p t a n a 的卸载
5 . 2	F i r e f o x
	5 . 2 . 1 错误控制台：E r r o r C o n s o l e
	5 . 2 . 2 优秀的调试插件：F i r e b u g
5 . 3	H T T P 调试工具：F i d d l e r
	5 . 3 . 1 F i d d l e r 的下载和安装
	5 . 3 . 2 使用F i d d l e r
5 . 4	小结
第3篇	A j a x 应用技术分析篇
第6章	H e l l o W o r l d ! 分析
6 . 1	X M L H t t p R e q u e s t 对象详解
	6 . 1 . 1 初始化请求
	6 . 1 . 2 设置请求的H T T P 头信息
	6 . 1 . 3 发送请求
	6 . 1 . 4 获取请求的当前状态
	6 . 1 . 5 指定请求状态改变时的事件处理句柄
	6 . 1 . 6 返回当前请求的H T T P 状态码
	6 . 1 . 7 从返回信息中获取指定的H T T P 头
	6 . 1 . 8 获取返回信息的所有H T T P 头
	6 . 1 . 9 取得返回的数据
	6 . 1 . 1 0 取消当前请求
6 . 2	搭建基本的A j a x 开发框架
	6 . 2 . 1 创建X M L H t t p R e q u e s t 对象
	6 . 2 . 2 发送请求和回调函数
	6 . 2 . 3 一个封装好的基本A j a x 应用程序开发框架
6 . 3	小结
第7章	完善的A j a x 应用程序：A j a x 留言本
7 . 1	留言本的需求
7 . 2	留言本的基本设计
	7 . 2 . 1 系统环境
	7 . 2 . 2 留言的数据和操作
	7 . 2 . 3 数据库设计
	7 . 2 . 4 后台功能模块
	7 . 2 . 5 前台脚本功能模块
	7 . 2 . 6 系统文件结构和文件清单

### 7.3 留言本的实现

#### 7.3.1 创建数据库

#### 7.3.2 完成前台界面：index.html

#### 7.3.3 完成前台界面：样式表

#### 7.3.4 完成后台功能模块：数据库操作模块

#### 7.3.5 完成后台功能模块：留言本逻辑处理模块

#### 7.3.6 完成后台功能模块：接口模块

#### 7.3.7 完成前台功能模块：基本Ajax功能模块

#### 7.3.8 完成前台功能模块：读取和发送留言

#### 7.3.9 整合留言本程序

### 7.4 留言本的功能测试

### 7.5 小结

## 第8章 调试技巧

### 8.1 深入解析Firebug的调试功能

#### 8.1.1 检查常规错误

#### 8.1.2 完善的log功能

#### 8.1.3 控制台的命令行功能

#### 8.1.4 断点、单步执行和变量信息

#### 8.1.5 在其他浏览器中使用Firebug的控制台

#### 8.1.6 屏蔽测试代码

### 8.2 使用Aptana的集成调试功能

#### 8.2.1 配置集成调试环境

#### 8.2.2 启动调试

#### 8.2.3 断点、单步执行和变量信息

#### 8.2.4 使用console.log和dump输出文本信息

#### 8.2.5 使用aptana.trace输出调用堆栈信息

#### 8.2.6 使用断言

#### 8.2.7 屏蔽调试代码

### 8.3 小结

## 第9章 常见问题

### 9.1 编码的处理

#### 9.1.1 文件编码与声明编码

#### 9.1.2 Ajax请求乱码

#### 9.1.3 发送数据乱码

### 9.2 控制缓存

### 9.3 选择合适的请求方式

### 9.4 控制多个Ajax请求

#### 9.4.1 轮询模式

#### 9.4.2 事件响应模式

### 9.5 Ajax请求的安全性

#### 9.5.1 身份验证

#### 9.5.2 防范SQL注入

#### 9.5.3 防范JavaScript注入

### 9.6 小结

## 第4篇 Ajax应用进阶篇

## 第10章 数据的组织方式：XML

### 10.1 XML概述

### 10.2 XML语法规则

#### 10.2.1 XML声明

#### 10.2.2 根节点

- 1 0 . 2 . 3 开始和结束标记
  - 1 0 . 2 . 4 属性
  - 1 0 . 2 . 5 合理地嵌套包含
  - 1 0 . 2 . 6 大小写敏感性
  - 1 0 . 2 . 7 空白被保留
  - 1 0 . 2 . 8 X M 的注释
  - 1 0 . 2 . 9 元素的命名
  - 1 0 . 2 . 1 0 转义字符
  - 1 0 . 2 . 1 1 C D A T A 部件
- 1 0 . 3 X M L 命名空间
- 1 0 . 4 X M L S c h e m a
  - 1 0 . 4 . 1 基本示例
  - 1 0 . 4 . 2 定义元素
  - 1 0 . 4 . 3 简单类型
  - 1 0 . 4 . 4 复合类型
  - 1 0 . 4 . 5 定义属性
  - 1 0 . 4 . 6 默认值
  - 1 0 . 4 . 7 约束特殊值
  - 1 0 . 4 . 8 列表类型
  - 1 0 . 4 . 9 联合类型
  - 1 0 . 4 . 1 0 匿名类型定义
  - 1 0 . 4 . 1 1 简单的复合类型
  - 1 0 . 4 . 1 2 混合内容
  - 1 0 . 4 . 1 3 任意类型
  - 1 0 . 4 . 1 4 分组和引用
  - 1 0 . 4 . 1 5 命名冲突
  - 1 0 . 4 . 1 6 关联 X M L 与 X M L S c h e m a
- 1 0 . 5 X M L D O M
  - 1 0 . 5 . 1 创建 X M L D O M 对象
  - 1 0 . 5 . 2 加载 X M L 文档
  - 1 0 . 5 . 3 加载 X M L 片段
  - 1 0 . 5 . 4 取得 X M L 内容
- 1 0 . 6 强大的检索工具：X P a t h
  - 1 0 . 6 . 1 基本示例
  - 1 0 . 6 . 2 选取节点
  - 1 0 . 6 . 3 谓词
  - 1 0 . 6 . 4 通配符
  - 1 0 . 6 . 5 使用多个路径
  - 1 0 . 6 . 6 坐标轴
  - 1 0 . 6 . 7 运算符
  - 1 0 . 6 . 8 路径表达式的步语法
  - 1 0 . 6 . 9 X P a t h 函数
- 1 0 . 7 格式化 X M L 工具：X S L
  - 1 0 . 7 . 1 基本示例
  - 1 0 . 7 . 2 X S L 声明
  - 1 0 . 7 . 3 使用模板
  - 1 0 . 7 . 4 取得数据
  - 1 0 . 7 . 5 使用 f o r - e a c h 元素代替模板
  - 1 0 . 7 . 6 使用 s o r t 元素进行排序
  - 1 0 . 7 . 7 流程控制



- 1 0 . 7 . 8 创建元素和属性
  - 1 0 . 7 . 9 指定输出格式
- 1 0 . 8 在客户端格式化XML
- 1 0 . 9 跨浏览器的XML开发框架：zXML
- 1 0 . 1 0 应用实例：Ajax文章列表程序（XML）
  - 1 0 . 1 0 . 1 确认需求
  - 1 0 . 1 0 . 2 系统设计
  - 1 0 . 1 0 . 3 系统实现：创建数据库
  - 1 0 . 1 0 . 4 系统实现：完成后台模块
  - 1 0 . 1 0 . 5 系统实现：完成前台界面
  - 1 0 . 1 0 . 6 系统实现：完成前台模块
  - 1 0 . 1 0 . 7 系统实现：编写xsl样式表
  - 1 0 . 1 0 . 8 整合系统
  - 1 0 . 1 0 . 9 系统测试
- 1 0 . 1 1 小结
- 第 1 1 章 数据的组织方式：JSON
  - 1 1 . 1 JSON的语法结构
    - 1 1 . 1 . 1 JSON的基本结构
    - 1 1 . 1 . 2 JSON中值的类型
  - 1 1 . 2 JSON的语言支持
    - 1 1 . 2 . 1 在JavaScript中读取JSON数据
    - 1 1 . 2 . 2 在JavaScript中输出JSON数据
    - 1 1 . 2 . 3 在PHP中使用JSON
    - 1 1 . 2 . 4 在C#中输出JSON数据
    - 1 1 . 2 . 5 在C#中输出带类型说明的JSON数据
    - 1 1 . 2 . 6 在C#中读取JSON数据
    - 1 1 . 2 . 7 更多语言支持
  - 1 1 . 3 JSON的优点和不足
    - 1 1 . 3 . 1 JSON的优点
    - 1 1 . 3 . 2 JSON的不足
  - 1 1 . 4 将XML转换为JSON
  - 1 1 . 5 小结
- 第 1 2 章 JavaScript面向对象编程（OOP）
  - 1 2 . 1 JavaScript中的类
    - 1 2 . 1 . 1 创建类
    - 1 2 . 1 . 2 类的属性
    - 1 2 . 1 . 3 类的方法
    - 1 2 . 1 . 4 公有属性和私有属性
    - 1 2 . 1 . 5 公有方法和私有方法
    - 1 2 . 1 . 6 静态属性和静态方法
    - 1 2 . 1 . 7 原型对象prototype
  - 1 2 . 2 JavaScript中的继承
    - 1 2 . 2 . 1 对象冒充
    - 1 2 . 2 . 2 使用对象冒充实现继承
    - 1 2 . 2 . 3 继承原型对象中的属性和方法
    - 1 2 . 2 . 4 封装继承方法
  - 1 2 . 3 更多技巧
    - 1 2 . 3 . 1 属性的封装
    - 1 2 . 3 . 2 实现多态
    - 1 2 . 3 . 3 命名空间

- 1 2 . 3 . 4 实现短类名
- 1 2 . 4 使用 J S V M 进行代码组织
  - 1 2 . 4 . 1 下载和配置 J S V M
  - 1 2 . 4 . 2 路径和文件名的约定
  - 1 2 . 4 . 3 编写类
  - 1 2 . 4 . 4 类的按需加载
  - 1 2 . 4 . 5 在类中引用其他类

#### 1 2 . 5 小结

### 第 1 3 章 跨浏览器的兼容性问题

- 1 3 . 1 使用 `getElementById` 代替 `idName`
- 1 3 . 2 表单元素的引用问题
- 1 3 . 3 访问集合对象成员的问题
- 1 3 . 4 读取自定义属性的问题
- 1 3 . 5 常量的定义问题
- 1 3 . 6 `input` 元素的 `type` 属性读写问题
- 1 3 . 7 模态窗口的问题
- 1 3 . 8 `frame` 的操作问题
- 1 3 . 9 `innerText` 的问题
- 1 3 . 1 0 对父元素的引用问题
- 1 3 . 1 1 `getElementsByTagName` 的问题
- 1 3 . 1 2 `outerText` 的问题
- 1 3 . 1 3 `outerHTML` 的问题
- 1 3 . 1 4 小结

### 第 1 4 章 `Prototype.js` 框架介绍

- 1 4 . 1 `Prototype.js` 常用方法介绍
  - 1 4 . 1 . 1 使用 `$` 方法代替 `document.getElementById`
  - 1 4 . 1 . 2 使用 `$$` 方法获得元素引用
  - 1 4 . 1 . 3 根据 `css` 类名取得元素集合
  - 1 4 . 1 . 4 使用 `Try.these` 尝试运行多个函数
  - 1 4 . 1 . 5 使用 `$F` 方法来获得表单元素的值
  - 1 4 . 1 . 6 数组迭代的简化方法
  - 1 4 . 1 . 7 设置元素的样式
  - 1 4 . 1 . 8 切换元素的隐现状态
  - 1 4 . 1 . 9 序列化表单值
  - 1 4 . 1 . 1 0 转换 HTML 标签
- 1 4 . 2 基于 `Prototype.js` 的类和继承
  - 1 4 . 2 . 1 使用 `Class.create()` 创建类
  - 1 4 . 2 . 2 `Prototype.js` 中的继承
- 1 4 . 3 `Prototype.js` 中的事件处理
  - 1 4 . 3 . 1 注册事件处理函数
  - 1 4 . 3 . 2 注销事件处理函数
- 1 4 . 4 `Prototype.js` 的 Ajax 功能
  - 1 4 . 4 . 1 `Ajax.Request` 方法详解
  - 1 4 . 4 . 2 用 `Ajax.Updater` 更新界面
  - 1 4 . 4 . 3 用 `Ajax.PeriodicalUpdater` 定时更新
  - 1 4 . 4 . 4 使用 `Form.request` 方法无刷新提交表单
- 1 4 . 5 基于 `Prototype.js` 的留言本程序
- 1 4 . 6 小结

y l d

界面

第 1 5 章	关于架构的思考
1 5 . 1	R E S T 架构模式
1 5 . 2	W e b 应用程序的发展史
1 5 . 2 . 1	提供静态文档的W e b 站点
1 5 . 2 . 2	早期的动态W e b 应用程序
1 5 . 2 . 3	融入式W e b 应用程序
1 5 . 3	A j a x 为R E S T 带来新的契机
1 5 . 3 . 1	缓存A j a x 程序引擎
1 5 . 3 . 2	缓存A j a x 数据
1 5 . 4	小结
第 1 6 章	A j a x 的缺陷及补救
1 6 . 1	搜索引擎的收录问题
1 6 . 1 . 1	问题产生的原因
1 6 . 1 . 2	解决办法
1 6 . 2	前进和后退的问题
1 6 . 2 . 1	问题产生的原因
1 6 . 2 . 2	F i r e f o x 下的解决办法
1 6 . 2 . 3	在 I E 下的解决办法
1 6 . 3	小结
第 5 篇	A j a x 实战篇
第 1 7 章	A j a x 高级表单验证程序
1 7 . 1	确定需求
1 7 . 2	基本设计
1 7 . 3	代码实现
1 7 . 3 . 1	实现E v e n t M a n a g e r 类
1 7 . 3 . 2	实现R e q u e s t 类
1 7 . 3 . 3	实现V a l i d a t o r 类
1 7 . 3 . 4	实现F o r m I t e m V a l i d a t o r 类
1 7 . 3 . 5	实现F o r m V a l i d a t o r 类
1 7 . 4	测试
第 1 8 章	A j a x 动态联动菜单
1 8 . 1	确定需求
1 8 . 2	基本设计
1 8 . 3	实例代码
1 8 . 4	测试
第 1 9 章	A j a x 聊天室
1 9 . 1	确定需求
1 9 . 2	基本设计
1 9 . 2 . 1	系统结构
1 9 . 2 . 2	实体及数据库设计
1 9 . 2 . 3	后台功能模块
1 9 . 2 . 4	请求控制器
1 9 . 2 . 5	前台界面
1 9 . 2 . 6	前台功能模块
1 9 . 3	实例代码
1 9 . 3 . 1	建立数据库
1 9 . 3 . 2	实现后台功能模块
1 9 . 3 . 3	请求控制器
1 9 . 3 . 4	界面H T M L 和C S S 代码
1 9 . 3 . 5	前台功能模块

19.3.6 加入表情图标

19.3.7 整合程序

19.4 测试

附录A JavaScript速查手册

附录B HTML DOM速查手册

附录C Prototype速查手册