**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。

**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>

这是处理所有服务器通信的对象。

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。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];

**使用 Microsoft 浏览器**

Microsoft 浏览器 Internet Explorer 使用 MSXML 解析器处理 XML。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。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();。

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;  
  
try {  
  xmlHttp = new ActiveXObject("Msxml2.XMLHTTP");  
} catch (e) {  
  try {  
    xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");  
  } catch (e2) {  
    xmlHttp = false;  
  }  
}

if (!xmlHttp && typeof XMLHttpRequest != 'undefined') {  
  xmlHttp = new XMLHttpRequest();  
}

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。

XMLHttpRequest对象可以在不向服务器提交整个页面的情况下，实现局部更新网页。当页面全部加载完毕后，[客户端](http://baike.baidu.com/view/930.htm)通过该对象向服务器请求数据，服务器端接受数据并处理后，向客户端反馈数据。 XMLHttpRequest 对象提供了对 HTTP 协议的完全的访问，包括做出 POST 和 HEAD 请求以及普通的 GET 请求的能力。XMLHttpRequest 可以同步或异步返回 Web 服务器的响应，并且能以文本或者一个 DOM 文档形式返回内容。尽管名为 XMLHttpRequest，它并不限于和 XML 文档一起使用：它可以接收任何形式的[文本文档](http://baike.baidu.com/view/2135273.htm)。XMLHttpRequest 对象是名为 AJAX 的 Web 应用程序架构的一项关键功能。

**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() {  
  var city = document.getElementById("city").value;  
  var state = document.getElementById("state").value;  
    
  if ((city == null) || (city == "")) return;  
  if ((state == null) || (state == "")) return;

  var url = "/scripts/getZipCode.php?city=" + city + "&state=" + state;

  xmlHttp.open("GET", url, true);

  xmlHttp.onreadystatechange = updatePage;

  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。

**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。

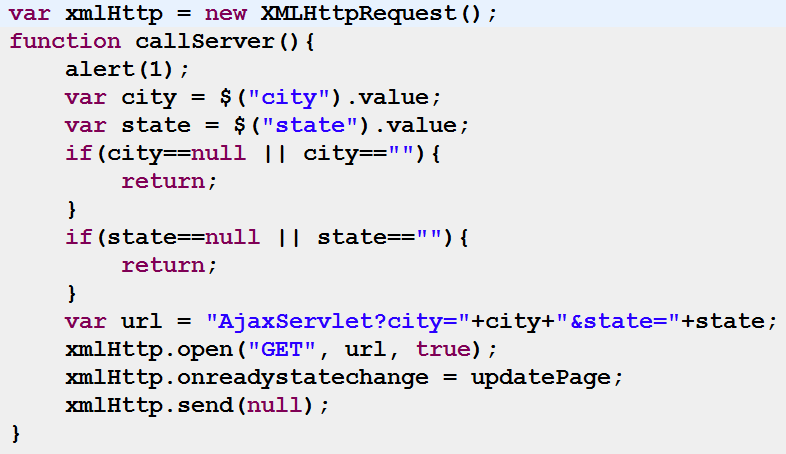
连接 **Web**表单

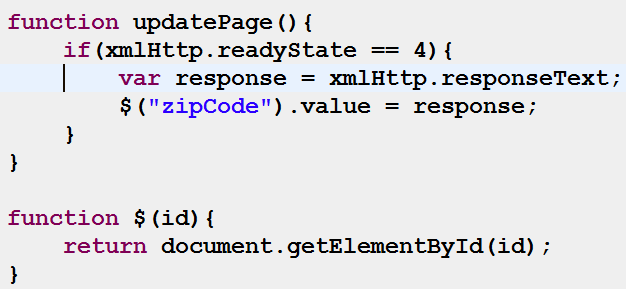
还有什么呢？实际上没有多少了。一个 JavaScript 方法捕捉用户输入表单的信息并将其发送到服务器，另一个 JavaScript 方法监听和处理响应，并在响应返回时设置字段的值。所有这些实际上都依赖于调用 第一个 JavaScript 方法，它启动了整个过程。最明显的办法是在 HTML 表单中增加一个按钮，但这是 2001 年的办法，您不这样认为吗？还是像 清单 7 这样利用 JavaScript 技术吧。

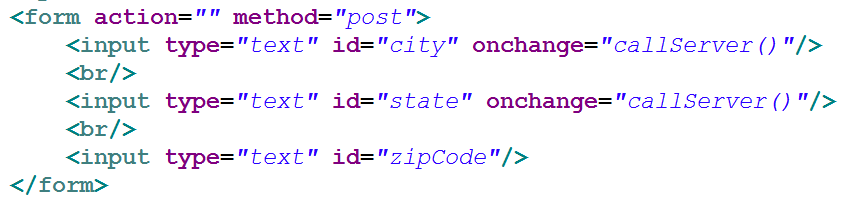
**清单 7. 启动一个 Ajax 过程**

<form>  
 <p>City: <input type="text" name="city" id="city" size="25"   
       onChange="callServer();" /></p>  
 <p>State: <input type="text" name="state" id="state" size="25"   
       onChange="callServer();" /></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！







**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。

对于旧版本浏览器：

<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>

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）。

**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。

**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。

**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=" + 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。

**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。

**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**

**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());  
     }  
   }