

PACKT  
PUBLISHING



From Technologies to solutions

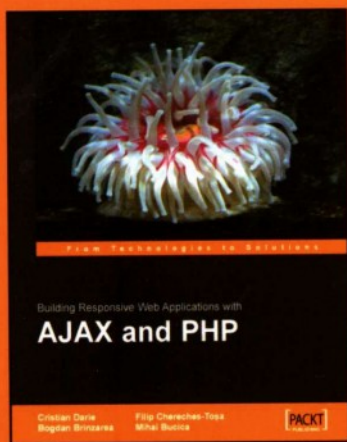
AJAX and PHP Building Responsive Web Applications

# Ajax与PHP

## Web 开发

[罗] Cristian Darie Filip Cherecheș-Toșa 著  
Bogdan Brinzarea Mihai Bucica  
王德民 王新颖 刘昕 译

 人民邮电出版社  
POSTS & TELECOM PRESS



在掌握基础知识后，本书将带您漫步在现实生活的案例学习中，这些例子包括相关模式和最好的体验，可以应用于您自己的应用程序中：

- 服务器端的表单验证；
- 在线聊天协作工具；
- 定制预录入文本条目解决方案；
- 使用 SVG 实时绘图；
- 数据库支持的、可编辑的、可定制的数据表格；
- RSS 聚集器应用程序；
- 使用 <http://scricp.aculo.us> 中包含的 JavaScript 工具集完成服务器端管理的具有拖放功能的分类列表。

在本书的网站 <http://ajaxphp.packtpub.com> 上，包含了本书的内容更新以及 Ajax 学习实例的在线演示。该网站上还有附录 B 和附录 C，可带您进入利用强大的工具进行调试、改进和定制代码以及用 XPath 和 XSLT 共同工作的领域。

分类建议：计算机 / 网络技术 / PHP  
人民邮电出版社网址：[www.ptpress.com.cn](http://www.ptpress.com.cn)

ISBN 978-7-115-15785-0



9 787115 157850 &gt;

ISBN 978-7-115-15785-0/TP

定价：39.00 元



From Technologies to solutions

# Ajax 与 PHP

## Web 开发

[罗] Cristian Darie Filip Cherecheș-Toșa 著  
Bogdan Brinzarea Mihai Bucica

王德民 王新颖 刘昕 译

人民邮电出版社  
北京



## 图书在版编目 (CIP) 数据

Ajax 与 PHP Web 开发 / (罗) 达里 (Darie, C.) 等著; 王德民, 王新颖, 刘昕译. —北京: 人民邮电出版社, 2007.4

ISBN 978-7-115-15785-0

I. A... II. ①达...②王...③王...④刘... III. ①计算机网络—程序设计  
②PHP 语言—程序设计 IV. TP393.09 TP312

中国版本图书馆 CIP 数据核字 (2007) 第 009979 号

### 版 权 声 明

Copyright ©Packt Publishing 2006. First published in the English language under the title  
AJAX and PHP: Building Responsive Web Applications.  
All Rights Reserved.

本书由英国 Packt Publishing 公司授权人民邮电出版社出版。未经出版者书面许可, 对本书的任何部分不得以  
任何方式或任何手段复制和传播。

版权所有, 侵权必究。

### 内 容 提 要

本书指导读者使用 Ajax、PHP 及其他相关技术建立快速响应的网页。本书不仅从 Ajax 的客户端和服务器端技术两个方面指导读者逐步掌握 Ajax 基础应用, 还通过实例详细演示了 Ajax 表单验证、Ajax 聊天室、Ajax Suggest、使用 SVG 实现的 Ajax 实时绘图程序、Ajax Grid、Ajax RSS 阅读器和 Ajax 拖放等的实现过程。同时还介绍了在 Windows 和 UNIX 系统下的应用程序环境配置和相关名词解释。

本书适合已经掌握 PHP、XML、JavaScript 和 MySQL 基础知识, 并想要了解 Ajax 核心和工作原理的读者阅读。

## Ajax 与 PHP Web 开发

- 
- ◆ 著 [罗] Cristian Darie Filip Cherecheș-Toșa  
Bogdan Brinzarea Mihai Bucica
  - 译 王德民 王新颖 刘 昕
  - 责任编辑 刘映欣
  - ◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号  
邮编 100061 电子函件 315@ptpress.com.cn  
网址 <http://www.ptpress.com.cn>  
北京密云春雷印刷厂印刷  
新华书店总店北京发行所经销
  - ◆ 开本: 800×1000 1/16  
印张: 20  
字数: 451 千字 2007 年 4 月第 1 版  
印数: 1-5 000 册 2007 年 4 月北京第 1 次印刷

著作权合同登记号 图字: 01-2006-5371 号

ISBN 978-7-115-15785-0/TP

定价: 39.00 元

读者服务热线: (010)67132705 印装质量热线: (010)67129223

# 译者序

---


Ajax 全称为 Asynchronous JavaScript and XML (异步 JavaScript 和 XML)，是一种开发交互式网页应用的新兴技术。Ajax 技术集成了目前在浏览器中通过 JavaScript 脚本可以使用的所有技术，并以一种崭新的方式来使用这些技术，使得古老的 B/S (浏览器/服务器) 方式的 Web 开发焕发了新的活力。

应该说，使用 Ajax 开发应用程序在技术上已经不成问题，主要问题在于 Ajax 目前还不普及，熟练掌握 Ajax 还有一定的困难，用 Ajax 开发还远不如做一个 JSP/ASP 方便。然而，有需求出现就会有人提供相应的产品和相应的解决方案，目前已经有了一些基于 Ajax 技术的开发构件和开发平台出现。不论是在 .NET 下还是在 Java 环境下，都有这类产品可以帮助用户轻松地搭建基于 Ajax 技术的富客户端应用 (或者富因特网应用 RIA)，从而使用户可以轻松地开发出基于 Ajax 技术的 Web 应用，就像过去用成熟控件开发 C/S (客户端/服务器) 应用一样。因此，Ajax 技术的应用将会飞速地发展。

本书深入浅出地剖析了 Ajax 的各个方面，包括 Ajax 的基础技术，如 JavaScript、DOM、CSS 以及 Ajax 的核心技术 XMLHttpRequest。书中以服务器端 PHP 及数据库 MySQL 应用程序为依托，详细讲解了 Ajax 技术应用上的各种实例，包括 Ajax 表单验证、Ajax 聊天室、Ajax Suggest、使用 SVG 实现 Ajax 实时绘图程序、Ajax Grid、Ajax RSS 阅读器和 Ajax 拖放等。同时，本书还介绍了如何在 Windows 和 UNIX 系统下进行应用程序环境配置及相关的名词解释。相信无论是首次接触 Ajax 技术的初学者，还是已经有一定开发经验的技术人员，都可以从本书中获得丰富信息。

感谢向林艳、曹艳、孟丽霞、吕贵林、张大鹏、裴立秋、毕永东等积极参与本书的翻译工作。虽然在翻译过程中我们尽量做到尊重原意、翻译准确，并尽力修正了一些原文的小错误，但由于水平有限，不当和疏漏之处在所难免，敬请广大读者不吝指正。

译者  
2007年1月



# 目 录

<b>第 1 章 Ajax 与未来的 Web 应用程序</b> .....	1
1.1 通过 Web 发布的功能 .....	2
1.2 20 世纪 90 年代开始出现网站 .....	3
1.2.1 HTTP 与 HTML .....	3
1.2.2 PHP 及其他服务器端技术 .....	5
1.2.3 JavaScript 及其他客户端技术 .....	5
1.2.4 缺少的是什么 .....	6
1.3 理解 Ajax .....	7
1.4 使用 Ajax 和 PHP 建立一个简单应用程序 .....	10
1.5 小结 .....	22
<b>第 2 章 JavaScript 下灵活的客户端技术</b> .....	23
2.1 JavaScript 和 Document Object Model .....	23
2.2 JavaScript 事件和 DOM .....	27
2.3 关于 DOM 的进一步介绍 .....	31
2.4 JavaScript、DOM 和 CSS .....	34
2.5 使用 XMLHttpRequest 对象 .....	38
2.5.1 创建 XMLHttpRequest 对象 .....	39
2.5.2 使用 XMLHttpRequest 初始化服务器请求 .....	43
2.5.3 服务器响应处理 .....	46
2.6 使用 XML 结构 .....	54
2.6.1 处理更多的错误和抛出异常 .....	60
2.6.2 建立 XML 结构 .....	63
2.7 小结 .....	64
<b>第 3 章 使用 PHP 和 MySQL 实现服务器端技术</b> .....	65
3.1 PHP 与 DOM .....	65
3.2 参数传递与 PHP 错误处理 .....	72

3.3	连接远程服务器与 JavaScript 的安全性	83
3.4	使用代理服务器脚本	89
3.5	重复异步请求框架	96
3.6	使用 MySQL	108
3.6.1	创建数据库表	109
3.6.2	数据操作	111
3.6.3	连接数据库并执行查询操作	112
3.7	程序封装与程序结构	117
3.8	小结	130
<b>第 4 章</b>	<b>Ajax 表单验证</b>	<b>131</b>
4.1	实现 Ajax 表单验证	132
4.2	小结	161
<b>第 5 章</b>	<b>Ajax 聊天</b>	<b>162</b>
5.1	Ajax 聊天简介	162
5.2	实现 Ajax 聊天	163
5.3	小结	186
<b>第 6 章</b>	<b>Ajax 建议和自动完成</b>	<b>187</b>
6.1	Ajax 建议和自动完成简介	187
6.2	实现 Ajax 建议和自动完成	188
6.3	小结	215
<b>第 7 章</b>	<b>使用 SVG 实现 Ajax 实时绘制图表</b>	<b>216</b>
7.1	使用 Ajax 和 SVG 实现实时图表	216
7.2	小结	231
<b>第 8 章</b>	<b>Ajax 数据表格</b>	<b>232</b>
8.1	使用客户端 XSLT 实现 Ajax 数据表格	233
8.2	小结	257
<b>第 9 章</b>	<b>Ajax RSS 阅读器</b>	<b>258</b>
9.1	使用 RSS	258
9.1.1	RSS 文档结构	258
9.1.2	Google Reader	259

---

9.2 实现基于 Ajax 的 RSS 阅读器 .....	260
9.3 小结 .....	275
<b>第 10 章 Ajax 的拖放功能 .....</b>	<b>276</b>
10.1 在 Web 上使用拖放功能 .....	276
10.1.1 购物车 .....	276
10.1.2 分类列表 .....	277
10.2 创建 Ajax 拖放分类列表应用 .....	277
10.3 小结 .....	296
<b>附录 A 环境配置 .....</b>	<b>297</b>
A.1 Windows 下的环境配置 .....	298
A.1.1 安装 Apache .....	298
A.1.2 安装 MySQL .....	300
A.1.3 安装 PHP .....	300
A.2 *nix 下的环境配置 .....	302
A.2.1 安装 Apache .....	302
A.2.2 安装 MySQL .....	302
A.2.3 安装 PHP .....	303
A.3 安装 phpMyAdmin .....	304
A.4 配置 Ajax 数据库 .....	305





## Ajax 与未来的 Web 应用程序

我的小堂弟第一次见到计算机时说“计算机，给我画一个机器人！”（我解释说是因为我教导计算机不要听陌生人的话，所以它才不接受这个命令）。如果您是，您的第一反映也许是“多愚蠢”或“多好笑”，但这样的想法是错误的。我们所接受的教育和大脑的思维模式已经使我们在很大程度上学会了如何使用计算机。人们要接受教育来适应计算机，弥补计算机无法像人一样思考的缺陷（另一方面，人类并不能很好地了解自身，但这是另一码事）。

这个小故事与人类本能的使用计算机的方式有关。在一个理想世界里，计算机应该能够按照我堂弟的口头命令完成任务。在过去的几年里，实现用户友好的技术水平已经得到了快速的发展，但是，距离真正意义上的智能计算机的出现还有相当长的路要走。在此之前，人们需要学习的是如何让计算机来为自己工作，在某种程度上结束向 DOS 窗口输入命令的习惯。

许多计算机的工作方式是通过软件来完成的，它通过用户界面，允许人们凭直觉（或喜好）来工作，这种现象并不是偶然的。这就像是单击鼠标右键的功能，如拖放功能，或通过简单的文本框就可以在 0.1 秒的时间里在整个 Internet 上搜索相关的内容等奇妙功能。软件工业就是这样（或者说其中有利可图的部分），它是一个发现、分析和学习的过程。现今市场上到处都是这样昂贵的程序：有着鲜明的按钮、图标、窗口以及向导。

通过软件工业我们发现，就像是一辆红色运动跑车需要有一个强大的引擎一样，软件需要具有可用性和易用性。当以商业的眼光和人性的眼光来看都觉得很好的时候，那将是一件非常美好的事情，因为商业利润是或多或少地与客户的满意度成比例的。

我们计划在本书中采用非常简单实用的方式进行讲解，但在进行您最喜欢的事情（编写代码）之前有必要退回小小的一步，这就是记住我们在做什么，以及我们为什么要这么做。我们总是喜欢按部就班地使用一项技术，却容易忘记技术存在的真正目的是为了人们的生活更加愉悦，工作更加高效。

理解人类大脑的工作方式是建立最优应用软件的关键。我们要达到这一目标还相距甚远，而我们应做的是为终端用户提供直观的用户界面。终端用户并不真正关心他们所使用的操作系统，只要系统按要求正常工作。这对程序员来说，是非常重要的而且必须牢记的一点，许多程序员即使在与终端用户打交道的时候也习惯于使用技术的观点来思考和交谈（尽管在

一个典型的开发团队中程序员不直接和终端用户打交道)。如果不同意,请回想一下在和非技术人员交谈的过程中您使用过多少次数据库这个词。

在使用计算机系统的过程中通过观察人们的需求和习惯,产生了软件可用性 (software usability) 这个术语——它是一种艺术,要迎合用户对界面的期望,要理解用户工作的规律,从而开发软件应用。

以前,可用性技术主要用于桌面应用程序 (desktop applications) 中,这只是因为当时没有 Web 应用程序 (Web applications) 工具。然而,随着 Internet 技术的逐渐成熟,这项技术得到了越来越有力的支持。

现代因特网技术不仅支持搭建更好的在线系统,同时还支持搭建更好的内部或专项应用。拥有界面友好的网站对于在线业务来说至关重要,因为因特网从不停息,用户经常会迅速地迁移到另一个更大,看起来或感觉更好的网站上。同时,建立友好的 Web 用户界面为企业软件提供了另一种选择,因为以前企业主要使用的是桌面应用。

通常在建立友好的用户界面方面,桌面应用要比 Web 应用更简单,因为在 Web 最初的设计上,它主要是用于传送文字和图片,而没有其他复杂的功能。在过去的几年里这个问题明显变得更加重要,因为越来越多的软件服务和功能通过 Web 来提供。

同时,为了增强页面的动态性,开发了许多其他技术(还有一些技术正在开发中),这使得 Web 应用的功能得到了增强。比较著名的例子有 Java applets 和 Macromedia Flash,它们都需要用户为浏览器安装它们各自的组件。

### 1.1 通过 Web 发布的功能

Web 应用程序是指通过 Web 服务器来完成应用程序的功能,并将运行的结果通过网络(如因特网或企业内部网)传递给终端用户。终端用户使用瘦客户 (thin client, Web 浏览器) 来运行 Web 应用程序,客户端知道如何显示和执行从服务器端接收到的数据。而桌面应用程序的大多数的数据处理功能是以厚客户 (thick client, 也称作富客户或胖客户) 为基础来实现的。

Web 应用程序的梦想就是希望有一天它能在用户界面和运行效率方面都像它成熟(而且功能强大)的亲戚——桌面应用程序一样强大。任何一个人们使用的计算机软件都要比它以前更重要,因为现今的计算机用户比过去的更加多样化,过去的用户主要是技术用户。现在却需要将一份表示清楚的报告交给销售部经理 (Cindy), 而且还需要为销售员 (Dave) 提供易于使用的数据输入表单。

编写的应用软件要让每一个使用它的用户都感到满意,因为终端用户满意至关重要。就出现 Web 应用程序而言,当应用的界面和操作不暴露出它的功能是通过本地桌面还是光纤或远程实现时,就完成了它进化到成熟的过程。通过 Web 传递可用的界面曾遭到过质疑,仅仅因为过去人们无法在桌面应用中完成拖放和在一个窗口下同时执行多任务这样的功能。

建立 Web 应用程序的另一个问题就是标准化。现在，每一个 Web 访问都必须通过至少两种或三种浏览器的测试，以确保所有的访问者都能充分利用网站。

## Web 应用程序的优势

诚然，试图利用 Web 来传递功能的时候会遇到很多头疼的事情。那为何还要不厌其烦地选择 Web 而不直接使用桌面应用程序呢？尽管 Web 应用程序目前还有一些问题，但它具有用户友好性，它们得到了相当大程度的普及，因为它们与桌面应用程序相比，有着大量的重要技术方面的优势。

- **Web 应用程序易于传递而且发布便宜。**使用 Web 应用程序，公司可以减少 IT 部门客户端软件安装成本。对于 Web 应用程序来说，所有用户所需要的只是一台接入因特网或企业内部网的计算机和一个能正常工作的 Web 浏览器。

- **Web 应用程序易于更新而且便宜。**软件维护成本通常是很惊人的。因为更新现有的软件和安装新软件的成本很接近，前面提到的 Web 应用程序的优点在这里同样适用。只要服务器端程序得到了更新，客户就能使用新版本了。

- **Web 应用程序针对终端用户的需求变化灵活。**只需要将 Web 应用程序安装在服务器上——任何现代操作系统均可——就可以立即使用 Mac、Windows 或 Linux 系统通过因特网/企业内部网进行访问。只要应用程序安装恰当，任何一个现代的 Web 浏览器，如 Internet Explorer、Mozilla FireFox、Opra 或 Safari 上看到的效果都是一样的。

- **Web 应用程序易于实现数据集中存储。**当多个地方需要访问相同的数据时，将所有的数据存放在一个地方要比数据分散存储容易得多。这样可以避免潜在的数据异步操作，降低安全风险。

本书将进一步研究如何使用现代 Web 技术来建立更好的 Web 应用程序，并尽可能多地使用 Web 提供的功能。在介绍详细内容之前，让我们简单地回顾一下历史。

## 1.2 20 世纪 90 年代开始出现网站

虽然 Internet 出现的历史要比这个时间稍微长一些，但超文本传输协议（HTTP）出现在 1991 年，当时只用于在 Internet 上传送数据。在它早期的几个版本里，其功能只是打开和关闭链接。HTTP 后期的版本（1996 年出现的 1.0 和 1999 年出现的 1.1）成了现在读者所熟悉和使用的协议。

### 1.2.1 HTTP 与 HTML

所有的 Web 浏览器都支持 HTTP，HTTP 出色地完成了当初的构想——解析简单的 Web 内容。无论何时通过读者喜欢的 Web 浏览器发送网页请求，都得使用 HTTP。比如，在 FireFox

的地址栏输入 `www.moaila.org`，默认情况下会把它解析为 `http://www.mozilla.org`。

Internet 使用的标准文档类型是超文本标记语言 (HTML)，它使用标记使得浏览器能够理解、解析和显示。HTML 是一种描述文档格式和内容的语言，它主要由静态文本和图片组成。HTML 并不是为建立具有交互性内容或用户界面友好的复杂的 Web 应用程序而设计的。当通过 HTTP 请求获取另一个 HTML 页面时，需要对整个网页重新加载，而且被请求的服务器上必须有该页面。显而易见，由于有这些因素的限制，很难建立起一个灵活有趣的网站。

然而，HTTP 和 HTML 都是相当成功的，它们能够同时被 Web 服务器和客户端 (浏览器) 识别。它们是迄今为止的 Internet 的基础。图 1.1 显示了用户从因特网通过 HTTP 发送请求并得到响应的简单传输过程。

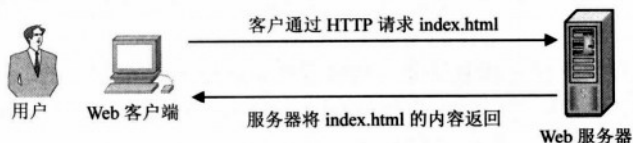


图 1.1 一个简单的 HTTP 请求

### ⚠ 注意

这里需要记住 3 点：

(1) HTTP 传输总是发生在 Web 客户端 (做出请求的软件，如 Web 浏览器) 和 Web 服务器 (做出应答的软件，如 Apache 或 IIS) 之间。本书从现在起，当说到“客户端”时指 Web 客户端，当说到“服务器”时指 Web 服务器。

(2) 用户是指使用客户端的人。

(3) 诚然，HTTP (以及它的安全版，HTTPS) 是 Internet 中使用的最重要的协议，但它并不是惟一的。各种 Web 服务器使用不同的协议来完成多种任务，通常与简单的 Web 浏览无关。在本书中使用最多的是 HTTP，当提到“Web 请求”的时候默认使用的是 HTTP，除非特别指明了使用的是其他协议。

非常肯定的一点是，HTTP 与 HTML 结合能完成的工作是非常有限的——它仅能在因特网上为用户提供静态内容 (HTML 网页)。为了弥补这方面的不足，开发了许多其他的技术。

虽然从现在起讲到的 Web 请求仍然使用 HTTP 来传输数据，但数据可以动态地通过 Web 服务器得到 (也就是通过使用数据库)，这些数据不再是只允许客户端简单显示的普通 HTML 页面，它们允许客户完成一些功能。

这项使得 Web 操作非常灵活的技术可以归为以下两类：

- 客户端技术，使得 Web 客户端除了能够显示静态文档外，还能完成更灵活的操作。这些技术是 HTML 的扩展，而没有完全取代它。
- 服务器端技术，通过服务器存储逻辑来建立动态 Web 页面。

## 1.2.2 PHP 及其他服务器端技术

服务器端 Web 技术使得 Web 服务器不再局限于仅仅返回请求的 HTML 文件，它能进行复杂的计算，进行面向对象的编程，与数据库共同工作，等等。

想象一下亚马逊公司 (Amazon) 需要为每个访客进行个性化用户产品推荐时所处理的数据量，或者 Google 在海量的数据中搜索用户请求的数据量。的确，服务器端处理正是引起 Web 技术革命的动力，也是如今的因特网如此有用的原因。

需要记住的一点就是，无论服务器端发生了什么，客户端收到的响应显然都应当是客户端能够理解的语言，如 HTML，尽管前面提到过它有很多限制。

PHP 是用于实现服务器端逻辑的技术。在第 3 章中将要介绍 PHP，在本书建立 Ajax 的教学例子中将要用到 PHP。同样应该知道，PHP 的竞争者很多，比如 ASP.NET (Active Server Pages, Microsoft 开发的 Web 技术)、Java Server Pages (JSP)、Perl、ColdFusion、Ruby on Rails，等等。它们都允许程序员建立服务器端功能。

PHP 不仅是一项服务器端技术，它还支持脚本语言，即编程人员可以创建 PHP 脚本。图 1.2 展示了向 `index.php` 发送请求时，服务器端并不是返回 `index.php` 的内容，而是执行 `index.php` 后将结果返回给客户端，这些结果必须以 HTML 或其他能被客户端理解的格式表示。

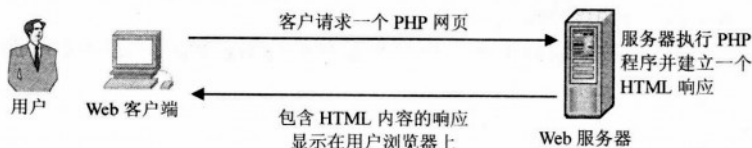


图 1.2 客户端请求一个 PHP 页面

通常还需要在服务器端配置一个数据库服务器来管理数据。本书实例使用的是 MySQL 数据库，当然，使用其他数据库时，工作原理也是一样的。读者将在第 3 章中学习如何在 PHP 中调用数据库。

尽管 PHP 能够通过数据库根据不同的用户需求做出不同的响应，但是浏览器上显示的依然是一个静态的、令人厌倦的、不够灵活的网页。

更加灵活、功能更加强大的客户端需求，促使了一些独立的新技术的产生，也就是客户端技术。现在来看看目前浏览器是如何解析 HTML 之外的其他语言的。

## 1.2.3 JavaScript 及其他客户端技术

客户端技术彼此间有很多差异，我们从 Web 客户端载入和执行谈起。JavaScript 是一种脚本语言，它可以将代码写在简单的文本中并嵌入到 HTML 页面来增强页面功能。当客户端请求一个 HTML 页面时，HTML 页面中可以包含 JavaScript 脚本。所有现代的浏览器都支持

JavaScript 脚本语言，而不需要在系统中安装新的组件。

JavaScript 是一种凭自身能力而存在的语言（理论上说它并不依赖于 Web 开发技术），大多数的 Web 客户端在各种平台下都支持 JavaScript，并且 JavaScript 还有一些面向对象的功能。JavaScript 并不是一种编译语言，所以它不适合做高精度的计算或者用于编写设备驱动，而且它还必须在装载到客户端后才被解释执行，所以它不够安全可靠，但将 JavaScript 用于 Web 页面中却是相当有效的。

开发人员采用 JavaScript 技术可以建立更完善的网页。例如用户忘记提交所有信息（如口令、账号等）或 E-mail 地址格式填写不正确时，客户端表单验证功能可以避免重载全部页面（瞬时丢失填写的全部数据）。尽管 JavaScript 有这些潜力，但它从来就没有在真正意义上实现像桌面应用程序那样的用户友好性。

其他流行的客户端技术还有 Java applets 和 Macromedia Flash。Java applets 采用流行而强大的 Java 语言编写，并通过 Java 虚拟机来执行，这就需要在客户端安装 Java 虚拟机。Java applets 能够实现较为复杂的系统，但是 Java applets 在 Web 应用中的流行程度已大不如以前，因为它需要消耗大量的系统资源，有时需要很长的启动时间，而且对于需求不多的简单 Web 应用较小的需求来说它显得过于笨重而强大。

Macromedia Flash 对创建动画和图像效果来说是一个非常强大的工具，事实上它还是通过 Web 来传送这种应用的标准。Flash 同样需要在客户端安装浏览器插件。基于 Flash 的技术越来越强大，新的技术层出不穷。

当 HTML 技术同服务器端技术和客户端技术相结合时，就能够建立非常强大的 Web 应用程序。

## 1.2.4 缺少的是什么

已经有了这么多的选择，为什么还会有人期待新的技术呢？现在还缺少什么呢？

正如本章开始所指出的那样，技术为市场需求服务。部分市场希望建立功能更加强大的 Web 客户端而不使用 Flash 和 Java applets 或其他一些过分浮华或代价过高的技术。由于这些原因，开发者们经常使用 HTML、JavaScript 和 PHP（或其他服务器端技术）建立网站和 Web 应用程序。这种情况下典型的请求方式如图 1.3 所示，它显示了在向 PHP 服务器发送 HTTP 请求，收到执行 PHP 程序生成 HTML 和 JavaScript 组成的应答的过程。

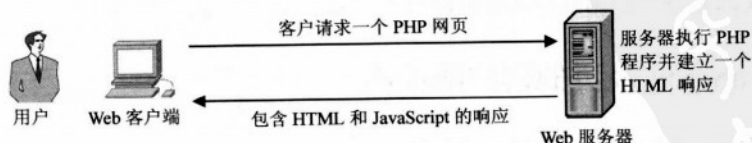


图 1.3 HTTP、HTML、PHP 和 JavaScript 工作流程

这个过程隐藏的问题是客户端需要服务器端数据时，每发送一个新的 HTTP 请求都必须

重载网页，而这期间用户不能做任何操作。网页重载是当今 Web 技术新的克星，此时 Ajax 以拯救者的身份出现了。

## 1.3 理解 Ajax

Ajax 是 Asynchronous JavaScript and XML 的缩写。相信读者对它的含义还不够了解。简单地说，Ajax 可以理解为“增强的 JavaScript”，实质上它提供了一种可以调用后台服务器获得数据的客户端 JavaScript 技术，它支持更新部分网页内容时不重载整个网页。访客请求一个典型的支持 Ajax 的网页时，所发生的一切如图 1.4 所示。

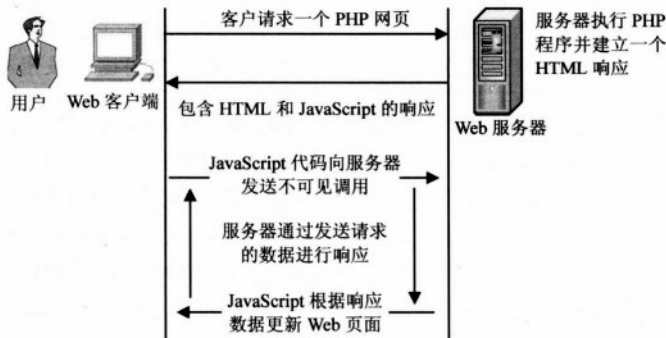


图 1.4 一个典型的 Ajax 调用

通过观察可以发现，Ajax 在执行用户请求时，努力平衡客户端功能和服务器端功能。目前为止，客户端技术和服务器端技术一直被认为是相对独立的技术，各个部分在响应用户请求时分别起作用。Ajax 的出现解决了页面装载时客户和服务器的平衡问题，在用户操作网页的同时允许客户和服务器在后台通信。

举一个简单的例子，假设有这样一个表单，要求用户填写一些信息（如姓名、E-mail 地址、密码和账号等），需要在到达业务终端之前进行验证。没有 Ajax 时有两种技术来验证数据。一种方法是让用户输入数据，并把这些数据传送出去，在服务器端验证数据。这种情况下用户需要经历一段停止工作等待载入新的页面的时间。另一种方法是在客户端验证数据，但这种方式在客户端载入了大量的数据（比如需要验证输入的城市是否与所输入的国家匹配）时就不太可行（适用）。

在支持 Ajax 的情形下，Web 应用程序可以通过调用后台服务器来验证数据，同时用户可以继续填写内容。比如，在用户选择一个国家后，浏览器调用服务器迅速显示这个国家的城市列表，而不会打断用户当前的操作。在第 4 章中将讲到这个数据验证的例子。

Ajax 带来的变化是无穷的。为了使读者在感官上对 Ajax 有更深刻的理解，请看下面一些在当今很流行的例子。

• **Google Suggest** 帮助用户完成 **Google** 搜索。它的功能是惊人的，读者可以通过 <http://www.google.com/webhp?complete=1> 来访问它。**Yahoo Instant Search** 也提供了相同的功能，网址是 <http://instant.search.yahoo.com/>（读者将在第 6 章中学习如何完成相同的功能）。

• **Gmail** (<http://www.gmail.com>)。现在 GMail 已经相当流行，这里就不对它做更多的介绍了。另外，以 Web 为基础的 E-mail 服务如 **Yahoo Mail** 和 **Hotmail** 也在向着提供以 Ajax 技术为功能的服务迈进。

• **Google Maps** (<http://maps.google.com>)、**Yahoo Maps** (<http://maps.yahoo.com>) 以及 **Windows Live Local** (<http://local.live.com>)。

• 其他服务，如 <http://www.writely.com> 和 <http://basecamphq.com>。

在本书中还将看到更多的例子。

**提示** 就像其他技术一样，Ajax 可能会被过度地或错误地使用。在网站中使用了 Ajax 技术并不意味着网站更加优秀了。这与是否正确使用这项技术有关。

所以，Ajax 是一种创建更加灵活、交互性更强的 Web 应用技术，它使得网页在用户工作时能够透明地异步调用服务器。Ajax 是一种 Web 开发人员可以用来创建更加灵活的 Web 应用程序的工具，而且它在与人进行交互方面比传统的 Web 应用程序更好。

Ajax 技术可以在现有的 Web 浏览器上实现，比如 Mozilla Firefox、Internet Explorer 或 Opera，所以访问一个 Ajax 网站的时候不需要安装额外的插件。Ajax 由以下几部分组成：

• **JavaScript** 是 Ajax 的核心组成部分，使用它来实现客户端的功能。在 JavaScript 函数中，将大量地使用文档对象模型（DOM）来管理部分 HTML 页面。

• **XMLHttpRequest** 对象使得 JavaScript 能够异步地访问服务器，使得在后台执行的同时用户可以继续工作。访问服务器可以简单理解为向服务器上的文件或者脚本发送 HTTP 请求。HTTP 请求易于建立而且不会引起与防火墙相关的问题。

• 服务器端技术，要求能够获取来自 JavaScript 客户端的请求。在本书中使用 PHP 来完成服务器端的功能。

客户-服务器通信时双方需要一种传送数据并理解数据的方法。传送数据部分比较简单。客户脚本（使用 XMLHttpRequest 对象）可以通过使用 GET 或 POST 方法传送名-值对给服务器。任何服务器脚本读取这些值都是非常容易的。

服务器脚本简单地通过 HTTP 做出响应，和普通网站不同的是，这里的响应使用一种能被客户端的 JavaScript 代码轻松识别出来的格式。建议使用 XML 格式，因为它具有支持面广的优点，而且许多函数库操作 XML 文档比较容易。当然读者也可以选择其他喜欢的格式（甚至可以传送普通文档），JavaScript 对象标记 JavaScript Object Notation（JSON）是除 XML 之外另一种流行格式。

本书假定读者对 Ajax 的组成部分有所了解，当然 XMLHttpRequest 对象除外，相对来说它不太流行。但读者必须确保在第 2 章和第 3 章中，针对同一个页面，能弄明白各个部分是如何工作的。到现在为止，本章余下的工作是把重点放在一个更大的构想里，并会写一个 Ajax



程序来满足许多已经没有耐心的读者。

### ⚠ 注意

Ajax 的任何一个组成部分都不是新提出来的,也不是像当前炒作的那样 Ajax 是一场革命(或至少是一个进化): Ajax 的各部分在 1998 年之前就已经存在。Ajax 这一名词首次出现在 2005 年 Jesse James Garret 的 <http://www.adaptivepath.com/publications/essays/archives/000385.php> 文章中,它随着在 Google 中的多项应用逐渐流行起来。

Ajax 的创新在于它是市场上第一个强烈支持标准化而且集中精力引导这场变革的技术。结果许多 Ajax 的库函数正在开发中,许多使用 Ajax 技术的网站涌现出来。微软公司也在它的 Atlas 项目中使用了 Ajax 开发技术。

在建立 Web 应用程序时使用 Ajax 将给用户带来如下潜在的好处:

- 它可以实现一个更加优秀、响应更加迅速的网站或 Web 应用程序;
- 它的流行将促进建立统一的开发模式,开发者们不需要为完成一个基本的功能建立新的模式;

- 它使用现有的开发技术;
- 它使用现有的开发技巧;
- Ajax 完美地集成了 Web 浏览器提供的功能(如网页大小的重定义,页面导航等)。

Ajax 能够成功使用的场景有:

- 需要服务器端对表单做出快速验证的地方,尤其适用于无法在网页的第一次装载中就将所有需要的验证信息全部加载到客户端的时候。第 4 章有一个这样的验证表单的例子。

- 实现简单的在线聊天而不需要其他如 Java Runtime Machine 或 Flash 之类的库。读者将在第 5 章中建立一个这样的程序。

- 完成类似 Google Suggest 的功能,第 6 章有一个这样的例子。

- 使用其他已有技术使得功能更加强大。在第 7 章,将使用可缩放矢量图形(Scalable Vector Graphics, SVG)来完成实时绘图功能;在第 10 章,可以使用扩展的 Ajax 库来创建一个简单的拖放列表。

- 编写迅速响应的数据表格(data grids)代码,能快速更新服务器的数据库。相关应用的例子在第 8 章。

- 创建需要根据不同外部信息进行实时更新的应用程序。在第 9 章,本书将建立一个简单的 RSS 聚集器。

Ajax 的潜在问题:

- 网址在整个工作过程中是不变的,因此很难将支持 Ajax 的网页加入到书签中。在 Ajax 的应用中,书签因不同的应用而有不同的含义,通常代表的是需要保存状态(想想桌面应用程序——也没有书签的概念)。

- 搜索引擎可能无法为 Ajax 应用站点的每个部分都建立索引。

• 浏览器的后退按钮不能完成传统 Web 应用程序的功能，因为所有的操作在同一个页面完成。

• 如果客户端禁用 JavaScript，那么所有的 Ajax 应用程序将无法工作，所以应在站点中做出另一种备选方案，无论如何，都不能失去访客。

最后，在进入编写第一个 Ajax 程序之前，以下一些链接对读者进入令人振奋的 Ajax 世界也许会有所帮助。

• <http://ajaxblog.com>，一个专门讲解 Ajax 的博客。

• <http://www.fiftyfoureleven.com/resources/programming/xmlhttprequest>，一篇全面讲解 Ajax 技术的文章。

• <http://www.ajaxian.com>，是 *Pragmatic Ajax* 的作者 Ben Galbraith 和 Dion Almaer 的 Ajax 站点。

• <http://www.ajaxmatters.com>，是关于 Ajax 信息的站点，包含很多有用的链接。

• <http://ajaxpatterns.org>，关于可重用的 Ajax 设计模式。

• <http://www.ajaxinfo.com>，包含 Ajax 文章和相关链接资源。

• <http://dev.fiaminga.com>，包含各种 Ajax 资源和教程。

• <http://ajaxguru.blogspot.com>，一个很受欢迎的 Ajax 相关的 Web 博客。

• <http://www.sitepoint.com/article/remote-scripting-ajax>，Cameron Adams 写得很好的关于 Ajax 文章：Usable Interactivity with Remote Scripting。

• <http://developer.mozilla.org/en/docs/Ajax>，Mozilla's 关于 Ajax 的网页。

• <http://en.wikipedia.org/wiki/Ajax>，Wikipedia 关于 Ajax 的网页。

上述列表并不完全，想获得更多在线资源，可以访问 Google。后面章节中，我们会有针对性地提到更多资源。

## 1.4 使用 Ajax 和 PHP 建立一个简单应用程序

现在我们来编写一些代码！本章将全面介绍如何建立一个简单的 Ajax 应用程序。

### ⚠ 注意

这个练习是为了满足大多数希望尽快开始编写代码的读者，但这要求读者对 JavaScript、PHP 和 XML 比较熟悉。如果对上面的内容还不熟，或者觉得有一定的难度，那么请跳过这里进入第 2 章。在第 2 章和第 3 章里本书将对 Ajax 相关技术作详细介绍，这样编写代码就容易多了。

这里将建立一个简单的 Ajax Web 应用程序，这个程序叫 quickstart，要求用户填写姓名，服务器端在用户输入的同时向客户端发回响应。图 1.5 显示了用户进入时的初始页面 index.html（注意，在进入 quickstart 文件夹时即使没有具体地指出网页的名字，也将自动调用 index.html）。

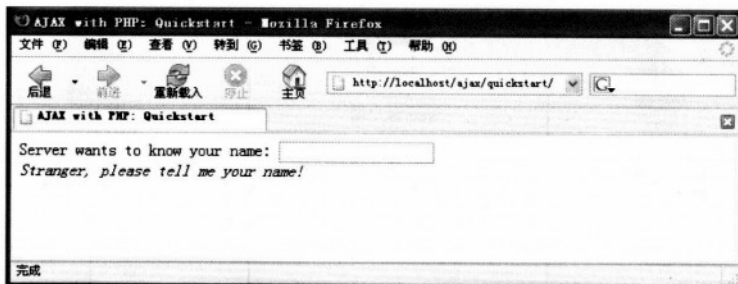


图 1.5 QuickStart 应用程序的首页

在用户输入的同时，服务器端将以一定的时间间隔被异步调用，来查看系统是否认识当前的用户。服务器端以大约每秒一次的频率被自动调用，这样就不需要一个按钮（比如“提交”按钮）将输入的内容发送给服务器（这种方法在实际的用户登录中也许不恰当，但是它是一个很好的演示 Ajax 功能的例子）。

用户输入的名字不同，服务器端返回的信息就会不一样，如图 1.6 所示。

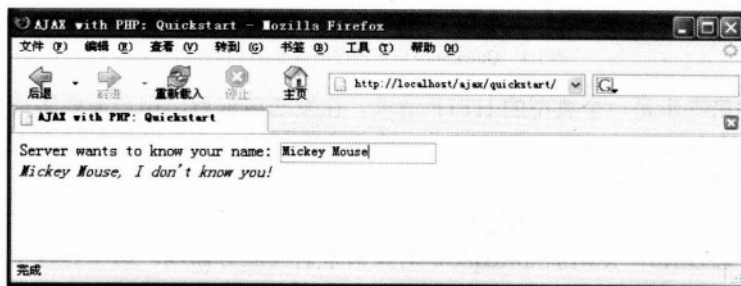


图 1.6 用户从 Web 应用程序收到实时响应

大家可以通过 <http://ajaxphp.packtpub.com/ajax/quickstart> 在线测试这个例子。

也许一开始会觉得这里没有什么特别的地方。使用这个例子的首要目的，是为了易于理解。这个例子的特别之处在于消息自动从服务器端返回，而不需要用户等待（消息在用户输入用户名的时候显示出来）。虽然需要调用服务器来获得新的数据，但这个网页不需要重载就可以显示新的数据。这对于非 Ajax 的 Web 开发技术来说，要实现这一功能并不简单。

这个应用程序使用了以下 3 个文件：

- (1) `index.html` 是用户最初的请求文件。
- (2) `quickstart.js` 是 JavaScript 脚本文件，它在客户端载入 `index.html` 的时候被一同载入。这个文件负责在需要服务器端的功能时向服务器端发送异步请求。

(3) `quickstart.php` 是服务器端的 PHP 脚本文件，它被客户端的 `quickstart.js` 的 JavaScript 代码调用。

图 1.7 显示了运行这个应用程序时所发生的操作。

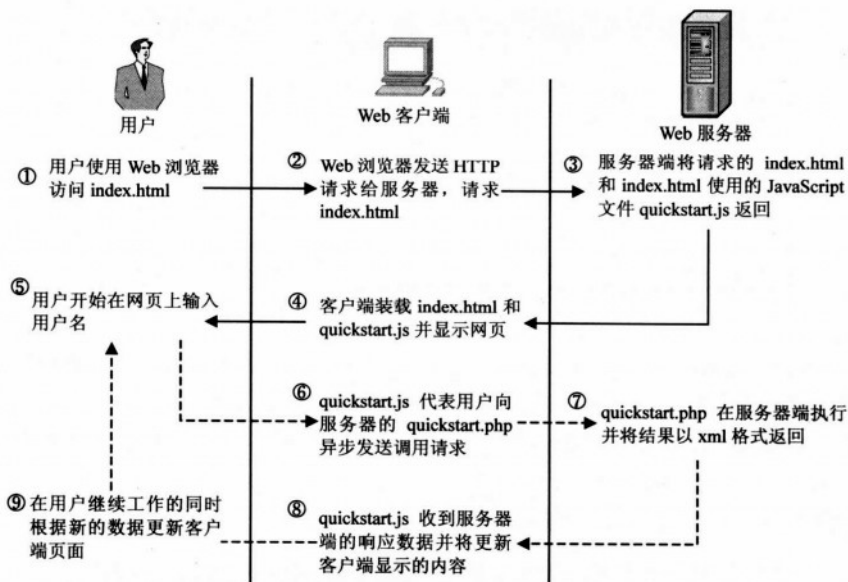


图 1.7 QuickStart 应用程序的内部工作流程图

第①步到第⑤步是一个典型的 HTTP 请求。在发送请求后, 用户需要等待页面的载入。在其他典型 (非 Ajax) 的 Web 应用程序中, 当客户端需要向服务器发送请求来获得新的数据时, 用户需要等待页面重载。

从第⑤步到第⑨步显示的是 Ajax 形式的调用——更确切地说, 是一个发送异步 HTTP 请求的过程, 通过使用 XMLHttpRequest 对象请求访问服务器端。这个过程中, 用户可以正常地继续使用网页, 就像使用一个普通的桌面应用程序一样。在从服务器端接收数据和更新数据的过程中, 没有任何页面的重载和更新。

现在就在机器上实现这些代码吧! 在开始实现代码之前, 要求按附录 A 安装 PHP 和 Apache 并为本书的例子建立一个数据库, 完成工作环境配置 (在 quickstart 这个例子中不需要使用数据库。)

### ⚠ 注意

本书的所有例子都假定读者已经在计算机中配置了与附录 A 相同的工作环境。如果设置的环境不一样, 就需要在运行时作一些改变, 如使用不同的文件夹等。

## 》》》 实现步骤——Quickstart Ajax

(1) 附录 A 教读者如何在计算机中安装一个 Web 服务器和创建一个 ajax 的 Web 目录, 把这个目录作为本书中所有实例的主目录。在 ajax 目录下, 创建一个 quickstart 文件夹。

(2) 在 quickstart 文件夹下，创建 index.html 文件，并将下面的代码写入文件中。

```
<!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>
    <title>Ajax with PHP: Quickstart</title>
    <script type="text/javascript" src="quickstart.js"></script>
  </head>
  <body onload='process() '>
    Server wants to know your name:
    <input type="text" id="myName" />
    <div id="divMessage" />
  </body>
</html>
```

(3) 创建 quickstart.js，加入如下代码：

```
//定义 XMLHttpRequest 对象
var xmlhttp = createXmlHttpRequestObject();

//获取 XMLHttpRequest 对象
function createXmlHttpRequestObject()
{
  //用来存储将要使用的 XMLHttpRequest 对象
  var xmlhttp;
  //如果在 Internet Explorer 下运行
  if(window.ActiveXObject)
  {
    try
    {
      xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
    }
    catch (e)
    {
      xmlhttp = false;
    }
  }
  //如果在 Mozilla 或其他浏览器下运行
  else
  {
    try
```

```
{
    xmlHttp = new XMLHttpRequest();
}
catch (e)
{
    xmlHttp = false;
}
}
//返回创建的对象或显示错误信息
if (!xmlHttp)
{
    alert("Error creating the XMLHttpRequest object.");
}
else
{
    return xmlHttp;
}

//使用 XMLHttpRequest 对象创建异步 HTTP 请求
function process()
{
    //在 xmlHttp 对象不忙时进行处理
    if (xmlHttp.readyState == 4 || xmlHttp.readyState == 0)
    {
        //获取用户在表单中输入的姓名
        name = encodeURIComponent(document.getElementById("myName").value);
        //在服务器端执行 quickstart.php
        xmlHttp.open("GET", "quickstart.php?name=" + name, true);
        //定义获取服务器端响应的方法
        xmlHttp.onreadystatechange = handleServerResponse;
        //向服务器发送请求
        xmlHttp.send(null);
    }
    else
    {
        //如果服务器忙, 1 秒后重试
        setTimeout('process()', 1000);
    }
}

//当收到服务器端的消息时自动执行
function handleServerResponse()
{
    //在处理结束时进入下一步
```

```

if (xmlHttp.readyState == 4)
{
    //状态为 200 表示处理成功结束
    if (xmlHttp.status == 200)
    {
        //获取服务器端发来的 XML 消息
        xmlResponse = xmlHttp.responseXML;
        //获取 XML 中的文档对象 (根对象)
        xmlDocumentElement = xmlResponse.documentElement;
        //获取第一个文档子元素的文本消息
        helloMessage = xmlDocumentElement.firstChild.data;
        //使用从服务器端发来的消息更新客户端显示的内容
        document.getElementById("divMessage").innerHTML =
            '<i>' + helloMessage + '</i>';

        //重新开始
        setTimeout('process()', 1000);
    }
    //如果 HTTP 的状态不是 200 表示发生错误
    else
    {
        alert("There was a problem accessing the server: " + xmlHttp.statusText);
    }
}
}

```

(4) 创建 quickstart.php, 加入如下代码:

```

<?php
//我们将创建一个 XML 格式的输出
header('Content-Type: text/xml');
//创建 XML 头
echo '<?xml version="1.0" encoding="UTF-8" standalone="yes"?>';
//创建<response>元素
echo '<response>';

//获取用户姓名
$name = $_GET['name'];
//根据从客户端获取的用户名创建输出
$userNames = array('CRISTIAN', 'BOGDAN', 'FILIP', 'MIHAI', 'YODA');
if (in_array(strtoupper($name), $userNames))
    echo 'Hello, master ' . htmlentities($name) . '!';

```

```
else if (trim($name) == '')
    echo 'Stranger, please tell me your name!';
else
    echo htmlentities($name) . ', I don\'t know you!';
//关闭 <response> 元素
echo '</response>';
?>
```

(5) 现在，在浏览器地址栏输入 `http://localhost/ajax/quickstart/`，就可以访问到这个程序了。进入页面后，读者将会看到如图 1.5 和图 1.6 所示的页面。

### ⚠ 注意

如果在程序运行中出现错误，检查环境是否按照附录 A 的步骤正确安装和配置。大多数时候都是因为一个小小的问题（如书写错误）而发生错误。在第 2 章和第 3 章将学习如何处理 JavaScript 和 PHP 代码出现的错误。

## >>> 程序说明

这是件有趣事——理解在代码中发生了什么（记住，本书将在接下来的两章中做更为详细的讲解）。

从交互的第一个页面 `index.html` 开始，这个文件调用了神秘的 JavaScript 文件 `quickstart.js`，为客户端建立了一个非常简单的 Web 界面。下面是从 `index.html` 中摘录的部分代码，注意黑体的部分：

```
<body onload='process() '>
    Server wants to know your name:
    <input type="text" id="myName" />
<div id="divMessage" />
</body>
```

在页面载入时，调用 `quickstart.js` 中的 `process()` 方法。这引起了 `<div>` 元素载入从服务器端获得的消息。

在查看 `process()` 方法发生了什么之前，先来看看服务器端发生了什么。在 Web 服务器端有一个 `quickstart.php` 脚本，是它构造了返回给客户端的 XML 格式的消息。这个 XML 消息包含一个 `<response>` 元素，它包含了需要服务器端返回给客户端的消息。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<response>
... 服务器端希望传送给客户端的消息...
</response>
```



如果客户端接收到的用户名 (username) 为空, 消息将是 “Stranger, please tell me your name! ”。如果用户名 (username) 是 Crisitan、Bogdan、Filip、Mihai 或 Yoda 之一, 服务器将返回 “Hello, master<用户名>!” 如果输入的是其他名字, 将返回 “<用户名>, I don't know you!” 所以, 如果 Mickey Mouse 输入它的名字, 服务器端将返回具有如下 XML 格式的消息:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<response>
Mickey Mouse, I don't know you!
</response>
```

Quickstart.php 脚本首先创建一个 XML 文档头, 然后开始建立 <response> 元素:

```
<?php
//我们将创建一个 XML 格式的输出
header('Content-Type: text/xml');
//创建 XML 头
echo '<?xml version="1.0" encoding="UTF-8" standalone="yes"?>';
//创建<response>元素
echo '<response>';
```

黑体字的那行标记了输出的是 XML 文档, 这一点很重要, 因为客户端希望接收到的是 XML (如果消息头不含 Content-Type: text/xml 的内容, 客户端用于解析 XML 的 API 将会抛出一个错误)。设置了头部以后, 通过连接字符串生成 XML 响应。事实上返回给客户端的内容包含在 <response> 元素里, <response> 是根元素, 它是基于 GET 参数从客户端获得的名字生成的:

```
//获取用户姓名
$name = $_GET['name'];
//根据从客户端获取的用户名创建输出
$userNames = array('CRISTIAN', 'BOGDAN', 'FILIP', 'MIHAI', 'YODA');
if (in_array(strtoupper($name), $userNames))
    echo 'Hello, master ' . htmlentities($name) . '!';
else if (trim($name) == '')
    echo 'Stranger, please tell me your name!';
else
    echo htmlentities($name) . ', I don\'t know you!';
//关闭 <response> 元素
echo '</response>';
?>
```

用户输入的内容 (这里认为是用户名) 通过 GET 参数从客户端传送给服务器。当将文本内容传送回客户端时, 使用 PHP 方法 htmlentities 来替换 HTML 代码中的一些字符 (如 &、>),

确保消息能够在客户端安全地显示，消除潜在的安全问题。

### ! 说明

编写产品代码时在服务器端生成客户端文本（而不是直接在客户端生成）是非常糟糕的事情。理想情况下，服务器端的任务是用一种通用格式来传送数据，然后由接收端处理安全和格式问题。这种方法更有意义，如果有一天您需要将精确的文本直接插入数据库中，但数据库需要的序列格式不一样时（在这种情况下，就应该由数据库处理脚本而不是由服务器端来完成格式化工作）。在 quickstart 这个例子里，在 PHP 脚本中生成 HTML 代码使得程序简短并易于理解和解释。

如果读者对 quickstart.php 的内容感兴趣，在浏览器地址中输入 `http://localhost/ajax/quickstart/quickstart.php?name=Yoda` 将会看到它生成的结果。通过客户端的 GET 来传递参数的优点是它可以在浏览器中非常简单地效仿这样的请求过程，因为使用 GET 意味着用户将参数的名/值对附加到 URL 串的后面。读者看到的结果如图 1.8 所示。

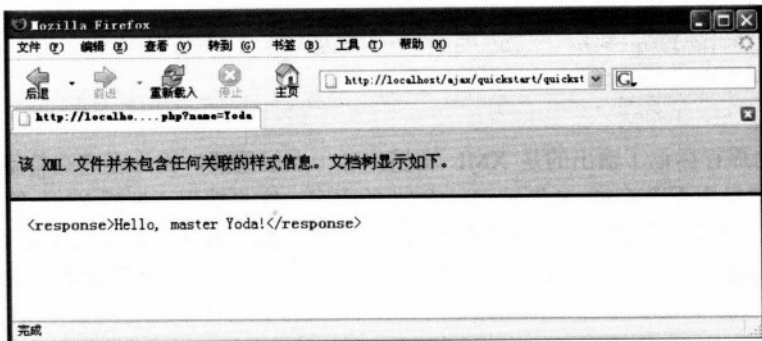


图 1.8 quickstart.php 生成的 XML 数据

通过 quickstart.js 中的 `handleServerResponse()` 在客户端读取 XML 消息。更确切地说，下面的代码获取了“Hello, master Yoda”消息：

```
//获取服务器端发来的 XML 消息
xmlResponse = xmlHttp.responseXML;
//获取 XML 中的文档对象（根对象）
xmlDocumentElement = xmlResponse.documentElement;
//获取第一个文档子元素的文本消息
helloMessage = xmlDocumentElement.firstChild.data;
```

这里，`xmlHttp` 是 `XMLHttpRequest` 对象，用来从客户端调用服务器端 `quickstart.php` 脚本。`responseXML` 提取接收到的 XML 文档中的内容。XML 具有层次结构，XML 文档的根元素叫做文档元素。在本例中，文档元素是 `<response>`，它只包含了一个简单的子元素，也就是我们所看到的显示出来的消息。一旦接收到消息，它就通过使用 DOM 访问 `divMessage`

元素并显示在客户端的 `index.html` 页面上:

```
//使用从服务器端发来的消息更新客户端显示的内容
document.getElementById("divMessage").innerHTML = helloMessage ;
```

`document` 元素是 JavaScript 对象, 它允许用户控制 HTML 代码的各个元素。

`quickstart.js` 中余下的代码负责向服务器端发送请求以获取 XML 消息。

`createXmlHttpRequestObject()` 函数用于创建和返回一个 `XMLHttpRequest()` 对象。这个方法要比需要的长度长一些, 因为我们需要它能够跨浏览器使用, 本书将在第 2 章对它进行详细介绍, 目前读者需要了解的就是它有什么作用。`XMLHttpRequest` 实例名为 `xmlHttp`, 用于在 `process()` 中向服务器端发送异步请求:

```
//使用 XMLHttpRequest 对象创建异步 HTTP 请求
function process ()
{
    //在 xmlHttp 对象不忙时进行处理
    if (xmlHttp.readyState == 4 || xmlHttp.readyState == 0)
    {
        //获取用户在表单中输入的姓名
        name = encodeURIComponent(document.getElementById("myName").value);
        //在服务器端执行 quickstart.php
        xmlHttp.open("GET", "quickstart.php?name=" + name, true);
        //定义获取服务器端响应的方法
        xmlHttp.onreadystatechange = handleServerResponse;
        //向服务器发送请求
        xmlHttp.send(null);
    }
    else
        //如果服务器忙, 1 秒后重试
        setTimeout('process()', 1000);
}
```

这里所看到的的就是 Ajax 的核心, 即向服务器端发送异步请求的代码。

为什么异步调用服务器如此重要呢? 异步请求在开始调用接收到的响应的过程中具有不冻结进程 (以及用户体验) 的特点。异步处理通过事件驱动体系结构来实现, 最好的使用这种方式的例子就是建立图形用户界面: 如果没有事件驱动, 就需要不停地检查用户是否单击了按钮或重绘了窗口。如果使用事件, 读者就可以在事件控制台进行需要的操作。使用 Ajax, 可以使得向服务器端发送请求后, 服务器端作出的响应能被自动发现。

如果使用同步的请求又会是怎样呢? 那么需要将 `xmlhttp.open` 参数置为 `false`, 然后按下面的方法手动调用 `handleServerResponse` 方法。这种情况下, 在客户端与服务器通信的时候

不能使用输入框（这种情况，等待时间的长短与连接速度有很大关系。如果服务器就是本机，这个时间几乎是感觉不到的）。

```
///使用 XMLHttpRequest 对象创建调用服务器请求
function process()
{
    //获取用户在表单中输入的姓名
    name = encodeURIComponent(document.getElementById("myName").value);
    //在服务器端执行 quickstart.php
    xmlhttp.open("GET", "quickstart.php?name=" + name, false);
    //尽力同步服务器请求（在结束前不能处理）
    xmlhttp.send(null);
    //读取响应
    handleServerResponse();
}
```

process()方法使用 XMLHttpRequest 对象初始化一个新的服务器请求。然而，这只有在 XMLHttpRequest 对象空闲的时候进行。在本例中，如果网络连接的速度非常慢，等待服务器响应的的时间可能需要 1 秒以上。所以，process()方法初始化请求时首先验证它是否为空：

```
///使用 XMLHttpRequest 对象创建异步 HTTP 请求
function process()
{
    //在 xmlhttp 对象不忙时进行处理
    if (xmlhttp.readyState == 4 || xmlhttp.readyState == 0)
    {
```

如果连接繁忙，必须使用 setTimeout 值以便如果服务器 1 秒后没有返回值时重新开始(函数的第二个参数表示等待执行的毫秒数)：

```
///如果服务器忙，1 秒后重试
setTimeout('process()', 1000);
```

如果这行为空，用户可以安全地发送新的请求。这行代码为向服务器端请求作准备，但它不和服务器端打交道：

```
///在服务器端执行 quickstart.php
xmlhttp.open("GET", "quickstart.php?name=" + name, true);
```

第一个参数指明了向服务器端传送用户名的方法，读者可以选择 GET 或 POST（详细介绍见第 3 章）。第二个参数是要访问的服务器端页面。如果第一个参数是 GET，那么名/值对就会显式地显示在 URL 中。如果希望进行异步调用，第三个参数就要设置为 true。当进行异步调用时，不需要等待响应时间。当请求的状态改变时，可以定义另一个被自动调用

的方法:

```
//定义获取服务器端响应的方法
xmlHttpRequest.onreadystatechange = handleServerResponse;
```

一旦设置了这个选项, 就可以休息了, 系统会在任何请求发出的时候自动执行 `handleServerResponse` 方法。准备好之后, 就可以开始调用 `XMLHttpRequest` 方法来发送请求:

```
//向服务器发送请求
xmlHttpRequest.send(null);
}
```

现在看看 `handleServerResponse` 方法。

```
//当收到服务器端的消息时自动执行
function handleServerResponse()
{
//在处理结束时进入下一步
if (xmlHttpRequest.readyState == 4)
{
//状态为 200 表示处理成功结束
if (xmlHttpRequest.status == 200)
{
```

`handleServerResponse` 在请求状态改变时被多次调用。只有在 `xmlHttpRequest.readyState` 状态为 4 的时候向服务器发送结束请求, 可以读取服务器发回的结果。还可以看到 HTTP 传输报告状态为 200, 这表示在 HTTP 请求中没有错误发生。这些情况下, 不用理睬服务器端的响应, 而直接将消息发送给用户。

在响应被接收和使用之后使用 `setTimeout` 方法, 使 `process()` 方法在 1 秒后执行 (注意在客户端重复执行任务并不是必须的, 或者说是 Ajax 专有的):

```
//重新开始
setTimeout('process()', 1000);
```

最后, 再重复一次在用户进入页面后都发生了什么 (图 1.7 体现了这一过程)。

- (1) 用户进入 `index.html` (与图 1.7 第①~④步对应)。
- (2) 用户开始 (或继续) 输入他或她的姓名 (与图 1.7 第⑤步对应)。
- (3) 当执行 `quickstart.js` 中的 `process()` 时, 它异步调用服务器端的 `quickstart.php`。用户输入的信息通过 URL 的形式 (使用 GET 方式) 传送给服务器。`handleServerResponse` 方法用于捕捉请求状态的变化。
- (4) `quickstart.php` 在服务器端执行。它将服务器端希望返回给客户端的消息写成 XML 的格式。

(5) 客户端的 `handleServerResponse` 方法随着请求状态的变化被多次执行。最后一次调用它的时间是在成功接收消息时。客户端读取 XML 消息，并将消息提取出来显示在网页上。

(6) 用户端显示的内容根据服务器发回的新消息而变化，但用户可以继续输入内容而且不会被打断。1 秒的延迟之后，这个过程又从图 1.7 的第②步重新开始。

## 1.5 小结

本章对 Ajax 做了一个简单的介绍。要想学习如何建立 Ajax 应用程序，理解 Ajax 的重要性和重要之处非常重要。就像其他技术一样，Ajax 并不能解决所有的问题，但是它提供了解决一些问题的方法。

Ajax 结合了客户端和服务器端的功能，增强了网站的用户体验。客户端 JavaScript 代码中的 `XMLHttpRequest` 对象是异步调用服务器的关键元素。这章较短而且很可能给读者留下了很多问题，这就对了！读者可以在本书中的后续章节中找到这些问题的答案，而且还会看到很多有趣的例子。



## JavaScript 下灵活的客户端技术

有人常说：一幅图画顶得上千言万语。这也可看作是优良代码的写照。接下来的两章包括大量知识，将为以后开发 Ajax 应用奠定基础。

希望第 1 章已经引发了读者对 Ajax 的兴趣，使读者能够坚持学习含有大量原理的第 2 章。另一方面，如果觉得第 1 章的学习有点吃力，本章将会让进度慢一些。本章将通过各种简短的例子来帮助读者学习 Ajax 理论。本章重点讲述 Ajax 客户端技术，包括：

- JavaScript;
- The JavaScript DOM;
- 层叠样式表 (Cascading Style Sheets——CSS);
- XMLHttpRequest 对象;
- 扩展标记语言 (Extensible Markup Language, XML)。

本章将学习如何使这些组件流畅地配合在一起工作，并且为将来的 Ajax 程序开发打下坚实的基础。本章还将学习如何有效地实现差错控制技术，以及如何有效地编写代码。第 3 章将介绍今后实例中会用到的服务器端的技术，如 PHP、MySQL 等。

作为一名优秀的 Ajax 开发者，必须清楚地知道各部分的工作原理，并掌握它们协调工作的机制。本书中，我们假设读者已经有了相关的技术经验。

根据读者所掌握的知识，希望能花些时间在阅读第 2 章和第 3 章之前或之后，浏览一下 <http://ajaxphp.packtpub.com>，那里提供了一些工具，可以使程序员的编程工作变得更加简单。千万不要略过这一部分，因为它很重要，高效率地使用正确的工具有很多好处。在 <http://ajaxphp.packtpub.com> 可以找到本书的所有例子。

### 2.1 JavaScript 和 Document Object Model

正如第 1 章中所提到的，JavaScript 是 Ajax 的核心。JavaScript 在语法上与以前优秀的 C 语言有一点相似。JavaScript 是一种解释性语言（不需要编译），并且有一定的面向对象编程能力。JavaScript 并不是用来编写功能强大的应用程序的，而是用来实现一个 Web 应用程序

客户端功能的脚本语言。

很多主流的浏览器都完全支持 JavaScript 语言。虽然它们可能自己执行 JavaScript 脚本，但通常情况下，要实现功能就需要将脚本和 HTML 代码一起下载到客户端浏览器。JavaScript 编码需要完整无改动地到达客户端，这既是优点也是缺点。在决定一个 Web 解决方案的框架之前，必须考虑到这些方面。以下这些网址很好地介绍了 JavaScript：

- <http://www.echoecho.com/javascript.htm>
- <http://www.devlearn.com/javascript/jsvars.html>
- <http://www.w3schools.com/js/default.asp>

JavaScript 的一个优势在于，在客户端它有能力操作上级（父）HTML 文档，它通过 DOM 接口来实现。DOM 在多种语言和技术下都是有效的，包括 JavaScript、PHP、C#及 C++等。在这一章里我们将讲述如何在 JavaScript 和 PHP 下使用 DOM。DOM 可以把 XML 当作文档一样进行操作（如创建、修改、解析、查找等），也包括 HTML。

在客户端，可以使用 DOM 和 JavaScript 来：

- 操作正在使用的 HTML 页面；
- 读出并解析从服务器端接收的 XML 文档；
- 建立新的 XML 文档。

在服务器端，可以使用 DOM 和 PHP 来：

- 完成 XML 文档，通常用于发送到客户端；
- 解读从各种来源收到的 XML 文档。

在 <http://quirksmode.org/dom/intro.html> 和 <http://javascriptkit.com/javatutors/dom.shtml> 可找到关于 DOM 的详尽介绍，在下面的地址可以找到一个好玩的 DOM 游戏 <http://www.topxml.com/learning/games/b/default.asp>，在网址 <http://krook.org/jsdom/> 可以找到有助于理解 JavaScript DOM 的资料。在 <http://www.mozilla.org/docs/dom/refernce/javascript.html> 可以找到 Mozilla 浏览器关于 JavaScript DOM 的资料。

在本章的第一个例子中，将使用 JavaScript 中的 DOM 来操作 HTML 文档。当在 HTML 文件中加入 JavaScript 代码时，有一段是在 <body> 中的 <Script> 元素内写 JavaScript 代码。以下面这个 HTML 文件为例，它在下载之后执行了一些简单的 JavaScript 代码。请注意文件对象，是 JavaScript 中默认对象，它与 HTML 页面的 DOM 对象交互。这里是用写的方法向页面添加内容：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
<head>
<title>Ajax Foundations: JavaScript and DOM</title>
<script type="text/javascript">
//声明新的变量
```



```

var date = new Date();
var hour = date.getHours();
//根据条件显示内容
if(hour>=22 || hour<=5)
    document.write("You should go to sleep");
else
    document.write("Hello, world!");
</script>
</head>
<body>
</body>
</html>

```

当脚本执行时，`document.write` 命令生成输出，添加在页面的`<body>`元素中。生成的内容变成页面 HTML 代码的一部分，所以可以在需要的地方添加 HTML 标记。

建议尽可能地写出规范的和有效的 HTML 代码。编写代码尽量使 HTML 格式，以最大限度地适应大多数现有的和未来的浏览器。网址 <http://www.w3.org/QA/2002/04/web-quality> 有一篇关于未来 Web 标准的文章。网址 <http://alistapart.com/stories/doctype> 是一篇非常有用的对 DOCTYPE 元素的解释。关于标准的争论似乎是无止境的，一些人热衷于严格遵循标准，还有一些人则只在意他们的页面在某一些浏览器中看起来是否很美观。除了几个为了便于理解而打破常规的特例外，本书中的例子都是有效的 HTML 代码。事实上，出于各种原因，很少有网页能做到严格遵循标准。

通常更习惯将 JavaScript 代码写在分开的.js 文件中，再在.html 文件中引用它们。这使得 HTML 代码保持整洁，并且使得 JavaScript 代码组织在一个文档中。在 HTML 代码中可以通过在`<head>`元素中增加以下`<script>`元素，来引用 JavaScript 文档。

```

<html>
<head>
<script type="text/javascript" src="file.js"></script>
</head>
</html>

```

**⚠ 注意** 如果在`<script>`和`</script>`之间没有任何代码，也不要尝试使用短结构`<script type="text/javascript" src="file.js" />`，这将使得 IE6 无法加载 JavaScript 页面。

让我们来做一个简短的练习。

## 》》》 实现步骤——从 JavaScript 和 Dom 开始

(1) 在 ajax 文件夹下创建 foundations 文件夹。本章和下一章的所有例子都使用这个文

文件夹。

(2) 在 `foundations` 文件夹下，创建一个名为 `jsdom` 的子文件夹。

(3) 在 `jsdom` 文件夹中，添加一个名为 `jsdom.html` 的文件，并写入如下代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
  <head>
    <title>Ajax Foundations: JavaScript and DOM</title>
    <script type="text/javascript" src="jsdom.js"></script>
  </head>
  <body>
    I love you!
  </body>
</html>
```

(4) 在相同文件夹下创建一个名为 `jsdom.js` 的文件，写入如下代码：

```
//声明新的变量
var date = new Date();
var hour = date.getHours();
//根据条件显示内容
if (hour >= 22 || hour <= 5)
  document.write("Goodnight, world!");
else
  document.write("Hello, world!");
```

(5) 在浏览器中，输入如下网址 `http://localhost/ajax/foundations/jsdom/jsdom.html`，假如不是很晚的话，则出现如图 2.1 所示的页面（如果过了晚上 10 点，就会是另外一个页面）。

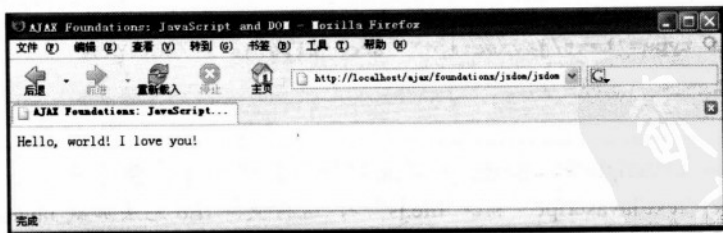


图 2.1 使用 JavaScript 和 DOM 实现 Hello World

## 程序说明

这段代码很简单，不需要过多的解释。下面是需要理解的主要思想。

- 因为没有涉及服务器端脚本（如 PHP 代码），可以直接从本地硬盘中直接将文件加载到浏览器中，而不是通过 HTTP Web 服务器。如果直接从本地执行文件，浏览器通常自动地使用本地地址打开它，比如 `file:///c:/apache2/htdocs/ajax/foundations/jsdom/jsdom.html`。

- 当从本地地址而不是从 Web 服务器加载一个含有 JavaScript 代码的 HTML 页面时，IE 浏览器将会提示用户要使用高级权限来执行一段代码（更多细节在第 3 章讲述）。

- JavaScript 不需要声明变量，所以可以避免变量关键字。尽管这不是一个受欢迎的做法。

- 下载 HTML 页面后，JavaScript 脚本自动执行。还可以使用 JavaScript 函数将代码分组，使这些成组代码只有在明确调用时才被执行。

- JavaScript 代码在解析其他 HTML 代码前执行，所以它的输出在 HTML 输出之前显示。可以注意到“Hello world!”在“I love you”之前出现。

关于上述代码有这样一个问题：无法控制 JavaScript 代码在哪里输出，如何显示。就像上面说的，JavaScript 输出首先出现，然后才是 `<body>` 元素中的内容。这个设定与最简单的应用无关。

除了一些最简单的应用外，在 HTML 页面加载时只有无条件运行的 JavaScript 代码是不够的。通常都需要对部分 JavaScript 代码的执行时间和方式进行更多的控制，最典型的就是什么时候使用 JavaScript 函数，以及在 HTML 页面上的某些事件触发（如单击按钮）后执行这些函数。

## 2.2 JavaScript 事件和 DOM

接下来的练习将通过 JavaScript 代码建立一个 HTML 页面。在准备建立一个含有动态内容的页面时，首先需要建立一个模版（包含一些静态的部分），然后使用占位符来建立动态的部分。占位符必须是有惟一标志的 HTML 元素（设置了 ID 属性的元素）。迄今为止我们只用 `<div>` 元素作为占位符，在本书以后的例子中将会遇到更多的例子。

请看下面的 HTML 文档：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
  <head>
    <title>Ajax Foundations: More JavaScript and DOM</title>
  </head>
  <body >
    Hello Dude! Here's a cool list of colors for you:
    <br />
    <ul>
```

```

<li>Black</li>
<li>Orange</li>
<li>Pink</li>
</ul>
</body>
</html>

```

假如需要动态生成<ul>元素里的所有内容。在 Ajax 应用程序中典型的做法是在需要动态生成内容的位置放置一个空的<div>元素（命名元素）。

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
<head>
<title>Ajax Foundations: More JavaScript and DOM</title>
</head>
<body >
Hello Dude! Here's a cool list of colors for you:
<br />
<div id="myDivElement" />
</body>
</html>

```

在这个例子中，将使用<div>元素将 JavaScript 代码生成的内容嵌入 HTML 文档中，但是请记住 HTML 元素的 ID 是可以自由设定的。在 JavaScript 代码执行后，将<ul>元素加入<div>元素中，将得到如下的 HTML 结构。

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
<head>
<title> Colors </title>
</head>
<body >
Hello Dude! Here's a cool list of colors for you:
<br/>
<div id="myDivElement" >
<ul>
<li>Black</li>
<li>Orange</li>
<li>Pink</li>
<ul>

```

```

</div>
</body>
</html>

```

下面练习的目的是：

- 通过 JavaScript 函数访问<div>元素。
- 在 HTML 模板加载之后执行 JavaScript 代码，所以可以访问<div>元素（在<head>元素中执行的 JavaScript 代码无法访问任何 HTML 元素）。可以通过<body>元素的装载事件来调用 JavaScript 代码。
- 将 JavaScript 代码组织在函数中，以便更容易控制。

## 》》》 实现步骤——使用 JavaScript 事件和 DOM

- (1) 在前面建立的 foundations 文件夹下，创建一个名为 morejsdom 的文件夹。
- (2) 在 morejsdom 文件夹下，创建一个名为 morejsdm.html 的文件，并加入如下代码：

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
  <head>
    <title>Ajax Foundations: More JavaScript and DOM</title>
    <script type="text/javascript" src="morejsdom.js"></script>
  </head>
  <body onload="process()">
    Hello Dude! Here's a cool list of colors for you:
    <br />
    <div id="myDivElement" />
  </body>
</html>

```

- (3) 创建一个名为 morejsdom.js 的文件，加入如下代码：

```

function process()
{
  //创建 HTML 代码
  var string;
  string = "<ul>"
    + "<li>Black</li>"
    + "<li>Orange</li>"
    + "<li>Pink</li>"
    + "</ul>";
}

```

```
//获得 页面中<div> 元素的引用  
myDiv = document.getElementById("myDivElement");  
//向 <div> 添加内容  
myDiv.innerHTML = string;  
}
```

(4) 使用浏览器打开 more.jsdom.html, 得到如图 2.2 所示的页面。

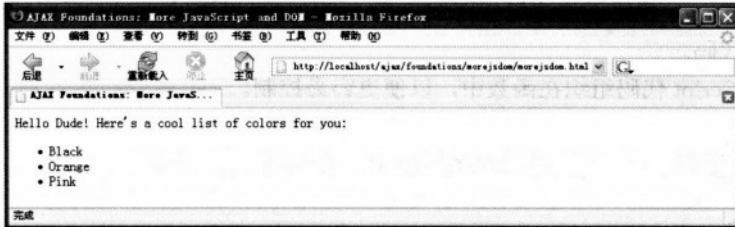


图 2.2 简单的 HTML 网页

## 程序说明

代码相当简单。在 HTML 代码中, 下面代码片段中加黑的部分比较重要。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"  
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">  
<html>  
  <head>  
    <title>Ajax Foundations: More JavaScript and DOM</title>  
    <script type="text/javascript" src="morejsdom.js"></script>  
  </head>  
  <body onload="process()">  
    Hello Dude! Here's a cool list of colors for you:  
    <br/>  
    <div id="myDivElement" />  
  </body>  
</html>
```

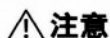
一切都是由<script>元素调用 JavaScript 源文件开始的。JavaScript 文件中包含一个由 body 元素的 onload 事件引发的事件驱动函数 process()。onload 事件在 HTML 文件完全加载后引发, 所以当 process 函数执行的时候, 它访问的是整个 HTML 结构。process 函数首先创建需要加入 div 元素的 HTML 代码:

```
function process()  
{
```

```
//创建 HTML 代码
var string;
string = "<ul>"
    + "<li>Black</li>"
    + "<li>Orange</li>"
    + "<li>Pink</li>"
    + "</ul>";
```

然后，通过 `document` 对象的 `getElementById` 函数获得一个 `myDivElement` 元素的引用。`document` 是 JavaScript 的默认对象，代表当前 HTML 文档。

```
//获得页面中<div>元素的引用
myDiv = document.getElementById("myDivElement");
```



**注意**

JavaScript 既允许使用单引号，也允许使用双引号来表示 `string` 类型的变量。上一段的代码也可以写成：

```
myDiv=document.getElementById('myDivElement');
```

在 JavaScript 中，只要前后保持一致，使用它们中的任何一个都是可以的。但如果同一脚本中，同时使用了双引号和单引号，将引起解析错误。本书中将使用双引号。

最后，将已经编好的 `string` 变量的 HTML 代码加入到 `myDivElement` 元素中。

```
//在 div 元素中加入内容
myDiv.innerHTML = string;
}
```

在这个例子中我们使用了 DOM 的 `innerHTML` 属性，将已经准备好的 HTML 代码加入到文档中。

## 2.3 关于 DOM 的进一步介绍

在前面的练习中，加入字符串组成的简单 HTML 结构，通过这种方式建立了元素列表。同样的 HTML 结构也可以通过 DOM 编程创建。在下一个练习中，将编程生成：

```
<div id="myDivElement" >
  嘿！以下是一个很酷列表：
  <br/>
  <ul>
    <li>Black</li>
```

```
<li>Orange</li>
<li>Pink</li>
<ul>
</div>
```

DOM 文档就是元素的分层组织形式，每一层元素都有一个或者更多的属性。在这个 HTML 代码段中有一个 id 属性值为 myDivElement 的<div>属性。通过文档对象可以访问的根节点是<body>。在完成上面的 HTML 文档后，可以得出如图 2.3 所示的结构。

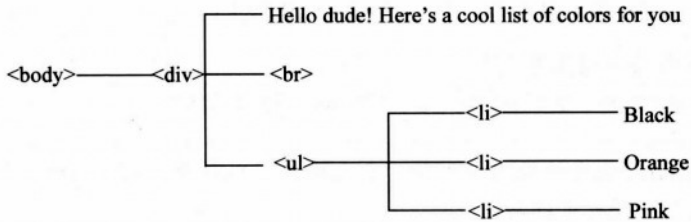


图 2.3 HTML 元素的层次图

在图 2.3 所示中，可以看到 HTML 结构，包含<body>、<div>、<br>、<ul>和<li>元素以及 4 个文本节点“Hello...”、“Black”、“Orange”、“Pink”。在下一个练习中，我们将使用 DOM 的 createElement、createTextNode 和 appendChild 函数来创建这个结构。

### 》》》 实现步骤——更多关于 DOM 的练习

- (1) 在 foundations 文件夹下，创建一个名为 evenmorejsdom 的子文件夹。
- (2) 在 evenmorejsdom 文件夹下，创建一个名为 evenmorejsdom.html 的文件，加入如下代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
  <head>
    <title>Ajax Foundations: Even More JavaScript and DOM</title>
    <script type="text/javascript" src="evenmorejsdom.js"></script>
  </head>
  <body onload="process()">
    <div id="myDivElement" />
  </body>
</html>
```



(3) 创建一个名为 `evenmorejsdom.js` 的文件，加入如下代码：

```
function process()
{
    //创建第一个文本节点
    oHello = document.createTextNode
        ("Hello Dude! Here's a cool list of colors for you:");

    //创建 <ul> 元素
    oUl = document.createElement("ul")

    //创建第一个 <li>元素并添加文本节点
    oLiBlack = document.createElement("li");
    oBlack = document.createTextNode("Black");
    oLiBlack.appendChild(oBlack);

    //创建第二个<li> 元素并添加文本节点
    oLiOrange = document.createElement("li");
    oOrange = document.createTextNode("Orange");
    oLiOrange.appendChild(oOrange);

    //创建第三个<li> 元素并添加文本节点
    oLiPink = document.createElement("li");
    oPink = document.createTextNode("Pink");
    oLiPink.appendChild(oPink);

    //把 <li>元素作为子节点加入<ul> 元素中
    oUl.appendChild(oLiBlack);
    oUl.appendChild(oLiOrange);
    oUl.appendChild(oLiPink);

    //获得关于 页面中<div> 元素的引用
    myDiv = document.getElementById("myDivElement");

    //将内容加入 <div> 元素
    myDiv.appendChild(oHello);
    myDiv.appendChild(oUl);
}
```

(4) 在浏览器中打开 `evenmorejsdom.html`，会得到如图 2.4 所示的页面。

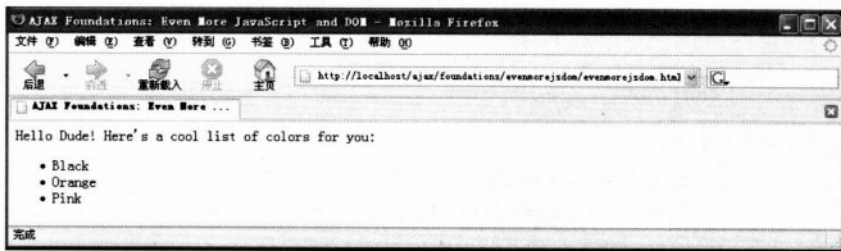


图 2.4 更多关于 JavaScript 和 DOM

## 程序说明

和上一个例子一样，但是可以看出 `process` 函数比上一个例子使用了更多的代码。尽管多了几行，但是功能一样简单。这清楚地表明使用 DOM 建立 HTML 结构不是最佳的选择。但是，基于以下原因，在特殊的环境中使用 DOM 确实能使得编程变得更加简单。

- 因为不必考虑文本的格式，只考虑到建立元素结构，这使得通过程序建立动态的 HTML 结构变得更加容易，比如使用循环建立元素。
- 作为一种程序的结果，例如，不需要手动添加结束标签。当添加一个“`ui`”元素的时候，DOM 会生成一个 `<ui>` 标签并附加一个结束标签 `</ui>`。
- 可以把各个节点看作是独立的节点，然后再考虑如何建立层次关系。只需告诉它想怎么样，DOM 会处理一些未完成的细节。

## 2.4 JavaScript、DOM 和 CSS

大家一定非常熟悉 CSS (Cascading Style Sheets)。CSS 允许 HTML 文件引用一个集中设置格式风格的文档。如果在一个网页中，设置得当并且一致，CSS 将允许编辑 CSS 文件，只改变一点设置就可改变整个（或部分的）页面的外观。关于 CSS 的书和介绍有很多，从下面两个网址可以找到一些免费的介绍 CSS 的文章——<http://www.w3.org/style/css/> 和 <http://www.w3schools.com/css/default.asp>。尽管发明 Ajax 这个名字的文章 (<http://www.adaptivepath.com/publications/essays/archives/000385.php>) 提到了 CSS 作为 Ajax 的一个部分，但是在技术上，并不一定需要 CSS 才能成功创建动态网页。CSS 由于其优点而被大力推广。

下面将作一个简单的练习，演示如何使用 CSS 以及使用 DOM 控制 HTML 元素的风格。这是在建立 Ajax 应用时经常要做的工作。在下面的练习中，将要画一个漂亮的表格以及两个按钮 Set Style 1 和 Set Style 2。这两个按钮可以通过改变当前的风格来改变表格的颜色和外观。图 2.5 展示了我们想要的风格。

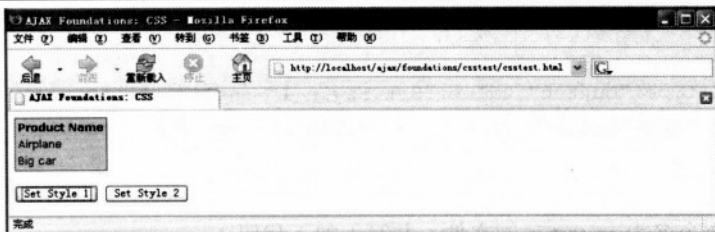


图 2.5 使用了 JavaScript 和 CSS 的表格

## 》》》 实现步骤——使用 CSS 和 JavaScript

- (1) 在 foundations 文件夹下创建一个名为 csstest 的子文件夹。
- (2) 在新建的 csstest 文件夹下创建一个名为 csstest.html 的文件，并加入如下代码：

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
  <head>
    <title>Ajax Foundations: CSS</title>
    <script type="text/javascript" src="csstest.js"></script>
    <link href="styles.css" type="text/css" rel="stylesheet"/>
  </head>

  <body>
    <table id="table">
      <tr>
        <th id="tableHead">
          Product Name
        </th>
      </tr>
      <tr>
        <td id="tableFirstLine">
          Airplane
        </td>
      </tr>
      <tr>
        <td id="tableSecondLine">
          Big car
        </td>
      </tr>
    </table>
  </body>
</html>

```

```
</table>
<br />
<input type="button" value="Set Style 1" onclick="setStyle1();" />
<input type="button" value="Set Style 2" onclick="setStyle2();" />
</body>
</html>
```

(3) 创建一个名为 `csstest.js` 的文件，并写入如下代码：

```
//将表格的风格改变为 style 1
function setStyle1()
{
    //获取 HTML 元素的引用
    oTable = document.getElementById("table");
    oTableHead = document.getElementById("tableHead");
    oTableFirstLine = document.getElementById("tableFirstLine");
    oTableSecondLine = document.getElementById("tableSecondLine");
    //设置风格
    oTable.className = "Table1";
    oTableHead.className = "TableHead1";
    oTableFirstLine.className = "TableContent1";
    oTableSecondLine.className = "TableContent1";
}

//将表格的风格改变为 style 2
function setStyle2()
{
    //获取 HTML 元素的引用
    oTable = document.getElementById("table");
    oTableHead = document.getElementById("tableHead");
    oTableFirstLine = document.getElementById("tableFirstLine");
    oTableSecondLine = document.getElementById("tableSecondLine");
    //设置风格
    oTable.className = "Table2";
    oTableHead.className = "TableHead2";
    oTableFirstLine.className = "TableContent2";
    oTableSecondLine.className = "TableContent2";
}
```

(4) 最后，创建 CSS 文档 `styles.css`：

```
.Table1
{
```

```
border: DarkGreen 1px solid;
background-color: LightGreen;
}
.TableHead1
{
font-family: Verdana, Arial;
font-weight: bold;

font-size: 10pt;
}
.TableContent1
{
font-family: Verdana, Arial;
font-size: 10pt;
}

.Table2
{
border: DarkBlue 1px solid;
background-color: LightBlue;
}
.TableHead2
{
font-family: Verdana, Arial;
font-weight: bold;
font-size: 10pt;
}
.TableContent2
{
font-family: Verdana, Arial;
font-size: 10pt;
}
```

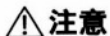
(5) 在浏览器中打开 <http://localhost/ajax/foundations/css/css.html>，可以测试两个按钮的工作情况。

## 程序说明

style.css 文件中包含了两种可以应用于 csstest.html 中表格的风格。当用户单击某个 set style 按钮时，JavaScript DOM 将指定的风格应用到表格的元素上。

在 `Setstyle` 方法的第一部分，使用了 `getElementById` 函数来获得需要设置 CSS 风格的元素的引用：

```
//获得 HTML 元素的引用
oTable = document.getElementById("table");
oTableHead = document.getElementById("tableHead");
oTableFirstLine = document.getElementById("tableFirstLine");
oTableSecondLine = document.getElementById("tableSecondLine");
```

**注意**

对于很多其他网页开发工作来说，CSS 已经成为不同浏览器间明显不同的主要原因。例如，在先前的程序段中，如果把对象命名为与 HTML 页面中关联的元素名称相同（如将 `otable` 改名为 `table`），IE 将停止工作。IE 不允许与 HTML 文档中已存在对象的 ID 同名。这个问题没什么太大的意义，因为对象有不同的作用域，但是如果想让代码在 IE 下一切正常的话，最好多加注意。

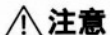
在初始化对象时，确保所有浏览器都能支持的安全做法是用元素的类名属性（`className`）来设置它们的 CSS 样式：

```
//设置风格
oTable.className = "Table1";
oTableHead.className = "TableHead1";
oTableFirstLine.className = "TableContent1";
oTableSecondLine.className = "TableContent1";
```

## 2.5 使用 XMLHttpRequest 对象

XMLHttpRequest 对象使得 JavaScript 代码能够实现异步 HTTP 服务器请求，即在后台实现 HTTP 请求，接收响应，更新部分网页时，在视觉上不中断用户的体验。当向服务器请求数据时，可以使用户界面还能够响应是很重要的。

XMLHttpRequest 对象最初由微软在 1999 年作为 IE 中的一个 ActiveX 对象创建。最后成为除了 IE6 外所有浏览器都把 XMLHttpRequest 作为一个本地对象支持的事实上的标准。

**注意**

尽管 XMLHttpRequest 已经成为浏览器的事实标准，但它并不是 W3C 标准。类似的功能由 W3C DOM Level 3 Load and Save 标准提出，但还未被浏览器支持。

使用 XMLHttpRequest 典型的操作顺序如下：

(1) 创建一个 XMLHttpRequest 对象的实例。

(2) 使用 XMLHttpRequest 向服务器页面发送异步的请求，定义当接收到服务器响应后可自动执行的回调函数。

(3) 在回调函数中处理服务器响应。

(4) 转到第 2 步。

来看一下这些步骤的代码实现。

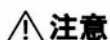
## 2.5.1 创建 XMLHttpRequest 对象

不同浏览器实现 XMLHttpRequest 的方法不同。在 IE6 以及 IE 更老的版本中，XMLHttpRequest 是 ActiveX 控制控件。可以像下面语法这样声明：

```
Xmlhttp = new ActiveObject("Microsoft.XMLHttp");
```

在其他浏览器中，XMLHttpRequest 作为本地对象，可以用以下的方式来建立实例：

```
Xmlhttp = new XMLHttpRequest();
```



**注意** 可以想象 ActiveX XMLHttpRequest 库有很多不同作用和版本。每一款微软的软件，包括 IE 和 MDAC，都发布了新版本的 ActiveX 控件。Microsoft.XMLHttp 是最早的版本，虽然新版本在性能和特点方面有所改进，但老版本在基本操作中是可以放心使用的。以后将学习如何自动使用更新的版本。

下面是本书中跨浏览器的 XMLHttpRequest 实例化代码的简化版本：

```
//创建一个 XMLHttpRequest 实例
function createXmlHttpRequestObject()
{
    //定义一个用于存储 XMLHttpRequest 对象的引用
    var xmlhttp;
    //try 程序段将适应除了 IE6 及其更早版本外的所有浏览器
    try
    {
        //尝试创建 XMLHttpRequest 对象
        xmlhttp = new XMLHttpRequest();
    }
    catch(e)
    {
        //假设是 IE6 或其更早版本
        try
        {
            //尝试创建 XMLHttpRequest 对象
```

```
        xmlhttp = new ActiveXObject("Microsoft.XMLHttp");
    }
    catch (e) {}
}
//返回创建的对象或显示错误信息
if (!xmlHttp)
    alert("Error creating the XMLHttpRequest object.");
else
    return xmlhttp;
}
```

这个函数将返回一个 XMLHttpRequest 对象实例。功能实现依赖于 JavaScript 的 try/catch 机制。

try/catch 机制最初源自面向对象程序语言 (OOP)，为 JavaScript 提供了强大的异常控制 (Exception-handling) 技术。一般地，当 JavaScript 代码出现错误时，程序将抛出一个异常 (Exception)。异常以对象的形式存在，包含描述错误的细节。使用 try/catch 语法，可以捕获异常并在本地处理，使得错误无须传递到用户的浏览器上。

try/catch 语法如下：

```
try
{
    //可能产生异常的代码
}
catch(e)
{
    //只有当 try 程序段出现异常才执行
    //可通过参数 e 获取异常的细节
}
```

可以将任何可能产生错误的代码放到 try 程序段中，如果产生错误，执行会立即转到 catch 段。如果没有发生错误，则 catch 段不会执行。

运行时产生的异常会从产生点沿着程序的调用堆栈一直向上传递，如果不在本地进行异常处理，浏览器就会捕获异常，最终会显示给访问者一个错误信息，显得很不好看。

响应异常的方法与异常发生时的情况息息相关，有时可以简单地忽略掉错误，有时在代码中加错误标记，或者向用户显示一个出错信息。在本书中会遇到各种情况。

在上面的例子中，如果想建立一个 XMLHttpRequest 对象，可以首先尝试将它作为一个浏览器本地对象来创建，代码如下：

```
//下面的代码适合除 IE6 及更早版本 IE 外所有的浏览器。
Try
```



```

{
//尝试建立 XMLHttpRequest 对象
    XmlHttp = new XMLHttpRequest();
}

```

IE7、Mozilla、Opera 及其他浏览器会很好地执行上述代码，不会产生错误，因为本地支持 XMLHttpRequest。但 IE6 及 IE 更早的版本无法认出 XMLHttpRequest 对象，会产生异常，执行将会转到 catch 段。对于 IE6 及 IE 更早的版本，需要把 XMLHttpRequest 对象当作 ActiveX 控件创建：

```

catch(e)
{
//假设浏览器是 IE6 及 IE 更早的版本
    try
    {
        xmlHttp = new ActiveXObject("Microsoft.XMLHttp");
    }
    catch(e){}
}

```

使用 JavaScript 的编程者越多，创建 XMLHttpRequest 对象的方法越多，令人惊奇的是这些方法都能很好的发挥作用。本书更倾向于在 try/catch 块中使用这个方法，因为这样可以更好地适应未来的浏览器，并且使用较少的代码完成正确的错误检查。

另外，还可以在创建之前，使用 typeof 检查浏览器是否支持 XMLHttpRequest：

```

if(typeof XMLHttpRequest != "undefined" )
    xmlHttp = new XMLHttpRequest();

```

通常，使用 typeof 是十分有效的。但在这个例子中，使用 typeof 并不能取代 try/catch 异常处理，结果仅仅是多写了些代码。

另一个可行的办法是使用 JavaScript 特有的对象检测。允许检测浏览器是否支持某个特定的对象，方法如下：

```

if(window.ActiveXObject)
    xmlHttp = new XMLHttpRequest();

```

例如，通过检测 window.ActiveXObject，可以确定浏览器是否是 IE。同样，它除了增加些代码外没有任何优点，但这种思想值得借鉴。

如果要使用对象检测方法，必须保证在检测 ActiveX 对象之前首先检测 XMLHttpRequest 对象。这个建议的是 IE7 同时支持 ActiveX 和 XMLHttpRequest 而后者更加合适，因为它可以给您最新的对象版本。对于 ActiveX，需要使用相当一部分代码才能确定它是否为较新的

版本，但却无法保证是最新版本。

在 `createXMLHttpRequestObject` 函数结尾处，在做了各种尝试后，我们做了测试，并返回了有效的 `XMLHttpRequest` 实例。

```
//返回创建的对象或显示出错误信息
if (!xmlHttp)
    alert("Error creating the XMLHttpRequest object.");
else
    return xmlHttp;
```

**⚠ 注意** 对象检测的副作用比这个特性本身还要有用。对象检测可以评估 JavaScript 的对象实例是否有效，如 `xmlHttp`。一个好处是 `(!xmlHttp)` 表达式为真的话，不仅表示 `xmlHttp` 无效，还可能表示 `xmlHttp` 为空或未定义。

## 》》》 为 IE 创建更好的对象

当浏览器为 Internet Explorer 6 时，可以改进 `createXMLHttpRequestObject` 函数，使其能够认出最新版本的 ActiveX 控件。大多数情况下，可以使用 `ActiveXObject("Microsoft.XMLHttp")` 提供的基础函数来实现，但如果打算使用更新的版本，则需要另作处理。

典型的解决办法是，尝试创建一个已知的最新版本，如果失败，忽略产生的错误，尝试创建次数的版本，直到获得对象而不是异常为止。最新版本的 `XMLHTTPActiveX Object` 程序 ID (prog ID) 是 `MSXML2.XMLHTTP.6.0`。如果想了解程序 ID 的更多细节，或者想对复杂的程序 ID 有一个简单清楚的认识，可以参考一些网上的免费资源，如 <http://puna.net.nz/etc/xml/msxml.htm>。

下面是 `createXMLHttpRequestObject` 函数的升级版。新加入部分作了突出显示。

```
//创建 XMLHttpRequest 实例
function createXMLHttpRequestObject()
{
    //定义一个用于存储 XMLHttpRequest 对象的引用
    var xmlHttp;
    //try 程序段将适应除了 IE6 及其更早版本外的所有浏览器
    try
    {
        //尝试创建 XMLHttpRequest 对象
        xmlHttp = new XMLHttpRequest();
    }
    catch(e)
```

```

{
    //假设是 IE6 或其更早版本
    var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
                                     "MSXML2.XMLHTTP.5.0",
                                     "MSXML2.XMLHTTP.4.0",
                                     "MSXML2.XMLHTTP.3.0",
                                     "MSXML2.XMLHTTP",
                                     "Microsoft.XMLHTTP");

    //顺序尝试创建每个版本，直到有一个创建成功
    for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
    {
        try
        {
            //尝试创建 XMLHttpRequest 对象
            xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
        }
        catch (e) {} //忽略产生的错误
    }
}

//返回创建的对象或显示错误信息
if (!xmlHttp)
    alert("Error creating the XMLHttpRequest object.");
else
    return xmlHttp;
}

```

这段代码虽然看起来很复杂，但功能却十分简单。首先尝试建立一个 MSXML2.XMLHttp.6.0 ActiveX 对象。如果失败，忽略产生的错误，然后代码尝试创建 MSXML2.XMLHttp.5.0 ActiveX 对象，如此下去，直到成功创建一个对象为止。

值得注意的是在新代码中，使用了对象检测 (!xmlHttp) 来确保在成功创建对象后，有效地终止了循环的执行。

## 2.5.2 使用 XMLHttpRequest 初始化服务器请求

XMLHttpRequest 对象创建后，可以做很多有趣的事情。虽然由于浏览器和版本问题，有很多种方法创建 XMLHttpRequest 对象，但所有 XMLHttpRequest 实例都有相同的 API 并支持同样的功能（事实上，由于每个浏览器的实现方法不同，所以不能完全保证这点）。

表 2-1 描述了 XMLHttpRequest 对象的方法和属性，可以从中了解更多关于 XMLHttpRequest 的细节。

表 2-1

方法/属性和描述

方法/属性	描 述
abort()	停止当前的请求
getAllResponseHeaders()	将所有响应的头部组为一个字符串返回
getResponseHeader("headerLabel")	将指定响应的头部组为一个字符串返回
open("method","URL" [,asyncFlag[, "userName" [, "password"]]])	初始化请求参数
send(content)	发送 HTTP 请求
setRequestHeader("label","value")	设置请求头部的标签/值
onreadystatechange	常用于设置控制请求状态变化的回复函数
readyState	返回请求的状态: 0 = 未初始化 1 = 下载中 2 = 下载已完成 3 = 交互的 4 = 完成
responseText	将服务器响应以字符串形式返回
responseXML	将服务器响应以 XML 文档形式返回
status	返回请求的状态码
statusText	返回请求的状态信息

对于每个服务器请求都要使用的方法是 `open` 和 `send`。方法 `open` 通过各种参数来设置请求，`send` 则用于向服务器发送请求。在调用 `send` 前需要设置 `onreadystatechange` 属性，当请求状态发生改变时执行回调程序异步地发出请求这就是 Ajax 的机制。

方法 `open` 是用来初始化请求的。它有两个必需的参数和几个可选的参数。方法 `open` 并不发起一个到服务器的连接。第一个参数表示向服务器发送数据的方法，可以是 `get`、`put` 或 `post`。第二个参数是 URL，表明发送消息的目的地。URL 可以是绝对地址也可以是相对地址。如果没有标明 URL 是通过 HTTP 访问的资源，第一个参数将被忽略。

第三个参数是 `async`，用于表示请求是否需要异步操作。值为 `true` 表示在执行 `send()` 后，脚本继续执行，无须等待响应。值为 `false` 表示在得到响应之前，脚本停止运行，并且冻结页面的功能。为了实现异步处理，需要设置 `async` 的属性为 `true`，并且利用 `onreadystatechange` 事件中处理从服务端接收到的响应。

用 GET 方法来传递参数是通过使用 URL 序列串传递的，如 `http://localhost/ajax/test.php?param1=x&&param2=y`。这个服务器请求传递了两个参数——其中的一个叫做 `param1`（值为 `x`），另一叫做 `param2`（值为 `y`）。

```
//调用服务器页面，执行服务器端操作
xmlHttp.open("GET"," http://localhost/ajax/test.php?param1=x&&param2=y",
true);
xmlHttp.onreadystatechange = handleRequestStateChange;
xmlHttp.send(null);
```

POST 方法是通过 send 方法的参数序列串来传递参数的，而不是在原始的 URL 上添加字符串，形式如下。

```
//调用服务器页面，执行服务器端操作
xmlHttp.open("POST", "http://localhost/ajax/test.php", true);
xmlHttp.onreadystatechange = handleRequestStateChange;
xmlHttp.send("param1=x&&param2=y");
```

上面两个例子具有相同的效果。在实践中，因为可以模拟浏览器的 GET 请求，所以使用 GET 方法有助于调试程序，这样就可以清楚地看到服务器脚本的运行结果。在发送数据大于 512 字节时，GET 方法失效，需要使用 POST 方法。

在例子中将实现 HTTP 请求的代码放在 JavaScript 文件中一个名为 process() 的函数内。其不包含任何错误处理的最简单的实现代码如下。

```
Function process()
{
//调用服务器页面，执行服务器端操作
    xmlHttp.open("GET", "server_script.php", true);
    xmlHttp.onreadystatechange = handleRequestStateChange;
    xmlHttp.send(null);
}
```

这个方法有如下面潜在的问题：

- xmlHttp 中可能没有任何有效的 XMLHttpRequest 实例，process() 也能执行。比如用户浏览器不支持 XMLHttpRequest 时，就可能产生这种现象。这将导致产生无法控制异常。如果存在不一致问题我们的其他错误控制就会失效。

- process() 函数也没有对其他可能发生的错误做好保护。例如，本章后面将演示一些浏览器在 XMLHttpRequest 对象无法访问时，将显示安全性异常（第 3 章将有更多关于安全问题的介绍）。

更加安全可靠的 process() 函数如下。

```
//调用时读服务器端的文件
function process()
{
//只有在 xmlHttp 不为空的时候才继续
    if(xmlHttp)
    {
        //尝试连接服务器
        try
        {
            //对从服务器上读文件进行初始化
```

```
xmlHttp.open("GET","server_script.php",true);
xmlHttp.onreadystatechange = handleRequestStateChange;
xmlHttp.send(null);
}
//如果失败,显示错误。
catch(e)
{
    alert("can't connect to sever:\n"+e.toString());
}
}
```

如果 `xmlHttp` 为空,将不显示任何信息,因为我们假设在 `creat XMLHttpRequestObject` 时已经显示过了。我们要确保显示其他的连接问题。

### 2.5.3 服务器响应处理

当进行异步请求时(如前文显示的小段代码),在接收到服务器响应之前,`xmlHttp.send()` 并没有冻结,而是继续照常执行。`handleRequestStateChange` 是一种回调方法,用来控制请求状态的转变。通常调用 4 次,每次请求都将进入一个新的阶段。`readyState` 属性的值可能是下面之一:

- 0 = 未初始化
- 1 = 下载中
- 2 = 下载已完成
- 3 = 交互的
- 4 = 完成

除了中间状态 3 外,其他的都很容易从字面上理解。交互(interactive)状态是响应被部分接收时的状态。在 Ajax 应用中,只使用标志着服务器响应已经被接收的完成状态。

下面的代码是典型的 `handleRequestStateChange` 程序,黑体处是读取服务器响应部分:

```
//当请求的状态发生改变时执行函数
function handleRequestStateChange()
{
    //等到过程结束后继续
    if (xmlHttp.readyState == 4)
    {
        //只有 HTTP 状态为 "OK"时继续
        if (xmlHttp.status == 200)
        {
```

```

    //获取响应
    response = xmlhttp.responseText;
    //(使用 xmlhttp.responseXML 读取 XML 响应作为一个 DOM 对象)
    //对响应进行处理
    //...
    //...
}
}
}

```

我们再一次成功使用了 try/catch 来控制初始化服务器连接以及读取服务器响应时的异常。下面是一个更加可靠的 handleRequestStateChange 函数。

```

//当请求状态改变时函数执行
function handleRequestStateChange()
{
    //如果处理结束, 继续
    if (xmlhttp.readyState == 4)
    {
        //只有在 HTTP 状态为"OK"时继续
        if (xmlhttp.status == 200)
        {
            try
            {
                //得到响应
                response = xmlhttp.responseText;
                //处理响应
                //...
                //...
            }
            catch(e)
            {
                //显示错误信息
                alert("Error reading the response: " + e.toString());
            }
        }
        else
        {
            //显示状态信息
            alert("There was a problem retrieving the data:\n" +
                xmlhttp.statusText);
        }
    }
}

```

```

    }
  }
}

```

接下来，让我们看看这些函数如何在一起工作。

## 》》》 实现步骤——使用 XMLHttpRequest 实现异步调用

- (1) 在 foundations 文件夹下，建立一个名为 async 的子文件夹。
- (2) 在 async 文件夹下，建立文件 async.txt 并且加入如下内容：

```
Hello client!
```

- (3) 在相同的文件夹下建立文件 async.html，并写入如下代码：

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
  <head>
    <title>Ajax Foundations: Using XMLHttpRequest</title>
    <script type="text/javascript" src="async.js"></script>
  </head>
  <body onload="process()">
    Hello, server!
    <br/>
    <div id="myDivElement" />
  </body>
</html>

```

- (4) 创建一个名为 async.js 的文件，加入如下代码：

```

//创建一个 XMLHttpRequest 对象的实例
var xmlhttp = createXmlHttpRequestObject();

//创建 XMLHttpRequest 对象实例的函数
function createXmlHttpRequestObject()
{
  //用于存储 XMLHttpRequest 对象的引用
  var xmlhttp;
  //try 程序段将适应除了 IE6 及其更早版本外的所有浏览器
  try
  {

```



```
//尝试创建 XMLHttpRequest 对象
xmlHttp = new XMLHttpRequest();
}
catch(e)
{
    //假设是 IE6 或其更早版本
    var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
                                     "MSXML2.XMLHTTP.5.0",
                                     "MSXML2.XMLHTTP.4.0",
                                     "MSXML2.XMLHTTP.3.0",
                                     "MSXML2.XMLHTTP",
                                     "Microsoft.XMLHTTP");
    //顺序尝试创建每一个对象，直到成功为止
    for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
    {
        try
        {
            //尝试创建 XMLHttpRequest 对象
            xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
        }
        catch (e) {}
    }
}
//返回已经创建的对象，或显示错误信息
if (!xmlHttp)
    alert("Error creating the XMLHttpRequest object.");
else
    return xmlHttp;
}

//调用读取一个服务器上的文本
function process()
{
    //当 xmlHttp 不为空时继续
    if (xmlHttp)
    {
        //尝试连接服务器
        try
        {
```

```
//开始读取服务器上的 async.txt 文件
xmlHttp.open("GET", "async.txt", true);
xmlHttp.onreadystatechange = handleRequestStateChange;
xmlHttp.send(null);
}
//如果出现异常, 显示错误信息
catch (e)

{
    alert("Can't connect to server:\n" + e.toString());
}
}

//处理 HTTP 响应的函数
function handleRequestStateChange()
{
    //获取页面上 <div> 元素的引用
    myDiv = document.getElementById("myDivElement");
    //显示请求的状态
    if (xmlHttp.readyState == 1)
    {
        myDiv.innerHTML += "Request status: 1 (loading) <br/>";
    }
    else if (xmlHttp.readyState == 2)
    {
        myDiv.innerHTML += "Request status: 2 (loaded) <br/>";
    }
    else if (xmlHttp.readyState == 3)
    {
        myDiv.innerHTML += "Request status: 3 (interactive) <br/>";
    }
    //当转换到状态 4 时, 读取服务器的响应
    else if (xmlHttp.readyState == 4)
    {
        //直到 HTTP 的状态为 "OK"时继续
        if (xmlHttp.status == 200)
        {
            try
            {

```

```

//读取服务器的信息
response = xmlhttp.responseText;
//显示信息
myDiv.innerHTML +=
    "Request status: 4 (complete). Server said: <br/>";
myDiv.innerHTML += response;
}
catch (e)
{
    //显示错误信息
    alert("Error reading the response: " + e.toString());
}
else
{
    //显示状态信息
    alert("There was a problem retrieving the data:\n" +
        xmlhttp.statusText);
}
}
}
}

```

(5) 在浏览器中打开 <http://localhost/ajax/foundations/async/async.html> (这次必须通过 HTTP 服务器来访问, 本地访问已不再适用), 结果与图 2.6 所示相似。



图 2.6 HTTP 请求的 4 种状态代码

### ⚠ 注意

如果浏览器中的内容与图不完全一致, 请不要担心。因为一些浏览器在实现 XMLHttpRequest 时忽略了一些状态代码, 比如 Opera 只支持状态码 3、4, IE 中 XMLHttpRequest 最近的版本只报告状态码 2、3、4。

## 程序说明

为了解整个执行的流程，让我们从程序开始的 `async.html` 文件开始。

```
<html>
  <head>
    <title>Ajax Foundations: Using XMLHttpRequest</title>
    <script type="text/javascript" src="async.js"></script>
  </head>
  <body onload="process()">
```

这段代码隐藏了非常有趣的功能。首先，引用了 `async.js` 文件，而此时那个文件中的代码正在被解析。注意，JavaScript 函数中的代码并没有自动执行，而其余的代码自动执行。这个 JavaScript 文件中只有一行代码没有打包在函数中：

```
//包含着一个 XMLHttpRequest 对象
var xmlhttp = createXMLHttpRequestObject();
```

这样就使得 `xmlhttp` 变量在一开始就包含了一个 `XMLHttpRequest` 对象的实例。`XMLHttpRequest` 对象的实例是由先前的 `createXMLHttpRequestObject` 函数实现的。

`onload` 事件触发了 `process()` 函数的执行。`process()` 函数的执行依赖于 `xmlhttp` 对象的初始化，所以 `process()` 函数只需要关心初始化服务器请求。适当的容错机制用于处理潜在的问题。初始化服务器请求的代码如下：

```
//开始从服务器读取文件 aysnc.txt.
xmlhttp.open("GET", "aysnc.txt", true);
xmlhttp.onreadystatechange = handleRequestStateChange;
xmlhttp.send(null);
```

### 注意

这个脚本无法在本地硬盘上直接执行，如使用 `file://resource` 这种形式，必须通过 HTTP 加载，如果要从本地加载，必须指明 `.txt` 文件的完整路径，这样就可能遇到安全性问题，我们会在以后处理。

假设 HTTP 请求都已成功地初始化并异步执行，则在每次请求状态改变时执行 `handleRequestStateChange` 方法。在实际的程序中会忽略掉状态 4 以外的其他状态。但在本例中为了能够看到前面提到的回调函数的实际执行情况，显示了所有状态信息。

`handleRequestStateChange` 自身并不活跃但它被调用时能很好地完成任务。与同步的 HTTP 调用相比，异步调用允许在收到响应前做其他的任务。

`handleRequestStateChange` 函数首先获得名为 `myDivElement` 的 HTML 元素的引用，用来

显示 HTTP 请求状态变化的过程。

```
function handleRequestStateChange()
{
    //获取页面上<div>元素的引用
    myDiv = document.getElementById("myDivElement");
    //显示请求的状态
    if (xmlHttp.readyState == 1)
    {
        myDiv.innerHTML += "Request status: 1 (loading) <br/>";
    }
    else if (xmlHttp.readyState == 2)
```

当状态变化到 4 时，使用了 `xmlHttp.responseText` 中的代码来处理读取服务器响应。

```
//当转换到状态 4 时，读取服务器的响应。
else if (xmlHttp.readyState == 4)
{
    //直到 HTTP 的状态为"OK"时继续
    if (xmlHttp.status == 200)
    {
        try
        {
            //读取服务器的信息
            response = xmlHttp.responseText;
            //显示信息
            myDiv.innerHTML +=
                "Request status: 4 (complete). Server said: <br/>";
            myDiv.innerHTML += response;
        }
        catch(e)
        {
            //显示错误信息
            alert("Error reading the response: " + e.toString());
        }
    }
    else
    {
        //显示状态信息
        alert("There was a problem retrieving the data:\n" +
            xmlHttp.statusText);
    }
}
```

```

}
}

```

除了容错控制之外，还应注意读出服务器响应的 `xmlHttpRequest.responseText` 方法。这个方法有个类似方法，名叫 `xmlHttpRequest.xml`，用于处理 XML 格式的响应。

### ⚠ 注意

事实上，除了 `XMLHttpRequest` 对象的 `responseXML` 方法的名字外没有任何 XML 出现（刚刚练习过的就是一个很好的例子）。这个对象叫“`HttpRequest`”更合适。XML 前缀可能是当初 Microsoft 为了好听才加上去的，那个时候 XML 是个非常时髦的词，就像今天的 Ajax 一样。将来有一天，如果看见名叫 `AjaxRequest` 的对象也不要感到奇怪。

## 2.6 使用 XML 结构

XML 文档与 HTML 文档类似，都以文本为基础，包含层次结构的元素，可以打包和传送各种数据，所以在过去的几年中非常流行。

顺便说一下，XML 将 X 带到了 Ajax 的名字和 `XMLHttpRequest` 的前缀里。需要注意，是否使用 XML 是可选择的。在先前的练习中，实现了一个异步调用，只接收一个文本的简单程序，不涉及 XML。

### ⚠ 注意

XML 及与 XML 相关的技术构成了很大一个主题。您会经常听人们谈论到 DTD、schema、名字空间（namespace）XSLT 和 Xpath、XLink 和 XPointer 等。在本书中会经常使用 XML 来传输简单的数据结构。下面是一些关于 XML 的相关介绍 <http://www.xmlnews.org/docs/xml-basics.html>，<http://www.w3schools.com/xml/default.asp> 也是一个好去处。在 <http://ajaxphp.packtpub.com> 的附录 C 中有关于 XSLT 和 Xpath 的介绍。

可以通过 DOM 控制 XML 文件，就像使用 DOM 控制 HTML 文件一样。下面这个练习和先前的从服务器上读取静态文件类似。不同的是要使用 DOM 来读取 XML 文档。

### 》》》 实现步骤——使用 `XMLHttpRequest` 和 XML 实现异步调用

(1) 在 `foundations` 文件夹下，建立一个名为 `xml` 的子文件夹。

(2) 在 `xml` 文件夹下，创建一个名为 `books.xml` 的文件，用来存放由 JavaScript 的 DOM 读取的 XML 结构。写入如下代码：

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<response>

```

```

<books>
  <book>
    <title>
      Building Responsive Web Applications with Ajax and PHP
    </title>
    <isbn>
      1-904811-82-5
    </isbn>
  </book>
  <book>
    <title>
      Beginning PHP 5 and MySQL E-Commerce: From Novice to Professional
    </title>
    <isbn>
      1-59059-392-8
    </isbn>
  </book>
</books>
</response>

```

(3) 在同一个文件夹下，创建文件 books.html 文件，写入如下代码：

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
  <head>
    <title>Ajax Foundations: JavaScript and XML</title>
    <script type="text/javascript" src="books.js"></script>
  </head>
  <body onload="process()">
    Server, tell me your favorite books!
    <br/>
    <div id="myDivElement" />
  </body>
</html>

```

(4) 最后，创建 books.js 文件：

```

//获得一个 XmlHttpRequest 对象
var xmlHttp = createXmlHttpRequestObject();
//创建一个 XMLHttpRequest 实例
function createXmlHttpRequestObject()

```

```
{
    //存储 XMLHttpRequest 对象的引用
    var xmlhttp;
    //适合除了 IE6 及其更老版本以外的所有浏览器
    try
    {
        //尝试创建 XMLHttpRequest 对象
        xmlhttp = new XMLHttpRequest();
    }
    catch(e)
    {
        //假设是 IE6 或其更老版本
        var XmlHttpVersions = new Array('MSXML2.XMLHTTP.6.0',
                                          'MSXML2.XMLHTTP.5.0',
                                          'MSXML2.XMLHTTP.4.0',
                                          'MSXML2.XMLHTTP.3.0',
                                          'MSXML2.XMLHTTP',
                                          'Microsoft.XMLHTTP');
        //顺序尝试建立每个对象，直到有一个创建成功为止
        for (var i=0; i<XmlHttpVersions.length && !xmlhttp; i++)
        {
            try
            {
                //尝试创建 XMLHttpRequest 对象
                xmlhttp = new ActiveXObject(XmlHttpVersions[i]);
            }
            catch (e) {}
        }
    }
    //返回创建的对象或显示错误信息
    if (!xmlhttp)
        alert("Error creating the XMLHttpRequest object.");
    else
        return xmlhttp;
}

//读取服务器上的文件
function process()
{
    //直到 xmlhttp 不为空时才继续执行
```



```
if (xmlHttp)
{
    //尝试连接服务器
    try
    {
        //开始读取服务器上的文件
        xmlHttp.open("GET", "books.xml", true);
        xmlHttp.onreadystatechange = handleRequestStateChange;
        xmlHttp.send(null);
    }
    //如果失败,则显示错误信息
    catch (e)
    {
        alert("Can't connect to server:\n" + e.toString());
    }
}

//当 HTTP 请求的状态变化时,调用函数
function handleRequestStateChange()
{
    //当 readyState 为 4 时,已经成功读取服务器响应
    if (xmlHttp.readyState == 4)
    {
        //直到 HTTP 状态为“OK”时,才继续
        if (xmlHttp.status == 200)
        {
            try
            {
                //处理服务器的响应
                handleServerResponse();
            }
            catch (e)
            {
                //显示错误信息
                alert("Error reading the response: " + e.toString());
            }
        }
        else
        {
```

```
//显示状态信息
alert("There was a problem retrieving the data:\n" +
      xmlhttp.statusText);
}
}

//处理从服务器接收到的响应
function handleServerResponse ()
{
    //读取服务器信息
    var xmlResponse = xmlhttp.responseXML;
    //获取 XML 的 document 元素
    xmlRoot = xmlResponse.documentElement;
    //获取书的 titles 和 ISBN 数组
    titleArray = xmlRoot.getElementsByTagName("title");
    isbnArray = xmlRoot.getElementsByTagName("isbn");
    //生成 HTML 输出
    var html = "";
    //通过得到的数组创建一个 HTML 结构
    for (var i=0; i<titleArray.length; i++)
        html += titleArray.item(i).firstChild.data +
                ", " + isbnArray.item(i).firstChild.data + "<br/>";
    //获取页面中<div>元素的引用
    myDiv = document.getElementById("myDivElement");
    //显示 HTML 输出
    myDiv.innerHTML = "Server says: <br />" + html;
}
```

(5) 打开 <http://localhost/ajax/foundations/xml/books.html>, 如图 2.7 所示。

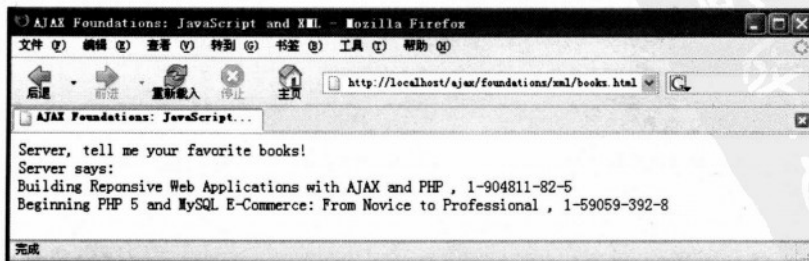


图 2.7 服务器显示对话的内容

## 程序说明

大多数代码都是我们用过的，看起来很熟悉。较新的内容出现在 `handleServerResponse` 函数中。`handleRequestStateChange` 会调用它。

`handleServerResponse` 函数首先找到服务器的 XML 格式的响应。

```
//操作从服务器收到的响应
function handleServerResponse()
{//读取服务器的消息。
    var xmlResponse = xmlHttp.responseXML;
```

`XMLHttpRequest` 对象的 `responseXML` 方法将收到的响应包装成 DOM 文档。如果响应不是一个有效的 XML 文档，浏览器将会抛出错误。每款浏览器对 JavaScript 和 DOM 的实现都有自己的方法，所以错误提示内容取决于浏览器。

后面将解释 XML 代码。假设 XML 文档是有效的，让我们看看如何读取它。众所周知，一个 XML 文档必须有且只有一个 `document` 元素作为它的根元素。在这个例子中就是 `<response>` 元素。就像在练习中做的那样，首先引用 `document` 元素：

```
//获得 XML 的 document 元素
xmlRoot = xmlResponse.documentElement;
```

下一步是建立两个数组，一个存放书的标题 (`title`)，另一个存放书的 ISBN。DOM 函数 `getElementsByTagName()` 函数，解析整个 XML 文档并返回指定的元素内容。

```
//获取书的 titles 和 ISBN 数组
titleArray = xmlRoot.getElementsByTagName("title");
isbnArray = xmlRoot.getElementsByTagName("isbn");
```

读取 XML 文档的一种方法是使用 DOM。更有效的方法是使用 XPath，允许用户在 XML 文档上实现更强大的查询功能。

生成的两个数组是 DOM 元素数组。在例子中，我们打算显示 `title` 元素和 `isbn` 元素的第一个子元素（第一个元素是文本元素，包含要显示的内容）。

```
//生成 HTML 输出
var html = "";
//通过得到的队列创建一个 HTML 结构
for (var i=0; i<titleArray.length; i++)
    html += titleArray.item(i).firstChild.data +
            ", " + isbnArray.item(i).firstChild.data + "<br/>";
//获取页面中<div>元素的引用
myDiv = document.getElementById("myDivElement");
```

```
//显示 HTML 输出
myDiv.innerHTML = "Server says: <br />" + html;
}
```

黑体部分是用来建立 HTML 结构的，用于插入 books.html 页面的 div 元素中。

## 2.6.1 处理更多的错误和抛出异常

如前面谈到的，如果试图读取的 XML 文档无效，那么每一款浏览器都会按自己办法进行处理。下面做一个简单的测试，在 book.xml 中删掉一个封闭标签</response>。Firefox 会向 JavaScript 控制台抛出一个错误，但除此之外不向用户显示任何错误。当然这样做并不是很好，因为没有多少用户浏览网页时会去看 JavaScript 控制台。

打开 Firefox 的 JavaScript 控制台 Tools，选择 JavaScript Console，结果如图 2.8 所示。请参考 <http://ajaxphp.packtpub.com> 中的附录 B，那里提供更多关于 JavaScript Console 的细节介绍以及其他有助调试的优秀工具。

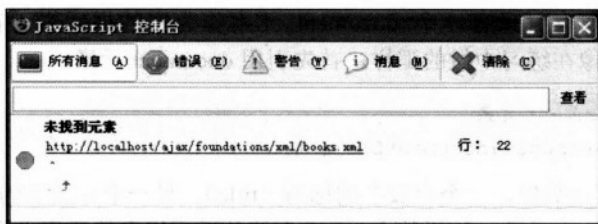


图 2.8 非常有用的 Firefox JavaScript 控制台

但是真正令人不愉快的是，除了 IE（所有版本）外其他的浏览器使用 try/catch 机制时都没有捕获这种错误。就像 Firefox、Mozilla1.7 没有抛出任何错误，更糟的是，甚至没有在 JavaScript 控制台上反映出任何东西，如图 2.9 所示，它忽略掉全部错误，就像什么也没有发生一样（Firefox 浏览器的结果与之相似）。

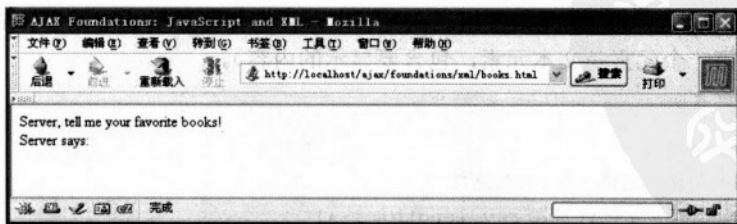


图 2.9 Mozilla 浏览器对出现的问题没有反应

相反，Opera 则非常地友好（如果您是一个开发者）。它完全忽略了用于捕捉错误的 try/catch 块，显示出非常详细的错误信息，如图 2.10 所示。这对开发非常有好处，但开发者

一般不希望访问者看到这些。

在著书时，由于某些原因，IE 好像是惟一使用 catch 块截取异常，并显示出错误信息（虽然没什么帮助）的浏览器，如图 2.11 所示。



图 2.10 Opera 显示了非常详尽的错误信息

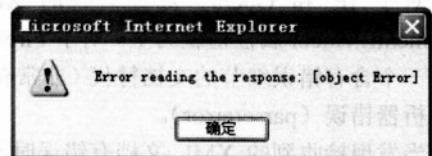


图 2.11 IE 捕获的异常

无论是设计的还是默认的，浏览器在捕获错误方面并不像想象的那样出色。当发生一些无法被正常 try/catch 机制捕获的错误时，找出可选择的解决方案就变得非常重要了。可以对读取 XML 代码的 handleServerResponse 函数做如下修改。

```
//接收服务器端响应处理
function handleServerResponse()
{
    //读取服务器信息
    var xmlResponse = xmlHttp.responseXML;
    //捕获针对 IE 和 Opera 潜在的错误
    if(!xmlResponse || !xmlResponse.documentElement)
        throw("Invalid XML structure:\n"+xmlHttp.responseText);
    //捕获针对 Firefox 的潜在错误
    var rootNodeName = xmlResponse.documentElement.nodeName;
    if(rootNodeName == "parsererror")
        throw("Invalid XML structure:\n"+xmlHttp.responseText);
    //获取 XML 的 document 元素
    xmlRoot = xmlResponse.documentElement;
    //获取书的 titles 和 ISBN 数组
    titleArray = xmlRoot.getElementsByTagName("title");
    isbnArray = xmlRoot.getElementsByTagName("isbn");
}
```

```

//生成 HTML 输出
var html = "";
//通过得到的数组创建一个 HTML 结构
for (var i=0; i<titleArray.length; i++)
    html += titleArray.item(i).firstChild.data +
        ", " + isbnArray.item(i).firstChild.data + "<br/>";
//获取页面中<div>元素的引用
myDiv = document.getElementById("myDivElement");
//显示 HTML 输出
myDiv.innerHTML = "Server says: <br />" + html;
}

```

对于 IE 和 Opera，如果后台的 XML 文档是无效的，那么 xmlResponse 对象的 documentElement 属性值就为空。对于 Firefox 浏览器，XML 文档将是完美有效的，但是文档将被一个含有错误细节的文档替代（非常有意思的报错方式）；这种情况下，文档元素就被称作解析器错误（parsererror）。

当发现接收到的 XML 文档有错误时应抛出一个异常。抛出异常表示产生了一个定制的错误，由 JavaScript 的 throw 关键字完成。这个异常将被 handleServerResponse 的 catch 块捕获，并显示给访问者，如图 2.12 所示。

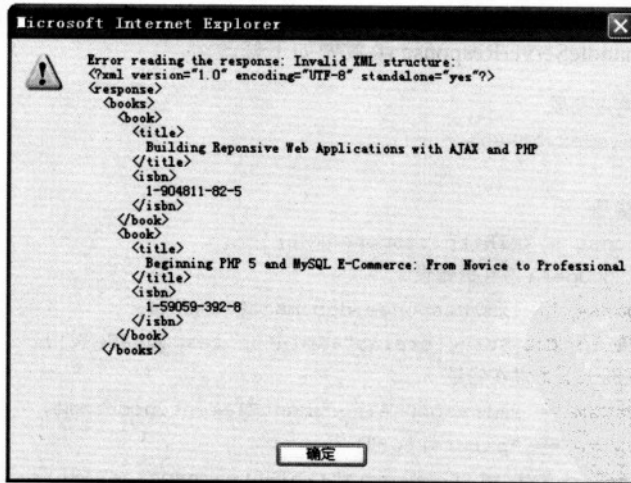


图 2.12 所有测试浏览器都显示相同的错误信息

下面的代码可能会给读者造成迷惑：

```

if(!xmlResponse || !xmlResponse.documentElement )
    throw("Invalid XML Structure:\n"+xmlHttp.responseText);

```

很明显，如果 `xmlResponse` 无效，那么读取它的 `documentElement` 属性时就会有再次产生错误的危险。实际上，JavaScript 只在必要时按着从左往右的顺序计算逻辑表达式。在实例中，如果 `(!xmlResponse)` 为真，第二个表达式根本不会被计算，因为无论如何最终结果也是真。JavaScript 和其他编程语言都支持这一特性，称之为缩短估值 (`short-circuit evaluation`)，在下面的网址可以找到更多关于它的文章 <http://www.webreference.com/javascript/reference/core/expr.html>。

## 2.6.2 建立 XML 结构

XML 和 DOM 无处不在。在本章中使用了 DOM 利用已有的名为 `document` 的 DOM 对象创建 HTML 元素，读者还学会了读取来自服务器的 XML 文档。还有一个没有涉及的重要细节就是使用 JavaScript 的 DOM 创建全新的 XML 文档。如果想要在客户端创建 XML 文档，发送给服务器并在服务器端读取，就需要实现这一功能。

我们不再举更多的例子，但我们将给出缺少的部分。创建一个新的 XML 文档的关键就是创建 XML 文档本身。当向 HTML 输出添加元素时，使用了固有的文档，但在创建新的文档时这样却不行。

当使用 JavaScript 创建一个新的 DOM 对象时，会遇到与创建 `XMLHttpRequest` 对象时一样的问题——创建对象的方法依赖于浏览器。下面是返回新的 DOM 对象实例的通用函数：

```
function creatDomObject
{
    //声明一个存储 DOM 对象的引用
    var xmlDoc
    //创建 XML 文档
    if(document.implementation && document.implementation.createDocument)
    {
        xmlDoc = document.implementation.createDocument("", "", null );
    }
    //针对 IE 的代码
    else if(window.ActiveXObject)
    {
        xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
    }
    //返回创建对象或显示错误信息
    if(!xmlDoc)
    alter("Error creating the DOM object.")
    else
    return xmlDoc
}
```

在执行这个函数后，将返回一个 DOM 对象，然后就可以根据需要进行操作。下面的网址中有关于创建 DOM 对象的更多细节：<http://www.webreference.com/programming/javascript/domwrapper/index.html>。关于如何使用 DOM 对象，请参见本章前面的部分。

## 2.7 小结

本章涉及很多领域的内容。特别是那些刚接触这些技术的读者，对于 HTML、JavaScript、CSS、DOM、XML、XMLHttpRequest 可能会感觉很难掌握。对哪里还觉得没有信心，可以参考文中给出的参考资源。当对上述内容基本掌握后，请进入第 3 章，我们将为读者讲述如何在服务器端使用 PHP 和 MySQL 技术，以及让它们和 Ajax 客户端进行友好交互的方法。





## 使用 PHP 和 MySQL 实现服务器端技术

如果 Ajax 的主要目的是建立更灵巧的客户端，那么与这些客户端对应的服务器也必须同样地灵巧，否则二者之间就有可能会产生冲突。

第 2 章中仅讲了从服务器上读取静态文本或 XML 文档，本章将展开服务器端的一些工作。利用 PHP 产生动态输出，并且使用 MySQL 管理并存储后端数据。本章主要学习以下内容。

- 使用 PHP 执行服务器端的功能；
- 利用参数传递实现客户端与服务器端的通信；
- 在客户端与服务器端使用 XML；
- 使用 PHP 脚本以避免潜在的 JavaScript 安全问题；
- 在执行客户端的重复任务；
- 利用 MySQL 数据库一起工作；
- 优化应用程序的体系结构。

### 3.1 PHP 与 DOM

在第 2 章中从服务器读取数据是异步的，这是一个标准的机制，而且在本节中会多次使用同样的规范，不同的是从服务器返回的数据是静态文件（文本或 XML）。

大多数情况下，需要服务器端进行处理并生成动态结果，本书将使用 PHP 完成服务器端的这部分功能。如果本书的读者并不熟悉 PHP，可以利用“PHP 指南 (php tutorial)”作为关键字进行在线搜索，可以找到包括 PHP 的官方网址 <http://php.net/tut.php> 在内的很多相关资源。如果习惯于在工作中学习新知识，可以看 Cristian Darie 与 Mihai Bucica 的一些电子书，如 *Beginning PHP 5 and MySQL E-Commerce: From Novice to Professional*。

甚至可以使用第 6 章的“Suggest and Autocomplete”应用程序（该程序的网址为：<http://ajaxphp.packtpub.com/ajax/suggest/>），用它可以找到 PHP 的帮助页面。

本章的第一个练习是使用 PHP 的 DOM 控件编写一个 PHP 脚本来创建 XML 输出，这个输出能被客户端读取。PHP 的 DOM 功能类似于 Java 的 DOM 功能，其官方文件在 <http://www.php.net/>

manual/en/ref.dom.php。

将在服务器上创建的 XML 文档几乎与第 2 章中作为 XML 文件存储的 XML 文档一样，但这次它是动态生成的：

```
<response>
  <books>
    <book>
      <title>Building Reponsive web Applications with AJAX and PHP</title>
      <isbn>1-904811-82-5</isbn>
    </book>
  </books>
</response>
```

## 》》》 实现步骤——用 PHP 实现 Ajax

- (1) 在根目录下创建一个名为 php 的子目录。
- (2) 在 php 目录下建立一个名为 phptest.html 的文件，其内容如下：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
  <head>
    <title>Practical Ajax: Using the PHP DOM</title>
    <script type="text/javascript" src="phptest.js"></script>
  </head>
  <body onload="process()">
    The Ajax book of 2006 is:
    <br />
    <div id="myDivElement" />
  </body>
</html>
```

(3) 客户端代码 phptest.js 与第 2 章 XML 练习中的 books.js 几乎完全一样，不同的部分用黑体标注。

```
//保留一个 XMLHttpRequest 请求
var xmlhttp = createXmlHttpRequestObject();

//创建一个 XMLHttpRequest 请求
function createXmlHttpRequestObject()
```

```
{
    //为 XMLHttpRequest 对象存储索引
    var xmlHttp;
    //用于除了 IE6 及其更早期的版本以外的所有浏览器
    try
    {
        //试图创建一个 XMLHttpRequest 对象
        xmlHttp = new XMLHttpRequest();
    }
    catch(e)
    {
        //用于 IE6 及其早期版本
        var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
                                          "MSXML2.XMLHTTP.5.0",
                                          "MSXML2.XMLHTTP.4.0",
                                          "MSXML2.XMLHTTP.3.0",
                                          "MSXML2.XMLHTTP",
                                          "Microsoft.XMLHTTP");

        //尝试每个程序直到其中一个能够运行
        for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
        {
            try
            {
                //创建 XMLHttpRequest 对象
                xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
            }
            catch (e) {}
        }
    }
    //返回被创建的对象或显示一个出错信息
    if (!xmlHttp)
        alert("Error creating the XMLHttpRequest object.");
    else
        return xmlHttp;
}
//从服务器读取一个文件
function process()
{
    //仅在 xmlHttp 不为空时继续程序
    if (xmlHttp)
```

```
{
    //尝试连接服务器
    try
    {
        //从服务器读取文件启动
        xmlHttp.open("GET", "phptest.php", true);
        xmlHttp.onreadystatechange = handleRequestStateChange;
        xmlHttp.send(null);
    }
    //失败时显示出错信息
    catch (e)
    {
        alert("Can't connect to server:\n" + e.toString());
    }
}
//当 HTTP 状态请求发生改变时调用函数
function handleRequestStateChange()
{
    //当 readyState 为 4 时, 准备接收服务器响应
    if (xmlHttp.readyState == 4)
    {
        //仅在 HTTP 状态为"OK"时继续程序
        if (xmlHttp.status == 200)
        {
            try
            {
                //根据服务器的响应进行处理
                handleServerResponse();
            }
            catch(e)
            {
                //显示出错信息
                alert("Error reading the response: " + e.toString());
            }
        }
        else
        {
            //显示状态信息
            alert("There was a problem retrieving the data:\n" +
```

```

        xmlhttp.statusText);
    }
}
}
//处理来自于服务器的响应
function handleServerResponse()
{
    //读取来自服务器的信息
    var xmlResponse = xmlhttp.responseXML;
    //在 IE 或 Opera 下抛出可能的错误
    if (!xmlResponse || !xmlResponse.documentElement)
        throw("Invalid XML structure:\n" + xmlhttp.responseText);
    //在 Firefox 下抛出可能的错误
    var rootNodeName = xmlResponse.documentElement.nodeName;
    if (rootNodeName == "parsererror") throw("Invalid XML structure");
    //获得 XML 文件元素
    xmlRoot = xmlResponse.documentElement;
    //获得 book 标题与 ISBN 数组
    titleArray = xmlRoot.getElementsByTagName("title");
    isbnArray = xmlRoot.getElementsByTagName("isbn");
    //生成 HTML 输出
    var html = "";
    //利用数组重作并创建一个 HTML 结构
    for (var i=0; i<titleArray.length; i++)
        html += titleArray.item(i).firstChild.data +
            ", " + isbnArray.item(i).firstChild.data + "<br/>";
    //获得页面上<div>元素的参考
    myDiv = document.getElementById("myDivElement");
    //显示 HTML 输出
    myDiv.innerHTML = html;
}

```

(4) 最后, phptest.php 文件如下:

```

<?php
//设置输出内容的类型为 xml
header('Content-Type: text/xml');
//创建新的 XML 文件
$dom = new DOMDocument();

//建立根<response>元素

```

```
$response = $dom->createElement('response');
$dom->appendChild($response);

//建立<books> 元素并将其作为<response>的子元素
$books = $dom->createElement('books');
$response->appendChild($books);

//为 book 创建标题元素
$title = $dom->createElement('title');
$titleText = $dom->createTextNode
    ('Building Reponsive Web Applications with Ajax and PHP');
$title->appendChild($titleText);

//为 book 创建 isbn 元素
$isbn = $dom->createElement('isbn');
$isbnText = $dom->createTextNode('1-904811-82-5');
$isbn->appendChild($isbnText);

//创建<book>元素
$book = $dom->createElement('book');
$book->appendChild($title);
$book->appendChild($isbn);

//将<book>作为<books>的子元素
$books->appendChild($book);

//在一个字符串变量中建立 XML 结构
$xmlString = $dom->saveXML();
//输出 XML 字符串
echo $xmlString;
?>
```

(5) 首先通过一个简单的测试观察 `phptest.php` 所返回的内容，在浏览器地址栏中输入 `http://localhost/ajax/foundations/php/phptest.php`，确认产生一个已构造好的 XML 结构。

**注意** 如果没有产生预期的结果，则不仅要检查代码，还要查看 PHP 的安装，关于如何正确安装这方面的细节问题可以查看附录 A。

(6) 当服务器返回正确的响应后，可以在浏览器的地址栏中输入 `http://localhost/ajax/foundations/php/phptest.html`，如图 3.2 所示。

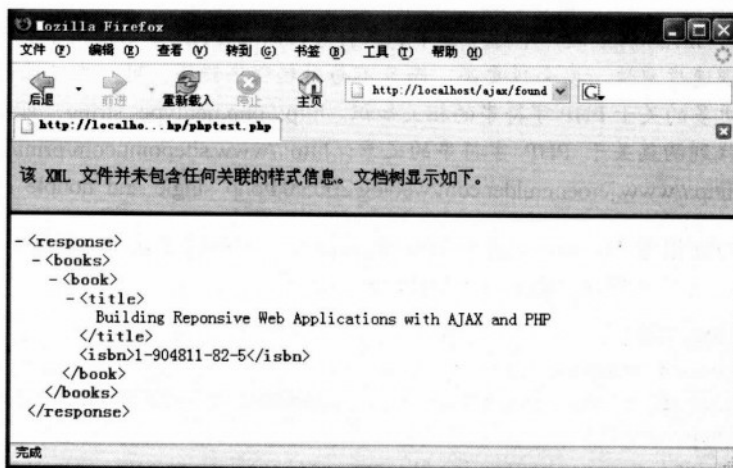


图 3.1 PHP 生成的简单 XML 结构

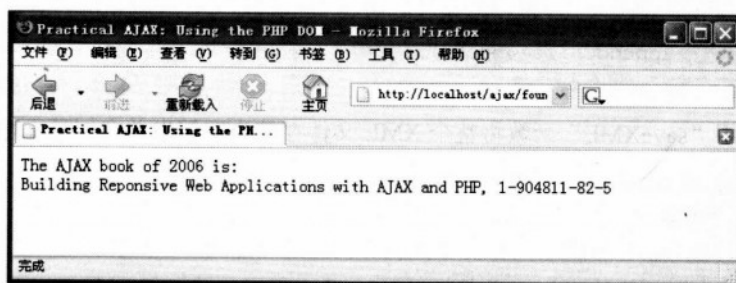


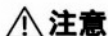
图 3.2 使用 PHP 的 Ajax

## 程序说明

当要产生一个 XML 结构时，无论是在客户端还是在服务器端，都必须选择是利用 DOM 还是用连接字符串创建 XML 文件。前例中的 PHP 脚本 `phptest.php` 从向 `Text/XML` 设置输出内容开始。

```
<?php
//设置输出内容的类型为 xml
header('content-type:text/xml');
```

关于 `header` 的 PHP 文件可以在 <http://www.php.net/manual/en/function.header.php> 中查阅（可以通过在 Suggestion 应用中搜索“header”找到帮助页面）。

**注意**

通常 JavaScript 文件中对字符串加双引号，而在 PHP 中则使用单引号。这样能使处理速度更快、安全性更高，而且不易引起程序错误。可以在下面的网址中学习更多的关于 PHP 字符串的相关知识：<http://php.net/types.string>。可以在下面的网址找到两篇关于 PHP 字符串的文章：<http://www.sitepoint.com/print/quick-php-tips> 与 [http://www.jeroenmulder.com/weblog/2005/04/php\\_single\\_and\\_double\\_quotes.php](http://www.jeroenmulder.com/weblog/2005/04/php_single_and_double_quotes.php)。

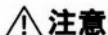
PHP 的 DOM 很像 JavaScript 的 DOM，但并没有太多不同之处，它们都是从创建 DOM 对象文件开始，只不过 PHP 中使用 DOMDocument 类表示。

```
//创建新的 XML 文件
$xml = new DOMDocument();
然后，用 createElement、createTextNode、appendChild 等方法继续建立 XML 文件。
//建立根<response>元素
$response = $xml->createElement('response');
$xml->appendChild($response);
//建立<books> 元素并将其作为<response>的子元素
$books = $xml->createElement('books');
$response->appendChild($books);
...
```

最后，使用“saveXML”函数将整个 XML 文件存储为字符串并在屏幕上输出。

```
$xmlString = $xml->saveXML();
//输出 XML 字符串
echo $xmlString;
?>
```

这样这个 XML 文件就会用第 2 章中所使用的方法在客户端读取和显示。

**注意**

大多数情况下，XML 文档将在服务器端生成，在客户端读取，不过也可以用其他方式实现这个目的。第 2 章中曾介绍过如何使用 JavaScript 的 DOM 创建 XML 文件和元素，就像在下面的练习中一样，可以用 GET 或 POST 方法将这些结构直接转移到 PHP 中。同样可以使用 DOM 从 PHP 中读取 XML 文件，或者使用一种名叫 SimpleXML 的更容易使用的 API，第 9 章中将使用这种方法建立 RSS 阅读器。

## 3.2 参数传递与 PHP 错误处理

前面的 PHP 练习中忽略了编写 PHP 脚本时两个非常普遍的问题：



- 经常需要将参数传递到服务器端的 (PHP) 脚本。
- 既然客户端有很好的保护措施, 服务器端也应该实现错误处理机制。

可以用 GET 或 POST 方法将参数传递给 PHP 脚本, 使用一种 PHP 专用技术处理 PHP 错误。在下面的练习中, 把参数传递给一个 PHP 脚本, 并且实现一种错误处理机制, 然后仿造一些值来进行测试。该应用程序的运行情况如图 3.3 所示。

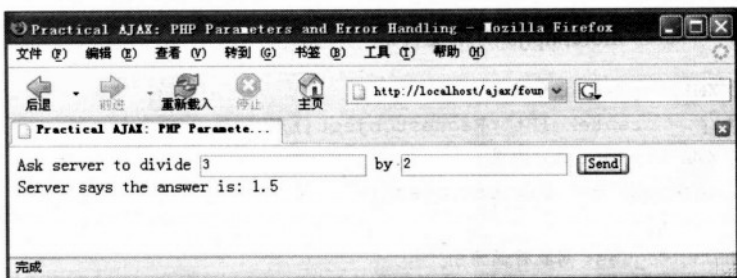


图 3.3 PHP 参数传递与错误处理

这部分内容将对服务器进行异步访问, 请求服务器完成两个数字的除法运算, 当服务器正常运行时, 将用类似如下所示的 XML 文件返回结果:

```
<?xml version="1.0"?>
<response>1.5</response>
```

下面的 PHP 出错实例中, 服务器脚本不会生成一个 XML 串, 而是返回被客户端截取的错误信息, 错误信息是普通的文本格式文本 (做完练习后自然会明白其原因)。

## 》》》 实现步骤——PHP 参数传递与错误处理

- (1) 在根目录下创建一个名为 morephp 的新目录。
- (2) 在 morephp 目录下建立一个名为 morephp.html 的新文件。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
  <head>
    <title>Practical Ajax: PHP Parameters and Error Handling</title>
    <script type="text/javascript" src="morephp.js"></script>
  </head>
  <body>
    Ask server to divide
    <input type="text" id="firstNumber" />
```

```
by
<input type="text" id="secondNumber" />
<input type="button" value="Send" onclick="process()" />
<div id="myDivElement" />
</body>
</html>
```

(3) 建立一个名为 `morephp.js` 的新文件。

```
//保留一个XMLHttpRequest 请求
var xmlHttp = createXmlHttpRequestObject();
//创建一个XMLHttpRequest 请求
function createXmlHttpRequestObject()
{
    //为XMLHttpRequest 对象存储索引
    var xmlHttp;
    //用于除了IE6及其更早期的版本以外的所有浏览器
    try
    {
        //试图创建一个XMLHttpRequest 对象
        xmlHttp = new XMLHttpRequest();
    }
    catch(e)
    {
        //用于IE6及其早期版本
        var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
            "MSXML2.XMLHTTP.5.0",
            "MSXML2.XMLHTTP.4.0",
            "MSXML2.XMLHTTP.3.0",
            "MSXML2.XMLHTTP",
            "Microsoft.XMLHTTP");

        //尝试每个程序直到其中一个能够运行
        for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
        {
            try
            {
                //创建XMLHttpRequest 对象
                xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
            }
            catch (e) {}
        }
    }
}
```

```
}
//返回被创建的对象或显示一个出错信息
if (!xmlHttp)
    alert("Error creating the XMLHttpRequest object.");
else
    return xmlHttp;
}
//从服务器读取一个文件
function process()
{
    //仅在 xmlHttp 不为空时继续程序
    if (xmlHttp)
    {
        //尝试连接服务器
        try
        {
            //得到由用户输入的两个值
            var firstNumber = document.getElementById("firstNumber").value;
            var secondNumber = document.getElementById("secondNumber").value;
            //建立参数字符串
            var params = "firstNumber=" + firstNumber +
                "&secondNumber=" + secondNumber;
            //启动异步 HTTP 请求
            xmlHttp.open("GET", "morephp.php?" + params, true);
            xmlHttp.onreadystatechange = handleRequestStateChange;
            xmlHttp.send(null);
        }
        //失败时显示出错信息
        catch (e)
        {
            alert("Can't connect to server:\n" + e.toString());
        }
    }
}
//当 HTTP 请求状态改变时调用函数
function handleRequestStateChange()
{
    //当 readyState 为 4 时, 准备接收服务器响应
    if (xmlHttp.readyState == 4)
    {
```

```
//仅在 HTTP 状态为"OK"时程序继续
if (xmlHttp.status == 200)
{
    try
    {
        //根据服务器的响应进行处理
        handleServerResponse();
    }
    catch(e)
    {
        //显示出错信息
        alert("Error reading the response: " + e.toString());
    }
}
else
{
    //显示状态信息
    alert("There was a problem retrieving the data:\n" +
        xmlHttp.statusText);
}
}
}
//处理来自于服务器的响应
function handleServerResponse()
{
    //将服务器响应包恢复为一个 XML DOM 对象
    var xmlResponse = xmlHttp.responseXML;
    //在 IE 或 Opera 下抛出可能的错误
    if (!xmlResponse || !xmlResponse.documentElement)
        throw("Invalid XML structure:\n" + xmlHttp.responseText);
    //在 Firefox 下抛出可能的错误
    var rootNodeName = xmlResponse.documentElement.nodeName;
    if (rootNodeName == "parsererror")
        throw("Invalid XML structure:\n" + xmlHttp.responseText);
    //获得根元素(文件元素)
    xmlRoot = xmlResponse.documentElement;
    //测试期望收到的 XML 文件
    if (rootNodeName != "response" || !xmlRoot.firstChild)
        throw("Invalid XML structure:\n" + xmlHttp.responseText);
    //需要显示的值为根<response>的子元素
```

```

responseText = xmlRoot.firstChild.data;
//显示用户信息
myDiv = document.getElementById("myDivElement");
myDiv.innerHTML = "Server says the answer is: " + responseText;
}

```

(4) 创建一个名为 morephp.php 的文件:

```

<?php
//载入出错处理模块
require_once('error_handler.php');
//指定输出为一个 XML 文件
header('Content-Type: text/xml');
//计算结果
$firstNumber = $_GET['firstNumber'];
$secondNumber = $_GET['secondNumber'];
$result = $firstNumber / $secondNumber;
//建立一个新的 XML 文件
$dom = new DOMDocument();
//建立根<response>元素并将其加入文件
$response = $dom->createElement('response');
$dom->appendChild($response);
//以文本模式将计算出的平方值加入<response>的子元素
$responseText = $dom->createTextNode($result);
$response->appendChild($responseText);
//在一个字符串变量中建立 XML 结构
$xmlString = $dom->saveXML();
//输出 XML 字符串
echo $xmlString;
?>

```

(5) 最后, 建立错误处理文件 error\_handler.php:

```

<?php
//设置用户出错处理方法为 error_handler
set_error_handler('error_handler', E_ALL);
//出错处理函数
function error_handler($errNo, $errStr, $errFile, $errLine)
{
    //清除已生成的所有输出
    if(ob_get_length()) ob_clean();
    //输出出错信息

```

```

$error_message = 'ERRNO: ' . $errNo . chr(10) .
                 'TEXT: ' . $errStr . chr(10) .
                 'LOCATION: ' . $errFile .
                 ', line ' . $errLine;

echo $error_message;
//阻止处理任何更多的 PHP 脚本
exit;
}
?>

```

(6) 在浏览器中载入 <http://localhost/ajax/foundations/morephp/morephp.html> 并运行, 结果如图 3.3 所示。

## 程序说明

现在读者应该熟悉了客户端的几乎所有代码, 因此下面主要关注服务器端, 这里有两个文件——`morephp.php` 与 `error_handler.php`。

`morephp.php` 文件用来输出保存除法结果的 XML 文件, 然而, 它却是由载入错误处理程序开始的, 该程序用来捕获各种错误, 给出一个比默认情况更友好的出错信息, 并将其发送回客户端。

```

<?php
//载入出错处理模块
Require_once('error_handler.php');

```

### 注意

PHP5 像其他面向对象语言一样支持异常, 但是使用 PHP5 将受到只能使用自身抛出和捕获的异常对象的限制, 而且在构建大的体系结构时这些对象有助于改进代码。PHP 的核心在发生问题时不产生异常。可能是由于向后兼容的问题, 当发生问题时, PHP5 生成错误, 而不是抛出异常。这是更原始的解决运行时问题的方式。例如, 我们无法像处理异常那样捕获一个错误在本地处理完, 然后让脚本继续正常运行, 处理错误的最好办法是定义一个函数来自动, 当脚本失效后调用这个函数并且提供最后一次机会最后做一些处理, 如记录错误、关闭数据库连接或者“友好地”向站点的访问者提供一些信息。

前面的代码中, `error_handler.php` 脚本就是用来处理错误的。接收到错误后, 它会将默认的出错信息转换为更友好的出错信息。然而需要注意的是, `error_handler.php` 能够捕获大多数的错误, 但并不是全部! PHP 代码不能捕获致命错误并产生不受程序控制的输出。例如, 当忘记在变量名字前加一个“\$”字符时产生的解析错误, 在 PHP 代码执行前就会被截取, 因此它不能被 PHP 代码所捕获, 但它们会被记录在 Apache 错误日志文件中。



**注意** 当 PHP 脚本运行异常时，时刻注意 Apache 错误日志是很重要的，它可以帮助我们许多令人头疼的事情中解脱出来。该文件的默认位置与名字是：Apache2\logs\error.log。

在设置了错误处理程序之后，下面继续设置 XML 的目录类型，并用前面程序所接收到的第二个数字去除第一个数字。注意，\$\_GET 方法读取使用 GET 方法发送的变量，而如果用 POST 方法发送变量则应使用\$\_POST 方法读取。另外，可以使用\$\_REQUEST 得到以任何方式（包括 cookies）发送的变量。不过由于这种方法相对其他方法来说要慢一些，因此通常推荐尽量避免使用这个方法。

```
//指定输出为一个 XML 文件
Header('Content-Type: text/xml');
//计算结果
$firstNumber = $_GET['firstNumber'];
$secondNumber = $_GET['secondNumber'];
$result = $firstNumber / $secondNumber;
```

如果\$secondNumber 为 0，除法操作就会产生一个错误。本例中，希望错误处理脚本截取这个错误。注意在实际问题中，更专业的做法是在除法计算前检查变量值，但本例中，我们的主要目的是检验错误脚本。

求出值之后，和前面的例子一样，也要将其打包为一个新的 XML 文件并输出它的值。

```
//建立一个新的 XML 文件
$dom = new DOMDocument();
//建立根<response>元素并将其加入文件
$response = $dom->createElement('response');
$dom->appendChild($response);
//以文本模式将计算出的平方值加入<response>的子元素
$responseText = $dom->createTextNode($result);
$response->appendChild($responseText);
//在一个字符串变量中建立 XML 结构
$xmlString = $dom->saveXML();
//输出 XML 字符串
echo $xmlString;
?>
```

现在不妨看一下错误处理脚本 error\_handler.php，这个文件的任务是截取 PHP 所产生的任何错误信息，同时输出一个有意义的并能被 JavaScript 代码所显示的出错信息（如图 3.4 所示）。

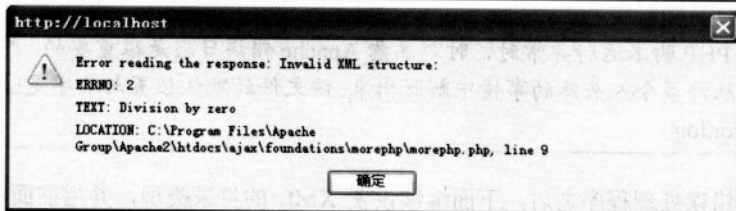


图 3.4 友好的出错信息

如果没有自定义的出错处理，所收到的错误信息将如图 3.5 所示。

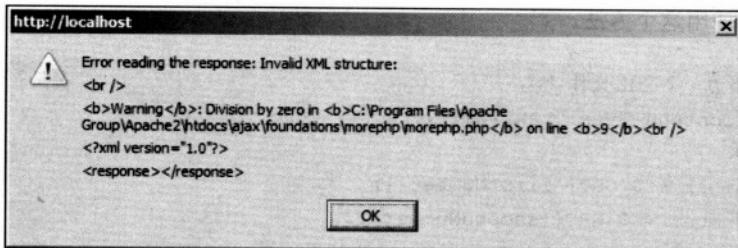


图 3.5 不友好的出错信息

### ⚠ 注意

如果 `php.ini` 中的 `display_errors` 设置为 `on`，出错信息将如图 3.5 所示。默认情况下，该操作被取消并被 Apache 错误日志所记录。在编写代码时，这个操作对显示错误是有用的。如果这段代码为产品代码，那么这两种错误信息就不恰当了，不要把调试信息显示给终端用户。

在 `error_handler.php` 中到底都有哪些内容呢？首先，该文件使用 `set_error_handler` 函数建立了一个新的出错处理函数。

```
<?php
//设置用户出错处理方法为 error_handler
set_error_handler('error_handler', E_ALL);
```

当发生错误时，首先调用 `ob_clean()` 删除任何如图 3.5 中 `<response></response>` 字节所示的已产生的输出。

```
//出错处理函数
function error_handler($errNo, $errStr, $errFile, $errLine)
{
    //清除已生成的所有输出
    if(ob_get_length()) ob_clean();
```

当然，如果在一些特定的调试中需要保留这些字节，可以对 `ob_clean` 调用进行注释。实



实际的出错信息是由一系列的变量构成的——\$errNo、\$errStr、\$errFile 以及 \$errLine，并用 chr 函数生成回车。

```
//输出出错信息
$error_message = 'ERRNO: ' . $errNo . chr(10) .
                 'TEXT: ' . $errStr . chr(10) .
                 'LOCATION: ' . $errFile .
                 ', line ' . $errLine;

echo $error_message;
//阻止处理任何更多的 PHP 脚本
exit;
}
?>
```

**注意** 前面的出错处理方案实际上是一种十分简化的方法，仅仅适合在书写、调试代码时使用。而作为一种成熟的解决方案，要求显示给终端用户的应该是不包含任何技术细节的友好信息。如果希望将出错细节打包为一个供客户端读取的 XML 文件，一定要记住，在上面的函数中不会处理解析错误与致命错误，这些错误将在 PHP 配置文件（php.ini）的设置中表现出来。

前面的例子也给出了这样一种可能：用户能尝试同时对服务器发出多个请求（可以通过快速单击“Send”按钮来实现）。而当试图对一个繁忙的 XMLHttpRequest 对象发出请求时，其“open”方法就会产生异常，利用 try/catch 结构可以使代码得到很好的保护，不过这时图 3.6 所示的错误信息看起来不是很友好。

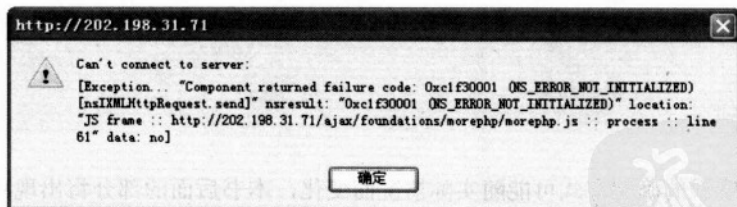


图 3.6 对一个繁忙的 XMLHttpRequest 发出请求

这条信息也许恰好是我们所需要的，不过在一些特定的环境下，我们可能更需要反映这类错误与其他错误的不同之处。例如，在网站的建设过程中，我们可能需要在页面上显示一个说明，或给出一个友好的类似“please try again later”的信息，这可以通过修改下面代码段 process() 函数实现：

```
//从服务器读取一个文件
function process()
```

```
{
    //仅在 xmlhttp 不为空时继续程序
    if (!xmlhttp) return;
    //如果 XMLHttpRequest 忙则不再向服务器发送请求
    if!(xmlhttp.readyState==0||xmlhttp.readyState==4)
        alert("Can't connect to server, please try again later.");
    else
    {
        //尝试连接服务器
        try
        {
            //得到由用户输入的两个值
            var firstNumber = document.getElementById("firstNumber").value;
            var secondNumber = document.getElementById("secondNumber").value;
            //建立参数字符串
            var params = "firstNumber=" + firstNumber +
                "&secondNumber=" + secondNumber;
            //启动异步 HTTP 请求
            xmlhttp.open("GET", "morephp.php?" + params, true);
            xmlhttp.onreadystatechange = handleRequestStateChange;
            xmlhttp.send(null);
        }
        //失败时显示出错信息
        catch (e)
        {
            alert("Can't connect to server:\n" + e.toString());
        }
    }
}
```

处理这些错误的确切方式可能随实际情况而变化，本书后面的部分将出现更多有用的解决方案：

- 有时可能更希望简单地将错误忽略掉；
- 其他时候需要用前面给出的代码来显示一个自定义的错误信息。

大多数情况下我们首先要尽力避免产生错误——阻止一个问题的发生总要好过等问题发生后再去处理。例如，如果试图用一个 XMLHttpRequest 对象向服务器发出请求，而该对象正忙于处理一个更早的请求时，一般就会出现“连接忙（connection busy）”类型的错误，而这类错误通常有很多方法可以避免。

- 可以为每个需要发送到服务器的信息打开一个新的连接（如创建一个新的 XMLHttpRequest

对象), 这种方式便于执行并且可能在许多情况下都是有效的。但通常还是要避免使用这种方法, 因为它可能会影响到服务器的性能, 比如在服务器可能还没有完成以前的请求时, 程序仍不断地打开连接并发出新的请求。同时这种方法也不能保证按与发出请求同样的次序收到响应——特别是在服务器繁忙或网速慢时。

- 可以用一个数组记录信息并在连接可用以后将其发送出去, 在本书的许多实例中都可以看到这种方法, 如 Ajax 表单验证以及 Ajax 聊天。

- 用下面的方法则可以完全忽视该信息: 不再尝试对同一连接重复发出请求而使用前面的出错处理代码。

### 3.3 连接远程服务器与 JavaScript 的安全性

前面的 PHP 练习能够平稳地运行结束, 不必对此感到惊讶, 这是因为异步调用的服务器 (PHP) 脚本与载入 HTML 文件的服务器是同一台机器。

Web 浏览器使用十分严格 (也是不同) 的方法去控制用 JavaScript 代码所访问的资源, 如果想通过 JavaScript 代码访问另一台服务器, 事情会变得很麻烦, 这也恰恰是后面的练习中我们所要做的。不过在此之前, 先介绍一些理论。

JavaScript 代码运行在其父 HTML 文件的安全特权下。在默认的情况下, 当从服务器载入一个 HTML 页面时, 这个 HTML 页中的 JavaScript 代码将仅允许向该服务器发送 HTTP 请求。任何其他服务器都是潜在的敌人, 并且不幸的是, 不同的浏览器对应的处理方式是不同的。

Internet Explorer 是一种友好的 Web 浏览器, 这就意味着其安全性更低, 但功能更丰富。IE 有一种基于“区域”的安全模式, 区域分为 4 种: Internet、本地 Intranet、受信任的站点以及受限制的站点。不同的区域有不同的安全设置, 可以按以下的路径进行更改: 工具栏/Internet 选项/安全。当访问 Web 资源时, 该 Web 资源将被自动指向一个安全区域, 并应用特定的安全操作。

默认安全操作可以随着系统的不同而变化。在默认的情况下, 对从本地载入的文件资源的脚本 (不是通过 Web 服务器, 更不是本地 Web 服务器), IE 将给予全部的权限。因此如果要载入 C:\ajax\..., 脚本将平稳运行 (在运行前会发出警告, 所载入的脚本具有完全的权限)。而如果通过 HTTP (比如 <http://localhost/ajax/.../ping.html>) 载入 JavaScript 代码, 并且该段 JavaScript 代码试图向另一台服务器发出请求, IE 将弹出一个对话框以确认用户是否为该行给予许可。

Firefox 以及其他基于 Mozilla 的浏览器具有一种基于“特权”的更受限也更复杂的安全机制。这类浏览器不会自动弹出确认窗口。反之, JavaScript 代码必须使用一个 Mozilla 专用的 API 来查询所需要的行为。如果幸运的话, 浏览器会向用户弹出一个要求用户进行输入的对话框, 根据用户的输入情况对 JavaScript 代码授权 (或反之)。如果不走运, 这类浏览器将

完全忽视代码请求。默认情况下基于 Mozilla 的浏览器监听来自于本地 (file:///) 资源的特权请求, 并将完全忽视通过 HTTP 所载入的脚本的请求——除非这些脚本是被标记的 (尽管这些是可以手工更改的默认设置)。从下面的网址可以找到更多的关于脚本标记的内容: <http://www.mozilla.org/projects/security/components/signed-scripts.html>。

下面的练习将创建一个从在线服务网站 <http://www.random.org> 读随机数的 JavaScript 程序, 这个网站提供了一个能产生真正的随机数的服务。由 <http://www.random.org/http.html> 所载入的页面阐明了如何通过 HTTP 访问该服务器。基于这点编写程序时, 需要查看下面的网址中所给出的指导: <http://www.random.org/guidelines.html>。最后, 来感受一下究竟生成什么样的随机数。在浏览器的地址栏中输入: <http://www.random.org/cgi-bin/randnum>。需要说明的是, 当调用没有选项时, 默认在 1 和 100 之间产生 100 个随机数。客户端每次请求一个 1~100 之间的随机数, 方法是向 <http://www.random.org/cgi-bin/randnum?num=1&min=1&max=100> 发送请求, 如图 3.7 所示。

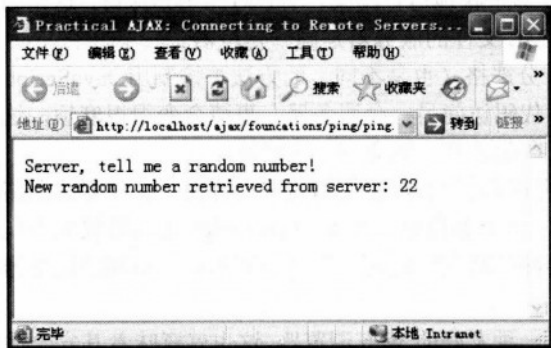


图 3.7 连接到远程服务器

## 》》》 实现步骤——连接远程服务器

- (1) 在根目录下创建一个名为 ping 的新目录。
- (2) 在 ping 目录下建立一个名为 ping.html 的新文件, 文件内容如下:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
  <head>
    <title>Practical Ajax: Connecting to Remote Servers</title>
    <script type="text/javascript" src="ping.js"></script>
  </head>
  <body onload="process()">
```

```

    Server, tell me a random number!<br/>
    <div id="myDivElement" />
  </body>
</html>

```

(3) 建立一个名为 ping.js 的新文件，代码如下：

```

//保留一个 XMLHttpRequest 请求
var xmlHttp = createXmlHttpRequestObject();
//保留远程服务器地址和参数
var serverAddress = "http://www.random.org/cgi-bin/randnum";
var serverParams = "num=1" + //how many random numbers to generate
                  "&min=1" + //the min number to generate
                  "&max=100"; //the max number to generate

//创建一个 XMLHttpRequest 请求
function createXmlHttpRequestObject()
{
  //为 XMLHttpRequest 对象存储索引
  var xmlHttp;
  //用于除了 IE6 及其更早期的版本以外的所有浏览器
  try
  {
    //试图创建一个 XMLHttpRequest 对象
    xmlHttp = new XMLHttpRequest();
  }
  catch(e)
  {
    //用于 IE6 及其早期版本
    var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
                                     "MSXML2.XMLHTTP.5.0",
                                     "MSXML2.XMLHTTP.4.0",
                                     "MSXML2.XMLHTTP.3.0",
                                     "MSXML2.XMLHTTP",
                                     "Microsoft.XMLHTTP");

    //尝试每个程序直到其中一个能够运行
    for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
    {
      try
      {
        //创建 XMLHttpRequest 对象

```

```
        xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
    }
    catch (e) {}
}
}
//返回被创建的对象或显示一个出错信息
if (!xmlHttp)
    alert("Error creating the XMLHttpRequest object.");
else
    return xmlHttp;
}
//异步调用服务器
function process()
{
    //仅在 xmlHttp 不为空时继续程序
    if (xmlHttp)
    {
        //尝试连接服务器
        try
        {
            //为基于 Mozilla 的浏览器请求呼叫远程服务器的许可
            try
            {
                //如果浏览器不是规定的则生成一个被忽略的错误
                //Mozilla
                netscape.security.PrivilegeManager.enablePrivilege('UniversalBrowserRead');
            }
            catch(e) {} //忽略错误
            //启动服务器访问
            xmlHttp.open("GET", serverAddress + "?" + serverParams, true);
            xmlHttp.onreadystatechange = handleRequestStateChange;
            xmlHttp.send(null);
        }
        //失败时显示出错信息
        catch (e)
        {
            alert("Can't connect to server:\n" + e.toString());
        }
    }
}
```

```
//当 HTTP 请求状态改变时调用函数
function handleRequestStateChange()
{
    //当 readyState 为 4 时, 准备接收服务器响应
    if (xmlHttp.readyState == 4)
    {
        //仅在 HTTP 状态为 "OK" 时程序继续
        if (xmlHttp.status == 200)
        {
            try
            {
                //根据服务器的响应进行处理
                handleServerResponse();
            }
            catch(e)
            {
                //显示出错信息
                alert("Error reading the response: " + e.toString());
            }
        }
        else
        {
            //显示状态信息
            alert("There was a problem retrieving the data:\n" +
                xmlHttp.statusText);
        }
    }
}

//处理来自于服务器的响应
function handleServerResponse()
{
    //重新接收服务器的响应
    var response = xmlHttp.responseText;
    //获得页面上<div>元素的索引
    myDiv = document.getElementById('myDivElement');
    //显示 HTML 输出
    myDiv.innerHTML = "New random number retrieved from server: "
        + response + "<br/>";
}

```

(4) 在浏览器中载入 `http://localhost/ajax/foundations/ping/ping.html`。如果是默认配置的 IE 浏览器，则运行结果如图 3.8 所示，浏览器将询问是否允许脚本连接远程服务器。而对于默认配置的 Firefox 或 Opera 来说，将分别显示如图 3.9 与图 3.10 所示的安全出错信息。

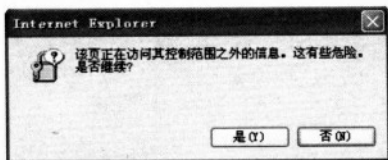


图 3.8 IE 浏览器请求许可



图 3.9 Firefox 浏览器拒绝访问

(5) 现在试着载入直接来自于文件系统的同一个 HTML 文件，其路径类似于 `file:///c:/Apache2/htdocs/ajax/foundations/ping/ping.html`。在默认配置下，IE 运行没有问题，这是因为该页面属于受信任区域。Firefox 则弹出一个如图 3.11 所示的确认信息框，而 Opera 将显示与图 3.10 完全相同的错误信息。



图 3.10 Opera 浏览器拒绝访问

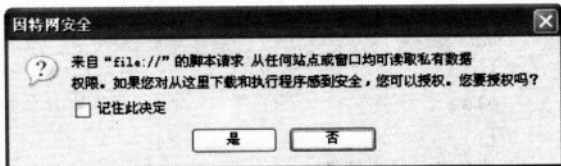


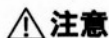
图 3.11 Firefox 浏览器请求许可

## 程序说明

Opera 实际上是当前最安全的浏览器，没有任何办法能让 Opera8.5 允许 JavaScript 代码访问与其所载入服务器不同的服务器。

IE 按照区域设置的指示运行，默认情况下，IE 通过给予本地文件最高的信任级别而使其使用起来非常轻松，但当来自于因特网的脚本有一定的危害性时，其运行要经过询问确认。

对于 Firefox 来说，当您希望做些事情时必须向其询问，问题是在默认情况下，除非脚本被标记或是从本地载入，否则它不会接受您的任何请求。然而，大多数情况下，要求站点的访问者改变浏览器的设置是不现实的。



**注意**

包括来自于未标记的脚本在内，有一种办法可以让 Firefox 执行所有的请求，只要在地址栏输入 `about:config`，并将 `signed.applets.codebase_principal_support` 选项的值改为“true”即可。



下面是请求 Firefox 允许远程服务器访问的代码:

```
//为基于 Mozilla 的浏览器请求呼叫远程服务器的许可
try
{
    //如果浏览器不是规定的则生成一个被忽略的错误
    //Mozilla
    netscape.security.PrivilegeManager.enablePrivilege('UniversalBrowserRead');
}
catch(e) {}
//忽略错误
```

这段代码是专用于 Mozilla 的浏览器的, 使用 try/catch 结构, 在这段代码中的任何错误都会被忽视, 而其他浏览器使用时将产生异常。

### 3.4 使用代理服务器脚本

很明显, 除非有一种能够控制环境的方法, 例如能够确定用户使用的是 IE 或 Firefox (这种情况下, 需要标记脚本或手工配置浏览器, 使其有更高的许可权), 否则让 JavaScript 代码访问远程服务器的操作是不可能的。

好在变通方法并不难, 不用 JavaScript 直接访问远程服务器, 而使用本地服务器上的 PHP 脚本访问代表客户端的远程服务器即可, 如图 3.12 所示。



图 3.12 使用代理 PHP 脚本访问远程服务器

可以使用 `file_get_contents` 函数通过 PHP 从远程服务器读取数据, 从下面的网址中可以找到关于这个函数的文件: <http://www.php.net/manual/en/function.file-get-contents.php>。

#### ⚠ 注意

常用的也是更有效的可以替换 `file_get_contents` 的方法是使用名为“Client URL Library(CURL)”的库。关于 CURL 的更多内容请访问 <http://curl.haxx.se>、<http://www.php.net/curl> 以及 <http://www.zend.com/zend/tut/tutorial-thome3.php>。不过对于最基本的需求而言, `file_get_contents` 完成得更简洁而且完美。

下面通过代码进行试验，我们期望完成的功能同前面的练习一样，即得到一个随机数并将其显示出来，不过这一次将能够在所有的浏览器下工作。

## 》》》 实现步骤——使用代理服务器脚本实现远程服务器访问

- (1) 在根目录下创建一个名为 `proxyping` 的新目录。
- (2) 在 `proxyping` 目录下建立一个名为 `proxyping.html` 的新文件：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
<head>
<title>Practical Ajax: Accessing Remote Server through Proxy PHP Script
</title>
<script type="text/javascript" src="proxyping.js"></script>
</head>
<body onload="process()">
Server, tell me a random number!<br/>
<div id="myDivElement" />
</body>
</html>
```

(3) 在同一个目录下创建 `proxyping.js` 文件，注意这个文件类似于“`ping.js`”，下面代码中黑体部分是新增的（从处理 Mozilla 安全问题的 `process()` 中复制了这些代码，并在头部改变了服务器的地址。由于一次只需要得到一个数字，因此删除了“`num`”参数，并增加了一个出错处理方法）。

```
//保留一个 XMLHttpRequest 请求
var xmlhttp = createXmlHttpRequestObject();
//保留远程服务器地址和参数
var serverAddress = "proxyping.php";
var serverParams = "&min=1" + //the min number to generate
                "&max=100"; //the max number to generate
//创建一个 XMLHttpRequest 请求
function createXmlHttpRequestObject()
{
  //为 XMLHttpRequest 对象存储索引
  var xmlhttp;
  //用于除了 IE6 及其更早期的版本以外的所有浏览器
  try
```

```
{
    //试图创建一个 XMLHttpRequest 对象
    xmlhttp = new XMLHttpRequest();
}
catch(e)
{
    //用于 IE6 及其早期版本
    var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
                                     "MSXML2.XMLHTTP.5.0",
                                     "MSXML2.XMLHTTP.4.0",
                                     "MSXML2.XMLHTTP.3.0",
                                     "MSXML2.XMLHTTP",
                                     "Microsoft.XMLHTTP");

    //尝试每个程序直到其中一个能够运行
    for (var i=0; i<XmlHttpVersions.length && !xmlhttp; i++)
    {
        try
        {
            //创建 XMLHttpRequest 对象
            xmlhttp = new ActiveXObject(XmlHttpVersions[i]);
        }
        catch (e) {}
    }
}
//返回被创建的对象或显示一个出错信息
if (!xmlhttp)
    alert("Error creating the XMLHttpRequest object.");
else
    return xmlhttp;
}
//异步调用服务器
function process()
{
    //仅在 xmlhttp 不为空时继续程序
    if (xmlhttp)
    {
        //尝试连接服务器
        try
        {
            //启动服务器访问
```

```
xmlHttp.open("GET", serverAddress + "?" + serverParams, true);
xmlHttp.onreadystatechange = handleRequestStateChange;
xmlHttp.send(null);
}
//失败时显示出错信息
catch (e)
{
    alert("Can't connect to server:\n" + e.toString());
}
}
//当 HTTP 请求状态改变时调用函数
function handleRequestStateChange()
{
    //当 readyState 为 4 时, 准备接收服务器响应
    if (xmlHttp.readyState == 4)
    {
        //仅在 HTTP 状态为"OK"时程序继续
        if (xmlHttp.status == 200)
        {
            try
            {
                //根据服务器的响应进行处理
                handleServerResponse();
            }
            catch(e)
            {
                //显示出错信息
                alert("Error reading the response: " + e.toString());
            }
        }
        else
        {
            //显示状态信息
            alert("There was a problem retrieving the data:\n" +
                xmlHttp.statusText);
        }
    }
}
//处理来自于服务器的响应
```

```
function handleServerResponse()
{
    //重新接收服务器的响应
    var response = xmlhttp.responseText;
    //如果相应超过 3 个字符或者为空字符, 认为收到的是一个服务器端错误报告
    if(response.length > 3 || response.length == 0)
        throw(response.length == 0 ? "Server error" : response);
    //获得页面上<div>元素的索引
    myDiv = document.getElementById("myDivElement");
    //显示 HTML 输出
    myDiv.innerHTML = "Server says: " + response + "<br/>";
}
}
```

#### (4) 建立代理 PHP 脚本, proxying.php:

```
<?php
//载入出错处理模块
require_once('error_handler.php');
//确定用户浏览器不会缓存结果
header('Expires: Wed, 23 Dec 1980 00:30:00 GMT');
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');
header('Cache-Control: no-cache, must-revalidate');
header('Pragma: no-cache');
//重新获得参数
$num = 1; //this is hardcoded on the server
$min = $_GET['min'];
$max = $_GET['max'];
//保留远程服务器地址与参数
$serverAddress = 'http://www.random.org/cgi-bin/randnum';
$serverParams = 'num=' . $num . //how many random numbers to generate
                '&min=' . $min . //the min number to generate
                '&max=' . $max; //the max number to generate
//重新获得来自外部服务器的随机数
$randomNumber = file_get_contents($serverAddress . '?' . $serverParams);
//输出随机数
echo $randomNumber;
?>
```

(5) 最后, 增添出错处理功能。这样做会使录入的内容有所增加, 但对解决问题是有益的。由于没有什么需要改变的, 可以从其他练习中将代码复制粘贴过来, 然后建立一个名为 `error_handler.php` 的新文件, 其代码如下:

```

<?php
//设置用户出错处理方法为 error_handler
set_error_handler('error_handler', E_ALL);
//出错处理函数
function error_handler($errNo, $errStr, $errFile, $errLine)
{
    //清除已生成的所有输出
    if(ob_get_length()) ob_clean();
    //输出出错信息
    $error_message = 'ERRNO: ' . $errNo . chr(10) .
                    'TEXT: ' . $errStr . chr(10) .
                    'LOCATION: ' . $errFile .
                    ', line ' . $errLine;
    echo $error_message;
    //阻止处理任何更多的 PHP 脚本
    exit;
}
?>

```

(6) 在包括 Opera 在内的任何一种浏览器的地址栏中输入如下地址：<http://localhost/ajax/foundations/proxying/proxying.html>，如图 3.13 所示，即可看到期盼已久的随机数了！

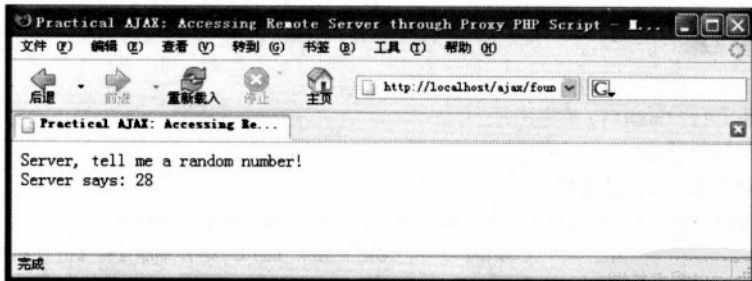


图 3.13 使用代理 PHP 脚本访问远程服务器

## 程序说明

由于运行 JavaScript 代码访问装载它的服务器，因此可以在服务器内加入一个名为 proxying.php 的脚本，用它访问随机数生成服务器。

为了使客户端仍能完全控制接收到的数字，我们将 min 与 max 这两个参数传递给 PHP 脚本，而 PHP 脚本按顺序将它传递给随机数生成服务器。我们不希望让客户端一次请求多于一个数字，因此没有从客户端传递 num 参数。本例中，当响应超过 3 个字符时，就认为收到

了一份服务器出错报告：

```
//处理来自于服务器的响应
function handleServerResponse()
{
    //重新接收服务器的响应
    var response = xmlhttp.responseText;
    //如果响应超过 3 个字符或者为空字符，认为收到的是一个服务器端错误报告
    if(response.length > 3 || response.length == 0)
        throw(response.length == 0 ? "Server error" : response);
}
```

### ⚠ 注意

错误可能出现在客户端，也可能出现在服务器端。可以对代码中的部分关键字执行 try/catch 机制，以尽量使客户端受到保护。

相反，当服务器发生错误时，这个错误不会作为客户端错误传播给客户端。因此，在客户端必须自行分析从服务器端收到的输入，如果不是所需要的，可以使用“throw”自行抛出一个错误。

如果 php.ini 中的 display\_errors 设置为“off”，那么当一个 PHP 解析或致命错误发生时，该错误仅仅记录在 Apache 错误日志文件中（Apache/logs/error.log），并且此时不输出脚本。因此，如果收到一个空响应，可以认定服务器端出现错误，可以在客户端产生普通的出错信息。

举一个例子，如果没有可用的网络连接（所以远程服务器不可达），而用户又试图载入页面时，将导致产生如下错误（如果 php.ini 中的 display\_errors 设置为“off”，会显示不同的错误信息），如图 3.14 所示。

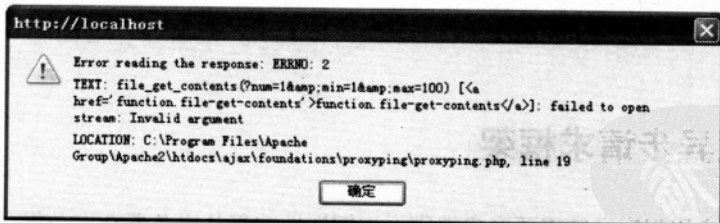


图 3.14 没有可用的网络连接时的出错信息

proxyping.php 文件中的代码只使用“GET”方法访问随机数生成服务器所收到的参数。这里有一个有趣的细节需要注意，这个脚本是在标题中设置“页面失效”（page expiration）时使用的方法，由于总是有相同的 URL 或请求串访问服务器，因此设置页面超时是很重要的，并且客户端浏览器可能会自动隐藏结果——这并不是我们所希望的，因为如果这样，这些结果就不再是严格的随机数了。

```
?php
//载入出错处理模块
require_once('error_handler.php');
//确定用户浏览器不会缓存结果
header('Expires: Wed, 23 Dec 1980 00:30:00 GMT');
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');
header('Cache-Control: no-cache, must-revalidate');
header('Pragma: no-cache');
```

在 <http://www.sitepoint.com/article/php-anthology-2-5-caching> 上有一篇很好的介绍页缓存与 PHP 的文章，可以去查阅一下。proxyping.php 的剩余部分仅使用 file\_get\_contents 函数重新得到随机数生成服务器的响应，并输出给客户端。

```
//重新获得参数
$num = 1; //this is hardcoded on the server
$min = $_GET['min'];
$max = $_GET['max'];
//保留远程服务器地址与参数
$serverAddress = 'http://www.random.org/cgi-bin/randnum';
$serverParams = 'num=' . $num . //how many random numbers to generate
                '&min=' . $min . //the min number to generate
                '&max=' . $max; //the max number to generate
//重新获得来自外部服务器的随机数
$randomNumber = file_get_contents($serverAddress . '?' . $serverParams);
//输出随机数
echo $randomNumber;
?>
```

### 3.5 重复异步请求框架

建立 Ajax 应用程序经常需要客户端脚本周期性地重复从服务器获取数据，这类例子非常多，本书会多次遇到，在实际中可能遇到得更多。

JavaScript 提供了 4 个函数，用于完成重复（或预定）功能——setTimeout、setInterval、clearTimeout 和 clearInterval，其使用方法如下：

```
//使用 setTimeout 与 clearTimeout
timerId = window.setTimeout("function()", interval_in_milliseconds);
window.clearTimeout(timerId);
```



```
//使用 setInterval 与 clearInterval
timerId = window. setInterval("function()", interval_in_milliseconds);
window. clearInterval (timeId);
```

在指定的时间周期过后，`setInterval` 中的函数被执行一次。`setInterval` 则是在指定的周期内重复执行函数，直到使用了 `clearInterval` 函数。对于大多数的 Ajax 情节而言，我们更倾向于使用 `setTimeout`，这是因为在服务器被访问时，对该函数的控制会更加灵活一些。

作为一个快速示范，可以通过做如下改进扩展读取随机数的客户端。

- 当发出一个服务器请求时，我们将一直等待直到收到包含随机数的响应为止，1 秒钟后使用 `setTimeout` 重新产生序列（创建新的服务器请求）。对于这种方法，两次请求的时间间隔是一秒钟加上重新得到随机数所用的时间。如果希望在严格的周期内完成请求，则必须使用 `setInterval`，但这需要检查 `XMLHttpRequest` 对象是否在忙于完成前一次请求（这种情况在网速慢或者服务器忙时发生）。

- 在新例子中，有时会检测服务器的可用性，随机数生成服务有一个随机数缓冲器，用于缓存请求，可以在 <http://www.random.org/cgi-bin/checkbuf> 核对缓冲器的使用程度。我们的程序将每产生 10 个请求核查一次该页面，只有在缓冲器使用率低于 50% 以上时才请求新的随机数。

Web 应用程序如图 3.15 所示。

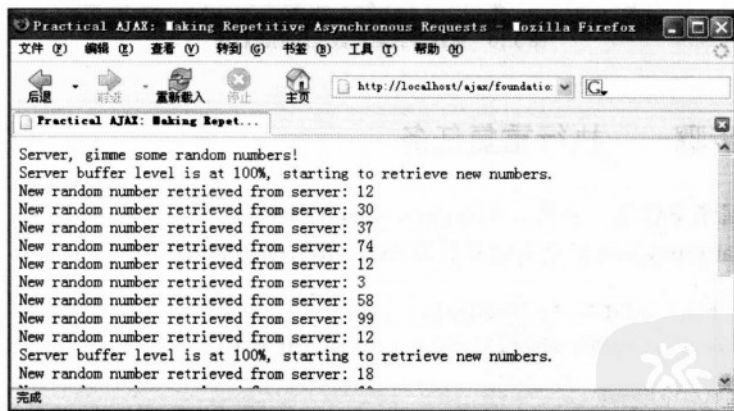


图 3.15 发出重复异步请求

这个重复性的任务自然有其开始之处，程序是从 `process()` 开始的，该函数判定发出何种服务器请求；也可以请求产生一个新的随机数，或者查看随机数生成服务器的缓冲器使用情况，每产生 10 次请求检查一次缓冲器，并且在默认情况下，只要缓冲器利用率低于 50%，就重新请求新的随机数。程序的流程图如图 3.16 所示。

使用默认代码的情况下，`setTimeout` 仅在 HTTP 请求成功之后重启程序，`catch` 模块中没有 `setTimeout`（根据使用的特定解决方案，发生错误后也可以尝试再次访问服务器）。

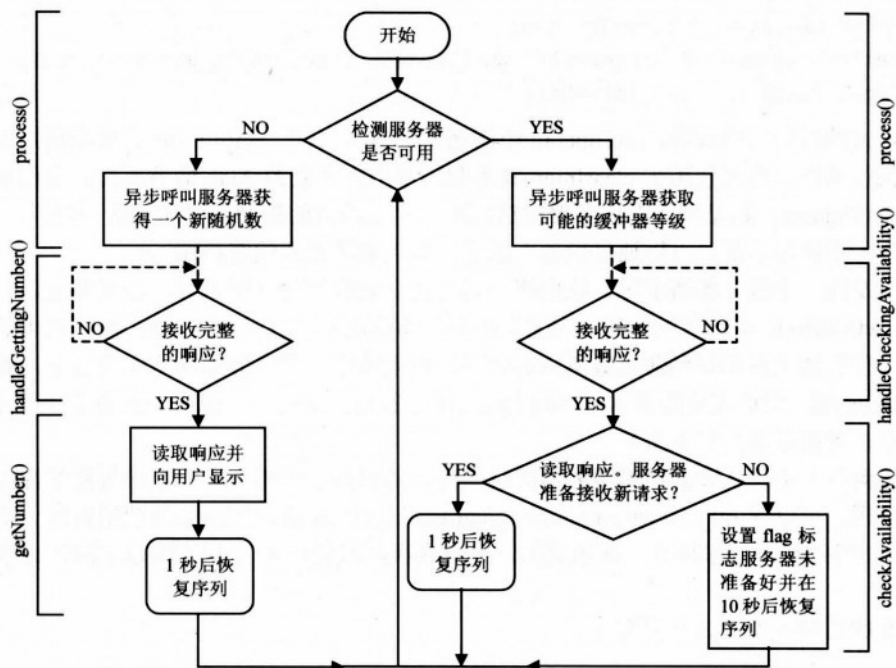


图 3.16 重新得到随机数程序的流程图

## 》》》 实现步骤——执行重复任务

- (1) 在根目录下建立一个名为 smartproxying 的新目录。
- (2) 在 smartproxying 目录下建立名为 smartproxying.html 的文件：

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
  <head>
    <title>Practical Ajax: Making Repetitive Asynchronous Requests</title>
    <script type="text/javascript" src="smartproxying.js"></script>
  </head>
  <body onload="process()">
    Server, gimme some random numbers!<br/>
    <div id="myDivElement" />
  </body>
</html>
  
```

(3) 在同一目录下建立 smartproxyping.js 文件。

```
//保留一个XMLHttpRequest 请求
var xmlHttp = createXmlHttpRequestObject();
//保留远程服务器地址和参数
var serverAddress = "smartproxyping.php";
var getNumberParams = "action=GetNumber" + //get a new random number
    "&min=1" + //the min number to generate
    "&max=100"; //the max number to generate
var checkAvailabilityParams = "action=CheckAvailability";
//用于测试服务器是否可用的变量
var requestsCounter = 0; //counts how many numbers have been retrieved
var checkInterval = 10; //counts interval for checking server availability
var updateInterval = 1; //how many seconds to wait to get a new number
var updateIntervalIfServerBusy = 10; //seconds to wait when server busy
var minServerBufferLevel = 50; //what buffer level is considered acceptable

//创建一个XMLHttpRequest 请求
function createXmlHttpRequestObject()
{
    //为XMLHttpRequest 对象存储索引
    var xmlHttp;
    //用于除了IE6及其更早期的版本以外的所有浏览器
    try
    {
        //试图创建一个XMLHttpRequest 对象
        xmlHttp = new XMLHttpRequest();
    }
    catch(e)
    {
        //用于IE6及其早期版本
        var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
            "MSXML2.XMLHTTP.5.0",
            "MSXML2.XMLHTTP.4.0",
            "MSXML2.XMLHTTP.3.0",
            "MSXML2.XMLHTTP",
            "Microsoft.XMLHTTP");

        //尝试每个程序直到其中一个能够运行
        for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
        {
```

```
try
{
    //创建 XMLHttpRequest 对象
    xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
}
catch (e) {}
}
}
//返回被创建的对象或显示一个出错信息
if (!xmlHttp)
    alert("Error creating the XMLHttpRequest object.");
else
    return xmlHttp;
}

//异步调用服务器
function process()
{
    //仅在 xmlHttp 不为空时继续程序
    if (xmlHttp)
    {
        //尝试连接服务器
        try
        {
            //恰好开始或者找到指定的请求数字
            //测试能否连接服务器, 若不能则申请一个新随机数
            if (requestsCounter % checkInterval == 0)
            {
                //测试服务器是否可用
                xmlHttp.open("GET", serverAddress + "?" +
                    checkAvailabilityParams, true);
                xmlHttp.onreadystatechange = handleCheckingAvailability;
                xmlHttp.send(null);
            }
        }
        else
        {
            //获得一个新随机数
            xmlHttp.open("GET", serverAddress + "?" + getNumberParams, true);
            xmlHttp.onreadystatechange = handleGettingNumber;
```

```
        xmlHttp.send(null);
    }
}
catch(e)
{
    alert("Can't connect to server:\n" + e.toString());
}
}

//当 HTTP 请求状态改变时调用函数
function handleCheckingAvailability()
{
    //当 readyState 为 4 时, 准备接收服务器响应
    if (xmlHttp.readyState == 4)
    {
        //仅在 HTTP 状态为 "OK" 时程序继续
        if (xmlHttp.status == 200)
        {
            try
            {
                //根据服务器的响应进行处理
                checkAvailability();
            }
            catch(e)
            {
                //显示出错信息
                alert("Error reading server availability:\n" + e.toString());
            }
        }
        else
        {
            //显示状态信息
            alert("Error reading server availability:\n" + xmlHttp.statusText);
        }
    }
}

//处理来自于服务器的响应
function checkAvailability()
```

```
{
    //重新接收服务器的响应
    var response = xmlhttp.responseText;
    //如果响应足够长或者为空字符,认为收到的是一个服务器端错误报告
    if(response.length > 5 || response.length == 0)
        throw(response.length == 0 ? "Server error" : response);
    //obtain a reference to the <div> element on the page
    myDiv = document.getElementById("myDivElement");
    //获得页面上<div>元素的索引
    if (response >= minServerBufferLevel)
    {
        //向用户显示新的信息
        myDiv.innerHTML += "Server buffer level is at " + response + "%, "
            + "starting to retrieve new numbers. <br/>";
        //Counter自增加以开始重新接收新数字
        requestsCounter++;
        //重新排序
        setTimeout("process();", updateInterval * 1000);
    }
    else
    {
        //向用户显示新的信息
        myDiv.innerHTML += "Server buffer is too low (" + response + "%), "
            + "will check again in " + updateIntervalIfServerBusy
            + " seconds. <br/>";
        //重新排序
        setTimeout("process();", updateIntervalIfServerBusy * 1000);
    }
}
//当 HTTP 状态请求发生改变时调用函数
function handleGettingNumber()
{
    //当 readyState 为 4 时,准备接收服务器响应
    if (xmlhttp.readyState == 4)
    {
        //仅在 HTTP 状态为 "OK" 时程序继续
        if (xmlhttp.status == 200)
        {
            try
            {
```

```

        //根据服务器的响应进行处理
        getNumber();
    }
    catch(e)
    {
        //显示出错信息
        alert("Error receiving new number:\n" + e.toString());
    }
}
else
{
    //显示状态信息
    alert("Error receiving new number:\n" + xmlhttp.statusText);
}
}
}
//处理来自于服务器的响应
function getNumber()
{
    //重新接收服务器的响应
    var response = xmlhttp.responseText;
    //如果响应足够长或者为空字符,认为收到的是一个服务器端错误报告
    if(response.length > 5 || response.length == 0)
        throw(response.length == 0 ? "Server error" : response);
    //获得页面上<div>元素的索引
    myDiv = document.getElementById("myDivElement");
    //显示HTML输出
    myDiv.innerHTML += "New random number retrieved from server: "
+ response + "<br/>";
    //请求计数器加1
    requestsCounter++;
    //重新排序
    setTimeout("process();", updateInterval * 1000);
}
}

```

(4) 在同一目录下建立 smartproxyping.php 文件。

```

<?php
//载入出错处理模块
require_once('error_handler.php');
//确定用户浏览器不会缓存结果

```

```
header('Expires: Wed, 23 Dec 1980 00:30:00 GMT'); //time in the past
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');
header('Cache-Control: no-cache, must-revalidate');
header('Pragma: no-cache');
//重新获得参数
$action = $_GET['action'];
//测试可用性或得到新随机数
if ($action == 'GetNumber')
{
    $num = 1; //value is hardcoded because client can't deal with more numbers
    $min = $_GET['min'];
    $max = $_GET['max'];
    //保留远程服务器地址与参数
    $serverAddress = 'http://www.random.org/cgi-bin/randnum';
    $serverParams = 'num=' . $num . //how many random numbers to generate
        '&min=' . $min . //the min number to generate
        '&max=' . $max; //the max number to generate
    //重新获得来自外部服务器的随机数
    $randomNumber = file_get_contents($serverAddress . '?' . $serverParams);
    //输出随机数
    echo $randomNumber;
}
elseif ($action == 'CheckAvailability')
{
    //返回缓冲器使用情况的页面地址
    $serverAddress = 'http://www.random.org/cgi-bin/checkbuf';
    //以'x%'形式接收缓冲器使用情况
    $bufferPercent = file_get_contents($serverAddress);
    //取出数字
    $buffer = substr($bufferPercent, 0, strlen($bufferPercent) - 2);
    //重复数字
    echo $buffer;
}
else
{
    echo 'Error talking to the server.';
}
?>
```

(5) 在同一目录下建立 `error_handler.php` 文件，其内容同前：



```

<?php
//设置用户出错处理方法为 error_handler
set_error_handler('error_handler', E_ALL);
//出错处理函数
function error_handler($errNo, $errStr, $errFile, $errLine)
{
    //清除已生成的所有输出
    if(ob_get_length()) ob_clean();
    //输出出错信息
    $error_message = 'ERRNO: ' . $errNo . chr(10) .
                    'TEXT: ' . $errStr . chr(10) .
                    'LOCATION: ' . $errFile .
                    ', line ' . $errLine;

    echo $error_message;
    //阻止处理任何更多的 PHP 脚本
    exit;
}
?>

```

(6) 载入 <http://localhost/ajax/foundations/smartproxyping/smartproxyping.html> 页面，输出如图 3.15 所示。

## 》》》 程序说明

本例中的客户端知道如何随时监测服务器的可用性，随机数生成服务器提供了一个可以用来查看缓冲器使用情况的网页：<http://www.random.org/cgi-bin/checkbu>。

Smartproxyping.js 中的 JavaScript 代码以定义一批用来控制程序运行的全局变量为起点：

```

//保留远程服务器地址和参数
var serverAddress = "smartproxyping.php";
var getNumberParams = "action=GetNumber" + //get a new random number
                      "&min=1" + //the min number to generate
                      "&max=100"; //the max number to generate
var checkAvailabilityParams = "action=CheckAvailability";
//用于测试服务器是否可用的变量
var requestsCounter = 0; //读者的数量
var checkInterval = 10; //检查服务器可用性的间隔
var updateInterval = 1; //新数的等待时间(单位: 秒)
var updateIntervalIfServerBusy = 10; //服务器忙时的等待时间(秒)
var minServerBufferLevel = 50; //what buffer level is considered acceptable

```

这些变量包含生成服务器请求所必需的数据，`getNumberParams` 包含请求新随机数的询问字符串参数，而 `checkAvailabilityParams` 包含用来检测服务器缓冲器使用情况的参数，其余变量用来控制完成异步请求的时间间隔。

与前一个练习相比，这个练习新增了两个处理服务器响应的函数——`handleCheckingAvailability` 与 `handleGettingNumber`，发生这些的根本原因在于 `process()` 函数，该函数根据所请求的服务器行为指派使用这些回调函数。

和其他练习一样，本程序中的 `process()` 函数也不只是调用了一次，而是调用了多次。而且每次都必须判断需要采取何种行为——请求一个新随机数还是查看服务器缓冲器的使用情况。变量 `requestCounter` 用来跟踪自上一次缓冲器检查开始，有多少次重新得到随机数，以帮助我们进行判断。

```
function process()
{
    //...
    if (requestsCounter % checkInterval == 0)
    {
        //测试服务器是否可用
        xmlHttp.open("GET", serverAddress + "?" + checkAvailabilityParams, true);
        xmlHttp.onreadystatechange = handleCheckingAvailability;
        xmlHttp.send(null);
    }
    else
    {
        //获得一个新随机数
        xmlHttp.open("GET", serverAddress + "?" + getNumberParams, true);
        xmlHttp.onreadystatechange = handleGettingNumber;
        xmlHttp.send(null);
    }
    //...
}
```

`handleCheckingAvailability` 与 `handleGettingNumber` 函数功能相似，都是前面的练习中 `handleRequestStateChange` 函数的特定版本。这两个函数的功能是等到来自服务器的响应成功接收时，调用辅助函数 `checkAvailability` 和 `getNumber` 立刻处理响应。

注意“action”询问字符串参数，其作用是通知 PHP 脚本执行哪些远程服务器请求。服务器端加载出错处理模块后，在 `smartproxying.php` 中读取该 `action` 的参数并由其值判断下一步的行动。

```
<?php
//加载错误处理模块
```

```

require_once('error_handler.php');
//获取 action 参数
$action = $_GET['action'];
//检查可用性还是获取新随机数?
if ($action == 'GetNumber')
{
    //...

```

如果 action 是 GetNumber，就使用 PHP 的功能 `file_get_contents` 从服务器读取新的随机数：

```

if ($action == 'GetNumber')
{
    $num = 1; //赋初值 1, 因为客户端处理不了更多的数值
    $min = $_GET['min'];
    $max = $_GET['max'];
    //保存远程服务器的地址和参数
    $serverAddress = 'http://www.random.org/cgi-bin/randnum';
    $serverParams = 'num=' . $num . //生成多少随机数
                  '&min=' . $min . //生成的最小值
                  '&max=' . $max; //生成的最大值
    //从外部服务器取回随机数
    $randomNumber = file_get_contents($serverAddress . '?' . $serverParams);
    //生成随机数
    echo $randomNumber;
}

```

如果 action 是 CheckAvailability 则调用：

```

elseif ($action == 'CheckAvailability')
{
    //返回缓冲器使用情况的页面地址
    $serverAddress = 'http://www.random.org/cgi-bin/checkbuf';
    //以 '%x%' 形式接收缓冲器使用情况
    $bufferPercent = file_get_contents($serverAddress);
    //取出数字
    $buffer = substr($bufferPercent, 0, strlen($bufferPercent) - 2);
    //重复数字
    echo $buffer;
}

```

注意 `file_get_contents` 调用不是异步的，也不是必需的。PHP 脚本不直接连接用户，而且脚本一直有效直到需要结束时。客户端的 `checkAvailability` 与 `getNumber` 函数接收这些 PHP

脚本生成的响应。这些函数一开始读取响应并检查其大小：

```
//handles the response received from the server
function getNumber()
{
    //重新接收服务器的响应
    var response = xmlhttp.responseText;
    //如果响应足够长或者为空字符，认为收到的是一个服务器端错误报告
    if(response.length > 5 || response.length == 0)
        throw(response.length == 0 ? "Server error" : response);
}
```

#### ⚠ 注意

这是一种检查 PHP 脚本是否成功执行的方法，根据响应的长度判断执行是否成功是一种十分简单却很有效的方法。事实上，PHP 抛出那些的致命错误不能捕获或者处理，这使得它很难建立一个通用的并且有效的出错处理机制。

除了检测错误之外，在实际问题中还要十分认真地考虑如何处理错误——操作是无穷的，要根据实际情况进行选择。牢记“用户不关心错误的技术细节”。例如，在我们的问题中，只需简单地输出一条类似于“服务器临时故障，请稍后访问”的信息足矣。

然而，如果要输出准确的出错信息，就要在 PHP 的致命错误输出 HTML 格式的信息时考虑客户造成的错误要使用“\n”换成新的一行输出，如果想用 JavaScript 对话框显示该信息，则需要对其进行相应的排版。

更新客户端的显示后，使用 `setTimeout` 重新初始化排序：

```
//重新排序
setTimeout("process();", updateInterval * 1000);
}
```

## 3.6 使用 MySQL

运行任何应用程序如果期望生成有用的动态输出时，就必需使用后台存储。最常见的方式就是在关系数据库管理系统（RDBMS）存储应用程序的数据，所谓 RDBMS，就是一个能够存储、管理数据的非常有效的工具。

与很多其他因素一样，数据库并不是 Ajax 的一部分，但没有数据库支持就不可能建立真正的应用程序。本书的例子将给出简单但并不包含需要复杂数据的应用程序，但它仍需要数据库的支持。本书选择了 MySQL，在现在 PHP 开发者中非常流行。由于大部分数据库的功

能都大同小异，因此只需要做一点小小的改动就可以将其移植到其他数据库系统。

建立一个使用数据库的应用程序需要掌握以下基本技能。

- (1) 建立能够存储数据的数据库表。
- (2) 编写 SQL 语句操作数据。
- (3) 使用 PHP 代码连接到 MySQL 数据库。
- (4) 向数据库发送 SQL 查询请求并检索结果。



**注意**

再次强调，这里只是简单介绍 PHP 与 MySQL 数据库的基础知识，PHP 与 MySQL 的免费在线指南编写得很好，因此可以通过这种方式查找相关知识。

### 3.6.1 创建数据库表

创建数据表需要了解有关数据库结构的基本概念。一个数据表由列（域）和行（记录）组成，创建数据表需要定义数据表的域，不同的域具有不同的属性，本书中将用到：

- 主键
- 数据类型
- 空列与非空列
- 默认列值
- 自增列
- 索引

主键是使每一行能够被唯一识别的一个特殊列（或这些列的组合），主键列不允许有重复值，因此每个值都是唯一的。如果主键列超过一列，则多列（而不是单独的列）列值必须惟一。在技术上，主键是对一列的约束（一个规定），但是习惯上，提到“主键”的时候通常指被设定了“PRIMARY-KEY”约束的列。当一个列规定主键以后，在该列将同时创建一个惟一的索引，主要目的是为了改进搜索性能。

每一列都有一个数据类型，用来描述其大小与行为。数据类型包括 3 个重要的类别（数字类型、字符与字符串类型、日期与时间类型），且每种类别又包含多种数据类型。关于这些问题可以在 <http://dev.mysql.com/doc/refman/5.0/en/data-types.html> 找到 MySQL 官方文件。

当创建一个新的数据表时，需要确定哪些值是必备的，并用“NOT NULL”属性标识，表示该列不允许存储空值。NULL 的意思就是未指定，读取时如果表中某项为“空”，表示该域中的值还没有指定。需要注意的是，空串、包含空格的串或值为 0（针对数字列而言）都是真实的值，即非空的值，主键的域不允许为空值。

有时为了避免某列的值为空，替换（或补充）办法是指定一个“默认值”。这样，创建新记录后，如果某个域没有赋值，就使用默认值。对于默认值而言，可以设立一个函数以便在需要时恢复其值。

使系统自动生成值的另一个方法是使用自增列，这是一个会经常被用到的针对主键列的操作，能够自动生成 ID。可以只对数字列设置自增操作，以自动增加新产生的数字，因此任何值都不会被生成两次。

索引是用来改进数据库操作性能的数据库对象。索引是能明显增强在所设置域内搜索效率的，不过在更新及插入操作时，索引也必须及时进行更新，因此索引会降低这两种操作的速度。适当的结合索引可以极大地加快应用程序的运行速度，本书中的例子，依赖于主键列上建立的索引。

可以使用 SQL 代码创建一个数据表或使用可视化界面，下面是创建一个简单数据表的 SQL 命令。

```
CREATE TABLE USERS
(
  user_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  user_name VARCHAR(32) NOT NULL,
  PRIMARY KEY (user_id)
);
```

如果对所创建的表不满意，可以用“ALTER TABLE”进行修改操作，或者用“DROP TABLE”操作将其完全删除，还可以用“TRUNCATE TABLE”迅速删除并重新创建一个表，这个操作与删除所有记录的效果相同，但速度更快且能清除自动产生的索引。

下面的每个练习都会给出建立必需数据表的 SQL 代码，可以使用诸如 phpMyAdmin（附录 A 讲述了其安装的过程）这样的程序执行该代码。为了使用 phpMyAdmin 执行 SQL 代码，需要在“数据库列表”中选择一个数据库名并连接到该数据库，然后在主面板中单击 SQL 标签，如图 3.17 所示。

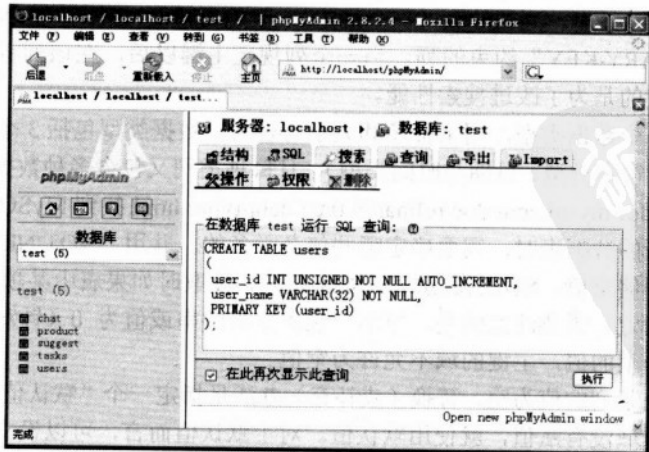


图 3.17 使用 phpMyAdmin 执行 SQL 代码

phpMyAdmin 也支持可视化建表，使用方式如图 3.18 所示。



图 3.18 使用 phpMyAdmin Designer 创建新表

### ⚠ 注意

如果读者困惑于“表类型”操作，请看下面的内容。MySQL 不同于其他数据库产品的地方是它集成了一些数据引擎，包括最流行的两个 MyISAM 与 InnoDB。其中令人感兴趣的是，在一个简单的数据库中，可以创建不同类型的表，且在创建表的同时即可指定其类型，如果不指定类型将使用默认设置，大多数设置为 MyISAM。每个引擎都有其各自的优缺点，不过功能最强的应该是 InnoDB，它完全支持 ACID（Atomicity、Consistency、Isolation 及 Durability）特性，即原子性、一致性、隔离性以及持久性，还支持行级锁、外键及参照完整性等。MyISAM 区别于其他引擎的一个明显标志是它支持快速的全文本搜索。

## 3.6.2 数据操作

可以用 SQL 的 DML（Data Manipulation Language，即数据操作语言）命令对数据进行各种操作——查询、插入、更新以及删除，用来在数据表中完成恢复、新增、修改以及删除记录的功能，这些功能非常强大，也非常灵活。它们基本的语法如下。

```
SELECT <column list>
FROM <table name(s)>
[WHERE <restrictive condition(s)>]

INSERT INTO <table name> [(column list)]
VALUES (column values)

UPDATE <table name>
SET <column name> = <new value> [, <column name> = <new value> ... ]
```

```
[WHERE <restrictive condition>]
```

```
DELETE FROM <table name>
```

```
[WHERE <restrictive condition>]
```

下列内容必须牢记：

- SQL 代码可以写为一行或者多行，只要美观就行。
- 如果需要一次执行多个 SQL 命令，必须用分号（；）将这些命令分隔开。
- 语法中方括号之间的值是可选的，不过需要注意 DELETE 的声明，如果没有指定限制条件，“所有”元素都将被删除。
- 使用“SELECT”时，可以用通配符“\*”代替列的列表，它包含所有存在的表列。
- SQL 不区分大小写，但习惯上用大写字母写 SQL 声明，而表与域名用小写，保持一致性总是有益的。

可以用前面提到的“USERS”表练习这些命令的使用方法，在 phpMyAdmin 中任意打开一个 SQL 标签并执行如下命令：

```
INSERT INTO users (user_name) VALUES ('john');
INSERT INTO users (user_name) VALUES ('sam');
INSERT INTO users (user_name) VALUES ('ajax');
SELECT user_id, user_name FROM users;
UPDATE users SET user_name='cristian' WHERE user_id=1;
SELECT user_id, user_name FROM users;
DELETE FROM users WHERE user_id=3;
SELECT * FROM users WHERE user_id>1;
```

在本书学习中，将会遇到更复杂的语句，在需要时书中会有解释。不要忘记 SQL 是一个很大的主题，如果迄今为止读者仍没有写过足够多的 SQL 代码，就必须做些额外的练习。

### 3.6.3 连接数据库并执行查询操作

在下面的例子中，连接到数据库的代码由 PHP 编写。如图 3.19 所示，数据库并非由客户端直接访问，而是通过服务器上 PHP 代码写成的业务逻辑连接。

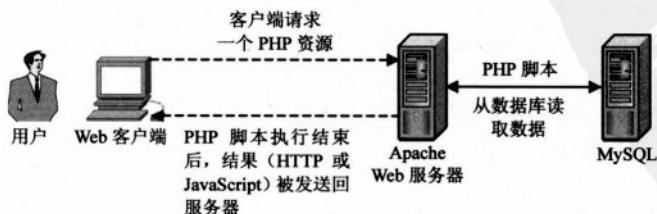


图 3.19 用户通过功能层连接 MySQL



为了访问所需要的数据，PHP 代码需要经过数据库的验证。

与其他的安全体系一样，数据库安全包含两个重要的概念：验证和授权。验证是一个通过使用一些分类注册机制（通常是输入一个用户名和密码）使用户能够被惟一识别的过程；授权是指资源可以被已验证的用户访问并执行其请求。

如果介绍附录 A 按照的方法对 MySQL 的安全进行设置，则需要连接到本地 MySQL 服务器上名为“ajax”的数据库，其用户名为“ajaxuser”，密码为“practical”。这些内容将被保存到一个名为“config.php”的配置文件，必要时可以很容易地更新该文件。config.php 的脚本如下：

```
<?
//定义数据库连接数据
define('DB_HOST', 'localhost');
define('DB_USER', 'ajaxuser');
define('DB_PASSWORD', 'practical');
define('DB_DATABASE', 'ajax');
?>
```

在执行数据库操作时将使用这组数据，任何对数据库的操作都是由以下必不可少的三步组成的：

- (1) 打开数据库连接。
- (2) 执行 SQL 查询并读取结果。
- (3) 关闭数据库连接。

因为打开数据库连接会消耗服务器资源，因此好的习惯是，尽可能晚一些打开数据库连接，并尽快将其关闭。下面的代码片段是一个简单的“打开连接、从服务器读取数据并关闭连接”的 PHP 脚本。

```
$mysqli = new mysqli(DB_HOST, DB_USER, DB_PASSWORD, DB_DATABASE);
//需要执行何种 SQL 查询?
$query = 'SELECT user_id, user_name FROM users';
//执行查询
$result = $mysqli->query($query);
//处理结果.....
//...
//关闭输入 stream
$result->close();
//关闭数据库连接
$mysqli->close();
```



**注意**

注意，这里使用 mysqli 库访问 MySQL，它是 mysql 库新改进的版本，提供面向对象和面向过程的访问 MySQL 数据库的接口，并可以使用 MySQL 更先进的功能。如果读者使用的是低版本的 MySQL 或 PHP 不支持 mysqli，也可以用 mysql 代替。

下面的练习中并不包含 Ajax 的特殊功能，仅仅是从 PHP 代码访问 MySQL 数据库的简单例子。

## 》》》 实现步骤——使用 PHP 与 MySQL 工作

(1) 连接 Ajax 数据库，使用下面的代码创建一个名为 users 的表：

```
CREATE TABLE users
(
    user_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    user_name VARCHAR(32) NOT NULL,
    PRIMARY KEY (user_id)
);
```

(2) 执行如下的插入 (INSERT) 命令，将示例数据存放于 users 表中：

```
INSERT INTO users (user_name) VALUES ('bogdan');
INSERT INTO users (user_name) VALUES ('filip');
INSERT INTO users (user_name) VALUES ('mihai');
INSERT INTO users (user_name) VALUES ('emilian');
INSERT INTO users (user_name) VALUES ('paula');
INSERT INTO users (user_name) VALUES ('cristian');
```

**⚠ 注意** 由于“user\_id”是一个自增列，因此它的值由数据库产生。

(3) 在根目录下创建名为 mysql 的新目录。

(4) 在 mysql 目录下建立名为 config.php 的文件，并写入配置数据库的代码，使用这些代码时要注意根据实际情况改变一些相应的值：

```
<?php
//定义数据库连接数据
define('DB_HOST', 'localhost');
define('DB_USER', 'ajaxuser');
define('DB_PASSWORD', 'practical');
define('DB_DATABASE', 'ajax');
?>
```

(5) 加入标准出错处理文件 error\_handler.php，可以从前面的练习中复制：

```
<?php
//设置用户出错处理方法为 error_handler
set_error_handler('error_handler', E_ALL);
//出错处理函数
```

```

function error_handler($errNo, $errStr, $errFile, $errLine)
{
    //清除已生成的所有输出
    ob_clean();
    //输出出错信息
    $error_message = 'ERRNO: ' . $errNo . chr(10) .
                    'TEXT: ' . $errStr . chr(10) .
                    'LOCATION: ' . $errFile .
                    ', line ' . $errLine;
    echo $error_message;
    //阻止处理任何更多的 PHP 脚本
    exit;
}
?>

```

(6) 建立名为 `index.php` 的新文件，其代码如下：

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/
xhtml11/DTD/xhtml11.dtd">
<html>
  <head>
    <title>Practical Ajax: Working with PHP and MySQL</title>
  </head>
  <body>

    <?php
    //载入配置文件
    require_once('error_handler.php');
    require_once('config.php');
    //连接到数据库
    $mysqli = new mysqli(DB_HOST, DB_USER, DB_PASSWORD, DB_DATABASE);
    //要执行的 SQL 查询
    $query = 'SELECT user_id, user_name FROM users';
    //执行查询
    $result = $mysqli->query($query);
    //通过结果循环
    while ($row = $result->fetch_array(MYSQLI_ASSOC))
    {
        //提取用户 id 和名字
        $user_id = $row['user_id'];
        $user_name = $row['user_name'];
    }

```

```

//输出数据
echo 'Name of user #' . $user_id . ' is ' . $user_name . '<br/>';
}
//关闭输入 stream
$result->close();
//关闭数据库连接
$mysqli->close();
?>

</body>
</html>

```

(7) 在浏览器中载入页面 <http://localhost/ajax/foundations/mysql/index.php>, 对脚本进行测试, 如图 3.20 所示。

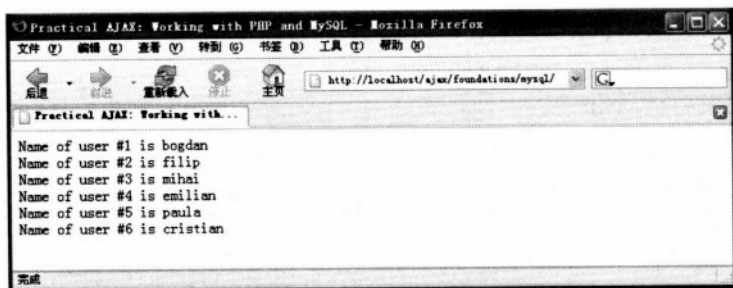


图 3.20 这些用户名是从数据库中读取的

## 程序说明

首先要注意这个练习中没有 Ajax 运行, 这个例子示范了 PHP 的数据访问功能。所有值得关注的事情都发生在 `index.php` 中, 在载入出错处理程序后真正的工作才开始, 其配置脚本如下:

```

<?php
//载入配置文件
require_once('error_handler.php');
require_once('mysql.class.php');

```

然后, 如同前面所提到的, 创建一个新的数据库连接:

```

//连接到数据库
$mysqli = new mysqli(DB_HOST, DB_USER, DB_PASSWORD, DB_DATABASE);

```

注意，数据库连接关联到数据库服务器中特定的数据库。我们所连接的数据库是 Ajax，包括前面所创建的 users 表，当在已创建的连接上执行查询任务时，一定可以访问 users 表了。

```
//要执行的 SQL 查询
$query = 'SELECT user_id, user_name FROM users';
//执行查询
$result = $mysqli->query($query);
```

上述命令执行后，变量 \$result 包含一个指向结果序列的指针，可以用 fetch\_array 方法将其逐行读出。这个方法返回一个数组，内容是当前结果所在行的各域，并将指针指向下一个结果行。可以用“while”循环对结果逐行分析，并读取每一行各自的域：

```
//通过结果循环
while ($row = $result->fetch_array(MYSQLI_ASSOC))
{
    //提取用户 id 和名字
    $user_id = $row['user_id'];
    $user_name = $row['user_name'];
    //输出数据
    echo 'Name of user #' . $user_id . ' is ' . $user_name . '<br/>';
}
```

最后，关闭已经打开的数据库对象，这样就不会浪费资源。为了避免影响同时正在运行的其他查询，也不要对数据库加任何锁：

```
//关闭输入 stream
$result->close();
//关闭数据库连接
$mysqli->close();
?>
```

## 3.7 程序封装与程序结构

本章的最后一节是建立程序结构的基本模式，在后面的学习中将沿用这种模式。前面已经建立了大多数的基本架构模块，还需要将服务器端的业务逻辑分离出来放在一个单独的类中，这些内容将在下面的练习中演示。

迄今为止，服务器端的代码通常是用单个 PHP 文件来实现的，为了实现更灵活高效的设计模式，我们把服务器端的 PHP 功能模块分为两个文件。

- 第一个脚本名为 appname.php (appname 是读者所申请的名字)，将是客户端 JavaScript 代码的主要访问点，负责处理由 POST 和 GET 所得到的输入参数，并根据这些参数进行判断。

• 第二个脚本名为 `appname.class.php`，包含一个名为 `Appname` 的帮助类，它将实际的处理过程封装了起来。这个类的方法由 `appname.php` 根据请求的行为进行调用。

完全理解这些代码的涵义需要掌握 OOP 的基本知识以及如何用 PHP 完成这些工作，本书中不再重复这些内容。请读者注意以下这些方面：

• OOP 基于类的概念，而类是为对象所设计的。类由类成员组成，包括方法（类中的函数）、构造函数、析构函数以及类域（其他 OOP 语言甚至包括更多的类成员类型）。类域类似于变量，不过类域只在类的范围内起作用。

• 在类中，可以执行构造函数与析构函数这两种特殊方法。构造函数名为 `__construct()`，当在类中创建新实例时，构造函数会自动执行。由于在类中创建一个新的对象需要立即执行构造函数，因此构造函数在代码需要初始化不同的类成员时很有用。

• 析构函数被命名为 `__destruct()`，当对象消灭时将自动运行该函数。析构函数的作用是释放资源，大多数时候，使用析构函数关闭数据库连接，确保没有任何打开的数据库连接浪费资源。

• 显然，恰好在需要之前创建数据库连接要比使用类的构造函数创建连接稍好一些；同样，在使用之后立即关闭连接也要优于使用类的析构函数关闭连接。然而，我们还是选择使用构造函数与析构函数，这样会得到更优质的代码，会减少错误的发生，比如，忘记了关闭连接。

访问任何一个类成员都必须指定其所属的对象，如果要访问当前类的成员，需要用特定的 `$this` 对象指向当前类的实例。

公共接口由类的 `public` 成员组成，可以从外部还可以由创建类实例的程序使用。类成员可以是 `public`、`private` 或 `protected` 三种类型，`private` 成员仅在类内部使用，而 `protected` 成员可以被子类访问。

将应用程序的功能分成多个层次是很重要的，这样就能够允许建立灵活并可扩展性的应用，而且必要时很容易更新。在 Cristian Darie 和 Mihai Bucica 关于 PHP 电子商务的书中，甚至能够学到如何使用一个名为 `Smarty` 的模板引擎，该模板引擎允许从 HTML 模板进一步分离表示层逻辑，以便设计者们不必再为站点的程序设计部分烦恼。

### ⚠ 注意

在准备设计代码时，牢记体系结构的功能、灵活以及可度量性直接关系到需要花费在设计体系架构与编写基本代码之间的时间比例。关于这些问题的参考资料可以到 <http://ajaxphp.packtpub.com/ajax/> 免费下载。

最后的这个练习中，将建立一个名为 `friendly` 的简单但完整的 Ajax 应用程序，该程序实现了前面例子中包含的大多数技术，它有一个标准结构，由以下文件组成：

• `index.html` 是用户最初所载入的文件，其中包括对 `friendly.php` 发出异步请求的 JavaScript 代码。

• `friendly.css` 是该应用程序中所使用的 CSS 类型的文件。

• `friendly.js` 是与 `index.html` 一起在客户端载入的 JavaScript 文件，它将异步请求发送给名为 `friendly.php` 的 PHP 脚本，完成胖客户接口要求的各种功能。

- friendly.php 是与 index.html 在同一台服务器上的 PHP 脚本，通过 index.html 中的 JavaScript 代码异步请求服务器端功能。由于在客户端运行时，JavaScript 代码不一定被允许访问其他服务器，因此这些文件保存在同一台服务器上是非常重要的。大多数时候，friendly.php 将使用 PHP 文件 friendly.class.php 中的功能来完成其任务。

- friendly.class.php 是一个包含 Friendly 类的 PHP 脚本，Friendly 类中包括用来支持 friendly.php 功能的业务逻辑和数据库操作。

- config.php 用于保存应用程序的全局配置选项，如数据库连接等。

- error\_handler.php 内含有错误处理机制，将文本格式出错信息改为可读性更强的格式。

Friendly 程序会在设置的时间间隔（默认为 5 秒钟）内，从 MySQL 练习时所创建的“users”表中读取两条随机记录，并从本章前面用过的随机数生成服务器中读取一个随机数。服务器使用该数据给出类似于“User paula works with user emilian at project #33”的信息，这些信息将被客户端读取并显示出来，如图 3.21 所示。

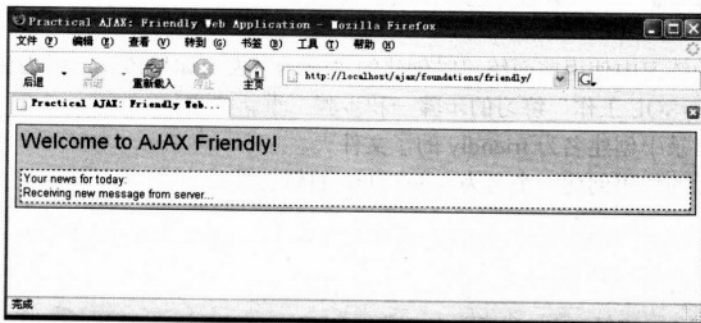


图 3.21 Friendly Web 应用程序

当出现异步请求时程序将显示“Reading the new message from server”，出现这个信息的原因是在模拟更复杂的服务器端功能时，服务器增加了一个人工延迟。

在出错时，该程序被配置为显示一个详细的出错信息。以便调试，如图 3.22 所示，或者是如图 3.23 所示的更友好的错误信息。

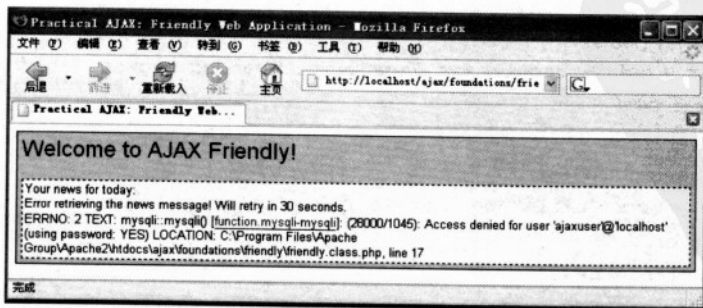


图 3.22 详细的出错页面——丢失数据库密码

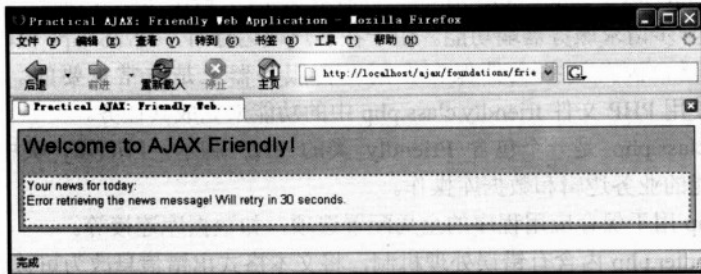


图 3.23 友好的出错页面

既然知道了要做什么，下面开始做练习。

## 》》》 实现步骤——建立友好的应用程序

(1) 下面的练习中使用前面练习时创建的“users”表，如果该表已经丢失，请按前面的“使用 PHP 和 MySQL 工作”练习的步骤一和步骤二重新建立。

(2) 在根目录中创建名为 friendly 的子文件夹。

(3) 用下面的代码创建一个名为 index.html 的新文件：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/
xhtml11/DTD/xhtml11.dtd">
<html>
  <head>
    <title>Practical Ajax: Friendly Web Application</title>
    <link href="friendly.css" rel="stylesheet" type="text/css"/>
    <script type="text/javascript" src="friendly.js"></script>
  </head>
  <body onload="process()">
    <noscript>
      <strong>
        This example requires a JavaScript-enabled browser!<br/><br/>
      </strong>
    </noscript>
    <div class="project">
      <span class="title">Welcome to Ajax Friendly!</span>
      <br/><br/>
      <div class="news">
        Your news for today:
        <div id="myDivElement" />
      </div>
    </div>
  </body>
</html>
```



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

(4) 加入一个名为 `friendly.css` 的新文件:

```
body
{
  font-family: Arial, Helvetica, sans-serif;
  font-size: small;
  background-color: #fffccc;
}

input
{
  margin-bottom: 3px;
  border: #000099 1px solid;
}

.title
{
  font-size: x-large;
}

div.project
{
  background-color: #99ccff;
  padding: 5px;
  border: #000099 1px solid;
}

div.news
{
  background-color: #ffbb8;
  padding: 2px;
  border: 1px dashed;
}
```

(5) 现在加入 JavaScript 源文件 `friendly.js`:

```
//保留一个 XMLHttpRequest 请求
```

```
var xmlHttp = createXmlHttpRequestObject();
//保留远程服务器地址和参数
var serverAddress = "friendly.php?action=GetNews";
//确定访问服务器频率的变量
var updateInterval = 5; //how many seconds to wait to get new message
var errorRetryInterval = 30; //seconds to wait after server error
//当值为真时,显示详细的出错信息
var debugMode = true;
//创建一个XMLHttpRequest请求
function createXmlHttpRequestObject()
{
    //为XMLHttpRequest对象存储索引
    var xmlHttp;
    //用于除了IE6及其更早期的版本以外的所有浏览器
    try
    {
        //试图创建一个XMLHttpRequest对象
        xmlHttp = new XMLHttpRequest();
    }
    catch(e)
    {
        //用于IE6及其早期版本
        var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
                                           "MSXML2.XMLHTTP.5.0",
                                           "MSXML2.XMLHTTP.4.0",
                                           "MSXML2.XMLHTTP.3.0",
                                           "MSXML2.XMLHTTP",
                                           "Microsoft.XMLHTTP");

        //尝试每个程序直到其中一个能够运行
        for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
        {
            try
            {
                //创建XMLHttpRequest对象
                xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
            }
            catch (e) {}
        }
    }
    //返回被创建的对象或显示一个出错信息
```

```
    if (!xmlHttp)
        alert("Error creating the XMLHttpRequest object.");
    else
        return xmlHttp;
}

//在页面上显示一个新信息的函数
function display($message)
{
    //获得页面上<div>元素的索引
    myDiv = document.getElementById("myDivElement");
    //显示信息
    myDiv.innerHTML = $message + "<br/>";
}

//显示出错信息的函数
function displayError($message)
{
    //显示出错信息, 当 debugMode 为真时显示更多的技术细节
    display("Error retrieving the news message! Will retry in " +
        errorRetryInterval + " seconds." +
        (debugMode ? "<br/>" + $message : ""));
    //重新排序
    setTimeout("process();", errorRetryInterval * 1000);
}

//异步调用服务器
function process()
{
    //仅在 xmlHttp 不为空时程序继续
    if (xmlHttp)
    {
        //尝试连接服务器
        try
        {
            //remove this line if you don't like the 'Receiving...' message
            display("Receiving new message from server...")
            //make asynchronous HTTP request to retrieve new message
            xmlHttp.open("GET", serverAddress, true);
            xmlHttp.onreadystatechange = handleGettingNews;
```

```
        xmlhttp.send(null);
    }
    catch(e)
    {
        displayError(e.toString());
    }
}

//当 HTTP 请求状态改变时调用函数
function handleGettingNews()
{
    //当 readyState 为 4 时, 准备接收服务器响应
    if (xmlhttp.readyState == 4)
    {
        //仅在 HTTP 状态为"OK"时程序继续
        if (xmlhttp.status == 200)
        {
            try
            {
                //根据服务器的响应进行处理
                getNews();
            }
            catch(e)
            {
                //显示出错信息
                displayError(e.toString());
            }
        }
        else
        {
            //显示出错信息
            displayError(xmlhttp.statusText);
        }
    }
}

//处理来自于服务器的响应
function getNews()
{
```

```

//重新接收服务器的响应
var response = xmlhttp.responseText;
//服务器错误?
if (response.indexOf("ERRNO") >= 0
    || response.indexOf("error") >= 0
    || response.length == 0)
    throw(response.length == 0 ? "Server error." : response);
//显示该信息
display(response);
//重新排序
setTimeout("process();", updateInterval * 1000);
}

```

(6) 现在开始写服务器端脚本，开始创建 friendly.php:

```

<?php
//载入出错处理模块
require_once('error_handler.php');
require_once('friendly.class.php');
//确定用户浏览器不会缓存结果
header('Expires: Wed, 23 Dec 1980 00:30:00 GMT'); //time in the past
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');
header('Cache-Control: no-cache, must-revalidate');
header('Pragma: no-cache');
//读入行为参数
$action = $_GET['action'];
//get news方法
if ($action == 'GetNews')
{
    //建立一个新的 Friendly 类实例
    $friendly = new Friendly();
    //使用 Friendly 的功能重新接收新信息
    $news = $friendly->getNews();
    //显示被客户端读取的信息
    echo $news;
}
else
{
    echo 'Communication error: server doesn\'t understand command.';
}
?>

```

(7) 用下面的内容创建 `friendly.class.php` 脚本:

```
<?php
//载入出错处理序列
require_once ('error_handler.php');
//载入配置
require_once ('config.php');

//存储 Friendly web 应用功能的类
class Friendly
{
    //存储数据库连接
    private $mMysqli;

    //构造函数打开数据库连接
    function __construct()
    {
        $this->mMysqli = new mysqli(DB_HOST, DB_USER, DB_PASSWORD,
                                   DB_DATABASE);
    }

    //生成 news 信息
    public function getNews()
    {
        //存储新闻行
        $news = 'No news for today.';
        //SQL 用来从数据库选择两个随机用户
        $query = 'SELECT user_name FROM users ' .
                'ORDER BY RAND() ' .
                'LIMIT 2';

        //执行查询
        $result = $this->mMysqli->query($query);
        //检索 user 表的行
        $row1 = $result->fetch_array(MYSQLI_ASSOC);
        $row2 = $result->fetch_array(MYSQLI_ASSOC);
        //关闭输入流
        $result->close();
        //生成 news
        if (!$row1 || !$row2)
        {
```

```

    $news = 'The project needs more users!';
}
else
{
//建立 HTML 格式的 news 信息
    $name1 = '<b>' . $row1['user_name'] . '</b>';
    $name2 = '<b>' . $row2['user_name'] . '</b>';
    $randNum = $this->getRandomNumber();
    $news = 'User ' . $name1 . ' works with user ' . $name2 .
        ' at project #' . $randNum . '.';
}
//输出 news 行
return $news;
}

//在 1 和 100 之间返回一个随机数
private function getRandomNumber()
{
    //执行延迟 1/4 秒
    usleep(250000);
    //保留远程服务器地址与参数
    $serverAddress = 'http://www.random.org/cgi-bin/randnum';
    $serverParams = 'num=1&min=1&max=100';
    //重新获得来自远程服务器的随机数
    $randomNumber = file_get_contents($serverAddress . '?' .
        $serverParams);

    //输出随机数
    return trim($randomNumber);
}

//用析构函数关闭数据库连接
function __destruct()
{
    $this->mMysqli->close();
}
}
?>

```

(8) 加入配置文件 config.php:

```
<?php
```

```
//定义数据库连接数据
define('DB_HOST', 'localhost');
define('DB_USER', 'ajaxuser');
define('DB_PASSWORD', 'practical');
define('DB_DATABASE', 'ajax');
?>
```

(9) 最后，加入出错处理脚本 `error_handler.php`:

```
<?php
//设置用户出错处理方法为 error_handler
set_error_handler('error_handler', E_ALL);
//出错处理函数
function error_handler($errNo, $errStr, $errFile, $errLine)
{
    //清除已生成的所有输出
    if(ob_get_length()) ob_clean();
    //输出出错信息
    $error_message = 'ERRNO: ' . $errNo . chr(10) .
                    'TEXT: ' . $errStr . chr(10) .
                    'LOCATION: ' . $errFile .
                    ', line ' . $errLine;
    echo $error_message;
    //阻止处理任何更多的 PHP 脚本
    exit;
}
?>
```

(10) 载入 `http://localhost/ajax/foundations/friendly/`。

## 程序说明

程序实现的大多数原理都曾在本书前面提到过，因此此处只从客户端代码开始简单分析新增的部分。`index.html` 中的新增内容是由使用 `<noscript>` 的部分组成，该元素为那些不支持 JavaScript 或禁止 JavaScript 运行的浏览器提供最低的支持：

```
<body onload="process()">
  <noscript>
    <strong>
      This example requires a JavaScript-enabled browser!<br/><br/>
    </strong>
  </noscript>
```



支持 JavaScript 的浏览器会略过<noscript>与</noscript>之间的全部内容，而不支持 JavaScript 的浏览器将分析并显示那些 HTML 代码。

客户端的 JavaScript 文件 friendly.js 有一些新增的内容。

- 将处理显示用户信息的公有功能分组，处理用户信息显示的分到 display 与 displayError 函数组。但 displayError 只有在 debugMode 为真时显示信息（变量 debugMode 在文件的开始部分定义）。

- 某处抛出异常后，在 catch 模块中调用 displayError，并使用 setTimeout 重新启动生成服务器请求的序列。可以设置在修改 errorRetryInterval 变量后发生错误时，在发出一个新的服务器请求前脚本的等待时间。

- 修改 updateInterval 变量可以改变新消息显示的频率。

- getNews()中有一种十分简单的机制来检查从服务器收到的文本是一个服务器端错误而不是我们所要等待的信息。这种机制验证响应是否包含“ERRNO”（由服务器端的常规错误处理生成的），或者是“errors”（在致命错误或解析错误时由 PHP 自动产生），或响应是否为空（如果在 php.ini 中设置 displayErrors 操作为“off”，就没有错误文本产生）。上面的任何一种情况都会手动抛出一个错误，由错误处理机制接收，并将出错信息通知给用户。

在服务器端，所有动作都从客户端调用的 friendly.php 开始，friendly.php 中最重要的部分是在由 friendly.class.php 定义的 Friendly 类中创建一个新实例，并调用 getNews 方法：

```
//读入行为参数
$action = $_GET['action'];
//get news 方法
if ($action == 'GetNews')
{
    //建立一个新 Friendly 类的实例
    $friendly = new Friendly();
    //使用 Friendly 的功能重新接收新信息
    $news = $friendly->getNews();
    //显示客户端读取的信息
    echo $news;
}
```

服务器端的主要功能在 friendly.class.php 中，由其激活 friendly.php 完成整个工作中最关键的部分，在 friendly.class.php 中可以找到 Friendly 类，其中包括以下 4 个成员。

- \$mMysqli: 一个私有域，用来存储在对象生命周期内打开的数据库连接。
- \_\_construct(): 类的构造函数，通过打开数据库连接初始化 \$mMysqli，由于在类的实例创建时构造函数会自动运行，因此可以假定在类的所有方法中连接都安全可用。
- \_\_destruct(): 类的析构函数，用来关闭数据库连接，当类中的实例被撤销时自动运行。
- getRandomNumber(): 这是一个用于返回随机数的 private 帮助方法，private 方法不能

在创建类的实例的程序中调用，这意味着它只提供内部功能。`getRandomNumber` 中的代码与前面的练习差不多，同样调用外部 `random.org` 服务器，以便得到一个新随机数。PHP 函数 `usleep` 用来人工加入 1/4 秒的延迟，保证 “Receiving new message from server...” 信息在客户端的显示时间稍长一些。

- `getNews()`: 这是一个 `public` 方法，外部程序访问可以获得一个新 “news” 信息。该方法从数据库获得两个随机用户名，使用 `getRandomNumber` 方法获取一个随机数，并给出诸如 “User *x* works with user *y* at project #*z* ” 类的信息（的确有些缺乏想象力，不过笔者实在想不出更有趣的了——对不起!）。注意 `$this` 这个特殊的对象，它用于访问 `$mMysqli` 和 `getRandomNumber()`，类成员仅仅能被使用类中的实例以及 PHP 中 `$this` 所指向的当前实例访问。

## 3.8 小结

非常有希望！我们已经愉快做完本章中给出的几个小例子，更多的在后面的章节中。本章简要介绍了一个典型的 Ajax 应用程序如何使用服务器端的技术，练习了包括简单的服务器功能在内的几个实例，PHP 在这些功能方面表现得很完美。本章还简单介绍了数据库的基础知识，并且通过本书创建的第一个表练习了一些数据库的简单操作。

在下面的章节中，将讲述更有趣的例子，使用更高级的代码执行这些功能。第 4 章中将建立支持 Ajax 的表单验证页面，保证即使客户端不支持 JavaScript 和 Ajax 也能安全工作。



验证输入数据是高质量安全软件应用的必要保证。在 Web 应用环境下，验证是尤其敏感的一部分，因为应用可以随便访问，而访问者的方式和意图却多种多样。

验证不是一个可以开玩笑的事情，因为非法的数据存在破坏应用功能的潜在危险，甚至可以使最敏感的应用领域的数据库崩溃。

输入数据的验证是指检查用户输入的数据是否和先前定义好的规则是否相符。这些规则是根据应用的商业标准来建立的。例如，如果要求日期按照 YYYY-MM-DD 格式输入，那么“Februry 28”这个日期就被认为是无效的。E-mail 地址和电话号码也是这样的例子，它们需要按照有效格式进行检查。

**⚠ 注意** 在发布应用的软件需求文档中仔细定义好输入数据的验证规则，然后使用这些规则对数据进行一致性验证。

在 Ajax 之前，Web 表单验证是在整个表单提交后由服务端来实现的。在某些情况下，客户端也有一些 JavaScript 代码来执行简单的验证，比如检查 E-mail 地址是否有效或者用户名是否已经输入。

传统的 Web 表单验证技术存在的问题有：

- 服务端验证表单受 HTTP 的限制，因为 HTTP 是一种无状态的协议。除非写一种特殊代码来解决这一问题，否则在提交一个含有非法数据的网页时，给用户返回的是一个需要重新填写的空白表单。

- 在提交一个网页时，用户需要等待整个网页加载。因为填写表单时产生的每个错误，都会导致一个新网页重载。

在本章将创建一个不仅实现优秀的传统技术并且增加了 Ajax 风格的表单验证，从而建立更加友好和响应更加敏捷的表单。

即使实现了 Ajax 验证，服务器端验证也是强制的。因为服务器是防御非法数据的最后一关。到达客户端的 JavaScript 代码，不仅通过浏览器设置能使其永久失效，而且很容易修改或者是绕过它们。

这一章的代码可以在线核对，请访问 <http://ajaxphp.packtpub.com>。

## 4.1 实现 Ajax 表单验证

这一章将建立的表单验证应用既实现了在表单提交后通过服务器端验证表单数据，又实现了在用户浏览表单时的 Ajax 验证。最后的验证是在服务端执行的，如图 4.1 所示。

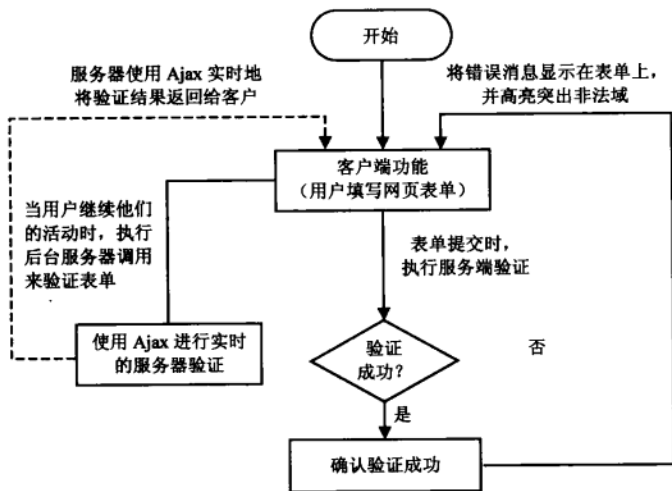


图 4.1 用户继续活动时，验证被无缝地执行

表单提交时进行最后的服务端验证是必须的。如果有人浏览器设置中使 JavaScript 失效，客户端的 Ajax 验证将不再起作用。这会暴露敏感数据，从而允许恶意访问者破坏支持服务器的重要数据（例如通过 SQL 注入）。



通常在服务器上验证用户输入。

如图 4.2 所示，要建立的应用于验证注册表单，它使用了 Ajax 验证（客户端）和典型的服务端验证。

- Ajax 风格——当光标从表单上移开时，域值被发送到服务器，数据验证后返回结果（0 代表失败，1 代表成功）。如果验证失败，将显示一个友好的错误消息，向用户通报这个域的验证结果，如图 4.3 所示。

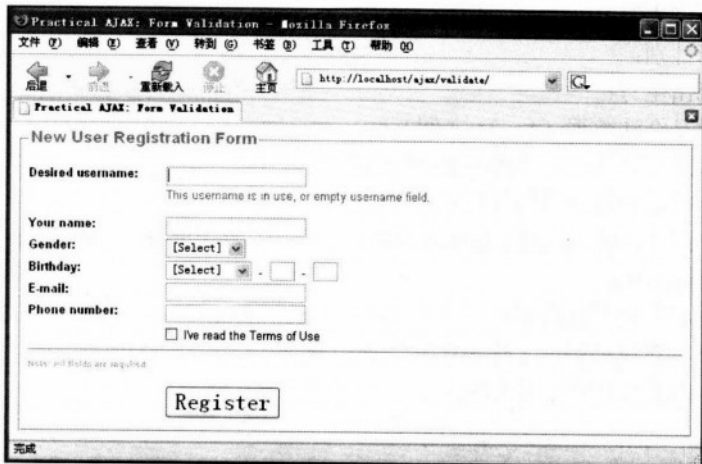


图 4.2 用户注册表单

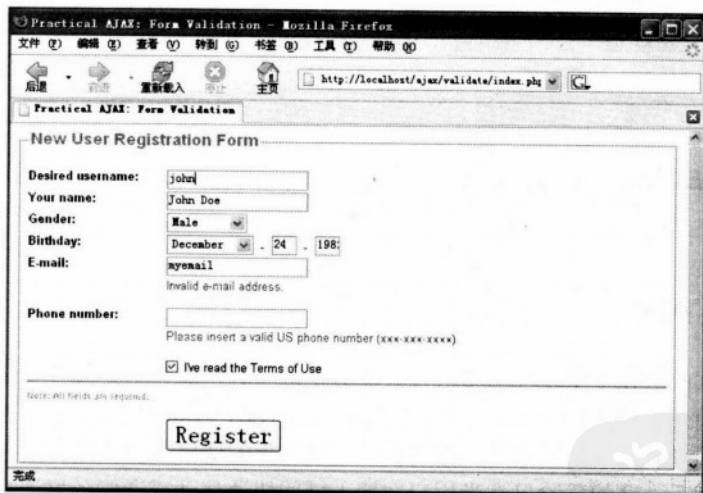


图 4.3 进行表单验证

• PHP 风格——当整个表单提交时，这是在服务器上按照某些规则检查用户输入常用的验证方法。如果没有发现任何错误，并且输入数据合法，浏览器被重定向到一个成功页面，如图 4.4 所示。如果验证失败，用户被返回到有高亮显示的非法域的表单页，如图 4.3 所示。

不论是 Ajax 验证，还是 PHP 验证都要按照以下规则来检查输入数据。

- 用户名不能是数据库中存在的；

- 名字域不能为空;
- 必须选择性别;
- 必须选择出生月;
- 生日必须是合法数据 (1~31 之间);
- 出生年必须是合法年 (1900~2000 之间);
- 日期必须考虑到每个月的天数;
- E-mail 地址必须按合法的 E-mail 格式书写, 比如 filip@yahoo.co.uk 或者 cristian@

subdomain.domain..com;

- 电话号码必须按照标准 US 格式书写: XXX-XXX-XXXX;
- 必须选择“我已经阅读了使用条款”这一项。

从下面的屏幕截图中观察应用的执行情况。

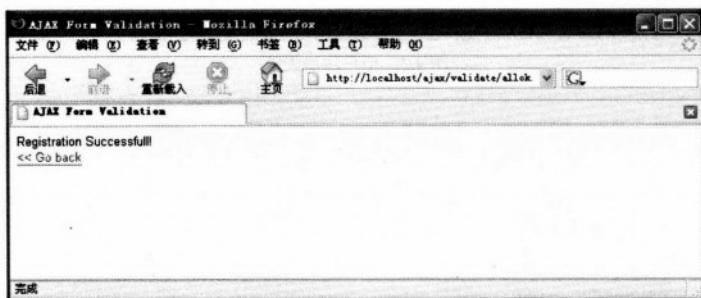


图 4.4 成功提交

## 线程安全的 Ajax

如果一个代码块在多个线程同时执行时能够正确运行, 就说它是线程安全的。本章讲述的第一个例子中, 外部因素——用户——直接影响了 Ajax 请求。每次用户把光标移开输入框或者更改一个选择时, 我们需要向服务器发送一个异步的请求来验证输入数据。

这项技术背后隐藏的危险只有当用户在输入域间快速移动或者服务器连接很慢时才会显现出来。在这些情况下, Web 应用将试图通过一个 XMLHttpRequest 对象来创建新的服务器请求, 这个 XMLHttpRequest 对象仍然忙于等待先前请求的响应 (这将产生一个错误并且应用会完全停止运行)。

根据目前的状况, 这个问题理想的解决方案是:

- 为每个发给服务器的消息创建一个新的 XMLHttpRequest 实例。这种方法很容易实现, 但是如果多个请求同时发送, 它会降低服务器的性能, 并且不能保证按顺序收到响应。
- 在一个队列中记录消息, 当 XMLHttpRequest 对象能够创建新请求时再发送出去。这些请求是以期望的顺序创建的。在重视消息顺序的应用中, 队列的使用尤为重要。

- 计划在某一段具体时间后自动地重试发送请求。这个方法与队列的使用方法很相似，因为您不会一次向服务器发送多个请求。但是它不仅不能保证请求的顺序，也不能保证响应接收的顺序。

- 忽略消息。

在这一章，本书第一次实现一个消息队列。当用户移开输入元素时，就会在队列中加入一个验证该元素值的消息。当 XMLHttpRequest 对象明确地创建一个新请求时，它从队列中取出第一个消息。

这个队列是先进先出（FIFO）的结构，能保证消息按照正确的顺序发送。想感受一下这是如何工作的，可以进入本章的演示页（或者实现其代码），多次快速按 Tab 键，然后等待观察验证响应是如何逐个显示的。

注意，解决这些问题仅仅是搞清楚不受您控制的元素能引发服务器请求。另外，就像第 3 章中的 Friendly 应用，是在接收到响应后初始化新请求的，而实现线程安全代码不会有太大的差别。

现在到编码的时候了！

## 》》》 实现步骤——Ajax 表单验证

### 提示

如果阅读了前面的章节，您就应该已经建立好了用户表。如果是，可以跳过步骤（1）和步骤（2）。

（1）连接 Ajax 数据库，用以下代码创建一个命名为 users 的表。

```
CREATE TABLE users
(
  user_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  user_name VARCHAR(32) NOT NULL,
  PRIMARY KEY (user_id)
);
```

（2）执行下面的 INSERT 命令，用一些示例数据来填充 users 表（user\_id 是自动增加的列，它的值由数据库自动生成）。

```
INSERT INTO users (user_name) VALUES ('bogdan');
INSERT INTO users (user_name) VALUES ('filip');
INSERT INTO users (user_name) VALUES ('mihai');
INSERT INTO users (user_name) VALUES ('emilian');
INSERT INTO users (user_name) VALUES ('paula');
INSERT INTO users (user_name) VALUES ('cristian');
```

(3) 在 Ajax 文件夹下创建一个新文件夹并命名为 `validate`。

(4) 现在开始按照介绍的顺序编写代码。创建一个名为 `validate.css` 的文件，加入以下代码：

```
body
{
    font-family: Arial, Helvetica, sans-serif;
    font-size: 0.8em;
    color: #000000;
}

label
{
    float: left;
    width: 150px;
    font-weight: bold;
}

input, select
{
    margin-bottom: 3px;
}

.button
{
    font-size: 2em;
}

.left
{
    margin-left: 150px;
}

.txtFormLegend
{
    color: #777777;
    font-weight: bold;
    font-size: large;
}
```



```

.txtSmall
{
    color: #999999;
    font-size: smaller;
}

.hidden
{
    display: none;
}

.error
{
    display: block;
    margin-left: 150px;
    color: #ff0000;
}

```

(5) 创建一个名为 `index_top.php` 的文件，并加入以下代码。主页面 `index.php` 将加载这个脚本。

```

<?php
//激活 PHP 会话
session_start();
//建立 HTML <option> 标签
function buildOptions($options, $selectedOption)
{
    foreach ($options as $value => $text)
    {
        if ($value == $selectedOption)
        {
            echo '<option value="' . $value .
                '" selected="selected">' . $text . '</option>';
        }
        else
        {
            echo '<option value="' . $value . '"' . $text . '</option>';
        }
    }
}
//初始化性别选项数组

```

```
$genderOptions = array("0" => "[Select]",
                       "1" => "Male",
                       "2" => "Female");

//初始化月份选项数组
$monthOptions = array("0" => "[Select]",
                      "1" => "January",
                      "2" => "February",
                      "3" => "March",
                      "4" => "April",
                      "5" => "May",
                      "6" => "June",
                      "7" => "July",
                      "8" => "August",
                      "9" => "September",
                      "10" => "October",
                      "11" => "November",
                      "12" => "December");

//初始化一些会话变量,防止PHP抛出注意事项
if (!isset($_SESSION['values']))
{
    $_SESSION['values']['txtUsername'] = '';
    $_SESSION['values']['txtName'] = '';
    $_SESSION['values']['selGender'] = '';
    $_SESSION['values']['selBthMonth'] = '';
    $_SESSION['values']['txtBthDay'] = '';
    $_SESSION['values']['txtBthYear'] = '';
    $_SESSION['values']['txtE-mail'] = '';
    $_SESSION['values']['txtPhone'] = '';
    $_SESSION['values']['chkReadTerms'] = '';
}
if (!isset($_SESSION['errors']))
{
    $_SESSION['errors']['txtUsername'] = 'hidden';
    $_SESSION['errors']['txtName'] = 'hidden';
    $_SESSION['errors']['selGender'] = 'hidden';
    $_SESSION['errors']['selBthMonth'] = 'hidden';
    $_SESSION['errors']['txtBthDay'] = 'hidden';
    $_SESSION['errors']['txtBthYear'] = 'hidden';
    $_SESSION['errors']['txtE-mail'] = 'hidden';
}
```

```

$_SESSION['errors']['txtPhone'] = 'hidden';
$_SESSION['errors']['chkReadTerms'] = 'hidden';
}
?>

```

(6) 创建 index.php 文件，加入以下代码：

```

<?php
require_once ('index_top.php');
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Practical Ajax: Form Validation</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <link href="validate.css" rel="stylesheet" type="text/css" />
    <script type="text/javascript" src="validate.js"></script>
  </head>

  <body onload="setFocus();">
    <fieldset>
      <legend class="txtFormLegend">New User Registration Form</legend>
      <br />
      <form name="frmRegistration" method="post"
        action="validate.php?validationType=php">

        <!-- Username -->
        <label for="txtUsername">Desired username:</label>
        <input id="txtUsername" name="txtUsername" type="text"
          onblur="validate(this.value, this.id)"

          value="<?php echo $_SESSION['values']['txtUsername'] ?>" />
        <span id="txtUsernameFailed"
          class="<?php echo $_SESSION['errors']['txtUsername'] ?>">
          This username is in use, or empty username field.
        </span>
        <br />

        <!-- Name -->
        <label for="txtName">Your name:</label>

```

```
<input id="txtName" name="txtName" type="text"
      onBlur="validate(this.value, this.id)"
      value="<?php echo $_SESSION['values']['txtName'] ?>" />
<span id="txtNameFailed"
      class="<?php echo $_SESSION['errors']['txtName'] ?>"
      Please enter your name.
</span>
<br />

<!-- Gender -->
<label for="selGender">Gender:</label>
<select name="selGender" id="selGender"
        onBlur="validate(this.value, this.id)"
        <?php buildOptions($genderOptions,
                          $_SESSION['values']['selGender']); ?>
</select>
<span id="selGenderFailed"
      class="<?php echo $_SESSION['errors']['selGender'] ?>"
      Please select your gender.
</span>
<br />

<!-- Birthday -->
<label for="selBthMonth">Birthday:</label>

<!-- Month -->
<select name="selBthMonth" id="selBthMonth"
        onBlur="validate(this.value, this.id)"
        <?php buildOptions($monthOptions,
                          $_SESSION['values']['selBthMonth']); ?>
</select>
&nbsp;-&nbsp;
<!-- Day -->
<input type="text" name="txtBthDay" id="txtBthDay" maxlength="2"
      size="2"
      onBlur="validate(this.value, this.id)"
      value="<?php echo $_SESSION['values']['txtBthDay'] ?>" />
&nbsp;-&nbsp;
<!-- Year -->
<input type="text" name="txtBthYear" id="txtBthYear" maxlength="4"
```

```

        size="2" onblur="validate(document.getElementById
('selBthMonth') .options[document.getElementById('selBthMonth').selectedIndex].
value+'#' +document.getElementById('txtBthDay').value+'#' +this.value, this.id)"
        value="<?php echo $_SESSION['values']['txtBthYear'] ?>" />

<!-- Month, Day, Year validation -->
<span id="selBthMonthFailed"
    class="<?php echo $_SESSION['errors']['selBthMonth'] ?>"
    Please select your birth month.
</span>
<span id="txtBthDayFailed"
    class="<?php echo $_SESSION['errors']['txtBthDay'] ?>"
    Please enter your birth day.
</span>
<span id="txtBthYearFailed"
    class="<?php echo $_SESSION['errors']['txtBthYear'] ?>"
    Please enter a valid date.
</span>
<br />

<!-- E-mail -->
<label for="txtE-mail">E-mail:</label>
<input id="txtE-mail" name="txtE-mail" type="text"
    onblur="validate(this.value, this.id)"
    value="<?php echo $_SESSION['values']['txtE-mail'] ?>" />
<span id="txtE-mailFailed"
    class="<?php echo $_SESSION['errors']['txtE-mail'] ?>"
    Invalid e-mail address.
</span>
<br />

<!-- Phone number -->
<label for="txtPhone">Phone number:</label>
<input id="txtPhone" name="txtPhone" type="text"
    onblur="validate(this.value, this.id)"
    value="<?php echo $_SESSION['values']['txtPhone'] ?>" />
<span id="txtPhoneFailed"
    class="<?php echo $_SESSION['errors']['txtPhone'] ?>"
    Please insert a valid US phone number (xxx-xxx-xxxx).

```

```

        </span>
        <br />

        <!-- Read terms checkbox -->
        <input type="checkbox" id="chkReadTerms" name="chkReadTerms"
            class="left"
            onblur="validate(this.checked, this.id)"
            <?php if ($_SESSION['values']['chkReadTerms'] == 'on')
                echo 'checked="checked"' ?> />
        I've read the Terms of Use
        <span id="chkReadTermsFailed"
            class="<?php echo $_SESSION['errors']['chkReadTerms'] ?>">
            Please make sure you read the Terms of Use.
        </span>

        <!-- End of form -->
        <hr />
        <span class="txtSmall">Note: All fields are required.</span>
        <br /><br />
        <input type="submit" name="submitbutton" value="Register"
            class="left button" />
    </form>
</fieldset>
</body>
</html>

```

(7) 创建 `allok.php` 文件，加入以下代码：

```

<?php
    //清除保存在会话中的数据
    session_start();
    session_destroy();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>Ajax Form Validation</title>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <link href="validate.css" rel="stylesheet" type="text/css" />
    </head>

```

```

<body>
  Registration Successfull!<br />
  <a href="index.php" title="Go back">&lt;&lt; Go back</a>
</body>
</html>

```

(8) 创建 validate.js 文件。这个文件实现客户端功能，包含 Ajax 请求：

```

//建立一个 XMLHttpRequest 实例
var xmlhttp = createXmlHttpRequestObject();
//建立远程服务器地址
var serverAddress = "validate.php";
//当设置为 true 时，显示详细的错误消息
var showErrors = true;
//初始化验证请求的缓存
var cache = new Array();

//创建一个 XMLHttpRequest 实例
function createXmlHttpRequestObject()
{
  //将要存储 XMLHttpRequest 对象的索引
  var xmlhttp;
  //这将在所有浏览器上工作，除了 IE6 和老版本
  try
  {
    //尝试创建 XMLHttpRequest 对象
    xmlhttp = new XMLHttpRequest();
  }
  catch(e)
  {
    //假定是 IE6 或者老版本
    var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
                                     "MSXML2.XMLHTTP.5.0",
                                     "MSXML2.XMLHTTP.4.0",
                                     "MSXML2.XMLHTTP.3.0",
                                     "MSXML2.XMLHTTP",
                                     "Microsoft.XMLHTTP");

    //尝试每一个 id 直到一个工作为止
    for (var i=0; i<XmlHttpVersions.length && !xmlhttp; i++)
    {

```

```
    try
    {
        //尝试创建 XMLHttpRequest 对象
        xmlhttp = new ActiveXObject(XmlHttpVersions[i]);
    }
    catch (e) {} //忽略潜在的错误
}
}
//返回已创建的对象或者显示一个错误消息
if (!xmlhttp)
    displayError("Error creating the XMLHttpRequest object.");
else
    return xmlhttp;
}

//显示错误消息的函数
function displayError($message)
{
    //如果 showErrors 是 false 就忽略错误
    if (showErrors)
    {
        //关闭错误显示
        showErrors = false;
        //显示错误消息

        alert("Error encountered: \n" + $message);
        //10 秒后重试验证
        setTimeout("validate()", 10000);
    }
}

//函数处理每个表单域的验证
function validate(inputValue, fieldID)
{
    //如果 xmlhttp 不是空的就继续
    if (xmlhttp)
    {
        //如果接收到非空参数, 就以要发送到服务器进行验证的请求串格式加到缓存
        if (fieldID)
        {
```



```
//进行编码以便安全将它们添加到一个HTTP请求查询串
inputValue = encodeURIComponent(inputValue);
fieldID = encodeURIComponent(fieldID);
//添加值到队列中
cache.push("inputValue=" + inputValue + "&fieldID=" + fieldID);
}
//尝试连接到服务器
try
{
    //仅在XMLHttpRequest对象空闲并且缓存不为空时继续
    if ((xmlHttp.readyState == 4 || xmlHttp.readyState == 0)
        && cache.length > 0)
    {
        //从缓存获取一个新的参数集
        var cacheEntry = cache.shift();
        //构造一个服务器请求来验证取出的数据
        xmlHttp.open("POST", serverAddress, true);
        xmlHttp.setRequestHeader("Content-Type",
            "application/x-www-form-urlencoded");
        xmlHttp.onreadystatechange = handleRequestStateChange;
        xmlHttp.send(cacheEntry);
    }
}
catch (e)
{
    //当连接服务器失败时显示一个错误
    displayError(e.toString());
}
}

//处理HTTP响应的函数
function handleRequestStateChange()
{
    //当readyState是4时,读取服务器响应
    if (xmlHttp.readyState == 4)
    {
        //HTTP状态是"OK"时继续
        if (xmlHttp.status == 200)
        {
```

```
try
{
    //从服务器读取响应
    readResponse();
}
catch(e)

{
    //显示错误消息
    displayError(e.toString());
}
}
else
{
    //显示错误消息
    displayError(xmlHttp.statusText);
}
}
}

//读取服务器响应
function readResponse()
{
    //接收服务器响应
    var response = xmlHttp.responseText;
    //服务器错误?
    if (response.indexOf("ERRNO") >= 0
        || response.indexOf("error:") >= 0
        || response.length == 0)
        throw(response.length == 0 ? "Server error." : response);
    //以XML格式获取响应(假定响应是有效的XML)
    responseXml = xmlHttp.responseXML;
    //获取文档元素
    xmlDoc = responseXml.documentElement;
    result = xmlDoc.getElementsByTagName("result")[0].firstChild.data;
    fieldID = xmlDoc.getElementsByTagName("fieldid")[0].firstChild.data;
    //找到显示错误的HTML元素
    message = document.getElementById(fieldID + "Failed");
    //显示错误或者隐藏错误
    message.className = (result == "0") ? "error" : "hidden";
}
```

```

//再次调用 validate(), 避免仍有值遗留在缓存中
setTimeout("validate();", 500);
}

//设置光标在第一个表单域
function setFocus()
{
    document.getElementById("txtUsername").focus();
}

```

(9) 增加企业逻辑。先创建 config.php 文件, 加入以下代码:

```

<?php
//定义数据库连接数据
define('DB_HOST', 'localhost');
define('DB_USER', 'ajaxuser');
define('DB_PASSWORD', 'practical');
define('DB_DATABASE', 'ajax');
?>

```

(10) 在一个名为 error\_handler.php 的文件内创建错误处理代码:

```

<?php
//设置用户错误处理方法为 error_handler
set_error_handler('error_handler', E_ALL);

//错误处理函数
function error_handler($errNo, $errStr, $errFile, $errLine)
{
    //清除所有已经生成的输出
    if(ob_get_length()) ob_clean();

    //输出错误消息
    $error_message = 'ERRNO: ' . $errNo . chr(10) .
                    'TEXT: ' . $errStr . chr(10) .
                    'LOCATION: ' . $errFile .
                    ', line ' . $errLine;

    echo $error_message;
    //阻止处理任意 PHP 脚本
    exit;
}
?>

```

(11) PHP 脚本 validate.php 处理客户的 Ajax 调用, 同时处理表单提交时的验证:

```
<?php
//开启 PHP 会话
session_start();
//加载错误处理脚本和验证类
require_once ('error_handler.php');
require_once ('validate.class.php');

//创建新的验证对象
$validator = new Validate();

//读取验证类型 (PHP 或者 Ajax)
$validationType = '';
if (isset($_GET['validationType']))
{
    $validationType = $_GET['validationType'];
}

//Ajax 验证还是 PHP 验证?
if ($validationType == 'php')
{
    //PHP 验证由 ValidatePHP 方法执行, 返回访问者应该被重定向的网页 (如果所有数据
    //有效是 allok.php, 否则是 index.php)
    header("Location:" . $validator->ValidatePHP());
}
else
{
    //Ajax 验证由 ValidateAjax 方法执行, 结果用来组成一个返回给客户的 XML 文档
    $response =
        '<?xml version="1.0" encoding="UTF-8" standalone="yes"?>' .
        '<response>' .
        '  <result>' .
        '    $validator->ValidateAjax($_POST['inputValue'], $_POST['fieldID']) .
        '  </result>' .
        '  <fieldid>' .
        '    $_POST['fieldID'] .
        '  </fieldid>' .
        '</response>';
    //生成响应
}
```

```
if(ob_get_length()) ob_clean();
header('Content-Type: text/xml');
echo $response;
}
?>
```

(12) 支持验证功能的类叫做 `validate`，并且它嵌入在一个名为 `validate.class.php` 的脚本文件里，如下所示：

```
<?php
//加载错误处理和数据库配置
require_once ('config.php');

//支持 Ajax 和 PHP Web 表单验证的类
class Validate
{
    //存储数据库连接
    private $mMysqli;

    //构造函数打开数据库连接
    function __construct()
    {
        $this->mMysqli = new mysqli(DB_HOST, DB_USER, DB_PASSWORD, DB_DATABASE);
    }

    //析构函数关闭数据库连接
    function __destruct()
    {
        $this->mMysqli->close();
    }

    //支持 Ajax 验证，校验一个单一的值
    public function ValidateAjax($inputValue, $fieldID)
    {
        //检查哪个域应该被验证并执行验证
        switch($fieldID)
        {
            //检查用户名是否有效
            case 'txtUsername':
                return $this->validateUserName($inputValue);
            break;
        }
    }
}
```

```
//检查名字是否有效
case 'txtName':
    return $this->validateName($inputValue);
    break;

//检查性别是否已选
case 'selGender':
    return $this->validateGender($inputValue);
    break;

//检查出生月是否有效
case 'selBthMonth':
    return $this->validateBirthMonth($inputValue);
    break;

//检查出生日是否有效
case 'txtBthDay':
    return $this->validateBirthDay($inputValue);
    break;

//检查出生年是否有效
case 'txtBthYear':
    return $this->validateBirthYear($inputValue);
    break;

//检查E-mail是否有效
case 'txtE-mail':
    return $this->validateE-mail($inputValue);
    break;

//检查电话号码是否有效
case 'txtPhone':
    return $this->validatePhone($inputValue);

    break;

//检查“我已经阅读条款”复选框是否选择
case 'chkReadTerms':
    return $this->validateReadTerms($inputValue);
```

```
        break;
    }
}

//表单提交时验证所有表单域
public function ValidatePHP()
{
    //错误标志,当错误被查找到时变成1.
    $errorsExist = 0;
    //清除错误会话标志
    if (isset($_SESSION['errors']))
        unset($_SESSION['errors']);
    //默认情况下所有域被认为是有效的
    $_SESSION['errors']['txtUsername'] = 'hidden';
    $_SESSION['errors']['txtName'] = 'hidden';
    $_SESSION['errors']['selGender'] = 'hidden';
    $_SESSION['errors']['selBthMonth'] = 'hidden';
    $_SESSION['errors']['txtBthDay'] = 'hidden';
    $_SESSION['errors']['txtBthYear'] = 'hidden';
    $_SESSION['errors']['txtE-mail'] = 'hidden';
    $_SESSION['errors']['txtPhone'] = 'hidden';
    $_SESSION['errors']['chkReadTerms'] = 'hidden';

    //验证用户名
    if (!$this->validateUserName($_POST['txtUsername']))
    {
        $_SESSION['errors']['txtUsername'] = 'error';
        $errorsExist = 1;
    }

    //验证名字
    if (!$this->validateName($_POST['txtName']))
    {
        $_SESSION['errors']['txtName'] = 'error';
        $errorsExist = 1;
    }

    //验证性别
    if (!$this->validateGender($_POST['selGender']))
    {
```

```
$_SESSION['errors']['selGender'] = 'error';
$errorsExist = 1;
}

//验证出生月
if (!$this->validateBirthMonth($_POST['selBthMonth']))
{
    $_SESSION['errors']['selBthMonth'] = 'error';
    $errorsExist = 1;
}

//验证出生日
if (!$this->validateBirthDay($_POST['txtBthDay']))
{
    $_SESSION['errors']['txtBthDay'] = 'error';
    $errorsExist = 1;
}

//验证出生年和日期
if (!$this->validateBirthYear($_POST['selBthMonth'] . '#' .
                                $_POST['txtBthDay'] . '#' .
                                $_POST['txtBthYear']))
{
    $_SESSION['errors']['txtBthYear'] = 'error';
    $errorsExist = 1;
}

//验证 E-mail
if (!$this->validateE-mail($_POST['txtE-mail']))
{
    $_SESSION['errors']['txtE-mail'] = 'error';
    $errorsExist = 1;
}

//验证电话号码
if (!$this->validatePhone($_POST['txtPhone']))
{
    $_SESSION['errors']['txtPhone'] = 'error';
    $errorsExist = 1;
}
```



```

}

//验证读条款
if (!isset($_POST['chkReadTerms']) ||
    !$this->validateReadTerms($_POST['chkReadTerms']))
{
    $_SESSION['errors']['chkReadTerms'] = 'error';
    $_SESSION['values']['chkReadTerms'] = '';
    $errorsExist = 1;
}

//如果没有出现任何错误, 指向一个成功的验证网页
if ($errorsExist == 0)
{
    return 'allok.php';
}
else
{
    //如果发现错误, 保存当前用户输入
    foreach ($_POST as $key => $value)
    {
        $_SESSION['values'][$key] = $_POST[$key];
    }
    return 'index.php';
}
}

//验证用户名字(必须不为空, 而且不能是已经注册的)
private function validateUserName($value)
{
    //去除空格并转换输入值
    $value = $this->mMysqli->real_escape_string(trim($value));
    //空的用户名是无效的
    if ($value == null)
        return 0; //无效
    //检查用户名在数据库中是否存在
    $query = $this->mMysqli->query('SELECT user_name FROM users ' .
        'WHERE user_name="' . $value . '"');
    if ($this->mMysqli->affected_rows > 0)
        return '0'; //无效
}

```

```
else
    return '1'; //有效
}

//验证名字
private function validateName($value)
{
    //去除空格并转换输入值
    $value = trim($value);
    //空的用户名是无效的
    if ($value)
        return 1; //有效
    else
        return 0; //无效
}

//验证性别
private function validateGender($value)
{
    //用户必须有一个性别
    return ($value == '0') ? 0 : 1;
}

//验证出生月
private function validateBirthMonth($value)
{
    //月不能为空, 并且在 1~12 之间
    return ($value == '' || $value > 12 || $value < 1) ? 0 : 1;
}

//验证出生日
private function validateBirthDay($value)
{
    //日不能为空, 并且在 1~31 之间
    return ($value == '' || $value > 31 || $value < 1) ? 0 : 1;
}

//验证出生年和整个日期
private function validateBirthYear($value)
{
    //有效的年份在 1900~2000 之间
```

```

//获取整个日期 (mm#dd#yyyy)
$date = explode('#', $value);
//日期如果没有日、月、年就不可能是有效的
if (!$date[0]) return 0;
if (!$date[1] || !is_numeric($date[1])) return 0;
if (!$date[2] || !is_numeric($date[2])) return 0;
//检查日期
return (checkdate($date[0], $date[1], $date[2])) ? 1 : 0;
}

//验证 E-mail
private function validateE-mail($value)
{
    //验证 E-mail 格式: (*@*.*, *@*.*.*, *.*@*.*, *.*@*.*.*)
    return (!eregi('^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*
    (\.[a-z]{2,3})$', $value)) ? 0 : 1;
}

//验证电话号码
private function validatePhone($value)
{
    //有效的电话号码格式: ###-###-####
    return (!eregi('^([0-9]{3}-*[0-9]{3}-*[0-9]{4})$', $value)) ? 0 : 1;
}

//检查用户已经阅读使用条款
private function validateReadTerms($value)
{
    //有效值是'true'

    return ($value == 'true' || $value == 'on') ? 1 : 0;
}
}
?>

```

(13) 在浏览器中填写 <http://localhost/ajax/validate/index.php>, 测试脚本。

## 程序说明

Ajax 验证技术允许我们验证表单域, 同时, 如果有任何验证错误, 都会通知用户。其画

龙点睛之处在于我们做的所有这些工作没有中断用户的活动。这就叫做合理的表单验证。

这种合理的验证可以与提交表单时发生的纯服务器端 PHP 验证相结合。在服务器上, PHP 脚本 `validate.php` 在另一个在 PHP 脚本 `validate.class.php` 的协助下对这两种验证都支持。

让我们检查一下代码, 首先从处理客户端验证的脚本 `index.php` 开始。在这个验证示例中, 客户页不是简单的 HTML 文件, 而是一个 PHP 文件, 因此它的有些部分会在服务端动态生成。这是非常必要的, 因为我们想要在表单提交后并且服务端验证失败时保留表单域值。如果没有 PHP 代码的帮助, 当 `index` 页面重载后, 所有域都会变为空。

`index.php` 开始时载入脚本 `index_top.php`, 它通过调用 `session_start()` 开启一个会话, 定义了一些 `index.php` 后面要用到的变量和函数, 并初始化了一些会话变量 (`$_SESSION['values']` 和 `$_SESSION['errors']`), 避免了 PHP 发送关于变量没有初始化的通知。

注意 `index.php` 中 `body` 标签里的 `onload` 事件, 它调用了定义在 `validate.js` 中的 `setFocus()` 函数, 此函数作用是将输入光标放置在第一个表单域。

在 `index.php` 后面的部分, 将看到下面一段代码重复出现, 只有一些细微的变化。

```
<!-- Username -->
    <label for="txtUsername">Desired username:</label>
    <input id="txtUsername" name="txtUsername" type="text"
        onBlur="validate(this.value, this.id)"

        value="<?php echo $_SESSION['values']['txtUsername'] ?>" />
    <span id="txtUsernameFailed"
        class="<?php echo $_SESSION['errors']['txtUsername'] ?>">
        This username is in use, or empty username field.
    </span>
<br />
```

在验证执行失败后, 这段代码显示了一个表单域的标签并在表单域下面显示错误消息。

**注意** 在这个示例中, 我们将错误消息显示在验证域下面, 也可以在 `validate.css` 中改变错误 CSS 类的属性, 以自定义消息的位置和外观。

当用户移开输入元素时生成输入元素的 `onblur` 事件, 它通过两个参数触发 JavaScript 函数 `validate()`——域值和域 ID。这个函数将向脚本 `validate.php` 发送异步 HTTP 请求, 以处理 Ajax 验证。服务器脚本需要知道要验证哪个域以及该域的输入值是什么。

第一个页面载入时值属性对应该是空的, 但提交表单后, 它将拥有输入值, 避免表单由于验证错误而重载。我们在提交表单时使用会话变量来保存用户输入, 以避免验证失败和表单刷新。

元素 `span` 包含了验证失败时显示的错误消息。这个 `span` 最初是通过使用隐藏 CSS 类隐藏的, 但我们在验证失败时将 CSS 类改成错误显示。

在 `validate.js` 中，验证函数通过两个参数（域值和域 ID）调用 `validate.php`，向服务器发送一个 Ajax 请求。

记住 `XMLHttpRequest` 不能同时建立两个 HTTP 请求，因此如果一个对象忙于处理先前的请求，我们先保存当前请求的细节。这在连接网络或因特网慢的情况下显得尤为重要。使用 FIFO 结构的缓存系统保存请求的细节。幸运的是，JavaScript 数组类提供了我们需要的具体功能（通过它的 `push` 和 `shift` 方法），因此使用它来完成缓存目的：

```
var cache=new Array();
```

因此 `validate()` 首先向缓存增加验证数据（如果这个函数接收到任何数据）。

```
//这个函数处理任意表单域的验证
function validate(inputValue, fieldID)
{
    //仅在 xmlhttp 非空时继续
    if (xmlHttp)
    {
        //如果接收到非空的参数，就以发送到服务器进行验证的查询串的形式将它
        //们加到缓存
        if (fieldID)
        {
            //将值编码用于安全地添加它们到一个 HTTP 请求查询串
            inputValue = encodeURIComponent(inputValue);
            fieldID = encodeURIComponent(fieldID);
            //将值添加到队列
            cache.push("inputValue=" + inputValue + "&fieldID=" + fieldID);
        }
    }
}
```

这给缓存队列尾部增加了一个新元素。缓存条目由两部分组成——验证域的值和 ID，两部分用“&”分隔开。注意新元素的域 ID 不为空时才能加入缓存。当函数调用只是检查缓存是否包含未验证的元素，而不是给缓存增加新条目时，域 ID 的值是空的。

### ⚠ 注意

从缓存获取的域 ID 和值将发送到服务器进行验证。为了确保它们能成功到达目的地并且没有被修改，它们没有使用 JavaScript 的 `encodeURIComponent` 函数。这样可以向服务器安全地传送任何字母，包括通常引发问题的“&”字母。要获得更多细节，<http://xkr.us/articles/javascript/encode-compare/> 有关于 JavaScript 转义方法的精彩文章。

如果 `XMLHttpRequest` 对象是空闲的，可以初始化新的 HTTP 请求，使用 `shift()` 从缓存中获取一个新值进行验证（这个函数同时从缓存队列中移除了该条目，这对于我们来说是理想的做法）。注意这个值不是刚才用 `push` 增加的，在 FIFO 结构的缓存中，最先插入的（未验

证) 记录最先被获取。

```
//试图连接到服务器
try
{
    //仅在 XMLHttpRequest 对象空闲且缓存不为空时继续
    if ((xmlHttp.readyState == 4 || xmlHttp.readyState == 0)
        && cache.length > 0)
    {
        //从缓存获取一个新的参数集
        var cacheEntry = cache.shift();
```

如果 XMLHttpRequest 对象的状态是 0 或者 4, 它是指没有活动的请求, 我们就可以发送一个新请求。当发送新请求时, 可以使用从缓存中读出的数据, 它已经包含了格式化的询问串。

```
//构建一个服务器请求来验证抽取的数据
xmlHttp.open("POST", serverAddress, true);
xmlHttp.setRequestHeader("Content-Type",
    "application/x-www-form-urlencoded");
xmlHttp.onreadystatechange = handleRequestStateChange;
xmlHttp.send(cacheEntry);
}
```

处理服务器响应的函数叫做 `handleRequestStateChange`, 并且每次从服务器成功接收响应时依次调用 `readResponse()`。这个方法开始时先检查从服务器接收的内容是不是服务端错误报告。

```
//阅读服务器的响应
function readResponse()
{
    //接收服务器响应
    var response = xmlHttp.responseText;
    //服务器错误?
    if (response.indexOf("ERRNO") >= 0
        || response.indexOf("error:") >= 0
        || response.length == 0)
        throw(response.length == 0 ? "Server error." : response);
```

这种基本检查完成后, 就可以阅读服务器的响应, 它告诉我们其值是否有效:

```
//以 XML 的形式获取响应 (假定响应是有效的 XML)
responseXml = xmlHttp.responseXML;
```

```
//获取文档元素
xmlDoc = responseXml.documentElement;
result = xmlDoc.getElementsByTagName("result")[0].firstChild.data;
fieldID = xmlDoc.getElementsByTagName("fieldid")[0].firstChild.data;
```

根据结果，将与测试元素有关的错误消息的 CSS 类改成隐藏的（如果验证成功）或者错误的（如果验证失败）。可以用 `classname` 属性来改变元素的 CSS 类。

```
//找到显示错误的 HTML 元素
message = document.getElementById(fieldID + "Failed");
//显示错误或者隐藏错误
message.className = (result == "0") ? "error" : "hidden";
//再次调用 validate(), 避免有值留在缓存中
setTimeout("validate();", 500);
}
```

负责服务端处理的 PHP 脚本是 `validate.php`。开始时它载入错误处理脚本（`error_handler.php`）和处理数据验证的验证类（`validate.class.php`），然后寻找 `validationType` 的 GET 变量，这仅在表单提交时存在，因为表单的 `action` 属性是“`validate.php?validationType=php`”。

```
//读取验证类型（PHP 还是 Ajax?）
$validationType = '';
if (isset($_GET['validationType']))
{
    $validationType = $_GET['validationType'];
}
```

根据 `$validationType` 的值，或者执行 Ajax 验证，或者执行 PHP 验证。

```
//Ajax 验证还是 PHP 验证?
if ($validationType == 'php')
{
    //PHP 验证由 ValidatePHP 方法执行，返回访问者应该重定向页（如果所有数据
    //有效是 alok.php，否则返回到 index.php）
    header("Location:" . $validator->ValidatePHP());
}
else
{
    //Ajax 验证由 ValidateAjax 方法执行，结果形成一个 XML 文档返回给客户
    $response =
        '<?xml version="1.0" encoding="UTF-8" standalone="yes"?>' .
        '<response>' .
        '<result>' .
```

```

        $validator->ValidateAjax($_POST['inputValue'], $_POST['fieldID']) .
    '</result>' .
    '<fieldid>' .
        $_POST['fieldID'] .
    '</fieldid>' .
    '</response>';
//生成响应
if(ob_get_length()) ob_clean();
header('Content-Type: text/xml');
echo $response;
}
?>

```

如果正在处理典型的服务端验证，可以调用 `validatePHP()` 方法，它返回浏览器重定向页的名字（如果验证成功是 `allok.php`，如果失败是 `index.php`）。每个域的验证结果存储在会话中，如果 `index.php` 重新载入，它会显示这些域没有通过测试。

在 Ajax 调用的情况下，服务器产生一个定义域是否合法的响应。这个响应是一个短的 XML 文档，看起来如下：

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<response>
  <result>0</result>
  <fieldid>txtUsername</fieldid>
</response>

```

如果结果是 0，那么 `txtUsername` 就是非法的，应该相应地做上标记。如果结果是 1，那么域值就是合法的。

下一步来看一下 `validate.class.php`。这个类在构造函数中创建了一个数据库连接，并在析构函数中关闭此连接。它有两个 `public` 方法：`ValidateAjax`（处理 Ajax 验证）和 `ValidatePHP`（处理典型的服务端验证）。

Ajax 验证要求有两个参数，一个存有待验证的值（`$inputValue`），另一个存有表单域的 ID（`$fieldID`）。交换块为每个表单域都载入一个专门的验证。这个功能在验证失败时返回 0，验证成功时返回 1。

PHP 验证函数不需要参数，因为它总是验证整个表单（在表单提交后）。首先初始化 `$errorExist` 标志为 0。当域的验证失败时，这个标志被设为 1，这样我们就知道验证失败了。然后需要确认没有设置旧的会话变量，这是为了确保旧错误要清空。

之后，我们开始按照一套自创的规则来检查每个表单域。如果验证失败，增加标志值（`$errorsExist=1`），并设置会话变量，该变量将错误消息的 CSS 类设定为错误。如果到最后，`$errorsExist` 标志仍然被设置为 0，则意味着整个验证已经成功并且返回成功页的名字，并将



浏览器重定向到本页。

如果发现错误，我们保存当前用户输入到会话变量，它将在 `index.php` 填充表单时被用到（记住如果默认的话，当载入一个页面时，所有域是空的）。下面是如何保存当前用户输入：

```
foreach ($_POST as $key => $value)
{
    $_SESSION['values'][$key] = $_POST[$key];
}
```

`$_POST` 是一个存有所有表单元素名字和值的数组，可以用 `foreach` 遍历这个数组。这意味着对于 `$_POST` 数组中的每个元素，都要在 `$_SESSION['values']` 数组中创建一个新的元素。

对于 `validate.css` 没有什么特殊的内容要介绍。成功页 (`allok.php`) 也非常简单，它仅仅显示成功提交确认。

## 4.2 小结

虽然没有声称已经建立了完美的验证技术，但我们对这个概念可运行性提供了证明：一个好的应用程序应该能够关注用户的输入并对输入做相应的验证。

仅仅使用 `JavaScript` 是做不到的，用户也不想等待整个表单提交后才验证域值。

用 `Ajax` 充当客户端验证来取代简单的 `JavaScript` 验证的原因是，在许多案例中，表单域需要根据数据库进行检查（像本章例子中的用户名域）。同样，在大多数情况下，将所有的业务逻辑（包括验证）存储到服务器中心位置是比较专业的做法。

`Ajax` 是如此便捷，您不这样认为吗？



# 第 5 章

## Ajax 聊天

我们生活在一个通信变得非常重要的世界里，人们需要同他人快速容易地进行交流。E-mail、电话、邮件以及在线聊天是以书写文字的形式让人们进行思想交流的媒体。通信时一个重要的方面是对方的响应。从另一方返回的 E-mail 和信件不生动活泼，而电话和在线聊天提供动态的通信方式。在这一章中，我们将建立一个支持 Ajax 的在线聊天解决方案。

### 5.1 Ajax 聊天简介

大多数计算机之间的通信都是通过桌面应用程序完成的。这些应用使用 P2P（对等）系统以分散的方式互相进行通信。但是如果您所在公司内部出于安全考虑阻止用户访问除 HTTP80 端口之外的其他端口的连接时，这也许就不是一个可行的选择。如果是这种情况，您就的确面对一个真正的问题。

现在有许多基于语音和视频的网络聊天解决方案，它们大多都是基于 Java Applets 的。Applets 在浏览器间存在普遍的安全问题，而且有时它们甚至不使用 80 端口进行通信。因此，它们也不能解决和公司外部朋友进行联系的问题。

这就是 Ajax 要解决的地方，它为我们的问题带来了一个解决办法。可以付出很少的努力就将因特网中继聊天（IRC）客户端整合到浏览器，还可以开发自己的网络聊天解决方案。

您是否厌烦了工作时或在网吧内被告知不能安装或者使用您所喜欢的消息工具？您或许发现以前曾经有过这样的情形。现在我们来看看如何使用 Ajax 聊天解决方案打破以往这种令人遗憾的情形。

### Ajax 聊天解决方案

或许目前可用的让人印象最深刻的解决方案是 [www.meebo.com](http://www.meebo.com)，我们相信很多人已经听说了，如果没有的话，现在就去看看吧。首要的也是最重要的特征就是，它允许通过一个网络界面登录您最喜欢的便捷消息系统。Meebo 的登录界面如图 5.1 所示。

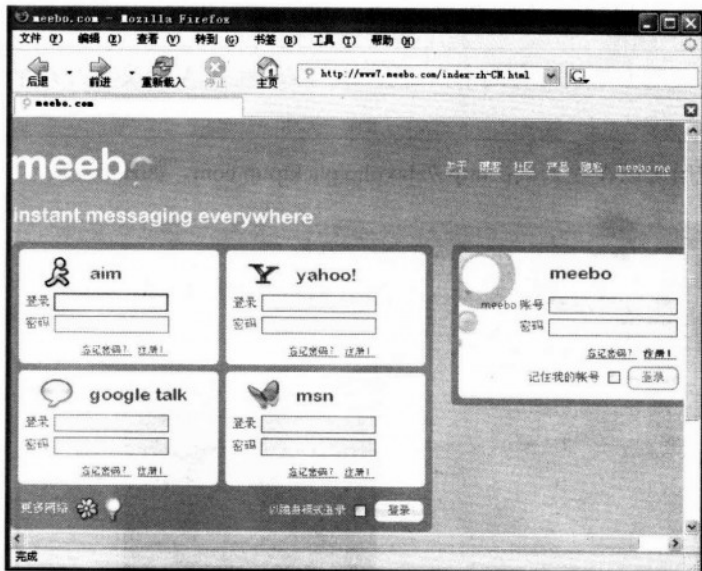


图 5.1 Meebo

Meebo 使用一个用户友好的单点登录界面来访问它的所有服务，没有任何弹出窗口和 Java applets 等。通过使用基于 Ajax 的方案，您可以忘记开始提到的那些问题。

Meebo 不是惟一提供聊天功能的网络应用程序。尽管 Ajax 还很年轻，但您可以发现其他一些聊天应用程序甚至解决方案都是在它的基础上建立的：

- <http://www.plasticshore.com/projects/chat/index.html>
- <http://treehouse.ofb.net/chat/?lang=en>
- <http://www.chategory.org>
- <http://www.socket7.net/lace/>
- <http://drupal.org/node/27689>

是开始工作的时候了。在本章的后续部分，我们将实现自己的在线聊天应用。

## 5.2 实现 Ajax 聊天

我们将保持应用是简单的、模块化的和可扩展的。为了做到这一点，我们不实现登录模块、聊天室、在线用户表等功能。为了保持它的简单风格，我们试图将精力集中在本章的目标——Ajax 聊天上。我们将实现基本的聊天功能：在不引起任何网页重新加载的情况下发送和接收消息。我们也允许用户挑选消息的颜色，这又是一个学习 Ajax 机制的好机会。

从本章下面要讲的应用程序出发，通过实现前面方案中的其他模块或这里尚未提及的模

块，我们可以很容易地扩展这个应用。如果感兴趣的话，可以将这部分作为自己的课后作业。

**注意** 要想使这些示例运行起来，需要 GD 库。附录 A 的安装介绍中包含如何支持 GD 库。

这个聊天应用可以在线测试：<http://ajaxphp.packtpub.com>，如图 5.2 所示。

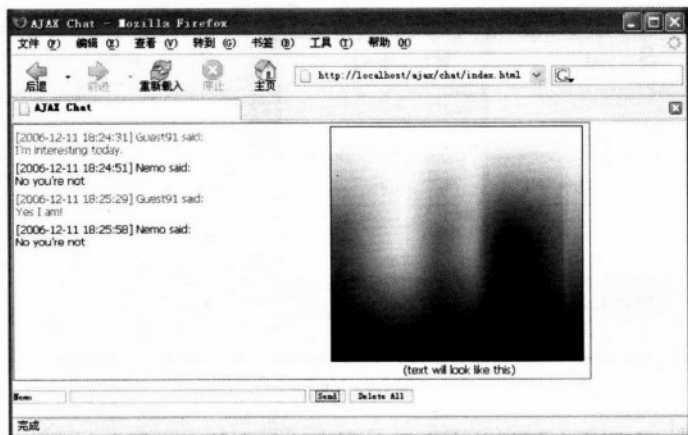


图 5.2 Ajax 聊天

本章的新奇之处是会有两个 XMLHttpRequest 对象。第一个处理更新聊天窗口，第二个处理颜色取色（当点到图片上时，坐标被发送到服务器上，服务器用颜色代码响应）。

Ajax 聊天的消息被保存到一个队列中（FIFO 结构的），就像在第 4 章了解到的，即使服务器很慢，也不会丢失消息，而且它们会以发送时的顺序到达服务器。与现在因特网上的其他方式不同，我们保证在一次请求还未结束之前，不向服务器重复发送多个请求。

## 》》》 实现步骤——Ajax 聊天

(1) 连接 Ajax 数据库，并用下面的代码创建表 chat:

```
CREATE TABLE chat
(
  chat_id int(11) NOT NULL auto_increment,
  posted_on datetime NOT NULL,
  user_name varchar(255) NOT NULL,
  message text NOT NULL,
  color char(7) default '#000000',
  PRIMARY KEY (chat_id)
```

```
);
```

(2) 在 Ajax 文件夹下，创建新文件夹 chat。

(3) 从下载代码中将 palette.png 文件复制到 chat 文件夹下。

(4) 我们将创建一个由服务器功能开始的应用。在 chat 文件夹下，创建文件 config.php，并把数据库配置代码加到此文件里（改变这些变量值来匹配配置）。

```
<?php
//定义数据库连接数据
define('DB_HOST', 'localhost');
define('DB_USER', 'ajaxuser');
define('DB_PASSWORD', 'practical');
define('DB_DATABASE', 'ajax');
?>
```

(5) 现在加入标准错误处理文件 error\_handler.php:

```
<?php
//设置用户错误处理方法为 error_handler
set_error_handler('error_handler', E_ALL);
//错误处理函数
function error_handler($errNo, $errStr, $errFile, $errLine)
{
    //清除所有已经生成的输出
    if(ob_get_length()) ob_clean();
    //输出错误消息
    $error_message = 'ERRNO: ' . $errNo . chr(10) .
                    'TEXT: ' . $errStr . chr(10) .
                    'LOCATION: ' . $errFile .
                    ', line ' . $errLine;
    echo $error_message;
    //阻止处理任何其他 PHP 脚本
    exit;
}
?>
```

(6) 创建文件 chat.php 并加入下面代码:

```
<?php
//参考包含 Chat 类的文件
require_once("chat.class.php");
//取回执行的操作
$mode = $_POST['mode'];
```

```
//默认情况下最后的 id 是 0
$cid = 0;
//创建一个新的聊天实例
$chat = new Chat();
//如果操作是 SendAndRetrieve
if($mode == 'SendAndRetrieveNew')
{
    //取回用来添加一个新消息的作用参数
    $name = $_POST['name'];
    $message = $_POST['message'];
    $color = $_POST['color'];
    $cid = $_POST['id'];

    //检查是否我们有合法的值
    if ($name != '' && $message != '' && $color != '')
    {
        //把消息传递到数据库
        $chat->postMessage($name, $message, $color);
    }
}
//如果操作是 DeleteAndRetrieve
elseif($mode == 'DeleteAndRetrieveNew')
{
    //删除所有已存在的消息
    $chat->deleteMessages();
}
//如果操作是 Retrieve
elseif($mode == 'RetrieveNew')
{
    //获取客户取回的最后消息的 id
    $cid = $_POST['id'];
}
//清除输出
if(ob_get_length()) ob_clean();
//发送 Headers 阻止浏览器缓存
header('Expires: Mon, 26 Jul 1997 05:00:00 GMT');
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');
header('Cache-Control: no-cache, must-revalidate');
header('Pragma: no-cache');
header('Content-Type: text/xml');
```

```
//从服务器取回新消息
echo $chat->retrieveNewMessages($id);
?>
```

(7) 创建文件 chat.class 并加入下面代码:

```
<?php
//加载配置文件
require_once('config.php');
//加载错误处理模块
require_once('error_handler.php');

//包含服务端聊天功能的类
class Chat
{
    //数据库处理
    private $mMysql;

    //构造函数打开数据库连接
    function __construct()
    {
        //连接到数据库
        $this->mMysql = new mysqli(DB_HOST, DB_USER, DB_PASSWORD,
                                DB_DATABASE);
    }
    //析构函数关闭数据库连接
    public function __destruct()
    {
        $this->mMysql->close();
    }

    //截去包含消息的表
    public function deleteMessages()
    {
        //建立增加一个新消息到服务器的 SQL 查询
        $query = 'TRUNCATE TABLE chat';
        //执行 SQL 查询
        $result = $this->mMysql->query($query);
    }
}

/*
```

```

postMessages 方法向数据库中插入消息
- $name 代表 post 消息的用户的名称
- $message 是 post 的消息
- $color 包含用户选取的颜色
*/
public function postMessage($name, $message, $color)
{
    //转换变量数据, 以便安全地将它们添加到数据库
    $name = $this->mMysqli->real_escape_string($name);
    $message = $this->mMysqli->real_escape_string($message);
    $color = $this->mMysqli->real_escape_string($color);
    //建立增加一个新消息到服务器的 SQL 查询
    $query = 'INSERT INTO chat(posted_on, user_name, message, color) ' .
        'VALUES (NOW(), "' . $name . '", "' . $message .
        '","' . $color . '")';
    //执行 SQL 查询
    $result = $this->mMysqli->query($query);
}
/*
retrieveNewMessages 方法提取已经发送给服务器的新消息
-$id 参数由客户端发送, 而且它是客户端接收到的最后一个消息的 id
通过$id 从数据库获取最近的消息, 并利用 XML 格式返回给客户端。
*/
public function retrieveNewMessages($id=0)
{
    //转换变量数据
    $id = $this->mMysqli->real_escape_string($id);
    //组成取回新消息的 SQL 查询
    if($id>0)
    {
        //取回比$id 新的消息
        $query =
        'SELECT chat_id, user_name, message, color, ' .
        '    DATE_FORMAT(posted_on, "%Y-%m-%d %H:%i:%s") ' .
        '    AS posted_on ' .
        ' FROM chat WHERE chat_id > ' . $id .
        ' ORDER BY chat_id ASC';
    }
    else
    {

```



```

//第一次加载仅仅从服务器取回最后 50 个消息
$query =
' SELECT chat_id, user_name, message, color, posted_on FROM ' .
' (SELECT chat_id, user_name, message, color, ' .
' DATE_FORMAT(posted_on, "%Y-%m-%d %H:%i:%s") AS posted_on ' .
' FROM chat ' .
' ORDER BY chat_id DESC ' .
' LIMIT 50) AS Last50' .
' ORDER BY chat_id ASC';
}
//执行查询
$result = $this->mMysqli->query($query);

//建立 XML 响应
$response = '<?xml version="1.0" encoding="UTF-8" standalone="yes"?>';
$response .= '<response>';
//输出清除标志
$response .= $this->isDatabaseCleared($id);
//检查我们是否有任何结果
if($result->num_rows)
{
//在所有取来的消息间循环来建立结果消息
while ($row = $result->fetch_array(MYSQLI_ASSOC))
{
    $id = $row['chat_id'];
    $color = $row['color'];
    $userName = $row['user_name'];
    $time = $row['posted_on'];
    $message = $row['message'];
    $response .= '<id>' . $id . '</id>' .
        '<color>' . $color . '</color>' .
        '<time>' . $time . '</time>' .
        '<name>' . $userName . '</name>' .
        '<message>' . $message . '</message>';
}
//尽快关闭数据库连接
$result->close();
}

//完成 XML 响应并返回它

```

```

    $response = $response . '</response>';
    return $response;
}
/*
    isDatabaseCleared 方法查看自从上次访问服务器后数据库是否被清空。
    参数包含最后从客户端接收的消息的 id。
*/
private function isDatabaseCleared($id)
{
    if($id>0)
    {
        //通过检查其 id 号比客户的最后 id 号小的行数，我们检查是否一个截去操作同时//被执行
        $check_clear = 'SELECT count(*) old FROM chat where chat_id<=' . $id;
        $result = $this->mMysqli->query($check_clear);
        $row = $result->fetch_array(MYSQLI_ASSOC);

        //如果一个截去操作发生了，白色书写板需要被重新设置
        if($row['old']==0)
            return '<clear>>true</clear>';
    }
    return '<clear>>false</clear>';
}
}
?>

```

(8) 创建文件 `get_color.php` 并加入下面代码:

```

<?php
//图片文件的名字
$imgfile='palette.png';
//加载图片文件
$img=imagecreatefrompng($imgfile);
//获得用户单击点的坐标
$offsetx=$_GET['offsetx'];
$offsety=$_GET['offsety'];
//获取单击的颜色
$rgb = ImageColorAt($img, $offsetx, $offsety);
$r = ($rgb >> 16) & 0xFF;
$g = ($rgb >> 8) & 0xFF;
$b = $rgb & 0xFF;
//返回颜色代码

```

```
printf('%%02s%02s%02s', dechex($r), dechex($g), dechex($b));  
?>
```

(9) 现在开始处理客户端了。先创建文件 `chat.css` 并加入下面代码:

```
body  
{  
    font-family: Tahoma, Helvetica, sans-serif;  
    margin: 1px;  
    font-size: 12px;  
    text-align: left  
}  
  
#content  
{  
    border: DarkGreen 1px solid;  
    margin-bottom: 10px  
}  
  
input  
{  
    border: #999 1px solid;  
    font-size: 10px  
}  
  
#scroll  
{  
    position: relative;  
    width: 340px;  
    height: 270px;  
    overflow: auto  
}  
  
.item  
{  
    margin-bottom: 6px  
}  
  
#colorpicker  
{  
    text-align:center
```

}

(10) 创建文件 index.html 并加入下面代码:

```
<!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" xml:lang="en" lang="en">
<head>
  <title>Ajax Chat</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <link href="chat.css" rel="stylesheet" type="text/css" />
  <script type="text/javascript" src="chat.js" ></script>
</head>
<body onload="init();" >
  <noscript>
    Your browser does not support JavaScript!!
  </noscript>
  <table id="content">
    <tr>
      <td>
        <div id="scroll">
        </div>
      </td>
      <td id="colorpicker">
        
        <br />
        <input id="color" type="hidden" readonly="true" value="#000000" />
        <span id="sampleText">
          (text will look like this)
        </span>
      </td>
    </tr>
  </table>
  <div>
    <input type="text" id="userName" maxlength="10" size="10" onblur=
"checkUsername();" />
    <input type="text" id="messageBox" maxlength="2000" size="50"
      onkeydown="handleKey(event)" />
    <input type="button" value="Send" onclick="sendMessage();" />
    <input type="button" value="Delete All" onclick="deleteMessages();" />
  </div>
</body>
</html>
```

```

    </div>
  </body>
</html>

```

(11) 创建文件 chat.js 并加入下面代码:

```

/* chatURL - 更新聊天消息的 URL */
var chatURL = "chat.php";
/* getColorURL - 获取已选 RGB 颜色的 URL */
var getColorURL = "get_color.php";
/* 创建 XMLHttpRequest 对象, 更新聊天消息并获取选中的颜色 */
var xmlhttpGetMessages = createXmlHttpRequestObject();
var xmlhttpGetColor = createXmlHttpRequestObject();
/* 变量, 表示多长时间访问一次服务器 */
var updateInterval = 1000; //需要等待多少毫秒来获取新消息
//当设置为 true 时, 显示详细的错误消息
var debugMode = true;
/* 初始化消息缓存 */
var cache = new Array();
/* lastMessageID - 最近聊天消息的 ID */
var lastMessageID = -1;
/* mouseX, mouseY - 事件的坐标 */
var mouseX, mouseY;
/* 创建 XMLHttpRequest 实例 */
function createXmlHttpRequestObject()
{
  //将存储 XMLHttpRequest 对象的索引
  var xmlhttp;
  //这将在所有浏览器上工作, 除了 IE6 和老版本
  try
  {
    //尝试创建 XMLHttpRequest 对象
    xmlhttp = new XMLHttpRequest();
  }
  catch(e)
  {
    //假定是 IE6 或者老版本
    var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
                                      "MSXML2.XMLHTTP.5.0",
                                      "MSXML2.XMLHTTP.4.0",
                                      "MSXML2.XMLHTTP.3.0",

```

```
        "MSXML2.XMLHTTP",
        "Microsoft.XMLHTTP");

//尝试每一个 id 号直到一个工作
for (var i=0; i<XmlHttpRequestVersions.length && !xmlHttpRequest; i++)
{
    try
    {
        //尝试创建 XMLHttpRequest 对象
        xmlHttpRequest = new ActiveXObject(XmlHttpRequestVersions[i]);
    }
    catch (e) {}
}
//返回创建的对象或者显示一个错误消息
if (!xmlHttpRequest)
    alert("Error creating the XMLHttpRequest object.");
else
    return xmlHttpRequest;
}

/* 聊天初始化, 当聊天页面加载时执行 */
function init()
{
    //获取用户写新消息的文本框的索引
    var oMessageBox = document.getElementById("messageBox");
    //阻止 autofill 功能开启
    oMessageBox.setAttribute("autocomplete", "off");
    //参考“文本将看起来像这样”的消息
    var oSampleText = document.getElementById("sampleText");
    //设置默认颜色为黑色
    oSampleText.style.color = "black";
    //当表单加载时确保我们的用户有一个默认的随机名字
    checkUsername();
    //开始更新聊天窗口
    requestNewMessages();
}

//此函数确保用户名不为空, 并且如果是空的话就生成一个随机的名字
function checkUsername()
{
```

```
//确保我们的用户在表单加载时有一个默认的随机名字
var oUser=document.getElementById("userName");
if(oUser.value == "")
    oUser.value = "Guest" + Math.floor(Math.random() * 1000);
}
/* 按下发送按钮时调用此函数 */
function sendMessage()
{
    //保存消息到一个本地变量并且清除文本框
    var oCurrentMessage = document.getElementById("messageBox");
    var currentUser = document.getElementById("userName").value;
    var currentColor = document.getElementById("color").value;
    //不发送空消息
    if (trim(oCurrentMessage.value) != "" &&
        trim(currentUser) != "" && trim (currentColor) != "")
    {
        //如果我们需要发送和接收消息
        params = "mode=SendAndRetrieveNew" +
            "&id=" + encodeURIComponent(lastMessageID) +
            "&color=" + encodeURIComponent(currentColor) +
            "&name=" + encodeURIComponent(currentUser) +
            "&message=" + encodeURIComponent(oCurrentMessage.value);
        //把消息加到队列
        cache.push(params);
        //清除文本框
        oCurrentMessage.value = "";
    }
}

/* 按下删除按钮时调用此函数 */
function deleteMessages()
{
    //设置说明我们正在删除消息的标志
    params = "mode=DeleteAndRetrieveNew";
    //把消息加到队列
    cache.push(params);
}

/* 使用异步请求获取新消息, 显示新消息、删除新消息 */
function requestNewMessages()
```

```
{
    //从网页取回用户名和颜色
    var currentUser = document.getElementById("userName").value;
    var currentColor = document.getElementById("color").value;
    //如果 xmlhttpGetMessages 不为空就继续
    if(xmlHttpGetMessages)
    {
        try
        {
            //如果有一个服务器操作在进行就不开始另一个这样的操作
            if (xmlHttpGetMessages.readyState == 4 ||
                xmlhttpGetMessages.readyState == 0)
            {
                //我们将存储构造服务器请求的参数
                var params = "";
                //如果有请求存储在队列中, 取出最早进入的一个
                if (cache.length>0)
                    params = cache.shift();
                //如果缓存是空的, 就接收新消息
                else
                    params = "mode=RetrieveNew" +
                        "&id=" +lastMessageID;
                //调用服务器页来执行服务端操作
                xmlhttpGetMessages.open("POST", chatURL, true);
                xmlhttpGetMessages.setRequestHeader("Content-Type",
                    "application/x-www-form-urlencoded");
                xmlhttpGetMessages.onreadystatechange = handleReceivingMessages;
                xmlhttpGetMessages.send(params);
            }
            else
            {
                //我们为了新消息将再次检查
                setTimeout("requestNewMessages();", updateInterval);
            }
        }
        catch (e)
        {
            displayError(e.toString());
        }
    }
}
```



```
}

/* 更新消息时处理 http 响应 */
function handleReceivingMessages()
{
    //如果处理完成就继续
    if (xmlHttpGetMessages.readyState == 4)
    {
        //如果 HTTP 状态为“OK”就继续
        if (xmlHttpGetMessages.status == 200)
        {
            try
            {
                //处理服务器的响应
                readMessages();
            }
            catch (e)
            {
                //显示错误消息
                displayError(e.toString());
            }
        }
        else
        {
            //显示错误消息
            displayError(xmlHttpGetMessages.statusText);
        }
    }
}

/* 更新消息时处理服务器响应 */
function readMessages()
{
    //取回服务器的响应
    var response = xmlHttpGetMessages.responseText;
    //服务器错误?
    if (response.indexOf("ERRNO") >= 0
        || response.indexOf("error:") >= 0
        || response.length == 0)
        throw(response.length == 0 ? "Void server response." : response);
}
```

```
//取回文档元素
response = xmlhttpGetMessages.responseXML.documentElement;
//取回说明聊天窗口是否已经关闭的标志
clearChat = response.getElementsByTagName("clear").item(0).firstChild.data;
//如果标志被设为 true, 我们需要清除聊天窗口
if(clearChat == "true")
{
    //清除聊天窗口并重新设置 id
    document.getElementById("scroll").innerHTML = "";
    lastMessageID = -1;
}
//从服务器响应中取回数组长
idArray = response.getElementsByTagName("id");
colorArray = response.getElementsByTagName("color");
nameArray = response.getElementsByTagName("name");
timeArray = response.getElementsByTagName("time");
messageArray = response.getElementsByTagName("message");
//把新消息添加到聊天窗口中
displayMessages(idArray, colorArray, nameArray, timeArray, messageArray);
//最后接收到消息的 ID 本地保存
if(idArray.length>0)
    lastMessageID = idArray.item(idArray.length - 1).firstChild.data;
//重启序列
setTimeout("requestNewMessages();", updateInterval);
}

/* 向聊天列表中加入新消息 */
function displayMessages(idArray, colorArray, nameArray,
                        timeArray, messageArray)
{
    //每次循环增加一个新消息
    for(var i=0; i<idArray.length; i++)
    {
        //获取消息细节
        var color = colorArray.item(i).firstChild.data.toString();
        var time = timeArray.item(i).firstChild.data.toString();
        var name = nameArray.item(i).firstChild.data.toString();
        var message = messageArray.item(i).firstChild.data.toString();
        //组成显示消息的 HTML 代码
        var htmlMessage = "";
```

```

    htmlMessage += "<div class=\"item\" style=\"color:\" + color + "\">";
    htmlMessage += "[" + time + "]" + name + " said: <br/>";
    htmlMessage += message.toString();
    htmlMessage += "</div>";
    //显示消息
    displayMessage (htmlMessage);
}
}

//显示一个消息
function displayMessage (message)
{
    //获取滚动对象
    var oScroll = document.getElementById("scroll");
    //检查滚动是否是向下
    var scrollDown = (oScroll.scrollHeight - oScroll.scrollTop <=
        oScroll.offsetHeight );
    //显示消息
    oScroll.innerHTML += message;
    //向下滚动滚动栏
    oScroll.scrollTop = scrollDown ? oScroll.scrollHeight : oScroll.scrollTop;
}

//显示错误消息的函数
function displayError(message)
{
    //显示错误消息, 如果 debugMode 为 true 就显示更多技术细节
    displayMessage("Error accessing the server! "+
        (debugMode ? "<br/>" + message : ""));
}

/* 检测值时按 Enter 键 */
function handleKey(e)
{
    //获取事件
    e = (!e) ? window.event : e;
    //获取按下字符的代码
    code = (e.charCode) ? e.charCode :
        ((e.keyCode) ? e.keyCode :
            ((e.which) ? e.which : 0));
}

```

```
//处理 keydown 事件
if (e.type == "keydown")
{
    //如果 Enter 键按下 (编码 13)
    if(code == 13)
    {
        //发送当前的消息
        sendMessage();
    }
}

/* 删除字符串开头和结尾的空格*/
function trim(s)
{
    return s.replace(/(^\s+)|(\s+$)/g, "");
}

/* 计算页面中鼠标坐标的函数 */
function getMouseXY(e)
{
    //浏览器细节
    if(window.ActiveXObject)
    {
        mouseX = window.event.x + document.body.scrollLeft;
        mouseY = window.event.y + document.body.scrollTop;
    }
    else
    {
        mouseX = e.pageX;
        mouseY = e.pageY;
    }
}

/* 进行服务器访问以获得所选颜色的 RGB 编码 */
function getColor(e)
{
    getMouseXY(e);
    //如果 XMLHttpRequest 对象为空就不做任何事情
    if(xmlHttpGetColor)
```

```
{
    //用鼠标当前位置初始化偏移位置
    var offsetX = mouseX;
    var offsetY = mouseY;
    //获取参照
    var oPalette = document.getElementById("palette");
    var oTd = document.getElementById("colorpicker");
    //计算窗口中的偏移位置
    if(window.ActiveXObject)
    {
        offsetX = window.event.offsetX;
        offsetY = window.event.offsetY;
    }
    else
    {
        offsetX -= oPalette.offsetLeft + oTd.offsetLeft;
        offsetY -= oPalette.offsetTop + oTd.offsetTop;
    }
}

//异步调用服务器来查找单击的颜色
try
{
    if (xmlHttpGetColor.readyState == 4 ||
        xmlHttpGetColor.readyState == 0)
    {
        params = "?offsetx=" + offsetX + "&offsety=" + offsetY;
        xmlHttpGetColor.open("GET", getColorURL+params, true);
        xmlHttpGetColor.onreadystatechange = handleGettingColor;
        xmlHttpGetColor.send(null);
    }
}
catch(e)
{
    //显示错误消息
    displayError(xmlHttp.statusText);
}
}

/* 处理 http 响应 */
function handleGettingColor()
```

```
{
    //如果处理完成, 决定处理返回的数据
    if (xmlHttpGetColor.readyState == 4)
    {
        //仅在HTTP 状态是“OK” 时进行
        if (xmlHttpGetColor.status == 200)
        {
            try
            {
                //改变颜色
                changeColor();
            }
            catch (e)
            {
                //显示错误消息
                displayError(xmlHttpGetColor.statusText);
            }
        }
        else
        {
            //显示错误消息
            displayError(xmlHttpGetColor.statusText);
        }
    }
}

/* 改变显示消息的颜色 */
function changeColor()
{
    response=xmlHttpGetColor.responseText;
    //服务器错误?
    if (response.indexOf("ERRNO") >= 0
        || response.indexOf("error:") >= 0
        || response.length == 0)
        throw(response.length == 0 ? "Can't change color!" : response);
    //改变颜色
    var oColor = document.getElementById("color");
    var oSampleText = document.getElementById("sampleText");
    oColor.value = response;
    oSampleText.style.color = response;
}
```

(12) 讲述完这些之后，现在看一下这个应用程序的运行情况。在浏览器地址栏中输入 `http://localhost/ajax/chat/index.html`，看一下聊天窗口开始时的样子，如图 5.3 所示。

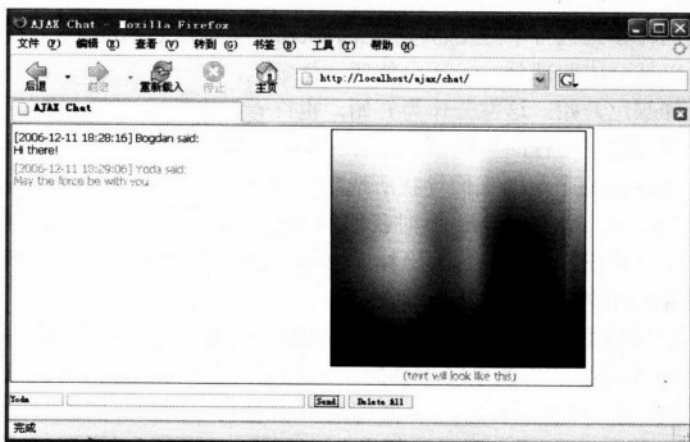


图 5.3 聊天窗口

可以观察到默认情况下消息的颜色是黑色（RGB 代码为#000000）。在图 5.3 所示中，可以看到默认的随机名字是 Guest91。当初始加载这个聊天窗口时，所有先前发送的消息都会显示出来。可以简单地在调色板图片上单击一下喜欢的颜色，就可以改变消息的颜色。

## 程序说明

从技术上讲，这个应用程序是分成两个小程序来构建最终解决方案的：

- 聊天应用程序；
- 选择颜色应用程序。

聊天应用程序实现了发送和接收消息的基本功能。每个用户可以选择昵称并发送消息。包含所有发送消息的聊天窗口是通过从服务器异步接收的消息来更新的。

使用调色板来让用户为他们写的文本选择颜色，这个调色板容纳了整个色谱。当单击调色板时，鼠标坐标被发送到服务器，即可获取颜色代码。

如果分析一下代码，细节就会变得很明了。先来看一下 `index.html` 文件。这个脚本惟一人感兴趣的就是用 DHTML 实现了滚动区。关于滚动的一些信息可以在 `http://www.dynweb.com/dhtml/scroll/` 中找到。基本上，让页面的一部分带一个滚动条在旁边的方法就是一层嵌套在另一层里面。在我们的示例中，`div scroll` 和它的内层就使用了这个技巧。

外层是 `scroll`，它有固定的宽度和高度，而且最有用的属性就是 `overflow`。一般来说，一个大块框的内容受该框内容边缘的限制。在某些情况下，一个框会溢出，即它的内容部分或

者整个位于框外。在 CSS 中，这个属性说明了当一个元素溢出它的范围后会发生什么。要获取更多细节，请访问 <http://www.w3.org/TR/REC-CSS2/visufx.html>，以查看 overflow 的定义。

现在已经定义了大块框以及当其内容超出范围时会发生什么，我们可以轻松地猜测到大块框的内部内容最终会超出框的大小。内部内容包括了聊天中写的消息。

下一步，我们转移到包含了应用程序的 JavaScript 部分的文件 chat.js。

整个文件可以被分成两部分：一部分处理选择颜色，另一部分处理聊天消息。

我们先由选择颜色开始。这部分作为开始，也许看起来很难证明它是比较容易实现的。我们要先有一个整体处理过程的全局概念。首先我们得有一个调色板图片，它包含了可见颜色的整个色谱。PHP 有两个帮助我们寻找所选颜色 RGB 代码的函数——`imagecreatefrompng` 和 `imagecolorat`。当讲到 `get_color.php` 页时我们将看到更多关于这两个函数的内容。现在只需要知道这两个函数允许我们获得图片上坐标 `x` 和 `y` 位置上的像素的 RGB 代码。像素的位置是通过 `getMouseXY` 函数获取的。

`getColor` 函数获取用户单击调色板图片时所选颜色的 RGB 代码。首先它从事件中获取鼠标的坐标，然后计算已经产生的单击事件的坐标在图片上的相对值。它是这样做到的：从 `getMouseXY` 函数获取的位置减去窗口中 `td` 元素和 `td` 位置里的图片的相对位置。在计算出相对坐标作为 `offsetx` 和 `offsety` 后，服务器页将会返回被选颜色的 RGB 代码。HTTP 请求对象状态的改变是由 `handleGettingColor` 函数处理的。

`handleGettingColor` 函数检查到发往服务器的请求完成且没有发生错误时，调用 `changeColor` 函数。这个函数将把从服务器返回的 RGB 代码置于文本域并用给定代码着色样例文本。

现在来看看聊天应用程序是如何工作的。

默认情况下，当页初始化并且 `onblur` 事件发生时，`checkUsername` 函数被调用。这个函数生成一个强制的用户名确保它不为空。

按下发送按钮时，`sendMessage` 函数被调用。这个函数将当前消息增加到要发送到服务器的消息队列中。在加到队列前通过调用 `trim` 函数先把消息头尾空格去掉，并用 `encodeURIComponent` 来编码消息，确保它成功进入队列。

当 `keydown` 事件发生时调用 `handleKey` 函数。当按下 `Enter` 键时，`sendMessage` 函数被调用，这样就能确保按下发送按钮和在消息框控制内按下 `Enter` 键有同样的效果。

`deleteMessage` 函数将删除加入发送到服务器的消息。`requestNewMessages` 函数负责发送聊天消息。它从队列中获取一个消息然后发送给服务器。HTTP 请求对象状态的改变由 `handleReceivingMessages` 函数处理。

`handleReceivingMessages` 检查到发往服务器的请求完成并且没有错误发生时，`readMessages` 函数会调用。

`readMessages` 函数检查是否其他人删除了所有聊天消息，如果是这样，客户的聊天窗口就是空的。为了把新消息追加到聊天中，我们调用了 `displayMessages` 函数。这个函数将参数看作是与消息相符的数组。它把新消息组成 HTML 并调用 `displayMessages` 函数把它们追加



到聊天中的消息后面。起初，`displayMessages` 函数检查滚动条是否在消息列表的底部。这对于在函数最后复位滚动条位置以使聚焦在最后的新消息上是很必要的。

最后介绍的函数是 `init` 函数。它的作用是获取聊天消息，确保用户名不是空的，设置文本颜色为黑色，并关闭自动完成功能。

对于错误处理部分，我们使用了 `displayError` 函数，它将错误消息作为参数依次调用 `displayMessage` 函数。

下面转移到应用的服务端，先从介绍 `chat.php` 文件开始。服务器是这样来处理客户请求的：

- 获取客户的参数；
- 识别要被执行的操作；
- 执行必要的操作；
- 给客户返回结果。

请求包括模式参数，它说明了由服务器执行的下面操作之一：

- `SendAndRetrieve` 先是新消息被插入数据库，然后所有新消息被取出并返回给客户；
- `DeleteAndRetrieve` 所有消息被删除但新消息可能存在并被取出返回给客户；
- `Retrieve` 新消息被取出并返回给客户。

`chat.php` 后的业务逻辑在 `chat.class.php` 脚本中，它包含了 `chat` 类。

`deleteMessages` 方法删除了数据表中的所有信息。

`postMessages` 方法把所有的新消息插入到数据库中。

`isDatabaseCleared` 方法检查是否删除了所有消息。基本上，通过提供从服务器获取最后消息的 ID，然后检查它是否仍然存在，就能检测出是否所有消息都被删除了。

`retrieveNewMessages` 方法得到在最后请求期间从服务器获取的最后消息（由它的 id 标识）之后的所有消息（如果一个最后请求存在，或者所有消息在其他框中），并通过调用 `isDatabaseCleared` 方法检查数据库是否已经空了。这个函数组成响应并返回给客户。

`config.php` 文件包括了数据库配置参数，`error_handler.php` 文件包含了处理错误块。

现在来看一下 `get_color.php` 文件里的颜色选择函数在服务端是如何实现的。

上面提到了两个 PHP 函数，用来获取图片中像素的 RGB 代码的。来看一下它们是如何工作的：

- `Imagecreatefrompng(string filename)` 返回一个代表从指定文件名得到的 PNG 格式图片的标识符。

- `int imagecolorat(resource image,int x,int y)` 返回由 `image` 指定的图片中指定位置像素的颜色索引。如果 PHP 是由 GD library 2.0 或者更高版本编译的，并且图片是全屏图片，这个函数返回像素 RGB 代码的整数值。

结果中的第一个 8 位包含了蓝色代码，下一个 8 位包含了绿色代码，最后一个 8 位包含了红色代码。使用位移和屏蔽就能得到不同的红、绿、蓝组成的整数值。剩下要做的就是将它们转换成十六进制值，连接起这些值，然后返回给客户。

把这些东西包装好，先从呈现给用户的界面开始，由 HTML、CSS 和 JavaScript 脚本组成的客户端应用是在 `index.htm`、`chat.css` 和 `chat.js` 文件里实现的。在看完界面是什么样的，并知道从 Web 服务器获取的数据为了能呈现给用户做了何种处理之后，我们进行下一步来看服务端应用。

由客户端调用的文件是 `chat.php` 和 `get_color.php`。最后一步是由显示连接数据库参数文件（`config.php`）、错误处理模块（`error_handler.php`）和包含核心功能的脚本（`chat.class.php`）组成的。

### 5.3 小结

在本章开始，介绍了在因特网上人们与其他人以动态方式通信时为什么会遇到一些问题，我们看到了这些问题的解决办法是什么以及 Ajax 聊天是如何带来一些新的、有用的、充满活力的东西的。在看过一些其他 Ajax 聊天的实现后，我们开始建立自己的解决方案。逐步地，我们实现了 Ajax 聊天，并让它保持简单、容易扩展和模块化的风格。

阅读完这章后，可以尝试通过增加下面特征来改进方案：

- 聊天室；
- 简单命令行（加入/离开聊天室，聊天室间切换）；
- 悄悄话。



## Ajax 建议和自动完成

建议和自动完成是大多数浏览器、E-mail 客户、源代码编辑器、Word 处理器和操作系统中很受用户喜爱的功能。建议和自动完成就像是一枚硬币的两面——它们联合在一起。通常，它们之间没有差别，但自动完成使用得更频繁。

自动完成是指应用程序有能力预测用户想写的单词或句子。这个功能加快了交互过程，使得用户界面更加友好，帮助用户使用正确的词汇，并帮助他们避免打字错误。

在浏览器中可以看到自动完成的例子，用户在地址栏里输入新地址或者填写某一表单时，那个浏览器的自动完成引擎就会触发。在 E-mail 程序中，它是非常有用的，通过输入很少字母就能够选择接收者。

相信大家一定会喜欢有代码完成功能的源代码文本编辑器。长变量名使得代码容易理解，但是输入困难，除非编辑器支持代码自动完成功能。在一些编辑器中，在输入对象和句点时，会得到这个对象公共成员的滚动列表，就像用手指输入的文本一样。Microsoft 已经在 Visual Studio 集成开发环境中实现了这个功能，并以 IntelliSense 的名字申请了专利。GNU Emacs 编辑器在 Microsoft 之前就一直支持自动完成功能。

在操作系统内核中，比如 Unix 的 bash、sh 或者 Windows 的命令提示、命令名、文件名以及路径的自动完成通常是在输入单词的头几个字母后按 tab 键来完成的。在有一个很长的路径要输入时，这个功能非常有用。

### 6.1 Ajax 建议和自动完成简介

自动完成功能曾是桌面应用程序的一个重要特征。最近这个功能在 web 应用中流行起来了（注意 Web 浏览器中的典型表单自动完成，或者记住密码功能，是在 web 浏览器本地实现的，它们不是网站的功能）。

全部要做的是，使用已经被整合到桌面应用中的功能来丰富 web 应用的用户界面。<http://demo.script.aculo.us/ajax/autocompleter> 是非常精彩的实现了自动完成功能的例子。

这个功能最流行的例子就是 Google Suggest。

## Google Suggest

为什么是 Google Suggest 呢？因为它是最流行的使用 Ajax 完成建议和自动完成功能的 Web 实现。不论是否相信，Google 并不是第一个实现这种技术的。Christian Stocker 于 2004 年 4 月在他的 Bitflux 博客 [http://blog.bitflux.ch/archive/2004/07/13/livesearch\\_roundup.html](http://blog.bitflux.ch/archive/2004/07/13/livesearch_roundup.html) 中用到了这个功能，比 Google 发布早 7 个月。一篇精确描述网页中如何使用 JavaScript 实现自动完成文本框的文章可追溯到 2003 年 9 月——<http://www.sitepoint.com/article/life-autocomplete-textboxes>。众所周知，XMLHttpRequest 已有多年历史。因此，Google 没有创造任何东西；它只是把这些东西整合到一个完美的例子中。

Google Suggest 的访问网址是 <http://www.google.com/webhp?complete=1&hl=en>，如图 6.1 所示。

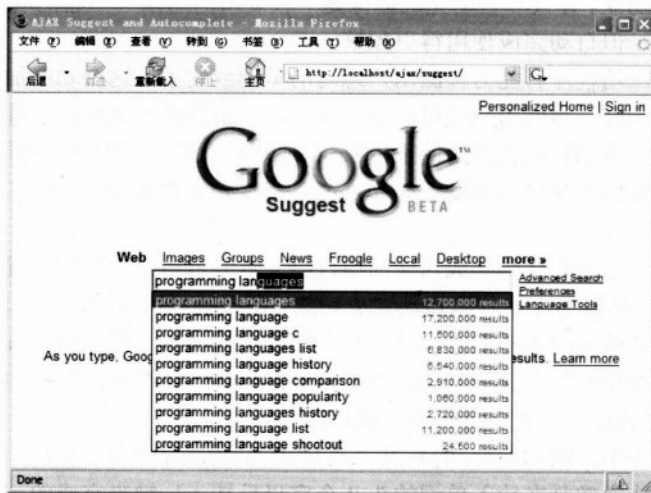


图 6.1 Google Suggest 测试阶段第二版本

在类似 Google Suggest 这样的应用中，JavaScript 脚本比较聪明的做法就是，它把先前收到的建议用某一关键词缓存起来。因此，如果打一个关键字，然后往回删除几个字母，那么根据请求收到的旧的建议已经被缓存起来，没有必要再取出来。

同样的技术在 Gmail (<http://www.gmail.com>) 和 Google 地图 (<http://maps.google.com>) 中也已经实现了。

## 6.2 实现 Ajax 建议和自动完成

这一章将开发建议和自动完成功能，帮助用户在 <http://www.php.net> 中查找 PHP 函数和

函数的官方帮助页，这章需要的 PHP 函数数据库包括来自 <http://www.php.net/quickref.php> 的所有 PHP 函数。

在应用中要实现以下功能（如图 6.2 所示）：

- 当用户打字时匹配函数检索结果并显示在一个可滚动的下拉列表中。
- 当前关键字是返回结果的第一个建议，并自动补上缺失的字母。增加的部分高亮显示。
- 与查询关键字匹配的开头几个字母在下拉列表中用黑体显示。
- 下拉列表是可滚动的，但滚动条仅在结果列表超出预先定义的建议数目时才出现。

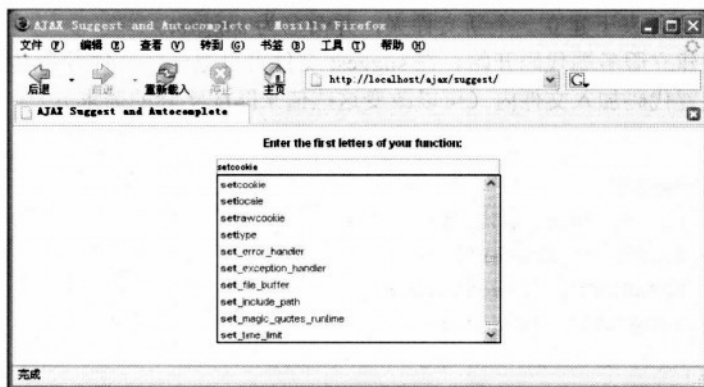


图 6.2 大量感兴趣的函数

## 》》》 实现步骤——Ajax 建议和自动完成

(1) 和平常一样，从创建必要的数据库结构开始。在 `ajax` 数据库中建立一个新表，命名为 `suggest`，它只包含一个单独的字段（`name`），同时也是主键：

```
CREATE TABLE suggest
(
    name VARCHAR(100) NOT NULL DEFAULT '',
    PRIMARY KEY (name)
);
```

(2) 用完整的 PHP 函数列表填充 `suggest` 表，这些函数来自 <http://www.php.net/quickref.php>，因为这个表含有 4000 多条记录，在这里只显示了前十条。为得到完整列表请下载代码脚本：

```
INSERT INTO suggest (name) VALUES
('abs'),
('acos'),
```

```

('acosh'),
('addslashes'),
('addslashes'),
('aggregate'),
('aggregate_info'),
('aggregate_methods'),
('aggregate_methods_by_list'),
('aggregate_methods_by_regexp');

```

(3) 在 ajax 文件夹下建立一个新文件夹，并命名为 suggest。

(4) 我们由建立服务端代码开始。在 suggest 文件夹里建一个文件并命名为 config.php，将以下数据库配置代码加入文件内（可以改变这些值来匹配实际的需求）：

```

<?php
//定义数据库连接数据
define('DB_HOST', 'localhost');
define('DB_USER', 'ajaxuser');
define('DB_PASSWORD', 'practical');
define('DB_DATABASE', 'ajax');
?>

```

(5) 然后再加一个标准错误处理文件 error\_handler.php:

```

<?php
//设置用户错误控制函数
set_error_handler('error_handler', E_ALL);
//错误控制函数
function error_handler($errNo, $errStr, $errFile, $errLine)
{
    //清除所有已产生的输出
    if(ob_get_length()) ob_clean();
    //输出错误信息
    $error_message = 'ERRNO: ' . $errNo . chr(10) .
                    'TEXT: ' . $errStr . chr(10) .
                    'LOCATION: ' . $errFile .
                    ', line ' . $errLine;

    echo $error_message;
    //防止继续处理 PHP 脚本
    exit;
}
?>

```

(6) 创建文件 suggest.php 并加入以下代码:

```
<?php
//引用含有 Suggest 类的文件
require_once('suggest.class.php');
//创建 Suggest 实例
$suggest = new Suggest();
//获取作为参数传递的关键词
$keyword = $_GET['keyword'];
//清除输出
if(ob_get_length()) ob_clean();
//发送标题阻止浏览器缓存
header('Expires: Mon, 26 Jul 1997 05:00:00 GMT ');
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . 'GMT');
header('Cache-Control: no-cache, must-revalidate');
header('Pragma: no-cache');
header('Content-Type: text/xml');
//向客户端返回结果
echo $suggest->getSuggestions($keyword);
?>
```

(7) 创建文件 suggest.class.php 并加入以下代码:

```
<?php
//加载错误控制模块 require_once('error_handler.php');
//load configuration file
require_once('config.php');

//实现服务器端建议和自动完成功能的类 class Suggest
{
    //数据库控制
    private $mMysqli;

    //打开数据库连接
    function __construct()
    {
        //连接数据库
        $this->mMysqli = new mysqli(DB_HOST, DB_USER, DB_PASSWORD,
                                   DB_DATABASE);
    }
}
```

```
//关闭数据库连接
function __destruct()
{
    $this->mMysqli->close();
}

//返回所有由$keyword 开始的 PHP 函数
public function getSuggestions($keyword)
{
    //离关键字字符串
    $patterns = array('/\s+/', '/"/', '/%/');
    $replace = array('');
    $keyword = preg_replace($patterns, $replace, $keyword);
    //创建 SQL 查询, 从数据库获取匹配的功能
    if($keyword != '')
        $query = 'SELECT name ' .
            'FROM suggest ' .
            'WHERE name LIKE "' . $keyword . '%"';
    //如果关键字为空, 创建一个不返回任何结果的查询
    else
        $query = 'SELECT name ' .
            'FROM suggest ' .
            'WHERE name=""';
    //执行 SQL 查询
    $result = $this->mMysqli->query($query);
    //创建 XML 响应
    $output = '<?xml version="1.0" encoding="UTF-8" standalone="yes"?>';
    $output .= '<response>';
    //如果已经去的结果, 循环将它们放入输出.
    if($result->num_rows)
        while ($row = $result->fetch_array(MYSQLI_ASSOC))
            $output .= '<name>' . $row['name'] . '</name>';
    //关闭结果流
    $result->close();
    //添加结束标签
    $output .= '</response>';
    //返回结果
    return $output;
}
//Suggest 类结束
```



```
}
?>
```

(8) 创建文件 `index.html` 并加入以下代码:

```
<!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" xml:lang="en" lang="en">
  <head>
    <title>Ajax Suggest and Autocomplete</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <link href="suggest.css" rel="stylesheet" type="text/css" />
    <script type="text/javascript" src="suggest.js"></script>
  </head>
  <body>
    <noscript>
      Your browser does not support JavaScript!!
    </noscript>
    <div id="content" onclick="hideSuggestions();">
      <div id="message">Enter the first letters of your function:</div>
      <input type="text" name="keyword" id="keyword" maxlength="70"
        size="69" onkeyup = "handleKeyUp(event)" value="" />
      <div id="scroll">
        <div id="suggest">
        </div>
      </div>
    </div>
  </body>
</html>
```

(9) 创建文件 `suggest.css` 并加入以下代码:

```
body
{
  font-family: helvetica, sans-serif;
  margin: 0px;
  padding: 0px;
  font-size: 12px;
}

#content
{
```

```
height: 100%;  
width: 100%;  
text-align:center  
}  
  
#message  
{  
font-weight: bold;  
text-align: center;  
margin-left: 10px;  
margin-bottom: 10px;  
margin-top: 10px  
}  
  
a  
{  
text-decoration: none;  
margin: 0px;  
color: #173f5f  
}  
  
input  
{  
border: #999 1px solid;  
font-family: helvetica, sans-serif;  
font-weight: normal;  
font-size: 10px  
}  
  
#scroll  
{  
position: relative;  
margin: 0 auto;  
visibility: hidden;  
background-color: white;  
z-index: 1;  
width: 300px;  
height: 180px;  
border-top-style: solid;  
border-right-style: solid;
```

```
border-left-style: solid;
border-collapse: collapse;
border-bottom-style: solid;
border-color: #000000;
border-width: 1px;
overflow: auto
}

#scroll div
{
margin: 0 auto;
text-align:left
}

#suggest table
{
width: 270px;
font-size: 11px;
font-weight: normal;
color: #676767;
text-decoration: none;
border: 0px;
padding: 0px;
text-align:left;
margin: 0px
}

.highlightrow
{
background-color: #999999;
cursor: pointer
}
```

(10) 创建文件 suggest.js 并加入以下代码:

```
/* URL to the PHP page called for receiving suggestions for a keyword*/
var getFunctionsUrl = "suggest.php?keyword=";
/* URL 用于观察已选择的建议结果*/
var phpHelpUrl="http://www.php.net/manual/en/function.";
/*已经生成 HTTP 请求的关键词 */
var httpRequestKeyword = "";
```

```
/* 最近建议的关键词*/
var userKeyword = "";
/* 对关键词返回建议的数目*/
var suggestions = 0;
/* 一个建议最多能显示的字符数*/
var suggestionMaxLength = 30;
/*标志在最近一次 keyup 事件后，上下箭头键是否被按下 */
var isKeyUpDownPressed = false;
/* 最近一次用于自动完成的关键词*/
var autoCompleteedKeyword = "";
/* 标志是否有针对当前请求的关键词的结果*/
var hasResults = false;
/* 使用 clearTimeout 方法退出评估的标志*/
var timeoutId = -1;
/* 通过鼠标或上下箭头选择的建议*/
var position = -1;
/* 缓存已接收的针对不同关键词的建议*/
var oCache = new Object();
/* 可显示建议的最大最小位置*/
var minVisiblePosition = 0;
var maxVisiblePosition = 9;
//当设置为真时，显示错误信息的细节
var debugMode = true;
/* 用于与服务器通信的 XMLHttpRequest 对象 */
var xmlhttpGetSuggestions = createXmlHttpRequestObject();
/* 初始功能控制的 onload 事件 */
window.onload = init;

//创建 XMLHttpRequest 实例
function createXmlHttpRequestObject()
{
    //用于存放 XMLHttpRequest 对象的引用
    var xmlhttp;
    //下面适用于除了 IE6 及其更早版本外所有的浏览器
    try
    {
        //尝试创建 XMLHttpRequest 对象
        xmlhttp = new XMLHttpRequest();
    }
    catch(e)
```

```
{
    //假设是 IE6 或其更早版本
    var XmlHttpRequestVersions = new Array("MSXML2.XMLHTTP.6.0",
                                            "MSXML2.XMLHTTP.5.0",
                                            "MSXML2.XMLHTTP.4.0",
                                            "MSXML2.XMLHTTP.3.0",
                                            "MSXML2.XMLHTTP",
                                            "Microsoft.XMLHTTP");

    //尝试, 直到创建成功。
    for (var i=0; i<XmlHttpRequestVersions.length && !xmlHttpRequest; i++)
    {
        try
        {
            // 尝试建立 XMLHttpRequest 对象
            xmlHttpRequest = new ActiveXObject(XmlHttpRequestVersions[i]);
        }
        catch (e) {}
    }
}

//返回创建的对象或错误信息
if (!xmlHttpRequest)
    alert("Error creating the XMLHttpRequest object.");
else
    return xmlHttpRequest;
}

/* 初始化页面函数 */
function init()
{
    //获得关键字输入控制
    var oKeyword = document.getElementById("keyword");
    //阻止浏览器自动填充功能
    oKeyword.setAttribute("autocomplete", "off");
    //重置关键字, 并设置焦点。
    oKeyword.value = "";
    oKeyword.focus();
    //设置检查关键字更新的超时值
    setTimeout("checkForChanges()", 500);
}
```

```
/* 将关键字添加值数组中 */
function addToCache(keyword, values)
{
    //在缓存中创建一个新的串入口
    oCache[keyword] = new Array();
    //将所有关键字的入口加入缓存 for(i=0; i<values.length; i++)
        oCache[keyword][i] = values[i];
}

/*
    检查参数中的关键字是否在缓存中, 或找到缓存中与关键字前缀最长匹配, 并根据当前关键字参数把
    前缀加入缓冲
*/
function checkCache(keyword)
{
    //检查缓存中是否有当前关键字
    if(oCache[keyword])
        return true;
    //尝试找到匹配的最长前缀
    for(i=keyword.length-2; i>=0; i--)
    {
        //计算出当前前缀关键字
        var currentKeyword = keyword.substring(0, i+1);
        //检查缓存中是否有当前前缀关键字
        if(oCache[currentKeyword])
        {
            //缓存中已有的当前关键字结果
            var cacheResults = oCache[currentKeyword];
            //匹配关键字的结果在当前缓冲结果中
            var keywordResults = new Array();
            var keywordResultsSize = 0;
            //尝试找出所有符合当前前缀的结果
            for(j=0; j<cacheResults.length; j++)
            {
                if(cacheResults[j].indexOf(keyword) == 0)
                    keywordResults[keywordResultsSize++] = cacheResults[j];
            }
            //将所有关键词前缀结果加入缓存
            addToCache(keyword, keywordResults);
            return true;
        }
    }
}
```

```
    }  
  }  
  //如果没有匹配的结果  
  return false;  
}  
  
/* 生成 HTTP 请求用于找到当前关键字的建议*/  
function getSuggestions(keyword)  
{  
  /*如果关键字不为空且最后按的键不是上下箭头键*/  
  if(keyword != "" && !isKeyUpDownPressed)  
  {  
    //检查关键词是否在缓存中  
    isInCache = checkCache(keyword);  
    //如果关键字在缓存中  
    if(isInCache == true)  
    {  
      //从缓存中获取结果  
      httpRequestKeyword=keyword;  
      userKeyword=keyword;  
      //显示缓存中的结果  
      displayResults(keyword, oCache[keyword]);  
    }  
    //如果关键字不在缓存中,生成 HTTP 请求  
    else  
    {  
      if(xmlHttpGetSuggestions)  
      {  
        try  
        {  
          /*如果 XMLHttpRequest 对象没有忙于前一个请求*/  
          if (xmlHttpGetSuggestions.readyState == 4 ||  
              xmlHttpGetSuggestions.readyState == 0)  
          {  
            httpRequestKeyword = keyword;  
            userKeyword = keyword;  
            xmlHttpGetSuggestions.open("GET",  
                                        getFunctionsUrl + encode(keyword), true);  
            xmlHttpGetSuggestions.onreadystatechange =  
              handleGettingSuggestions;  
          }  
        }  
      }  
    }  
  }  
}
```

```
        xmlhttpGetSuggestions.send(null);
    }
    //如果 XMLHttpRequest 对象忙
    else
    {
        //保留用户需要的关键字
        userKeyword = keyword;
        //清除先前设置的超时请求
        if(timeoutId != -1)
            clearTimeout(timeoutId);
        //0.5 秒后再试一次
        timeoutId = setTimeout("getSuggestions(userKeyword);", 500);
    }
}
catch(e)
{
    displayError("Can't connect to server:\n" + e.toString());
}
}
}
}

/* 将 XML 节点的所有子节点转换为数组 */
function xmlToArray(resultsXml)
{
    //初始化结果数组
    var resultsArray= new Array();
    //在得到的 XML 节点中循环取得结果
    for(i=0;i<resultsXml.length;i++)
        resultsArray[i]=resultsXml.item(i).firstChild.data;
    //将节点的内容转化为数组
    return resultsArray;
}

/* 处理服务器针对所请求关键字建议的响应 */
function handleGettingSuggestions()
{
    //如果过程已经结束, 决定如何处理数据
    if (xmlhttpGetSuggestions.readyState == 4)
```



```
{
    //只有 HTTP 状态为 ok 时
    if (xmlHttpGetSuggestions.status == 200)
    {
        try
        {
            //处理服务器响应
            updateSuggestions();
        }
        catch(e)
        {
            //显示错误信息
            displayError(e.toString());
        }
    }
    else
    {
        displayError("There was a problem retrieving the data:\n" +
            xmlHttpGetSuggestions.statusText);
    }
}

/* 处理服务器响应函数*/
function updateSuggestions()
{
    //获取服务器响应
    var response = xmlHttpGetSuggestions.responseText;
    //判断是否服务器错误
    if (response.indexOf("ERRNO") >= 0
        || response.indexOf("error:") >= 0
        || response.length == 0)
        throw(response.length == 0 ? "Void server response." : response);
    //获取 document 元素
    response = xmlHttpGetSuggestions.responseXML.documentElement;
    //初始化新的函数数组
    nameArray = new Array();
    //检查是否有关于关键字的结果
    if(response.childNodes.length)
    {
```

```
/* 名字数组从 document 元素转化的数组中获取新的函数*/
nameArray= xmlToArray(response.getElementsByTagName("name"));
}
//检查是否已经搜索其他的关键字
if(httpRequestKeyword == userKeyword)
{
    //显示结果数组
    displayResults(httpRequestKeyword, nameArray);
}
else
{
    //在缓存中添加结果
    //不需要显示已经失效的结果
    addToCache(httpRequestKeyword, nameArray);
}
}

/* 使用当前建议生成列表*/
function displayResults(keyword, results_array)
{
    //开始创建 HTML 表格存放结果
    var div = "<table>";
    //如果搜索的结果不在缓存中, 将其添加入缓存.
    if(!oCache[keyword] && keyword)
        addToCache(keyword, results_array);
    //如果结果数组为空, 显示一个信息
    if(results_array.length == 0)
    {
        div += "<tr><td>No results found for <strong>" + keyword +
            "</strong></td></tr>";
        //设置一个标志表示没有查询到结果
        //结果计数器复位
        hasResults = false;
        suggestions = 0;
    }
    //显示结果
    else
    {
        //复位当前选择的建议
        position = -1;
    }
}
```

```

//复位标志上下箭头键是否按下的标志
isKeyUpDownPressed = false;
/*设置表示关键字结果的标志位 */
hasResults = true;
//获取缓存中结果的数目
suggestions = oCache[keyword].length;
//循环将结果生成 HTML 列表
for (var i=0; i<oCache[keyword].length; i++)
{
    //获取当前函数
    crtFunction = oCache[keyword][i];
    //为当前函数设置链接函数名的字符串
    crtFunctionLink = crtFunction;
    //重置串链接
    while(crtFunctionLink.indexOf("_") !=-1)
        crtFunctionLink = crtFunctionLink.replace("_","-");
    //开始创建链接到当前函数的 PHP 帮助页面的 HTML 行
    div += "<tr id='tr" + i +
        "' onclick='location.href=document.getElementById(\"a" + i +
        "\").href;' onmouseover='handleOnMouseOver(this);' " +
        "onmouseout='handleOnMouseOut(this);'>" +
        "<td align='left'><a id='a" + i +
        "' href='" + phpHelpUrl + crtFunctionLink + ".php";
    //检查当前链接名字长度是否超出能显示函数名字字符数目的最长限制
    if(crtFunction.length <= suggestionMaxLength)
    {
        //将前缀匹配的函数名字和关键字绑定在一起
        div += "'><b>" +
            crtFunction.substring(0, httpRequestKeyword.length) +
            "</b>"
        div += crtFunction.substring(httpRequestKeyword.length,
            crtFunction.length) +
            "</a></td></tr>";
    }
    else
    {
        //检查当前关键字长度是否超出可显示字符数目限制
        if(httpRequestKeyword.length < suggestionMaxLength)
        {
            /*将前缀匹配的函数名字和关键字绑定在一起*/

```

```
        div += "'><b>" +
            crtFunction.substring(0, httpRequestKeyword.length) +
            "</b>"

        div += crtFunction.substring(httpRequestKeyword.length,
            suggestionMaxLength) +
            "</a></td></tr>";
    }
    else
    {
        //绑定全部函数名字
        div += "'><b>" +
            crtFunction.substring(0, suggestionMaxLength) +
            "</b></td></tr>"
    }
}
}
}
//创建 HTML 表格结束
div += "</table>";
//获取建议和卷轴对象
var oSuggest = document.getElementById("suggest");
var oScroll = document.getElementById("scroll");
//卷到列表顶端
oScroll.scrollTop = 0;
//更新建议列表使其可视
oSuggest.innerHTML = div;
oScroll.style.visibility = "visible";
//如果输入当前关键字前已经请求到结果
if(results_array.length > 0)
    autocompleteKeyword();
}

/* 周期性检查输入的关键字是否变化*/
function checkForChanges()
{
    //获取关键字对象
    var keyword = document.getElementById("keyword").value;
    //检查关键字是否为空
    if(keyword == "")
    {
```

```
// 隐藏建议
hideSuggestions();
// 复位关键字
userKeyword="";
httpRequestKeyword="";
}
// 设置新检查计时器
setTimeout("checkForChanges()", 500);
// 检查是否发生变化
if((userKeyword != keyword) &&
    (autocompletedKeyword != keyword) &&
    (!isKeyUpDownPressed))
// 更新建议
    getSuggestions(keyword);
}

/* 处理按键函数 */
function handleKeyUp(e)
{
    // 获取事件
    e = (!e) ? window.event : e;
    // 获取事件目标
    target = (!e.target) ? e.srcElement : e.target;
    if (target.nodeType == 3)
        target = target.parentNode;
    // 获取按键的字符代码
    code = (e.charCode) ? e.charCode :
        ((e.keyCode) ? e.keyCode :
        (e.which) ? e.which : 0));
    // 检查事件是否是按键抬起
    if (e.type == "keyup")
    {
        isKeyUpDownPressed = false;
        // 检查是否对当前字符感兴趣
        if ((code < 13 && code != 8) ||
            (code >= 14 && code < 32) ||
            (code >= 33 && code <= 46 && code != 38 && code != 40) ||
            (code >= 112 && code <= 123))
        {
            // 简单忽略不感兴趣的字符
```

```
    }
    else
    /* 如果按了 Enter 键, 转向当前函数的 PHP 帮助页面 */
    if(code == 13)
    {
        //检查是否有函数是当前选择的
        if(position>=0)
        {
            location.href = document.getElementById("a" + position).href;
        }
    }
    else
    //如果按下↓键则转向下一个建议
    if(code == 40)
    {
        newTR=document.getElementById("tr"+(++position));
        oldTR=document.getElementById("tr"+(--position));
        //释放已前选择的建议
        if(position>=0 && position<suggestions-1)
            oldTR.className = "";
        //选择新的建议更新关键字
        if(position < suggestions - 1)
        {
            newTR.className = "highlightrow";
            updateKeywordValue(newTR);
            position++;
        }
        e.cancelBubble = true;
        e.returnValue = false;
        isKeyUpDownPressed = true;
        //如果当前窗口失效, 向下滚动。
        if(position > maxVisiblePosition)
        {
            oScroll = document.getElementById("scroll");
            oScroll.scrollTop += 18;
            maxVisiblePosition += 1;
            minVisiblePosition += 1;
        }
    }
    else
```

```
//如果按下↑键则转向前一个建议
if(code == 38)
{
    newTR=document.getElementById("tr"+(--position));
    oldTR=document.getElementById("tr"+(++position));
    //释放前一个选择的位置
    if(position>=0 && position <= suggestions - 1)
    {
        oldTR.className = "";
    }
    //选择新的建议,更新关键字.
    if(position > 0)
    {
        newTR.className = "highlightrow";
        updateKeywordValue(newTR);
        position--;
        //如果当前窗口失效,向上滚动.
        if(position<minVisiblePosition)
        {
            oScroll = document.getElementById("scroll");
            oScroll.scrollTop -= 18;
            maxVisiblePosition -= 1;
            minVisiblePosition -= 1;
        }
    }
    else
        if(position == 0)
            position--;
    e.cancelBubble = true;
    e.returnValue = false;
    isKeyUpDownPressed = true;
}
}
}

/* 根据当前选择的建议更新关键字*/
function updateKeywordValue(oTr)
{
    //获取关键字对象
    var oKeyword = document.getElementById("keyword");
```

```
//获取当前函数链接
var crtLink = document.getElementById("a" +
                                     oTr.id.substring(2,oTr.id.length)).toString();
//重置_ , 不考虑.php扩展
crtLink = crtLink.replace("-", "_");
crtLink = crtLink.substring(0, crtLink.length - 4);
//更新关键字的值
oKeyword.value = unescape(crtLink.substring/phpHelpUrl.length, crtLink.length));
}

/* 删除所有建议的风格*/
function deselectAll()
{
    for(i=0; i<suggestions; i++)
    {
        var oCrtTr = document.getElementById("tr" + i);
        oCrtTr.className = "";
    }
}

/* 处理鼠标进入建议区域事件*/
function handleOnMouseOver(oTr)
{
    deselectAll();
    oTr.className = "highlightrow";
    position = oTr.id.substring(2, oTr.id.length);
}

/* 处理鼠标离开建议区域事件*/
function handleOnMouseOut(oTr)
{
    oTr.className = "";
    position = -1;
}

/* 退出串*/
function encode(uri)
{
    if (encodeURIComponent)
```



```
        return encodeURIComponent(uri);
    }

    if (escape)
    {
        return escape(uri);
    }
}

/*隐藏含有建议的层*/
function hideSuggestions()
{
    var oScroll = document.getElementById("scroll");
    oScroll.style.visibility = "hidden";
}

/*选择一个文本对象的范围作为参数传递*/
function selectRange(oText, start, length)
{
    //检查是否在 IE 或 FF 环境下
    if (oText.createTextRange)
    {
        //IE
        var oRange = oText.createTextRange();
        oRange.moveStart("character", start);
        oRange.moveEnd("character", length - oText.value.length);
        oRange.select();
    }
    else
        //FF
        if (oText.setSelectionRange)
        {
            oText.setSelectionRange(start, length);
        }
    oText.focus();
}

/* 自动完成输入关键字*/
function autocompleteKeyword()
{
```

```

//获取关键字对象
var oKeyword = document.getElementById("keyword");
//复位选择建议的位置
position=0;
//释放所有的建议
deselectAll();
//突出显示选择的建议
document.getElementById("tr0").className="highlightrow";
//根据建议更新关键字的值
updateKeywordValue(document.getElementById("tr0"));
//申请提前输入风格
selectRange(oKeyword,httpRequestKeyword.length,oKeyword.value.length);
//设置自动完成词为关键字值
autocompletedKeyword=oKeyword.value;
}

/* 显示错误信息*/
function displayError(message)
{
//如果 debugMode 值为 true, 显示错误信息更详细的技术细节
alert("Error accessing the server! "+
      (debugMode ? "\n" + message : ""));
}

```

(11) 现在可以开始测试代码了。在浏览器中输入地址 <http://localhost/ajax/suggest/>。假如正在查询 `strstr` 的帮助页, 在输入“s”之后, 将显示出由这个字母开始的函数列表, 如图 6.3 所示。

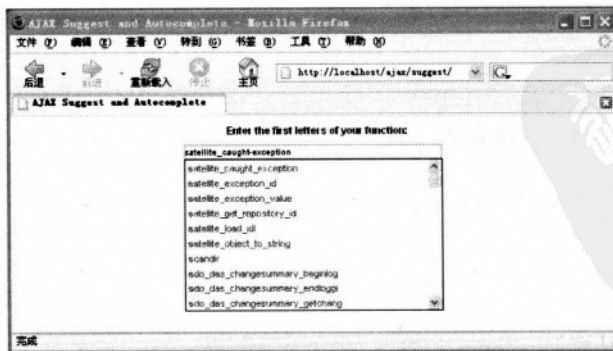


图 6.3 PHP 知道许多以“s”打头的函数

(12) PHP 有许多 s 打头的函数。可以看到第一个匹配的函数在查询框中自动完成, 而且

有一个很长的要滚动的函数列表。输入单词的第二个字母 t。

(13) 函数列表像预期地一样减少了。通过继续输入名字, 或者使用键盘上的上下箭头, 或者使用鼠标, 来查找感兴趣的函数。查找到它的时候, 按下 Enter 键或是使用鼠标单击, 结果如图 6.4 所示。

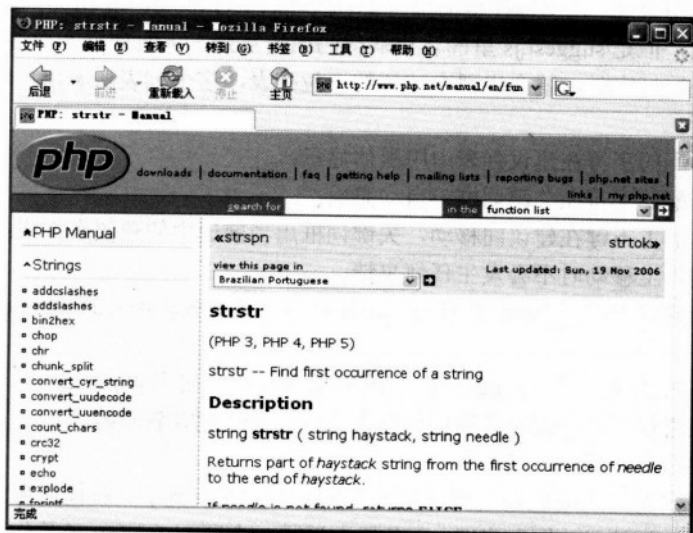


图 6.4 strstr 的 PHP 文档

## 程序说明

让我们从 index.html 文件开始。

这个脚本让人感兴趣的部分是滚动域可以用 DHTML 实现。关于滚动的内容可以从 <http://www.dyn-web.com/dhtml/scroll/> 找到。让页面的一部分有个滚动栏在旁边就是要有两层, 一层嵌入在另一层。在例子中, div scroll 和 div suggest 用到了这个技巧。

外层是 scroll。它有固定的宽度和高度, 而且它最有用的属性是 overflow (上下滑动)。通常情况下, 文本框的边界受文本框内容的限制。在某些情况下, 框会溢出, 即它的部分内容会位于框外。在 CSS 中, 这个属性说明了当一个元素溢出它的范围后会发生什么。在 <http://www.w3.org/TR/REC-CSS2/visufx.html> 中可以找到 overflow 可能的值。

另一个有趣的事情是如何能够让一个对象居中。典型的 align=center 属性在 XHTML 1.0 中是无效的, 因此需要找一个变通的办法。解决办法是把要居中的对象的 margin 属性设置成 auto。如果有一个有效的 doctype, IE 6 会使元素自动居中。否则, 在早期版本的情况下, 这个属性会被忽视。对于早期版本的 IE, 需要为想要居中元素的父元素的 text-align 属性设置成 center。这是因为 IE 没有正确地把 text-align 属性应用到所有的块元素中, 而是仅仅应用到

使事务工作的内部元素中。

输入控制处理 `keyup` 事件，事实上它触发了一个取出并显示当前关键词建议的过程。`div` 中的内容处理 `click` 事件，当用户在建议区域外单击时不会显示建议，直到用户改变当前关键词才显建议。

对于这个应用来说，几乎每件事情都与 JavaScript、DOM 和 CSS 相关。服务端非常简单，而且它易于实现，但是 `suggest.js` 里的客户端代码比较复杂。下面列举客户端实现的功能。

- 当用户开始打字时，就会出现有建议的下拉列表，这个列表随着用户输入新单词或者删除字母而更新；
- 第一个匹配的字符在建议列表中用黑体显示；
- 第一个匹配的字符在关键词框中自动完成；
- 用键盘上的上下键在建议间移动，关键词框用当前选中的建议来完成；
- 鼠标在建议上移动时不会发生任何事情；
- 当在一个建议上按下 `Enter` 键或者单击鼠标时，网页被重定向到 `php.net` 网站上的 PHP 帮助页；
- 当用户在关键词框内按 `Enter` 键时，网页也会被重定向到 `php.net`；
- 当鼠标在建议列表外或者关键词框外单击时，建议列表就隐藏了；
- 建议缓存在客户端。

有一个函数，它定期地检查关键词是否改变了。如果改变了，就向服务器发送一个包含当前关键词的 HTTP 请求。作为响应，服务器页返回一个匹配的 PHP 函数作为那个关键词的建议。客户端浏览器在下拉列表中显示这些建议。用户可以用上下箭头或者鼠标浏览建议。输入新字母或者删除字母，建议列表就会更新。在看完前面部分的图片，而且对这个处理过程有了大体概念后，现在来看一下这些是如何实现的。

`createXmlHttpRequestObject` 是用来创建 `XMLHttpRequest` 请求的函数。

`Init` 函数只引发关键词框的自动完成属性。这么做是为了阻止浏览器初始化它们自己的自动完成引擎。因为设置“`autocomplete = off`”，根据 XHTML 定义这不是一个有效的属性，这样会使 HTML 失效。这个属性由 Microsoft 提出并被大多数浏览器采用。

检查关键词是否改变的函数是 `checkForUpdates`。如果改变了，它启动更新建议列表的处理过程。为了浏览建议列表，可以使用 `handleKeyUp` 函数。将在本章后续部分介绍更多相关内容。

我们已经讨论了缓存结果，这是一种在应用程序中非常有用的技术。因此，用两个函数来处理缓存对象——`checkCache` 和 `addToCache`。

`checkCache` 函数检查一个给定的关键词是否在缓存中。如果不在，它试图在缓存值列表中找到该关键词的最长匹配前缀，通过调用 `addToCache` 函数将这些匹配前缀添加到缓存中。

`addToCache` 函数为给定关键词向缓存插入列表，列表代表该关键词的建议。

`getSuggestions` 函数用来取回新建议。如果当前关键词已经在缓存中（`checkCache` 函数），直接用这些缓存的建议来填充建议列表。如果正在处理请求，那么请求的关键词将存储起来，

并且要为该功能设置超时值。这样一来，能保证保存了最后一个关键词，而且不能产生一个使用该关键词的服务器调用，但是在当前请求完成后，就能立刻使用这个最后的关键词初始化一个服务器调用。

`handleGettingSuggestions` 函数检查到如果服务器请求完成并且没有发生错误时，就调用 `updateSuggestions` 函数。

`updateSuggestions` 函数检查是否有必要更新建议列表。我们检查是否在服务器调用期间又有新的服务器请求。通过这种方法，可以知道用户是否修改了当前关键词。如果有修改，就不需要显示这些获取来的用户不再感兴趣的建议了。尽管如此，客户还是要缓存所有来自服务器的建议。

`xmlToArray` 函数将 XML 节点集合转换成一个数组。

真正建立建议列表的函数是 `displayResults`。它的参数是关键词以及数组形式的函数访问列表。第一件要做的事就是缓存当前结果，这样当我们再次查找同样的关键词时，不需要再次调用 web 服务器。搜寻数组中的所有建议并动态建一个包含这些内容的表。如果没有可获得的建议，就显示一个消息。

`updateKeywordValue` 函数负责使用包含在建议中的当前选中的指定 `tr` 对象值来更新当前关键词。

`hideSuggestions` 函数隐藏了包含当前关键词的所有建议的 `div` 元素。

`deselectAll` 函数取消选择当前选中的建议。

`handleOnMouseOver` 和 `handleOnMouseOut` 函数处理当光标移开或放在一个建议的 `tr` 范围上时发生的事件。这些函数在事件发生的同时更新了建议的样式。

`encode` 函数转义参数字符串，并且在调用服务器页时被 `getSuggestions` 函数调用。

下一步，我们将谈一谈 `handleKeyUp` 函数，它用于在结果和提交之间进行导航。我们仅对少数关键字感兴趣，其他键就可以忽略了。在使用前，要保证编码在每个浏览器上都能工作。为此，需要写几行代码来测试。

我们需要知道键的编码才能去考虑需要哪些字符。接收到的参数 `event` 对象有一个属性 `keyCode`，`keyCode` 存有按下键的编码。表 6-1 包括了大多数特定键。

表 6-1 特定键

键	代 码	键	代 码
Backspace	8	Esc	27
Tab	9	Page Up	33
Enter	13	Page Down	34
Shift	16	End	35
Ctrl	17	Home	36
Alt	18	Left Arrow	37
Pause/Break	19	Up Arrow	38
Caps Lock	20	Right Arrow	39

续表

键	代 码	键	代 码
Down Arrow	40	F6	117
Print Screen	44	F7	118
Delete	46	F8	119
F1	112	F9	120
F2	113	F10	121
F3	114	F11	122
F4	115	F12	123
F5	116		

按下 Enter 键（码 13）时，网页提交到 php.net，如果有建议被选择，它就显示当前选择建议的帮助说明。单击上下箭头时，当前选择建议会随着上下移动。当前关键词也会随着当前选择建议的值来更新。在它们修改关键词时没有处理任何其他按键，我们已经使用 `checkForChanges` 函数处理这部分。

另一个问题就是有多于十个的建议可获得时，就会有一个可滚动的 `div` 域。就像前面介绍的，让用户能够通过使用上下箭头键浏览结果。如果用户要的是一个当前不可见的结果，为了让此结果可见，需要在域中滚动。为实现这点，可以保存当前可见结果的最小和最大位置。这就像是有一个窗口，它可以根据箭头的移动和当前选择结果，在结果间移动。

`selectRange` 和 `autocompleteKeyword` 函数在提前输入上使用了一个技巧，就是用第一个建议的剩下字母来自动完成当前关键词。缺失的部分用高亮显示的文本增加到当前关键词后。`select()`方法选择所有文本，因此仅仅选择文本一部分是不可能的。为做到部分选择，IE 提供了一种解决方案，而 Mozilla/Firefox 提供了另一种解决方案。同一问题在所有浏览器中表现不同，这已不是第一次遇到的现象，所以必须考虑每一种情况并分别解决它们。在 Firefox 中，问题是相似的，因为只有一个函数做所有的工作——`setSelectionRange`，这个函数有两个参数——选择的开始位置和选择的长度。在 IE 中，为了达到同样的目的，必须使用 `TextRange` 对象。进一步仔细地看一下这个对象，因为它可能对将来的应用很有用，而且在本应用中，也需要知道它是如何作用的。

`TextRange` 对象能执行查询或选择文本的任务。文本域允许选择文本中的字符、单词和句子。它们中的每一个都是该对象的逻辑单元。为了使用这样的对象，必须遵循下面的步骤：

- 创建文本域。
- 对选中的文本应用一种方法。

可以复制文本，在文本中查询，还可以像例子中一样选择文本的一部分。可以在 `body`、`textarea` 或者 `button` 元素中调用 `createTextRange` 方法来创建对象。

每个对象都有指定文本范围的开始和结束位置。当创建新文本域时，开始和结束位置默认包含了整个内容。要修改文本域范围，可以使用 `move`、`moveStart` 和 `moveEnd` 函数。每个函数都有两个参数——第一个参数指定逻辑单元，第二个参数指定要移动单元的数目。结果

包含了被移动单元的数目。select 方法使选择等价于当前对象。要查看完整的对象权能，单击下面的 MSDN 链接 <http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dyncontent/textrange.asp>。

接收完建议并将它们插入页面后，需要用第一个建议值来自动完成关键词，这是通过使用上面描述的 selectRange 函数完成的。

对于错误处理部分，使用 displayError 函数，它显示了一个含有错误消息的警告框。

好了！我们已经看过应用的客户端如何运行了，接下来看一下服务器端。

对于服务器端，事情非常简单。Suggest.php 文件接收客户传送的表示要查找关键词的参数，然后它调用 suggest.class.php 中 Suggest 类的方法，来查找与关键词匹配的建议。web 服务器返回一个包含了匹配当前关键词的 PHP 函数的 XML 响应。就像我们自己看到的，工作都集中于客户端，服务器端基本上没有太多内容。

使用 Ajax 建议和自动完成方案实现的 PHP 帮助，比我们最初想到的更有挑战性。上面已经提到，我们有许多事情要处理。希望这些问题对于以后的应用程序处理能够提供有用的解决方案，同时也能作为学习其他应用的基础。

## 6.3 小结

在本章开始给出了建议和自动完成的定义。我们看到了这些概念不论在代码编辑器还是在操作系统控制台领域都很流行。

本章开发的应用提供了一个在线 PHP 帮助，链接的是官方帮助文档 [www.php.net](http://www.php.net)。

从很多方面看，这里有很多和 Google Suggest 相似的功能，但是它还有一些额外的特征。



## 使用 SVG 实现 Ajax 实时绘制图表

可缩放矢量制图 (SVG) 是一种新兴的技术, 就像 Flash 技术一样, 它将有机会成为 Web 图形下一代的重要技术。SVG 是一种用来定义二维图形的语言。SVG 和网页必然没有的联系, 但是它适合网页图片的处理, 补充浏览器自身提供的功能。目前为止, 很多 SVG 实现和标准都很不完善, 希望在不久的将来会有所改进。

SVG 是 2003 年 1 月份由万维网联盟 (W3C) 推出的标准。为 SVG 的形成做出贡献的知名公司有 Sun Microsystems、Adobe、Apple、IBM 和 Kodak 等。当前的规范是 SVG 1.2。W3C 的 SVG 可移动技术、SVG 打印和 sXBL 得到了大多数浏览器和平台的支持。

SVG 的主要特性有:

• SVG 图像是由 XML 格式定义的, 所以它可以很容易通过已有的编辑软件和解析器来编辑和处理;

- SVG 图像具有可缩放性, 可以在不降低质量的条件下进行缩放、调整大小和旋转;
- SVG 具有字体元素, 所以可以很好地保留图片的文字和图形显示;
- SVG 包含了灵活的可以声明的元素;
- SVG 允许包含多名字空间的 XML 应用;
- SVG 允许使用 XML DOM 和 SVG uDOM 的子集, 并基于脚本处理文档树。

对于一个刚开始接触 SVG 的初学者, 可以查看以下资源:

- SVG W3C 网站 <http://www.w3.org/Graphics/SVG>;
- 讲解 SVG 的网站 [http://www.w3schools.com/svg/svg\\_intro.asp](http://www.w3schools.com/svg/svg_intro.asp);
- 提供很多非常有价值的 SVG 链接的网站 <http://www.svgi.org>;
- SVG 的参考手册 [http://www.w3schools.com/svg/svg\\_reference.asp](http://www.w3schools.com/svg/svg_reference.asp);
- SVG 文档结构讲解 <http://www.w3.org/TR/SVG/struct.html>;
- SVG 例子 <http://www.carto.net/papers/svg/samples/>和 <http://svg-whiz.com/samples.html>。

### 7.1 使用 Ajax 和 SVG 实现实时图表

在开始之前, 请确保浏览器支持 SVG。即将学习的这个例子已经在 Firefox 1.5、支持 Adobe



SVG Viewer 的 Internet Explorer 和 Apache Batik 上通过了测试。可以通过访问 <http://ajaxphp.packpub.com> 测试在线演示。

Firefox 浏览器对 SVG 提供了完善的支持。但是在 SVG 的早期版本中, SVG 运行确实存在着一些问题, 在编写程序的过程中必须要考虑这些问题, 并且性能也不很理想。

要加载 SVG 到 Internet Explorer, 首先必须安装 SVG 插件。我们测试过程中使用的是 Adobe 公司的 SVG 插件, 可以通过 <http://www.adobe.com/svg/viewer/install/main.html> 下载该插件。安装过程非常简单, 只需要下载 SVGView.exe 安装文件并执行。第一次加载 SVG 页面时, 要求确认是否同意声明中的条款。

最后, 我们使用了 Apache 的 Batik SVG 查看器对程序进行了测试。在这种环境下, 直接加载 SVG 文件就可以, 因为它不支持通过 HTML 来加载 SVG 脚本(也许您想检查一下 Batik 对 DOM 浏览的支持, 它可以很好地显示层次结构中的 SVG 节点)。

在本章的学习中, 将创建一个简单的图表应用, 它通过异步方式从 PHP 脚本获取输入数据。它可以产生任何的数据, 但是在例子中使用一个简单的算法来产生随机数。图 7.1 展示了程序的示例输出。

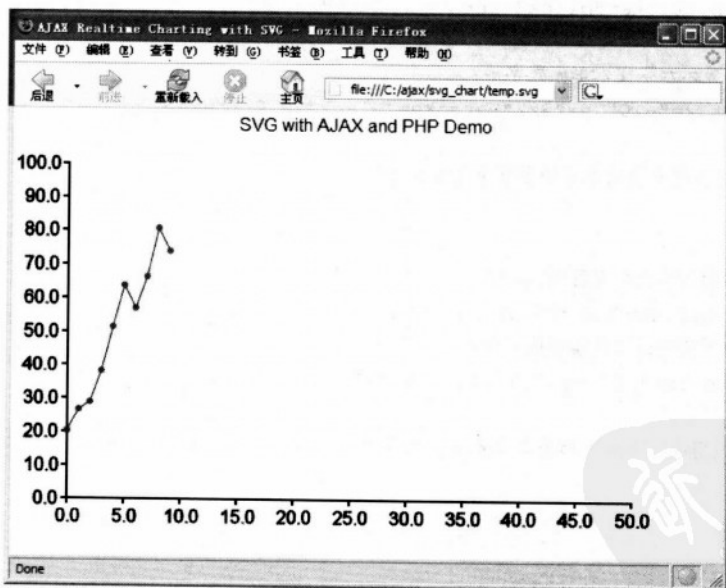


图 7.1 SVG 图表

图 7.1 显示的实际上就是静态 SVG 文件 temp.svg 的内容, 它描述了程序运行过程中输出的一个快照, 而不是程序运行过程中的屏幕截图。temp.svg 这个脚本存放在下载源码的本章目录中, 如果浏览器支持 SVG, 可以通过浏览器直接访问它。

我们首先来看一下 temp.svg 的内容, 感觉一下想通过 JavaScript 代码动态生成什么。注

意，SVG 脚本既可以在客户端也可以在服务器端生成。在程序中，服务器端只生成随机的坐标，客户端的 JavaScript 使用这些坐标来形成 SVG 输出。

来看一下剥离的 temp.svg 的代码：

```
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" onload="init(evt)">
  <a xlink:href="http://ajaxphp.packtpub.com">
    <text x="200" y="20">
      SVG with Ajax and PHP Demo
    </text>
  </a>
  <!--所有的图表元素都通过 50, 50 进行了分组和转换 -->
  <g transform="translate(50, 50)">
    <!--对所有的坐标轴元素（线和文本标注）进行分组-->
    <g>
      <!--定义图表格子的轴和单元的路径 -->
      <path stroke="black" stroke-width="2" d="...这里定义路径..."/>

      <!--显示水平文本单元数字 -->
      <text x="-10" y="322" stroke="black">0.0</text>
      ...
      ...其他用来绘制水平和垂直单元的文本节点
      ...
    </g>
    <!--在图表节点之间画线 -->
    <path stroke="black" stroke-width="1" fill="none" d="...定义..."/>
    <!--使用蓝色的圆点绘制图表节点 -->
    <circle cx="00" cy="239.143" r="3" fill="blue"/>
    ...
    ...其他用来绘制水平和垂直单元的文本节点
    ...
  </g>
</svg>
```

让我们深入研究这个程序片段，确认图表的所有元素。SVG 格式支持元素组的概念，就是将<g>元素下的所有元素归为一组。在 temp.svg 文件中有两个组：第一组包括了图表的所有元素，通过 (50, 50) 像素进行转换；第二组是第一组的一个子组，它包含了图表的坐标轴和数据。

SVG 知道如何处理众多的元素类型，这也是它有生机的原因（是的，SVG 是非常强大的）。在例子中使用了一些基本的元素：path（用来绘制坐标轴线和图表曲线）、test（用来绘

制坐标数据，当鼠标移动到一个坐标点上时动态显示坐标的值——这个特性不包含在这个程序片段中)和 `circle` (用来绘制圆点来代表坐标点)。

可以在下面连接中找到这些元素的文档：

[http://www.w3schools.com/svg/svg\\_path.asp](http://www.w3schools.com/svg/svg_path.asp);

[http://www.w3schools.com/svg/svg\\_circle.asp](http://www.w3schools.com/svg/svg_circle.asp);

[http://www.w3schools.com/svg/svg\\_text.asp](http://www.w3schools.com/svg/svg_text.asp)。

所有的路径都通过 `path` 来定义。在图 7.1 所示图表中看到的曲线是通过 `path` 元素中的代码绘制的，`path` 元素的完整代码如下：

```
//给图表节点连线
<path stroke="black" stroke-width="1" fill="none"
      d="M0,239.143 L10,220.286 L20,213.429 L30,185.571 L40,145.714
        L50,108.857 L60,129 L70,101.143 L80,58.2857 L90,78.4286"/>
```

从这段代码中可以看到一个细节，就是图表节点的 `mouseover` 和 `mouseout` 事件。在代码中，`mouseover` 事件（把鼠标放到节点上就触发这个事件）调用一个 JavaScript 函数在节点上方以文本形式显示节点的坐标值，`mouseout` 事件就是让显示的这个坐标值消失。可以在图 7.2 中看到这个特性，它动态显示 SVG 图表。

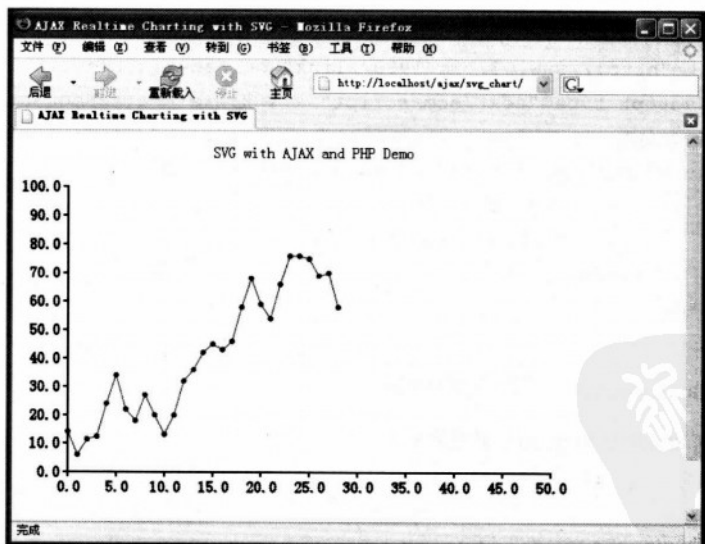


图 7.2 SVG 动态制图

### ⚠ 注意

在 Firefox 中，要想随时获取 SVG 图表动态产生的内容，在图表上单击右键，选择 `Select All`，然后再次右键单击图表，选择 `View Selection Source`。

现在已经知道要实现的是什么是了，让我们开始吧，是行动的时候了！

## 》》》 实现步骤——构建一个实时 SVG 图表

- (1) 首先在 ajax 文件夹中创建 svg\_chart 子目录。
- (2) 在 svg\_chart 目录中创建 index.html 文件，写入以下代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
  <head>
    <title>Ajax Realtime Charting with SVG</title>
  </head>
  <body>
    <embed src="chart.svg" width="600" height="450" type="image/svg+xml" />
  </body>
</html>
```

- (3) 然后创建 chart\_svg 文件，写入下面内容：

```
<svg width="100%" height="100%" xmlns=http://www.w3.org/2000/svg
xmlns:xlink="http://www.w3.org/1999/xlink" onload="init(evt)">
  <script type="text/ecmascript" xlink:href="ajaxRequest.js"/>
  <script type="text/ecmascript" xlink:href="realTimeChart.js"/>
  <a xlink:href="http://ajaxphp.packtpub.com">
    <text x="200" y="20">
      SVG with Ajax and PHP Demo
    </text>
  </a>
</svg>
```

- (4) 创建 ajaxrequest.js，写入下面代码：

```
//用来保存 XMLHttpRequest 对象的引用
var xmlHttp = null;

//创建一个 XMLHttpRequest 实例
function createXmlHttpRequestObject()
{
  //用来保存 XMLHttpRequest 对象的引用
  var xmlHttp;
  //这个可以在所有的浏览器下工作，除了 IE 6 和早期的版本
```

```
try
{
    //创建一个 XMLHttpRequest 对象
    xmlhttp = new XMLHttpRequest();
}
catch(e)
{
    //设定 IE6 和早期的版本
    var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
                                      "MSXML2.XMLHTTP.5.0",
                                      "MSXML2.XMLHTTP.4.0",
                                      "MSXML2.XMLHTTP.3.0",
                                      "MSXML2.XMLHTTP",
                                      "Microsoft.XMLHTTP");
    //尝试每一个 prog id 直到有一个能正常工作
    for (var i=0; i<XmlHttpVersions.length && !xmlhttp; i++)
    {
        try
        {
            //创建一个 XMLHttpRequest 对象
            xmlhttp = new ActiveXObject(XmlHttpVersions[i]);
        }
        catch (e) {}
    }
}
//返回创建的对象, 或者显示一个错误的消息
if (!xmlhttp)
    alert("Error creating the XMLHttpRequest object.");
else
    return xmlhttp;
}

//开始一个 Ajax 请求
function ajaxRequest(url, callback)
{
    //保存对函数的引用, 当收到服务器的响应时调用此函数
    var innerCallback = callback;
    //当第一次调用这个方法时, 建立一个 XMLHttpRequest 对象
    if (!xmlhttp) xmlhttp = createXmlHttpRequestObject();
    //当连接清除后, 初始化一个新的服务器请求
```

```

if (xmlHttp && (xmlHttp.readyState == 4 || xmlHttp.readyState == 0))
{
    xmlHttp.onreadystatechange = handleGettingResults;
    xmlHttp.open("GET", url, true);
    xmlHttp.send(null);
}
else
    //如果连接忙,延迟1秒重试
    setTimeout("ajaxRequest(url,callback)", 1000);

//当请求状态改变时,调用该函数
function handleGettingResults()
{
    //当事务处理完时就继续执行.
    if (xmlHttp.readyState == 4)
    {
        //HTTP200 状态表示事务已经成功执行
        if (xmlHttp.status == 200)
        {
            //执行回调函数,传递给服务器一个应答
            innerCallback(xmlHttp.responseText)
        }
        else
        {
            //显示一个错误的信息
            alert("Couldn't connect to server");
        }
    }
}
}
}

```

(5) 客户端的工作是由 RealTimeChart.js 来完成的。

```

//SVG 名字空间
var svgNS = "http://www.w3.org/2000/svg";
//SVG 文档处理句柄
var documentSVG = null;
//保存根<g>元素,它对图表的元素进行了分组.
var chartGroup = null;
//从服务器请求新数据的时间间隔
var updateInterval = 1000;

```

```
//用来转换坐标的像素
var x = 50, y = 50;
//图表的宽高
var height = 300, width = 500;
//坐标轴原点
var xt1 = 0, yt1 = 0;
//坐标轴上的最大值
var xt2 = 50, yt2 = 100;
//坐标轴水平和垂直刻度值
var xDivisions = 10, yDivisions = 10;
//初次显示时默认的文本高度和宽度(之后重新计算)
var defaultTextWidth = 30, defaultTextHeight = 20;
//保存对图表刻度的引用,以便重新计算位置
var xIndexes = new Array(xDivisions + 1);
var yIndexes = new Array(yDivisions + 1);
//保存显示的文字和显示当前选中的点
var currentNodeInfo;
//保存服务器生成的最新值
var lastX = -1, lastY = -1;
//共享的 svg 元素
var chartGroup, dataGroup, dataPath;
//初始化图表
function init(evt)
{
    /**** 准备图表的数据****/
    //获取一个 SVG 文档句柄
    documentSVG = evt.target.ownerDocument;
    //创建一个 <g>元素用来为图表中的元素分组
    chartGroup = documentSVG.createElementNS(svgNS, "g");
    chartGroup.setAttribute("transform", "translate(" + x + " " + y + ")");

    /****保存坐标轴 X 和 Y 上的数据****/
    axisGroup = documentSVG.createElementNS(svgNS, "g");
    //使用<path>元素中的数据创建 X 坐标轴
    axisPath = documentSVG.createElementNS(svgNS, "path");
    //坐标轴线是两个像素宽的黑色线。
    axisPath.setAttribute("stroke", "black");
    axisPath.setAttribute("stroke-width", "2");

    /**** 创建 X 和 Y 轴上的分界线 ****/
```

```
//创建 X 轴上分界线的路径定义文本
pathText = "M 0 " + height;
//对 X 轴添加分界线 (与上一条分界线不同)
for (var i = 0; i <= xDivisions; i++)
    pathText += "l 0 5 l 0 -5 " +
        ((i == xDivisions) ? "" : ("1 " + width/xDivisions + " 0"));
//创建 Y 轴上分界线的路径定义文本
pathText += "M 0 " + height;
//对 Y 轴添加分界线 (与上一条分界线不同)
for (var i = 0; i <= yDivisions; i++)
    pathText += "l -5 0 l 5 0 " +
        ((i == yDivisions) ? "" : ("1 0 -" + height / yDivisions));
//在路径中增加路径定义 (<d>属性)
axisPath.setAttribute("d", pathText);
//增加这个路径到坐标轴组中
axisGroup.appendChild(axisPath);

/****创建 X 和 Y 轴文本节点 ****/
//对 X 轴增加文本节点
for (var i = 0; i <= xDivisions; i++)
{
    //为分界线创建 <text> 节点
    t = documentSVG.createElementNS(svgNS, "text");
    //保存该节点供以后参考
    xIndexes[i] = t;
    //为<text>节点创建文本
    t.appendChild(documentSVG.createTextNode(
        (xt1 + i * ((xt2 - xt1) / xDivisions)).toFixed(1)));
    //设置<text> 节点的 X 和 Y 值
    t.setAttribute("x", i * width / xDivisions - defaultTextWidth / 2);
    t.setAttribute("y", height + 30 + defaultTextHeight);
    //当第一次加载图时, 我们不想显示文本节点
    t.setAttribute("stroke", "white");
    //增加<text>节点到坐标轴组
    axisGroup.appendChild(t);
}
//对 Y 轴增加文本节点
for (var i = 0; i <= yDivisions; i++)
{
    //为分界线创建<text>节点
```



```

t = documentSVG.createElementNS(svgNS, "text");
//保存该节点供以后参考
yIndexes[i] = t;
//为<text>节点创建文本
t.appendChild(documentSVG.createTextNode(
    (yt1 + i * ((yt2 - yt1) / yDivisions)).toFixed(1)));
//设置<text> 节点的 X 和 Y 值
t.setAttribute("x", -30 -defaultTextWidth);
t.setAttribute("y", height - i * height / yDivisions
    + defaultTextHeight / 2);
//当第一次加载图时, 我们不想让文本节点显示
t.setAttribute("stroke", "white");
//增加<text>节点到坐标轴组
axisGroup.appendChild(t);
}

//把坐标轴组加入图表
chartGroup.appendChild(axisGroup);

/****为<path>元素准备绘图数据****/
dataPath = documentSVG.createElementNS(svgNS, "path");
dataPath.setAttribute("stroke", "black");
dataPath.setAttribute("stroke-width", "1");
dataPath.setAttribute("fill", "none");
//为图表组加入数据路径
chartGroup.appendChild(dataPath);

/**** 结束初始化 ****/
//增加图表组到 SVG 文档
documentSVG.documentElement.appendChild(chartGroup);
//要想在 Firefox 中正确显示数据, 这个是必须的
setTimeout("refreshXYIndexes()", 500);
//初始化服务器请求
setTimeout("updateChart()", updateInterval);
}

//这个函数重绘轴线上的值, 并使它们可见 (在 Firefox 中文本的位置要正确)
function refreshXYIndexes()
{
    //重绘 X 轴上的文本节点

```

```
for (var i = 0; i <= xDivisions; i++)
    if (typeof xIndexes[i].getBBox != "undefined")
        try
        {
            textWidth = xIndexes[i].getBBox().width;
            textHeight = xIndexes[i].getBBox().height;
            xIndexes[i].setAttribute("x", i*width/xDivisions - textWidth/2);
            xIndexes[i].setAttribute("y", height + 10 + textHeight);
            xIndexes[i].setAttribute("stroke", "black");
        }
        catch(e) {}
//重绘 Y 轴上的文本节点
for (var i = 0; i <= yDivisions; i++)
    if (typeof yIndexes[i].getBBox != "undefined")
        try
        {
            twidth = yIndexes[i].getBBox().width;
            theight = yIndexes[i].getBBox().height;
            yIndexes[i].setAttribute("y", height-i*height/yDivisions + theight/2);
            yIndexes[i].setAttribute("x", -10 -twidth);
            yIndexes[i].setAttribute("stroke", "black");
        }
        catch(e) {}
}

//当鼠标放到一个节点时, 调用该函数显示坐标值
function createPointInfo(x, y, whereX, whereY)
{
    //确保在同一时刻只显示一个坐标值
    if (currentNodeInfo) removePointInfo();
    //创建文本节点
    currentNodeInfo = documentSVG.createElementNS(svgNS, "text");
    currentNodeInfo.appendChild(documentSVG.createTextNode(("+x+", "+y+")));
    //设置坐标
    currentNodeInfo.setAttribute("x", whereX.toFixed(1));
    currentNodeInfo.setAttribute("y", whereY - 10);
    //增加该节点到组中
    chartGroup.appendChild(currentNodeInfo);
}
```

```
//取消对坐标值的显示
function removePointInfo()
{
    chartGroup.removeChild(currentNodeInfo);
    currentNodeInfo = null;
}

//在图上绘制一个新的节点
function addPoint(X, Y)
{
    //保存该节点值
    lastX = X;
    lastY = Y;
    //从上一个生成的节点开始
    if (X == xt2)
        window.location.reload(false);
    //计算新的坐标值
    coordX = (X - xt1) * (width / (xt2 - xt1));
    coordY = height - (Y - yt1) * (height / (yt2 - yt1));
    //在图表上绘制蓝色节点
    var circle = documentSVG.createElementNS(svgNS, "circle");
    circle.setAttribute("cx", coordX); //X position
    circle.setAttribute("cy", coordY); //Y position
    circle.setAttribute("r", 3); //radius
    circle.setAttribute("fill", "blue"); //color
    circle.setAttribute("onmouseover", "createPointInfo(" + X + "," + Y + "," + coordX + "," + coordY + ")");
    circle.setAttribute("onmouseout", "removePointInfo()");
    chartGroup.appendChild(circle);
    //连线到新的节点
    current = dataPath.getAttribute("d"); //current path definition
    //更新路径的定义
    if (!current || current == "")
        dataPath.setAttribute("d", " M " + coordX + " " + coordY);
    else
        dataPath.setAttribute("d", current + " L " + coordX + " " + coordY);
}

//初始化异步请求, 重新获得新的图表数据
function updateChart()
```

```

{
    //构建查询串
    param = "?lastX=" + lastX + ((lastY != -1) ? "&lastY=" + lastY : "");
    //通过 Ajax 发送请求
    if (window.getURL)
        //Adobe's SVG Viewer 和 Apache Batik 支持该特性
        getURL("svg_chart.php" + param, handleResults);
    else
        //Mozilla 支持这个特性, 在 ajaxRequest.js 中实现
        ajaxRequest("svg_chart.php" + param, handleResults);
}

//回调函数用来读取服务器端返回的数据
function handleResults(data)
{
    //get the response data
    if (window.getURL)
        responseText = data.content;
    else
        responseText = data;
    //把一对坐标值分解成 X 和 Y 值
    var newCoords = responseText.split(",");
    //在坐标系中绘制新的节点
    addPoint(newCoords[0], newCoords[1]);
    //重启序列
    setTimeout("updateChart()", updateInterval)
}

```

(6) 最后创建服务端脚本 `svg_chart.php`:

```

<?php
//变量初始化
$maxX = 50; //our max X
$maxY = 100; //our max Y
$maxVariation = $maxY / 7; //在 Y 轴上, 一步的最大值
//客户端最后节点的 X 值 (默认为-1)
if (isset($_GET['lastX']))
    $lastX = $_GET['lastX'];
else
    $lastX = -1;
//客户端最后节点的 Y 值 (默认是随机数)

```

```

if (isset($_GET['lastY']))
    $lastY = $_GET['lastY'];
else
    $lastY = rand(0, $maxY);
//计算一个新的随机数
$randomY = (int) ($lastY + $maxVariation - rand(0, $maxVariation*2));
//确保新的 Y 在 0 和 $maxY 之间
while ($randomY < 0) $randomY += $maxVariation;
while ($randomY > $maxY) $randomY -= $maxVariation;
//生成新的数据对
$output = $lastX + 1 . ',' . $randomY;
//清除输出
if(ob_get_length()) ob_clean();
//发送头阻止浏览器缓存
header('Expires: Fri, 25 Dec 1980 00:00:00 GMT');
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . 'GMT');
header('Cache-Control: no-cache, must-revalidate');
header('Pragma: no-cache');
//把结果发送给客户端
echo $output;
?>

```

(7) 访问 [http://localhost/ajax/svg\\_char](http://localhost/ajax/svg_char)，为您的新图表自豪吧！

## 程序说明

让我们从服务端开始，简单看看代码中的关键部分。Svg\_char.php 脚本调用，生成一对 (x,y) 坐标，这些坐标被客户端显示到图表中，客户端支持把刚才生成的坐标值传递给服务器端，服务器端根据客户传来的数据生成新的坐标。这个情景很好的模拟了现实世界。

客户端产生的坐标作为参数 lastx 和 lasty 通过 Get 方法发送给服务器端。为了独立测试服务器端怎么处理客户端发送过来的参数，访问 [http://localhost/ajax/svg\\_chart.php?lastx=5&lasty=44](http://localhost/ajax/svg_chart.php?lastx=5&lasty=44)，如图 7.3 所示。

在客户端，所有的事情都是从 index.html 开始的，非常简单，它要做的就是加载 SVG 文件，目前最好的方法就是使用 <embed> 元素，但是 W3C 并不支持它（也可以使用 <object> 和 <iframe>，但是它们确实存在问题，可以查看 [http://www.w3schools.com/svg/svg\\_inhtml.asp](http://www.w3schools.com/svg/svg_inhtml.asp)）。

```

<body>
    <embed src="chart.svg" width="600" height="450" type="image/svg+xml" />
</body>

```

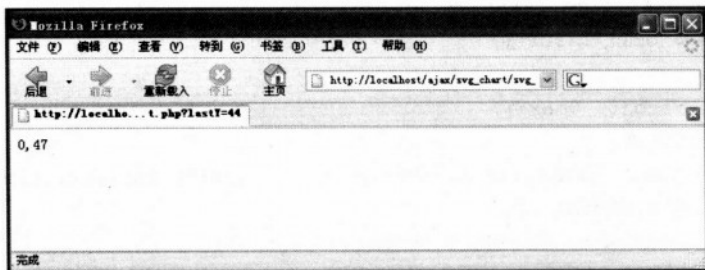


图 7.3

现在该看 `chart_svg` 文件了，但是它确实没有吸引人的地方。因为它的所有功能都是通过 JavaScript 脚本来做的，但是它包含了图表的标题：

```
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" onload="init(evt)">
  <script type="text/ecmascript" xlink:href="ajaxRequest.js"/>
  <script type="text/ecmascript" xlink:href="realTimeChart.js"/>
  <a xlink:href="http://ajaxphp.packtpub.com">
    <text x="200" y="20">
      SVG with Ajax and PHP Demo
    </text>
  </a>
</svg>
```

`Chart_svg` 文件引用了两个 JavaScript 文件。

`ajaxRequest.js` 包含了引擎，它使用 `XMLHttpRequest` 对象实现了 HTTP 的异步请求。当使用 Firefox 浏览器时 `realTimeChart.js` 使用这个引擎来获取服务器端的最新数据。对 Adobe SVG 和 Apache Batik 的实现，替换方法是通过它们的 `getURL` 方法来使用这些引擎提供的特定功能。查看 `realTimeChart.js` 中的 `updateChart` 方法可以了解更多的细节。

`ajaxRequest.js` 采用了和以前不同的编码风格。但是最重要的是要理解如下内容：

- 所有的 HTTP 请求都使用同一个 `XMLHttpRequest` 实例，而不是自动为每一个调用都创建一个新的 `XMLHttpRequest` 对象。通过这种方法，我们保证了收到的所有响应都是与请求的顺序一致的，这对图表程序来说是很重要的（对很多其他的程序来说也要求响应按一定的次序到达）。如果连接正忙于处理前一个请求，程序就会等待 1 秒，然后重试。这个技术也非常适合在访问 Web 服务端资源时使用。

- `ajaxRequest` 把访问的 URL 作为参数，并且当收到服务端的响应时调用回调函数。如果想灵活地从不同的地方访问同一功能，这种是一种很好的技术。

- `handle GettingResults()` 定义在 `ajaxRequest` 方法中。这是 JavaScript 的一个特性，使用它可以模拟 OOP 的功能。到目前为止我们还没有使用这个特性，因为只有写大型程序时才能确

实带来好处，并且对没有 OPP 技术经验的程序员来说学习它需要更长的时间。如果喜欢这个技术，可以很容易地在程序里实现它。

`realTimeChar.js` 包含了产生 SVG 图表的所有代码，代码中包含了这个技术的详细注释，这里我们简单地描述一下每一个方法。

- 当页面加载初始化图表的时候调用 `init()` 函数。这个方法产生 SVG 代码，这些代码包含了整个图表的坐标轴和初始化框架。首先，把坐标轴单位数字颜色绘制成白色的，使它们不可见。因为 Firefox SVG 的实现中在对屏幕进行着色之前不允许计算字体大小和正确位置，所以需要这么做。使用预先计算的值是不行的，因为格子是可以调整的并且坐标轴上也可以赋不同的值。为了克服这个缺点，`init()` 使用 `setTimeout()` 来控制在半秒钟后执行 `refreshXYIndexes()`。

- 即使在 Firefox 1.5 中，`RefreshXYIndexes()` 也能计算出坐标轴上数据的正确位置。设置完新坐标后，函数改变文字的颜色为黑色，使之可见。

- `Onmouseover` 功能调用 `CreatePointInfo()` 函数，显示图表中节点的坐标值。

- `Onmouseout` 功能调用 `removePointInfo()` 函数，让显示的节点坐标值信息消失。

- `UpdateChart()` 函数完成异步请求的初始化。`GetURL()` 方法 (Adobe SVG 和 Apache Batik 支持该方法) 将被使用，如果它可用，否则将使用 `ajaxRequest` 方法来进行请求。当请求服务器时，前面产生的坐标值对将通过 GET 方法传递，服务器端使用它来计算新的坐标值。

- `HandleResults()` 是一个回调方法，当接收到服务端的响应时它将被 `ajaxRequest` 调用。它主要是读取服务器端生成的新坐标值，并把它传递给 `addPoint()`。

- `AddPoint()` 接收新的坐标值并把它加入到图表中。这一坐标值在保存后才能使用，因为下一个请求时这个坐标值将发送给服务器，服务器使用它来为客户端计算下一个新的坐标值，并启用管理状态的简单机制：对于每一个请求，X 的值增 1，Y 的值随机生成，但是 Y 的生成考虑到了前面生成的 Y 值。

## 7.2 小结

无论是否喜欢 SVG，都必须承认它确实可以实现具有强大功能的网页。测试完它的强大功能之后，在未来的项目中就会考虑是否采用它。如果确实认可 SVG，可以找一个可视化编辑工具，它会使您创建 SVG 变得很容易。也可以考虑购买一些 SVG 书籍。

## Ajax 数据表格

与桌面应用程序相比，网页应用程序在数据表格方面确实存在严重的弱点。事实上，当使用数据表格换页的时候，或者更新数据表格中一个细节的时候，页面需要重新加载所有的数据，从可用性角度来看它对应用程序不利。从技术上说，完全重新加载页面效率很低，不必要地浪费了网络资源。

现在应该知道这个问题有更巧妙的解决方法。可以使用 ajax 来更新表格中的数据而不刷新页面。可以保持客户端页面美观的设计而且页面不会出现闪烁。刷新的仅仅是表数据而不是整个页面。

本章将接触一项新技术，使用 Extensible Stylesheet Language Transformation (XSLT) 和 XML Path Language (XPath) 来产生客户端输出。XSLT 和 Xpath 是 Extensible Stylesheet Language (XSL) 的一部分。XSLT 允许定义一组规则把 XML 文档转换成其他格式。Xpath 是一种强大的查询语言，从 XML 文档查询并获取数据。当使用它们来创建用户页面终端时，XSLT 能实现一个灵活的架构，服务器端输出数据为 XML 格式，然后使用 XSL 将数据转化为 HTML 格式。可以在 <http://ajaxphp.packtpub.com> 附录 C 中找到 XSL 的介绍，[http://en.wikipedia.org/wiki/Extensible\\_Stylesheet\\_Language](http://en.wikipedia.org/wiki/Extensible_Stylesheet_Language) 对 XSL 也有很好的描述。

**注意**

可以在客户端和服务端使用 XSL 转换，本章的转换在客户端实现。这不需要服务器端有任何特殊的特性，但是对客户端存在一些限制。在第 9 章会看到如何使用 PHP 把这种转换放在服务器端，这种情况下需要启动 PHP 的这个特性，这样所有的客户端都能很好工作，因为客户端直接接收的是可显示的 HTML 编码。

在这一章中，您将使用：


- 1) XSL 把从服务器端接收到的 XML 数据转换生成 HTML 数据表格。
- 2) Ajax 实现可编辑的数据表格，用户可以在产品页和编辑产品详细情况页之间切换而不需要重载页面。



## 8.1 使用客户端 XSLT 实现 Ajax 数据表格

在这个题材的学习中，将建立一个支持 Ajax 的可编辑的数据表格。表格中的产品通过 <http://www.the-joke-shop.com/> 很友好地呈现在我们面前。

图 8.1 所示显示了产品的第二页，如图 8.2 所示显示了单击编辑链接后表格的样子，并且其中一个产品已经进入了编辑模式。



AJAX Grid

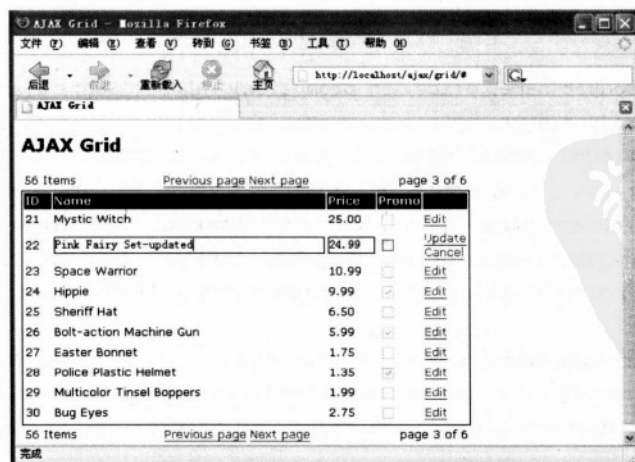
56 Items Previous page Next page page 3 of 6

ID	Name	Price	Promo	Edit
21	Mystic Witch	25.00	<input type="checkbox"/>	Edit
22	Pink Fairy Set	24.99	<input type="checkbox"/>	Edit
23	Space Warrior	10.99	<input checked="" type="checkbox"/>	Edit
24	Hippie	9.99	<input checked="" type="checkbox"/>	Edit
25	Sheriff Hat	6.50	<input type="checkbox"/>	Edit
26	Bolt-action Machine Gun	5.99	<input checked="" type="checkbox"/>	Edit
27	Easter Bonnet	1.75	<input type="checkbox"/>	Edit
28	Police Plastic Helmet	1.35	<input checked="" type="checkbox"/>	Edit
29	Multicolor Tinsel Boppers	1.99	<input type="checkbox"/>	Edit
30	Bug Eyes	2.75	<input type="checkbox"/>	Edit

56 Items Previous page Next page page 3 of 6

完成

图 8.1 运行时的 Ajax 表格



AJAX Grid

56 Items Previous page Next page page 3 of 6

ID	Name	Price	Promo	Edit
21	Mystic Witch	25.00	<input type="checkbox"/>	Edit
22	Pink Fairy Set-updated	24.99	<input type="checkbox"/>	Update Cancel
23	Space Warrior	10.99	<input checked="" type="checkbox"/>	Edit
24	Hippie	9.99	<input checked="" type="checkbox"/>	Edit
25	Sheriff Hat	6.50	<input type="checkbox"/>	Edit
26	Bolt-action Machine Gun	5.99	<input checked="" type="checkbox"/>	Edit
27	Easter Bonnet	1.75	<input type="checkbox"/>	Edit
28	Police Plastic Helmet	1.35	<input checked="" type="checkbox"/>	Edit
29	Multicolor Tinsel Boppers	1.99	<input type="checkbox"/>	Edit
30	Bug Eyes	2.75	<input type="checkbox"/>	Edit

56 Items Previous page Next page page 3 of 6

完成

图 8.2 编辑模式下的 Ajax Grid

因为将动态生成很多输出数据，所以这是很好的学习 XSLT 的机会。首先书写代码，然后再对这些代码进行解释。程序由下面文件组成：

- grid.php
- grid.class.php
- error\_handler.php
- config.php
- grid.css
- index.html
- grid.xsl
- grid.js

## 》》》 实现步骤——Ajax 表格

(1) 首先为这次练习准备数据库。至少需要有产品信息表。可以执行下载代码中的 SQL 脚本 product.sql 或者手动输入（下面的代码片段仅创建了前十条产品记录；使用下载代码可以实现全部产品列表）：

```
CREATE TABLE product
(
    product_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(50) NOT NULL DEFAULT '',
    price DECIMAL(10,2) NOT NULL DEFAULT '0.00',
    on_promotion TINYINT NOT NULL DEFAULT '0',
    PRIMARY KEY (product_id)
);
INSERT INTO product(name, price, on_promotion) VALUES('Santa Costume', 14.99, 0);
INSERT INTO product(name, price, on_promotion) VALUES('Medieval Lady', 49.99, 1);
INSERT INTO product(name, price, on_promotion) VALUES('Caveman', 12.99, 0);
INSERT INTO product(name, price, on_promotion) VALUES('Costume Ghoul', 18.99, 0);
INSERT INTO product(name, price, on_promotion) VALUES('Ninja', 15.99, 0);
INSERT INTO product(name, price, on_promotion) VALUES('Monk', 13.99, 0);
INSERT INTO product(name, price, on_promotion) VALUES('Elvis Black Costume',
35.99, 0);
INSERT INTO product(name, price, on_promotion) VALUES('Robin Hood', 18.99, 0);
INSERT INTO product(name, price, on_promotion) VALUES('Pierot Clown', 22.99, 1);
INSERT INTO product(name, price, on_promotion) VALUES('Austin Powers', 49.99, 0);
```

(2) 在 ajax 目录下创建 grid 子目录。

(3) 从服务端开始书写代码，在 `grid` 目录中创建文件 `grid.php`，它用来响应客户的异步请求。

```
<?php
//加载错误处理脚本和 Grid 类
require_once('error_handler.php');
require_once('grid.class.php');
//'action' 参数应该是 FEED_GRID_PAGE 或 UPDATE_ROW
if (!isset($_GET['action']))
{
    echo 'Server error: client command missing.';
    exit;
}
else
{
    //把要执行的动作放到$action 变量中。
    $action = $_GET['action'];
}
//创建 Grid 实例
$grid = new Grid($action);
//有效的 action 值是 FEED_GRID_PAGE 和 UPDATE_ROW
if ($action == 'FEED_GRID_PAGE')
{
    //获取页数码
    $page = $_GET['page'];
    //读取这一页产品数据信息
    $grid->readPage($page);
}
else if ($action == 'UPDATE_ROW')
{
    //获取参数
    $id = $_GET['id'];
    $on_promotion = $_GET['on_promotion'];
    $price = $_GET['price'];

    $name = $_GET['name'];
    //更新记录
    $grid->updateRecord($id, $on_promotion, $price, $name);
}
else
```

```
    echo 'Server error: client command unrecognized.';
    //清除输出
    if(ob_get_length()) ob_clean();
    //发送报头阻止浏览器缓存
    header('Expires: Fri, 25 Dec 1980 00:00:00 GMT'); //过去的时间
    header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . 'GMT');
    header('Cache-Control: no-cache, must-revalidate');
    header('Pragma: no-cache');
    header('Content-Type: text/xml');
    //生成 XML 格式的输出
    header('Content-type: text/xml');
    echo '<?xml version="1.0" encoding="UTF-8"?>';
    echo '<data>';
    echo '<action>' . $action . '</action>';
    echo $grid->getParamsXML();
    echo $grid->getGridXML();
    echo '</data>';
    ?>
```

(4) 创建文件 grid.class.php, 加入下面代码:

```
<?php
//加载配置文件
require_once('config.php');
//启动 session
session_start();
//包含了处理产品列表的功能
class Grid
{
    //表格页数
    public $mTotalPages;
    //表格条目计数
    public $mItemsCount;
    //将被返回的页面索引
    public $mReturnedPage;
    //数据库句柄
    private $mMysqli;
    //数据库句柄
    private $grid;
    //类构造函数
    function __construct()
```

```

{
    //创建 MySQL 连接
    $this->mMysqli = new mysqli(DB_HOST, DB_USER, DB_PASSWORD,
                               DB_DATABASE);
    //调用 countAllRecords 获取表格记录数
    $this->mItemsCount = $this->countAllRecords();
}
//类析构函数, 关闭数据连接
function __destruct()
{
    $this->mMysqli->close();
}
//读取产品页并将它保存到$this->grid 中
public function readPage($page)
{
    //创建一个 SQL 查询, 返回一个产品页
    $queryString = $this->createSubpageQuery('SELECT * FROM product',
                                             $page);
    //执行查询
    if ($result = $this->mMysqli->query($queryString))
    {
        //返回相关的数组
        while ($row = $result->fetch_assoc())
        {
            //建立包含产品信息的 XML 结构
            $this->grid .= '<row>';
            foreach($row as $name=>$val)
                $this->grid .= '<' . $name . '>' .
                               htmlspecialchars($val) .
                               '</' . $name . '>';
            $this->grid .= '</row>';
        }
        //关闭流结果
        $result->close();
    }
}
//更新产品
public function updateRecord($id, $on_promotion, $price, $name)
{
    //转义输入数据, 以便在 SQL 语句中安全使用

```

```
$id = $this->mMysqli->real_escape_string($id);
$on_promotion = $this->mMysqli->real_escape_string($on_promotion);
$price = $this->mMysqli->real_escape_string($price);
$name = $this->mMysqli->real_escape_string($name);
//建立 SQL 语句, 更新产品信息
$queryString = 'UPDATE product SET name="' . $name . '", ' .
               'price=' . $price . ', ' .
               'on_promotion=' . $on_promotion .
               ' WHERE product_id=' . $id;

//执行 SQL 语句
$this->mMysqli->query($queryString);
}
//返回当前请求的数据
public function getParamsXML()
{
    //计算当前页号
    $previous_page =
        ($this->mReturnedPage == 1) ? '' : $this->mReturnedPage-1;
    //计算下一页页号
    $next_page = ($this->mTotalPages == $this->mReturnedPage) ?
        '' : $this->mReturnedPage + 1;
    //返回参数
    return '<params>' .
           '<returned_page>' . $this->mReturnedPage . '</returned_page>' .
           '<total_pages>' . $this->mTotalPages . '</total_pages>' .
           '<items_count>' . $this->mItemsCount . '</items_count>' .
           '<previous_page>' . $previous_page . '</previous_page>' .
           '<next_page>' . $next_page . '</next_page>' .
           '</params>';
}
//以 XML 格式返回数据表格
public function getGridXML()
{
    return '<grid>' . $this->grid . '</grid>';
}
//返回表格的数据记录总数
private function countAllRecords()
{
    /* 如果记录数没有保存在 Session 中, 则从数据库中读取 */
    if (!isset($_SESSION['record_count']))
```

```

{
    //返回记录总数
    $count_query = 'SELECT COUNT(*) FROM product';
    //执行查询语句并返回结果
    if ($result = $this->mMysqli->query($count_query))
    {
        //返回第一行
        $row = $result->fetch_row();
        /* 返回第一行第一列值, 并保存在 Session 中*/
        $_SESSION['record_count'] = $row[0];
        //关闭数据库句柄
        $result->close();
    }
}
//从 session 中读取并返回记录总数
return $_SESSION['record_count'];
}
//接受一个 SELECT 查询, 返回所有的产品信息, 修改时只返回一个产品页
private function createSubpageQuery($queryString, $pageNo)
{
    //如果产品很少则就不需要分页
    if ($this->mItemsCount <= ROWS_PER_VIEW)
    {
        $pageNo = 1;
        $this->mTotalPages = 1;
    }
    //否则计算有多少页, 并写一个 ELECT 查询
    else
    {
        $this->mTotalPages = ceil($this->mItemsCount / ROWS_PER_VIEW);
        $start_page = ($pageNo - 1) * ROWS_PER_VIEW;
        $queryString .= ' LIMIT ' . $start_page . ', ' . ROWS_PER_VIEW;
    }
    //保存返回的数值
    $this->mReturnedPage = $pageNo;
    //返回新的查询串
    return $queryString;
}
//Grid 类定义结束
}

```

```
?>
```

(5) 加入配置文件 `config.php`:

```
<?php
//定义数据连接数据
define('DB_HOST', 'localhost');
define('DB_USER', 'ajaxuser');
define('DB_PASSWORD', 'practical');
define('DB_DATABASE', 'ajax');
//定义数据表格中的可见行数
define('ROWS_PER_VIEW', 10);
?>
```

(6) 创建错误处理脚本 `error_handler`, 加入下面代码:

```
<?php
//设置用户错误处理句柄
set_error_handler('error_handler', E_ALL);
//错误句柄处理函数
function error_handler($errNo, $errStr, $errFile, $errLine)
{
    //清除所有已经生成的输出
    ob_clean();
    //输出错误信息
    $error_message = 'ERRNO: ' . $errNo . chr(10) .
                    'TEXT: ' . $errStr . chr(10) .
                    'LOCATION: ' . $errFile .
                    ', line ' . $errLine;
    echo $error_message;
    //避免处理其他的 PHP 脚本
    exit;
}
?>
```

(7) 现在书写客户端, 从 `index.html` 开始:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Ajax Grid</title>
<script type="text/javascript" src="grid.js"></script>
```



```

    <link href="grid.css" type="text/css" rel="stylesheet"/>
</head>
    <body onload="init();">
    <div id="gridDiv" />
</body>
</html>

```

(8) 现在创建 XSLT 文件 grid.xml, JavaScript 使用它来产生输出。

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <h2>Ajax Grid</h2>
    <xsl:call-template name="menu"/>
    <form id="grid_form_id">
      <table class="list">
        <tr>
          <th class="th1">ID</th>
          <th class="th2">Name</th>
          <th class="th3">Price</th>
          <th class="th4">Promo</th>
          <th class="th5"></th>
        </tr>
        <xsl:for-each select="data/grid/row">
          <xsl:element name="tr">
            <xsl:attribute name="id">
              <xsl:value-of select="product_id" />
            </xsl:attribute>
            <td><xsl:value-of select="product_id" /></td>
            <td><xsl:value-of select="name" /> </td>
            <td><xsl:value-of select="price" /></td>
            <td>
              <xsl:choose>
                <xsl:when test="on_promotion &gt; 0">
                  <input type="checkbox" name="on_promotion"
                    disabled="disabled" checked="checked"/>
                </xsl:when>
                <xsl:otherwise>
                  <input type="checkbox" name="on_promotion"
                    disabled="disabled"/>
                </xsl:otherwise>
              </xsl:choose>
            </td>
          </xsl:element>
        </xsl:for-each>
      </table>
    </form>
  </template>

```

```

        </xsl:choose>
    </td>
    <td>
        <xsl:element name="a">
            <xsl:attribute name = "href">#</xsl:attribute>
            <xsl:attribute name = "onclick">
                editId(<xsl:value-of select="product_id" />, true)
            </xsl:attribute>
            Edit
        </xsl:element>
    </td>
</xsl:element>
</xsl:for-each>
</table>
</form>
<xsl:call-template name="menu" />
</xsl:template>
<xsl:template name="menu">
    <xsl:for-each select="data/params">
        <table>
            <tr>
                <td class="left">
                    <xsl:value-of select="items_count" /> Items
                </td>
                <td class="right">
                    <xsl:choose>
                        <xsl:when test="previous_page>0">
                            <xsl:element name="a" >
                                <xsl:attribute name="href" >#</xsl:attribute>
                                <xsl:attribute name="onclick">
                                    loadGridPage(<xsl:value-of select="previous_page"/>)
                                </xsl:attribute>
                                Previous page
                            </xsl:element>
                        </xsl:when>
                    </xsl:choose>
                </td>
                <td class="left">
                    <xsl:choose>
                        <xsl:when test="next_page>0">

```

```

        <xsl:element name="a">
            <xsl:attribute name = "href" >#</xsl:attribute>
            <xsl:attribute name = "onclick">
                loadGridPage(<xsl:value-of select="next_page"/>)
            </xsl:attribute>
            Next page
        </xsl:element>
    </xsl:when>
</xsl:choose>
</td>
<td class="right">
    page <xsl:value-of select="returned_page" />
    of <xsl:value-of select="total_pages" />
</td>
</tr>
</table>
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>

```

(9) 创建 grid.js 文件。

```

//保存 XMLHttpRequest 对象的引用
var xmlhttp = createXmlHttpRequestObject();
//XSLT 文件的名称
var xsltFileUrl = "grid.xsl";
//返回的 XML 格式数据
var feedGridUrl = "grid.php";
//数据表格 div 的 ID
var gridDivId = "gridDiv";
//状态 div 的 ID
var statusDivId = "statusDiv";
//存放临时行数据
var tempRow;
//保存正在编辑的行产品 ID
var editableId = null;
//XSLT 文档
var stylesheetDoc;
//一切都从这里开始
function init()
{

```

```
//测试用户浏览器是否支持 XSLT 功能
if(window.XMLHttpRequest && window.XSLTProcessor && window.DOMParser)
{
    //加载数据表格
    loadStylesheet();
    loadGridPage(1);
    return;
}
//测试用户浏览器是否支持旧版本的 XSLT
if (window.ActiveXObject && createMsxml2DOMDocumentObject())
{
    //加载数据表格
    loadStylesheet();
    loadGridPage(1);
    //退出函数
    return;
}
//如果用户浏览器不支持，则警告用户
alert("Your browser doesn't support the necessary functionality.");
}
function createMsxml2DOMDocumentObject()
{
    //保存 MSXML 对象引用
    var msxml2DOM;
    //数据表格可以使用 MSXML
    var msxml2DOMDocumentVersions = new Array("Msxml2.DOMDocument.6.0",
                                                "Msxml2.DOMDocument.5.0",
                                                "Msxml2.DOMDocument.4.0");
    //尝试寻找一个有效的 MSXML 对象
    for (var i=0; i<msxml2DOMDocumentVersions.length && !msxml2DOM; i++)
    {
        try
        {
            //创建一个对象
            msxml2DOM = new ActiveXObject(msxml2DOMDocumentVersions[i]);
        }
        catch (e) {}
    }
    //返回创建的对象或显示一个错误信息
    if (!msxml2DOM)
```

```
    alert("Please upgrade your MSXML version from \n" +
        "http://msdn.microsoft.com/XML/XMLDownloads/default.aspx");
else
    return msxml2DOM;
}
//创建一个XMLHttpRequest 实例
function createXmlHttpRequestObject()
{
    //保存XMLHttpRequest 对象的引用
    var xmlHttp;
    //这将在所有的浏览器上工作,除了IE6和它的早期版本
    try
    {
        //创建一个XMLHttpRequest 对象
        xmlHttp = new XMLHttpRequest();
    }
    catch(e)
    {
        //IE 6或更早的版本
        var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
            "MSXML2.XMLHTTP.5.0",
            "MSXML2.XMLHTTP.4.0",
            "MSXML2.XMLHTTP.3.0",
            "MSXML2.XMLHTTP",
            "Microsoft.XMLHTTP");

        //尝试每一个ID,直到有一个能工作
        for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
        {
            try
            {
                //创建一个XMLHttpRequest 对象
                xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
            }
            catch (e) {}
        }
    }
    //返回创建的对象或返回一个错误信息
    if (!xmlHttp)
        alert("Error creating the XMLHttpRequest object.");
    else
```

```
        return xmlhttp;
    }
    //使用同步请求加载来自服务器端的 stylesheet
    function loadStylesheet()
    {
        //加载来自服务器的文件
        xmlhttp.open("GET", xsltFileUrl, false);
        xmlhttp.send(null);
        //加载 XSLT 文档
        if (this.DOMParser) //使用浏览器的内置功能
        {
            var dp = new DOMParser();
            stylesheetDoc = dp.parseFromString(xmlhttp.responseText, "text/xml");
        }
        else if (window.ActiveXObject) //Internet Explorer?
        {
            stylesheetDoc = createMsxml2DOMDocumentObject();
            stylesheetDoc.async = false;
            stylesheetDoc.load(xmlhttp.responseXML);
        }
    }

    //生成异步请求以加载数据表格的一页
    function loadGridPage(pageNo)
    {
        //当加载新的一页时禁止编辑功能
        editableId = false;
        //只用当 XMLHttpRequest 空闲的时候才继续执行
        if (xmlhttp && (xmlhttp.readyState == 4 || xmlhttp.readyState == 0))
        {
            var query = feedGridUrl + "?action=FEED_GRID_PAGE&page=" + pageNo;
            xmlhttp.open("GET", query, true);
            xmlhttp.onreadystatechange = handleGridPageLoad;
            xmlhttp.send(null);
        }
    }
    //处理来自服务器端的一个新的产品数据
    function handleGridPageLoad()
    {
        //当 readyState 为 4 时, 读取服务器端的响应
```

```
if (xmlHttp.readyState == 4)
{
    //只用当 HTTP 的状态为"OK"时才继续执行
    if (xmlHttp.status == 200)
    {
        //读取响应
        response = xmlHttp.responseText;
        //服务器错误?
        if (response.indexOf("ERRNO") >= 0
            || response.indexOf("error") >= 0
            || response.length == 0)
        {
            //显示错误信息
            alert(response.length == 0 ? "Server serror." : response);
            //退出函数
            return;
        }
        //服务器以 XML 格式响应
        xmlResponse = xmlHttp.responseXML;
        //服务器内置功能吗?
        if (window.XMLHttpRequest && window.XSLTProcessor &&
            window.DOMParser)
        {
            //加载 XSLT 文档
            var xsltProcessor = new XSLTProcessor();
            xsltProcessor.importStylesheet(stylesheetDoc);
            //为新的一个产品生成 HTML
            page = xsltProcessor.transformToFragment(xmlResponse, document);
            //显示这一页的产品
            var gridDiv = document.getElementById(gridDivId);
            gridDiv.innerHTML = "";
            gridDiv.appendChild(page);
        }
        //Internet Explorer 的代码
        else if (window.ActiveXObject)
        {
            //加载 XSLT 文档
            var theDocument = createMsxml2DOMDocumentObject();
            theDocument.async = false;
            theDocument.load(xmlResponse);
        }
    }
}
```

```
//显示这一页的产品
var gridDiv = document.getElementById(gridDivId);
gridDiv.innerHTML = theDocument.transformNode(stylesheetDoc);
}
}
else
{
    alert("Error reading server response.")
}
}
}
//如果 editMode 是 true, 则进入编辑模式, 否则取消编辑模式
function editId(id, editMode)
{
    //获取表格的<tr>元素
    var productRow = document.getElementById(id).cells;
    //我们是否开启编辑模式?
    if(editMode)
    {
        //每次只能有一行处于编辑模式
        if(editableId) editId(editableId, false);
        //保存当前数据, 用户也可以取消改变
        save(id);
        //创建文本编辑框
        productRow[1].innerHTML =
            '<input class="editName" type="text" name="name" ' +
            'value="' + productRow[1].innerHTML + '">';
        productRow[2].innerHTML =
            '<input class="editPrice" type="text" name="price" ' +
            'value="' + productRow[2].innerHTML + '">';
        productRow[3].getElementsByTagName("input")[0].disabled = false;
        productRow[4].innerHTML = '<a href="#" ' +
            'onclick="updateRow(document.forms.grid_form_id,' + id +
            ')">Update</a><br/><a href="#" onclick="editId(' + id +
            ',false)">Cancel</a>';
        //保存编辑的产品 ID
        editableId = id;
    }
    //如果关闭了编辑模式
    else
```



```

{
    productRow[1].innerHTML = document.forms.grid_form_id.name.value;
    productRow[2].innerHTML = document.forms.grid_form_id.price.value;
    productRow[3].getElementsByTagName("input")[0].disabled = true;
    productRow[4].innerHTML = '<a href="#" onclick="editId(' + id +
        ',true)">Edit</a>';

    //没有可编辑的产品
    editableId = null;
}
}
//保存编辑之前的值
function save(id)
{
    //获取一行产品
    var tr = document.getElementById(id).cells;
    //保存数据
    tempRow = new Array(tr.length);
    for(var i=0; i<tr.length; i++)
        tempRow[i] = tr[i].innerHTML;
}
//取消编辑, 恢复初始值
function undo(id)
{
    //获取一行产品
    var tr = document.getElementById(id).cells;
    //赋值以前的值
    for(var i=0; i<tempRow.length; i++)
        tr[i].innerHTML = tempRow[i];
    //不可编辑
    editableId = null;
}
//如果连接顺利的话更新表格中的一行
function updateRow(grid, productId)
{
    //只有当 XMLHttpRequest 对象空闲时, 继续执行
    if (xmlHttp && (xmlHttp.readyState == 4 || xmlHttp.readyState == 0))
    {
        var query = feedGridUrl + "?action=UPDATE_ROW&id=" + productId +
            "&" + createUpdateUrl(grid);
        xmlHttp.open("GET", query, true);
    }
}

```

```
xmlHttp.onreadystatechange = handleUpdatingRow;
xmlHttp.send(null);
}
}
//更新产品时处理服务器端的响应
function handleUpdatingRow()
{
    //当读取状态码为 4 时, 读取服务器响应
    if(xmlHttp.readyState == 4)
    {
        //只有 HTTP 状态为 OK 时才继续
        if(xmlHttp.status == 200)
        {
            //读取响应
            response = xmlHttp.responseText;
            //服务器错误?
            if (response.indexOf("ERRNO") >= 0
                || response.indexOf("error") >= 0
                || response.length == 0)
                alert(response.length == 0 ? "Server serror." : response);
            //如果一切正常, 取消编辑模式
            else
                editId(editableId, false);
        }
        else
        {
            //如果发生错误回滚所做的改变
            undo(editableId);
            alert("Error on server side.");
        }
    }
}
//创建一个查询串参数来更新一行
function createUpdateUrl(grid)
{
    //初始查询串
    var str = "";
    //使用可编辑的表格元素写查询语句
    for(var i=0; i<grid.elements.length; i++)
        switch(grid.elements[i].type)
```

```

{
  case "text":
  case "textarea":
    str += grid.elements[i].name + "=" +
          (grid.elements[i].checked ? 1 : 0) + "&";
    break;
  case "checkbox":
    if (!grid.elements[i].disabled)
      str += grid.elements[i].name + "=" +
            (grid.elements[i].checked ? 1 : 0) + "&";
    break;
}
//返回查询串
return str;
}

```

(10) 最后创建 grid.css 文件:

```

body
{
  font-family: Verdana, Arial;
  font-size: 10pt
}
table
{
  width: 500px;
}
td.right
{
  color: darkblue;
  text-align: right;
  width: 125px
}
td.left
{
  color: darkblue;
  text-align: left;
  width: 125px
}
table.list
{

```

```
border: black 1px solid;
}
th
{
text-align: left;
background-color: navy;
color: white
}
th.th1
{
width: 30px
}
th.th2
{
width: 300px
}
input.editName
{
border: black 1px solid;
width: 300px
}
input.editPrice
{
border: black 1px solid;
width: 50px
}
```

(11) 通过浏览器访问 <http://localhost/ajax/grid/>，看看所有的功能是否像预想的一样工作（可以参考图 8.1 和图 8.2）。

## 程序解释

下面从服务器端开始分析代码。服务端的核心在数据库部分。在例子中，有一个数据表 `product`，含有如下字段：

- `Product_id` 是表的主键，存放产品数字 ID。
- `Name` 是产品名称。
- `Price` 是产品价格。
- `On_promot` 是一位字段（只对应 0 和 1 两个值，尽管 Mysql 允许更多的值，由版本决定），它定义了产品是否推销。我们在表格中使用这个字段是为了演示如何使用 `checkbox`

来显示位值。

和通常一样，服务器端有 PHP 脚本，在这个例子中是 `grid.php`，它是客户异步请求的主访问入口点。

`Grid.php` 接收参数名为 `action` 的查询字符串，它告诉 `grid.php` 执行什么动作。Action 的值有下面两种。

- `FEED_GRID_PAGE`: 这个值用来获取一页的产品信息。除了这个参数，服务器端还需要 `page` 参数，`page` 指定了返回哪一页的产品信息。
- `UPDATE_NOW`: 这个值用来更新被用户编辑的那行产品的细节，对于这个操作，服务器也需要接收产品新的信息通过 4 个参数 `id`、`name`、`price` 和 `on_promotion` 传递。

要查看服务端产生的数据，简单地访问：`http://localhost/ajax/grid/grid.php?action=FEED_GRID_PAGE&page=1` 即可。使用默认的数据库信息，产生的数据输出如图 8.3 所示。

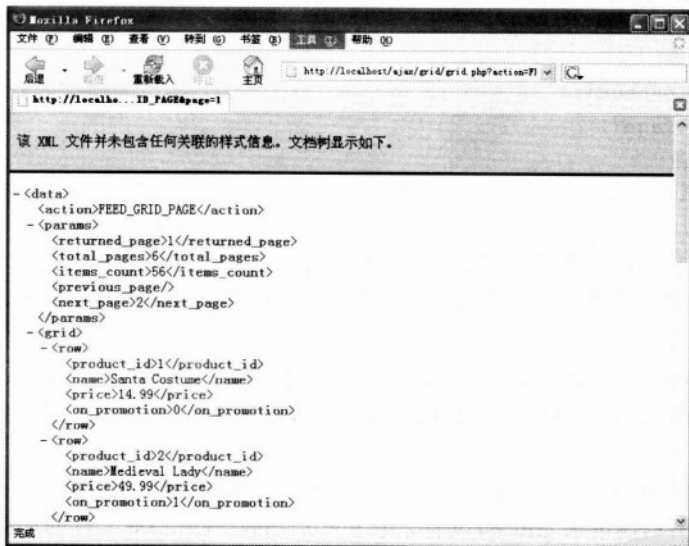


图 8.3 服务器端返回第一页产品信息

在客户端，这些数据使用 XSL 解析和转换成 HTML 表格。这段代码已经在 Mozilla 和 Internet Explorer 上进行了测试，在写这本书的时候这些浏览器是支持这些功能的。Opera 在 9 版的时候将支持 XSL。

XSL 转换规则定义在 `grid.xml` 中。XSL 的初学者请访问 `http://ajaxphp.packtpub.com` 的附录 C，想深入了解可以看一些书和在线资源。XSL 是一个很庞大的结构，如果想很好地掌握它，就要准备学很多东西。

`init()` 函数是客户端脚本 `grid.js` 中的第一个函数。这个函数检查用户的浏览器是否支持 XSL 转换这个特性。

```

//一切从下面开始
function init()
{
    //测试用户的浏览器是否支持内置的 XSLT 功能
    if(window.XMLHttpRequest && window.XSLTProcessor && window.DOMParser)
    {
        //下载表格
        loadStylesheet();
        loadGridPage(1);
        return;
    }
    //如果用户使用的是 IE, 测试是否正确支持 XSLT.
    if (window.ActiveXObject && createMsxml2DOMDocumentObject())
    {
        //下载表格
        loadStylesheet();
        loadGridPage(1);
        //退出函数
        return;
    }
    //如果浏览器功能测试失败, 向用户发出警告。
    alert("Your browser doesn't support the necessary functionality.");
}

```

如果是 Internet Explorer (在这种情况下, 用户也需要一个最新的 MSXML) 或者是本身就支持 XMLHttpRequest、XSLTProcessor 和 DOMParser 的浏览器, 那么这个函数都会继续执行。

理解第二个函数 loadStylesheet()是非常重要的, 当页面加载的时候这个函数只被调用一次, 为的是从服务器端请求 grid.xml 文件, 它在本地加载。Grid.xml 文件的加载使用了异步调用, 然后使用和用户的浏览器特定的技术将它存储起来。存储取决于浏览器是否内置了该功能, 或者在 Internet Explorer 中调用了 ActiveObject。

```

//使用同步请求, 从服务器端下载 stylesheet
function loadStylesheet()
{
    //从服务器端下在文件
    xmlhttp.open("GET", xsltFileUrl, false);
    xmlhttp.send(null);
    //尝试下载 XSLT 文档
    if (this.DOMParser) //作为内置该功能的浏览器

```

```

{
    var dp = new DOMParser();
    stylesheetDoc = dp.parseFromString(xmlHttp.responseText, "text/xml");
}
else if (window.ActiveXObject) //是 Internet Explorer 吗?
{
    stylesheetDoc = createMsxml2DOMDocumentObject();
    stylesheetDoc.async = false;
    stylesheetDoc.load(xmlHttp.responseXML);
}
}

```

当页面加载的时候调用 loadGridPage 函数，并且当用户每次单击上一页或下一页数据时该函数都被调用。这个函数用来和服务器异步通信，它指定了需要返回哪一页产品信息。

```

//使用异步请求，下载新的表格页面
function loadGridPage(pageNo)
{
    //下载新页面时，禁止编辑功能
    editableId = false;
    //只有在 XMLHttpRequest 对象不忙时继续
    if (xmlHttp && (xmlHttp.readyState == 4 || xmlHttp.readyState == 0))
    {
        var query = feedGridUrl + "?action=FEED_GRID_PAGE&page=" + pageNo;
        xmlHttp.open("GET", query, true);
        xmlHttp.onreadystatechange = handleGridPageLoad;
        xmlHttp.send(null);
    }
}

```

通过调用 handleGridPageLoad 回调函数来处理服务器的响应，除了错误处理机制，它还负责把从服务器端接收到的 XML 有效地转换成 HTML 代码并在客户端显示。转换的代码是和浏览器相关的，针对 Internet Explorer 和支持内置 XSL 的浏览器执行的功能不同。

```

//XML 格式的服务器响应
xmlResponse = xmlHttp.responseXML;
//作为本地功能支持的浏览器
if (window.XMLHttpRequest && window.XSLTProcessor && window.DOMParser)

    //下载 XSLT 文档
    var xsltProcessor = new XSLTProcessor();

```

```

xsltProcessor.importStylesheet (stylesheetDoc);
//生成新产品页面的 HTML 代码
page = xsltProcessor.transformToFragment(xmlResponse, document);
//显示产品页面
var gridDiv = document.getElementById(gridDivId);
gridDiv.innerHTML = "";
gridDiv.appendChild(page);
}
//Internet Explorer 代码
else if (window.ActiveXObject)
{
//下载 XSLT 文档
var theDocument = createMsxml2DOMDocumentObject();
theDocument.async = false;
theDocument.load(xmlResponse);
//显示产品页面
var gridDiv = document.getElementById(gridDivId);
gridDiv.innerHTML = theDocument.transformNode(stylesheetDoc);
}

```

现在我们看看 `editId` 函数。在表格中单击 `Edit` 或 `Cancel` 按钮时就会调用这个函数，来打开或关闭编辑模式。打开编辑模式时，产品的名字、价格和 `PROMOTION CheckBox` 将被传递到可编辑控件。关闭编辑模式时，这条记录就变成了不可编辑状态。

`Save()` 函数和 `undo()` 函数是用于帮助编辑行记录的。保存函数用来保存产品的初始值，如果用户改变主意不想修该产品值则单击 `Cancel` 连接，`undo` 函数将被调用并返回原来的值。

`updateRow` 函数支持更新一行的内容，单击 `Update` 连接时该函数将被调用。`UpdateRow` 函数对服务器做一个异步调用，同时指定了产品的新值，这些值将通过 `createUpdateUrl` 辅助函数组合到一个查询串中：

```

//如果连接，更新表格中的一行
function updateRow(grid, productId)
{
//只有在 XMLHttpRequest 对象不忙的时候继续
if (xmlHttp && (xmlHttp.readyState == 4 || xmlHttp.readyState == 0))
{
var query = feedGridUrl + "?action=UPDATE_ROW&id=" + productId +
"&" + createUpdateUrl(grid);
xmlHttp.open("GET", query, true);
xmlHttp.onreadystatechange = handleUpdatingRow;
xmlHttp.send(null);
}
}

```



```
}  
}
```

`handleUpdateingRow` 回调函数负责确保能成功修改产品，在这种情况下它会关闭行编辑模式，或者在服务器端有错误发生时它将在客户端显示一条错误信息。

```
//当 HTTP 状态为 "OK" 时继续  
if(xmlHttp.status == 200)  
{  
    //读取响应  
    response = xmlHttp.responseText;  
    //检查服务器错误  
    if (response.indexOf("ERRNO") >= 0  
        || response.indexOf("error") >= 0  
        || response.length == 0)  
        alert(response.length == 0 ? "Server error." : response);  
    //如果一切顺利，退出 edit 模式  
    else  
        editId(editableId, false);  
}
```

这种错误处理技术和其他练习中的实现是一样的。如果服务器返回一个错误消息，这个消息将发送给用户。如果 PHP 被配置为不输出错误，服务器将不会响应错误，这种情况下只显示一个一般的错误信息。

## 8.2 小结

在这一章，读者已经熟悉了使用 Ajax 技术来创建数据表格，并且接触到了 XSL，它允许用户实现一个功能强大的架构，这样应用程序的服务器端就不需要处理显示方面的内容了。

使用 XSL 处理要显示给访问者的格式化数据，是一个很专业的处理这类事务的方法。如果很关心网页的发展，那么推荐您学好 XSL。注意，这会消耗大量的时间和精力，但最终付出是值得的。

# 第 9 章

## Ajax RSS 阅读器

在过去的几年里，Web 变得越来越活跃了。现在，我们看到新出现了很多信息资源，比如每天出现新的站点（例如 <http://www.digg.com> 和 <http://www.newsvine.com>）和迅速增多的网络生活——网上博客（好像现在每个人都有一个网上博客）。

信息剧增的自然反应就是很多系统都对它们进行分组、过滤和聚合。具体实现是通过 Web 企业联合（Web syndication）来实现的，它就是一种 Web 联合表格，其他站点或应用程序可以访问其上网站的各个部分（例如，新闻、博客、文章等）。

为了让其他部分都能使用，那些共享的数据必须使用一种能在不同版式下展示的通用格式，而不是像原始数据那样。当使用这种通用格式时，RSS2.0 和 Atom 就是最好的选择。

要了解更多关于 RSS 和 Atom，请访问 Wikipedia——RSS 的页面链接 [http://en.wikipedia.org/wiki/RSS\\_\(protocol\)](http://en.wikipedia.org/wiki/RSS_(protocol))。

这一章将分析 RSS 文件格式，接下来看看 Google Reader，然后使用 Ajax 和 PHP 做一个 RSS 聚合 Web 页面。

### 9.1 使用 RSS

RSS 是广泛使用的基于 XML 的标准，用来在因特网应用程序之间交换信息。XML 的一个巨大优势就是它是纯文本的，这样它就被很容易被任何的应用程序读取。RSS 信息源（RSS feeds）可以看作是纯文本文件，但那样使用没有意义，它们将被特殊的应用程序读取，并根据其中的数据来生成网页内容。

然而，RSS 并不是使用 XML 表示信息源的惟一标准。在本节的学习中选择使用这种格式是因为它使用得很广泛。为了更好的理解 RSS，我们需要了解一些更深层的东西，那就是 RSS 文档结构。

#### 9.1.1 RSS 文档结构

RSS 的第一版在 1999 年创建，那就是我们熟悉的 0.9 版，到今天它已经发展到 2.0.1 版，

这一版本已经被开发社区冻结，因为在将来的开发中想使用另外一个名称。

下面是一个典型的 RSS 信息源：

```
<rss version="2.0">
<channel>
  <title>CNN.com</title>
  <link>http://www.example.org</link>
  <description>A short description of this feed</description>
  <language>en</language>
  <pubDate>Mon, 17 Oct 2005 07:56:23 EDT</pubDate>
  <item>
    <title>Catchy Title</title>
    <link>http://www.example.org/2005/11/catchy-title.html</link>
    <description>
      The description can hold any content you wish, including XHTML.
    </description>
    <pubDate>Mon, 17 Oct 2005 07:55:28 EDT</pubDate>
  </item>
  <item>
    <title>Another Catchy Title</title>
    <link>http://www.example.org/2005/11/another-catchy-title.html</link>
    <description>
      The description can hold any content you wish, including XHTML.
    </description>
    <pubDate>Mon, 17 Oct 2005 07:55:28 EDT</pubDate>
  </item>
</chanel>
</rss>
```

这个信息源可包含任意多的<item>条目，每一条都包含着不同的新闻或博客入口，或其他想保存的内容。

这完全是一个纯文本文件，就像前面描述的那样，需要使用一个特别的软件来解析 XML 并生成需要的信息。RSS 解析器也叫做聚合器，因为它可以从多个 RSS 资源中抽取和聚合信息。

这样的应用有 Google Reader，由 Google 公司在 2005 年秋季推出的在线服务。站点 <http://www.bloglines.com> 上有一个成熟的基于 Web 的 RSS 阅读器服务，它也是一个类似的应用。

## 9.1.2 Google Reader

Google Reader 提供了一个简单直观的支持 Ajax 的界面，用户通过这个界面来跟踪他们

订阅的 RSS。虽然提供这个服务还没有多长时间，但是它已经吸引了大量用户。如图 9.1 所示显示了一个运行着的 Google Reader，正在从 Pack Publishing 的 RSS 信息源中读取新闻条目。

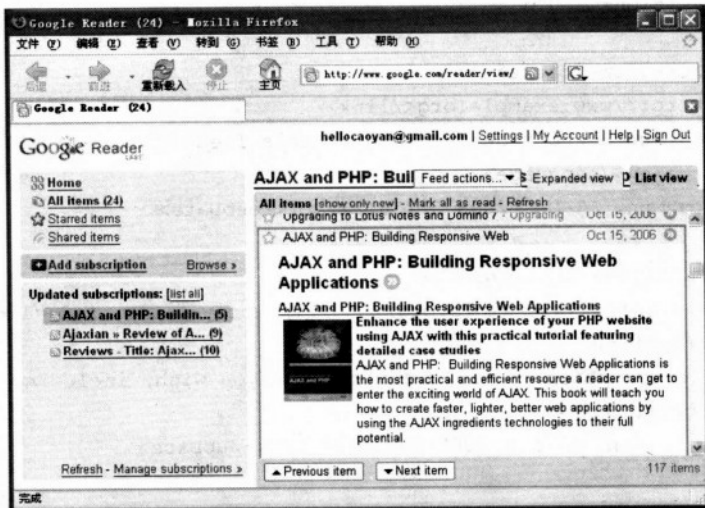


图 9.1 在 Google Reader 上管理 RSS 订阅 (Feeds)

## 9.2 实现基于 Ajax 的 RSS 阅读器

为了确保程序正确运行，需要在安装 PHP 时开启对 XSL 的支持。附录 A 中介绍如何安装包括对 XSL 的支持。

在下面的练习中，将创建一个支持 Ajax 的 RSS 阅读器应用程序。这个程序的主要特性如下：

(1) 我们将保持程序的简洁性。在服务器端的 PHP 文件中信息源列表里编码比较困难的部分。

(2) 使用 XSLT 将 RSS 数据转换成访问者可查看的内容。本章中，在服务器端使用 PHP 代码执行 XSL 的转换。

(3) 使用 SimpleXML 库读取来自服务器端的 XML 响应。PHP 5 中引入了 SimpleXML，可以在 <http://php.net/simplexml> 查看它的官方文档。SimpleXML 是一个非常优秀的库，使用它读取 XML 源文件要比使用 DOM 容易。

(4) 程序如图 9.2 所示。

信息源是动态加载的，并且在左栏显示连接。单击一个信息源连接，它将触发一个 HTTP 请求，服务端脚本将获取想要得到的 RSS 信息源。

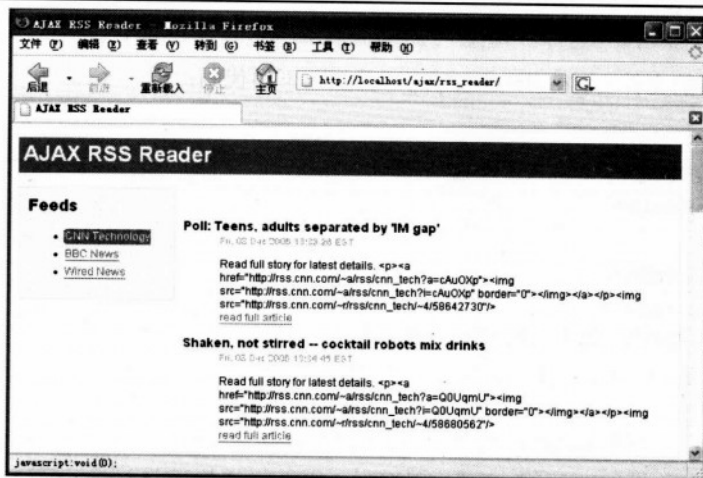


图 9.2 支持 Ajax 的 RSS 阅读器的开始页面

然后服务器将使用 XSL 来格式化这个信息源，返回 XML 字符串，并以人们易懂的格式显示结果。

## 》》》 实现步骤——建立 RSS 阅读器应用程序

- (1) 在 ajax 目录下创建新的目录 rss\_reader。
- (2) 从服务器端开始。创建 rss\_reader.php 并加入下面代码：

```
<?php
//加载帮助脚本
require_once ('error_handler.php');
require_once ('rss_reader.class.php');
//创建一个新的 RSS Reader 实例
$reader = new CRssReader(urldecode($_POST['feed']));
//清除输出
if(ob_get_length()) ob_clean();
//发送头以防止浏览器缓存
header('Expires: Fri, 25 Dec 1980 00:00:00 GMT'); //过去的时间
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . 'GMT');
header('Cache-Control: no-cache, must-revalidate');
header('Pragma: no-cache');
header('Content-Type: text/xml');
//返回消息给客户端
echo $reader->getFormattedXML();
```

```
?>
```

(3) 创建新的文件 `rss_reader.class.php`，并加入下面代码：

```
<?php
//这个类获取 RSS feed 并执行 XSLT 转换
class CRssReader
{
    private $mXml;
    private $mXsl;
    //构造函数，基于指定的信息源，创建 XML 对象
    function __construct($szFeed)
    {
        //从 SimpleXML 对象中获取 RSS feed object
        $this->mXml = simplexml_load_file(urldecode($szFeed));
        //从 SimpleXML 对象中获取 XSL 文件内容
        $this->mXsl = simplexml_load_file('rss_reader.xsl');
    }
    //基于取回的信息源创建一个 XML 格式的文档
    public function getFormattedXML()
    {
        //创建一个 XSLTProcessor 对象
        $proc = new XSLTProcessor;
        //附加上 XSL
        $proc->importStyleSheet($this->mXsl);
        //执行转换并返回 XML 串
        return $proc->transformToXML($this->mXml);
    }
}
?>
```

(4) 创建新的文件 `rss_reader.xsl`，并加入下面代码：

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <dl>
            <xsl:for-each select="rss/channel/item">
                <dt><h3><xsl:value-of select="title" /></h3></dt>
                <dd>
                    <span class="date"><xsl:value-of select="pubDate" /></span>
```

```

    <p>
      <xsl:value-of select="description" />
    <br />
    <xsl:element name="a">
      <xsl:attribute name="href">
        <xsl:value-of select="link" />
      </xsl:attribute>
      read full article
    </xsl:element>
  </p>
</dd>
</xsl:for-each>
</dl>
</xsl:template>
</xsl:stylesheet>

```

(5) 现在创建一个标准错误处理文件 `error_handler.php`，也可以从前面章节复制。它的代码如下：

```

<?php
//设置用户错误处理句柄
set_error_handler('error_handler', E_ALL);

//错误处理函数
function error_handler($errNo, $errStr, $errFile, $errLine)
{
    //清除产生的输出
    if(ob_get_length()) ob_clean();
    //输出错误信息
    $error_message = 'ERRNO: ' . $errNo . chr(10) .
                    'TEXT: ' . $errStr . chr(10) .
                    'LOCATION: ' . $errFile .
                    ', line ' . $errLine;
    echo $error_message;
    //退出程序，防止处理更多的脚本
    exit;
}
?>

```

(6) 在 `rss_reader` 文件夹中创建 `config.php`，输入以下内容：

```

<?php

```

```
//建立 feed
$feeds = array ('0' => array('title' => 'CNN Technology',
                             'feed' =>
                             'http://rss.cnn.com/rss/cnn_tech.rss'),
                '1' => array('title' => 'BBC News',
                             'feed' =>
                             'http://news.bbc.co.uk/rss/newsonline_uk_edition/front_page/rss.xml'),
                '2' => array('title' => 'Wired News',
                             'feed' =>
                             'http://wirednews.com/news/feeds/rss2/0,2610,3,00.xml'));
?>
```

(7) 创建 index.php, 并加入下面代码:

```
<?php
//加载 feed 列表
require_once ('config.php');
?>
<!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 RSS Reader</title>
    <link rel="stylesheet" type="text/css" href="rss_reader.css"/>
    <script src="rss_reader.js" type="text/javascript"></script>
  </head>
  <body>
    <h1>Ajax RSS Reader</h1>
    <div id="feeds">
      <h2>Feeds</h2>
      <ul id="feedList">
        <?php
          //显示 feeds
          for ($i = 0; $i < count($feeds); $i++)
          {
            echo '<li id="feed-' . $i . '"><a href="javascript:void(0);" ' .
            'onclick="getFeed(document.getElementById(\'feed-' . $i .
            '\'), \' . urlencode($feeds[$i][\'feed\']) . '\');">';
            echo $feeds[$i][\'title\'] . '</a></li>';
          }
        </?php
      </ul>
    </div>
  </body>
</html>
```



```

    ?>
  </ul>
</div>
<div id="content">
  <div id="loading" style="display:none">Loading feed...</div>
  <div id="feedContainer" style="display:none"></div>
  <div id="home">
    <h2>About the Ajax RSS Reader</h2>
    <p>
      The Ajax RSS reader is only a simple application that provides
      basic functionality for retrieving RSS feeds.
    </p>
    <p>
      This application is presented as a case study in
      <a href="https://www.packtpub.com/ajax_php/book"> Building
      Responsive Web Applications with Ajax and PHP</a>
      (Packt Publishing, 2006).
    </p>
  </div>
</div>
</body>
</html>

```

(8) 创建文件 `rss_reader.js`，并输入下面代码：

```

//加载一个XMLHttpRequest实例
var xmlhttp = createXmlHttpRequestObject();
//当设置为真时，显示详细错误信息
var showErrors = true;

//创建一个XMLHttpRequest实例
function createXmlHttpRequestObject()
{
  //保存XMLHttpRequest对象的引用
  var xmlhttp;
  //除了IE6和它的早期版本，所有的浏览器都能很好工作
  try
  {
    //创建一个XMLHttpRequest对象
    xmlhttp = new XMLHttpRequest();
  }
}

```

```
catch (e)
{
    //如果是 IE 6 或它的早期版本
    var XmlHttpRequestVersions = new Array("MSXML2.XMLHTTP.6.0",
                                             "MSXML2.XMLHTTP.5.0",
                                             "MSXML2.XMLHTTP.4.0",
                                             "MSXML2.XMLHTTP.3.0",
                                             "MSXML2.XMLHTTP",
                                             "Microsoft.XMLHTTP");

    //尝试每一个 ID, 直到有一个能正常工作
    for (var i=0; i<XmlHttpRequestVersions.length && !xmlHttp; i++)
    {
        try
        {
            //尝试创建一个 XMLHttpRequest 对象
            xmlHttp = new ActiveXObject(XmlHttpRequestVersions[i]);
        }
        catch (e) {} //忽略潜在的错误
    }
}

//返回创建的对象或显示一个错误信息
if (!xmlHttp)
    alert("Error creating the XMLHttpRequest object.");
else

    return xmlHttp;
}

//该函数显示错误信息
function displayError($message)
{
    //如果 showErrors 为假, 则忽略错误
    if (showErrors)
    {
        //关闭错误显示
        showErrors = false;
        //显示错误信息
        alert("Error encountered: \n" + $message);
    }
}

//从 feed 中获取标题并显示出来
```

```
function getFeed(feedLink, feed)
{
    //只用当 xmlhttp 有效的时候, 才继续执行
    if (xmlHttp)
    {
        //尝试连接服务器端
        try
        {
            if (xmlHttp.readyState == 4 || xmlhttp.readyState == 0)
            {
                /* 获取信息源数并遍历它们, 改变它们的容器类名 (<li>). */
                var numberOfFeeds =
                    document.getElementById("feedList").childNodes.length;
                for (i = 0; i < numberOfFeeds; i++)
                    document.getElementById("feedList").childNodes[i].className = "";
                //改变被单击的 feed 类名称, 让它变成高亮
                feedLink.className = "active";
                //当加载 feed 的时候显示 "Loading..." 消息
                document.getElementById("loading").style.display = "block";
                //调用服务页面来执行服务器端操作
                params = "feed=" + feed;
                xmlhttp.open("POST", "rss_reader.php", true);
                xmlhttp.setRequestHeader("Content-Type",
                    "application/x-www-form-urlencoded");
                xmlhttp.onreadystatechange = handleHttpGetFeeds;
                xmlhttp.send(params);
            }
            else
            {
                //如果连接忙的话, 1 秒后重试
                setTimeout("getFeed('" + feedLink + "', '" + feed + "')", 1000);
            }
        }
        //如果连接失败的话显示出错信息
        catch (e)
        {
            displayError(e.toString());
        }
    }
}
```

```
//获取 HTTP 响应
function handleHttpGetFeeds()
{
    //如果处理结束的话继续执行
    if (xmlHttp.readyState == 4)
    {
        //只用当 HTTP 状态是“OK”的时候才继续执行
        if (xmlHttp.status == 200)
        {
            try
            {
                displayFeed();
            }
            catch(e)
            {
                //显示错误信息
                displayError(e.toString());
            }
        }
        else
        {
            displayError(xmlHttp.statusText);
        }
    }
}

//处理服务器端响应
function displayFeed()
{
    //以文本格式读取服务端响应并检查错误
    var response = xmlHttp.responseText;
    //服务器是否有故障
    if (response.indexOf("ERRNO") >= 0
        || response.indexOf("error:") >= 0
        || response.length == 0)
        throw(response.length == 0 ? "Void server response." : response);
    //当 feed 返回时就隐藏“Loading...”消息
    document.getElementById("loading").style.display = "none";
    //添加 XSLed XML 内容到 DOM 结构中
    var titlesContainer = document.getElementById("feedContainer");
```

```
titlesContainer.innerHTML = response;
//使 feed 容器变得可见
document.getElementById("feedContainer").style.display = "block";
//清除页面文本
document.getElementById("home").innerHTML = "";
}
```

(9) 创建文件 `rss_reader.css`，并加入下面代码：

```
body
{
    font-family: Arial, Helvetica, sans-serif;
    font-size: 12px;
}
h1
{
    color: #ffffff;
    background-color: #3366CC;
    padding: 5px;
}
h2
{
    margin-top: 0px;
}
h3
{
    margin-bottom: 0px;
}

li
{
    margin-bottom: 5px;
}

div
{
    padding: 10px;
}

a, a:visited
{
```

```
color: #3366CC;
text-decoration: underline;
}

a:hover
{
color: #ffffff;
background-color: #3366CC;
text-decoration: none;
}

.active a
{
color: #ffffff;
background-color: #3366CC;
text-decoration: none;
}

.active a:visited
{
color: #ffffff;
background-color:#3366CC;
text-decoration:none;
}

.active a:hover
{
color:#ffffff;
background-color: #3366CC;
text-decoration: none;
}

#feeds
{
display: inline;
float: left;
width: 150px;
background-color: #f4f4f4;
border:1px solid #e6e6e6;
}
```

```

#content
{
    padding-left:170px;
    border:1px solid #f1f1f1;
}

#loading
{
    float: left;
    display: inline;

    width: 410px;
    background-color: #fffb8;
    color: #FF9900;
    border: 1px solid #ffcc00;
    font-weight: bold;
}

.date
{
    font-size: 10px;
    color: #999999;
}

```

(10) 通过浏览器访问 [http://localhost/ajax/rss\\_reader](http://localhost/ajax/rss_reader), 初始页面如图 9.3 所示。如果单击一个链接, 将会看到如图 9.2 所示的页面。

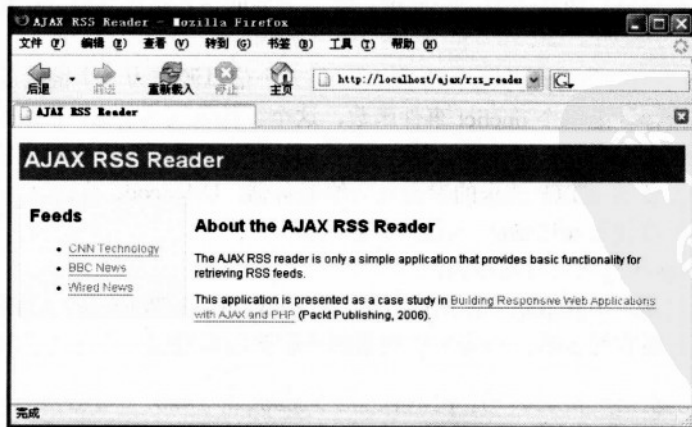


图 9.3 Ajax RSS 阅读器的首页

## 程序说明

程序确实不很专业，但是它的想法是很好的。实现这个结果不需要很多的代码，并且添加新的功能很容易。

这个应用程序的用户界面相当简单，都建立在 `index.php` 上。首先需要包含 `config.php` 文件，它定义了信息源，为的是在页面的左边显示信息源列表。`Feeds` 都定义在联合数组中。主数组的索引从 0 开始，它的每一个值也都是数组，数组的索引是信息源的标题，值是信息源的 URL。`$feeds` 数组定义如下：

```
$feeds = array ('0' => array('title' => 'CNN Technology',
                             'feed' => 'http://rss.cnn.com/rss/cnn_tech.rss'),
                '1' => array('title' => 'BBC News',
                             'feed' =>
                             'http://news.bbc.co.uk/rss/newsonline_uk_edition/front_page/rss.xml'),
                '2' => array('title' => 'Wired News',
                             'feed' =>
                             'http://wirednews.com/news/feeds/rss2/0,2610,3,00.xml'));
```

换成一种更好理解的形式，`$feeds` 数组如下：

ID	信息源标题 (title)	信息源链接(feed)
0	CNN Technology	http://rss.cnn.com/rss/cnn_tech.rss
1	BBC News	http://news.bbc.co.uk/rss/newsonline_uk_edition/front_page/rss.xml
2	Wired News	http://wirednews.com/news/feeds/rss2/0,2610,3,00.xml

我们决定把它存储在数组中是因为简单。如果需要的话，扩展代码并把信息源存入数据库也很容易。

在 `index.php` 中遍历信息源并无序显示它们。每一个信息源作为一个链接放在 `<li>` 元素中。我们给每一个链接都分配一个 `onclick` 事件函数，这个事件将调用 `getFeed` 函数。该函数接收两个参数：`<li>` 的 ID 和 `feeds` 的 URL。我们需要 ID 来高亮显示列表中对应的链接，还需要信息源的 URL，它作为 HTTP 请求的参数发送给服务器。`urlencode` 函数确保 URL 安全地发送给服务器，服务器使用 `urldecode` 函数进行解密。

关于 `index.php` 还有两个方面要讲：

(1) 隐藏初始化，当获取信息源的时候将显示 `<div>` 和 `id="loading"`，用来通知用户信息源正在加载。这是很有用处的，因为当连接很慢或服务器很慢时反应时间会很长。

```
<div id="loading" style="display:none">Loading feed...</div>
```

(2) `<div>` 中的 `id="feedContainer"`，这是信息源加载时实际使用的容器。信息源将被动态



地插入到这个 div 元素。

```
<div id="feedContainer"></div>
```

rss\_reader.js 包含了标准的 XMLHttpRequest 初始化、发送请求和获取响应的代码。getFeed 函数处理 HTTP 请求的发送。首先它遍历所有的信息源链接，并且通过设置它们的 CSS 类为空使它们不再高亮。然后让活跃的链接高亮显示。

```
/*获取 feeds 总数，遍历每个 feed 并改变它们容器的类名<li>）。*/
var numberOfFeeds =
    document.getElementById("feedList").childNodes.length;
for (i = 0; i < numberOfFeeds; i++)
    document.getElementById("feedList").childNodes[i].className = "";
//改变被双击的 feed 的类名，使它变为高亮
feedLink.className = "active";
```

下一步，显示“Loading feed...”消息。

```
//当加载 feed 时显示"Loading..." 消息
document.getElementById("loading").style.display = "block";
```

最后，我们发送一个以信息源标题为参数的 HTTP 请求。

```
//调用服务页面来执行服务器端操作
params = "feed=" + feed;
xmlHttp.open("POST", "rss_reader.php", true);
xmlHttp.setRequestHeader("Content-Type",
    "application/x-www-form-urlencoded");
xmlHttp.onreadystatechange = handleHttpGetFeeds;
xmlHttp.send(params);
```

Rss\_reader.php 脚本创建一个 CRssReader 类实例，并且显示一个 XSL 格式的 XML 文档，这个文档将返回给客户端。下面这些代码会完成这个复杂的工作。

```
$reader = new CRssReader(urldecode($_POST['feed']));
echo $reader->getFormattedXML();
```

CRssReader 定义在 rss\_reader.class.php 文件中。PHP 类处理 XML 的获取和格式化。利用 PHP5 的新扩展 SimpleXML 获取远程的 XML 文件非常简单。可以加载 XSL 模块并将它用于新获取的 XML 文件。

在这个类的构造函数中，获取 XML 并将其赋值给类的成员 \$mxml，同时把 XSL 赋值给类的成员 \$mxsl:

```
//构造函数，基于特定的 feed 创建 XML 对象
```

```
function __construct($szFeed)
{
    //从 SimpleXML 对象中取回 RSS feed
    $this->mXml = simplexml_load_file(urldecode($szFeed));
    //从 SimpleXML 对象中取回 XSL
    $this->mXsl = simplexml_load_file('rss_reader.xml');
}
```

getFormattedXML()函数创建一个新的 XSLProcessor 对象来执行 XSL 转换。当 XSL 执行完毕，transformToXML 方法返回一个格式化的 XML 文档。

```
//基于取回的 feed 创建一个 XML 格式的文档
public function getFormattedXML()
{
    //创建一个 XSLTProcessor 对象
    $proc = new XSLTProcessor;
    //附上 XSL
    $proc->importStyleSheet($this->mXsl);
    //执行转换并返回一个 XML 串，该串是格式化数据
    return $proc->transformToXML($this->mXml);
}
```

要使用 XSL 达到什么样的结果呢？就是遍历 XML 的每一个记录并且显示里面的数据。每个记录都是通过<item>和</item>标签来区分。

在 rss\_reader.xml 中我们定义下面的循环：

```
<xsl:for-each select="rss/channel/item">
```

比如，要显示当前标题，可以这样写：

```
<h3><xsl:value-of select="title" /></h3>
```

注意我们是如何使用 XSLT 来创建新的<a>元素的：

```
<xsl:element name="a">
  <xsl:attribute name="href">
    <xsl:value-of select="link" />
  </xsl:attribute>
  read full article
</xsl:element>
```

在这一章中，我们都使用这个技术来创建链接。

还有 CSS 代码，它使网页能按照我们的想法来产生输出。如果快速浏览一下 rss\_reader.css，就会很快明白。

## 9.3 小结

现在的 Web 不同于以往的 Web，并且未来的 Web 也肯定不同于现在的 Web。过去的 Web 是链接的集合，都是静态的，每个站点都需要保存一份。现在 Web 的特点在于站点或应用程序间信息的交换。

运用这一章学习的知识，可以建立一个更好的 RSS 阅读器，那为什么要停止不前呢？您已经拥有了强大的工具来建立更好的应用程序，它们也许会影响到未来的 Web。



# 第 10 章

## Ajax 的拖放功能

当拖放功能第一次引入网站中时，它就给人们带来了惊喜，为网站提供拖放功能确实是不凡的特性！此后，JavaScript 在人们的视野中就从一个提供一些动态效果的脚本语言发展为一个标准且强大的“可以做复杂事情”的语言。

现在已开发出多种框架和 JavaScript 工具集，并仍在不断推陈出新。script.aculo.us 是众多 JavaScript 流行工具集中的一个，它可以在网页中实现惊人的功能——相关内容可以查看其官方网站中的例子 (<http://script.aculo.us>)。Script.aculo.us 是一个在 MIT 许可格式下发布的开放源代码的 JavaScript 结构，只要包含版权说明，就可以在任何地方使用。

本章将建立支持数据库的 Ajax 分类列表，通过该实例的学习，您将学习如何把 Script.aculo.us 特性集成到自己的网站中。

### 10.1 在 Web 上使用拖放功能

在研究现存网站应用中的拖放功能之后，可以发现拖放功能至少在两个方面能够使人机界面和交互更加友好，拖放功能能够成功地应用在：

- 购物车
- 分类列表

#### 10.1.1 购物车

读者可能对传统的电子商务网站比较熟悉，但是在 Ajax 如此盛行的时代，新一代的购物车已经出现了，在这里用户必须使用拖放功能将产品添加到购物车中，而不是单击“添加到购物车”按钮。当有人怀疑这一特性的实用性时（我的祖母仍然喜欢用按钮，她不知道怎样使用拖放功能），视觉效果会更加令人印象深刻。

一些网站已经将拖放功能付诸实现，Panic Goods 就是一个例子，它是出售 T 恤的网站，网址是 <http://www.panic.com/goods>。

注意到屏幕底部有一个亮蓝色工具条了吗，那就是购物车。从分类中拖动 T 恤，然后放入到购物车中，看看购物车是怎样完成的。产品列在购物车中，很容易看到选了哪些商品及其数量。把商品拖出亮蓝色的工具条就可以把它们从购物车中移走。

### 10.1.2 分类列表

有一种列表可能每天都会用到，它就是 to-do 列表，常常用黄色来标注它，有人甚至使用特定的软件。但是现在有如此多新的网络应用可用，有一打的 to-do 列表应用是没问题的！我只说明 Ta-da 列表 (<http://www.tadalist.com>)，它是 37signals 公司创建的。这个公司重新定义了网络应用的概念，并将其发展到一个新的高度。Ta-da 列表，这款早期的产品是一种允许创建几个 to-do 列表的工具，每个 to-do 列表都有自己的项目（那就是，有事情可做），它是一个很有用的工具，很多人都在使用，虽然其中一些人已经更新使用 37 signals 公司的其他产品，如 Basecamp (<http://www.basecamp.com>) 和 Backpack (<http://www.backpackit.com>)。

尽管 Ta-da 列表有直观的用户界面和易用的操作，但它仍然缺乏一种能大大提高其应用性的基本特性，这就是通过拖放列表项来重新安排列表。要想在 Ta-da 列表中重新安排一个列表，必须单击链接刷新页面以便显示 4 个带箭头的按钮（向前、上移、下移和向后）。

虽然这样重排列表也可以达到目的，但拖放系统可以使这一过程完成得更快且更易使用。37 signals 公司已经在 Basecamp 中增加了这一功能，在这一产品中，to-do 列表有了拖拉功能。它以更高级的方式提供了拖放概念并且实用性很强。

## 10.2 创建 Ajax 拖放分类列表应用

能将 Ajax 分类列表与本书其他应用区分开的是它要使用两个外部 JavaScript 框架结构，即 Prototype 和 script.aculo.us。

“Prototype 是一种 JavaScript 的框架，旨在使得动态网站应用的开发变得更容易。”它是 Sam Stephenson 开发的，其功能强大，很快就变成了 JavaScript 的框架。



**注意** Prototype 的发布是遵循 MIT 许可形式的，可以在 <http://prototype.conio.net> 下载到。如果想进一步学习 Prototype，可以在 <http://www.particletree.com/feature/quick-guide-to-prototype> 查看相应指南。

Prototype 的特征如下：

- 完全的面向对象；
- 实用的功能；
- 表单帮助功能；

- 对 Ajax 的支持；
- 定期实施。

JavaScript 的另一个先驱是 Thomas Fuchs，他建立了强大的 JavaScript 库——`script.aculo.us`——它提供了一种引人入胜的可视化功能。在拖放应用中将使用这个库的部分特征（更具体一些，就是拖放特性）。`Script.aculo.us` 是建立在 Prototype 基础之上的，因此继承了 Prototype 的特性。

`Script.aculo.us` 的特征如下：

- 完全的面向对象；
- 可视化的效果（淡入、淡出、增长、收缩、盲下、盲上、震动等）；
- 对拖放的支持；
- 自动完成；
- 在位编辑；
- 滑动控制。

我们将要创建的应用程序是一个小的任务管理程序，它允许创建新任务，重新安排已有的任务，还可以删除任务。其特征归纳如下：

- 后台数据库；
- 拖放项；
- 用 Ajax 增加新任务；
- 当拖放时立即更新数据库；
- 通过将任务拖放到指定地点删除任务。

如图 10.1 所示是这一任务管理应用的界面。

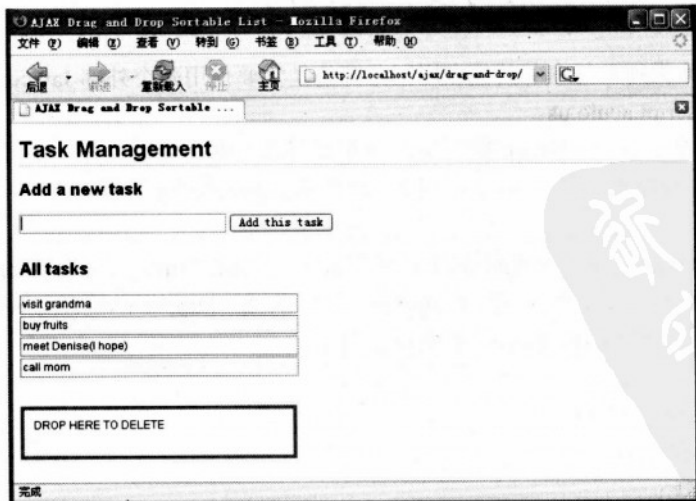


图 10.1 在一个简单的可视化界面中增加、重新安排及删除任务

在屏幕周围拖动项目会交换它们的位置。

当将一个任务放置在 DROP HERE TO DELETE 区域时, 在真正删除之前需要一个确认信息, 如图 10.2 所示。

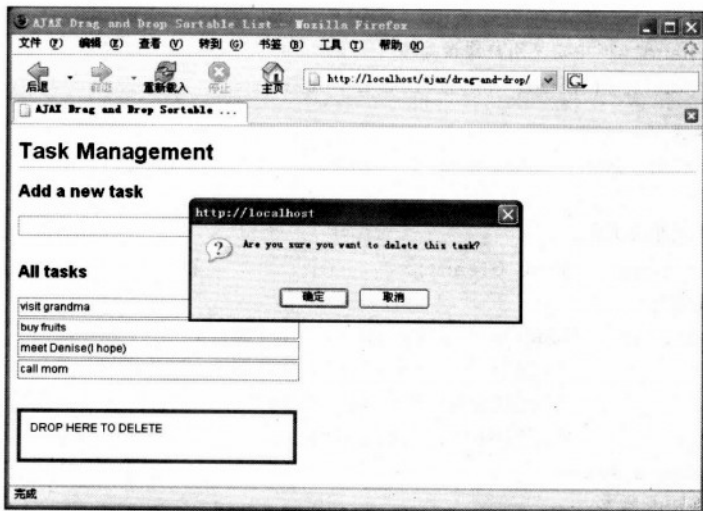


图 10.2 删除任务之前需要的确认信息

## 》》》 实现步骤——用 Ajax 完成任务管理应用

(1) 连接 ajax 数据库, 用下面的代码创建一个名为 tasks 的表。

```
CREATE TABLE tasks (
  id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  order_no INT UNSIGNED NOT NULL default '0',
  description VARCHAR(100) NOT NULL default '',
  PRIMARY KEY (id)
);
```

(2) 在 ajax 文件夹中创建文件夹 drag-and-drop。

(3) 在 drag-and-drop 文件夹中, 创建文件 config.php, 写入数据库配置如下代码:

```
<?php
//定义数据库连接数据
define('DB_HOST', 'localhost');
define('DB_USER', 'ajaxuser');
define('DB_PASSWORD', 'practical');
```

```
define('DB_DATABASE', 'ajax');
?>
```

(4) 现在添加错误处理文件 `error_handler.php`，从前面章节复制更容易。下面是它的代码。

```
<?php
//在 error_handler 中定义用户错误处理
set_error_handler('error_handler', E_ALL);

//错误处理函数
function error_handler($errNo, $errStr, $errFile, $errLine)
{
    //清除所有已生成的输出
    if(ob_get_length()) ob_clean();
    //输出错误信息
    $error_message = 'ERRNO: ' . $errNo . chr(10) .
                    'TEXT: ' . $errStr . chr(10) .
                    'LOCATION: ' . $errFile .
                    ', line ' . $errLine;

    echo $error_message;
    //预防处理 PHP 脚本
    exit;
}
?>
```

(5) 从 <http://script.aculo.us/downloads> 下载 `script.aculo.us` 库，并解压到 `drag-and-drop` 文件夹中，将 `script.aculo.us` 文件夹的名字改为 `scriptaculous`。

(6) 创建文件 `index.php`，添加如下代码：

```
<?php
    require_once ('taskslist.class.php');
?>
<!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 Drag and Drop Sortable List</title>
        <link href="drag-and-drop.css" rel="stylesheet" type="text/css" />
        <script src="drag-and-drop.js" type="text/javascript"></script>
        <script src="scriptaculous/lib/prototype.js" type="text/javascript">
        </script>
        <script src="scriptaculous/src/scriptaculous.js" type="text/javascript">
```



```

</script>
</head>
<body onload="startup()">
  <h1>Task Management</h1>
  <h2>Add a new task</h2>
  <div>
    <input type="text" id="txtNewTask" name="txtNewTask"
      size="30" maxlength="100" onkeydown="handleKey(event)"/>
    <input type="button" name="submit" value="Add this task"
      onclick="process('txtNewTask', 'addNewTask')" />
  </div>
  <br />
  <h2>All tasks</h2>
  <ul id="tasksList" class="sortableList"
    onmouseup="process('tasksList', 'updateList')">
    <?php
      $myTasksList = new TasksList();
      echo $myTasksList->BuildTasksList();
    ?>
  </ul>
  <br /><br />
  <div id="trash">
    DROP HERE TO DELETE
  <br /><br />
  </div>
</body>
</html>

```

(7) 创建文件 `tasklist.class.php`, 添加如下代码:

```

<?php
//加载错误处理和数据库配置
require_once ('error_handler.php');
require_once ('config.php');

//这个类建立一个任务列表, 并在其上完成添加/删除/重排操作
class TasksList
{
  //存储数据库连接
  private $mMysqli;

```

```
//constructor 函数打开数据库连接
function __construct()
{
    //连接到数据库
    $this->mMysqli = new mysqli(DB_HOST, DB_USER, DB_PASSWORD,
                               DB_DATABASE);
}

//destructor 函数用来关闭数据库连接
public function __destruct()
{
    $this->mMysqli->close();
}

//建立任务列表
public function BuildTasksList()
{
    //初始化输出
    $myList = '';
    //建立查询
    $result = $this->mMysqli->query('SELECT * FROM tasks ' .
                                   'ORDER BY order_no ASC');
    //将任务列表建成<li>元素
    while ($row = $result->fetch_assoc())
    {
        $myList .= '<li id="' . htmlentities($row['id']) . '">' .
                  htmlentities($row['description']) . '</li>';
    }
    //返回列表
    return $myList;
}

//处理服务器端数据处理
public function Process($content, $action)
{
    //完成客户请求操作
    switch($action)
    {
        //重排任务列表
        case 'updateList':
```

```

//重获更新详细信息
$new_order = explode('_', $content);
//更新列表

for ($i=0; $i < count($new_order); $i++)
{
    //避免从客户端接收数据
    $new_order[$i] =
        $this->mMysqli->real_escape_string($new_order[$i]);
    //更新任务
    $result = $this->mMysqli->query('UPDATE tasks SET order_no="' .
        $i . '" WHERE id="' . $new_order[$i] . '"');
}
$updateList = $this->BuildTasksList();
return $updateList;
break;

//增加一个新任务
case 'addNewTask':
    //避免输入数据
    $task = trim($this->mMysqli->real_escape_string($content));
    //只有任务名不为空时继续
    if ($task)
    {
        //获得最大得 order_no
        $result = $this->mMysqli->query('SELECT (MAX(order_no) + 1) ' .
            'AS order_no FROM tasks');
        $row = $result->fetch_assoc();
        //如果表为空, order_no 为空
        $order = $row['order_no'];
        if (!$order) $order = 1;
        //将新任务插入到列表底部
        $result = $this->mMysqli->query
            ('INSERT INTO tasks (order_no, description) ' .
                'VALUES (" . $order . '", "' . $task . '"');
        //返回更新后得任务列表
        $updateList = $this->BuildTasksList();
        return $updateList;
    }
break;

```

```

//删除任务
case 'delTask':
    //避免输入数据
    $content = trim($this->mMysqli->real_escape_string($content));
    //删除任务
    $result = $this->mMysqli->query('DELETE FROM tasks WHERE id="' .
                                    $content . '"');
    $updatedList = $this->BuildTasksList();
    return $updatedList;
    break;
}
}
?>

```

(8) 创建文件 `drag-and-drop.php`, 添加如下代码:

```

<?php
//加载帮助类
require_once ('taskslist.class.php');
//创建任务列表对象
$myTasksList = new TasksList();
//读取参数
$action = $_GET['action'];
$content = $_GET['content'];
//清除输出
if(ob_get_length()) ob_clean();
//被发送的 headers 用来防止隐藏浏览器
header('Expires: Fri, 25 Dec 1980 00:00:00 GMT'); //time in the past
header('Last-Modified: ' . gmdate( 'D, d M Y H:i:s') . 'GMT');
header('Cache-Control: no-cache, must-revalidate');
header('Pragma: no-cache');
header('Content-Type: text/html');
//执行客户请求并返回更新后的任务列表
echo $myTasksList->Process($content, $action);
?>

```

(9) 创建文件 `drag-and-drop.js`, 添加如下代码:

```

//保存 XMLHttpRequest 的一个实例
var xmlHttp = createXmlHttpRequestObject();
//当置为真时, 显示详细的错误信息

```

```
var showErrors = true;
//初始化请求缓存
var cache = new Array();

//创建一个 XMLHttpRequest 实例
function createXmlHttpRequestObject()
{
    //存储 XMLHttpRequest 对象的引用
    var xmlHttp;
    //这是为 IE6 及其旧版本之外的其他浏览器工作的
    try
    {
        //试着创建 XMLHttpRequest 对象
        xmlHttp = new XMLHttpRequest();
    }
    catch(e)
    {
        //假定是 IE6 或老版本
        var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
                                          "MSXML2.XMLHTTP.5.0",
                                          "MSXML2.XMLHTTP.4.0",
                                          "MSXML2.XMLHTTP.3.0",
                                          "MSXML2.XMLHTTP",
                                          "Microsoft.XMLHTTP");

        //试着捕获每个 id 直到其中有一个可以工作
        for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
        {
            try
            {
                //试着创建 XMLHttpRequest 对象
                xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
            }
            catch (e) {} //忽略潜在的错误
        }
    }
    //返回已创建的对象或显示错误信息
    if (!xmlHttp)
        alert("Error creating the XMLHttpRequest object.");
    else
        return xmlHttp;
}
```

```
}

//显示错误信息的函数
function displayError($message)
{
    //如果 showErrors 为假的话忽略错误信息
    if (showErrors)
    {
        //将错误显示关闭
        showErrors = false;
        //显示错误信息
        alert("Error encountered: \n" + $message);
    }
}

//定义可分类列表和放置区的详细代码
function startup()
{
    //用拖拉项将一个无序的列表转换为一个可分类的列表
    Sortable.create("tasksList", {tag:"li"});

    //定义一个用来删除任务的放置区
    Droppables.add("trash",
    {
        onDrop: function(element)
        {
            var deleteTask =
                confirm("Are you sure you want to delete this task?")
            if (deleteTask)
            {
                Element.hide(element);
                process(element.id, "delTask");
            }
        }
    });
}

//使列表项的 id 值 (<li>连续)
function serialize(listID)
{
```

```
//计算列表项
var length = document.getElementById(listID).childNodes.length;
var serialized = "";
//遍历每个元素
for (i = 0; i < length; i++)
{
    //获得当前元素
    var li = document.getElementById(listID).childNodes[i];
    //获取当前元素的 id, 不要文本部分
    var id = li.getAttribute("id");
    //只将数字放到 id 数组中
    serialized += encodeURIComponent(id) + "_";
}
//返回去掉后缀 '_' 的数组
return serialized.substring(0, serialized.length - 1);
}

//发送请求到服务器
function process(content, action)
{
    //仅当 xmlhttp 不为空时继续
    if (xmlHttp)
    {
        //将请求查询串初始化为空串
        params = "";
        //确保数值安全地发送到服务器
        content = encodeURIComponent(content);
        //根据不同的动作发送不同的参数
        if (action == "updateList")
            params = "?content=" + serialize(content) + "&action=updateList";
        else if (action == "addNewTask")
        {
            //准备发送到服务器的任务
            var newTask =
                trim(encodeURIComponent(document.getElementById(content).value));
            //不要添加空任务
            if (newTask)
                params = "?content=" + newTask + "&action=addNewTask";
        }
        else if (action == "delTask")
```

```
    params = "?content=" + content + "&action=delTask";  
    //不要在缓存中添加空参数  
    if (params) cache.push(params);  
  
    //尝试连接到服务器  
    try  
    {  
        //仅当连接空闲并且缓存不空时才继续下面的操作  
        if ((xmlHttp.readyState == 4 || xmlHttp.readyState == 0)  
            && cache.length>0)  
        {  
            //从缓存中获取值的下一个集合  
            var cacheEntry = cache.shift();  
            //初始化请求  
            xmlHttp.open("GET", "drag-and-drop.php" + cacheEntry, true);  
            xmlHttp.setRequestHeader("Content-Type",  
                                     "application/x-www-form-urlencoded");  
            xmlHttp.onreadystatechange = handleRequestStateChange;  
            xmlHttp.send(null);  
        }  
        else  
        {  
            setTimeout("process();", 1000);  
        }  
    }  
    //当失败时显示错误信息  
    catch (e)  
    {  
        displayError(e.toString());  
    }  
}  
//重新获取 HTTP 响应的函数  
function handleRequestStateChange()  
{  
    //当 readyState 是 4 的时候,也读取服务器的响应  
    if (xmlHttp.readyState == 4)  
    {  
        //仅当 HTTP 的状态为 "OK"时继续下面的操作  
        if (xmlHttp.status == 200)
```



```
{
    try
    {
        postUpdateProcess();
    }
    catch(e)
    {
        //显示错误信息
        displayError(e.toString());
    }
}
else
{
    displayError(xmlHttp.statusText);
}
}
}
//处理服务器响应
function postUpdateProcess()
{
    //读取响应
    var response = xmlHttp.responseText;
    //服务器错误?
    if (response.indexOf("ERRNO") >= 0 || response.indexOf("error") >= 0)
        alert(response);
    //更新任务列表
    document.getElementById("tasksList").innerHTML = response;
    Sortable.create("tasksList");
    document.getElementById("txtNewTask").value = "";
    document.getElementById("txtNewTask").focus();
}
/*处理 keydown 以检测是否按下 Enter 键*/
function handleKey(e)
{
    //获取事件
    e = (!e) ? window.event : e;
    //获取已经按下的字符的代码
    code = (e.charCode) ? e.charCode :
            ((e.keyCode) ? e.keyCode :
            ((e.which) ? e.which : 0));
```

```
//处理 keydown 事件
if (e.type == "keydown")
{
    //如果输入 (code 13) 被按下
    if(code == 13)
    {
        //发送当前信息
        process("txtNewTask", "addNewTask");
    }
}
/* 从字符串中移除开头和末尾的空格 */
function trim(s)
{
    return s.replace(/(^s+)|(\s+$)/g, "");
}
```

(10) 创建文件 `drag-and-drop.css`, 添加如下代码:

```
body
{
    font-family: Arial, Helvetica, sans-serif;
    font-size: 12px;
}
ul.sortableList
{
    list-style-type: none;
    padding: 0px;
    margin: 0px;
    width: 300px;
}
ul.sortableList li
{
    cursor: move;
    padding: 2px 2px;
    margin: 2px 0px;
    border: 1px solid #00CC00;
    background-color: #F4FFF5;
}
h1
{
```

```
border-bottom: 1px solid #cccccc;
}
#trash
{
border: 4px solid #ff0000;
width: 270px;
padding: 10px;
}
```

(11) 在浏览器中输入 `http://localhost/ajax/drag-and-drop`，测试其功能，看它是否能完成预想的功能（图 10.1 和图 10.2 可作参考）。

## 程序说明

增加任务的步骤如下。

(1) 用户输入任务。

(2) 当用户单击 Add this task 按钮或按下 Enter 键时，数据通过异步的 HTTP 请求发送到服务器端，服务器的脚本将新任务插入到数据库中，并返回更新列表，然后用 JavaScript 将它插入到代码中。

当重排列表时，发生下面这两件事情。

(1) 每一个任务都是一个 XHTML 列表元素——`<li>`元素。用户开始拖一项，在放的时候，HTTP 请求被发送到服务器端，这个请求由一系列连续的 ID 字符串组成，每个列表元素的 ID 都在其中。

(2) 当服务器更新数据库中每个元素的顺序时，在客户端就会看到重新排列的列表。

下面是怎样删除任务：

(1) 用户拖动一项内容并把它放到 DROP HERE TO DELETE 区。

(2) 向服务器端发送一个 HTTP 请求，该请求从数据库中删除任务，并且 XHTML 元素也立即销毁。

在 `index.php` 文件中包含了将要用到的 JavaScript 库：

```
<script src="drag-and-drop.js" type="text/javascript"></script>
<script src="scriptaculous/lib/prototype.js" type="text/javascript">
</script>
<script src="scriptaculous/src/scriptaculous.js" type="text/javascript">
</script>
```

第一行包含自定义的功能和与 Ajax 相关的任务，第二行包含了 Prototype 库，第三行包含了 `script.aculo.us` 库。

`<body>` 标签中的 `onload` 事件调用 `startup()` 函数，它用 `id="tasklist"` 作为分类元素

(`sortable.create`) 来定义无序列表。这就保证了列表中 `<li>` 元素的拖放功能。`startup()` 函数还定义了一个可放置元素 `Droppable.add`，可以在此删除任务。

而且，`startup()` 函数中定义在放置区删除列表项的操作：

```
onDrop: function(element)
{
    var deleteTask =
        confirm("Are you sure you want to delete this task?")
    if (deleteTask == true)
    {
        Element.hide(element);
        process(element.id, "delTask");
    }
}
```

这段代码要求用户输入确认信息，如果收到确认信息，就会将这一元素隐藏在幕后并调用 `process()`，由它来发送 HTTP 请求。

在 `index.php` 中有一小段动态创建列表的代码：

```
<ul id="tasksList" class="sortableList"
    onmouseup="process('tasksList', 'updateList')">
    <?php
        $myTasksList = new TasksList();
        echo $myTasksList->BuildTasksList();
    ?>
</ul>
```

单击增加任务按钮或按 `Enter` 键，就会添加新任务。

真正的 Ajax 请求由 `process()` 函数发送，该函数为这三种操作（重排列表/增加任务/删除任务）都发送请求。将要完成的操作作为它的参数。

当增加新任务时，`process()` 函数的第一个参数是文本域的 ID，其中我们已经输入了一个新任务。

```
<input type="button" name="submit" value="Add this task"
    onclick="process('txtNewTask', 'addNewTask')"/>
```

列表重排之后的数据库更新由 `onmouseup` 事件触发，这一事件存在于 `id` 为“`tasklist`”的无序列表中——我们的可分类列表中。此事件调用 `process()` 函数，它将列表的 ID 作为其第一个参数。

```
<ul id="tasksList" class="sortableList"
    onmouseup="process('tasksList', 'updateList')">
```

因为将要给服务器发送值数组，需要将数据序列化，这一任务是由 `serialize` 函数完成的，一个我们自己写的函数。该函数会统计可以收到的 `<li>` 元素，然后循环将每一个元素的 ID 加入到字符串中，最后还需要将返回值末尾的 “\_” 去掉。

```
function serialize(listID)
{
    //计算列表项
    var length = document.getElementById(listID).childNodes.length;
    var serialized = "";
    //遍历每个元素
    for (i = 0; i < length; i++)
    {
        //获得当前元素
        var li = document.getElementById(listID).childNodes[i];
        //在不将文本分开的情况下获取当前元素的 id
        var id = li.getAttribute("id");
        //只把数字加到 id 数组中
        serialized += encodeURIComponent(id) + "_";
    }
    //返回去掉后缀 “_” 的数组
    return serialized.substring(0, serialized.length - 1);
}
```

记住，XMLHttpRequest 不能同时完成两个 HTTP 请求，所以如果 XMLHttpRequest 对象正忙于处理先前的请求，我们就要存储当前请求的信息以备后用，这在连接网络或 Internet 较慢时尤为有用，请求的信息以 FIFO 形式使用缓存系统进行存储。幸运的是，JavaScript 的 Array() 类提供了我们所需的特定功能（通过它的 `push` 和 `shift()` 方法），因此，我们把它用作缓存。

```
Var cache=new Array();
```

所以，在 `process()` 中，在向服务器发送新请求之前，将当前请求保存到缓存中。

```
//仅当 xmlhttp 不为空时继续
if (xmlhttp)
{
    if(action)
        Cache.push(content +"&" +action);
}
```

这将在缓存数组的末尾增加一个新元素。元素由两部分组成：一个目录（HTML 元素的 ID）和一个由服务器完成的动作组成，它们之间由 “&” 分隔，注意只有在 `action` 不为空时才添加新元素，这只有在不根据用户的请求调用函数时才会发生，但要检查是否会产生一些

待判断的动作。

之后, 如果 XMLHttpRequest 对象空闲下来, 可以开始调用其他请求时, 我们就使用 shift() 来提取刚才用 push() 添加的请求, 在 FIFO 方案中, 最早存储的请求最早执行。

```
//尝试连接到服务器
try
{
    //仅当 XMLHttpRequest 对象不忙并且缓存不空时继续下面的操作
    if ((xmlHttp.readyState == 4 || xmlHttp.readyState == 0)
        && cache.length>0)
    {
        //从缓存中获取下一个值的集合
        var cacheEntry = cache.shift();
```

如果 HTTP 的状态是 0 或 4, 表明现在没有激活的请求, 我们可以发送一个新的请求。要想发送新的请求, 先要读取缓存中的数据, 并将当前的条目分成两个变量:

```
//分离数组元素
Var values=cacheEntry.split("&");
Content=values[0];
Action=values[1];
```

根据这些变量, 发送不同的参数值:

```
//根据不同的动作发送不同的参数
if (action == "updateList")
    params = "?content=" + serialize(content) + "&action=updateList";
else if (action == "addNewTask")
{
    //准备发送到服务器的任务
    var newTask =
        trim(encodeURIComponent(document.getElementById(content).value));
    //不要添加空任务
    if (newTask)
        params = "?content=" + newTask + "&action=addNewTask";
}
else if (action == "delTask")
    params = "?content=" + content + "&action=delTask";
```

下面的数据用来处理服务器请求:

```
xmlHttp.open("POST", "drag-and-drop.php", true);
```

```
xmlHttp.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
xmlHttp.onreadystatechange = handleRequestStateChange;
xmlHttp.send(params);
```

服务器的响应是由 `handleRequestAtateChange` 函数来处理的，然后调用 `postUpdateProcess()`，此处我们重新获取服务器响应，得到的可能是一个错误信息，也可能是一个更新列表字符串自动生成的 HTML 代码。

```
//读取响应
var response = xmlHttp.responseText;
//服务器错误?

if (response.indexOf("ERRNO") >= 0 || response.indexOf("error") >= 0)
    alert(response);
//更新任务列表
document.getElementById("tasksList").innerHTML = response;
Sortable.create("tasksList");
document.getElementById("txtNewTask").value = "";
document.getElementById("txtNewTask").focus();
}
```

最后两行代码清除了“新任务”文本域，并将指针指向这个域。

`drag-and-drop.php` 脚本确实是轻量级的，其中包含了 `taskslist.class.php`，它初始化一个 `TasksList` 对象，并在调用 `process` 类方法之后返回更新列表，这将完成三个动作之一增加任务、重排列表、删除任务。

`Taskslist.class.php` 文件是一个用来在服务列表中完成服务器端操作的类，它的构造函数创建一个数据库连接。之后有了两个公共的方法：

- `BuildTasksList` 为每一个任务创建列表项。
- `Process` 需要两个参数——`$content` 和 `$action`，第一个参数根据第二个参数处理用户数据，第二个参数提示脚本要完成的动作。

当更新列表时（如“`updateList`”），我们从 `$content` 数组中提取值，数组中保存着所有 `<li>` 元素新顺序的连续字符串，那就是所有的任务。接着我们将循环提取其值并且更新数据库。

要增加新任务，先要避免用户使用 `mysqli` 的 `real_escape_string` 输入。接着，需要从数据库中取出现存最大并给它增值，这就是新任务的顺序号，然后将任务插入数据库并返回一个包含顺序号和任务描述的字符串，这些就是要返回给客户端的内容，之后要根据接收到的数据，建立一个新的列表元素。

需要删除任务（如“`delTask`”），仅需要将任务从数据库中删除即可。

每种方法都返回一个带新任务列表的字符串，即一个 `<li>` 元素字符串。



提示

经常过滤用户数据

如果想避免麻烦，就要时常过滤用户的输入数据，我们使用 JavaScript 的 `encodeURIComponent` 函数向服务器发送数据，在服务器上，使用 `mysqli` 对象的 `real_escape_string` 方法阻止 SQL 注入，在服务器端还要使用 `htmlspecialchars` PHP 函数准备返回给客户端的文本内容。

## 10.3 小结

就是这样！现在已经用拖放功能完成了一个可工作的任务管理应用，完成所有功能只用了数量很少的代码，开发应用的下一步可能是通过双击任务使之可编辑，`script.aculo.us` 通过 `Ajax.InPlaceEditor` 提供了一个完成这个任务的强有力的方法。从 <http://wili.script.aculo.us/scriptaculous/show/Ajax.InPlaceEditor> 可查询到更多关于怎样完成这一任务的信息文档。

另一个分类列表的实际应用应该是在 CMS（Content Management System）中，用于管理页面、项目、产品和新闻等的顺序。最后，创建用户界面靠的就是您的想象力以及开拓力。





## 环境配置

为了避免在学习本书的例子时遇到不必要的麻烦，最好从一开始就安装必要的软件并正确配置环境。虽然读者可能已经有了一些开发 PHP 应用的经验，但我们还是要快速地讲解一下安装必要软件的步骤。

好消息是，所需要的软件都是免费、强大且非常容易安装和配置的。不好的消息是，有多种可能的配置，所以安装指南不一定能百分之百地应用在读者的机器上（例如，如果是 Windows 操作系统，则可能更倾向于使用 IIS 而不是 Apache 等）。

我们将根据不同的机器分别介绍 Windows 和 \*nix 的安装指导，我们还要准备数据库，本书中许多例子都要用到。这些指导不仅对 Windows 用户有用，对 \*nix 用户同样有用，所以不要错过附录末尾的这部分内容。

利用 Ajax 和 PHP 建立网站需要（一点也不惊奇）安装 PHP，最好是安装 PHP 5，因为在第 11 章中会用到它的一些特性。还需要一个 Web 服务器，会介绍安装 Apache，它被很多 PHP 开发者和网站开发公司所青睐。我们想尽量将本书的例子做得和现实中的方案一致，这就需要有一个数据库。我们会介绍安装 MySQL，它是 PHP 领域最受欢迎的数据库服务器。因为书中的 SQL 代码很简单所以可以很容易地将其使用在其他的数据库服务器或是 MySQL 老一些的版本，而不用改变主要的代码。

在本章的最后，介绍如何安装 phpMyAdmin，它是非常有用的管理数据库的网络工具，然后读者将学习怎样使用这个工具来创建新的数据库，并创建一个对该数据库拥有完全操作特权的用户。

下面，您将学到怎样安装 phpMyAdmin:

- 在 Windows 系统上安装 Apache2、PHP 5 和 MySQL 5;
- 在 \*nix 系统上安装 Apache2、PHP 5 和 MySQL 5;
- 安装 phpMyAdmin;
- 创建新的数据库，然后创建一个使用 phpMyAdmin 的数据库用户。

**提示**

不想手工安装所需的软件程序员，可以选择使用软件包，如 XAMPP，在一个安装文档中封装了所有需要的软件（甚至更多）。XAMPP 针对 Linux、Windows、Mac OS X 和 Solaris 都有封包，并且是免费的。可以从 <http://www.apachefriend.org/en/xampp.html> 下载 XAMPP 包。

如果决定使用 XAMPP 包，读者可以直接跳转到附录最后的 ajax 数据库安装。

## A.1 Windows 下的环境配置

下面将在 Windows 机器上安装以下软件产品：

- Apache 2
- PHP 5
- MySQL 5

### A.1.1 安装 Apache

可以从 <http://httpd.apache.org/download.cgi> 下载 Apache HTTP 服务器的最新 MSI 安装版本，下载最新的 Win32 二进制部分代码（MSI 安装），下载后 Apache 的名字可能是 `apache_2.x.y-win32-x86-no_ssl.msi`，然后执行它。

Apache 2 的默认安装路径为 `C:\Program Files\Apache Group\Apache2\`，安装提示允许选择不同的路径。可以选择一个更方便的路径（如：`c:\Apache`），这样使用 Apache 就可以更容易些。在安装过程中将会要求输入服务器的信息，如图 A.1 所示。

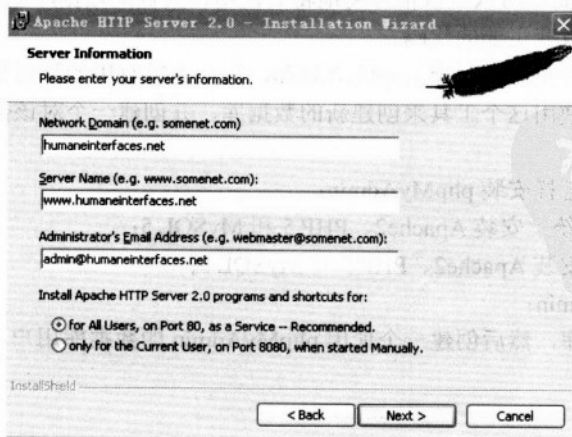


图 A.1 安装 Apache 2.0

如果不确定怎样填写的话,可以将前两项简单填为 localhost,最后一行添上您的电子邮箱即可,以后还可以通过编辑 Apache 配置文档修改这些信息。修改文件的默认路径为 c:\Program Files\Apache Group\Apache2\conf\httpd.conf。

也可以选择将 Apache 安装在 80 端口或 8080 端口,默认的端口为 80。如果机器之前已经装有另一个 Web 服务器(如 IIS),则需要将 Apache 安装在不同的端口上,如果选择将 Apache 安装在 8080 端口上,则需要在 Apache bin 文件夹下(默认为 C:\Program Files\Apache Group\Apache2\bin)手动启动 Apache 服务器,具体是在该文件夹下输入以下命令:

```
apache -k install
```

当 Web 服务器不在 80 端口上工作时,发出 HTTP 请求时就需要手动指定端口,如 http://localhost:8080/ajax/suggest。

在接下来的安装中,可以安全地使用默认选项。

Apache 服务器安装的同时会启动 Apache 服务监控程序,在任务栏上会显示出来。任务栏上的图标反映了当前网络服务器的状态(关闭、运行等),也允许用户启动、停止和重启 Apache 服务。



**注意**

记住,每次修改 httpd.conf 配置文件之后都需要重启(或关闭再启动)Apache 服务,这样是为了使更改生效。

安装完 Apache2 之后,确定它工作正常,如果安装在 80 端口,在浏览器中输入 http://localhost/。如果安装在 8080 端口,在浏览器中输入 http://localhost:8080/。可以看到如图 A.2 所示的 Apache 欢迎信息。

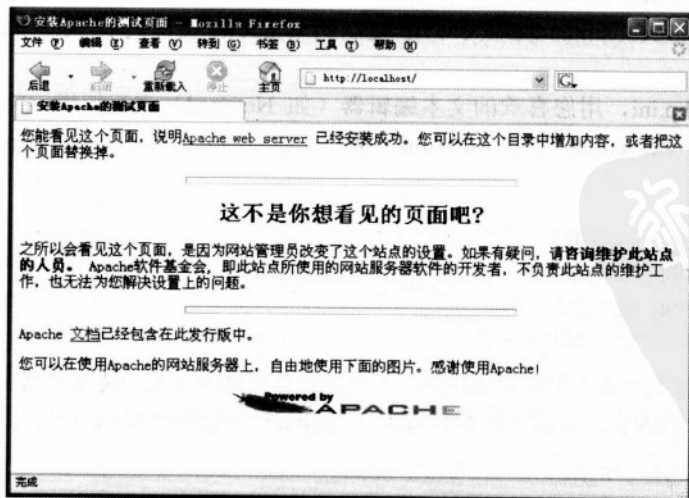


图 A.2 Apache 安装成功

## A.1.2 安装 MySQL

MySQL 的官方网站为 <http://www.mysql.com>。本书截稿时最新的稳定版本为 MySQL 5.0，可以从 <http://dev.mysql.com/downloads/mysql/5.0.html> 下载。不过最好清楚使用的 SQL 查询符合 SQL 92 标准，这样就可以以最小的转换代价在其他的数据库系统中使用了。

在下载页面中，滚动到 Windows 下载区，下载 Windows Essential 文件，它会要求选择一个镜像站点下载，下载的文件名可能会是 `mysql-essential-5.0.xx-win32.msi` 这样的。下载完之后，直接执行即可安装 MySQL 服务器。

安装完之后就有机会配置服务器了。在配置时，安全的做法是一直选择默认值。在要求设置 root 密码的步骤中，这与 `root@localhost` 用户有关，应选择一个别人不容易猜到而自己又容易记住的密码。

在学习本书的所有例子之前，请先参看附录最后“配置 Ajax 数据库”部分。

## A.1.3 安装 PHP

PHP 的官方网站为 <http://www.php.net>。可以从 <http://www.php.net/downloads.php> 下载 PHP 最新压缩包的 Windows 二进制部分（不是安装包）。最好不要使用安装包，因为它不包含扩充部分，而您可能会在项目中用到这部分内容，而且安装包不会给您做很多配置工作。

下载完 Windows 二进制包之后，按照下面的步骤安装 PHP：

(1) 解压压缩包（可能会是一个名为 `php-5.x.y-win32.zip` 的文件）到 `c:\PHP` 文件夹下，如果喜欢的话也可以选择其他的名字和路径。

(2) 将 `php.ini-recommended` 从 `c:\PHP` 复制到 Windows 文件夹下（`c:\windows`），将其重命名为 `php.ini`。

(3) 打开 `php.ini`，用您喜欢的文本编辑器（如 Notepad）进行编辑，将 `php_gd2.dll`，`php_mysql.dll` 和 `php_xsl.dll` 改为 `extension`（移除其前面的分号即可），并在 `php_mysqli` 中增加相似的修改：

```
extension=php_gd2.dll
extension=php_mysql.dll
extension=php_mysqli.dll
extension=php_xsl.dll
```

(4) 建议在开发环境下发送错误报告，不过这项是可选的（上一选择是默认的），这样会收到警告，说明某一修改会改变其他脚本的功能。在 `php.ini` 中找到 `error_reporting` 行并将其修改为：

```
error_reporting=E_ALL
```

(5) 将 `c:\PHP` 下的 `php5ts.dll` 和 `libmysql.dll` 复制到 Windows 的 `system32` 文件夹下（默

认为\windows\system32)。

(6) 将 c:\PHP\ext 下的 php\_mysql.dll、php\_mysqli.dll、php\_xsl.dll 和 php\_gd2.dll 复制到 Windows 的 system32 文件夹下。

(7) 打开 Apache 配置文件进行编辑，默认情况下，这个文件的路径为 c:\Program Files\Apache Group\Apache2\conf\httpd.conf。

(8) 在 httpd.conf 中，找到有许多 LoadModule 条目的部分，添加下面的内容：

```
LoadModule php5_module c:/php/php5apache2.dll
AddType application/x-httpd-php .php
```

(9) 仍在 httpd.conf 中，找到 DirectoryIndex 条目，将 index.php 加到这行的末尾，如下：

```
DirectoryIndex index.html index.html.var index.php
```

(10) 保存 httpd.conf 文件，通过在任务栏的通知区单击 Apache 服务监控程序来重启 Apache2 服务器（如果这一步出现任何错误，确认之前的安装步骤都是正确填写的）。如果 Apache 重新启动而没有任何错误，则是一个好的标志。

(11) 在 htdocs 文件夹下新建一个名为 ajax 的文件夹（默认为 c:\Program Files\Apache Group\Apache2\htdocs）。

(12) 确认您的 Apache 实例可以分析 PHP 代码，在 ajax 文件夹下创建名为 test.php 的文件，将下面代码加到其中：

```
<?php
phpinfo();
?>
```

(13) 在浏览器中输入 http://localhost/ajax/test.php（如果 Apache 工作在 8080 端口上则输入 http://localhost:8080/ajax/test.php），测试安装是否成功，结果页面如图 A.3 所示。

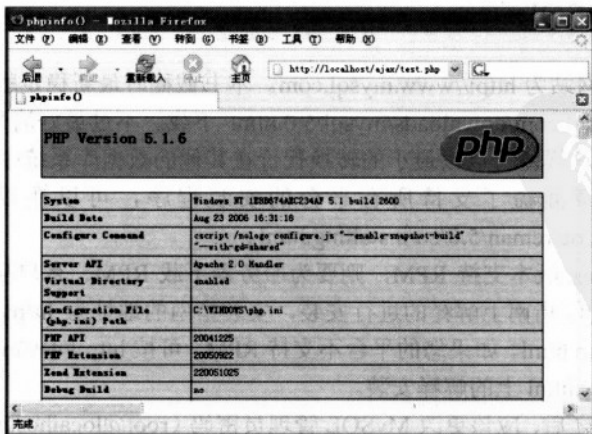


图 A.3 PHP 安装成功

恭喜您，已经完成了 Apache、MySQL 和 PHP 的安装！

配置的建立仍未完成，如果使用的是 Windows，请跳过 \*nix 下环境配置部分，直接到附录最后的 phpMyAdmin 安装和 Ajax 数据库建立部分。

## A.2 \*nix 下的环境配置

几乎所有的 UNIX 和 Linux 发行版本都包含 Apache、PHP 和 MySQL。但还应该检查一下这些程序的版本，最好是有 MySQL 4.1 或更新的版本，至少应该有 PHP 5，这是很重要的，因为这本书的代码在 PHP 低一些的版本中是不可以运行的。

### A.2.1 安装 Apache

在基于 UNIX 的服务器上安装 Apache，按照下面的步骤进行：

(1) 首先，从 <http://httpd.apache.org/download.cgi> 为系统下载 Apache 的 UNIX 源代码，并用如下命令解压：`tar -zxvf httpd-2.0.55.tar.gz`

(2) 在系统上编译安装 Apache 网络服务器。进入到包含代码的文件夹下并执行下面的命令，要求以根用户的身份进入：

```
./configure --prefix=/usr/local/apache2 --enable-so --enable-ssl --withssl  
--enable-auth-digest  
make  
make install
```

### A.2.2 安装 MySQL

MySQL 的官方网站为 <http://www.mysql.com>。本书截稿时最新稳定版本为 MySQL 5.0，可以从 <http://dev.mysql.com/downloads/mysql/5.0.html> 下载。不过最好清楚使用兼容 SQL 92 标准的 SQL 查询，这样就可以以最小的转换代价在其他的数据库系统中使用了。MySQL 5 安装指南的第 2 章涵盖了支持所有平台的安装程序，可以在以下地址找到它：<http://dev.mysql.com/doc/reman/5.0/en/installing.html>。

如果使用的 Linux 版本支持 RPM，则要为服务器下载 RPM、客户端程序和数据库以及头文件，根据 MySQL 指南上解释的进行安装，安装指南的地址为 <http://dev.mysql.com/doc/reman/5.0/en/linux-rpm.html>。如果您的平台不支持 RPM，可按 <http://dev.mysql.com/doc/reman/5.0/en/installing-binary.html> 上的解释安装。

安装完 MySQL 之后，应该更改 MySQL 管理员密码（root@localhost user），默认为空白，可以在 <http://dev.mysql.com/doc/mysql/en/Passwords.html> 读到关于 MySQL 密码的更多内容。

一种改变 root 密码的方法就是执行如下命令：

```
mysqladmin -u root password 'your_new_password.'
```

另外，可以通过使用控制台程序或数据库管理工具，如 phpMyAdmin，来访问 MySQL，执行下面命令：

```
SET PASSWORD FOR root@localhost=PASSWORD('your_new_password');
```

通过在控制台执行下面的命令可以测试 MySQL 服务器：`#mysql -u root -p`

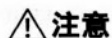
## A.2.3 安装 PHP

每次想要得到一个在 Linux 上运行的新 PHP 库时，都要重新编辑 PHP 模块，这就是为何要建议从一开始就做一个好的编译，把所有的库都加上的原因。

(1) 从 <http://www.php.net/downloads.php> 上下载完整的 PHP 5.x 的源代码文档，并将其内容放置到一个目录中，本书截稿时 PHP 的最新版本为 5.1.2。

(2) 进入放置 PHP 源代码的文件夹，执行下面的命令：

```
./configure --with-config-file-path=/etc --with-mysql=/usr/include/mysql
--with-apxs2=/usr/local/apache2/bin/apxs --with-zlib --with-gd --with-xsl
make
make install
```



**注意**

如果在 XAMPP 中编译 PHP，需要使用下面的配置命令：

```
./configure --with-config-file-path=/opt/lampp/etc --with-mysql=
/opt/lampp
--with-apxs2=/opt/lampp/bin/apxs --with-zlib --with-gd
```

在执行完 `make` 和 `make install` 之后，需要将新生成 `php_src/lib/libphp5.so` 文件复制到 `/opt/lampp/modules/libphp5.so` 下。

(3) 使用下面的命令将 `php.ini-recommended` 复制到 `/etc/php.ini` 文件夹下。

```
cp php.ini-recommended /etc/php.ini.
```

(4) 打开 Apache 配置文件 (`httpd.conf`)，找到 `DirectoryIndex` 条目，在行的末尾增加 `index.php`。

```
DirectoryIndex index.html index.html.var index.php
```

(5) 用下面的命令重启 Apache 服务器：

```
/usr/local/apache2/bin/apachectl restart
```

(6) 在 `htdocs` 文件夹下（默认为 `/usr/local/apache2/htdocs`）新建一个名为 `ajax` 的文件夹。

(7) 为确保 PHP 安装正常, 在刚才创建的 `ajax` 文件夹下创建一个名为 `test.php` 的文件, 并将下面的代码加入其中:

```
<?php
phpinfo();
?>
```

(8) 最后, 在浏览器中输入 `http://localhost/test.php`, 确定在 Apache 下正确安装了 PHP (得到结果如图 A.3 所示)。

## A.3 安装 phpMyAdmin

phpMyAdmin 是用 PHP 写的, 非常受欢迎的一款 MySQL 管理软件, 它允许用一种简单易用的 Web 界面来管理 MySQL 数据库, 官方网站是 `http://www.phpmyadmin.net`。根据下面的步骤安装和配置 phpMyAdmin。

(1) 首先从 `http://www.phpmyadmin.net/home_page/downloads.php` 上下载最新版本的 phpMyAdmin, 如果不能确定应该下载哪个文件, 最安全的方法就是下载 zip 文档。

(2) 将压缩文件解压到硬盘上, 这个文档包含了一个用完整 phpMyAdmin 版本号命名的文件夹(例如, 本书截稿时, phpMyAdmin 测试版本的这个文件夹叫做 `phpMyAdmin-2.8.0-beta1`)。

(3) 为了减少不必要的麻烦, 可将这一文件夹更名为 `phpMyAdmin`。

(4) 将 phpMyAdmin 文件夹移动到 Apache2 的 `htdocs` 文件夹下(默认为 `c:\Program Files\Apache Group\Apache2\htdocs`)。

(5) 确定安装的 phpMyAdmin 可被 Apache 访问, 在浏览器中输入 `http://localhost/phpMyAdmin`, 如果安装成功, 可以得到如图 A.4 所示界面。

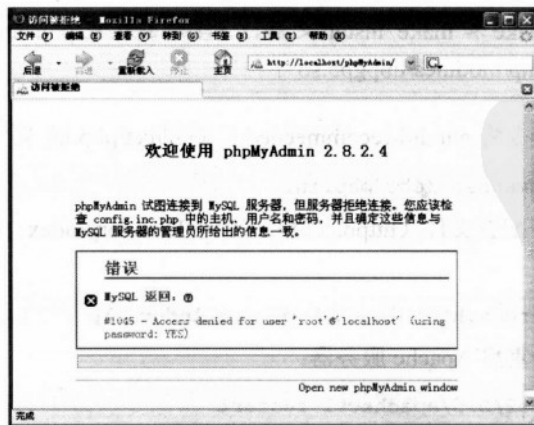
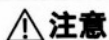


图 A.4 phpMyAdmin 还没有访问到 MySQL



(6) 错误信息已经提示得够多了, 需要指示 phpMyAdmin 怎样访问 MySQL 服务器。在 phpMyAdmin 文件夹下搜索一个名为 config.inc.php 的文件夹, 用下面的代码修改其选项, 如果没有找到此文件, 用下面的内容创建此文件:

```
<?php
$cfg['PmaAbsoluteUri'] = "http://localhost/phpMyAdmin/";
$cfg['Servers'][1]['host'] = "localhost";
$cfg['Servers'][1]['auth_type'] = 'config';
$cfg['Servers'][1]['user'] = "root";
$cfg['Servers'][1]['password'] = "password";
?>
```



**注意**

要到更多关于 phpMyAdmin 安装和配置的细节, 可以在 [http://www.Phpmyadmin.net/home\\_page/doc.php](http://www.Phpmyadmin.net/home_page/doc.php) 上查看其文档。要学习关于 MyAdmin 的内容, 请参考 *Mastering phpMyAdmin for Effective MySQL Management* (ISBN :1-904811-03-5)。

## A.4 配置 Ajax 数据库

在使用 phpMyAdmin 和 MySQL 的练习中, 建立数据库 ajax, 并创建一个拥有全部权限的 MySQL 用户, 可以以这一用户的身份使用 ajax 数据库, 并完成本书所有的例子。可以根据下面的步骤创建。

(1) 在浏览器中输入 <http://localhost/phpMyAdmin> 如果在 config.inc.php 中的配置内容书写正确, 将看到如图 A.5 所示的内容。



图 A.5 phpMyAdmin

(2) 在创建一个新数据库框中写入 `ajax`，并单击“创建”按钮。

(3) `phpMyAdmin` 没有可视化工具创建新用户，所以现在需要写一段 SQL 代码，需要创建一个有完全访问 `ajax` 数据库权限的用户，这一用户将在本书所有的例子中都用到，用户名为 `ajaxuser`，密码是 `practical`。添加此用户，单击页面顶部的 SQL 标志，书写以下代码：

```
GRANT ALL PRIVILEGES ON ajax.*  
TO ajaxuser@localhost IDENTIFIED BY "practical"
```

**注意** SQL 语句看上去不像一般的英语，不过有些地方要说明一下，在 `ajax.*` 中，`*` 表示在 `Ajax` 中的所有对象，所以这一命令告诉 MySQL，“让其赋予本地的 `ajax` 用户可以访问 `Ajax` 数据库的所有可能的权限，密码为 `practical`”。

(4) 单击 Go 按钮。

恭喜您，您已经配置好使用这本书需要的所有软件，希望您在学习 Ajax 时能充满乐趣。

[ General Information ]

书名=AJAX与PHP WEB开发

作者=(罗)Cristian Darie等著;王德民,王新颖,刘昕译

页数=306

出版社=北京市:人民邮电出版社

出版日期=2007

SS号=11827055

DX号=161000064988

URL=<http://book.szdnnet.org.cn/bookDetail.jsp?dxNumber=161000064988&d=6A8E6CDAF46D81A983A94C177C4D9B14>

第1章	Ajax与未来的Web应用程序
1.1	通过Web发布的功能
1.2	20世纪90年代开始出现网站
1.2.1	HTTP与HTML
1.2.2	PHP及其他服务器端技术
1.2.3	JavaScript及其他客户端技术
1.2.4	缺少的是什么
1.3	理解Ajax
1.4	使用Ajax和PHP建立一个简单应用程序
1.5	小结
第2章	JavaScript下灵活的客户端技术
2.1	JavaScript和Document Object Model
2.2	JavaScript事件和DOM
2.3	关于DOM的进一步介绍
2.4	JavaScript、DOM和CSS
2.5	使用XMLHttpRequest对象
2.5.1	创建XMLHttpRequest对象
2.5.2	使用XMLHttpRequest初始化服务器请求
2.5.3	服务器响应处理
2.6	使用XML结构
2.6.1	处理更多的错误和抛出异常
2.6.2	建立XML结构
2.7	小结
第3章	使用PHP和MySQL实现服务器端技术
3.1	PHP与DOM
3.2	参数传递与PHP错误处理
3.3	连接远程服务器与JavaScript的安全性
3.4	使用代理服务器脚本
3.5	重复异步请求框架
3.6	使用MySQL
3.6.1	创建数据库表
3.6.2	数据操作
3.6.3	连接数据库并执行查询操作
3.7	程序封装与程序结构
3.8	小结
第4章	Ajax表单验证
4.1	实现Ajax表单验证
4.2	小结
第5章	Ajax聊天
5.1	Ajax聊天简介
5.2	实现Ajax聊天
5.3	小结
第6章	Ajax建议和自动完成
6.1	Ajax建议和自动完成简介
6.2	实现Ajax建议和自动完成
6.3	小结
第7章	使用SVG实现Ajax实时绘制图表
7.1	使用Ajax和SVG实现实时图表
7.2	小结
第8章	Ajax数据表格
8.1	使用客户端XSLT实现Ajax数据表格
8.2	小结
第9章	Ajax RSS阅读器
9.1	使用RSS
9.1.1	RSS文档结构
9.1.2	Google Reader
9.2	实现基于Ajax的RSS阅读器
9.3	小结
第10章	Ajax的拖放功能
10.1	在Web上使用拖放功能

- 10.1.1 购物车
- 10.1.2 分类列表
- 10.2 创建 Ajax 拖放分类列表应用
- 10.3 小结

#### 附录 A 环境配置

- A.1 Windows 下的环境配置
  - A.1.1 安装 Apache
  - A.1.2 安装 MySQL
  - A.1.3 安装 PHP
- A.2 \*nix 下的环境配置
  - A.2.1 安装 Apache
  - A.2.2 安装 MySQL
  - A.2.3 安装 PHP
- A.3 安装 phpMyAdmin
- A.4 配置 Ajax 数据库