学习ajax教程第一天

Ajax 由 HTML、JavaScript™ 技术、DHTML 和 DOM 组成，这一杰出的方法可以将笨拙的 Web 界面转化成交互性的 Ajax 应用程序。本文的作者是一位 Ajax 专家，他演示了这些技术如何协同工作 —— 从总体概述到细节的讨论 —— 使高效的 Web 开发成为现实。他还揭开了 Ajax 核心概念的神秘面纱，包括 XMLHttpRequest 对象。

五年前，如果不知道 XML，您就是一只无人重视的丑小鸭。十八个月前，Ruby 成了关注的中心，不知道 Ruby 的程序员只能坐冷板凳了。今天，如果想跟上最新的技术时尚，那您的目标就是 Ajax。

但是，Ajax 不仅仅是一种时尚，它是一种构建网站的强大方法，而且不像学习一种全新的语言那样困难。

但在详细探讨 Ajax 是什么之前，先让我们花几分钟了解 Ajax 做什么。目前，编写应用程序时有两种基本的选择：

·桌面应用程序

·Web 应用程序

两者是类似的，桌面应用程序通常以 CD 为介质（有时候可从网站下载）并完全安装到您的计算机上。桌面应用程序可能使用互联网下载更新，但运行这些应用程序的代码在桌面计算机上。Web 应用程序运行在某处的 Web 服务器上 —— 毫不奇怪，要通过 Web 浏览器访问这种应用程序。

不过，比这些应用程序的运行代码放在何处更重要的是，应用程序如何运转以及如何与其进行交互。桌面应用程序一般很快（就在您的计算机上运行，不用等待互联网连接），具有漂亮的用户界面（通常和操作系统有关）和非凡的动态性。可以单击、选择、输入、打开菜单和子菜单、到处巡游，基本上不需要等待。

另一方面，Web 应用程序是最新的潮流，它们提供了在桌面上不能实现的服务（比如 Amazon.com 和 eBay）。但是，伴随着 Web 的强大而出现的是等待，等待服务器响应，等待屏幕刷新，等待请求返回和生成新的页面。

显然这样说过于简略了，但基本的概念就是如此。您可能已经猜到，Ajax 尝试建立桌面应用程序的功能和交互性，与不断更新的 Web 应用程序之间的桥梁。可以使用像桌面应用程序中常见的动态用户界面和漂亮的控件，不过是在 Web 应用程序中。

还等什么呢？我们来看看 Ajax 如何将笨拙的 Web 界面转化成能迅速响应的 Ajax 应用程序吧。

老技术，新技巧

在谈到 Ajax 时，实际上涉及到多种技术，要灵活地运用它必须深入了解这些不同的技术（本系列的头几篇文章将分别讨论这些技术）。好消息是您可能已经非常熟悉其中的大部分技术，更好的是这些技术都很容易学习，并不像完整的编程语言（如 Java 或 Ruby）那样困难。

下面是 Ajax 应用程序所用到的基本技术：

·HTML 用于建立 Web 表单并确定应用程序其他部分使用的字段。

·JavaScript 代码是运行 Ajax 应用程序的核心代码，帮助改进与服务器应用程序的通信。

·DHTML 或 Dynamic HTML，用于动态更新表单。我们将使用 div、span 和其他动态 HTML 元素来标记 HTML。

·文档对象模型 DOM 用于（通过 JavaScript 代码）处理 HTML 结构和（某些情况下）服务器返回的 XML。

Ajax 的定义

顺便说一下，Ajax 是 Asynchronous JavaScript and XML（以及 DHTML 等）的缩写。这个短语是 Adaptive Path 的 Jesse James Garrett 发明的（请参阅 参考资料），按照 Jesse 的解释，这不是 个首字母缩写词。

我们来进一步分析这些技术的职责。以后的文章中我将深入讨论这些技术，目前只要熟悉这些组件和技术就可以了。对这些代码越熟悉，就越容易从对这些技术的零散了解转变到真正把握这些技术（同时也真正打开了 Web 应用程序开发的大门）。

XMLHttpRequest 对象

要了解的一个对象可能对您来说也是最陌生的，即 XMLHttpRequest。这是一个 JavaScript 对象，创建该对象很简单，如清单 1 所示。

清单 1. 创建新的 XMLHttpRequest 对象

```
<script language="javascript" type="text/javascript">
var xmlHttp = new XMLHttpRequest();
</script>
```

下一期文章中将进一步讨论这个对象，现在要知道这是处理所有服务器通信的对象。继续阅读之前，先停下来想一想：通过 XMLHttpRequest 对象与服务器进行对话的是 JavaScript 技术。这不是一般的应用程序流，这恰恰是 Ajax 的强大功能的来源。

在一般的 Web 应用程序中，用户填写表单字段并单击 Submit 按钮。然后整个表单发送到服务器，服务器将它转发给处理表单的脚本（通常是 PHP 或 Java，也可能是 CGI 进程或者类似的东西），脚本执行完成后再发送回全新的页面。该页面可能是带有已经填充某些数据的新表单的 HTML，也可能是确认页面，或者是具有根据原来表单中输入数据选择的某些选项的页面。当然，在服务器上的脚本或程

序处理和返回新表单时用户必须等待。屏幕变成一片空白，等到服务器返回数据后再重新绘制。这就是交互性差的原因，用户得不到立即反馈，因此感觉不同于桌面应用程序。

Ajax 基本上就是把 JavaScript 技术和 XMLHttpRequest 对象放在 Web 表单和服务器之间。当用户填写表单时，数据发送给一些 JavaScript 代码而不是直接发送给服务器。相反，JavaScript 代码捕获表单数据并向服务器发送请求。同时用户屏幕上的表单也不会闪烁、消失或延迟。换句话说，JavaScript 代码在幕后发送请求，用户甚至不知道请求的发出。更好的是，请求是异步发送的，就是说 JavaScript 代码（和用户）不用等待服务器的响应。因此用户可以继续输入数据、滚动屏幕和使用应用程序。

然后，服务器将数据返回 JavaScript 代码（仍然在 Web 表单中），后者决定如何处理这些数据。它可以迅速更新表单数据，让人感觉应用程序是立即完成的，表单没有提交或刷新而用户得到了新数据。JavaScript 代码甚至可以对收到的数据执行某种计算，再发送另一个请求，完全不需要用户干预！这就是 XMLHttpRequest 的强大之处。它可以根据需要自行与服务器进行交互，用户甚至可以完全不知道幕后发生的一切。结果就是类似于桌面应用程序的动态、快速响应、高交互性的体验，但是背后又拥有互联网的全部强大力量。

加入一些 JavaScript

得到 XMLHttpRequest 的句柄后，其他的 JavaScript 代码就非常简单了。事实上，我们将使用 JavaScript 代码完成非常基本的任务：

·获取表单数据：JavaScript 代码很容易从 HTML 表单中抽取数据并发送到服务器。

·修改表单上的数据：更新表单也很简单，从设置字段值到迅速替换图像。

·解析 HTML 和 XML：使用 JavaScript 代码操纵 DOM（请参阅 下一节），处理 HTML 表单服务器返回的 XML 数据的结构。

对于前两点，需要非常熟悉 getElementById() 方法，如 清单 2 所示。

清单 2. 用 JavaScript 代码捕获和设置字段值

```
// Get the value of the "phone" field and stuff it in a variable called phone
var phone = document.getElementById("phone").value;
// Set some values on a form using an array called response
document.getElementById("order").value = response[0];
document.getElementById("address").value = response[1];
```

这里没有特别需要注意的地方，真是好极了！您应该认识到这里并没有非常复杂的东西。只要掌握了 XMLHttpRequest，Ajax 应用程序的其他部分就是如 清单 2 所示的简单 JavaScript 代码了，混合有少量的 HTML。同时，还要用一点儿 DOM，我们就来看看吧。

以 DOM 结束

最后还有 DOM，即文档对象模型。可能对有些读者来说 DOM 有点儿令人生畏，HTML 设计者很少使用它，即使 JavaScript 程序员也不大用到它，除非要完成某项高端编程任务。大量使用 DOM 的是 复杂的 Java 和 C/C++ 程序，这可能就是 DOM 被认为难以学习的原因。

幸运的是，在 JavaScript 技术中使用 DOM 很容易，也非常直观。现在，按照常规也许应该说明如何使用 DOM，或者至少要给出一些示例代码，但这样做也可能误导您。即使不理会 DOM，仍然能深入地探讨 Ajax，这也是我准备采用的方法。以后的文章将再次讨论 DOM，现在只要知道可能需要 DOM 就可以了。当需要在 JavaScript 代码和服务器之间传递 XML 和改变 HTML 表单的时候，我们再深入研究 DOM。没有它也能做一些有趣的工作，因此现在就把 DOM 放到一边吧。

获取 Request 对象

有了上面的基础知识后，我们来看看一些具体的例子。XMLHttpRequest 是 Ajax 应用程序的核心，而且对很多读者来说可能还比较陌生，我们就从这里开始吧。从 清单 1 可以看出，创建和使用这个对象非常简单，不是吗？等一等。

还记得几年前的那些讨厌的浏览器战争吗？没有一样东西在不同的浏览器上得到同样的结果。不管您是否相信，这些战争仍然在继续，虽然规模较小。但令人奇怪的是，XMLHttpRequest 成了这场战争的牺牲品之一。因此获得 XMLHttpRequest 对象可能需要采用不同的方法。下面我将详细地进行解释。

使用 Microsoft 浏览器

Microsoft 浏览器 Internet Explorer 使用 MSXML 解析器处理 XML（可以通过 参考资料 进一步了解 MSXML）。因此如果编写的 Ajax 应用程序要和 Internet Explorer 打交道，那么必须用一种特殊的方式创建对象。

但并不是这么简单。根据 Internet Explorer 中安装的 JavaScript 技术版本不同，MSXML 实际上有两种不同的版本，因此必须对这两种情况分别编写代码。请参阅 清单 3，其中的代码在 Microsoft 浏览器上创建了一个 XMLHttpRequest。

清单 3. 在 Microsoft 浏览器上创建 XMLHttpRequest 对象

```
var xmlHttp = false;
try {
  xmlHttp = new ActiveXObject("Msxml2.XMLHTTP");
} catch (e) {
  try {
    xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
```

```
  } catch (e2) {
    xmlHttp = false;
  }
}
```

您对这些代码可能还不完全理解，但没有关系。当本系列文章结束的时候，您将对 JavaScript 编程、错误处理、条件编译等有更深的了解。现在只要牢牢记住其中的两行代码：

```
xmlHttp = new ActiveXObject("Msxml2.XMLHTTP");
```

和

```
xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");。
```

这两行代码基本上就是尝试使用一个版本的 MSXML 创建对象，如果失败则使用另一个版本创建该对象。不错吧？如果都不成功，则将 xmlHttp 变量设为 false，告诉您的代码出现了问题。如果出现这种情况，可能是因为安装了非 Microsoft 浏览器，需要使用不同的代码。

处理 Mozilla 和非 Microsoft 浏览器

如果选择的浏览器不是 Internet Explorer，或者为非 Microsoft 浏览器编写代码，就需要使用不同的代码。事实上就是 清单 1 所示的一行简单代码：

```
var xmlHttp = new XMLHttpRequest object;。
```

这行简单得多的代码在 Mozilla、Firefox、Safari、Opera 以及基本上所有以任何形式或方式支持 Ajax 的非 Microsoft 浏览器中，创建了 XMLHttpRequest 对象。

结合起来

关键是要支持所有 浏览器。谁愿意编写一个只能用于 Internet Explorer 或者非 Microsoft 浏览器的应用程序呢？或者更糟，要编写一个应用程序两次？当然不！因此 代码要同时支持 Internet Explorer 和非 Microsoft 浏览器。清单 4 显示了这样的代码。

清单 4. 以支持多种浏览器的方式创建 XMLHttpRequest 对象

```
/* Create a new XMLHttpRequest object to talk to the Web server */
var xmlHttp = false;
/*@cc_on @*/
```

```
/*@if (@_jscript_version >= 5)
try {
  xmlHttp = new ActiveXObject("Msxml2.XMLHTTP");
} catch (e) {
  try {
    xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
  } catch (e2) {
    xmlHttp = false;
  }
}
@end @*/
if (!xmlHttp && typeof XMLHttpRequest != 'undefined') {
  xmlHttp = new XMLHttpRequest();
}
```

现在先不管那些注释掉的奇怪符号，如 @cc_on，这是特殊的 JavaScript 编译器命令，将在下一期针对 XMLHttpRequest 的文章中详细讨论。这段代码的核心分为三步：

1、建立一个变量 xmlHttp 来引用即将创建的 XMLHttpRequest 对象。

2、尝试在 Microsoft 浏览器中创建该对象：

1）尝试使用 Msxml2.XMLHTTP 对象创建它。

2）如果失败，再尝试 Microsoft.XMLHTTP 对象。

2、如果仍然没有建立 xmlHttp，则以非 Microsoft 的方式创建该对象。

最后，xmlHttp 应该引用一个有效的 XMLHttpRequest 对象，无论运行什么样的浏览器。

关于安全性的一点说明

安全性如何呢？现在浏览器允许用户提高他们的安全等级，关闭 JavaScript 技术，禁用浏览器中的任何选项。在这种情况下，代码无论如何都不会工作。此时必须适当地处理问题，这需要单独的一篇文章来讨论，要放到以后了（这个系列够长了吧？不用担心，读完之前也许您就掌握了）。现在要编写一段健壮但不够完美的代码，对于掌握 Ajax 来说就很好了。以后我们还将讨论更多的细节。

Ajax 世界中的请求/响应

现在我们介绍了 Ajax，对 XMLHttpRequest 对象以及如何创建它也有了基本的了解。如果阅读得很仔细，您可能已经知道与服务器上的 Web 应用程序打交道的是 JavaScript 技术，而不是直接提交给那个应用程序的 HTML 表单。

还缺少什么呢？到底如何使用 XMLHttpRequest。因为这段代码非常重要，您编写的每个 Ajax 应用程序都要以某种形式使用它，先看看 Ajax 的基本请求/响应模型是什么样吧。

发出请求

您已经有了一个崭新的 XMLHttpRequest 对象，现在让它干点活儿吧。首先需要一个 Web 页面能够调用的 JavaScript 方法（比如当用户输入文本或者从菜单中选择一项时）。接下来就是在所有 Ajax 应用程序中基本都雷同的流程：

1、从 Web 表单中获取需要的数据。

2、建立要连接的 URL。

3、打开到服务器的连接。

4、设置服务器在完成后要运行的函数。

5、发送请求。

清单 5 中的示例 Ajax 方法就是按照这个顺序组织的：

清单 5. 发出 Ajax 请求

```
function callServer() {
  // Get the city and state from the web form
  var city = document.getElementById("city").value;
  var state = document.getElementById("state").value;
  // Only go on if there are values for both fields
```

```
  if ((city == null) || (city == "")) return;
  if ((state == null) || (state == "")) return;
  // Build the URL to connect to
  var url = "/scripts/getZipCode.php?city=" + escape(city) + "&state=" + escape(state);
  // Open a connection to the server
  xmlHttp.open("GET", url, true);
  // Setup a function for the server to run when it's done
  xmlHttp.onreadystatechange = updatePage;
  // Send the request
  xmlHttp.send(null);
}
```

其中大部分代码意义都很明确。开始的代码使用基本 JavaScript 代码获取几个表单字段的值。然后设置一个 PHP 脚本作为链接的目标。要注意脚本 URL 的指定方式，city 和 state（来自表单）使用简单的 GET 参数附加在 URL 之后。

然后打开一个连接，这是您第一次看到使用 XMLHttpRequest。其中指定了连接方法 GET）和要连接的 URL。最后一个参数如果设为 true，那么将请求一个异步连接（这就是 Ajax 的由来）。如果使用 false，那么代码发出请求后将等待服务器返回的响应。如果设为 true，当服务器在后台处理请求的时候用户仍然可以使用表单（甚至调用其他 JavaScript 方法）。

xmlHttp（要记住，这是 XMLHttpRequest 对象实例）的 onreadystatechange 属性可以告诉服务器在运行完成 后（可能要用五分钟或者五个小时）做什么。因为代码没有等待服务器，必须让服务器知道怎么做以便您能作出响应。在这个示例中，如果服务器处理完了请求，一个特殊的名为 updatePage() 的方法将被触发。

最后，使用值 null 调用 send()。因为已经在请求 URL 中添加了要发送给服务器的数据（city 和 state），所以请求中不需要发送任何数据。这样就发出了请求，服务器按照您的要求工作。

如果没有发现任何新鲜的东西，您应该体会到这是多么简单明了！除了牢牢记住 Ajax 的异步特性外，这些内容都相当简单。应该感激 Ajax 使您能够专心编写漂亮的应用程序和界面，而不用担心复杂的 HTTP 请求/响应代码。

清单 5 中的代码说明了 Ajax 的易用性。数据是简单的文本，可以作为请求 URL 的一部分。用 GET 而不是更复杂的 POST 发送请求。没有 XML 和要添加的内容头部，请求体中没有要发送的数据；换句话说，这就是 Ajax 的乌托邦。

不用担心，随着本系列文章的展开，事情会变得越来越复杂。您将看到如何发送 POST 请求、如何设置请求头部和内容类型、如何在消息中编码 XML、如何增加请求的安全性，可以做的工作还有很多！暂时先不用管那些难点，掌握好基本的东西就行了，很快我们就会建立一整套的 Ajax 工具库。

处理响应

现在要面对服务器的响应了。现在只要知道两点：

·什么也不要做，直到 xmlHttp.readyState 属性的值等于 4。

·服务器将把响应填充到 xmlHttp.responseText 属性中。

其中的第一点，即就绪状态，将在下一篇文章中详细讨论，您将进一步了解 HTTP 请求的阶段，可能比您设想的还多。现在只要检查一个特定的值（4）就可以了（下一期文章中还有更多的值要介绍）。第二点，使用 xmlHttp.responseText 属性获得服务器的响应，这很简单。清单 6 中的示例方法可供服务器根据 清单 5 中发送的数据调用。

清单 6. 处理服务器响应

```
function updatePage() {
  if (xmlHttp.readyState == 4) {
    var response = xmlHttp.responseText;
    document.getElementById("zipCode").value = response;
  }
}
```

这些代码同样既不难也不复杂。它等待服务器调用，如果是就绪状态，则使用服务器返回的值（这里是用户输入的城市和州的 ZIP 编码）设置另一个表单字段的值。于是包含 ZIP 编码的 zipCode 字段突然出现了，而用户没有按任何按钮！这就是前面所说的桌面应用程序的感觉。快速响应、动态感受等等，这些都只因为有了小小的一段 Ajax 代码。

细心的读者可能注意到 zipCode 是一个普通的文本字段。一旦服务器返回 ZIP 编码，updatePage() 方法就用城市/州的 ZIP 编码设置那个字段的值，用户就可以改写该值。这样做有两个原因：保持例子简单，说明有时候可能希望 用户能够修改服务器返回的数据。要记住这两点，它们对于好的用户界面设计来说很重要。

连接 Web 表单

还有什么呢？实际上没有多少了。一个 JavaScript 方法捕捉用户输入表单的信息并将其发送到服务器，另一个 JavaScript 方法监听和处理响应，并在响应返回时设置字段的值。所有这些实际上都依赖于调用 第一个 JavaScript 方法，它启动了整个过程。最明显的办法是在 HTML 表单中增加一个按钮，但这是 2001 年的办法，您不这样认为吗？还是像 清单 7 这样利用 JavaScript 技术吧。

清单 7. 启动一个 Ajax 过程

```
<form>
 <p>City: <input type="text" name="city" id="city" size="25"
      /></p>
 <p>State: <input type="text" name="state" id="state" size="25"
      /></p>
 <p>Zip Code: <input type="text" name="zipCode" id="city" size="5" /></p>
</form>
```

如果感觉这像是一段相当普通的代码，那就对了，正是如此！当用户在 city 或 state 字段中输入新的值时， callServer() 方法就被触发，于是 Ajax 开始运行了。有点儿明白怎么回事了吧？好，就是如此！

结束语

现在您可能已经准备开始编写第一个 Ajax 应用程序了，至少也希望认真读一下 参考资料 中的那些文章了吧？但可以首先从这些应用程序如何工作的基本概念开始，对 XMLHttpRequest 对象有基本的了解。在下一期文章中，您将掌握这个对象，学会如何处理 JavaScript 和服务器的通信、如何使用 HTML 表单以及如何获得 DOM 句柄。

现在先花点儿时间考虑考虑 Ajax 应用程序有多么强大。设想一下，当单击按钮、输入一个字段、从组合框中选择一个选项或者用鼠标在屏幕上拖动时，Web 表单能够立刻作出响应会是什么情形。想一想异步 究竟意味着什么，想一想 JavaScript 代码运行而且不等待 服务器对它的请求作出响应。会遇到什么样的问题？会进入什么样的领域？考虑到这种新的方法，编程的时候应如何改变表单的设计？

如果在这些问题上花一点儿时间，与简单地剪切/粘贴某些代码到您根本不理解的应用程序中相比，收益会更多。在下一期文章中，我们将把这些概念付诸实践，详细介绍使应用程序按照这种方式工作所需要的代码。因此，现在先享受一下 Ajax 所带来的可能性吧。

学习Ajax教程第二天，JavaScript 和 Ajax 发出异步请求

多数 Web 应用程序都使用请求/响应模型从服务器上获得完整的 HTML 页面。常常是点击一个按钮，等待服务器响应，再点击另一个按钮，然后再等待，这样一个反复的过程。有了 Ajax 和 XMLHttpRequest 对象，就可以使用不必让用户等待服务器响应的请求/响应模型了。本文中，Brett McLaughlin 介绍了如何创建能够适应不同浏览器的 XMLHttpRequest 实例，建立和发送请求，并响应服务器。

本系列的上一期文章（请参阅 参考资料 中的链接），我们介绍了 Ajax 应用程序，考察了推动 Ajax 应用程序的基本概念。其中的核心是很多您可能已经了解的技术：JavaScript、HTML 和 XHTML、一点动态 HTML 以及 DOM（文档对象模型）。本文将放大其中的一点，把目光放到具体的 Ajax 细节上。

本文中，您将开始接触最基本和基础性的有关 Ajax 的全部对象和编程方法：XMLHttpRequest 对象。该对象实际上仅仅是一个跨越所有 Ajax 应用程序的公共线程，您可能已经预料到，只有彻底理解该对象才能充分发挥编程的潜力。事实上，有时您会发现，要正确地使用 XMLHttpRequest，显然不能 使用 XMLHttpRequest。这到底是怎么回事呢？

Web 2.0 一瞥

在深入研究代码之前首先看看最近的观点 —— 一定要十分清楚 Web 2.0 这个概念。听到 Web 2.0 这个词的时候，应该首先问一问“Web 1.0 是什么？”虽然很少听人提到 Web 1.0，实际上它指的就是具有完全不同的请求和响应模型的传统 Web。比如，到 Amazon.com 网站上点击一个按钮或者输入搜索项。就会对服务器发送一个请求，然后响应再返回到浏览器。该请求不仅仅是图书和书目列表，而是另一个完整的 HTML 页面。因此当 Web 浏览器用新的 HTML 页面重绘时，可能会看到闪烁或抖动。事实上，通过看到的每个新页面可以清晰地看到请求和响应。

Web 2.0（在很大程度上）消除了这种看得见的往复交互。比如访问 Google Maps 或 Flickr 这样的站点（到这些支持 Web 2.0 和 Ajax 站点的链接请参阅 参考资料）。比如在 Google Maps 上，您可以拖动地图，放大和缩小，只有很少的重绘操作。当然这里仍然有请求和响应，只不过都藏到了幕后。作为用户，体验更加舒适，感觉很像桌面应用程序。这种新的感受和范型就是当有人提到 Web 2.0 时您所体会到的。

需要关心的是如何使这些新的交互成为可能。显然，仍然需要发出请求和接收响应，但正是针对每次请求/响应交互的 HTML 重绘造成了缓慢、笨拙的 Web 交互的感受。因此很清楚，我们需要一种方法使发送的请求和接收的响应只 包含需要的数据而不是整个 HTML 页面。惟一需要获得整个新 HTML 页面的时候就是希望用户看到 新页面的时候。

但多数交互都是在已有页面上增加细节、修改主体文本或者覆盖原有数据。这些情况下，Ajax 和 Web 2.0 方法允许在不 更新整个 HTML 页面的情况下发送和接收数据。对于那些经常上网的人，这种能力可以让您的应用程序感觉更快、响应更及时，让他们不时地光顾您的网站。

XMLHttpRequest 简介

要真正实现这种绚丽的奇迹，必须非常熟悉一个 JavaScript 对象，即 XMLHttpRequest。这个小小的对象实际上已经在几种浏览器中存在一段时间了，它是本专栏今后几个月中要介绍的 Web 2.0、Ajax 和大部分其他内容的核心。为了让您快速地大体了解它，下面给出将要用于该对象的很少的几个 方法和属性。

·open()：建立到服务器的新请求。

·send()：向服务器发送请求。

·abort()：退出当前请求。

·readyState：提供当前 HTML 的就绪状态。

·responseText：服务器返回的请求响应文本。

如果不了解这些（或者其中的任何 一个），您也不用担心，后面几篇文章中我们将介绍每个方法和属性。现在应该 了解的是，明确用 XMLHttpRequest 做什么。要注意这些方法和属性都与发送请求及处理响应有关。事实上，如果看到 XMLHttpRequest 的所有方法和属性，就会发现它们都 与非常简单的请求/响应模型有关。显然，我们不会遇到特别新的 GUI 对象或者创建用户交互的某种超极神秘的方法，我们将使用非常简单的请求和非常简单的响应。听起来似乎没有多少吸引力，但是用好该对象可以彻底改变您的应用程序。

简单的 new

首先需要创建一个新变量并赋给它一个 XMLHttpRequest 对象实例。这在 JavaScript 中很简单，只要对该对象名使用 new 关键字即可，如 清单 1 所示。

清单 1. 创建新的 XMLHttpRequest 对象

```
<script language="javascript" type="text/javascript">
var request = new XMLHttpRequest();
</script>
```

不难吧？记住，JavaScript 不要求指定变量类型，因此不需要像 清单 2 那样做（在 Java 语言中可能需要这样）。

清单 2. 创建 XMLHttpRequest 的 Java 伪代码

```
XMLHttpRequest request = new XMLHttpRequest();
```

因此在 JavaScript 中用 var 创建一个变量，给它一个名字（如 “request”），然后赋给它一个新的 XMLHttpRequest 实例。此后就可以在函数中使用该对象了。

错误处理

在实际上各种事情都可能出错，而上面的代码没有提供任何错误处理。较好的办法是创建该对象，并在出现问题时优雅地退出。比如，任何较早的浏览器（不论您是否相信，仍然有人在使用老版本的 Netscape Navigator）都不支持 XMLHttpRequest，您需要让这些用户知道有些地方出了问题。清单 3 说明如何创建该对象，以便在出现问题的时候发出 JavaScript 警告。

清单 3. 创建具有错误处理能力的 XMLHttpRequest

```
<script language="javascript" type="text/javascript">
var request = false;
try {
  request = new XMLHttpRequest();
} catch (failed) {
  request = false;
}
if (!request)
  alert("Error initializing XMLHttpRequest!");
</script>
```

一定要理解这些步骤：

1、创建一个新变量 request 并赋值 false。后面将使用 false 作为判定条件，它表示还没有创建 XMLHttpRequest 对象。

2、增加 try/catch 块：

1）尝试创建 XMLHttpRequest 对象。

2）如果失败（catch (failed)）则保证 request 的值仍然为 false。

3、检查 request 是否仍为 false（如果一切正常就不会是 false）。

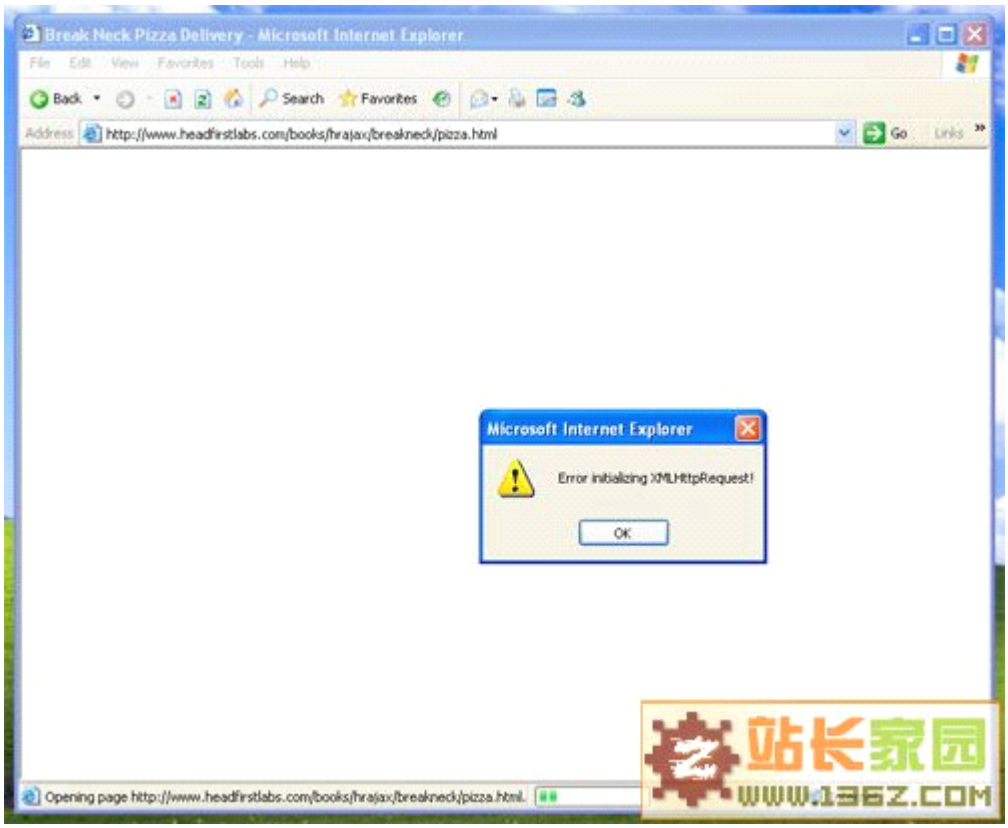
4、如果出现问题（request 是 false）则使用 JavaScript 警告通知用户出现了问题。

代码非常简单，对大多数 JavaScript 和 Web 开发人员来说，真正理解它要比读写代码花更长的时间。现在已经得到了一段带有错误检查的 XMLHttpRequest 对象创建代码，还可以告诉您哪儿出了问题。

应付 Microsoft

看起来似乎一切良好，至少在用 Internet Explorer 试验这些代码之前是这样的。如果这样试验的话，就会看到 图 1 所示的糟糕情形。

图 1. Internet Explorer 报告错误



显然有什么地方不对劲，而 Internet Explorer 很难说是一种过时的浏览器，因为全世界有 70% 在使用 Internet Explorer。换句话说，如果不支持 Microsoft 和 Internet Explorer 就不会受到 Web 世界的欢迎！因此我们需要采用不同的方法处理 Microsoft 浏览器。

经验证发现 Microsoft 支持 Ajax，但是其 XMLHttpRequest 版本有不同的称呼。事实上，它将其称为几种 不同的东西。如果使用较新版本的 Internet Explorer，则需要使用对象 Msxml2.XMLHTTP，而较老版本的 Internet Explorer 则使用 Microsoft.XMLHTTP。我们需要支持这两种对象类型（同时还要支持非 Microsoft 浏览器）。请看看 清单 4，它在前述代码的基础上增加了对 Microsoft 的支持。

Microsoft 参与了吗？

关于 Ajax 和 Microsoft 对该领域不断增长的兴趣和参与已经有很多文章进行了介绍。事实上，据说 Microsoft 最新版本的 Internet Explorer —— version 7.0，将在 2006 年下半年推出 —— 将开始直接支持 XMLHttpRequest，让您使用 new 关键字代替所有的 Msxml2.XMLHTTP 创建代码。但不要太激动，仍然需要支持旧的浏览器，因此跨浏览器代码不会很快消失。

清单 4. 增加对 Microsoft 浏览器的支持

```
<script language="javascript" type="text/javascript">
```

```
var request = false;
try {
  request = new XMLHttpRequest();
} catch (trymicrosoft) {
  try {
    request = new ActiveXObject("Msxml2.XMLHTTP");
  } catch (othermicrosoft) {
    try {
      request = new ActiveXObject("Microsoft.XMLHTTP");
    } catch (failed) {
      request = false;
    }
  }
}
if (!request)
  alert("Error initializing XMLHttpRequest!");
</script>
```

很容易被这些花括号迷住了眼睛，因此下面分别介绍每一步：

1、创建一个新变量 request 并赋值 false。使用 false 作为判断条件，它表示还没有创建 XMLHttpRequest 对象。

2、增加 try/catch 块：

1）尝试创建 XMLHttpRequest 对象。

2）如果失败（catch (trymicrosoft)）：

1>尝试使用较新版本的 Microsoft 浏览器创建 Microsoft 兼容的对象（Msxml2.XMLHTTP）。

2> 如果失败（catch (othermicrosoft)）尝试使用较老版本的 Microsoft 浏览器创建 Microsoft 兼容的对象（Microsoft.XMLHTTP）。

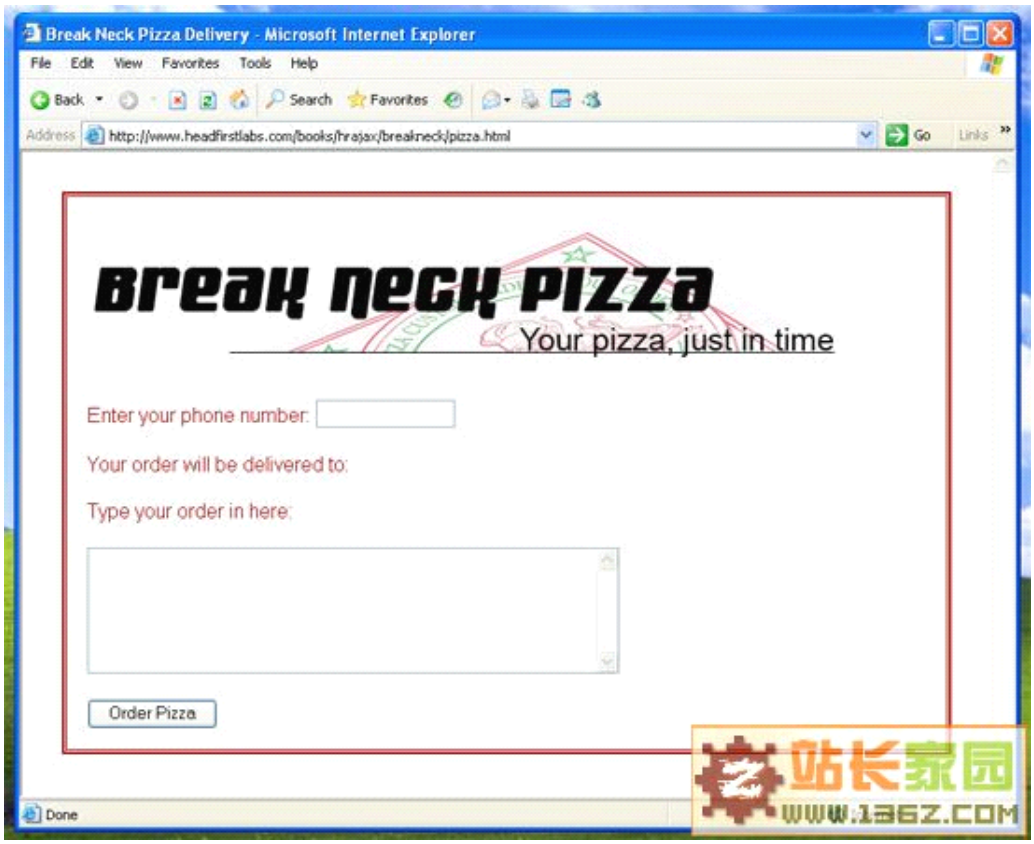
2）如果失败（catch (failed)）则保证 request 的值仍然为 false。

3、检查 request 是否仍然为 false（如果一切顺利就不会是 false）。

4、如果出现问题（request 是 false）则使用 JavaScript 警告通知用户出现了问题。

这样修改代码之后再使用 Internet Explorer 试验，就应该看到已经创建的表单（没有错误消息）。我实验的结果如 图 2 所示。

图 2. Internet Explorer 正常工作



静态与动态

再看一看清单 1、3 和 4，注意，所有这些代码都直接嵌套在 script 标记中。像这种不放到方法或函数体中的 JavaScript 代码称为静态 JavaScript。就是说代码是在页面显示给用户之前的某个时候运行。（虽然根据规范不能完全精确地知道这些代码何时运行对浏览器有什么影响，但是可以保证这些代码在用户能够与页面交互之前运行。）这也是多数 Ajax 程序员创建 XMLHttpRequest 对象的一般方式。就是说，也可以像 清单 5 那样将这些代码放在一个方法中。

清单 5. 将 XMLHttpRequest 创建代码移动到方法中

```
<script language="javascript" type="text/javascript">
```

```
var request;

function createRequest() {

  try {

    request = new XMLHttpRequest();

  } catch (trymicrosoft) {

    try {

     request = new ActiveXObject("Msxml2.XMLHTTP");

    } catch (othermicrosoft) {

     try {

       request = new ActiveXObject("Microsoft.XMLHTTP");

     } catch (failed) {

       request = false;

     }

    }

  }

  if (!request)

    alert("Error initializing XMLHttpRequest!");

}

</script>
```

如果按照这种方式编写代码，那么在处理 Ajax 之前需要调用该方法。因此还需要 清单 6 这样的代码。

清单 6. 使用 XMLHttpRequest 的创建方法

```
<script language="javascript" type="text/javascript">

var request;
```

```
function createRequest() {
  try {
    request = new XMLHttpRequest();
  } catch (trymicrosoft) {
    try {
      request = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (othermicrosoft) {
      try {
        request = new ActiveXObject("Microsoft.XMLHTTP");
      } catch (failed) {
        request = false;
      }
    }
  }
  if (!request)
    alert("Error initializing XMLHttpRequest!");
}
function getCustomerInfo() {
  createRequest();
  // Do something with the request variable
}
</script>
```

此代码惟一的问题是推迟了错误通知，这也是多数 Ajax 程序员不采用这一方法的原因。假设一个复杂的表单有 10 或 15 个字段、选择框等，当用户在第 14 个字段（按照表单顺序从上到下）输入文本时要激活某些 Ajax 代码。这时候运行 getCustomerInfo() 尝试创建

一个 XMLHttpRequest 对象，但（对于本例来说）失败了。然后向用户显示一条警告，明确地告诉他们不能使用该应用程序。但用户已经花费了很多时间在表单中输入数据！这是非常令人讨厌的，而讨厌显然不会吸引用户再次访问您的网站。

如果使用静态 JavaScript，用户在点击页面的时候很快就会看到错误信息。这样也很烦人，是不是？可能令用户错误地认为您的 Web 应用程序不能在他的浏览器上运行。不过，当然要比他们花费了 10 分钟输入信息之后再显示同样的错误要好。因此，我建议编写静态的代码，让用户尽可能早地发现问题。

用 XMLHttpRequest 发送请求

得到请求对象之后就可以进入请求/响应循环了。记住，XMLHttpRequest 惟一的目的是让您发送请求和接收响应。其他一切都是 JavaScript、CSS 或页面中其他代码的工作：改变用户界面、切换图像、解释服务器返回的数据。准备好 XMLHttpRequest 之后，就可以向服务器发送请求了。

欢迎使用沙箱

Ajax 采用一种沙箱安全模型。因此，Ajax 代码（具体来说就是 XMLHttpRequest 对象）只能对所在的同一个域发送请求。以后的文章中将进一步介绍安全和 Ajax，现在只要知道在本地机器上运行的代码只能对本地机器上的服务器端脚本发送请求。如果让 Ajax 代码在 www.breakneckpizza.com 上运行，则必须 www.breakneck.com 中运行的脚本发送请求。

设置服务器 URL

首先要确定连接的服务器的 URL。这并不是 Ajax 的特殊要求，但仍然是建立连接所必需的，显然现在您应该知道如何构造 URL 了。多数应用程序中都会结合一些静态数据和用户处理的表单中的数据来构造该 URL。比如，清单 7 中的 JavaScript 代码获取电话号码字段的值并用其构造 URL。

清单 7. 建立请求 URL

```
<script language="javascript" type="text/javascript">
  var request = false;
  try {
    request = new XMLHttpRequest();
  } catch (trymicrosoft) {
    try {
```

```
      request = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (othermicrosoft) {
      try {
        request = new ActiveXObject("Microsoft.XMLHTTP");
      } catch (failed) {
        request = false;
      }
    }
  }
  if (!request)
    alert("Error initializing XMLHttpRequest!");
  function getCustomerInfo() {
    var phone = document.getElementById("phone").value;
    var url = "/cgi-local/lookupCustomer.php?phone=" + escape(phone);
  }
</script>
```

这里没有难懂的地方。首先，代码创建了一个新变量 phone，并把 ID 为 “phone” 的表单字段的值赋给它。清单 8 展示了这个表单的 XHTML，其中可以看到 phone 字段及其 id 属性。

清单 8. Break Neck Pizza 表单

```
<body>
  <p><img src="/images/NullPic.gif" alt="Break Neck Pizza" /></p>
  <form action="POST">
    <p>Enter your phone number:
      <input type="text" size="14" name="phone" id="phone"
          />
```

```
  </p>
  <p>Your order will be delivered t</p>
  <div id="address"></div>
  <p>Type your order in here:</p>
  <p><textarea name="order" rows="6" cols="50" id="order"></textarea></p>
  <p><input type="submit" value="Order Pizza" id="submit" /></p>
 </form>
</body>
```

还要注意，当用户输入电话号码或者改变电话号码时，将触发 清单 8 所示的 getCustomerInfo() 方法。该方法取得电话号码并构造存储在 url 变量中的 URL 字符串。记住，由于 Ajax 代码是沙箱型的，因而只能连接到同一个域，实际上 URL 中不需要域名。该例中的脚本名为 /cgi-local/lookupCustomer.php。最后，电话号码作为 GET 参数附加到该脚本中："phone=" + escape(phone)。

如果以前没用见过 escape() 方法，它用于转义不能用明文正确发送的任何字符。比如，电话号码中的空格将被转换成字符 %20，从而能够在 URL 中传递这些字符。

可以根据需要添加任意多个参数。比如，如果需要增加另一个参数，只需要将其附加到 URL 中并用“与”（&）字符分开 [第一个参数用问号（?）和脚本名分开]。

打开请求

有了要连接的 URL 后就可以配置请求了。可以用 XMLHttpRequest 对象的 open() 方法来完成。该方法有五个参数：

request-type：发送请求的类型。典型的值是 GET 或 POST，但也可以发送 HEAD 请求。

url：要连接的 URL。

asynch：如果希望使用异步连接则为 true，否则为 false。该参数是可选的，默认为 true。

username：如果需要身份验证，则可以在此指定用户名。该可选参数没有默认值。 password：如果需要身份验证，则可以在此指定口令。该可选参数没有默认值。

open() 是打开吗？

Internet 开发人员对 open() 方法到底做什么没有达成一致。但它实际上并不是 打开一个请求。如果监控 XHTML/Ajax 页面及其连接脚本之间的网络和数据传递，当调用 open() 方法时将看不到任何通信。不清楚为何选用了这个名字，但显然不是一个好的选择。

通常使用其中的前三个参数。事实上，即使需要异步连接，也应该指定第三个参数为“true”。这是默认值，但坚持明确指定请求是异步的还是同步的更容易理解。

将这些结合起来，通常会得到清单 9 所示的一行代码。

清单 9. 打开请求

```
function getCustomerInfo() {
  var phone = document.getElementById("phone").value;
  var url = "/cgi-local/lookupCustomer.php?phone=" + escape(phone);
  request.open("GET", url, true);
}
```

一旦设置好了 URL，其他就简单了。多数请求使用 GET 就够了（后面的文章中将看到需要使用 POST 的情况），再加上 URL，这就是使用 open() 方法需要的全部内容了。

挑战异步性

本系列的后面一篇文章中，我将用很多时间编写和使用异步代码，但是您应该明白为什么 open() 的最后一个参数这么重要。在一般的请求/响应模型中，比如 Web 1.0，客户机（浏览器或者本地机器上运行的代码）向服务器发出请求。该请求是同步的，换句话说，客户机等待服务器的响应。当客户机等待的时候，至少会用某种形式通知您在等待：

·沙漏（特别是 Windows 上）。

·旋转的皮球（通常在 Mac 机器上）。

·应用程序基本上冻结了，然后过一段时间光标变化了。

这正是 Web 应用程序让人感到笨拙或缓慢的原因 —— 缺乏真正的交互性。按下按钮时，应用程序实际上变得不能使用，直到刚刚触发的请求得到响应。如果请求需要大量服务器处理，那么等待的时间可能很长（至少在这个多处理器、DSL 没有等待的世界中是如此）。而异步请求不等待服务器响应。发送请求后应用程序继续运行。用户仍然可以在 Web 表单中输入数据，甚至离开表单。没有旋转的皮球或者沙漏，应用程序也没有明显的冻结。服务器悄悄地响应请求，完成后告诉原来的请求者工作已经结束（具体的办法很快就会看到）。结果是，应用程序感觉不那么迟钝或者缓慢，而是响应迅速、交互性强，感觉快多了。这仅仅是 Web 2.0 的一部分，但它是很重要的一部分。所有老套的 GUI 组件和 Web 设计范型都不能克服缓慢、同步的请求/响应模型。

发送请求

一旦用 open() 配置好之后，就可以发送请求了。幸运的是，发送请求的方法的名称要比 open() 适当，它就是 send()。

send() 只有一个参数，就是要发送的内容。但是在考虑这个方法之前，回想一下前面已经通过 URL 本身发送过数据了：

```
var url = "/cgi-local/lookupCustomer.php?phone=" + escape(phone);
```

虽然可以使用 send() 发送数据，但也能通过 URL 本身发送数据。事实上，GET 请求（在典型的 Ajax 应用中大约占 80%）中，用 URL 发送数据要容易得多。如果需要发送安全信息或 XML，可能要考虑使用 send() 发送内容（本系列的后续文章中将讨论安全数据和 XML 消息）。如果不需要通过 send() 传递数据，则只要传递 null 作为该方法的参数即可。因此您会发现在本文中的例子中只需要这样发送请求（参见 清单 10）。

清单 10. 发送请求

```
function getCustomerInfo() {
  var phone = document.getElementById("phone").value;
  var url = "/cgi-local/lookupCustomer.php?phone=" + escape(phone);
  request.open("GET", url, true);
  request.send(null);
}
```

指定回调方法

现在我们所做的只有很少一点是新的、革命性的或异步的。必须承认，open() 方法中“true”这个小小的关键字建立了异步请求。但是除此之外，这些代码与用 Java servlet 及 JSP、PHP 或 Perl 编程没有什么两样。那么 Ajax 和 Web 2.0 最大的秘密是什么呢？秘密就在于 XMLHttpRequest 的一个简单属性 onreadystatechange。

首先一定要理解这些代码中的流程（如果需要请回顾 清单 10）。建立其请求然后发出请求。此外，因为是异步请求，所以 JavaScript 方法（例子中的 getCustomerInfo()）不会等待服务器。因此代码将继续执行，就是说，将退出该方法而把控制返回给表单。用户可以继续输入信息，应用程序不会等待服务器。

这就提出了一个有趣的问题：服务器完成了请求之后会发生什么？答案是什么也不发生，至少对现在的代码而言如此！显然这样不行，因此服务器在完成通过 XMLHttpRequest 发送给它的请求处理之后需要某种指示说明怎么做。

在 JavaScript 中引用函数：

JavaScript 是一种弱类型的语言，可以用变量引用任何东西。因此如果声明了一个函数 updatePage()，JavaScript 也将该函数名看作是一个变量。换句话说，可用变量名 updatePage 在代码中引用函数。

清单 11. 设置回调方法

```
function getCustomerInfo() {
    var phone = document.getElementById("phone").value;
    var url = "/cgi-local/lookupCustomer.php?phone=" + escape(phone);
    request.open("GET", url, true);
    request.onreadystatechange= updatePage;
    request.send(null);
  }
```

需要特别注意的是该属性在代码中设置的位置 —— 它是在调用 send() 之前 设置的。发送请求之前必须设置该属性，这样服务器在回答完成请求之后才能查看该属性。现在剩下的就只有编写 updatePage() 方法了，这是本文最后一节要讨论的重点。

处理服务器响应

发送请求，用户高兴地使用 Web 表单（同时服务器在处理请求），而现在服务器完成了请求处理。服务器查看 onreadystatechange 属性确定要调用的方法。除此以外，可以将您的应用程序看作其他应用程序一样，无论是否异步。换句话说，不一定要采取特殊的动作编写响应服务器的方法，只需要改变表单，让用户访问另一个 URL 或者做响应服务器需要的任何事情。这一节我们重点讨论对服务器的响应和一种典型的动作 —— 即时改变用户看到的表单中的一部分。

回调和 Ajax

现在我们已经看到如何告诉服务器完成后应该做什么：将 XMLHttpRequest 对象的 onreadystatechange 属性设置为要运行的函数名。这样，当服务器处理完请求后就会自动调用该函数。也不需要担心该函数的任何参数。我们从一个简单的方法开始，如 清单 12 所示。

清单 12. 回调方法的代码

```
<script language="javascript" type="text/javascript">
  var request = false;
```

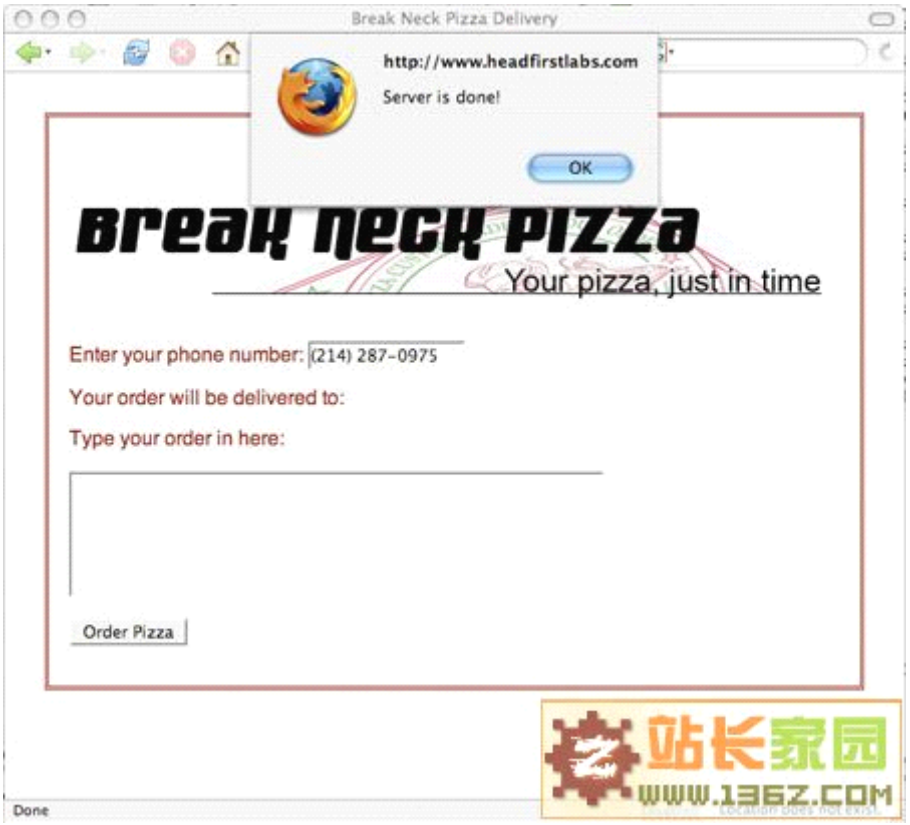
```
try {
  request = new XMLHttpRequest();
} catch (trymicrosoft) {
  try {
    request = new ActiveXObject("Msxml2.XMLHTTP");
  } catch (othermicrosoft) {
    try {
      request = new ActiveXObject("Microsoft.XMLHTTP");
    } catch (failed) {
      request = false;
    }
  }
}
if (!request)
  alert("Error initializing XMLHttpRequest!");
function getCustomerInfo() {
  var phone = document.getElementById("phone").value;
  var url = "/cgi-local/lookupCustomer.php?phone=" + escape(phone);
  request.open("GET", url, true);
  request.onreadystatechange= updatePage;
  request.send(null);
}
function updatePage() {
  alert("Server is done!");
```

}

</script>

它仅仅发出一些简单的警告，告诉您服务器什么时候完成了任务。在自己的网页中试验这些代码，然后在浏览器中打开（如果希望查看该例中的 XHTML，请参阅 清单 8）。输入电话号码然后离开该字段，将看到一个弹出的警告窗口（如 图 3 所示），但是点击 OK 又出现了……

图 3. 弹出警告的 Ajax 代码



根据浏览器的不同，在表单停止弹出警告之前会看到两次、三次甚至四次警告。这是怎么回事呢？原来我们还没有考虑 HTTP 就绪状态，这是请求/响应循环中的一个重要部分。

HTTP 就绪状态

前面提到，服务器在完成请求之后会在 XMLHttpRequest 的 onreadystatechange 属性中查找要调用的方法。这是真的，但还不完整。事实上，每当 HTTP 就绪状态改变时它都会调用该方法。这意味着什么呢？首先必须理解 HTTP 就绪状态。

HTTP 就绪状态表示请求的状态或情形。它用于确定该请求是否已经开始、是否得到了响应或者请求/响应模型是否已经完成。它还可以帮助确定读取服务器提供的响应文本或数据是否安全。在 Ajax 应用程序中需要了解五种就绪状态：

·0：请求没有发出（在调用 open() 之前）。

·1：请求已经建立但还没有发出（调用 send() 之前）。

·2：请求已经发出正在处理之中（这里通常可以从响应得到内容头部）。

·3：请求已经处理，响应中通常有部分数据可用，但是服务器还没有完成响应。

·4：响应已完成，可以访问服务器响应并使用它。

与大多数跨浏览器问题一样，这些就绪状态的使用也不尽一致。您也许期望任务就绪状态从 0 到 1、2、3 再到 4，但实际上很少是这种情况。一些浏览器从不报告 0 或 1 而直接从 2 开始，然后是 3 和 4。其他浏览器则报告所有的状态。还有一些则多次报告就绪状态 1。在上一节中看到，服务器多次调用 updatePage()，每次调用都会弹出警告框 —— 可能和预期的不同！

对于 Ajax 编程，需要直接处理的惟一状态就是就绪状态 4，它表示服务器响应已经完成，可以安全地使用响应数据了。基于此，回调方法中的第一行应该如 清单 13 所示。

清单 13. 检查就绪状态

```
function updatePage() {
  if (request.readyState == 4)
    alert("Server is done!");
}
```

修改后就可以保证服务器的处理已经完成。尝试运行新版本的 Ajax 代码，现在就会看到与预期的一样，只显示一次警告信息了。

HTTP 状态码

虽然 清单 13 中的代码看起来似乎不错，但是还有一个问题 —— 如果服务器响应请求并完成了处理但是报告了一个错误怎么办？要知道，服务器端代码应该明白它是由 Ajax、JSP、普通 HTML 表单或其他类型的代码调用的，但只能使用传统的 Web 专用方法报告信息。而在 Web 世界中，HTTP 代码可以处理请求中可能发生的各种问题。

比方说，您肯定遇到过输入了错误的 URL 请求而得到 404 错误码的情形，它表示该页面不存在。这仅仅是 HTTP 请求能够收到的众多错误码中的一种（完整的状态码列表请参阅 参考资料 中的链接）。表示所访问数据受到保护或者禁止访问的 403 和 401 也很常见。无论哪种情况，这些错误码都是从完成的响应 得到的。换句话说，服务器履行了请求（即 HTTP 就绪状态是 4）但是没有返回客户机预期的数据。

因此除了就绪状态外，还需要检查 HTTP 状态。我们期望的状态码是 200，它表示一切顺利。如果就绪状态是 4 而且状态码是 200，就可以处理服务器的数据了，而且这些数据应该就是要求的数据（而不是错误或者其他有问题的信息）。因此还要在回调方法中增加状态检查，如 清单 14 所示。

清单 14. 检查 HTTP 状态码

```
function updatePage() {
 if (request.readyState == 4)
  if (request.status == 200)
   alert("Server is done!");
}
```

为了增加更健壮的错误处理并尽量避免过于复杂，可以增加一两个状态码检查，请看一看 清单 15 中修改后的 updatePage() 版本。

清单 15. 增加一点错误检查

```
function updatePage() {
 if (request.readyState == 4)
  if (request.status == 200)
   alert("Server is done!");
  else if (request.status == 404)
   alert("Request URL does not exist");
  else
   alert("Error: status code is " + request.status);
}
```

现在将 getCustomerInfo() 中的 URL 改为不存在的 URL 看看会发生什么。应该会看到警告信息说明要求的 URL 不存在 —— 好极了！很难处理所有的错误条件，但是这一小小的改变能够涵盖典型 Web 应用程序中 80% 的问题。

读取响应文本

现在可以确保请求已经处理完成（通过就绪状态），服务器给出了正常的响应（通过状态码），最后我们可以处理服务器返回的数据了。返回的数据保存在 XMLHttpRequest 对象的 responseText 属性中。

关于 responseText 中的文本内容，比如格式和长度，有意保持含糊。这样服务器就可以将文本设置成任何内容。比方说，一种脚本可能返回逗号分隔的值，另一种则使用管道符（即 | 字符）分隔的值，还有一种则返回长文本字符串。何去何从由服务器决定。

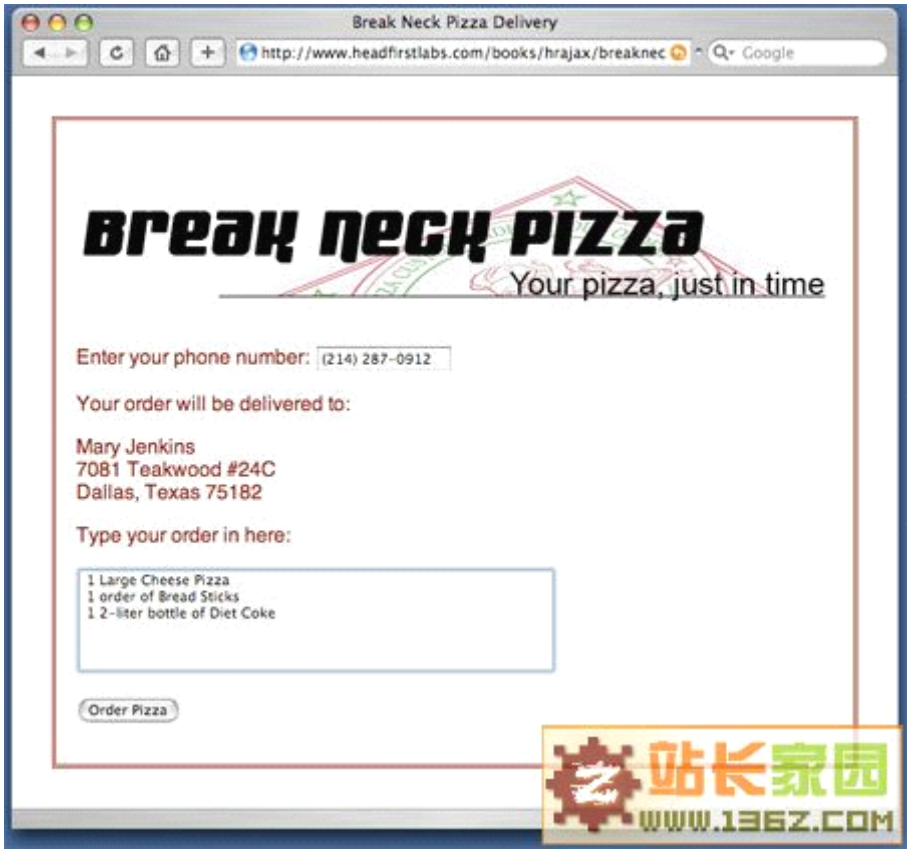
在本文使用的例子中，服务器返回客户的上一个订单和客户地址，中间用管道符分开。然后使用订单和地址设置表单中的元素值，清单 16 给出了更新显示内容的代码。

清单 16. 处理服务器响应

```
function updatePage() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      var response = request.responseText.split("|");
      document.getElementById("order").value = response[0];
      document.getElementById("address").innerHTML =
        response[1].replace(/\n/g, "");
    } else
      alert("status is " + request.status);
  }
}
```

首先，得到 responseText 并使用 JavaScript split() 方法从管道符分开。得到的数组放到 response 中。数组中的第一个值 —— 上一个订单 —— 用 response[0] 访问，被设置为 ID 为 “order” 的字段的值。第二个值 response[1]，即客户地址，则需要更多一点处理。因为地址中的行用一般的行分隔符（“\n”字符）分隔，代码中需要用 XHTML 风格的行分隔符
 来代替。替换过程使用 replace() 函数和正则表达式完成。最后，修改后的文本作为 HTML 表单 div 中的内部 HTML。结果就是表单突然用客户信息更新了，如图 4 所示。

图 4. 收到客户数据后的 Break Neck 表单



结束本文之前，我还要介绍 XMLHttpRequest 的另一个重要属性 responseXML。如果服务器选择使用 XML 响应则该属性包含（也许您已经猜到）XML 响应。处理 XML 响应和处理普通文本有很大不同，涉及到解析、文档对象模型（DOM）和其他一些问题。后面的文章中将进一步介绍 XML。但是因为 responseXML 通常和 responseText 一起讨论，这里有必要提一提。对于很多简单的 Ajax 应用程序 responseText 就够了，但是您很快就会看到通过 Ajax 应用程序也能很好地处理 XML。

结束语

您可能对 XMLHttpRequest 感到有点厌倦了，我很少看到一整篇文章讨论一个对象，特别是这种简单的对象。但是您将在使用 Ajax 编写的每个页面和应用程序中反复使用该对象。坦白地说，关于 XMLHttpRequest 还真有一些可说的内容。下一期文章中将介绍如何在请求中使用 POST 及 GET，来设置请求中的内容头部和从服务器响应读取内容头部，理解如何在请求/响应模型中编码请求和处理 XML。再往后我们将介绍常见 Ajax 工具箱。这些工具箱实际上隐藏了本文所述的很多细节，使得 Ajax 编程更容易。您也许会想，既然有这么多工具箱为何还要对底层的细节编码。答案是，如果不知道应用程序在做什么，就很难发现应用程序中的问题。

因此不要忽略这些细节或者简单地浏览一下，如果便捷华丽的工具箱出现了错误，您就不必挠头或者发送邮件请求支持了。如果了解如何直接使用 XMLHttpRequest，就会发现很容易调试和解决最奇怪的问题。只有让其解决您的问题，工具箱才是好东西。

因此请熟悉 XMLHttpRequest 吧。事实上，如果您有使用工具箱的 Ajax 代码，可以尝试使用 XMLHttpRequest 对象及其属性和方法重新改写。这是一种不错的练习，可以帮助您更好地理解其中的原理。

下一期文章中将进一步讨论该对象，探讨它的一些更有趣的属性（如 responseXML），以及如何使用 POST 请求和以不同的格式发送数据。请开始编写代码吧，一个月后我们再继续讨论。

学习Ajax教程第三天Ajax 中的高级请求和响应

对于很多 Web 开发人员来说，只需要生成简单的请求并接收简单的响应即可；但是对于希望掌握 Ajax 的开发人员来说，必须要全面理解 HTTP 状态代码、就绪状态和 XMLHttpRequest 对象。在本文中，Brett McLaughlin 将向您介绍各种状态代码，并展示浏览器如何对其进行处理，本文还给出了在 Ajax 中使用的比较少见的 HTTP 请求。

在本系列的 上篇文章 中，我们将详细介绍 XMLHttpRequest 对象，它是 Ajax 应用程序的中心，负责处理服务器端应用程序和脚本的请求，并处理从服务器端组件返回的数据。由于所有的 Ajax 应用程序都要使用 XMLHttpRequest 对象，因此您可能会希望熟悉这个对象，从而能够让 Ajax 执行得更好。

在本文中，我将在上一篇文章的基础上重点介绍这个请求对象的 3 个关键部分的内容：

·HTTP 就绪状态

·HTTP 状态代码

·可以生成的请求类型

这三部分内容都是在构造一个请求时所要考虑的因素；但是介绍这些主题的内容太少了。然而，如果您不仅仅是想了解 Ajax 编程的常识，而是希望了解更多内容，就需要熟悉就绪状态、状态代码和请求本身的内容。当应用程序出现问题时 —— 这种问题总是存在 —— 那么如果能够正确理解就绪状态、如何生成一个 HEAD 请求或者 400 的状态代码的确切含义，就可以在 5 分钟内调试出问题，而不是在各种挫折和困惑中度过 5 个小时。

下面让我们首先来看一下 HTTP 就绪状态。

深入了解 HTTP 就绪状态

您应该还记得在上一篇文章中 XMLHttpRequest 对象有一个名为 readyState 的属性。这个属性确保服务器已经完成了一个请求，通常会使用一个回调函数从服务器中读出数据来更新 Web 表单或页面的内容。清单 1 给出了一个简单的例子（这也是本系列的上一篇文章中的一个例子 —— 请参见 参考资料）。

XMLHttpRequest 或 XMLHttp：换名玫瑰

Microsoft™ 和 Internet Explorer 使用了一个名为 XMLHttp 的对象，而不是 XMLHttpRequest 对象，而 Mozilla、Opera、Safari 和 大部分非 Microsoft 浏览器都使用的是后者。为了简单性起见，我将这两个对象都简单地称为 XMLHttpRequest。这既符合我们在 Web 上看到的情况，又符合 Microsoft 在 Internet Explorer 7.0 中使用 XMLHttpRequest 作为请求对象的意图。（有关这个问题的更多内容，请参见 第 2 部分。）

清单 1. 在回调函数中处理服务器的响应

```
function updatePage() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      var response = request.responseText.split("|");
      document.getElementById("order").value = response[0];
      document.getElementById("address").innerHTML =
        response[1].replace(/\n/g, "<br />");
    } else
      alert("status is " + request.status);
  }
}
```

这显然是就绪状态最常见（也是最简单）的用法。正如您从数字 "4" 中可以看出的一样，还有其他几个就绪状态（您在上一篇文章中也看到过这个清单 —— 请参见 参考资料）：

·0：请求未初始化（还没有调用 open()）。

·1：请求已经建立，但是还没有发送（还没有调用 send()）。

·2：请求已发送，正在处理中（通常现在可以从响应中获取内容头）。

·3：请求在处理中；通常响应中已有部分数据可用了，但是服务器还没有完成响应的生成。

·4：响应已完成；您可以获取并使用服务器的响应了。

如果您希望不仅仅是了解 Ajax 编程的基本知识，那么就不但需要知道这些状态，了解这些状态是何时出现的，以及如何来使用这些状态。首先，您需要学习在每种就绪状态下可能碰到的是哪种请求状态。不幸的是，这一点并不直观，而且会涉及几种特殊的情况。

隐秘就绪状态

第一种就绪状态的特点是 readyState 属性为 0（readyState == 0），表示未初始化状态。一旦对请求对象调用 open() 之后，这个属性就被设置为 1。由于您通常都是在一对请求进行初始化之后就立即调用 open()，因此很少会看到 readyState== 0 的状态。另外，未初始化的就绪状态在实际的应用程序中是没有真正的用处的。

不过为了满足我们的兴趣，请参见 清单 2 的内容，其中显示了如何在 readyState 被设置为 0 时来获取这种就绪状态。

清单 2. 获取 0 就绪状态

```
function getSalesData() {
  // Create a request object
  createRequest();
  alert("Ready state is: " + request.readyState);
  // Setup (initialize) the request
  var url = "/boards/servlet/UpdateBoardSales";
  request.open("GET", url, true);
  request.onreadystatechange= updatePage;
  request.send(null);
}
```

在这个简单的例子中，getSalesData() 是 Web 页面调用来启动请求（例如点击一个按钮时）所使用的函数。注意您必须在调用 open()之前 来查看就绪状态。图 1 给出了运行这个应用程序的结果。

图 1. 就绪状态 0

显然，这并不能为您带来多少好处；需要确保 尚未 调用 open() 函数的情况很少。在大部分 Ajax 编程的真实情况中，这种就绪状态的唯一用法就是使用相同的 XMLHttpRequest 对象在多个函数之间生成多个请求。在这种（不常见的）情况中，您可能会在生成新请求之前希望确保请求对象是处于未初始化状态（readyState == 0）。这实际上是要确保另外一个函数没有同时使用这个对象。

查看正在处理的请求的就绪状态

除了 0 就绪状态之外，请求对象还需要依次经历典型的请求和响应的其他几种就绪状态，最后才以就绪状态 4 的形式结束。这就是为什么您在大部分回调函数中都可以看到 if (request.readyState == 4) 这行代码；它确保服务器已经完成对请求的处理，现在可以安全地更新 Web 页面或根据从服务器返回来的数据来进行操作了。

要查看这种状态发生的过程非常简单。如果就绪状态为 4，我们不仅要运行回调函数中的代码，而且还要在每次调用回调函数时都输出就绪状态。 清单 3 给出了一个实现这种功能的例子。

当 0 等于 4 时

在多个 JavaScript 函数都使用相同的请求对象时，您需要检查就绪状态 0 来确保这个请求对象没有正在使用，这种机制会产生问题。由于 readyState == 4 表示一个已完成的请求，因此您经常会发现那些目前没在使用的处于就绪状态的请求对象仍然被设置成了 4 —— 这是因为从服务器返回来的数据已经使用过了，但是从它们被设置为就绪状态之后就没有进行任何变化。有一个函数 abort() 会重新设置请求对象，但是这个函数却不是真正为了这个目的而使用的。如果您 必须 使用多个函数，最好是为每个函数都创建并使用一个函数，而不是在多个函数之间共享相同的对象。

清单 3. 查看就绪状态

```
function updatePage() {
  // Output the current ready state
  alert("updatePage() called with ready state of " + request.readyState);
}
```

如果您不确定如何运行这个函数，就需要创建一个函数，然后在 Web 页面中调用这个函数，并让它向服务器端的组件发送一个请求（例如 清单 2 给出的函数，或本系列文章的第 1 部分和第 2 部分中给出的例子）。确保在建立请求时，将回调函数设置为 updatePage()；要实现这种设置，可以将请求对象的 onreadystatechange 属性设置为 updatePage()。

这段代码就是 onreadystatechange 意义的一个确切展示 —— 每次请求的就绪状态发生变化时，就调用 updatePage()，然后我们就可以看到一个警告了。图 2 给出了一个调用这个函数的例子，其中就绪状态为 1。

图 2. 就绪状态 1

您可以自己尝试运行这段代码。将其放入 Web 页面中，然后激活事件处理程序（单击按钮，在域之间按 tab 键切换焦点，或者使用设置的任何方法来触发请求）。这个回调函数会运行多次 —— 每次就绪状态都会改变 —— 您可以看到每个就绪状态的警告。这是跟踪请求所经历的各个阶段的最好方法。

浏览器的不一致性

在对这个过程有一个基本的了解之后，请试着从几个不同的浏览器中访问您的页面。您应该会注意到各个浏览器如何处理这些就绪状态并不一致。例如，在 Firefox 1.5 中，您会看到以下就绪状态：

·1

·2

·3

·4

这并不奇怪，因为每个请求状态都在这里表示出来了。然而，如果您使用 Safari 来访问相同的应用程序，就应该看到 —— 或者看不到 —— 一些有趣的事情。下面是在 Safari 2.0.1 中看到的状态：

·2

·3

·4

Safari 实际上把第一个就绪状态给丢弃了，也并没有什么明显的原因说明为什么要这样做；不过这就是 Safari 的工作方式。这还说明了一个重要的问题：尽管在使用服务器上的数据之前确保请求的状态为 4 是一个好主意，但是依赖于每个过渡期就绪状态编写的代码的确会在不同的浏览器上得到不同的结果。

例如，在使用 Opera 8.5 时，所显示的就绪状态情况就更加糟糕了：

·3

·4

最后，Internet Explorer 会显示如下状态：

·1

·2

·3

·4

如果您碰到请求方面的问题，这就是用来发现问题的 首要之处。最好的方式是在 Internet Explorer 和 Firefox 都进行一下测试 —— 您会看到所有这 4 种状态，并可以检查请求的每个状态所处的情况。

接下来我们再来看一下响应端的情况。

显微镜下的响应 数据

一旦我们理解在请求过程中发生的各 个就绪状态之后，接下来就可以来看一下 XMLHttpRequest 对象的另外一个方面了 —— responseText 属性。回想一下在上一篇文章中我们介绍过的内容，就可以知道这个属性用来从服务器上获取数据。一旦服务器完成对请求的处理之后，就可以将响应请求数据所需要的任何数据放到请求的 responseText 中了。然后回调函数就可以使用这些数据，如 清单 1 和 清单 4 所示。

清单 4. 使用服务器上返回的响应

```
function updatePage() {
  if (request.readyState == 4) {
    var newTotal = request.responseText;
    var totalSoldEl = document.getElementById("total-sold");
    var netProfitEl = document.getElementById("net-profit");
    replaceText(totalSoldEl, newTotal);
    /* 图 out the new net profit */
    var boardCostEl = document.getElementById("board-cost");
    var boardCost = getText(boardCostEl);
    var manCostEl = document.getElementById("man-cost");
    var manCost = getText(manCostEl);
    var profitPerBoard = boardCost - manCost;
    var netProfit = profitPerBoard * newTotal;
```

```
        /* Update the net profit on the sales form */
        netProfit = Math.round(netProfit * 100) / 100;
        replaceText(netProfitEl, netProfit);
    }
```

清单 1 相当简单；清单 4 稍微有点复杂，但是它们在开始时都要检查就绪状态，并获取 responseText 属性的值。

查看请求的响应文本

与就绪状态类似，responseText 属性的值在整个请求的生命周期中也会发生变化。要查看这种变化，请使用如 清单 5 所示的代码来测试请求的响应文本，以及它们的就绪状态。

清单 5. 测试 responseText 属性

```
  function updatePage() {
    // Output the current ready state
    alert("updatePage() called with ready state of " + request.readyState +
        " and a response text of '" + request.responseText + "'");
    }
```

现在在浏览器中打开 Web 应用程序，并激活您的请求。要更好地看到这段代码的效果，请使用 Firefox 或 Internet Explorer，因为这两个浏览器都可以报告出请求过程中所有可能的就绪状态。例如在就绪状态 2 中，就没有定义 responseText （请参见 图 3）；如果 JavaScript 控制台也已经打开了，您就会看到一个错误。

图 3. 就绪状态为 2 的响应文本

不过在就绪状态 3 中，服务器已经在 responseText 属性中放上了一个值，至少在这个例子中是这样（请参见 图 4）。

图 4. 就绪状态为 3 的响应文本

您会看到就绪状态为 3 的响应在每个脚本、每个服务器甚至每个浏览器上都是不一样的。不过，这在调试应用程序中依然是非常有用的。

获取安全数据

所有的文档和规范都强调，只有在就绪状态为 4 时数据才可以安全使用。相信我，当就绪状态为 3 时，您很少能找到无法从 responseText 属性获取数据的情况。然而，在应用程序中将自己的逻辑依赖于就绪状态 3 可不是什么好主意 —— 一旦您编写了依赖于就绪状态 3 的完整数据的的代码，几乎就要自己来负责当时的数据不完整问题了。

比较好的做法是向用户提供一些反馈，说明在处于就绪状态 3 时，很快就会有响应了。尽管使用 alert() 之类的函数显然不是什么好主意 —— 使用 Ajax 然后使用一个警告对话框来阻塞用户显然是错误的 —— 不过您可以在就绪状态发生变化时更新表单或页面中的域。例如，对于就绪状态 1 来说要将进度指示器的宽度设置为 25%，对于就绪状态 2 来说要将进度指示器的宽度设置为 50%，对于就绪状态 3 来说要将进度指示器的宽度设置为 75%，当就绪状态为 4 时将进度指示器的宽度设置为 100%（完成）。

当然，正如您已经看到的一样，这种方法非常聪明，但它是依赖于浏览器的。在 Opera 上，您永远都不会看到前两个就绪状态，而在 Safari 上则没有第一个（1）。由于这个原因，我将这段代码留作练习，而没有在本文中包括进来。

现在应该来看一下状态代码了。

深入了解 HTTP 状态代码

有了就绪状态和您在 Ajax 编程技术中学习到的服务器的响应，您就可以为 Ajax 应用程序添加另外一级复杂性了 —— 这要使用 HTTP 状态代码。这些代码对于 Ajax 来说并没有什么新鲜。从 Web 出现以来，它们就已经存在了。在 Web 浏览器中您可能已经看到过几个状态代码：

·401：未经授权

·403：禁止

·404：没找到

您可以找到更多的状态代码（完整清单请参见 参考资料）。要为 Ajax 应用程序另外添加一层控制和响应（以及更为健壮的错误处理）机制，您需要适当地查看请求和响应中的状态代码。

200：一切正常

在很多 Ajax 应用程序中，您将看到一个回调函数，它负责检查就绪状态，然后继续利用从服务器响应中返回的数据，如 清单 6 所示。

清单 6. 忽略状态代码的回调函数

```
function updatePage() {
  if (request.readyState == 4) {
    var response = request.responseText.split("|");
    document.getElementById("order").value = response[0];
    document.getElementById("address").innerHTML =
      response[1].replace(/\n/g, "<br />");
```

```
    }
  }
```

这对于 Ajax 编程来说证明是一种短视而错误的方法。如果脚本需要认证，而请求却没有提供有效的证书，那么服务器就会返回诸如 403 或 401 之类的错误代码。然而，由于服务器对请求进行了应答，因此就绪状态就被设置为 4（即使应答并不是请求所期望的也是如此）。最终，用户没有获得有效数据，当 JavaScript 试图使用不存在的服务器数据时就可能会出现严重的错误。

它花费了最小的努力来确保服务器不但完成了一个请求，而且还返回了一个“一切良好”的状态代码。这个代码是 "200"，它是通过 XMLHttpRequest 对象的 status 属性来报告的。为了确保服务器不但完成了一个请求，而且还报告了一个 OK 状态，请在您的回调函数中添加另外一个检查功能，如 清单 7 所示。

清单 7. 检查有效状态代码

```
  function updatePage() {
    if (request.readyState == 4) {
      if (request.status == 200) {
        var response = request.responseText.split("|");
        document.getElementById("order").value = response[0];
        document.getElementById("address").innerHTML =
          response[1].replace(/\n/g, "<br />");
      } else
        alert("status is " + request.status);
    }
  }
```

通过添加这几行代码，您就可以确认是否存在问题，用户会看到一个有用的错误消息，而不仅仅是看到一个由断章取义的数据所构成的页面，而没有任何解释。

重定向和重新路由

在深入介绍有关错误的内容之前，我们有必要来讨论一下有关一个在使用 Ajax 时 并不需要 关心的问题 —— 重定向。在 HTTP 状态代码中，这是 300 系列的状态代码，包括：

·301：永久移动

·302：找到（请求被重新定向到另外一个 URL/URI 上）

·305：使用代理（请求必须使用一个代理来访问所请求的资源）

Ajax 程序员可能并不太关心有关重定向的问题，这是由于两方面的原因：

·首先，Ajax 应用程序通常都是为一个特定的服务器端脚本、servlet 或应用程序而编写的。对于那些您看不到就消失了的组件来说，Ajax 程序员就不太清楚了。因此有时您会知道资源已经移动了（因为您移动了它，或者通过某种手段移动了它），接下来要修改请求中的 URL，并且不会再碰到这种结果了。

更为重要的一个原因是：Ajax 应用程序和请求都是封装在沙盒中的。这就意味着提供生成 Ajax 请求的 Web 页面的域必须是对这些请求进行响应的域。因此 ebay.com 所提供的 Web 页面就不能对一个在 amazon.com 上运行的脚本生成一个 Ajax 风格的请求；在 ibm.com 上的 Ajax 应用程序也无法对在 netbeans.org 上运行的 servlets 发出请求。

·结果是您的请求无法重定向到其他服务器上，而不会产生安全性错误。在这些情况中，您根本就不会得到状态代码。通常在调试控制台中都会产生一个 JavaScript 错误。因此，在对状态代码进行充分的考虑之后，您就可以完全忽略重定向代码的问题了。

结果是您的请求无法重定向到其他服务器上，而不会产生安全性错误。在这些情况中，您根本就不会得到状态代码。通常在调试控制台中都会产生一个 JavaScript 错误。因此，在对状态代码进行充分的考虑之后，您就可以完全忽略重定向代码的问题了。

错误

一旦接收到状态代码 200 并且意识到可以很大程度上忽略 300 系列的状态代码之后，所需要担心的唯一一组代码就是 400 系列的代码了，这说明了不同类型的错误。回头再来看一下 清单 7，并注意在对错误进行处理时，只将少数常见的错误消息输出给用户了。尽管这是朝正确方向前进的一步，但是要告诉从事应用程序开发的用户和程序员究竟发生了什么问题，这些消息仍然是没有太大用处的。

首先，我们要添加对找不到的页的支持。实际上这在大部分产品系统中都不应该出现，但是在测试脚本位置发生变化或程序员输入了错误的 URL 时，这种情况并不罕见。如果您可以自然地报告 404 错误，就可以为那些困扰不堪的用户和程序员提供更多帮助。例如，如果服务器上的一个脚本被删除了，我们就可以使用 清单 7 中的代码，这样用户就会看到一个如 图 5 所示的非描述性错误。

边界情况和困难情况

看到现在，一些新手程序员就可能会这究竟是要讨论什么内容。有一点事实大家需要知道：只有不到 5% 的 Ajax 请求需要使用诸如 2、3 之类的就绪状态和诸如 403 之类的状态代码（实际上，这个比率可能更接近于 1% 甚至更少）。这些情况非常重要，称为 边界情况

（edge case） —— 它们只会在一些非常特殊的情况下发生，其中遇到的都是最奇特的问题。虽然这些情况并不普遍，但是这些边界情况却占据了大部分用户所碰到的问题的 80%!

对于典型的用户来说，应用程序 100 次都是正常工作的这个事实通常都会被忘记，然而应用程序只要一次出错就会被他们清楚地记住。如果您可以很好地处理边界情况（或困难情况），就可以为再次访问站点的用户提供满意的回报。

图 5. 常见错误处理

用户无法判断问题究竟是认证问题、没找到脚本（此处就是这种情况）、用户错误还是代码中有些地方产生了问题。添加一些简单的代码可以让这个错误更加具体。请参照 清单 8，它负责处理没找到的脚本或认证发生错误的情况，在出现这些错误时都会给出具体的消息。

清单 8. 检查有效状态代码

```
function updatePage() {
 if (request.readyState == 4) {
  if (request.status == 200) {
   var response = request.responseText.split("|");
   document.getElementById("order").value = response[0];
   document.getElementById("address").innerHTML =
    response[1].replace(/\n/g, "<br />");
  } else if (request.status == 404) {
   alert ("Requested URL is not found.");
  } else if (request.status == 403) {
   alert("Access denied.");
  } else
   alert("status is " + request.status);
 }
}
```

虽然这依然相当简单，但是它的确多提供了一些有用的信息。图 6 给出了与 图 5 相同的错误，但是这一次错误处理代码向用户或程序员更好地说明了究竟发生了什么。

图 6. 特殊错误处理

在我们自己的应用程序中，可以考虑在发生认证失败的情况时清除用户名和密码，并向屏幕上添加一条错误消息。我们可以使用类似的方法来更好地处理找不到脚本或其他 400 类型的错误（例如 405 表示不允许使用诸如发送 HEAD 请求之类不可接受的请求方法，而 407 则表示需要进行代理认证）。然而不管采用哪种选择，都需要从对服务器上返回的状态代码开始入手进行处理。

其他请求类型

如果您真希望控制 XMLHttpRequest 对象，可以考虑最后实现这种功能 —— 将 HEAD 请求添加到指令中。在前两篇文章中，我们已经介绍了如何生成 GET 请求；在马上就要发表的一篇文章中，您会学习有关使用 POST 请求将数据发送到服务器上的知识。不过本着增强错误处理和信息搜集的精神，您应该学习如何生成 HEAD 请求。

生成请求

实际上生成 HEAD 请求非常简单；您可以使用 "HEAD"（而不是 "GET" 或 "POST"）作为第一个参数来调用 open() 方法，如 清单 9 所示。

清单 9. 使用 Ajax 生成一个 HEAD 请求

```
function getSalesData() {
  createRequest();
  var url = "/boards/servlet/UpdateBoardSales";
  request.open("HEAD", url, true);
  request.onreadystatechange= updatePage;
  request.send(null);
}
```

当您这样生成一个 HEAD 请求时，服务器并不会像对 GET 或 POST 请求一样返回一个真正的响应。相反，服务器只会返回资源的 头（header），这包括响应中内容最后修改的时间、请求资源是否存在和很多其他有用信息。您可以在服务器处理并返回资源之前使用这些信息来了解有关资源的信息。

对于这种请求您可以做的最简单的事情就是简单地输出所有的响应头的内容。这可以让您了解通过 HEAD 请求可以使用什么。清单 10 提供了一个简单的回调函数，用来输出从 HEAD 请求中获得的响应头的内容。

清单 10. 输出从 HEAD 请求中获得的响应头的内容

```
function updatePage() {
 if (request.readyState == 4) {
  alert(request.getAllResponseHeaders());
 }
}
```

请参见 图 7，其中显示了从一个向服务器发出的 HEAD 请求的简单 Ajax 应用程序返回的响应头。

您可以单独使用这些头（从服务器类型到内容类型）在 Ajax 应用程序中提供其他信息或功能。

检查 URL

您已经看到了当 URL 不存在时应该如何检查 404 错误。如果这变成一个常见的问题 —— 可能是缺少了一个特定的脚本或 servlet —— 那么您就可能会希望在生成完整的 GET 或 POST 请求之前来检查这个 URL。要实现这种功能，生成一个 HEAD 请求，然后在回调函数中检查 404 错误；清单 11 给出了一个简单的回调函数。

清单 11. 检查某个 URL 是否存在

```
function updatePage() {
 if (request.readyState == 4) {
  if (request.status == 200) {
   alert("URL exists");
  } else if (request.status == 404) {
   alert("URL does not exist.");
  } else {
   alert("Status is: " + request.status);
  }
 }
}
```

诚实地说，这段代码的价值并不太大。服务器必须对请求进行响应，并构造一个响应来填充内容长度的响应头，因此并不能节省任何处理时间。另外，这花费的时间与生成请求并使用 HEAD 请求来查看 URL 是否存在所需要的时间一样多，因为它要生成使用 GET 或

POST 的请求，而不仅仅是如 清单 7 所示一样来处理错误代码。不过，有时确切地了解目前什么可用也是非常有用的；您永远不会知道何时创造力就会迸发或者何时需要 HEAD 请求！

有用的 HEAD 请求

您会发现 HEAD 请求非常有用的一个领域是用来查看内容的长度或内容的类型。这样可以确定是否需要发回大量数据来处理请求，和服务器是否试图返回二进制数据，而不是 HTML、文本或 XML（在 JavaScript 中，这 3 种类型的数据都比二进制数据更容易处理）。

在这些情况中，您只使用了适当的头名，并将其传递给 XMLHttpRequest 对象的 getResponseHeader() 方法。因此要获取响应的长度，只需要调用 request.getResponseHeader("Content-Length");。要获取内容类型，请使用 request.getResponseHeader("Content-Type");。

在很多应用程序中，生成 HEAD 请求并没有增加任何功能，甚至可能会导致请求速度变慢（通过强制生成一个 HEAD 请求来获取有关响应的数据，然后在使用一个 GET 或 POST 请求来真正获取响应）。然而，在出现您不确定有关脚本或服务器端组件的情况时，使用 HEAD 请求可以获取一些基本的数据，而不需要对响应数据真正进行处理，也不需要大量的带宽来发送响应。

结束语

对于很多 Ajax 和 Web 程序员来说，本文中介绍的内容似乎是太高级了。生成 HEAD 请求的价值是什么呢？到底在什么情况下需要在 JavaScript 中显式地处理重定向状态代码呢？这些都是很好的问题；对于简单的应用程序来说，答案是这些高级技术的价值并不是非常大。然而，Web 已经不再是只需实现简单应用程序的地方了；用户已经变得更加高级，客户期望能够获得更好的稳定性、更高级的错误报告，如果应用程序有 1% 的时间停机，那么经理就可能会因此而被解雇。

因此您的工作就不能仅仅局限于简单的应用程序了，而是需要更深入理解 XMLHttpRequest。

·如果您可以考虑各种就绪状态 —— 并且理解了这些就绪状态在不同浏览器之间的区别 —— 就可以快速调试应用程序了。您甚至可以基于就绪状态而开发一些创造性的功能，并向用户和客户回报请求的状态。

·如果您要对状态代码进行控制，就可以设置应用程序来处理脚本错误、非预期的响应以及边缘情况。结果是应用程序在所有的时间都可以正常工作，而不仅仅是只能一切都正常的情况下才能运行。

·增加这种生成 HEAD 请求的能力，检查某个 URL 是否存在，以及确认某个文件是否被修改过，这样就可以确保用户可以获得有效的页面，用户所看到的信息都是最新的，（最重要的是）让他们惊讶这个应用程序是如何健壮和通用。

本文的目的并非是要让您的应用程序显得十分华丽，而是帮助您去掉黄色聚光灯后重点昭显文字的美丽，或者外观更像桌面一样。尽管这些都是 Ajax 的功能（在后续几篇文章中就会介绍），不过它们却像是蛋糕表面的一层奶油。如果您可以使用 Ajax 来构建一个坚实

的基础，让应用程序可以很好地处理错误和问题，用户就会返回您的站点和应用程序。在接下来的文章中，我们将添加这种直观的技巧，这会让客户兴奋得发抖。（认真地说，您一定不希望错过下一篇文章！）

学习Ajax教程第四天，利用 DOM 进行 Web 响应

在 HTML 的外部文件中一样，标记的组织与其样式、格式和行为是分离的。虽然您肯定可以用 JavaScript 更改元素或文本的样式，但实际更改您的标记所布置的组织却更加有趣。

只要牢记您的标记只为您的页面提供组织、框架，您就能立于不败之地。再前进一小步，您就会明白浏览器是如何接受所有的文本组织并将其转变为超级有趣的一些东西的，即一组对象，其中每个对象都可被更改、添加或删除。

文本标记的优点

在讨论 Web 浏览器之前，值得考虑一下为什么纯文本绝对是存储 HTML 的最佳选择（有关详细信息，请参阅 有关标记的一些其他想法）。不考虑优缺点，只是回忆一下在每次查看页面时 HTML 是通过网络发送到 Web 浏览器的（为了简洁，不考虑高速缓存等）。真是再没有比传递文本再有效的方法了。二进制对象、页面图形表示、重新组织的标记块等等，所有这一切都比纯文本文件通过网络传递要更困难。

此外，浏览器也为此增光添彩。今天的浏览器允许用户更改文本大小、按比例伸缩图像、下载页面的 CSS 或 JavaScript（大多数情况），甚至更多，这完全排除了将任何类型的页面图形表示发送到浏览器上。但是，浏览器需要原 HTML，这样它才能在浏览器中对页面应用任何处理，而不是信任浏览器去处理该任务。同样地，将 CSS 从 JavaScript 分离和将 CSS 从 HTML 标记分离要求一种容易分离的格式。文本文件又一次成为该任务的最好方法。

最后但同样重要的一点是，记住，新标准（比如 HTML 4.01 与 XHTML 1.0 和 1.1）承诺将内容（页面中的数据）与表示和样式（通常由 CSS 应用）分离。如果程序员要将 HTML 与 CSS 分离，然后强制浏览器检索粘结页面各部分的一些页面表示，这会失去这些标准的多数优点。保持这些部分到达浏览器时都一直分离使得浏览器在从服务器获取 HTML 时有了前所未有的灵活性。

关于标记的其他想法

纯文本编辑：是对是错？

纯文本是存储标记的理想选择，但是不适合编辑 标记。大行其道的是使用 IDE，比如 Macromedia DreamWeaver 或更强势点的 Microsoft® FrontPage®，来操作 Web 页面标记。这些环境通常提供快捷方式和帮助来创建 Web 页面，尤其是在使用 CSS 和 JavaScript

时，二者都来自实际页面标记以外的文件。许多人仍偏爱好用古老的记事本或 vi（我承认我也是其中一员），这并不要紧。不管怎样，最终结果都是充满标记的文本文件。

网络上的文本：好东西

已经说过，文本是文档的最好媒体，比如 HTML 或 CSS，在网络上被千百次地传输。当我说浏览器表示文本很难时，是特指将文本转换为用户查看的可视图形页面。这与浏览器实际上如何从 Web 浏览器检索页面没有关系；在这种情况下，文本仍然是最佳选择。

文本标记的缺点

正如文本标记对于设计人员和页面创建者具有惊人的优点之外，它对于浏览器也具有相当出奇的缺点。具体来说，浏览器很难直接将文本标记可视地表示给用户（详细信息请参阅 有关标记的一些其他想法）。考虑下列常见的浏览器任务：

·基于元素类型、类、ID 及其在 HTML 文档中的位置，将 CSS 样式（通常来自外部文件中的多个样式表）应用于标记。

·基于 JavaScript 代码（通常位于外部文件）将样式和格式应用于 HTML 文档的不同部分。

·基于 JavaScript 代码更改表单字段的值。

·基于 JavaScript 代码，支持可视效果，比如图像翻转和图像交换。

复杂性并不在于编码这些任务；其中每件事都是相当容易的。复杂性来自实际实现请求动作的浏览器。如果标记存储为文本，比如，想要在 center-text 类的 p 元素中输入文本 (text-align: center)，如何实现呢？

·将内联样式添加到文本吗？

·将样式应用到浏览器中的 HTML 文本，并只保持内容居中或不居中？

·应用无样式的 HTML，然后事后应用格式？

这些非常困难的问题是如今很少有人编写浏览器的原因。（编写浏览器的人应该接受最由衷的感谢）

无疑，纯文本不是存储浏览器 HTML 的好办法，尽管文本是获取页面标记最好的解决方案。如果加上 JavaScript 更改 页面结构的能力，事情就变得有些微妙了。浏览器应该将修改过的结构重新写入磁盘吗？如何才能保持文档的最新版本呢？

无疑，文本不是答案。它难以修改，为其应用样式和行为很困难，与今天 Web 页面的动态本质在根本上相去甚远。

求助于树视图

这个问题的答案（至少是由当今 Web 浏览器选择的答案）是使用树结构来表示 HTML。参见 清单 1，这是一个表示为本文标记的相当简单又无聊的 HTML 页面。

清单 1. 文本标记中的简单 HTML 页面

```
<html>
 <head>
  <title>Trees, trees, everywhere</title>
 </head>
 <body>
  <h1>Trees, trees, everywhere</h1>
  <p>Welcome to a <em>really</em> boring page.</p>
  <div>
   Come again soon.
   <img src="/images/NullPic.gif" />
  </div>
 </body>
</html>
```

浏览器接受该页面并将之转换为树形结构，如图 1 所示。

为了保持本文的进度，我做了少许简化。DOM 或 XML 方面的专家会意识到空白对于文档文本在 Web 浏览器树结构中表示和分解方式的影响。肤浅的了解只会使事情变得模棱两可，所以如果想弄清空白的影响，那最好不过了；如果不想的话，那可以继续读下去，不要考虑它。当它成为问题时，那时您就会明白您需要的一切。

除了实际的树背景之外，可能会首先注意到树中的一切是以最外层的 HTML 包含元素，即 html 元素开始的。使用树的比喻，这叫做根元素。所以即使这是树的底层，当您查看并分析树的时候，我也通常以此开始。如果它确实奏效，您可以将整个树颠倒一下，但这确实有些拓展了树的比喻。

从根流出的线表示不同标记部分之间的关系。head 和 body 元素是 html 根元素的孩子；title 是 head 的孩子，而文本 “Trees, trees, everywhere” 是 title 的孩子。整个树就这样组织下去，直到浏览器获得与 图 1 类似的结构。

一些附加术语

为了沿用树的比喻，head 和 body 被叫做 html 的分支（branches）。叫分支是因为它们有自己的孩子。当到达树的末端时，您将进入主要的文本，比如 “Trees, trees, everywhere” 和 “really”；这些通常称为叶子，因为它们没有自己的孩子。您不需要记住所有这些术语，当您试图弄清楚特定术语的意思时，只要想像一下树结构就容易多了。

对象的值

既然了解了一些基本的术语，现在应该关注一下其中包含元素名称和文本的小矩形了（图 1）。每个矩形是一个对象；浏览器在其中解决一些文本问题。通过使用对象来表示 HTML 文档的每一部分，可以很容易地更改组织、应用样式、允许 JavaScript 访问文档，等等。

对象类型和属性

标记的每个可能类型都有自己的对象类型。例如，HTML 中的元素用 Element 对象类型表示。文档中的文本用 Text 类型表示，属性用 Attribute 类型表示，以此类推。

所以 Web 浏览器不仅可以使用对象模型来表示文档（从而避免了处理静态文本），还可以用对象类型立即辨别出某事物是什么。HTML 文档被解析并转换为对象集合，如图 1 所示，然后尖括号和转义序列（例如，使用 < 表示 <，使用 > 表示 >）等事物不再是问题了。这就使得浏览器的工作（至少在解析输入 HTML 之后）变得更容易。弄清某事物究竟是元素还是属性并确定如何处理该类型的对象，这些操作都十分简单了。

通过使用对象，Web 浏览器可以更改这些对象的属性。例如，每个元素对象具有一个父元素和一系列子元素。所以添加新的子元素或文本只需要向元素的子元素列表中添加一个新的子元素。这些对象还具有 style 属性，所以快速更改元素或文本段的样式非常简单。例如，要使用 JavaScript 更改 div 的高度，如下所示：

```
someDiv.style.height = "300px";
```

换句话说，Web 浏览器使用对象属性可以非常容易地更改树的外观和结构。将之比作浏览器在内部将页面表示为文本时必须进行的复杂事情，每次更改属性或结构都需要浏览器重新编写静态文件、重新解析并在屏幕上重新显示。有了对象，所有这一切都解决了。

现在，花点时间展开一些 HTML 文档并用树将其勾画出来。尽管这看起来是个不寻常的请求（尤其是在包含极少代码的这样一篇文章中），如果您希望能够操纵这些树，那么需要熟悉它们的结构。

在这个过程中，可能会发现一些古怪的事情。比如，考虑下列情况：

·属性发生了什么？

·分解为元素（比如 em 和 b）的文本呢？

·结构不正确（比如当缺少结束 p 标记时）的 HTML 呢？

一旦熟悉这些问题之后，就能更好地理解下面几节了。

严格有时是好事

如果尝试刚提到的练习 I，您可能会发现标记的树视图中存在一些潜在问题（如果不练习的话，那就听我说吧！）。事实上，在 清单 1 和 图 1 中就会发现一些问题，首先看 p 元素是如何分解的。如果您问通常的 Web 开发人员 “p 元素的文本内容是什么”，最常见的答案将是 “Welcome to a really boring Web page.”。如果将之与图 1 做比较，将会发现这个答案（虽然合乎逻辑）是根本不正确的。

实际上，p 元素具有三个 不同的子对象，其中没有一个包含完整的 “Welcome to a really boring Web page.” 文本。您会发现文本的一部分，比如 “Welcome to a ” 和 “ boring Web page”，但不是全部。为了理解这一点，记住标记中的任何内容都必须转换为某种类型的对象。此外，顺序无关紧要！如果浏览器显示正确的对象，但显示顺序与您在 HTML 中提供的顺序不同，那么您能想像出用户将如何响应 Web 浏览器吗？段落夹在页面标题和文章标题中间，而这不是您自己组织文档时的样式呢？很显然，浏览器必须保持元素和文本的顺序。

在本例中，p 元素有三个不同部分：

·em 元素之前的文本

·em 元素本身

·em 元素之后的文本

如果将该顺序打乱，可能会把重点放在文本的错误部分。为了保持一切正常，p 元素有三个子对象，其顺序是在 清单 1 的 HTML 中显示的顺序。而且，重点文本 “really” 不是 p 的子元素；而是 p 的子元素 em 的子元素。

理解这一概念非常重要。尽管 “really” 文本将可能与其他 p 元素文本一起显示，但它仍是 em 元素的直接子元素。它可以具有与其他 p 文本不同的格式，而且可以独立于其他文本到处移动。

要将之牢记在心，试着用图表示清单 2 和 3 中的 HTML，确保文本具有正确的父元素（而不管文本最终会如何显示在屏幕上）。

清单 2. 带有巧妙元素嵌套的标记

```
<html>
 <head>
  <title>This is a little tricky</title>
 </head>
 <body>
  <h1>Pay <u>close</u> attention, OK?</h1>
```

```
  <div>
    <p>This p really isn't <em>necessary</em>, but it makes the
      <span id="bold-text">structure <i>and</i> the organization</span>
      of the page easier to keep up with.</p>
  </div>
 </body>
</html>
```

清单 3. 更巧妙的元素嵌套

```
<html>
 <head>
  <title>Trickier nesting, still</title>
 </head>
 <body>
  <div id="main-body">
   <div id="contents">
    <table>
     <tr><th>Steps</th><th>Process</th></tr>
     <tr><td>1</td><td>Figure out the <em>root element</em>.</td></tr>
     <tr><td>2</td><td>Deal with the <span id="code">head</span> first,
       as it's usually easy.</td></tr>
     <tr><td>3</td><td>Work through the <span id="code">body</span>.
```



```
       Just <em>take your time</em>.</td></tr>
    </table>
   </div>
```

```
  <div id="closing">
    This link is <em>not</em> active, but if it were, the answers
    to this <a href="answers.html"><img src="/images/NullPic.gif" /></a> would
    be there. But <em>do the exercise anyway!</em>
  </div>
 </div>
</body>
</html>
```

在本文末的 GIF 文件 图 2 中的 tricky-solution.gif 和 图 3 中的 trickier-solution.gif 中将会找到这些练习的答案。不要偷看，先花些时间自动解答一下。这样能帮助您理解组织树时应用的规则有多么严格，并真正帮助您掌握 HTML 及其树结构。

属性呢？

当您试图弄清楚如何处理属性时，是否遇到一些问题呢？前已提及，属性确实具有自己的对象类型，但属性确实不是显示它的元素的子元素，嵌套元素和文本不在同一属性“级别”，您将注意到，清单 2 和 3 中练习的答案没有显示属性。

属性事实上存储在浏览器使用的对象模型中，但它们有一些特殊情况。每个元素都有可用属性的列表，且与子对象列表是分离的。所以 div 元素可能有一个包含属性“id”和另一个属性“class”的列表。

记住，元素的属性必须具有惟一的名称，也就是说，一个元素不能有两个“id”或两个“class”属性。这使得列表易于维护和访问。在下一篇文章将会看到，您可以简单调用诸如 getAttribute("id") 的方法来按名称获取属性的值。还可以用相似的方法调用来添加属性或设置（重置）现有属性的值。

值得指出的是，属性名的惟一性使得该列表不同于子对象列表。p 元素可以有多个 em 元素，所以子对象列表可以包含多个重复项。尽管子项列表和属性列表的操作方式相似，但一个可以包含重复项（对象的子项），而一个不能（元素对象的属性）。最后，只有元素具有属性，所以文本对象没有用于存储属性的附加列表。

凌乱的 HTML

在继续之前，谈到浏览器如何将标记转换为树表示，还有一个主题值得探讨，即浏览器如何处理不是格式良好的标记。格式良好 是 XML 广泛使用的一个术语，有两个基本意思：

·每个开始标记都有一个与之匹配的结束标记。所以每个 在文档中与 匹配，每个 与 匹配，等等。

·最里面的开始标记与最里面的结束标记相匹配，然后次里面的开始标记与次里面的结束标记相匹配，依此类推。所以 bold and italics 是不合法的，因为最里面的开始标记 与最里面的结束标记 匹配不当。要使之格式良好，要么 切换开始标记顺序，要么 切换结束标记顺序。（如果两者都切换，则仍会出现问题）。

深入研究这两条规则。这两条规则不仅简化了文档的组织，还消除了不定性。是否应先应用粗体后应用斜体？或恰恰相反？如果觉得这种顺序和不定性不是大问题，那么请记住，CSS 允许规则覆盖其他规则，所以，例如，如果 b 元素中文本的字体不同于 i 元素中的字体，则格式的应用顺序将变得非常重要。因此，HTML 的格式良好性有着举足轻重的作用。

如果浏览器收到了不是格式良好的文档，它只会尽力而为。得到的树结构在最好情况下将是作者希望的原始页面的近似，最坏情况下将面目全非。如果您曾将页面加载到浏览器中后看到完全出乎意料的结果，您可能在看到浏览器结果时会猜想您的结构应该如何，并沮丧地继续工作。当然，搞定这个问题相当简单：确保文档是格式良好的！如果不清楚如何编写标准化的 HTML，请咨询 参考资料 获得帮助。

DOM 简介

到目前为止，您已经知道浏览器将 Web 页面转换为对象表示，可能您甚至会猜想，对象表示是 DOM 树。DOM 表示 Document Object Model，是一个规范，可从 World Wide Web Consortium (W3C) 获得（您可以参阅 参考资料 中的一些 DOM 相关链接）。

但更重要的是，DOM 定义了对象的类型和属性，从而允许浏览器表示标记。（本系列下一篇文章将专门讲述在 JavaScript 和 Ajax 代码中使用 DOM 的规范。）

文档对象

首先，需要访问对象模型本身。这非常容易；要在运行于 Web 页面上的任何 JavaScript 代码中使用内置 document 变量，可以编写如下代码：

```
var domTree = document;
```

当然，该代码本身没什么用，但它演示了每个 Web 浏览器使得 document 对象可用于 JavaScript 代码，并演示了对象表示标记的完整树（图 1）。

每项都是一个节点

显然，document 对象很重要，但这只是开始。在进一步深入之前，需要学习另一个术语：节点。您已经知道标记的每个部分都由一个对象表示，但它不只是一个任意的对象，它是特定类型的对象，一个 DOM 节点。更特定的类型，比如文本、元素和属性，都继承自这个基本的节点类型。所以可以有文本节点、元素节点和属性节点。

如果已经有很多 JavaScript 编程经验，那您可能已经在使用 DOM 代码了。如果到目前为止您一直在跟踪本 Ajax 系列，那么现在您一定使用 DOM 代码有一段时间了。例如，代码行 var number = document.getElementById("phone").value; 使用 DOM 查找特定元素，然后检索该元素的值（在本例中是一个表单字段）。所以即使您没有意识到这一点，但您每次将 document 键入 JavaScript 代码时都会使用 DOM。

详细解释已经学过的术语，DOM 树是对象的树，但更具体地说，它是节点对象的树。在 Ajax 应用程序中或任何其他 JavaScript 中，可以使用这些节点产生下列效果，比如移除元素及其内容，突出显示特定文本，或添加新图像元素。因为都发生在客户端（运行在 Web 浏览器中的代码），所以这些效果立即发生，而不与服务器通信。最终结果通常是应用程序感觉起来响应更快，因为当请求转向服务器时以及解释响应时，Web 页面上的内容更改不会出现长时间的停顿。

在多数编程语言中，需要学习每种节点类型的实际对象名称，学习可用的属性，并弄清楚类型和强制转换；但在 JavaScript 中这都不是必需的。您可以只创建一个变量，并为它分配您希望的对象（正如您已经看到的）：

```
var domTree = document;

var phoneNumberElement = document.getElementById("phone");

var phoneNumber = phoneNumberElement.value;
```

没有类型，JavaScript 根据需要创建变量并为其分配正确的类型。结果，从 JavaScript 中使用 DOM 变得微不足道（将来有一篇文章会专门讲述与 XML 相关的 DOM，那时将更加巧妙）。

结束语

在这里，我要给您留一点悬念。显然，这并非是对 DOM 完全详尽的说明；事实上，本文不过是 DOM 的简介。DOM 的内容要远远多于我今天介绍的这些！

本系列的下一篇文章将扩展这些观点，并深入探讨如何在 JavaScript 中使用 DOM 来更新 Web 页面、快速更改 HTML 并为您的用户创建更交互的体验。在后面专门讲述在 Ajax 请求中使用 XML 的文章中，我将再次返回来讨论 DOM。所以要熟悉 DOM，它是 Ajax 应用程序的一个主要部分。

此时，深入了解 DOM 将十分简单，比如详细设计如何在 DOM 树中移动、获得元素和文本的值、遍历节点列表，等等，但这可能会让您有这种印象，即 DOM 是关于代码的，而事实上并非如此。

在阅读下一篇文章之前，试着思考一下树结构并用一些您自己的 HTML 实践一下，以查看 Web 浏览器是如何将 HTML 转换为标记的树视图的。此外，思考一下 DOM 树的组织，并用本文介绍的特殊情况实践一下：属性、有元素混合在其中的文本、没有文本内容的元素（比如 img 元素）。

如果扎实掌握了这些概念，然后学习了 JavaScript 和 DOM 的语法（下一篇文章），则会使得响应更为容易。

而且不要忘了，这里有清单 2 和 3 的答案，其中还包含了示例代码！