Ajax 基础教程

制作：badnewfish

联系：http://badnewfish.cnblogs.com

1.2 浏览器历史

提到 Web 浏览器，大多数人都会想到无处不在的 Microsoft Internet Explorer，直到最近像 Firefox、Safari 和 Opera 之类的浏览器日益兴起，这种情况才稍有改观。许多新手可能会误认为 IE 是市场上的第一个浏览器，其实不然。实际上，第一个 Web 浏览器出自 Berners-Lee 之手，这是他为 NeXT 计算机创建的（这个 Web 浏览器原来取名叫 WorldWideWeb，后来改名为 Nexus），并在 1990 年发布给 CERN 的人员使用。Berners-Lee 和 Jean-Francois Groff 将 WorldWideWeb 移植到 C，并把这个浏览器改名为 libwww。20 世纪 90 年代初出现了许多浏览器，包括 Nicola Pellow 编写的行模式浏览器（这个浏览器允许任何系统的用户都能访问 Internet，从 Unix 到 Microsoft DOS 都涵盖在内），还有 Samba，这是第一个面向 Macintosh 的浏览器。

1993 年 2 月，伊利诺伊大学 Urbana-Champaign 分校美国国家超级计算应用中心的 Marc Andreessen 和 Eric Bina 发布了 Unix 版本的 Mosaic。几个月之后，Aleks Totic 发布了 Mosaic 的 Macintosh 版本，这使得 Mosaic 成为第一个跨平台浏览器，它很快得到普及，并成为最流行的 Web 浏览器[1]。这项技术后来卖给了 Spyglass，最后又归入 Microsoft 的门下，并应用在 Internet Explorer 中。

1993 年，堪萨斯大学的开发人员编写了一个基于文本的浏览器，叫做 Lynx，它成为了字符终端的标准。1994 年，挪威奥斯陆的一个小组开发了 Opera，到 1996 年这个浏览器得到了广泛使用。1994 年 12 月，Netscape 发布了 Mozilla 的 1.0 版，第一个盈利性质的浏览器从此诞生。2002 年又发布了一个开源的版本，这最终发展为 2004 年 11 月发布的、现在十分流行的 Firefox 浏览器。

当 Microsoft 发布 Windows 95 时，IE 1.0 是作为 Microsoft Plus!包的一部分同时发布的。尽管这个浏览器与操作系统集成在一起，但大多数人还是坚持使用 Netscape、Lynx 或 Opera。IE 2.0 有了很大起色，增加了对 cookie、安全套接字层（Secure Socket Layer，SSL）和其他新兴标准的支持。2.0 版还可以用于 Macintosh，从而成为 Microsoft 的第一个跨平台浏览器。不过，大多数用户还是很执着，仍然坚持使用他们习用的浏览器。

不过到了 1996 年夏天，Microsoft 发布了 IE 3.0 版。几乎一夜之间，人们纷纷拥向 IE。当时，Netscape 的浏览器是要收费的，Microsoft 则免费提供 IE。关于浏览器领域谁主沉浮，因特网社区发生了两极分化，很多人担心 Microsoft 会像在桌面领域一样，在 Web 领域也一统天下。有些人则考虑到安全因素——果然不出所料，发布 3.0 版 9 天之后就报告了第一个安全问题。但是到 1999 年发布 IE 5 时，它已经成为使用最广的浏览器。

1.3 Web 应用的发展历程

最初，所有 Web 页面都是静态的，用户请求一个资源，服务器再返回这个资源。什么都不动，什么都不闪。坦率地讲，对于许多 Web 网站来说，这样也是可以的，这些网站的 Web

页面只是电子形式的文本，在一处生成，内容固定，再发布到多处。在浏览器发展的最初阶段，Web 页面的这种静态性不成问题，科学家只是使用因特网来交换研究论文，大学院校也只是通过因特网在线发布课程信息。企业界还没有发现这个新“渠道”会提供什么商机。实际上，以前公司主页显示的信息通常很少，无非是一些联系信息或者只是一些文档。不过没过多久，Web 用户就开始有新的要求了，希望能得到更动态的网上体验。个人计算机成为企业不可或缺的资源，而且从个人宿舍到住家办公室开始出现越来越多的计算机。随着 Windows 95 的问世，随着人们已经领教了 Corel WordPerfect 和 Microsoft Excel 丰富的功能，用户的期望也越来越高。

1.3.1 CGI

要让 Web 更为动态，第一个办法是公共网关接口（Common Gateway Interface，CGI）。与静态的 Web 获取不同，使用 CGI 可以创建程序，当用户发出请求时就会执行这个程序。假设要在 Web 网站上显示销售的商品，你可以利用 CGI 脚本来访问商品数据库，并显示结果。通过使用简单的 HTML 表单和 CGI 脚本，可以创建简单的网上店面，这样别人就可以通过浏览器来购买商品。编写 CGI 脚本可以用多种语言，从 Perl 到 Visual Basic 都可以，这使得掌握不同编程语言的人都能编写 CGI 脚本。

不过，要创建动态的 Web 页面，CGI 并不是最安全的方法。如果采用 CGI，将允许别人在你的系统上执行程序。大多数情况下这可能没有问题，但是倘若某个用户有恶意企图，则很可能会利用这一点，让系统运行你本来不想运行的程序。尽管存在这个缺陷，到如今 CGI 仍在使用。

1.3.2 applet

很显然，CGI 可以有所改进。1995 年 5 月，Sun 公司的 John Gage 和 Andreessen（目前在 Netscape 通信公司）宣布一种新的编程语言诞生，这就是 Java。Netscape Navigator 为这种新语言提供了支持，最初是为了支持机顶盒。（你可能原认为最早涉足智能家居的公司是 Microsoft 和 Sony 其实不然。）就像所有革命都机缘巧合一样，Java 和因特网的出现恰到好处，在适当的时间、适当的地点横空出世，Java 在 Web 上发布仅几个月，就已经有成千上万的人下载。由于 Netscape 的 Navigator 支持 Java，动态 Web 页面掀开了新的一页：applet 时代到来了。

applet 允许开发人员编写可嵌入在 Web 页面上的小应用程序。只要用户使用支持 Java 的浏览器，就可以在浏览器的 Java 虚拟机（Java Virtual Machine，JVM）中运行 applet。尽管 applet 可以做很多事情，但它也存在一些限制：通常不允许它读写文件系统，它也不能加载本地库，而且可能无法启动客户端上的程序。除了这些限制外，applet 是在一个沙箱安全模型中运行的，这是为了有助于防止用户运行恶意代码。

对许多人来说，最初接触 Java 编程语言就是从 applet 开始的，当时这是创建动态 Web 应用的一种绝好的方法。applet 允许你在浏览器中创建一个胖客户应用，不过要在平台的安全限制范围内。当时，在很多领域都广泛使用了 applet，但是，Web 社区并没有完全被 applet“征服”[2]。胖客户的开发人员都很熟悉一个问题：必须在客户端上部署适当的 Java 版本。因为 applet 在浏览器的虚拟机中运行，所以开发人员必须确保客户端安装了适当版本的 Java。尽管这个问题也可以解决，但它确实妨碍了 applet 技术的进一步推广。而且如果 applet 写得不好，很可能对客户主机造成影响，这使许多客户对于是否采用基于 applet 的解决方案犹豫不定。如果你还不太熟悉 applet，请看图 1-1，图中显示了 Sun 公司提供的时钟

applet。

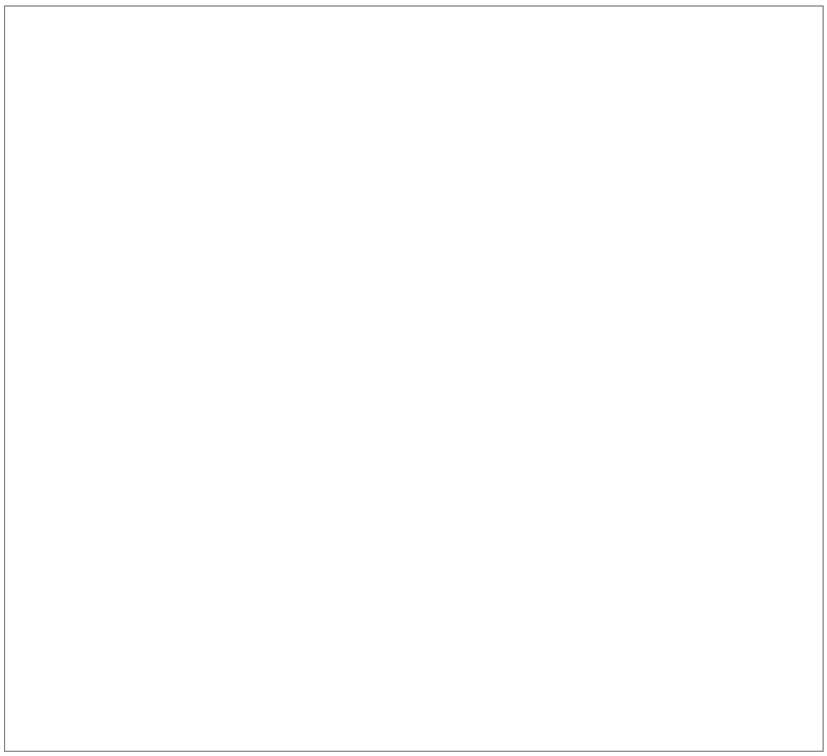


图 1-1　Sun 的时钟 applet

1.3.3 JavaScript

与此同时，Netscape 创建了一种脚本语言，并最终命名为 JavaScript（建立原型时叫做 Mocha，正式发布之前曾经改名为 LiveWire 和 LiveScript，不过最后终于确定为 JavaScript）。设计 JavaScript 是为了让不太熟悉 Java 的 Web 设计人员和程序员能够更轻松地开发 applet（当然，Microsoft 也推出了与 JavaScript 相对应的脚本语言，称为 VBScript）。Netscape 请 Brendan Eich 来设计和实现这种新语言，他认为市场需要的是一种动态类型脚本语言。由于缺乏开发工具，缺少有用的错误消息和调试工具，JavaScript 很受非议，但尽管如此，JavaScript 仍然是一种创建动态 Web 应用的强大方法。

最初，创建 JavaScript 是为了帮助开发人员动态地修改页面上的标记，以便为客户提供更丰富的体验。人们越来越认识到，页面也可以当作对象，因此文档对象模型（Document Object Model，DOM）应运而生。刚开始，JavaScript 和 DOM 紧密地交织在一起，但最后它们还是“分道扬镳”，并各自发展。DOM 是页面的一个完全面向对象的表示，该页面可以用某种脚本语言（如 JavaScript 或 VBScript）进行修改。

最后，万维网协会（World Wide Web Consortium，W3C）介入，并完成了 DOM 的标准化，而欧洲计算机制造商协会（ECMA）批准 JavaScript 作为 ECMAScript 规约。根据这些标准编写的页面和脚本，在遵循相应原则的任何浏览器上都应该有相同的外观和表现。

在最初的几年中，JavaScript 的发展很是坎坷，这是许多因素造成的。首先，浏览器支持很不一致，即使是今天，同样的脚本在不同浏览器上也可能有不同的表现；其次，客户可以自由地把 JavaScript 关闭，由于存在一些已知的安全漏洞，往往鼓励用户把 JavaScript 关掉。由于开发 JavaScript 很有难度（你会用 alert 吗？），许多开发人员退避三舍，有些开发人员

干脆不考虑 JavaScript，认为这是图形设计人员使用的一种“玩具”语言。许多人曾试图使用、测试和调试复杂的 JavaScript，并为此身心俱疲，所以大多数人在经历了这种痛苦之后，最终只能满足于用 JavaScript 创建简单的基于表单的应用。

1.3.4 servlet、ASP 和 PHP……哦，太多了！

尽管 applet 是基于 Web 的，但胖客户应用存在的许多问题在 applet 上也有所体现。在大量使用拨号连接的年代（就算是今天，拨号连接也很普遍），要下载一个复杂 applet 的完整代码，要花很多时间，用户不能承受。开发人员还要考虑客户端上的 Java 版本，有些虚拟机还有更多的要求[3]。理想情况下只需提供静态的 Web 页面就够了，毕竟，这正是设计因特网的本来目的。当然，尽管静态页面是静态的，但是如果能在服务器上□ □ 地生成内容，再把静态的内容返回，这就太好了。

在 Java 问世一年左右，Sun 引入了 servlet。现在 Java 代码不用再像 applet 那样在客户端浏览器中运行了，它可以在你控制的一个应用服务器上运行。这样，开发人员就能充分利用现有的业务应用，而且，如果需要升级为最新的 Java 版本，只需要考虑服务器就行了。Java 推崇“一次编写，到处运行”，这一点使得开发人员可以选择最先进的应用服务器和服务器环境，这也是这种新技术的另一个优点。servlet 还可以取代 CGI 脚本。

servlet 向前迈出了很大一步。servlet 提供了对整个 Java 应用编程接口（API）的完全访问，而且提供了一个完备的库可以处理 HTTP。不过，servlet 不是十全十美的。使用 servlet 设计界面可能很困难。在典型的 servlet 交互中，先要从用户那里得到一些信息，完成某种业务逻辑，然后使用一些“打印行”创建 HTML，为用户显示结果。代码清单 1-1 所示的代码就相当常见。

代码清单 1-1　简单的 servlet 代码

```
response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();


        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet SimpleServlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello World</h1>");
        out.println("<p>Imagine if this were more complex.</p>");
        out.println("</body>");
        out.println("</html>");


        out.close();
```

以上这一小段代码可以生成图 1-2 所示的一个相当简单的 Web 页面。

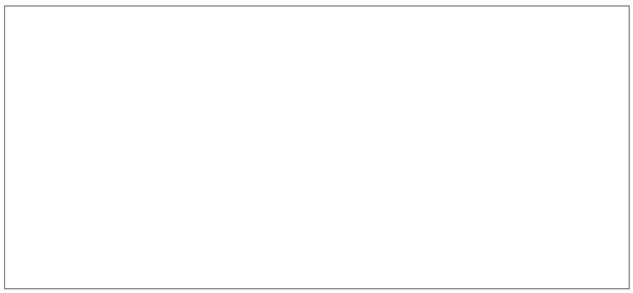


图 1-2　代码清单 1-1 中简单 servlet 的输出

servlet 不 仅容易出错，很难生成可视化显示，而且还无法让开发者尽展其才。一般地，编写服务器端代码的人往往是软件开发人员，他们只是对算法和编译器很精通，但不是 能设计公司精美网站的图形设计人员。业务开发人员不仅要编写业务逻辑，还必须考虑怎么创建一致的设计。因此，很有必要将表示与业务逻辑分离。因此 JSP（JavaServer Pages）出现了。

在某种程度上，JSP 是对 Microsoft 的 Active Server Pages (ASP）做出的回应。Microsoft 从 Sun 在 servlet 规约上所犯的错误汲取了教训，并创建了 ASP 来简化动态页面的开发。Microsoft 增加了非常好的工具支持，并与其 Web 服务器紧密集成。JSP 和 ASP 的设计目的都是为了将业务处理与页面外观相分离，从这个意义上讲，二者是相似的。虽然存在一些技术上的差别（Sun 也从 Microsoft 那里学到了教训），但它们有一个最大的共同点，即 Web 设计人员能够专心设计页面外观，而软件开发人员可以专心开发业务逻辑。代码清单 1-2 显示了一个简单的 JSP。

代码清单 1-2　简单的 JSP

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Hello World</title>
    </head>
    <body>

    <h1>Hello World</h1>
    <p>This code is more familiar for Web developers.</p>
```

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

这个代码会生成图 1-3 所示的输出。

图 1-3　简单 JSP 的输出

当然，Microsoft 和 Sun 并没有垄断服务器端解决方案。还有许多其他的方案在这个领域都有一席之地，如 PHP 和 ColdFusion 等等。有些开发人员喜欢新奇的工具，还有一些则倾向于更简单的语言。目前来看，所有这些解决方案完成的任务都是一样的，它们都是要动态生成 HTML。在服务器端生成内容可以解决发布问题。不过，与使用胖客户或 applet 所做的工作相比，用户从原始 HTML 得到的体验就太过单调和苍白了。下面几节将介绍几种力图提供更丰富用户体验的解决方案。

1.3.5 Flash

并不是只有 Microsoft 和 Sun 在努力寻找办法来解决动态 Web 页面问题。1996 年夏天，FutureWave 发布了一个名叫 FutureSplash Animator 的产品。这个产品起源于一个基于 Java 的动画播放器，FutureWave 很快被 Macromedia 兼并，Macromedia 则将这个产品改名为 Flash。

利用 Flash，设计人员可以创建令人惊叹的动态应用。公司可以在 Web 上发布高度交互性的应用，几乎与胖客户应用相差无几（见图 1-4）。不同于 applet、servlet 和 CGI 脚本，Flash 不需要编程技巧，很容易上手。在 20 世纪 90 年代末期，掌握 Flash 是一个很重要的特长，因为许多老板都非常需要有这种技能的员工。不过，这种易用性也是有代价的。

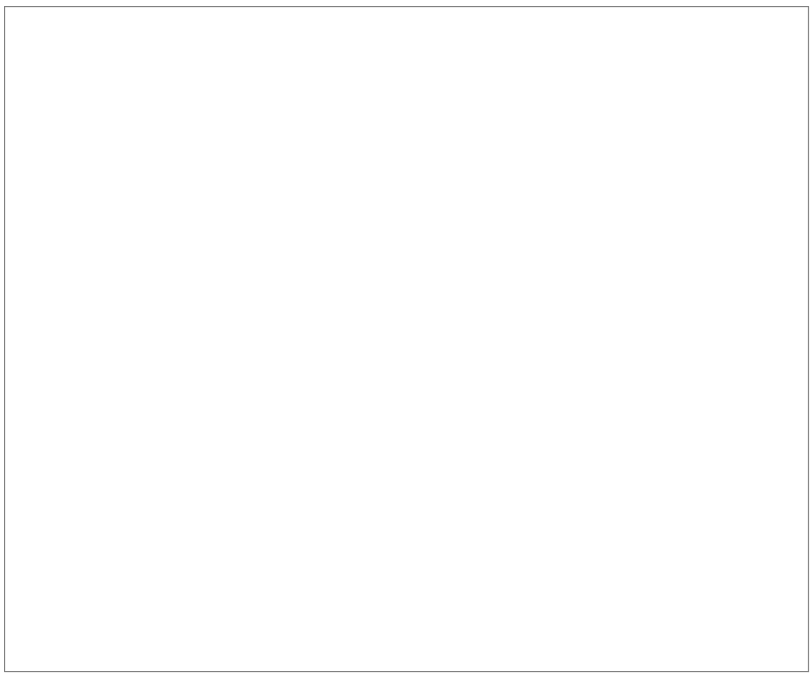


图 1-4　Flash 应用

像许多解决方案一样，Flash 需要客户端软件。尽管许多流行的操作系统和浏览器上都内置有所需的 Shockwave 播放器插件，但并非普遍都有。虽然能免费下载，但由于担心感染病毒，使得许多用户都拒绝安装这个软件。Flash 应用可能还需要大量网络带宽才能正常地工作，另外，由于没有广泛的宽带连接，Flash 的推广受到局限（因此产生了“跳过本页”之类的链接）。虽然确有一些网站选择建立多个版本的 Web 应用，分别适应于不同的连接速度，但是许多公司都无法承受支持两个或更多网站所增加的开发开销。

总之，创建 Flash 应用需要专用的软件和浏览器插件。applet 可以用文本编辑器编写，而且有一个免费的 Java 开发包，Flash 则不同，使用完整的 Flash 工具包需要按用户数付费，每个用户需要数百美元。尽管这些因素不是难以逾越的障碍，但它们确实减慢了 Flash 在动态 Web 应用道路上的前进脚步。

1.3.6 DHTML 革命

当 Microsoft 和 Netscape 发布其各自浏览器的第 4 版时，Web 开发人员有了一个新的选择：动态 HTML（Dynamic HTML，DHTML）。与有些人想像的不同 DHTML 不是一个 W3C 标准，它更像是一种营销手段。实际上，DHTML 结合了 HTML、层叠样式表（Cascading Style Sheets，CSS）、JavaScript 和 DOM。这些技术的结合使得开发人员可以动态地修改 Web 页面的内容和结构。

最初 DHTML 的反响很好。不过，它需要的浏览器版本还没有得到广泛采用。尽管 IE 和 Netscape 都支持 DHTML，但是它们的实现大相径庭，这要求开发人员必须知道他们的客户使用什么浏览器。而这通常意味着需要大量代码来检查浏览器的类型和版本，这就进一步增加了开发的开销。有些人对于尝试这种方法很是迟疑，因为 DHTML 还没有一个官方的标准。不过，将来新标准有可能会出现。

1.3.7 XML 衍生语言

20 世纪 90 年代中期，基于 SGML 衍生出了 W3C 的可扩展标记语言（eXtensible Markup Language，XML），自此以后，XML 变得极为流行。许多人把 XML 视为解决所有计算机开发问题的灵丹妙药，以至于 XML 几乎无处不在。实际上，Microsoft 就已经宣布，Office 12 将支持 XML 文件格式。

如今，我们至少有 4 种 XML 衍生语言可以用来创建 Web 应用（W3C 的 XHTML 不包括在内）：Mozilla 的 XUL；XAMJ，这是结合 Java 的一种开源语言；Macromedia 的 MXML；Microsoft 的 XAML。

XUL：XUL（读作“zool”）代表 XML 用户界面语言（XML User Interface Language），由 Mozilla 基金会推出。流行的 Firefox 浏览器和 Thunderbird 邮件客户端都是用 XUL 编写的。利用 XUL，开发人员能构建功能很丰富的应用，这个应用可以与因特网连接，也可以不与因特网连接。为了方便那些熟悉 DHTML 的开发人员使用，XUL 设计为可以跨平台支持诸如窗口和按钮等标准界面部件。虽然 XUL 本身不是标准，但它是基于各种标准的，如 HTML 4.0、CSS、DOM、XML 和 ECMAScript 等等。XUL 应用可以在浏览器上运行，也可以安装在客户端主机上。

当然，XUL 也不是没有缺点。它需要 Gecko 引擎，而且目前 IE 还没有相应的插件。尽管 Firefox 在浏览器市场中已经有了一定的份额，但少了 IE 的支持还是影响很大，大多数应用都无法使用 XUL。目前开展的很多项目都是力图在多个平台上使用 XUL，包括 Eclipse。

XAML：XAML（读作“zammel”）是 Microsoft 即将推出的操作系统（名为 Windows Vista）的一个组件。XAML 是可扩展应用标记语言（eXtensible Application Markup Language）的缩写，它为使用 Vista 创建用户界面定义了标准。与 HTML 类似，XAML 使用标记来创建标准元素，如按钮和文本框等。XAML 建立在 Microsoft 的 .NET 平台之上，而且可以编译为 .NET 类。

XAML 的局限应当很清楚。作为 Microsoft 的产品，它要求必须使用 Microsoft 的操作系统。多数情况下特别是在大公司中，这可能不成问题，但是有些小公司使用的不是 Microsoft 的操作系统，总不能削足适履吧，就像是没有哪家公司会因为买家没有开某种牌子的车来就把他拒之门外。Vista 交付的日期一再推迟，与此同时 XAML 也有了很大变化，不再只是一个播放器。据说，在未来几年内，我们可能会看到一个全新的 XAML。

MXML：Macromedia 创建了 MXML，作为与其 Flex 技术一同使用的一种标记语言。MXML 是最佳体验标记语言（Maximum eXperience Markup Language）的缩写，它与 HTML 很相似，可以以声明的方式来设计界面。与 XUL 和 XAML 类似，MXML 提供了更丰富的界面组件，如 DataGrid 和 TabNavigator，利用这些组件可以创建功能丰富的因特网应用。不过，MXML 不能独立使用，它依赖于 Flex 和 ActionScript 编程语言来编写业务逻辑。

MXML 与 Flash 有同样的一些限制。它是专用的，而且依赖于价格昂贵的开发和部署环境。尽管将来.NET 可能会对 MXML 提供支持，但现在 Flex 只能在 J2EE 应用服务器上运行，如 Tomcat 和 IBM 的 WebSphere，这就进一步限制了 MXML 的广泛采用。

XAMJ：让人欣喜的是，开源社区又向有关界面设计的 XML 衍生语言领域增加了新的成员。XAMJ 作为另一种跨平台的语言，为 Web 应用开发人员又提供了一个工具。这种衍生语言基于 Java，而 Java 是当前最流行的面向对象语言之一，XAMJ 也因此获得了面向对象语言的强大功能。XAMJ 实际上想要替代基于 XAML 或 HTML 的应用，力图寻找一种更为安全的方法，既不依赖于某种特定的框架，也不需要高速的因特网连接。

XAMJ 是一种编译型语言，建立在“clientlet”（小客户端）体系结构之上，尽管基于 XAMJ 的程序也可以是独立的应用，但通常都是基于 Web 的应用。在撰写本书时，XAMJ 还太新，我们还没有听到太多批评的声音。不过，批评是肯定会有的，让我们拭目以待。

当谈到“以 X 开头的东西”时，别忘了 W3C XForms 规约。XForms 设计为支持更丰富的用户界面，而且能够将数据与表示解耦合。毋庸置疑，XForms 数据是 XML，这样你就能使用现有的 XML 技术，如 XPath 和 XML Schema。标准 HTML 能做的，XForms 都能做，而且 XForms 还有更多功能，包括动态检查域值、与 Web 服务集成等等。不同于其他的许多 W3C 规约，XForms 不需要新的浏览器，你可以使用现在已有的许多浏览器去实现。与大多数 XML 衍生语言一样，XForms 是一种全新的方法，所以它要得到采纳尚需时日。

1.3.8 基本问题

有了以上了解，你怎么想？即使是要求最苛刻的客户应用，也已经把 Web 作为首选平台。很显然，基于 Web 的应用很容易部署，这种低门槛正是 Web 应 用最耀眼的地方。由于浏览器无处不在，而且无需下载和安装新的软件，用户利用基于浏览器的客户端就能很轻松地尝试新的应用。用户只需点击一个链接就能运行 你的应用程序，而不用先下载几兆比特的安装程序才行。基于浏览器的应用也不考虑操作系统是什么，也就是说，不仅使用不同操作系统（如 Linux 和 Mac OS X）的人能运行你的应用程序，而且你也不必考虑针对不同的操作系统开发和维护多个安装包。

既然基于 Web 的应用是前所未有的好东西，那我们为什么还要写这本书？如果回头看看因特网的起源就可以知道，最初因特网实际上就是为了科学家们和学术机构间交换文章和研究成果，这是一种简单的请求/响应模式。那时不需要会话状态，也不需要购物车，人们只是在交换文档。但现在你有很多办法来创建动态的 Web 应用，如果想让应用真正深入人心，赢得大量的用户，就必须在浏览器上大做文章，这说明，因特网以请求/响应模式作为基础，由此带来的同步性对你造成了妨碍。

与 Microsoft Word 或 Intuit Quicken 之类的胖客户应用相比，Web 模型当然只能根据一般用户需要做折中考虑。不过，由于 Web 应用很容易部署，而且浏览器的发展相当迅速，这意味着大多数用户都已经学会了适应。但是，还是有许多人认为 Web 应用只能算“二等公民”，给人的用户体验不是太好。因为因特网是一个同步的请求/响应系统，所以浏览器中的整个页面会进行刷新。最初，这种简单的请求并没有什么问题。如果用户做了一两处修改，就必须向服务器发回整个文档，而且要重新绘制整个页面。尽管这样是可行的，但是这种完全刷新的局限，意味着应用确实还很粗糙。

这并不是说开发人员只是袖手旁观，全然接受这种状况。Microsoft 对于交互式应用有一定了解，而且对于这种标准请求/响应模式的限制一直都不满意，因此提出了远程脚本（remote scripting）的概念。远程脚本看似神奇，其实很简单：它允许开发人员创建以异步方式与服务器交互的页面。例如，顾客可以从下拉列表中选择状态，这样就会在服务器上运行一个脚本，计算顾客的运费。更重要的是，显示这些运费时无需刷新整个面！当然，Microsoft 的方案只适用于它自己的技术，而且需要 Java，但有了这个进步，说明更丰富的浏览器应用并不是海市蜃楼。

对于同步页面刷新问题还有其他一些解决方案。针对 Microsoft 的远程脚本，Brent Ashley 在创建 JavaScript 远程脚本（JavaScript Remote Scripting，JSRS）时开发了一个平台中立（独立于平台）的方案。JSRS 依赖于一个客户端 JavaScript 库和 DHTML，可以向服务

器做异步的调用。与此同时，许多人利用了 IFRAME 标记，可以只加载页面中的某些部分，或者向服务器做“隐藏”的调用。尽管这是一个可行的方法，而且也为很多人所用，但它肯定不是最理想的，还有待改善。

1.3.9 Ajax

终于谈到这里了：客户希望得到一个功能更完备的应用，而开发人员想避开繁琐的部署工作，不想把可执行文件逐个地部署到数以千计的工作站上。我们已经做过很多尝试，但是任何方法都不像它原来标榜的那么完美。不过，最近一个极其强大的工具横空出世了。

是的，我们又有了一个新的选择，新的工具，可以创建的确丰富的基于浏览器的应用。这就是 Ajax。Ajax 不只是一个特定的技术，更应算是一种技巧，不过前面提到的 JavaScript 是其主要组件。我们知道，你可能会说“JavaScript 根本不值一提”，但是由于 Ajax 的出现，人们对这种语言又有了新的兴趣，应用和测试框架再加上更优秀的工具支持，减轻了开发人员肩头的重担。随着 Atlas 的引入，Microsoft 对 Ajax 投入了大力支持，而名声不太好的 Rails Web 框架也预置了充分的 Ajax 支持。在 Java 世界中，Sun 已经在其 BluePrints Solutions Catalog 中增加了许多 Ajax 组件。

坦率地讲，Ajax 并不是什么新鲜玩艺。实际上，与这个词相关的“最新”术语就是 XMLHttpRequest 对象（XHR），它早在 IE 5（于 1999 年春天发布）中就已经出现了，是作为 Active X 控件露面的。不过，最近出现的新现象是浏览器的支持。原先，XHR 对象只在 IE 中得到支持（因此限制了它的使用），但是从 Mozilla 1.0 和 Safari 1.2 开始，对 XHR 对象的支持开始普及。这个很少使用的对象和相关的基本概念甚至已经出现在 W3C 标准中：DOM Level 3 加载和保存规约（DOM Level 3 Load and Save Specification）。现在，特别是随着 Google Maps、Google Suggest、Gmail、Flickr、Netflix 和 A9 等应用变得越来越炙手可热，XHR 也已经成为事实上的标准。

与前面几页提到的方法不同，Ajax 在大多数现代浏览器中都能使用，而且不需要任何专门的软件或硬件。实际上，这种方法的一大优势就是开发人员不需要学习一种新的语言，也不必完全丢掉他们原先掌握的服务器端技术。Ajax 是一种客户端方法，可以与 J2EE、.NET、PHP、Ruby 和 CGI 脚本交互，它并不关心服务器是什么。尽管存在一些很小的安全限制，你还是可以现在就开始使用 Ajax，而且能充分利用你原有的知识。

你可能会问：“谁在使用 Ajax？”前面已经提到，Google 显然是最早采用 Ajax 的公司之一，而且已经用在很多技术上，随便说几个应用，如 Google Maps、Google Suggest 和 Gmail。Yahoo!也开始引入 Ajax 控件，另外 Amazon 提供了一个简洁的搜索工具，其中大量使用了这个技术，例如，在钻石的某一方面上移动滑块，这会带来动态更新的结果（见图 1-5）。并不是每次改变你的查询标准时都会更新页面，而是在移动滑块时就会查询服务器，从而更快、更容易地缩小你的选择范围。

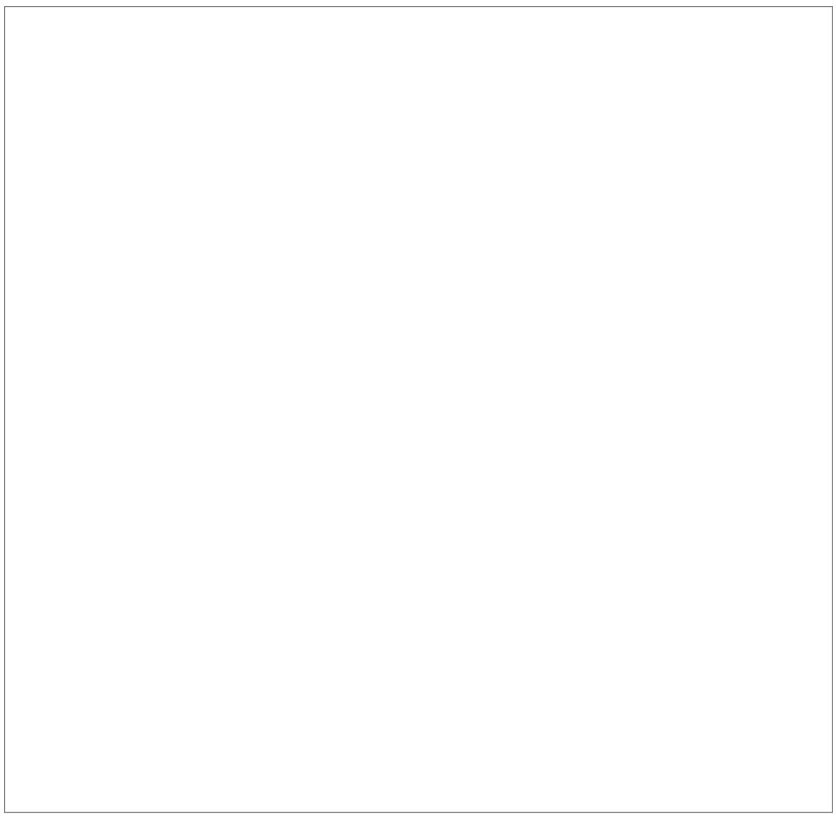


图 1-5　Amazon 的钻石搜索页面

Netflix 是一家很有名的 DVD 租借公司，它也使用了 Ajax。当用户浏览影片时，可以得到更详细的信息。当顾客把鼠标放在一个影片的图片上时，这个影片的 ID 就会发送到中心服务器，然后会出现一个“气泡”，提供这个影片的更多细节（见图 1-6）。同样，页面并不会刷新，每个影片的详细信息并不是放在隐藏的表单域中。利用这种方法，Netflix 可以提供影片的更多信息，而不会把页面弄乱。顾客浏览起来也更容易，他们不必点击影片，看完影片详细信息后再点击回到影片列表页面；他们只需把鼠标停在影片上，就这么简单！我们想要强调的是，Ajax 并不限于.com 之类的网站使用，普通公司的开发人员也开始涉足这个技术，有些人已经在使用 Ajax 来改善原来很丑陋的验证方案，或者用于动态地获取数据了。

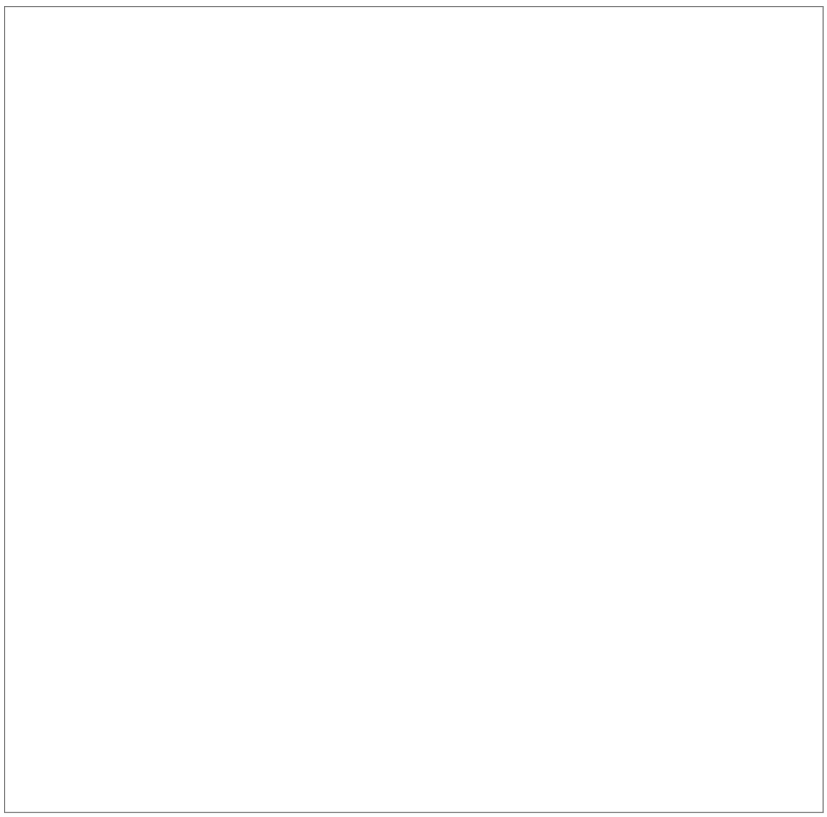


图 1-6　Netflix 的浏览页面特性

关键在于，因特网默认的请求/响应模式有了重大转变，这正是 Ajax 的核心所在，尽管这并非全新的内容。Web 应 用开发人员现在可以自由地与服务器异步交互，这说明他们可以完成许多原本只能在胖客户上完成的任务。例如，当用户输入邮政编码时，可以验证它是否正确，然 后自动用相应的城市名和州名填充到表单中的其他部分；或者，当用户选择美国时，可以引入美国各个州的一个下拉列表。以前也可以用其他方式模拟这些工作，但 是使用 Ajax 的话，这些工作会更加简单。

那么，是谁发明了 Ajax？要找出真正的源头，总免不了一场争论。不过有一点是确定的，2005 年 2 月，Adaptive Path 的 Jesse James Garrett 最早创造了这个词。在他的文章 *Ajax: A New Approach to Web Applications*（Ajax：Web 应用的一种新方法）中，Garrett 讨论了如何消除胖客户（或桌面）应用与瘦客户（或 Web）应用之间的界限。当然，当 Google 在 Google Labs 发布 Google Maps 和 Google Suggest 时，这个技术才真正为人所认识，而且此前已经有许多这方面的文章了。但确实是 Garrett 最早提出了这个好名字，否则我们就得啰啰嗦嗦地说上一大堆：异步（Asynchronous）、XMLHttpRequest、JavaScript、CSS、DOM 等等。尽管原来把 Ajax 认为是 Asynchronous JavaScript + XML（异步 JavaScript + XML）的缩写，但如今，这个词的覆盖面有所扩展，把允许浏览器与服务器通信而无需刷新当前页面的技术都涵盖在内。

你可能会说：“哦，那有什么大不了的？”这么说吧，使用 XHR 而且与服务器异步通信，就能创建更加动态的 Web 应用。例如，假设你有一个下拉列表，它是根据另外一个域或下拉列表的输入来填写的。在正常情况下，必须在加载第一个页面时把所有数据都发送给客户端，然后使用 JavaScript 根据输入来填写下拉列表。这么做并不困难，但是会让页面变得很臃肿，取决于这个下拉列表到底有多“动态”，页面有可能膨胀得过大，从而出现问题。利用 Ajax，当作为触发源的域有变化，或者失去了输入焦点，就可以向服务器发一个简单的请求，只要求得到更新下拉列表所需的部分信息即可。

来单独考虑一下验证。你写过多少次 JavaScript 验证逻辑？用 Java 或 C#编写验证逻辑可能很简单，但是由于 JavaScript 缺乏很好的调试工具，再加上它是一种弱类型语言，所以用 JavaScript 编写验证逻辑实在是一件让人头疼的事情，而且很容易出错。服务器上还很有可能重复这些客户端验证规则。使用 XHR，可以对服务器做一个调用，触发某一组验证规则。这些规则可能比你用 JavaScript 编写的任何规则都更丰富、更复杂，而且你还能得到功能强大的调试工具和集成开发环境（IDE）。

你现在可能又会说：“这些事情我早已经用 IFRAME 或隐藏框架做到了。”我们甚至还使用这种技术来提交或刷新过页面的一部分，而不是整个浏览器（页面）。不能不承认，这确实可行。不过，许多人认为这种方法只是一种修补手段，以弥补 XHR 原来缺乏对跨浏览器的支持。作为 Ajax 的核心，XHR 对象设计为允许从服务器异步地获取任意的数据。

我们讨论过，传统的 Web 应用遵循一种请求/响应模式。如果没有 Ajax，对于每个请求都会重新加载整个页面（或者利用 IFRAME，则是部分页面）。原来查看的页面会放到浏览器的历史栈中（不过，如果使用了 IFRAME，点击“后退”按钮不一定能得到用户期望的历史页面）。与此不同，用 XHR 做出的请求不会记录在浏览器的历史中。如果你的用户习惯于使用“后退”按钮在 Web 应用中进行导航，就可能会产生问题。

1.4 可用性问题

前面谈到的都是用户的期望，除此以外，可用性也不能不提。Ajax 方法相当新，还没有多少成熟的最佳实践。不过，标准 Web 设计原则还是适用的。随着时间推移，当越来越多的人开始尝试这种方法时，就会发现可能存在哪些限制，并建立适当的指导原则。也就是说，你应该让用户来指导你。根据在应用中使用 Ajax 的方式，你可能会动态地改变页面中的某些部分，习惯于整个浏览器刷新的用户可能不会注意到与以前相比有什么变化。这个问题引出了一些新的特性，如 37signals 所普及的黄褪技术（Yellow Fade Technique，YFT），这个特性已经用在 Ajax 的招牌应用 Basecamp 中了。

基本说来，YFT 是指“取页面中有变化的部分，并置为黄色”。假设你的应用原本没有大量使用黄色，用户就很可能会注意到这种改变。过一段时间后，再让黄色逐渐褪色，直到恢复为原来的背景色。当然，你也可以选用你喜欢的其他颜色，只要能把用户的注意力吸引到有变化的部分。

可能 YTF 并不适用于你的应用，你也可以选择用一种不那么张扬但仍很有用的方式来提醒用户。Gmail 在右上角显示了一个闪动的红色“Loading”加载记号，提醒用户正在获取数据（见图 1-7）。

图 1-7　Gmail 的“Loading”记号

究竟要使用 YFT 还是其他类似的技术，实际上取决于你的用户。最简单的方法是让一组用户代表来进行测试。可以通过文字问卷，也可以使用基于 Web 的原型应用，这要看你处在设计过程的哪个阶段。但是不论如何测试，在真正采用 Ajax 完成复杂设计之前都应该取得一些用户反馈。

而且要从小处做起。在刚开始使用 Ajax 时，不应该马上就创建一个可调整列的动态门户网站，而是应该先试着处理客户端验证，逐步转向服务器端。待有所了解后，可以再尝试更

动态的使用，如填写一个下拉列表，或者设置某些默认文本。

不管你要如何应用 Ajax，记住别做稀奇古怪的事情。我们知道，这不算是一个学术性的建议。不过，目前这方面还没有严格的规则。先听听用户怎么说，部署之前一定要先做测试，而且要记住，如果太过古怪，用户很快就会点击“跳过本页”链接跳过你精心设计的这些部分。

要知道使用 Ajax 时有几个常犯的错误。我们已经讨论过，有变化时如何向用户提供可视化的提示，不仅如此，Ajax 还会以其他方式改变标准的 Web 方法。首先，不同于 IFRAME 和隐藏框架，通过 XHR 做出请求不会修改浏览器的历史栈。在许多情况下这没有什么问题（你可能会点击后退箭头，只是要看看是不是什么都没有改变，但这么做能有几次呢？），不过，如果你的用户确实想用后退按钮，就有问题了。

其次，与其他基于浏览器的方法不同，Ajax 不会修改地址栏中显示的链接，这表明你不能轻松地为一个页面建立书签，或者向朋友发送一个链接。对于许多应用来说，可能没有这个要求，但是如果你的网站专门为人提供行车路线之类的东西，就要针对这个问题提供一个解决方案。

有一点很重要，使用 Ajax 不要过度。记住，JavaScript 会在客户端的浏览器上运行，如果有数千行 JavaScript 代码，可能会让用户感觉速度太慢。如果脚本编写不当，就会很快失去控制，特别是当通信量增加时。

Ajax 允许你异步地完成操作，这个最大的优点同时也是它最突出的缺点。我们以前总是告诉用户，Web 应用是以一种请求/响应模式完成操作的，用户也已经接受了这种思想。但是用了 Ajax，就不再有这个限制。我们可以只修改页面的一部分，如果用户没想到这一点，他们很可能会被搞糊涂。所以，你要注意一定要让用户明白这一点，不要想当然地以为他们知道。记住，只要有疑问，就要请用户代表进行测试！

1.5 相关技术

当你看到本书时，可能已经了解了在应用中实现 Ajax 所需的大多数技术。重申一句，我们想强调的是，Ajax 是一个客户端技术，不论你现在使用何种服务器端技术，都能使用 Ajax，而不管使用的是 Java、.NET、Ruby、PHP 还是 CGI。实际上，在这本书中我们并不考虑服务器端，而且假设你已经很清楚如何结合日常工作中使用的服务器端技术。在后面的几百页中，我们强调的重点是客户端技术和方法，创建丰富的基于浏览器的应用时需要用到这些技术。

尽管可以使用你喜欢的任何服务器端技术，但当使用 Ajax 时还是需要转变一下思想。在一般的 Web 应用中，服务器端代码会呈现一个完整的页面，并涉及一个完整的工作单元。利用 Ajax，可能只返回一点点文本，而且只涉及一个业务应用的很小子集。对于大多数有经验的 Web 开发人员来说，理解起来没有什么问题，但是一定要记住这一点。

一些新兴的框架有助于开发人员跳出 Ajax 的一些细节。不过，你还是要对 JavaScript 有所了解。我们知道，JavaScript 用起来可能很费劲。但很遗憾，对此没有什么办法。我们大多数人都学过这么一招，把“alert”作为一种系统类型输出来帮助调试，糟糕的是，这种技术使用得还很广。不过，现在我们有了新的利器。

除了 JavaScript，你还要熟悉其他一些与表示相关的技术，如 HTML、DOM 和 CSS。你不必是这方面的专家，但是基本了解还是必要的。本书中我们会谈到你需要知道的大多数内

容，没有谈到的内容可以参考网上的资源。

关于测试驱动（你肯定写过单元测试，对不对？），我们会介绍 JsUnit 和 Selenium（见图 1-8）。利用这些工具，可以先开发 JavaScript 测试，并检查浏览器兼容性测试。通常认为，下一代开发环境会对 JavaScript 提供更好的支持，另外一些与 Ajax 相关的技术会进一步减轻开发人员的负担。正在不断出现的脚本和框架也会使开发变得更为简单。

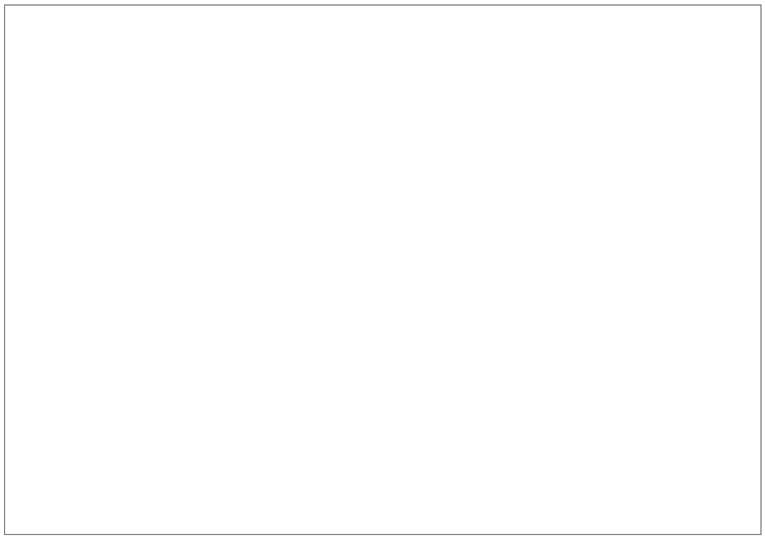


图 1-8　Selenium

1.6 使用场合

既然你已经对 Ajax 产生了兴趣，还要知道重要的一点，即什么时候应该使用 Ajax 技术，而什么时候不该用。首先，不要害怕在应用中尝试新的方法。我们相信，几乎每个 Web 应用都能从 Ajax 技术中获益，只不过不要矫枉过正，过于离谱就行了。从验证开始就很合适，但是不要限制你的主动性。你当然可以使用 Ajax 提交数据，但也许不能把它作为提交数据的主要方法。

其次，惟一会影响你应用 Ajax 的就是浏览器问题。如果大量用户（或者特别重要的用户）还在使用比较旧的浏览器，如 IE 5、Safari 1.2 或 Mozilla 1.0 之前的版本，Ajax 技术就不能奏效。如果这是一些很重要的用户，你就要使用针对目标用户的跨浏览器的方法，而放弃 Ajax，或者开发一个可以妥善降级的网站。浏览器支持可能不是一个重要因素，因为 Netscape Navigator 4 在市场上的份额很小。不过，还是应该查看 Web 日志，看看你的应用适用什么技术。

如前所述，验证和表单填写就非常适合采用 Ajax 实现。还可以使用 DOM 的“拖”技术建立真正动态的网站，如 Google 的个性化主页（见图 1-9）。

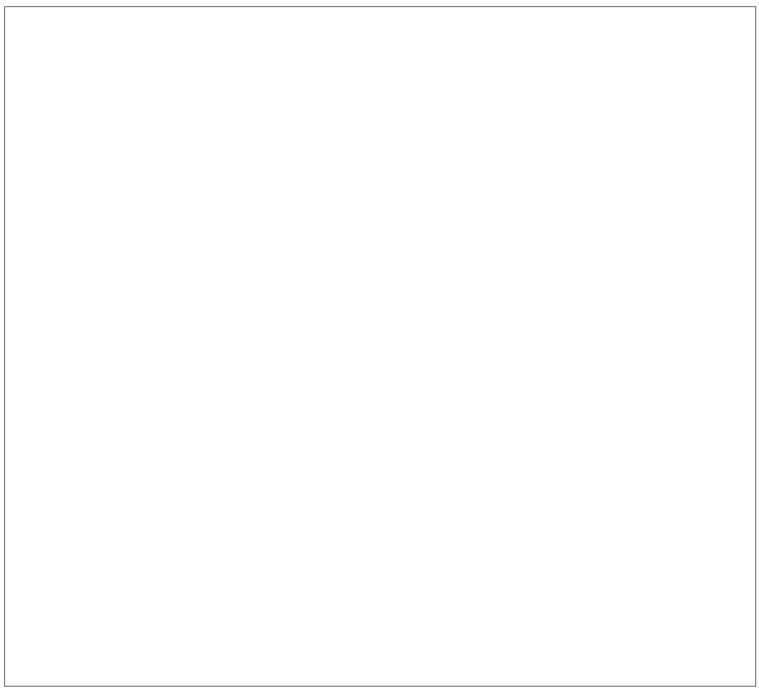


图 1-9　Google 的个性化主页

可以看到，Ajax 为 Web 应用开发提供了新的机会。你不会再因为以往的专用技术或技术折中方案而受到妨碍。利用 Ajax，胖客户与瘦客户之间的界限不再分明，真正的赢家则是你的用户。

1.7 设计考虑

既然对在哪里使用 Ajax 已经有所认识，下面再来谈谈应用 Ajax 的一些设计考虑。许多原则与 Web 应用的原则并无不同，不过还是有必要强调一下。要尽力减少客户和服务器之间的通信量。如果应用得当，Ajax 会使你的应用响应更快，但是如果每次用户从一个域移到另一个域时你都来回传递超量的数据，用户肯定不会满意。如果有疑问，按标准约定行事。如果大多数应用都那么做，可能你也应该那么做。如果还有问题，可以看看 Web 桌面应用的有关标准。为此已经建立了一些模式，而且以后还会有更多的模式（www.ajaxpatterns.org）。

在刚开始使用 Ajax 时，你的用户可能不清楚应用的工作机理的。多年来我们一直在告诉用户：Web 是以某种（同步）方式工作的，而 Ajax 则增加了异步组件，可能与之背道而驰。简单地说，不要让用户觉得奇怪。当用户用跳格键离开最后一个域时，如果以前的应用（没有使用 Ajax 的应用）没有保存表单，那么使用 Ajax 之后的应用也不要保存表单。

实现 Ajax 时最重要的问题是要力求简单，完全从用户出发，要尽量“傻瓜化”。要把用户放在心上，不要去做“简历驱动的设计”[4]。如果只是想让新老板接受你，并因此在应用中使用 Ajax，这是不合适的；如果使用 Ajax 能让你的用户有更丰富的体验，那就义无反顾地使用 Ajax 吧。但是别忘了，你会做，并不意味着你应该做。要理智一些，先考虑你的用户才对。

我们后面还会更多地谈到安全，但是这里需要先说明一点，Ajax 有一些安全考虑。记住，可以在浏览器中查看源代码，这说明任何人都能知道你是怎么创建小部件的。建立 XHR 对象时必须包含统一资源定位符（uniform resource locators，URL），所以可能会有恶意用户

修改你的网站，运行他们自己的代码。谨慎地使用 Ajax 可以降低这种风险。

1.8 小结

因 特网最初只是为连接研究人员，使他们共享信息，时至今日，因特网已经得到了巨大的发展。因特网开始时只有简单的文本浏览器和静态页面，但是如今几乎每家公 司都有一个亮丽的网站，想找到一个粗糙的网站倒是很不容易。最早谁能想得到，有一天人们能在网上共同研究新型汽车，或者购买最新的斯蒂芬·金的小说呢？

胖客户应用的开发人员都饱受部署之苦，因为要把应用部署到数以千计的用户机器上，他们急切地希望 Web 能够减轻他们的负担。多年以来，已经出现了许多 Web 应用技术，有些是专用的，有些需要高超的编程能力。尽管这些技术在用户体验方面各有千秋，但没有哪个技术能使瘦客户应用达到桌面应用的水平。不过，由于很容易部署，有更大的客户群体，而且维护开销更低，这说明尽管浏览器存在一定的局限性，但仍是许多应用的首选目标平台。

开发人员可以使用一些技巧来绕过因特网的麻烦限制。利用各种远程脚本方法和 HTML 元素，开发人员可以与服务器异步地通信，但是直到有主流浏览器对 XMLHttpRequest 对象提供了支持，真正的跨浏览器方法才有可能。Google、Yahoo 和 Amazon 等公司已经走在前面，我们终于看到基于浏览器的应用也能与胖客户应用不相上下。利用 Ajax，你可以尽享这两方面的好处：代码位于你能控制的服务器上，而且只要客户有浏览器就能访问一个能提供丰富用户体验的应用。

2.1 XMLHttpRequest 对象概述

在使用 XMLHttpRequest 对象发送请求和处理响应之前，必须先用 JavaScript 创建一个 XMLHttpRequest 对象。由于 XMLHttpRequest 不是一个 W3C 标准，所以可以采用多种方法使用 JavaScript 来创建 XMLHttpRequest 的实例。Internet Explorer 把 XMLHttpRequest 实现为一个 ActiveX 对象，其他浏览器（如 Firefox、Safari 和 Opera）把它实现为一个本地 JavaScript 对象。由于存在这些差别，JavaScript 代码中必须包含有关的逻辑，从而使用 ActiveX 技术或者使用本地 JavaScript 对象技术来创建 XMLHttpRequest 的一个实例。

很多人可能还记得从前的那段日子，那时不同浏览器上的 JavaScript 和 DOM 实现简直千差万别，听了上面这段话之后，这些人可能又会不寒而栗。幸运的是，在这里为了明确该如何创建 XMLHttpRequest 对象的实例，并不需要那么详细地编写代码来区别浏览器类型。你要做的只是检查浏览器是否提供对 ActiveX 对象的支持。如果浏览器支持 ActiveX 对象，就可以使用 ActiveX 来创建 XMLHttpRequest 对象。否则，就要使用本地 JavaScript 对象技术来创建。代码清单 2-1 展示了编写跨浏览器的 JavaScript 代码来创建 XMLHttpRequest 对象实例是多么简单。

代码清单 2-1　创建 XMLHttpRequest 对象的一个实例

```
var xmlHttp;


function createXMLHttpRequest() {
    if (window.ActiveXObject) {
        xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
```

```
    }
    else if (window.XMLHttpRequest) {
        xmlHttp = new XMLHttpRequest();
    }
}
```

可以看到，创建 XMLHttpRequest 对象相当容易。首先，要创建一个全局作用域变量 `xmlHttp` 来保存这个对象的引用。`createXMLHttpRequest` 方法完成创建 XMLHttpRequest 实例的具体工作。这个方法中只有简单的分支逻辑（选择逻辑）来确定如何创建对象。对 `window.ActiveXObject` 的调用会返回一个对象，也可能返回 `null`，`if` 语句会把调用返回的结果看作是 true 或 false（如果返回对象则为 true，返回 `null` 则为 false），以此指示浏览器是否支持 ActiveX 控件，相应地得知浏览器是不是 Internet Explorer。如果确实是，则通过实例化 ActiveXObject 的一个新实例来创建 XMLHttpRequest 对象，并传入一个串指示要创建何种类型的 ActiveX 对象。在这个例子中，为构造函数提供的字符串是 `Microsoft.XMLHTTP`，这说明你想创建 XMLHttpRequest 的一个实例。

如果 `window.ActiveXObject` 调用失败（返回 `null`），JavaScript 就会转到 `else` 语句分支，确定浏览器是否把 XMLHttpRequest 实现为一个本地 JavaScript 对象。如果存在 `window.XMLHttpRequest`，就会创建 XMLHttpRequest 的一个实例。

由于 JavaScript 具有动态类型特性，而且 XMLHttpRequest 在不同浏览器上的实现是兼容的，所以可以用同样的方式访问 XMLHttpRequest 实例的属性和方法，而不论这个实例创建的方法是什么。这就大大简化了开发过程，而且在 JavaScript 中也不必编写特定于浏览器的逻辑。

2.2 方法和属性

表 2-1 显示了 XMLHttpRequest 对象的一些典型方法。不要担心，稍后就会详细介绍这些方法。

表 2-1　标准 XMLHttpRequest 操作

方　　法	描　　述
`abort()`	停止当前请求
`getAllResponseHeaders()`	把 HTTP 请求的所有响应首部作为键/值对返回
`getResponseHeader("header")`	返回指定首部的串值
`open("method", "url")`	建立对服务器的调用。`method` 参数可以是 `GET`、`POST` 或 `PUT`。`url` 参数可以是相对 URL 或绝对 URL。这个方法还包括 3 个可选的参数
`send(content)`	向服务器发送请求
`setRequestHeader("header", "value")`	把指定首部设置为所提供的值。在设置任何首部之前必须先调用 `open()`

下面来更详细地讨论这些方法。

`void open(string method, string url, boolean asynch, string username, string password)`：这个方法会建立对服务器的调用。这是初始化一个请求的纯脚本方

法。它有两个必要的参数，还有 3 个可选参数。要提供调用的特定方法（GET、POST 或 PUT），还要提供所调用资源的 URL。另外还可以传递一个 Boolean 值，指示这个调用是异步的还是同步的。默认值为 true，表示请求本质上是异步的。如果这个参数为 false，处理就会等待，直到从服务器返回响应为止。由于异步调用是使用 Ajax 的主要优势之一，所以倘若将这个参数设置为 false，从某种程度上讲与使用 XMLHttpRequest 对象的初衷不太相符。不过，前面已经说过，在某些情况下这个参数设置为 false 也是有用的，比如在持久存储页面之前可以先验证用户的输入。最后两个参数不说自明，允许你指定一个特定的用户名和密码。

void send(content)：这个方法具体向服务器发出请求。如果请求声明为异步的，这个方法就会立即返回，否则它会等待直到接收到响应为止。可选参数可以是 DOM 对象的实例、输入流，或者串。传入这个方法的内容会作为请求体的一部分发送。

void setRequestHeader(string header, string value)：这个方法为 HTTP 请求中一个给定的首部设置值。它有两个参数，第一个串表示要设置的首部，第二个串表示要在首部中放置的值。需要说明，这个方法必须在调用 open()之后才能调用。

在所有这些方法中，最有可能用到的就是 open()和 send()。XMLHttpRequest 对象还有许多属性，在设计 Ajax 交互时这些属性非常有用。

void abort()：顾名思义，这个方法就是要停止请求。

string getAllResponseHeaders()：这个方法的核心功能对 Web 应用开发人员应该很熟悉了，它返回一个串，其中包含 HTTP 请求的所有响应首部，首部包括 Content-Length、Date 和 URI。

string getResponseHeader(string header)：这个方法与 getAllResponseHeaders()是对应的，不过它有一个参数表示你希望得到的指定首部值，并且把这个值作为串返回。

除了这些标准方法，XMLHttpRequest 对象还提供了许多属性，如表 2-2 所示。处理 XMLHttpRequest 时可以大量使用这些属性。

表 2-2　标准 XMLHttpRequest 属性

属　　性	描　　述
onreadystatechange	每个状态改变时都会触发这个事件处理器，通常会调用一个 JavaScript 函数
readyState	请求的状态。有 5 个可取值：0 = 未初始化，1 = 正在加载，2 = 已加载，3 = 交互中，4 = 完成
responseText	服务器的响应，表示为一个串
responseXML	服务器的响应，表示为 XML。这个对象可以解析为一个 DOM 对象
status	服务器的 HTTP 状态码（200 对应 OK，404 对应 Not Found（未找到），等等）
statusText	HTTP 状态码的相应文本（OK 或 Not Found（未找到）等等）

2.3 交互示例

看到这里，你可能想知道典型的 Ajax 交互是什么样。图 2-1 显示了 Ajax 应用中标准的交互模式。

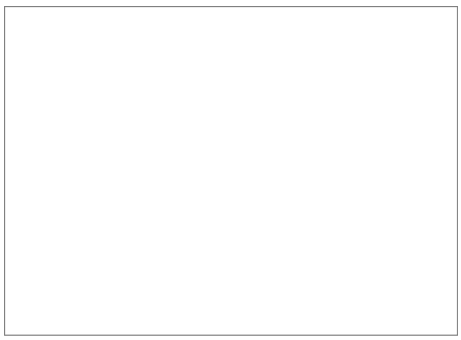


图 2-1　标准 Ajax 交互

不同于标准 Web 客户中所用的标准请求/响应方法，Ajax 应用的做法稍有差别。

1. 一个客户端事件触发一个 Ajax 事件。从简单的 `onchange` 事件到某个特定的用户动作，很多这样的事件都可以触发 Ajax 事件。可以有如下的代码：

   <input type="text"d="email" name="email" onblur="validateEmail()";>

2. 创建 XMLHttpRequest 对象的一个实例。使用 `open()` 方法建立调用，并设置 URL 以及所希望的 HTTP 方法（通常是 `GET` 或 `POST`）。请求实际上通过一个 `send()` 方法调用触发。可能的代码如下所示：

```
var xmlHttp;
function validateEmail() {
    var email = document.getElementById("email");
    var url = "validate?email=" + escape(email.value);
    if (window.ActiveXObject) {
       xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
    }
    else if (window.XMLHttpRequest) {
       xmlHttp = new XMLHttpRequest();
    }
xmlHttp.open("GET", url);
xmlHttp.onreadystatechange = callback;
xmlHttp.send(null);
}
```

3. 向服务器做出请求。可能调用 servlet、CGI 脚本，或者任何服务器端技术。
4. 服务器可以做你想做的事情，包括访问数据库，甚至访问另一个系统。
5. 请求返回到浏览器。`Content-Type` 设置为 text/xml——XMLHttpRequest 对象只能处理 text/html 类型的结果。在另外一些更复杂示例中，响应可能涉及更广，还包括 JavaScript、DOM 管理以及其他相关的技术。需要说明，你还需要设置另外一些首部，使浏览器不会在本地缓存结果。为此可以使用下面的代码：

```
response.setHeader("Cache-Control", "no-cache");
```

response.setHeader("Pragma", "no-cache");[1]

6. 在这个示例中，XMLHttpRequest 对象配置为处理返回时要调用 `callback()` 函数。这个函数会检查 XMLHttpRequest 对象的 `readyState` 属性，然后查看服务器返回的状态码。如果一切正常，`callback()` 函数就会在客户端上做些有意思的工作。以下就是一个典型的回调方法：

```
function callback() {
    if (xmlHttp.readyState == 4) {
        if (xmlHttp.status == 200) {
            //do something interesting here
        }
    }
}
```

可以看到，这与正常的请求/响应模式有所不同，但对 Web 开发人员来说，并不是完全陌生的。显然，在创建和建立 XMLHttpRequest 对象时还可以做些事情，另外当“回调”函数完成了状态检查之后也可以有所作为。一般地，你会把这些标准调用包装在一个库中，以便在整个应用中使用，或者可以使用 Web 上提供的库。这个领域还很新，但是在开源社区中已经如火如荼地展开了大量的工作。

通常，Web 上 提供的各种框架和工具包负责基本的连接和浏览器抽象，有些还增加了用户界面组件。有一些纯粹基于客户，还有一些需要在服务器上工作。这些框架中的很多只是 刚开始开发，或者还处于发布的早期阶段，随着新的库和新的版本的定期出现，情况还在不断发生变化。这个领域正在日渐成熟，最具优势的将脱颖而出。一些比较 成熟的库包括 libXmlRequest、RSLite、sarissa、JavaScript 对象注解（JavaScript Object Notation，JSON）、JSRS、直接 Web 远程通信（Direct Web Remoting，DWR）和 Rails on Ruby。这个领域日新月异，所以应当适当地配置你的 RSS 收集器，及时收集有关 Ajax 的所有网站上的信息！

2.4 GET 与 POST

你可能想了解 `GET` 和 `POST` 之间有什么区别，并想知道什么时候使用它们。从理论上讲，如果请求是幂等的就可以使用 `GET`，所谓幂等是指多个请求返回相同的结果。实际上，相应的服务器方法可能会以某种方式修改状态，所以一般情况下这是不成立的。这只是一种标准。更实际的区别在于净荷的大小，在许多情况下，浏览器和服务器会限制 URL 的长度 URL 用于向服务器发送数据。一般来讲，可以使用 `GET` 从服务器获取数据；换句话说，要避免使用 `GET` 调用改变服务器上的状态。

一般地，当改变服务器上的状态时应当使用 `POST` 方法。不同于 `GET`，需要设置 XMLHttpRequest 对象的 `Content-Type` 首部，如下所示：

```
xmlHttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
```

与 `GET` 不同，`POST` 不会限制发送给服务器的净荷的大小，而且 `POST` 请求不能保证是幂等的。

你做的大多数请求可能都是 GET 请求，不过，如果需要，也完全可以使用 POST。

2.5 远程脚本

我们已经介绍了 Ajax，下面来简单谈谈远程脚本。你可能会想：“Ajax 有什么大不了的？我早就用 IFRAME 做过同样的事情。”实际上，我们自己也曾用过这种方法。这在以前一般称为□程脚本（remote scripting），很多人认为这只是一种修修补补。不过，这确实提供了一种能避免页面刷新的机制。

2.5.1 远程脚本概述

基本说来，远程脚本是一种远程过程调用类型。你可以像正常的 Web 应用一样与服务器交互，但是不用刷新整个页面。与 Ajax 类似，你可以调用任何服务器端技术来接收请求、处理请求并返回一个有意义的结果。正如在服务器端有很多选择，客户端同样有许多实现远程脚本的选择。你可以在应用中嵌入 Flash 动画、Java applet，或者 ActiveX 组件，甚至可以使用 XML-RPC，但是这种方法过于复杂，因此除非你使用这种技术很有经验，否则这种方法不太合适。实现远程脚本的通常做法包括将脚本与一个 IFRAME（隐藏或不隐藏）结合，以及由服务器返回 JavaScript，然后再在浏览器中运行这个 JavaScript。

Microsoft 提供了自己的远程脚本解决方案，并聪明地称之为 Microsoft 远程脚本（Microsoft Remote Scripting，MSRS）。采用这种方法，可以像调用本地脚本一样调用服务器脚本。页面中嵌入 Java applet，以便与服务器通信，.asp 页面用于放置服务器端脚本，并用.htm 文件管理客户端的布局摆放。在 Netscape 和 IE 4.0 及更高版本中都可以使用 Microsoft 的这种解决方案，可以同步调用，也可以异步调用。不过，这种解决方案需要 Java，这意味着可能还需要附加的安装例程，而且还需要 Internet Information Services（IIS），因此会限制服务器端的选择。

Brent Ashley 为远程脚本创建了两个免费的跨平台库。JSRS 是一个客户端 JavaScript 库，它充分利用 DHTML 向服务器做远程调用。相当多的操作系统和浏览器上都能使用 JSRS。如果采用一些常用的、流行的服务器端实现（如 PHP、Python 和 Perl CGI），JSRS 一般都能在网站上安装并运行。Ashley 免费提供了 JSRS，而且还可以从他的网站（www.ashleyit.com /rs/main.htm）上得到源代码。

如果你觉得 JSRS 太过笨重，Ashley 还创建了 RSLite，这个库使用了 cookie。RSLite 仅限于少量数据和单一调用，不过大多数浏览器都能提供支持。

2.5.2 远程脚本的示例

为了进行比较，这里向你展示如何使用 IFRAME 来实现类似 Ajax 的技术。这非常简单，而且过去我们就用过这种方法（在 XMLHttpRequest 问世之前）。这个示例并没有真正调用服务器，只是想让你对如何使用 IFRAME 实现远程脚本有所认识。

这个示例包括两个文件：iframe.html（见代码清单 2-2）和 server.html（见代码清单 2-3）。server.html 模拟了本应从服务器返回的响应。

代码清单 2-2　iframe.html 文件

```
<html>
    <head>
       <title>Example of remote scripting in an IFRAME</title>
    </head>
    <script type="text/javascript">
       function handleResponse() {
          alert('this function is called from server.html');
       }
    </script>
    <body>
    <h1>Remote Scripting with an IFRAME</h1>

    <iframe id="beforexhr"
    name="beforexhr"
    style="width:0px; height:0px; border: 0px"
    src="blank.html"></iframe>

    <a href="server.html" target="beforexhr">call the server</a>

    </body>
</html>
```

代码清单 2-3　server.html 文件

```
<html>
    <head>
       <title>the server</title>
    </head>
    <script type="text/javascript">
       window.parent.handleResponse();
    </script>
    <body>
    </body>
</html>
```

图 2-2 显示了最初的页面。运行这个代码生成的结果如图 2-3 所示。

图 2-2　最初的页面

图 2-3　调用“服务器”之后的页面

2.6 如何发送简单请求

现在已经准备开始使用 XMLHttpRequest 对象了。我们刚刚讨论了如何创建这个对象，下面来看如何向服务器发送请求，以及如何处理服务器的响应。

最简单的请求是，不以查询参数或提交表单数据的形式向服务器发送任何信息。在实际中，往往都希望向服务器发送一些信息。

使用 XMLHttpRequest 对象发送请求的基本步骤如下：

1. 为得到 XMLHttpRequest 对象实例的一个引用，可以创建一个新的实例，也可以访问包含有 XMLHttpRequest 实例的一个变量。
2. 告诉 XMLHttpRequest 对象，哪个函数会处理 XMLHttpRequest 对象状态的改变，为此要把对象的 `onreadystatechange` 属性设置为指向 JavaScript 函数的指针。
3. 指定请求的属性。XMLHttpRequest 对象的 `open()` 方法会指定将发出的请求。`open()` 方法取 3 个参数：一个是指示所用方法（通常是 `GET` 或 `POST`）的串；一个是表示目标资源 URL 的串；一个是 Boolean 值，指示请求是否是异步的。
4. 将请求发送给服务器。XMLHttpRequest 对象的 `send()` 方法把请求发送到指定的目标资源。`send()` 方法接受一个参数，通常是一个串或一个 DOM 对象。这个参数作为请求体的一部分发送到目标 URL。当向 `send()` 方法提供参数时，要确保 `open()` 中指定的方法是 `POST`。如果没有数据作为请求体的一部分被发送，则使用 `null`。

这些步骤很直观：你需要 XMLHttpRequest 对象的一个实例，要告诉它如果状态有变化该怎么做，还要告诉它向哪里发送请求以及如何发送请求，最后还需要指导

XMLHttpRequest 发送请求。不过，除非你对 C 或 C++很了解，否则可能不明白函数指（function pointer）是什么意思。

函数指针与任何其他变量类似，只不过它指向的不是像串、数字、甚至对象实例之类的数据，而是指向一个函数。在 JavaScript 中，所有函数在内存中都编有地址，可以使用函数名引用。这就提供了很大的灵活性，可以把函数指针作为参数传递给其他函数，或者在一个对象的属性中存储函数指针。

对于 XMLHttpRequest 对象，`onreadystatechange` 属性存储了回调函数的指针。当 XMLHttpRequest 对象的内部状态发生变化时，就会调用这个回调函数。当进行了异步调用，请求就会发出，脚本立即继续处理（在脚本继续工作之前，不必等待请求结束）。一旦发出了请求，对象的 `readyState` 属性会经过几个变化。尽管针对任何状态都可以做一些处理，不过你最感兴趣的状态可能是服务器响应结束时的状态。通过设置回调函数，就可以有效地告诉 XMLHttpRequest 对象：“只要响应到来，就调用这个函数来处理响应。”

2.6.1 简单请求的示例

第一个示例很简单。这是一个很小的 HTML 页面，只有一个按钮。点击这个按钮会初始化一个发至服务器的异步请求。服务器将发回一个简单的静态文本文件作为响应。在处理这个响应时，会在一个警告窗口中显示该静态文本文件的内容。代码清单 2-4 显示了这个 HTML 页面和相关的 JavaScript。

代码清单 2-4　`simpleRequest.html` 页面

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Simple XMLHttpRequest</title>
<script type="text/javascript">
var xmlHttp;

function createXMLHttpRequest() {
      if (window.ActiveXObject) {
            xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
      }
      else if (window.XMLHttpRequest) {
            xmlHttp = new XMLHttpRequest();
      }
}

function startRequest() {
```

```
        createXMLHttpRequest();
        xmlHttp.onreadystatechange = handleStateChange;
        xmlHttp.open("GET", "simpleResponse.xml", true);
        xmlHttp.send(null);
}

function handleStateChange() {
        if(xmlHttp.readyState == 4) {
                if(xmlHttp.status == 200) {
                        alert("The server replied with: " + xmlHttp.responseText);
                }
        }
}
</script>
</head>

<body>
        <form action="#">
                <input type="button" value="Start Basic Asynchronous Request"
                        onclick="startRequest();"/>
        </form>
</body>
</html>
```

服务器的响应文件 `simpleResponse.xml` 只有一行文本。点击 HTML 页面上的按钮会生成一个警告框，其中显示 `simpleResponse.xml` 文件的内容。在图 2-4 中可以看到分别在 Internet Explorer 和 Firefox 中显示的包含服务器响应的相同警告框。

对 服务器的请求是异步发送的，因此浏览器可以继续响应用户输入，同时在后台等待服务器的响应。如果选择同步操作，而且倘若服务器的响应要花几秒才能到达，浏 览器就会表现得很迟钝，在等待期间不能响应用户的输入。这样一来，浏览器好像被冻住一样，无法响应用户输入，而异步做法可以避免这种情况，从而让最终用户 有更好的体验。尽管这种改善很细微，但确实很有意义。这样用户就能继续工作，而且服务器会在后台处理先前的请求。

图 2-4　第一个简单的异步请求

与 服务器通信而不打断用户的使用流程，这种能力使开发人员采用多种技术改善用户体验成为可能。例如，假设有一个验证用户输入的应用。用户在输入表单上填写各 个字段时，浏览器可以定期地向服务器发送表单值来进行验证，此时并不打断用户，他还可以继续填写余下的表单字段。如果某个验证规则失败，在表单真正发送到 服务器进行处理之前，用户就会立即得到通知，这就能大大节省用户的时间，也能减轻服务器上的负载，因为不必在表单提交不成功时完全重建表单的内容。

2.6.2 关于安全

如果讨论基于浏览器的技术时没有提到安全，那么讨论就是不完整的。XMLHttpRequest 对象要受制于浏览器的安全“沙箱”。XMLHttpRequest 对象请求的所有资源都必须与调用脚本在同一个域内。这个安全限制使得 XMLHttpRequest 对象不能请求脚本所在域之外的资源。

这个安全限制的强度因浏览器而异（见图 2-5）。IE 会显示一个警告，指出可能存在一个潜在的安全风险，但是用户可以选择是否继续发出请求。Firefox 则会断然停止请求，并在 JavaScript 控制台显示一个错误消息。

Firefox 确实提供了一些 JavaScript 技巧，使得 XMLHttpRequest 也可以请求外部 URL 的资源。不过，由于这些技术针对特定的浏览器，所以最好不要用，而且要避免使用 XMLHttpRequest 访问外部 URL。

2.7 DOM Level 3 加载和保存规约

到目前为止，我们讨论的解决方案都不是标准。尽管 XMLHttpRequest 得到了广泛支持，但是你已经看到了，创建 XMLHttpRequest 对象的过程会随浏览器不同而有所差异。许多人错误地认为 Ajax 得到了 W3C 的支持，但实际上并非如此。W3C 在一个新标准中解决了这一问题以及其他缺点，这个标准的名字相当长：DOM Level 3 加载和保存规约。这个规约的设计目的是以一种独立于平台和语言的方式，用 XML 内容修改 DOM 文档的内容。2004 年 4 月提出了 1.0 版本，但到目前为止，还没有浏览器实现这个规约。

什么时候加载和保存规约能取代 Ajax？谁也不知道。想想看有多少浏览器没有完全支持现有的标准，所以这很难说，但是随着越来越多的网站和应用利用了 Ajax 技术，可能以后的版本会得到支持。不过，较早的 DOM 版本就花了很长时间才得到采纳，所以你得耐心一点。在一次访谈中，DOM Activity 主席 Philippe Le Hégaret 称，需要花“相当长的时间”才

能得到广泛采纳。DOM Level 3 也得到了一些支持，Opera 的 XMLHttpRequest 实现就基于 DOM Level 3，而且 Java XML 处理 API（Java API for XML Processing，JAXP）1.3 版本也支持 DOM Level 3。不过，从出现了相应的 W3C 规约这一点来看，起码可以表明 Ajax 技术的重要性。

从 1997 年 8 月起，人们就一直在为解决浏览器之间的不兼容而努力，加载和保存规约则达到了极致。你可能注意到，标题里写的是“Level 3”，那么 Level 1 和 Level 2 呢？Level 1 在 1998 年 10 月完成，为我们带来了 HTML 4.0 和 XML 1.0。如今，Level 1 已经得到了广泛支持。2000 年 11 月，Level 2 完成，不过它被采纳得比较慢。CSS 就是 Level 2 的一部分。

开发人员能从加载和保存规约得到些什么？在理想情况下，它能解决我们目前遇到的许多跨浏览器问题。尽管 Ajax 很简单，但是你应该记得，仅仅是为了创建 XMLHttpRequest 对象的一个实例，就需要检查浏览器的类型。真正的 W3C 规约可以减少这种编写代码的工作。基本说来，加载和保存规约会为 Web 开发人员提供一个公共的 API，可以以一种独立于平台和语言的方式来访问和修改 DOM。换句话说，不论你的平台是 Windows 还是 Linux，也不论你用 VBScript 开发还是用 JavaScript 开发，都没有关系。还可以把 DOM 树保存为一个 XML 文档，或者将一个 XML 文档加载到 DOM。另外，规约还提供了对 XML 1.1、XML Schema 1.0 和 SOAP 1.2 的支持。这个规约很可能得到开发人员的广泛使用。

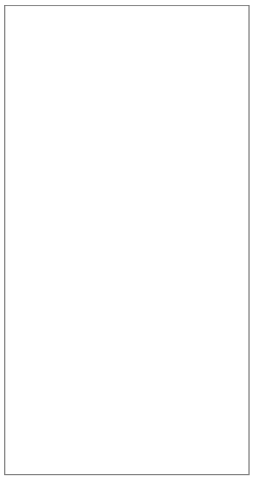
2.8 DOM

我们一直在说 DOM，如果你没有做过太多客户端的工作，可能不知道什么是 DOM。DOM 是一个 W3C 规约，可以以一种独立于平台和语言的方式访问和修改一个文档的内容和结构。换句话说，这是表示和处理一个 HTML 或 XML 文档的常用方法。

有一点很重要，DOM 的设计是以对象管理组织（OMG）的规约为基础的，因此可以用于任何编程语言。最初人们把它认为是一种让 JavaScript 在浏览器间可移植的方法，不过 DOM 的应用已经远远超出这个范围。

DOM 实际上是以面向对象方式描述的对象模型。DOM 定义了表示和修改文档所需的对象、这些对象的行为和属性以及这些对象之间的关系。可以把 DOM 认为是页面上数据和结构的一个树形表示，不过页面当然可能并不是以这种树的方式具体实现。假设有一个 Web 页面，如代码清单 2-5 所示。

代码清单 2-5　简单的表格



```
<table>
    <tbody>
        <tr>
            <td>Foo</td>
            <td>Bar</td>
        </tr>
    </tbody>
</table>
```

可以画出这个简单表格的 DOM，如图 2-6 所示。

DOM 规约好就好在它提供了一种与文档交互的标准方法。如果没有 DOM，Ajax 最有意思的方面也许根本就没有存在的可能。由于 DOM 不仅允许遍历 DOM 树，还可以编辑内容，因此可以建立极为动态的页面。

2.9 小结

尽管 Ajax 风格的技术已经用了很多年，但直到最近 XMLHttpRequest 对象才得到现代浏览器的采纳，而这也为开发丰富的 Web 应用开启了一个新的时代。在本章中，我们讨论了 Ajax 核心（即 XMLHttpRequest 对象）的相关基础知识。我们了解了 XMLHttpRequest 对象的方法和属性，而且展示了使用 XMLHttpRequest 对象的简单示例。可以看到，这个对象相当简单，无需你考虑其中很多的复杂性。适当地使用 JavaScript，再加上基本的 DOM 管理，Ajax 可以提供高度的交互性，而这在此前的 Web 上是做不到的。

第 1 章曾提到，利用 XMLHttpRequest，你不必将整个页面完全刷新，也不限于只能与服务器进行同步会话。在后面的几章中，我们会介绍如何将你已经掌握的服务器端技术与 XMLHttpRequest 的独特功能相结合，来提供高度交互性的 Web 应用

3.1 处理服务器响应

XMLHttpRequest 对象提供了两个可以用来访问服务器响应的属性。第一个属性 `responseText` 将响应提供为一个串，第二个属性 `responseXML` 将响应提供为一个 XML 对象。一些简单的用例就很适合按简单文本来获取响应，如将响应显示在警告框中，或者响应只是指示成功还是失败的词。

第 2 章中的例子就使用了 `responseText` 属性来访问服务器响应，并将响应显示在警告框中。

3.1.1 使用 **innerHTML** 属性创建动态内容

如果将服务器响应作为简单文本来访问，则灵活性欠佳。简单文本没有结构，很难用 JavaScript 进行逻辑性的表述，而且要想动态地生成页面内容也很困难。

如果结合使用 HTML 元素的 `innerHTML` 属性，`responseText` 属性就会变得非常有用。`innerHTML` 属性是一个非标准的属性，最早在 IE 中实现，后来也为其他许多流行的浏览器所采用。这是一个简单的串，表示一组开始标记和结束标记之间的内容。

通过结合使用 `responseText` 和 `innerHTML`，服务器就能“生产”或生成 HTML 内容，由浏览器使用 `innerHTML` 属性来“消费”或处理。下面的例子展示了一个搜索功能，这是使用 XMLHttpRequest 对象、其 `responseText` 属性和 HTML 元素的 `innerHTML` 属性实现的。点击 search（搜索）按钮将在服务器上启动“搜索”，服务器将生成一个结果表作为响应。浏览器处理响应时将 `div` 元素的 `innerHTML` 属性设置为 XMLHttpRequest 对象的 `response-`

Text 属性值。图 3-1 显示了点击 search 按钮而且在窗口内容中增加了结果表之后的浏览器窗口。

第 2 章的例子只是将服务器响应显示在警告框中，这个例子的代码与它很相似。具体步骤如下：

1. 点击 search 按钮，调用 startRequest 函数，它先调用 createXMLHttpRequest 函数来初始化 XMLHttpRequest 对象的一个新实例；
2. startRequest 函数将回调函数设置为 handleStateChange 函数；
3. startRequest 函数使用 open()方法来设置请求方法（GET）及请求目标，并且设置为异步地完成请求；
4. 使用 XMLHttpRequest 对象的 send()方法发送请求；
5. XMLHttpRequest 对象的内部状态每次有变化时，都会调用 handleStateChange 函数。一旦接收到响应（如果 readyState 属性的值为 4），div 元素的 innerHTML 属性就将使用 XMLHttpRequest 对象的 responseText 属性设置。

代码清单 3-1 显示了 innerHTML.html。代码清单 3-2 显示了 innerHTML.xml，表示搜索生成的内容。

代码清单 3-1　innerHTML.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
 <html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Using responseText with innerHTML</title>

<script type="text/javascript">
var xmlHttp;
function createXMLHttpRequest() {
    if (window.ActiveXObject) {
        xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
    }
    else if (window.XMLHttpRequest) {
        xmlHttp = new XMLHttpRequest();
    }
}

function startRequest() {
    createXMLHttpRequest();
    xmlHttp.onreadystatechange = handleStateChange;
```

```
    xmlHttp.open("GET", "innerHTML.xml", true);
    xmlHttp.send(null);
}

function handleStateChange() {
    if(xmlHttp.readyState == 4) {
        if(xmlHttp.status == 200) {
            document.getElementById("results").innerHTML = xmlHttp.responseText;
        }
    }
}
</script>
</head>

<body>
    <form action="#">
        <input type="button" value="Search for Today's Activities"
                onclick="startRequest();"/>
    </form>
    <div id="results"></div>
</body>
</html>
```

代码清单 3-2 `innerHTML.xml`

```
<table border="1">
    <tbody>
        <tr>
            <th>Activity Name</th>
            <th>Location</th>
            <th>Time</th>
        </tr>
        <tr>
            <td>Waterskiing</td>
            <td>Dock #1</td>
            <td>9:00 AM</td>
```

```
        </tr>
        <tr>
            <td>Volleyball</td>
            <td>East Court</td>
            <td>2:00 PM</td>
        </tr>
        <tr>
            <td>Hiking</td>
            <td>Trail 3</td>
            <td>3:30 PM</td>
        </tr>
    </tbody>
</table>
```

使用 `responseText` 和 `innerHTML` 可以大大简化向页面增加动态内容的工作。遗憾的是，这种方法存在一些缺陷。前面已经提到，`innerHTML` 属性不是 HTML 元素的标准属性，所以与标准兼容的浏览器不一定提供这个属性的实现。不过，当前大多数浏览器都支持 `innerHTML` 属性。可笑的是，IE 是率先使用 `innerHTML` 的浏览器，但它的 `innerHTML` 实现反而最受限制。如今许多浏览器都将 `innerHTML` 属性作为所有 HTML 元素的读/写属性。与此不同，IE 则有所限制，在表和表行之类的 HTML 元素上 `innerHTML` 属性仅仅是只读属性，从一定程度上讲，这就限制了它的用途。

3.1.2 将响应解析为 XML

你已经了解到，服务器不一定按 XML 格式发送响应。只要 `Content-Type` 响应首部正确地设置为 text/plain（如果是 XML，Content-Type 响应首部则是 text/xml），将响应作为简单文本发送是完全可以的。复杂的数据结构就很适合以 XML 格式发送。对于导航 XML 文档以及修改 XML 文档的结构和内容，当前浏览器已经提供了很好的支持。

浏览器到底怎么处理服务器返回的 XML 呢？当前浏览器把 XML 看作是遵循 W3C DOM 的 XML 文档。W3C DOM 指定了一组很丰富的 API，可用于搜索和处理 XML 文档。DOM 兼容的浏览器必须实现这些 API，而且不允许有自定义的行为，这样就能尽可能地改善脚本在不同浏览器之间的可移植性。

W3C DOM

W3C DOM 到底是什么？W3C 主页提供了清晰的定义：

> 文档象模型（DOM）是与平台和言无的接口，允程序和脚本地和更新文档的内容、构和式。文档可以一　理，理的果可以放回到所提供的面中。

不仅如此，W3C 还解释了为什么要定义标准的 DOM。W3C 从其成员处收到了大量请求，这些请求都是关于将 XML 和 HTML 文档的对象模型提供给脚本所要采用的方法。提案并没有提出任何新的标记或样式表技术，而只是力图确保这些可互操作而且与脚本语言无关的解

决方案能得到共识，并为开发社区所采纳。简单地说，W3C DOM 标准的目的是尽量避免 20 世纪 90 年代末的脚本恶梦，那时相互竞争的浏览器都有自己专用的对象模型，而且通常都是不兼容的，这就使得实现跨平台的脚本极其困难。

W3C DOM 和 JavaScript

W3C DOM 和 JavaScript 很容易混淆不清。DOM 是面向 HTML 和 XML 文档的 API，为文档提供了结构化表示，并定义了如何通过脚本来访问文档结构。JavaScript 则是用于访问和处理 DOM 的语言。如果没有 DOM，JavaScript 根本没有 Web 页面和构成页面元素的概念。文档中的每个元素都是 DOM 的一部分，这就使得 JavaScript 可以访问元素的属性和方法。

DOM 独立于具体的编程语言，通常通过 JavaScript 访问 DOM，不过并不严格要求这样。可以使用任何脚本语言来访问 DOM，这要归功于其一致的 API。表 3-1 列出了 DOM 元素的一些有用的属性，表 3-2 列出了一些有用的方法。

表 3-1 用于处理 XML 文档的 DOM 元素属性

属性名	描述
childNodes	返回当前元素所有子元素的数组
firstChild	返回当前元素的第一个下级子元素
lastChild	返回当前元素的最后一个子元素
nextSibling	返回紧跟在当前元素后面的元素
nodeValue	指定表示元素值的读/写属性
parentNode	返回元素的父节点
previousSibling	返回紧邻当前元素之前的元素

表 3-2 用于遍历 XML 文档的 DOM 元素方法

方法名	描述
getElementById(id) (document)	获取有指定惟一 ID 属性值文档中的元素
getElementsByTagName(name)	返回当前元素中有指定标记名的子元素的数组
hasChildNodes()	返回一个布尔值，指示元素是否有子元素
getAttribute(name)	返回元素的属性值，属性由 name 指定

有了 W3C DOM，就能编写简单的跨浏览器脚本，从而充分利用 XML 的强大功能和灵活性，将 XML 作为浏览器和服务器之间的通信介质。

从下面的例子可以看到，使用遵循 W3C DOM 的 JavaScript 来读取 XML 文档是何等简单。代码清单 3-3 显示了服务器向浏览器返回的 XML 文档的内容。这是一个简单的美国州名列表，各个州按地区划分。

代码清单 3-3 服务器返回的美国州名列表

```
<?xml version="1.0" encoding="UTF-8"?>
<states>
    <north>
```

```
        <state>Minnesota</state>
        <state>Iowa</state>
        <state>North Dakota</state>
    </north>
    <south>
        <state>Texas</state>
        <state>Oklahoma</state>
        <state>Louisiana</state>
    </south>
    <east>
        <state>New York</state>
        <state>North Carolina</state>
        <state>Massachusetts</state>
    </east>
    <west>
        <state>California</state>
        <state>Oregon</state>
        <state>Nevada</state>
    </west>
</states>
```

在浏览器上会生成具有两个按钮的 HTML 页面。点击第一个按钮，将从服务器加载 XML 文档，然后在警告框中显示列于文档中的所有州。点击第二个按钮也会从服务器加载 XML 文档，不过只在警告框中显示北部地区的各个州（见图 3-2）。

图 3-2 点击页面上的任何一个按钮都会从服务器加载 XML 文档，并在警告框中显示适当的结果

代码清单 3-4 显示了 parseXML.html。

代码清单 3-4 parseXML.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Parsing XML Responses with the W3C DOM</title>

<script type="text/javascript">
var xmlHttp;
var requestType = "";

function createXMLHttpRequest() {
      if (window.ActiveXObject) {
            xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
      }
      else if (window.XMLHttpRequest) {
            xmlHttp = new XMLHttpRequest();
      }
}

function startRequest(requestedList) {
      requestType = requestedList;
      createXMLHttpRequest();
      xmlHttp.onreadystatechange = handleStateChange;
      xmlHttp.open("GET", "parseXML.xml", true);
      xmlHttp.send(null);
}

function handleStateChange() {
      if(xmlHttp.readyState == 4) {
            if(xmlHttp.status == 200) {
                  if(requestType == "north") {
                        listNorthStates();
```

```
                }
                else if(requestType == "all") {
                        listAllStates();
                }
        }
    }
}

function listNorthStates() {
    var xmlDoc = xmlHttp.responseXML;
    var northNode = xmlDoc.getElementsByTagName("north")[0];

    var out = "Northern States";
    var northStates = northNode.getElementsByTagName("state");

    outputList("Northern States", northStates);
}

function listAllStates() {
    var xmlDoc = xmlHttp.responseXML;
    var allStates = xmlDoc.getElementsByTagName("state");

    outputList("All States in Document", allStates);
}

function outputList(title, states) {
    var out = title;
    var currentState = null;
    for(var i = 0; i < states.length; i++) {
        currentState = states[i];
        out = out + "\n- " + currentState.childNodes[0].nodeValue;
    }
    alert(out);
}
```

```
</script>
</head>

<body>
    <h1>Process XML Document of U.S. States</h1>
    <br/><br/>
    <form action="#">
        <input type="button" value="View All Listed States"
                onclick="startRequest('all');"/>
        <br/><br/>
        <input type="button" value="View All Listed Northern States"
                onclick="startRequest('north');"/>
    </form>
</body>
</html>
```

以上脚本从服务器获取 XML 文档并加以处理，它与前面看到的例子很相似，不过前面的例子只是将响应处理为简单文本。关键区别就在于 `listNorthStates` 和 `listAllStates` 函数。前面的例子从 XMLHttpRequest 对象获取服务器响应，并使用 XMLHttpRequest 对象的 `responseText` 属性将响应获取为文本。`listNorthStates` 和 `listAllStates` 函数则不同，它们使用了 XMLHttpRequest 对象的 `responseXML` 属性，将结果获取为 XML 文档，这样一来，你就可以使用 W3C DOM 方法来遍历 XML 文档了。

仔细研究一下 `listAllStates` 函数。它首先创建了一个局部变量，名为 `xmlDoc`，并将这个变量初始化设置为服务器返回的 XML 文档，这个 XML 文档是使用 XMLHttpRequest 对象的 `responseXML` 属性得到的。利用 XML 文档的 `getElementsByTagName` 方法可以获取文档中所有标记名为 `state` 的元素。`getElementsByTagName` 方法返回了包含所有 `state` 元素的数组，这个数组将赋给名为 `allStates` 的局部变量。

从 XML 文档获取了所有 `state` 元素之后，`listAllStates` 函数调用 `outputList` 函数，并在警告框中显示这些 `state` 元素。`listAllStates` 方法将迭代处理 `state` 元素的数组，将各元素的相应州名逐个追加到一个串中，这个串最后将显示在警告框中。

有一点要特别注意，即如何从 `state` 元素获取州名。你可能认为，`state` 元素会简单地提供属性或方法来得到这个元素的文本，但并非如此。

表示州名的文本实际上是 `state` 元素的子元素。在 XML 文档中，文本本身被认为是一个节点，而且必须是另外某个元素的子元素。由于表示州名的文本实际上是 `state` 元素的子元素，所以必须先从 `state` 元素获取文本元素，再从这个文本元素得到其文本内容。

`outputList` 函数的工作就是如此。它迭代处理数组中的所有元素，将当前元素赋给 `currentState` 变量。因为表示州名的文本元素总是 `state` 元素的第一个子元素，所以可以使用 `childNodes` 属性来得到文本元素。一旦有了具体的文本元素，就可以使用 `nodeValue` 属

性返回表示州名的文本内容。

listNorthStates 函数与 listAllStates 是类似的，只不过增加了一个小技巧。你只想得到北部地区的州，而不是所有州。为此，首先使用 getElementsByTagName 方法获取 north 标记，从而获得 XML 文档中的 north 元素。因为文档只包含一个 north 元素，而且 getElementsByTagName 方法总是返回一个数组，所以要用 [0] 记法来抽出 north 元素。这是因为，在 getElementsByTagName 方法返回的数组中，north 元素处在第一个位置上（也是惟一的位置）。既然有了 north 元素，接下来调用 north 元素的 getElementsByTagName 方法，就可以得到 north 元素的 state 子元素。有了 north 元素所有 state 子元素的数组后，再使用 outputList 方法在警告框中显示这些州名。

3.1.3 使用 W3C DOM 动态编辑页面

Web 最初只是作为媒介向各处分发静态的文本文档，如今它本身已经发展为一个应用开发平台。遗留的企业系统通常通过纯文本的终端部署，或者作为客户—服务器应用部署，这些遗留系统正在被完全通过 Web 浏览器部署的系统所取代。

随着最终用户越来越习惯于使用基于 Web 的应用，他们开始有了新的要求，需要一种更丰富的用户体验。用户不再满足于完全页面刷新，即每次在页面上编辑一些数据时页面都会完全刷新。他们想立即看到结果，而不是坐等与服务器完成完整的往返通信。

你已经了解了解析服务器发送的 XML 消息是多么容易。W3C DOM 提供了一些属性和方法，使你能轻松地遍历 XML 结构，并抽取所需的数据。

前面的例子对于服务器发送的 XML 响应并没有做多少有用的事情。在警告框中显示 XML 文档的值没有太大的实际意义。你真正想做到的是让用户享有丰富的客户体验，不再遭遇一般 Web 应用中常见的连续页面刷新问题。页面连续刷新不仅使用户不满意，还会浪费服务器上宝贵的处理器时间，因为页面刷新需要重新构建整个页面的内容，而且会不必要地使用网络带宽来传送刷新的页面。

当然，最好的解决办法是根据需要修改页面上已有的内容。如果页面上大多数数据没有改变，则不应刷新整个页面，只需要修改页面中信息有变化的部分。

以往，在 Web 浏览器的限制之下，这一点很难做到。浏览器只是一个工具，它解释特殊的标记（HTML），并根据一组预定的规则显示这些标记。Web 以及 Web 浏览器原来只是为了显示静态的信息，如果不以新页面的形式从服务器请求新的数据，这些信息不会改变。

除了一些例外情况，当前的浏览器都使用 W3C DOM 来表示 Web 页面的内容。这样做可以确保在不同的浏览器上 Web 页面会以同样的方式呈现，同时在不同的浏览器上，用于修改页面内容的脚本也会有相同的表现。Web 浏览器的 W3C DOM 和 JavaScript 实现越来越成熟，这大大简化了在浏览器上动态创建内容的任务。原来总是要苦心积虑地解决浏览器间的不兼容性，如今这已经不太需要。表 3-3 列出了用于动态创建内容的 DOM 属性和方法。

表 3-3 动态创建内容时所用的 W3C DOM 属性和方法

属性/方法	描述
document.createElement(tagName)	文档对象上的 createElement 方法可以创建由 tagName 指定的元素。如果以串 div 作为方法参数，就会生成一个 div 元素

document.createTextNode(text)	文档对象的 createTextNode 方法会创建一个包含静态文本的节点
<element>.appendChild(childNode)	appendChild 方法将指定的节点增加到当前元素的子节点列表（作为一个新的子节点）。例如，可以增加一个 option 元素，作为 select 元素的子节点
<element>.getAttribute(name) <element>.setAttribute(name, value)	这些方法分别获得和设置元素中 name 属性的值
<element>.insertBefore(newNode, targetNode)	这个方法将节点 newNode 作为当前元素的子节点插到 targetNode 元素前面
<element>.removeAttribute(name)	这个方法从元素中删除属性 name
<element>.removeChild(childNode)	这个方法从元素中删除子元素 childNode
<element>.replaceChild(newNode, oldNode)	这个方法将节点 oldNode 替换为节点 newNode
<element>.hasChildnodes()	这个方法返回一个布尔值，指示元素是否有子元素

关于浏览器的不兼容性

尽管当前 Web □ □ 器中 W3C DOM 和 JavaScript 的在不断改，但是存在一些特异性和不兼容性，使得用 DOM 和 JavaScript □ 行很是疼。

IE 的 W3C DOM 和 JavaScript □ □ 最受限制。2000 年初，一些称 IE 占据了整个器市 95% 的份，由于没有争力，Microsoft 决定不完全各个 Web □ 准。

□ 些特异大多都能得到解决，不做会脚本更是混乱不堪而且不合准。例如，如果使用 appendChild 将<tr>元素直接增加到<table>中，在 IE 中一行并不出，但在其他器中却会示出来。此的解决之道是，将<tr>元素增加到表的<tbody>元素中，解决法在所有器中都能正确工作。

□ 于 setAttribute 方法，IE 也有麻。IE 不能使用 setAttribute 正确地置 class 属性。此有一个跨器的解决方法，即同使用 setAttribute("class", "new- ClassName") 和 setAttribute("className","newClassName")。另外，在 IE 中不能使用 setAttribute □ 置 style 属性。最能保器兼容的技不是<element>.setA-ttribute("style, "font-weight:bold;")，而是<element>.style.cssText = "font - weight:bold;"。

本中的例子会尽可能地遵循 W3C DOM 和 JavaScript □ 准，不如果必确保大多数当前器的兼容性，可能也会稍稍偏离准。

下面的例子展示了如何使用 W3C DOM 和 JavaScript 来动态创建内容。这个例子是假想的房地产清单搜索引擎，点击表单上的 Search（搜索）按钮，会使用 XMLHttpRequest 对象以 XML 格式获取结果。使用 JavaScript 处理响应 XML，从而生成一个表，其中列出搜索到的结果（见图 3-3）。

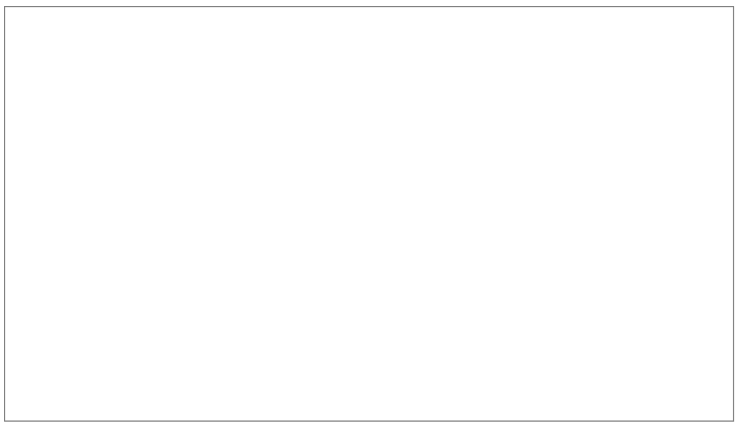


图 3-3 使用 W3C DOM 方法和 JavaScript 动态创建搜索结果

服务器返回的 XML 很简单（见代码清单 3-5）。根节点 `properties` 包含了得到的所有 `property` 元素。每个 `property` 元素包含 3 个子元素：`address`、`price` 和 `comments`。

代码清单 3-5 `dynamicContent.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<properties>
    <property>
        <address>812 Gwyn Ave</address>
        <price>$100,000</price>
        <comments>Quiet, serene neighborhood</comments>
    </property>
    <property>
        <address>3308 James Ave S</address>
        <price>$110,000</price>
        <comments>Close to schools, shopping, entertainment</comments>
    </property>
    <property>
        <address>98320 County Rd 113</address>
        <price>$115,000</price>
        <comments>Small acreage outside of town</comments>
    </property>
</properties>
```

具体向服务器发送请求并对服务器响应做出回应的 JavaScript 与前面的例子是一样的。不过，从 `handleReadyStateChange` 函数开始有所不同。假设请求成功地完成，接下来第一件事就是调用 `clearPreviousResults` 函数，将以前搜索所创建的内容删除。

clearPreviousResults 函数完成两个任务：删除出现在最上面的“Results”标题文本，并从结果表中清除所有行。首先使用 hasChildNodes 方法查看可能包括标题文本的 span 元素是否有子元素。应该知道，只有 hasChildNodes 方法返回 true 时才存在标题文本。如果确实返回 true，则删除 span 元素的第一个（也是惟一的）子节点，因为这个子节点表示的就是标题文本。

clearPreviousResults 的下一个任务是在显示搜索结果的表中删除所有行。所有结果行都是 tbody 节点的子节点，所以先使用 document.getElementById 方法得到该 tbody 节点的引用。一旦有了 tbody 节点，只要这个 tbody 节点还有子节点（tr 元素）就进行迭代处理。每次迭代时都会从表体中删除 childNodes 集合中的第一个子节点。当表体中再没有更多的表行时，迭代结束。

搜索结果表在 parseResults 函数中建立。这个函数首先创建一个名为 results 的局部变量，这是使用 XMLHttpRequest 对象的 responseXML 属性得到的 XML 文档。

使用 getElementsByTagName 方法来获得 XML 文档中包含所有 property 元素的数组，然后将这个数组赋给局部变量 properties。一旦有了 property 元素的数组，可以迭代处理数组中的各个元素，并获得 property 的 address、price 和 comments。

```
var properties = results.getElementsByTagName("property");
for(var i = 0; i < properties.length; i++) {
    property = properties[i];
    address = property.getElementsByTagName("address")[0].firstChild.nodeValue;
    price = property.getElementsByTagName("price")[0].firstChild.nodeValue;
    comments = property.getElementsByTagName("comments")[0].firstChild.nodeValue;


    addTableRow(address, price, comments);
}
```

下面来仔细分析这个循环，因为这正是 parseResults 函数的核心。在 for 循环中，首先得到数组中的下一个元素，并把它赋给局部变量 property。接下来，对于你感兴趣的各个子元素（address、price 和 comments），分别获得它们的节点值。

请考虑 address 元素，这是 property 元素的一个子元素。首先在 property 元素上调用 getElementsByTagName 方法来得到单个 address 元素。getElementsByTagName 方法返回一个数组，不过因为你知道有且仅有一个 address 元素，所以可以使用[0]记法来引用这个元素。

沿着 XML 结构继续向下，现在有了 address 标记的引用，你需要得到它的文本内容。记住，文本实际上是父元素的一个子节点，所以可以使用 firstChild 属性来访问 address 元素的文本节点。有了文本节点后，可以引用文本节点的 nodeValue 属性来得到文本。

采用同样的办法来得到 price 和 comments 元素的值，并把各个值分别赋给局部变量 price 和 comments。再将 address、price 和 comments 传递给名为 addTableRow 的辅助函数，它会用这些结果数据具体建立一个表行。

addTableRow 函数使用 W3C DOM 方法和 JavaScript 建立一个表行。使用 document.createElement 方法创建一个 row 对象，之后，再使用名为 createCellWithText 的辅助函数分别为 address、price 和 comments 值创建一个 cell 对象。createCellWithText 函数会创建并返回一个以指定的文本作为单元格内容的 cell 对象。

createCellWithText 函数首先使用 document.createElement 方法创建一个 td 元素，然后使用 document.createTextNode 方法创建一个包含所需文本的文本节点，所得到的文本节点追加到 td 元素。这个函数再把新创建的 td 元素返回给调用函数（addTableRow）。

addTableRow 函数对 address、price 和 comments 值重复调用 createCellWithText 函数，每一次向 tr 元素追加一个新创建的 td 元素。一旦向 row（行）增加了所有 cell（单元格），这个 row 就将被增加到表的 tbody 元素中。

就这么多！你已经成功地读取了服务器返回的 XML 文档，而且动态创建了一个结果表。代码清单 3-6 显示了这个例子完整的 JavaScript 和可扩展 HTML 代码。

代码清单 3-6 dynamicContent.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Dynamically Editing Page Content</title>

<script type="text/javascript">
var xmlHttp;

function createXMLHttpRequest() {
    if (window.ActiveXObject) {
        xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
    }
    else if (window.XMLHttpRequest) {
        xmlHttp = new XMLHttpRequest();
    }
}

function doSearch() {
    createXMLHttpRequest();
    xmlHttp.onreadystatechange = handleStateChange;
    xmlHttp.open("GET", "dynamicContent.xml", true);
```

```
        xmlHttp.send(null);
}

function handleStateChange() {
        if(xmlHttp.readyState == 4) {
                if(xmlHttp.status == 200) {
                        clearPreviousResults();
                        parseResults();
                }
        }
}

function clearPreviousResults() {
        var header = document.getElementById("header");
        if(header.hasChildNodes()) {
                header.removeChild(header.childNodes[0]);
        }

        var tableBody = document.getElementById("resultsBody");
        while(tableBody.childNodes.length > 0) {
                tableBody.removeChild(tableBody.childNodes[0]);
        }
}

function parseResults() {
        var results = xmlHttp.responseXML;

        var property = null;
        var address = "";
        var price = "";
        var comments = "";

        var properties = results.getElementsByTagName("property");
        for(var i = 0; i < properties.length; i++) {
```

```
            property = properties[i];
            address = property.getElementsByTagName("address")[0].firstChild.nodeValue;
            price = property.getElementsByTagName("price")[0].firstChild.nodeValue;
            comments = property.getElementsByTagName("comments")[0]
                                                                                .firstChild.nodeV
alue;
            addTableRow(address, price, comments);
        }
        var header = document.createElement("h2");
        var headerText = document.createTextNode("Results:");
        header.appendChild(headerText);
        document.getElementById("header").appendChild(header);

        document.getElementById("resultsTable").setAttribute("border", "1");
}

function addTableRow(address, price, comments) {
        var row = document.createElement("tr");
        var cell = createCellWithText(address);
        row.appendChild(cell);

        cell = createCellWithText(price);
        row.appendChild(cell);

        cell = createCellWithText(comments);
        row.appendChild(cell);

        document.getElementById("resultsBody").appendChild(row);
}

function createCellWithText(text) {
        var cell = document.createElement("td");
        var textNode = document.createTextNode(text);
        cell.appendChild(textNode);
```

```
        return cell;
}
</script>
</head>

<body>
   <h1>Search Real Estate Listings</h1>

   <form action="#">
      Show listings from
            <select>
                  <option value="50000">$50,000</option>
                  <option value="100000">$100,000</option>
                  <option value="150000">$150,000</option>
            </select>
            to
            <select>
                  <option value="100000">$100,000</option>
                  <option value="150000">$150,000</option>
                  <option value="200000">$200,000</option>
            </select>
      <input type="button" value="Search" onclick="doSearch();"/>
   </form>
  <span id="header">

   </span>

   <table id="resultsTable" width="75%" border="0">
      <tbody id="resultsBody">
      </tbody>
   </table>
</body>
</html>
```

3.2 发送请求参数

到此为止，你已经了解了如何使用 Ajax 技 术向服务器发送请求，也知道了客户可以采用多种方法解析服务器的响应。前面的例子中只缺少一个内容，就是你尚未将任何数据作为请求的一部分发送给服务器。 在大多数情况下，向服务器发送一个请求而没有任何请求参数是没有什么意义的。如果没有请求参数，服务器就得不到上下文数据，也无法根据上下文数据为客户创 建“个性化”的响应，实际上，服务器会向每一个客户发送同样的响应。

要想充分发挥 Ajax 技术的强大功能，这要求你向服务器发送一些上下文数据。假设有一个输入表单，其中包含需要输入邮件地址的部分。根据用户输入的 ZIP 编码，可以使用 Ajax 技术预填相应的城市名。当然，要想查找 ZIP 编码对应的城市，服务器首先需要知道用户输入的 ZIP 编码。

你需要以某种方式将用户输入的 ZIP 编码值传递给服务器。幸运的是，XMLHttpRequest 对象的工作与你以往惯用的 HTTP 技术（GET 和 POST）是一样的。

GET 方法把值作为名/值对放在请求 URL 中传递。资源 URL 的最后有一个问号（?），问号后面就是名/值对。名/值对采用 name=value 的形式，各个名/值对之间用与号（&）分隔。

下面是 GET 请求的一个例子。这个请求向 localhost 服务器上的 yourApp 应用发送了两个参数：firstName 和 middleName。需要注意，资源 URL 和参数集之间用问号分隔，firstName 和 middleName 之间用与号（&）分隔：

http://localhost/yourApp?firstName=Adam&middleName=Christopher

服务器知道如何获取 URL 中的命名参数。当前大多数服务器端编程环境都提供了简单的 API，使你能很容易地访问命名参数。

采用 POST 方法向服务器发送命名参数时，与采用 GET 方法几乎是一样的。类似于 GET 方法，POST 方法会把参数编码为名/值对，形式为 name=value，每个名/值对之间也用与号（&）分隔。这两种方法的主要区别在于，POST 方法将参数串放在请求体中发送，而 GET 方法是将参数追加到 URL 中发送。

如果数据处理不改变数据模型的状态，HTML 使用规约理论上推荐采用 GET 方法，从这可以看出，获取数据时应当使用 GET 方法。如果因为存储、更新数据，或者发送了电子邮件，操作改变了数据模型的状态，这时建议使用 POST 方法。

每个方法都有各自特有的优点。由于 GET 请求的参数编码到请求 URL 中，所以可以在浏览器中为该 URL 建立书签，以后就能很容易地重新请求。不过，如果是异步请求就没有什么用。从发送到服务器的数据量来讲，POST 方法更为灵活。使用 GET 请求所能发送的数据量通常是固定的，因浏览器不同而有所差异，而 POST 方法可以发送任意量的数据。

HTML form 元素允许通过将 form 元素的 method 属性设置为 GET 或 POST 来指定所需的方法。在提交表单时，form 元素自动根据其 method 属性的规则对 input 元素的数据进行编码。XMLHttpRequest 对象没有这种内置行为。相反，要由开发人员使用 JavaScript 创建查询串，其中包含的数据要作为请求的一部分发送给服务器。不论使用的是 GET 请求还是 POST 请求，创建查询串的技术是一样的。惟一的区别是，当使用 GET 发送请求时，查询串会追加到请求 URL 中，而使用 POST 方法时，则在调用 XMLHttpRequest 对象的 send() 方法时发送查询串。

图 3-4 显示了一个示例页面，展示了如何向服务器发送请求参数。这是一个简单的输入表单，要求输入名、姓和生日。这个表单有两个按钮，每个按钮都会向服务器发送名、姓和生日数据，不过一个使用 GET 方法，另一个使用 POST 方法。服务器以回显输入数据作为响应。在浏览器在页面上打印出服务器的响应时，请求响应周期结束。

图 3-4 浏览器使用 GET 或 POST 方法发送输入数据，服务器回显输入数据作为响应

代码清单 3-7 显示了 getAndPostExample.html，代码清单 3-8 显示了向浏览器回显名、姓和生日数据的 Java servlet。

代码清单 3-7 getAndPostExample.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Sending Request Data Using GET and POST</title>

<script type="text/javascript">
var xmlHttp;

function createXMLHttpRequest() {
      if (window.ActiveXObject) {
            xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
      }
      else if (window.XMLHttpRequest) {
            xmlHttp = new XMLHttpRequest();
      }
```

```
}

function createQueryString() {
	var firstName = document.getElementById("firstName").value;
	var middleName = document.getElementById("middleName").value;
	var birthday = document.getElementById("birthday").value;

	var queryString = "firstName=" + firstName + "&middleName=" + middleName
		+ "&birthday=" + birthday;

	return queryString;
}

function doRequestUsingGET() {
	createXMLHttpRequest();

	var queryString = "GetAndPostExample?";
	queryString = queryString + createQueryString()
		+ "&timeStamp=" + new Date().getTime();
	xmlHttp.onreadystatechange = handleStateChange;
	xmlHttp.open("GET", queryString, true);
	xmlHttp.send(null);
}

function doRequestUsingPOST() {
	createXMLHttpRequest();

	var url = "GetAndPostExample?timeStamp=" + new Date().getTime();
	var queryString = createQueryString();

	xmlHttp.open("POST", url, true);
	xmlHttp.onreadystatechange = handleStateChange;
	xmlHttp.setRequestHeader("Content-Type",
				"application/x-www-form-urlencoded;");
```

```
        xmlHttp.send(queryString);
}

function handleStateChange() {
        if(xmlHttp.readyState == 4) {
                if(xmlHttp.status == 200) {
                        parseResults();
                }
        }
}

function parseResults() {
        var responseDiv = document.getElementById("serverResponse");
        if(responseDiv.hasChildNodes()) {
                responseDiv.removeChild(responseDiv.childNodes[0]);
        }
        var responseText = document.createTextNode(xmlHttp.responseText);
        responseDiv.appendChild(responseText);
}

</script>
</head>

<body>
   <h1>Enter your first name, middle name, and birthday:</h1>

   <table>
        <tbody>
                <tr>
                        <td>First name:</td>
                        <td><input type="text" id="firstName"/>
                </tr>
                <tr>
                        <td>Middle name:</td>
```

```
                <td><input type="text" id="middleName"/>
            </tr>
            <tr>
                <td>Birthday:</td>
                <td><input type="text" id="birthday"/>
            </tr>
        </tbody>

    </table>

    <form action="#">
        <input type="button" value="Send parameters using GET"
                onclick="doRequestUsingGET();"/>


        <br/><br/>
        <input type="button" value="Send parameters using POST"
                onclick="doRequestUsingPOST();"/>
    </form>

    <br/>
    <h2>Server Response:</h2>

    <div id="serverResponse"></div>

</body>
</html>
```

代码清单 3-8 向浏览器回显名、姓和生日

```
package ajaxbook.chap3;

import java.io.*;
import java.net.*;
import javax.servlet.*;
```

```
import javax.servlet.http.*;

public class GetAndPostExample extends HttpServlet {

    protected void processRequest(HttpServletRequest request,
            HttpServletResponse response, String method)
    throws ServletException, IOException {

        //Set content type of the response to text/xml
        response.setContentType("text/xml");

        //Get the user's input
        String firstName = request.getParameter("firstName");
        String middleName = request.getParameter("middleName");
        String birthday = request.getParameter("birthday");

        //Create the response text
        String responseText = "Hello " + firstName + " " + middleName
                + ". Your birthday is " + birthday + "."
                + " [Method: " + method + "]";

        //Write the response back to the browser
        PrintWriter out = response.getWriter();
        out.println(responseText);

        //Close the writer
        out.close();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        //Process the request in method processRequest
        processRequest(request, response, "GET");
    }
```

```
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        //Process the request in method processRequest
        processRequest(request, response, "POST");
    }
}
```

下面先来分析服务器端代码。这个例子使用了 Java servlet 来处理请求，不过也可以使用任何其他服务器端技术，如 PHP、CGI 或.NET。Java servlet 必须定义一个 `doGet` 方法和一个 `doPost` 方法，每个方法根据请求方法（`GET` 或 `POST`）来调用。在这个例子中，`doGet` 和 `doPost` 都调用同样的方法 `processRequest` 来处理请求。

`processRequest` 方法先把响应的内容类型设置为 text/xml，尽管在这个例子中并没有真正用到 XML。通过使用 `getParameter` 方法从 `request` 对象获得 3 个输入字段。根据名、姓和生日，以及请求方法的类型，会建立一个简单的语句。这个语句将写至响应输出流，最后响应输出流关闭。

浏览器端 JavaScript 与前面的例子同样是类似的，不过这里稍稍增加了几个技巧。这里有一个工具函数 `createQueryString` 负责将输入参数编码为查询串。`createQueryString` 函数只是获取名、姓和生日的输入值，并将它们追加为名/值对，每个名/值对之间由与号（`&`）分隔。这个函数会返回查询串，以便 `GET` 和 `POST` 操作重用。

点击 Send Parameters Using GET（使用 `GET` 方法发送参数）按钮将调用 `doRequestUsingGET` 函数。这个函数与前面例子中的许多函数一样，先调用创建 XMLHttpRequest 对象实例的函数。接下来，对输入值编码，创建查询串。

在这个例子中，请求端点是名为 `GetAndPostExample` 的 servlet。在建立查询串时，要把 `createQueryString` 函数返回的查询串与请求端点连接，中间用问号分隔。

JavaScript 仍与前面看到的类似。XMLHttpRequest 对象的 `onreadystatechange` 属性设置为要使用 `handleStateChange` 函数。`open()` 方法指定这是一个 `GET` 请求，并指定了端点 URL，在这里端点 URL 中包含有编码的参数。`send()` 方法将请求发送给服务器，`handleSta-teChange` 函数处理服务器响应。

当请求成功完成时，`handleStateChange` 函数将调用 `parseResults` 函数。`parseResults` 函数获取 `div` 元素，其中包含服务器的响应，并把它保存在局部变量 `responseDiv` 中。使用 `responseDiv` 的 `removeChild` 方法先将以前的服务器结果删除。最后，创建包含服务器响应的新文本节点，并将这个文本节点追加到 `responseDiv`。

使用 `POST` 方法而不是 `GET` 方法基本上是一样的，只是请求参数发送给服务器的方式不同。应该记得，使用 `GET` 时，名/值对会追加到目标 URL。`POST` 方法则把同样的查询串作为请求体的一部分发送。

点击 Send Parameters Using POST（使用 `POST` 方法发送参数）按钮将调用 `doRequest-`

UsingPOST 函数。类似于 doRequestUsingGET 函数，它先创建 XMLHttpRequest 对象的一个实例，脚本再创建查询串，其中包含要发送给服务器的参数。需要注意，查询串现在并不连接到目标 URL。

接下来调用 XMLHttpRequest 对象的 open() 方法，这一次指定请求方法是 POST，另外指定了没有追加名/值对的“原”目标 URL。onreadystatechange 属性设置为 handleStateChange 函数，所以响应会以与 GET 方法中相同的方式得到处理。为了确保服务器知道请求体中有请求参数，需要调用 setRequestHeader，将 Content-Type 值设置为 application/x-www-form-urlencoded。最后，调用 send() 方法，并把查询串作为参数传递给这个方法。

点击两个按钮的结果是一样的。页面上会显示一个串，其中包括指定的名、姓和生日，另外还会显示所用请求方法的类型。

为什么要把时间戳追加到目标 URL？

在某些情况下，有些器会把多个 XMLHttpRequest □ 求的果存在同一个 URL每如果　个求的响不同，就会来不好的果。把当前戳追加到 URL 的最后，就能确保 URL 的惟一性，从而避免器存果。

3.2.1 请求参数作为 XML 发送

与几年前相比，当前浏览器上 JavaScript 的兼容性有了长足的进步，已经不可同日而语，再加上越来越成熟的 JavaScript 开发工具和技术，你可以决定把 Web 浏览器作为开发平台。并不只是依赖于浏览器来看待模型—视图—控制器模式中的视图，还可以用 JavaScript 实现部分业务模型。可以使用 Ajax 技术把模型中的变化持久存储到后台服务器。如果模型放在浏览器上，模型的变化可以一齐传递到服务器，从而减少对服务器的远程调用次数，还可能提高性能。

如果只是使用一个包含名/值对的简单查询串，这可能不够健壮，不足以向服务器传递大量复杂的模型变化。更好的解决方案是将模型的变化作为 XML 发送到服务器。怎么向服务器发送 XML 呢？

可以把 XML 作为请求体的一部分发送到服务器，这与 POST 请求中将查询串作为请求体的一部分进行发送异曲同工。服务器可以从请求体读到 XML，并加以处理。

下面的例子展示了对于一个 Ajax 请求如何向服务器发送 XML。图 3-5 显示了这个页面，其中有一个简单的选择框，用户可以选择宠物的类型。这是一个相当简化的例子，但是由此可以了解如何向服务器发送 XML。

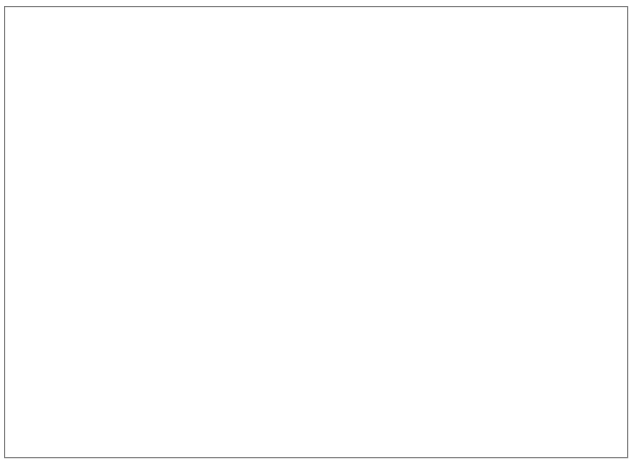


图 3-5　选择框中选中的项将作为 XML 发送到服务器

代码清单 3-9 显示了 `postingXML.html`。

代码清单 3-9 `postingXML.html`

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Sending an XML Request</title>

<script type="text/javascript">

var xmlHttp;

function createXMLHttpRequest() {
      if (window.ActiveXObject) {
            xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
      }
      else if (window.XMLHttpRequest) {
            xmlHttp = new XMLHttpRequest();
      }
}

function createXML() {
      var xml = "<pets>";
      var options = document.getElementById("petTypes").childNodes;
```

```
        var option = null;
        for(var i = 0; i < options.length; i++) {
                option = options[i];
                if(option.selected) {
                        xml = xml + "<type>" + option.value + "<\/type>";
                }
        }

        xml = xml + "<\/pets>";
        return xml;
}

function sendPetTypes() {
        createXMLHttpRequest();

        var xml = createXML();
        var url = "PostingXMLExample?timeStamp=" + new Date().getTime();

        xmlHttp.open("POST", url, true);
        xmlHttp.onreadystatechange = handleStateChange;
        xmlHttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded;");
        xmlHttp.send(xml);
}

function handleStateChange() {
        if(xmlHttp.readyState == 4) {
                if(xmlHttp.status == 200) {
                        parseResults();
                }
        }
}

function parseResults() {
        var responseDiv = document.getElementById("serverResponse");
```

```
        if(responseDiv.hasChildNodes()) {
                responseDiv.removeChild(responseDiv.childNodes[0]);
        }

        var responseText = document.createTextNode(xmlHttp.responseText);
        responseDiv.appendChild(responseText);
}


</script>
</head>
<body>
   <h1>Select the types of pets in your home:</h1>

   <form action="#">
      <select id="petTypes" size="6" multiple="true">
            <option value="cats">Cats</option>
            <option value="dogs">Dogs</option>
            <option value="fish">Fish</option>
            <option value="birds">Birds</option>
            <option value="hamsters">Hamsters</option>
            <option value="rabbits">Rabbits</option>
   </select>

   <br/><br/>
   <input type="button" value="Submit Pets" onclick="sendPetTypes();"/>
</form>

<h2>Server Response:</h2>

<div id="serverResponse"></div>

</body>
</html>
```

这个例子与前面的 POST 例子基本上是一样的。区别在于，不是发送由名/值对组成的查询串，而是向服务器发送 XML 串。

点击表单上的 Submit Pets（提交宠物）按钮将调用 sendPetTypes 函数。类似于前面的例子，这个函数首先创建 XMLHttpRequest 对象的一个实例，然后调用名为 createXML 的辅助函数，它根据所选的宠物类型建立 XML 串。

函数 createXML 使用 document.getElementbyId 方法获得 select 元素的引用，然后迭代处理所有 option 子元素，对于选中的每个 option 元素依据所选宠物类型创建 XML 标记，并逐个追加到 XML 中。循环结束时，要在返回到调用函数（sendPetTypes）之前向 XML 串追加结束 pets 标记。

一旦得到了 XML 串，sendPetTypes 函数继续为请求准备 XMLHttpObject，然后把 XML 串指定为 send()方法的参数，从而将 XML 发送到服务器。

在 createXML 方法中，为什么结束标记中斜线前面有一个反斜线？

SGML □ □ （HTML 就是从 SGML □ 展来的）中提供了一个技巧，利用个技巧可以出 script 元素中的束，但是其他内容（如始和注）不能。使用反斜可以避免把串解析。即使没有反斜，大多数器也能安全地理，但是根据格的 XHTML □ 准，使用反斜。

聪明的读者可能注意到，根据 XMLHttpRequest 对象的文档，send()方法可以将串和 XML 文档对象实例作为参数。那么，这个例子为什么使用串连接来创建 XML，而不是直接创建文档和元素对象呢？遗憾的是，对于从头构建文档对象，目前还没有跨浏览器的技术。IE 通过 ActiveX 对象提供这个功能，Mozilla 浏览器则通过本地 JavaScript 对象来提供，其他浏览器可能根本不支持，也可能通过其他途径来支持这个功能。

读取 XML 的服务器端代码如代码清单 3-10 所示，这个代码稍有些复杂。在此使用了 Java servlet 来读取请求，并解析 XML 串，不过你也可以使用其他的服务器端技术。

一旦收到 XMLHttpRequest 对象的请求，就会调用这个 servlet 的 doPost 方法。doPost 方法使用名为 readXMLFromRequestBody 的辅助方法从请求体中抽取 XML，然后使用 JAXP 接口将 XML 串转换为 Document 对象。

注意，Document 对象是 W3C 指定的 Document 接口的一个实例。因此，它与浏览器的 Document 对象有着同样的方法，如 getElementsByTagName。可以使用这个方法来得到文档中所有 type 元素的列表。对于文档中的每个 type 元素，会得到文本值（应该记得，文本值是 type 元素的第一个子节点），并逐个追加到串中。处理完所有 type 元素后，响应串写回到浏览器。

代码清单 3-10 PostingXMLExample.java

```
package ajaxbook.chap3;


import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class PostingXMLExample extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

        String xml = readXMLFromRequestBody(request);
        Document xmlDoc = null;
        try {
            xmlDoc =
                    DocumentBuilderFactory.newInstance().newDocumentBuilder()
                    .parse(new ByteArrayInputStream(xml.getBytes()));
        }
        catch(ParserConfigurationException e) {
            System.out.println("ParserConfigurationException: " + e);
        }
        catch(SAXException e) {
            System.out.println("SAXException: " + e);
        }

        /* Note how the Java implementation of the W3C DOM has the same methods
         * as the JavaScript implementation, such as getElementsByTagName and
         * getNodeValue.
         */
        NodeList selectedPetTypes = xmlDoc.getElementsByTagName("type");
        String type = null;
        String responseText = "Selected Pets: ";
        for(int i = 0; i < selectedPetTypes.getLength(); i++) {
            type = selectedPetTypes.item(i).getFirstChild().getNodeValue();
```

```
            responseText = responseText + " " + type;
        }

        response.setContentType("text/xml");
        response.getWriter().print(responseText);
    }

    private String readXMLFromRequestBody(HttpServletRequest request){
        StringBuffer xml = new StringBuffer();
        String line = null;
        try {
            BufferedReader reader = request.getReader();
            while((line = reader.readLine()) != null) {
                xml.append(line);
            }
        }
        catch(Exception e) {
            System.out.println("Error reading XML: " + e.toString());
        }
        return xml.toString();
    }
}
```

3.2.2 使用 JSON 向服务器发送数据

做了这么多，你已经能更顺手地使用 JavaScript 了，也许在考虑把更多的模型信息放在浏览器上。不过，看过前面的例子后（使用 XML 向服务器发送复杂的数据结构），你可能会改变主意。通过串连接来创建 XML 串并不好，这也不是用来生成或修改 XML 数据结构的健壮技术。

JSON 概述

XML 的一个替代方法是 JSON，可以在 `www.json.org` 找到。JSON 是一种文本格式，它独立于具体语言，但是使用了与 C 系列语言（如 C、C#、JavaScript 等）类似的约定。JSON 建立在以下两种数据结构基础上，当前几乎所有编程语言都支持这两种数据结构：

- 名/值对集合。在当前编程语言中，这实现为一个对象、记录或字典。
- 值的有序表，这通常实现为一个数组。

因为这些结构得到了如此众多编程语言的支持，所以 JSON 是一个理想的选择，可以作为异构系统之间的一种数据互换格式。另外，由于 JSON 是基于标准 JavaScript 的子集，所以在所有当前 Web 浏览器上都应该是兼容的。

JSON 对象是名/值对的无序集合。对象以 { 开始，以 } 结束，名/值对用冒号分隔。JSON 数组是一个有序的值集合，以[开始，以] 结束，数组中的值用逗号分隔。值可以是串（用双引号引起）、数值、true 或 false、对象，或者是数组，因此结构可以嵌套。图 3-6 以图形方式很好地描述了 JSON 对象的标记。

图 3-6 JSON 对象结构的图形化表示（摘自 www.json.org）

请考虑 employee 对象的简单例子。employee 对象可能包含名、姓、员工号和职位等数据。使用 JSON，可以如下表示 employee 对象实例：

```
var employee = {
    "firstName" : John
    , "lastName" : Doe
    , "employeeNumber" : 123
    , "title" : "Accountant"
}
```

然后可以使用标准点记法使用对象的属性，如下所示：

```
var lastName = employee.lastName;   //Access the last name
var title = employee.title;          //Access the title
employee.employeeNumber = 456;      //Change the employee number
```

JSON 有一点很引以为豪，这就是它是一个轻量级的数据互换格式。如果用 XML 来描述同样的 employee 对象，可能如下所示：

```
<employee>
    <firstName>John</firstName>
    <lastName>Doe</lastName>
    <employeeNumber>123</employeeNumber>
```

<title>Accountant</title>
</employee>

显然，JSON 编码比 XML 编码简短。JSON 编码比较小，所以如果在网络上发送大量数据，可能会带来显著的性能差异。

www.json.org 网站列出了至少与其他编程语言的 14 种绑定，这说明，不论在服务器端使用何种技术，都能通过 JSON 与浏览器通信。

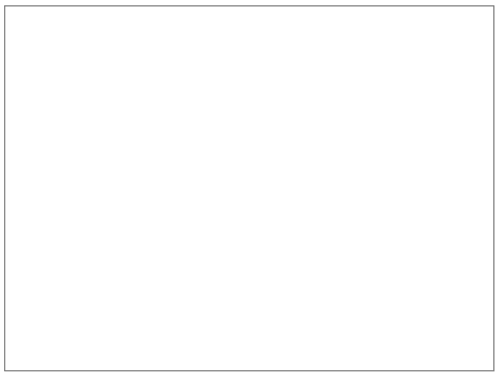
使用 JSON 的示例

下面是一个简单的例子，展示了如何使用 JSON 将 JavaScript 对象转换为串格式，并使用 Ajax 技术将这个串发送到服务器，然后服务器根据这个串创建一个对象。这个例子中没有业务逻辑，也几乎没有用户交互，它强调的是客户端和服务器端的 JSON 技术。图 3-7 显示了一个“字符串化的”Car 对象。

图 3-7“字符串化的”Car 对象

因为这个例子几乎与前面的 POST 例子完全相同，所以我们只关注 JSON 特定的技术。点击表单上的按钮将调用 doJSON 函数。这个函数首先调用 getCarObject 函数来返回一个新的 Car 对象实例，然后使用 JSON JavaScript 库（可以从 www.json.org 免费得到）将 Car 对象转换为 JSON 串，再在警告框中显示这个串。接下来使用 XMLHttpRequest 对象将 JSON 编码的 Car 对象发送到服务器。

因为有可以免费得到的 JSON-Java 绑定库，所以编写 Java servlet 来为 JSON 请求提供服务相当简单。更妙的是，由于对每种服务器端技术都有相应的 JSON 绑定，所以可以使用任何服务器端技术实现这个例子。



JSONExample servlet 的 doPost 方法为 JSON 请求提供服务。它首先调用 readJSONStr-ingFromRequestBody 方法从请求体获得 JSON 串，然后创建 JSONObject 的一个实例，向 JSONObject 构造函数提供 JSON 串。JSONObject 在对象创建时自动解析 JSON 串。一旦创建了 JSONObject，就可以使用各个 get 方法来获得你感兴趣的对象属性。

这里使用 getString 和 getInt 方法来获取 year、make、model 和 color 属性。这些属性连接起来构成一个串返回给浏览器，并在页面上显示。图 3-8 显示了读取 JSON 对象之后的服务器响应。

代码清单 3-11 显示了 jsonExample.html，代码清单 3-12 显示了 JSONExample.java。

代码清单 3-11 jsonExample.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>JSON Example</title>

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

var xmlHttp;

function createXMLHttpRequest() {
      if (window.ActiveXObject) {
            xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
      }
      else if (window.XMLHttpRequest) {
            xmlHttp = new XMLHttpRequest();
      }
}

function doJSON() {
      var car = getCarObject();

      //Use the JSON JavaScript library to stringify the Car object
      var carAsJSON = JSON.stringify(car);
      alert("Car object as JSON:\n " + carAsJSON);

      var url = "JSONExample?timeStamp=" + new Date().getTime();

      createXMLHttpRequest();
      xmlHttp.open("POST", url, true);
      xmlHttp.onreadystatechange = handleStateChange;
```

```
    xmlHttp.setRequestHeader("Content-Type",
                             "application/x-www-form-urlencoded;");
    xmlHttp.send(carAsJSON);
}

function handleStateChange() {
    if(xmlHttp.readyState == 4) {
        if(xmlHttp.status == 200) {
            parseResults();
        }
    }
}

function parseResults() {
    var responseDiv = document.getElementById("serverResponse");
    if(responseDiv.hasChildNodes()) {
        responseDiv.removeChild(responseDiv.childNodes[0]);
    }

    var responseText = document.createTextNode(xmlHttp.responseText);
    responseDiv.appendChild(responseText);
}

function getCarObject() {
    return new Car("Dodge", "Coronet R/T", 1968, "yellow");
}

function Car(make, model, year, color) {
    this.make = make;
    this.model = model;
    this.year = year;
    this.color = color;
}
```

```
</script>
</head>

<body>

   <br/><br/>
   <form action="#">
         <input type="button" value="Click here to send JSON data to the server"
            onclick="doJSON();"/>
   </form>

   <h2>Server Response:</h2>

   <div id="serverResponse"></div>

</body>
</html>
```

代码清单 3-12 JSONExample.java

```
package ajaxbook.chap3;

import java.io.*;
import java.net.*;
import java.text.ParseException;
import javax.servlet.*;
import javax.servlet.http.*;
import org.json.JSONObject;

public class JSONExample extends HttpServlet {

      protected void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
            String json = readJSONStringFromRequestBody(request);

            //Use the JSON-Java binding library to create a JSON object in Java
```

```
        JSONObject jsonObject = null;
        try {
            jsonObject = new JSONObject(json);
        }
        catch(ParseException pe) {
            System.out.println("ParseException: " + pe.toString());
        }

        String responseText = "You have a " + jsonObject.getInt("year") + " "
            + jsonObject.getString("make") + " " + jsonObject.getString("model")
            + " " + " that is " + jsonObject.getString("color") + " in color.";

        response.setContentType("text/xml");
        response.getWriter().print(responseText);
    }

    private String readJSONStringFromRequestBody(HttpServletRequest request){
        StringBuffer json = new StringBuffer();
        String line = null;
        try {
            BufferedReader reader = request.getReader();
            while((line = reader.readLine()) != null) {
                json.append(line);
            }
        }
        catch(Exception e) {
            System.out.println("Error reading JSON string: " + e.toString());
        }
        return json.toString();
    }
}
```

3.3 小结

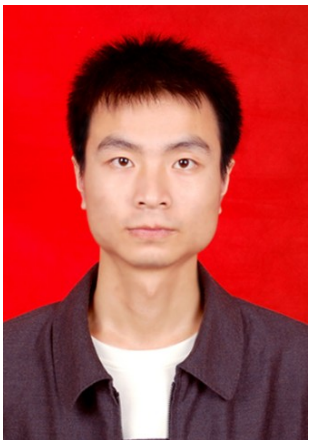
本章介绍了 XMLHttpRequest 对象与服务器之间相互通信的各种方法。XMLHttpRequest 对象可以使用 HTTP `GET` 或 `POST` 方法发送请求，请求数据可以作为查询串、XML 或 JSON 数据发送。处理请求之后，服务器一般会发送简单文本、XML 数据甚至 JSON 数据作为响应。每个格式都有自己最适用的场合。

如果不能根据请求的结果动态更新页面的内容，Ajax 就没有多大的用处。当前的浏览器都把 Web 页面的内容提供为一个遵循 W3C DOM 标准的对象模型。基于这个对象模型，就可以使用 JavaScript 之类的脚本语言在页面上增加、更新和删除内容，而不必与服务器建立往返通信。尽管还是存在一些特异的地方，但如果 Web 页面是根据 W3C 标准编写的，并使用标准 JavaScript 修改，那么在所有与标准兼容的浏览器上这些页面大多都有同样的表现。如今的浏览器还支持非标准的 `innerHTML` 属性，可以用来更新 Web 页面上的元素。

你现在已经熟悉了 XMLHttpRequest 对象，并且了解了如何使用 XMLHttpRequest 对象与服务器进行无缝通信。你还知道了怎样动态地更新 Web 页面的内容。下面再学些什么呢？

Ajax 的潜力无穷无尽，第 4 章将就此简单地谈一谈。知道如何使用 Ajax 只是一方面，如何在合适的环境中加以应用则是另一方面。下一章会介绍一些常见的情况，在这些情况下，Web 应用就很适合采用 Ajax 技术。

联系作者：



NAME: badnewfish

E-Mail: badnewfish@gmail.com

Blog: http://badnewfish.cnblogs.com