AJAX 开发简略

文档说明

参与人员：

作者	网名	联络
柯自聪	eamoi　educhina	eamoi@163.com（技术） zcke0728@hotmail.com（版权）

发布记录：

版本	日期	作者	说明
1.0	2005-10-28	柯自聪	创建，第一版
1.1	2005-11-7	柯自聪	增加 7.4、7.5 关于 DOM 和 XML 的内容

链接：

类别	网址
Blog	http://www.blogjava.net/eamoi/
MSN-Space	http://spaces.msn.com/members/eamoi/

在使用浏览器浏览网页的时候，当页面刷新很慢的时候，你的浏览器在干什么？你的屏幕内容是什么？是的，你的浏览器在等待刷新，而你的屏幕内容是一片空白，而你在屏幕前苦苦的等待浏览器的响应。开发人员为了克服这种尴尬的局面，不得不在每一个可能需要长时间等待响应的页面上增加一个 DIV，告诉用户“ 系统正在处理您的请求，请稍候…… ”。

现在，有一种越来越流行越热的“ 老 ”技术，可以彻底改变这种窘迫的局面。那就是 AJAX。如今，随着 Gmail、Google-maps 的应用和各种浏览器的支持，AJAX 正逐渐吸引全世界的眼球。

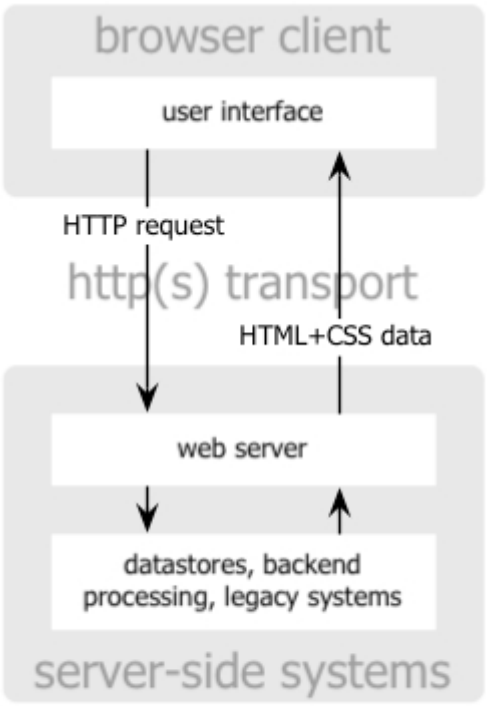
一、AJAX 定义

AJAX（Asynchronous JavaScript and XML）其实是多种技术的综合，包括 Javascript、XHTML 和 CSS、DOM、XML 和 XSTL、XMLHttpRequest。其中：
使用 XHTML 和 CSS 标准化呈现，使用 DOM 实现动态显示和交互，使用 XML 和 XSTL 进行数据交换与处理，使用 XMLHttpRequest 对象进行异步数据读取，使用 Javascript 绑定和处理所有数据。

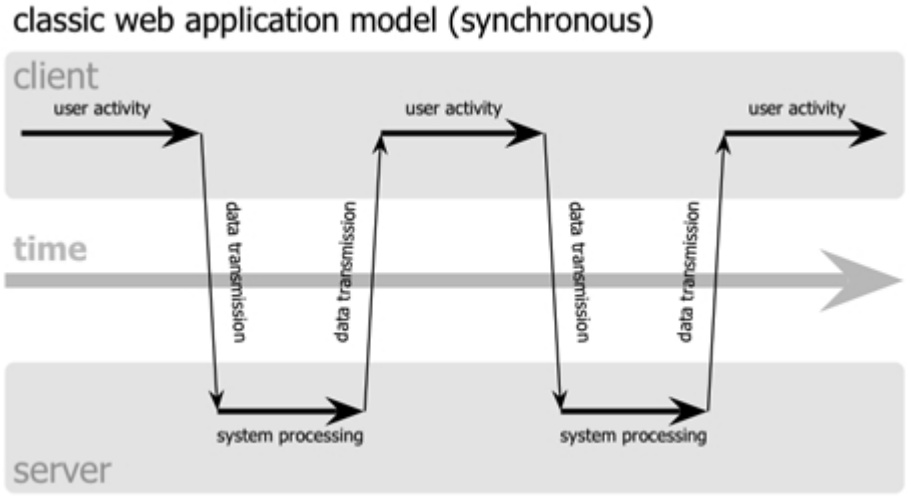
在 AJAX 提出之前，业界对于上述技术都只是单独的使用，没有综合使用，也是由于之前的技术需求所决定的。随着应用的广泛，AJAX 也成为香饽饽了。

二、现状与需要解决的问题

传统的 Web 应用采用同步交互过程，这种情况下，用户首先向 HTTP 服务器触发一个行为或请求的呼求。反过来，服务器执行某些任务，再向发出请求的用户返回一个 HTML 页面。这是一种不连贯的用户体验，服务器在处理请求的时候，用户多数时间处于等待的状态，屏幕内容也是一片空白。如下图：



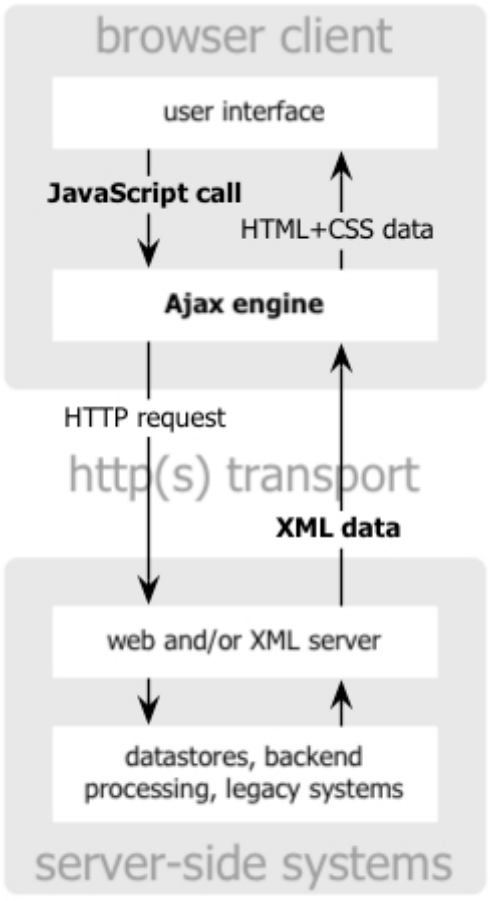
classic
web application model



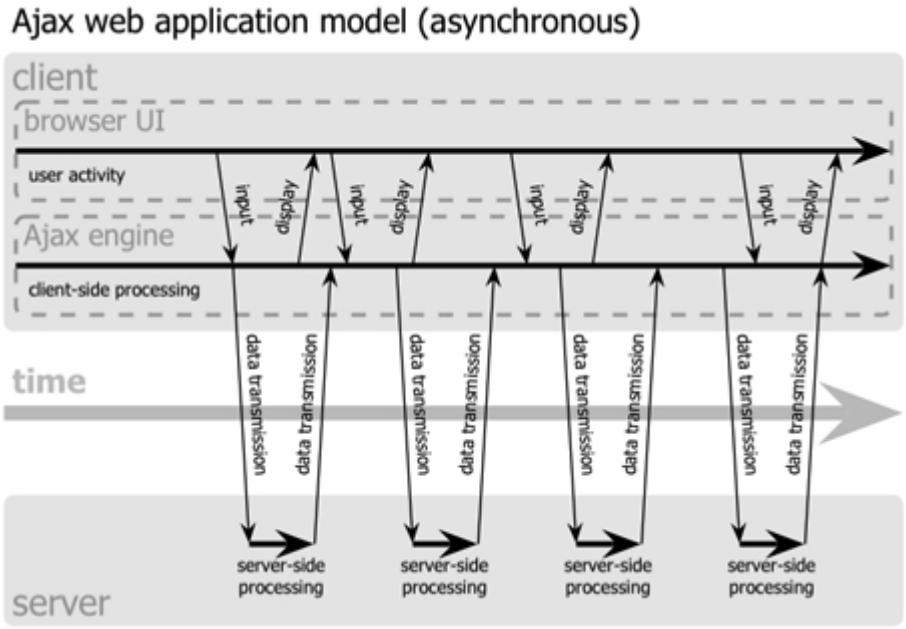
自从采用超文本作为 Web 传输和呈现之后，我们都是采用这么一套传输方式。当负载比较小的时候，这并不会体现出有什么不妥。可是当负载比较大，响应时间要很长，1 分钟、2 分钟……数分钟的时候，这种等待就不可忍受了。严重的，超过响应时间，服务器干脆告诉你页面不可用。另外，某些时候，我只是想改变页面一小部分的数据，那为什么我必须重新加载整个页面呢？！当软件设计越来越讲究人性化的时候，这么糟糕的用户体验简直与这种原则背道而驰。为什么老是要让用户等待服务器取数据呢？至少，我们应该减少用户等待的时间。现在，除了程序设计、编码优化和服务器调优之外，还可以采用 AJAX。

三、为什么使用 AJAX

与传统的 Web 应用不同，AJAX 采用异步交互过程。AJAX 在用户与服务器之间引入一个中间媒介，从而消除了网络交互过程中的处理—等待—处理—等待缺点。用户的浏览器在执行任务时即装载了 AJAX 引擎。AJAX 引擎用 JavaScript 语言编写，通常藏在一个隐藏的框架中。它负责编译用户界面及与服务器之间的交互。AJAX 引擎允许用户与应用软件之间的交互过程异步进行，独立于用户与网络服务器间的交流。现在，可以用 Javascript 调用 AJAX 引擎来代替产生一个 HTTP 的用户动作，内存中的数据编辑、页面导航、数据校验这些不需要重新载入整个页面的需求可以交给 AJAX 来执行。



Ajax
web application model



使用 AJAX，可以为 ISP、开发人员、终端用户带来可见的便捷：

- 减轻服务器的负担。AJAX 的原则是“按需取数据”，可以最大程度的减少冗余请求，和响应对服务器造成的负担。
- 无刷新更新页面，减少用户心理和实际的等待时间。特别的，当要读取大量的数据的时候，不用像 Reload 那样出现白屏的情况，AJAX 使用 XMLHTTP 对象发送请求并得到服务器响应，在不重新载入整个页面的情况下用 Javascript 操作 DOM 最终更新页面。所以在读取数据的过程中，用户所面对的不是白屏，是原来的页面内容（也可以加一个 Loading 的提示框让用户知道处于读取数据过程），只有当数据接收完毕之后才更新相应部分的内容。这种更新是瞬间的，用户几乎感觉不到。
- 带来更好的用户体验。
- 可以把以前一些服务器负担的工作转嫁到客户端，利用客户端闲置的能力来处理，减轻服务器和带宽的负担，节约空间和宽带租用成本。
- 可以调用外部数据。
- 基于标准化的并被广泛支持的技术，不需要下载插件或者小程序。
- 进一步促进页面呈现和数据的分离。

四、谁在使用 AJAX

在应用 AJAX 开发上面，Google 当仁不让是表率。Orkut、Gmail、Google Groups、Google Maps、Google Suggest 都应用了这项技术。Amazon 的 A9.com 搜索引擎也采用了类似的技术。

微软也在积极开发更为完善的 AJAX 应用：它即将推出代号为 Atlas 的 AJAX 工具。Atlas 的功能超越了 AJAX 本身，包括整合 Visual Studio 的调试功能。另外，新的 ASP.NET 控件将使客户端控件与服务器端代码的捆绑更为简便。Atlas 客户脚本框架（Atlas Clent Script Framework）也使与网页及相关项目的交互更为便利。但 Visual Studio 2005 中并不包含此项功能。

微软最近宣布 Atlas 客户脚本框架将包含如下内容（详细资料请访问 Atlas 计划网站）：

* 一个可扩展的核心框架，它添加了 JavaScript 功能：如生命同时期管理、继承管理、

多点传送处理器和界面管理。

* 一个常见功能的基本类库，有丰富的字符串处理、计时器和运行任务。
* 为 HTML 附加动态行为的用户界面框架。
* 一组用来简化服务器连通和网络访问的网络堆栈。
* 一组丰富的用户界面开发控件，如：自动完成的文本框、动画和拖放。
* 处理浏览器脚本行为差异的浏览器兼容层面。

典型的，微软将 AJAX 技术应用在 MSN Space 上面。很多人一直都对 MS Space 服务感到很奇怪，当提交回复评论以后，浏览器会暂时停顿一下，然后在无刷新的情况下把我提交的评论显示出来。这个就是应用了 AJAX 的效果。试想，如果添加一个评论就要重新刷新整个页面，那可真费事。

目前，AJAX 应用最普遍的领域是 GIS-Map 方面。GIS 的区域搜索强调快速响应，AJAX 的特点正好符合这种需求。

五、用 AJAX 改进你的设计

AJAX 虽然可以实现无刷新更新页面内容，但是也不是什么地方都可以用，主要应用在交互较多、频繁读数据、数据分类良好的 Web 应用中。现在，让我们举两个例子，看看如何用 AJAX 改进你的设计。

例子 1：数据校验

在输入 form 表单内容的时候，我们通常需要确保数据的唯一性。因此，常常在页面上提供“唯一性校验”按钮，让用户点击，打开一个校验小窗口；或者等 form 提交到服务器端，由服务器判断后在返回相应的校验信息。前者，window.open 操作本来就是比较耗费资源的，通常由 window. showModalDialog 代替，即使这样也要弹出一个对话框；后者，需要把整个页面提交到服务器并由服务器判断校验，这个过程不仅时间长而且加重了服务器负担。而使用 AJAX，这个校验请求可以由 XMLHttpRequest 对象发出，整个过程不需要弹出新窗口，也不需要将整个页面提交到服务器，快速又不加重服务器负担。

例子 2：按需取数据—级联菜单

以前，为了避免每次对菜单的操作引起的重载页面，不采用每次调用后台的方式，而是一次性将级联菜单的所有数据全部读取出来并写入数组，然后根据用户的操作用 JavaScript 来控制它的子集项目的呈现，这样虽然解决了操作响应速度、不重载页面以及避免向服务器频繁发送请求的问题，但是如果用户不对菜单进行操作或只对菜单中的一部分进行操作的话，那读取的数据中的一部分就会成为冗余数据而浪费用户的资源，特别是在菜单结构复杂、数据量大的情况下（比如菜单有很多级、每一级菜又有上百个项目），这种弊端就更为突出。

现在应用 AJAX，在初始化页面时我们只读出它的第一级的所有数据并显示，在用户操作一级菜单其中一项时，会通过 Ajax 向后台请求当前一级项目所属的二级子菜单的所有数据，如果再继续请求已经呈现的二级菜单中的一项时，再向后面请求所操作二级菜单项对应的所有三级菜单的所有数据，以此类推……这样，用什么就取什么、用多少就取多少，就不会有数据的冗余和浪费，减少了数据下载总量，而且更新页面时不用重载全部内容，只更新需要更新的那部分即可，相对于后台处理并重载的方式缩短了用户等待时间，也把对资源的

浪费降到最低。

例子 3：读取外部数据

AJAX 可以调用外部数据，因此，可以对一些开发的数据比如 XML 文档、RSS 文档进行二次加工，实现数据整合或者开发应用程序。

六、AJAX 的缺陷

AJAX 不是完美的技术。使用 AJAX，它的一些缺陷不得不权衡一下：

- AJAX 大量使用了 Javascript 和 AJAX 引擎，而这个取决于浏览器的支持。IE5.0 及以上、Mozilla1.0、NetScape7 及以上版本才支持，Mozilla 虽然也支持 AJAX，但是提供 XMLHttpRequest 的方式不一样。所以，使用 AJAX 的程序必须测试针对各个浏览器的兼容性。
- AJAX 更新页面内容的时候并没有刷新整个页面，因此，网页的后退功能是失效的；有的用户还经常搞不清楚现在的数据是旧的还是已经更新过的。这个就需要在明显位置提醒用户“ 数据已更新 ”。
- 对流媒体的支持没有 FLASH、Java Applet 好。
- 一些手持设备（如手机、PDA 等）现在还不能很好的支持 Ajax。

七、AJAX 开发

到这里，已经可以清楚的知道 AJAX 是什么，AJAX 能做什么，AJAX 什么地方不好。如果你觉得 AJAX 真的能给你的开发工作带来改进的话，那么继续看看怎么使用 AJAX 吧。

7.1、AJAX 应用到的技术

AJAX 涉及到的 7 项技术中，个人认为 Javascript、XMLHttpRequest、DOM、XML 比较有用。

A、XMLHttpRequest 对象

XMLHttpRequest 是 XMLHTTP 组件的对象，通过这个对象，AJAX 可以像桌面应用程序一样只同服务器进行数据层面的交换，而不用每次都刷新界面，也不用每次将数据处理的工作都交给服务器来做；这样既减轻了服务器负担又加快了响应速度、缩短了用户等待的时间。

IE5.0 开始，开发人员可以在 Web 页面内部使用 XMLHTTP ActiveX 组件扩展自身的功能，不用从当前的 Web 页面导航就可以直接传输数据到服务器或者从服务器接收数据。,Mozilla1.0 以及 NetScape7 则是创建继承 XML 的代理类 XMLHttpRequest；对于大多数情况，XMLHttpRequest 对象和 XMLHTTP 组件很相似，方法和属性类似，只是部分属性不同。

XMLHttpRequest 对象初始化：

```
<script language=”javascript”>
var http_request = false;
//IE 浏览器
http_request = new ActiveXObject("Msxml2.XMLHTTP");
http_request = new ActiveXObject("Microsoft.XMLHTTP");
//Mozilla 浏览器
http_request = new XMLHttpRequest();
</script>
```

XMLHttpRequest 对象的方法：

方法	描述
abort()	停止当前请求
getAllResponseHeaders()	作为字符串返问完整的 headers
getResponseHeader("headerLabel")	作为字符串返问单个的 header 标签
open("method","URL"[,asyncFlag[,"userName"[, "password"]]])	设置未决的请求的目标 URL，方法，和其他参数
send(content)	发送请求
setRequestHeader("label", "value")	设置 header 并和请求一起发送

XMLHttpRequest 对象的属性：

属性	描述
onreadystatechange	状态改变的事件触发器
readyState	对象状态(integer): 0 = 未初始化 1 = 读取中 2 = 已读取 3 = 交互中 4 = 完成
responseText	服务器进程返回数据的文本版本
responseXML	服务器进程返回数据的兼容 DOM 的 XML 文档对象
status	服务器返回的状态码，如：404 = "文件未找到" 、200 ="成功"
statusText	服务器返回的状态文本信息

B、Javascript

Javascript 一直被定位为客户端的脚本语言，应用最多的地方是表单数据的校验。现在，可以通过 Javascript 操作 XMLHttpRequest，来跟数据库打交道。

C、DOM

DOM（Document Object Model）是提供给 HTML 和 XML 使用的一组 API，提供了文件的表述结构，并可以利用它改变其中的内容和可见物。脚本语言通过 DOM 才可以跟页面进行交互。Web 开发人员可操作及建立文件的属性、方法以及事件都以对象来展现。比如，document 就代表页面对象本身。

D、XML

通过 XML（Extensible Markup Language），可以规范的定义结构化数据，是网上传输的数据和文档符合统一的标准。用 XML 表述的数据和文档，可以很容易的让所有程序共享。

7.2、AJAX 开发框架

这里，我们通过一步步的解析，来形成一个发送和接收 XMLHttpRequest 请求的程序框架。AJAX 实质上也是遵循 Request/Server 模式，所以这个框架基本的流程也是：对象初始化→发送请求→服务器接收→服务器返回→客户端接收→修改客户端页面内容。只不过这个过程是异步的。

A、初始化对象并发出 XMLHttpRequest 请求

为了让 Javascript 可以向服务器发送 HTTP 请求，必须使用 XMLHttpRequest 对象。使用之前，要先将 XMLHttpRequest 对象实例化。之前说过，各个浏览器对这个实例化过程实现不同。IE 以 ActiveX 控件的形式提供，而 Mozilla 等浏览器则直接以 XMLHttpRequest 类的形式提供。为了让编写的程序能够跨浏览器运行，要这样写：

```
if (window.XMLHttpRequest) { // Mozilla, Safari, ...
    http_request = new XMLHttpRequest();
} else if (window.ActiveXObject) { // IE
    http_request = new ActiveXObject("Microsoft.XMLHTTP");
}
```

有些版本的 Mozilla 浏览器处理服务器返回的未包含 XML mime-type 头部信息的内容时会出错。因此，要确保返回的内容包含 text/xml 信息。

```
http_request = new XMLHttpRequest();
http_request.overrideMimeType('text/xml');
```

B、指定响应处理函数

接下来要指定当服务器返回信息时客户端的处理方式。只要将相应的处理函数名称赋给

XMLHttpRequest 对象的 onreadystatechange 属性就可以了。比如：

```
http_request.onreadystatechange = processRequest;
```

需要指出的时，这个函数名称不加括号，不指定参数。也可以用 Javascript 即时定义函数的方式定义响应函数。比如：

```
http_request.onreadystatechange = function() {
};
```

C、发出 HTTP 请求

指定响应处理函数之后，就可以向服务器发出 HTTP 请求了。这一步调用 XMLHttpRequest 对象的 open 和 send 方法。

```
http_request.open('GET', 'http://www.example.org/some.file', true);
http_request.send(null);
```

open 的第一个参数是 HTTP 请求的方法，为 Get、Post 或者 Head。

open 的第二个参数是目标 URL。基于安全考虑，这个 URL 只能是同网域的，否则会提示“ 没有权限 ”的错误。这个 URL 可以是任何的 URL，包括需要服务器解释执行的页面，不仅仅是静态页面。目标 URL 处理请求 XMLHttpRequest 请求则跟处理普通的 HTTP 请求一样，比如 JSP 可以用 request.getParameter(“”)或者 request.getAttribute(“”)来取得 URL 参数值。

open 的第三个参数只是指定在等待服务器返回信息的时间内是否继续执行下面的代码。如果为 True，则不会继续执行，直到服务器返回信息。默认为 True。

按照顺序，open 调用完毕之后要调用 send 方法。send 的参数如果是以 Post 方式发出的话，可以是任何想传给服务器的内容。不过，跟 form 一样，如果要传文件或者 Post 内容给服务器，必须先调用 setRequestHeader 方法，修改 MIME 类别。如下：

```
http_request.setRequestHeader(“Content-Type”,”application/x-www-form-urlencoded”);
```

这时资料则以查询字符串的形式列出，作为 sned 的参数，例如：

```
name=value&anothername=othervalue&so=on
```

D、处理服务器返回的信息

在第二步我们已经指定了响应处理函数，这一步，来看看这个响应处理函数都应该做什么。

首先，它要检查 XMLHttpRequest 对象的 readyState 值，判断请求目前的状态。参照前文的属性表可以知道，readyState 值为 4 的时候，代表服务器已经传回所有的信息，可以开

始处理信息并更新页面内容了。如下：

```
if (http_request.readyState == 4) {
    // 信息已经返回，可以开始处理
} else {
    // 信息还没有返回，等待
}
```

服务器返回信息后，还需要判断返回的HTTP状态码，确定返回的页面没有错误。所有的状态码都可以在W3C的官方网站上查到。其中，200 代表页面正常。

```
if (http_request.status == 200) {
    // 页面正常，可以开始处理信息
} else {
    // 页面有问题
}
```

XMLHttpRequest 对成功返回的信息有两种处理方式：
responseText：将传回的信息当字符串使用；
responseXML：将传回的信息当 XML 文档使用，可以用 DOM 处理。

E、一个初步的开发框架

总结上面的步骤，我们整理出一个初步的可用的开发框架，供以后调用；这里，将服务器返回的信息用 window.alert 以字符串的形式显示出来：

```
<script language="javascript">
    var http_request = false;
    function send_request(url) {//初始化、指定处理函数、发送请求的函数
        http_request = false;
        //开始初始化 XMLHttpRequest 对象
        if(window.XMLHttpRequest) { //Mozilla 浏览器
            http_request = new XMLHttpRequest();
            if (http_request.overrideMimeType) {//设置 MiME 类别
                http_request.overrideMimeType("text/xml");
            }
        }
        else if (window.ActiveXObject) { // IE 浏览器
            try {
                http_request = new ActiveXObject("Msxml2.XMLHTTP");
            } catch (e) {
                try {
                    http_request = new ActiveXObject("Microsoft.XMLHTTP");
                } catch (e) {}
            }
        }
        if (!http_request) { // 异常，创建对象实例失败
            window.alert("不能创建 XMLHttpRequest 对象实例.");
            return false;
        }
        http_request.onreadystatechange = processRequest;
        // 确定发送请求的方式和 URL 以及是否同步执行下段代码
        http_request.open("GET", url, true);
        http_request.send(null);
    }
    // 处理返回信息的函数
    function processRequest() {
        if (http_request.readyState == 4) { // 判断对象状态
            if (http_request.status == 200) { // 信息已经成功返回，开始处理信息
                alert(http_request.responseText);
            } else { //页面不正常
                alert("您所请求的页面有异常。");
            }
        }
    }
</script>
```

7.3、简单的示例

接下来，我们利用上面的开发框架来做两个简单的应用。

A、数据校验

在用户注册的表单中，经常碰到要检验待注册的用户名是否唯一。传统的做法是采用 window.open 的弹出窗口，或者 window. showModalDialog 的对话框。不过，这两个都需要打开窗口。采用 AJAX 后，采用异步方式直接将参数提交到服务器，用 window.alert 将服务器返回的校验信息显示出来。代码如下：
在<body></body>之间增加一段 form 表单代码：

```
<form name="form1" action="" method="post">
用户名：<input type="text" name="username" value=""> 
<input type="button" name="check" value="唯一性检查" onClick="userCheck()">
<input type="submit" name="submit" value="提交">
</form>
```

在开发框架的基础上再增加一个调用函数：

```
function userCheck() {
	var f = document.form1;
	var username = f.username.value;
	if(username=="") {
		window.alert("用户名不能为空。");
		f.username.focus();
		return false;
	}
	else {
		send_request('sample1_2.jsp?username='+username);
	}
}
```

看看 sample1_2.jsp 做了什么：

```
<%@ page contentType="text/html; charset=gb2312" errorPage="" %>
<%
String username = request.getParameter("username");
if("educhina".equals(username)) out.print("用户名已经被注册，请更换一个用户名。");
else out.print("用户名尚未被使用，您可以继续。");
%>
```

运行一下，嗯，没有弹出窗口，没有页面刷新，跟预想的效果一样。如果需要的话，可以在

sample1_2.jsp 中实现更复杂的功能。最后，只要将反馈信息打印出来就可以了。

B、级联菜单

我们在第五部分提到利用 AJAX 改进级联菜单的设计。接下来，我们就演示一下如何“按需取数据”。

首先，在<body></body>中间增加如下 HTML 代码：

```
<table width="200" border="0" cellspacing="0" cellpadding="0">
    <tr>
        <td height="20">
            <a href="javascript:void(0)" onClick="showRoles('pos_1')">经理室</a>
        </td>
    </tr>
    <tr style="display:none">
        <td height="20" id="pos_1"> </td>
    </tr>
    <tr>
        <td height="20">
            <a href="javascript:void(0)" onClick="showRoles('pos_2')">开发部</a>
        </td>
    </tr>
    <tr style="display:none ">
        <td id="pos_2" height="20"> </td>
    </tr>
</table>
```

在框架的基础上增加一个响应函数 showRoles(obj)：

```
//显示部门下的岗位
function showRoles(obj) {
        document.getElementById(obj).parentNode.style.display = "";
        document.getElementById(obj).innerHTML = "正在读取数据..."
        currentPos = obj;
        send_request("sample2_2.jsp?playPos="+obj);
}
```

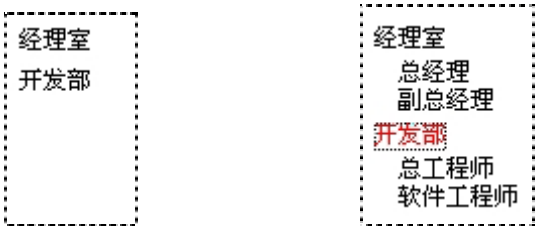
修改框架的 processRequest 函数：

```
// 处理返回信息的函数
function processRequest() {
   if (http_request.readyState == 4) { // 判断对象状态
      if (http_request.status == 200) { // 信息已经成功返回，开始处理信息
      document.getElementById(currentPos).innerHTML = http_request.responseText;
      } else { //页面不正常
         alert("您所请求的页面有异常。");
      }
   }
}
```

最后就是 smaple2_2.jsp 了：

```
<%@ page contentType="text/html; charset=gb2312" errorPage="" %>
<%
String playPos = request.getParameter("playPos");
if("pos_1".equals(playPos))
        out.print("  总经理<br>  副总经理");
else if("pos_2".equals(playPos))
        out.println("  总工程师<br>  软件工程师");
%>
```

运行一下看看效果：

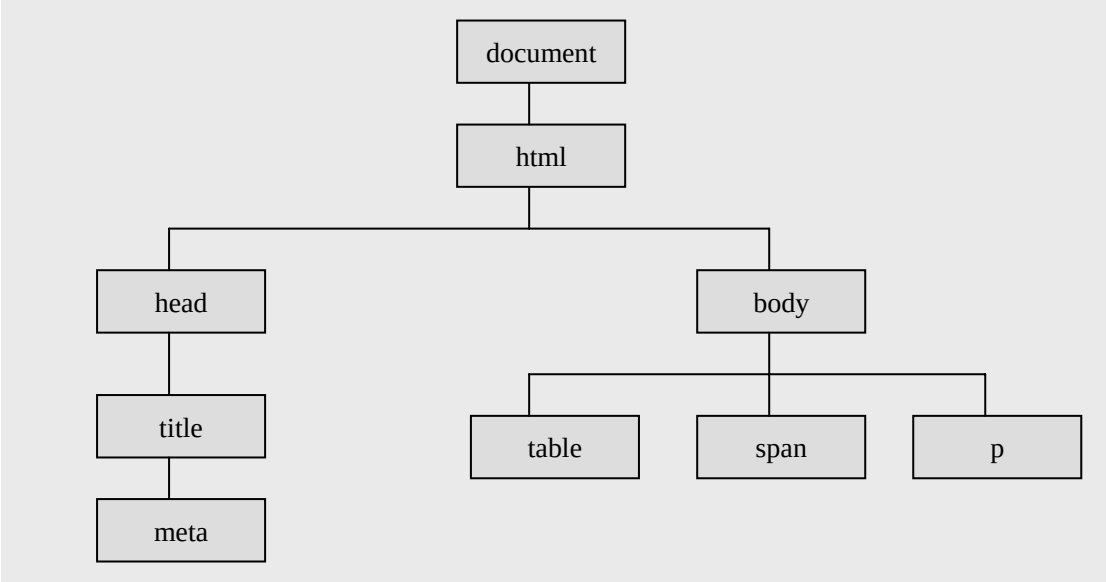


7.4、文档对象模型（DOM）

文档对象模型（DOM）是表示文档（比如 HTML 和 XML）和访问、操作构成文档的各种元素的应用程序接口（API）。一般的，支持 Javascript 的所有浏览器都支持 DOM。本文所涉及的 DOM，是指 W3C 定义的标准的文档对象模型，它以树形结构表示 HTML 和 XML 文档，定义了遍历这个树和检查、修改树的节点的方法和属性。

7.4.1、DOM 眼中的 HTML 文档：树

在 DOM 眼中，HTML 跟 XML 一样是一种树形结构的文档，<html>是根（root）节点，<head>、<title>、<body>是<html>的子（children）节点，互相之间是兄弟（sibling）节点；<body>下面才是子节点<table>、、<p>等等。如下图：



这个是不是跟 XML 的结构有点相似呢。不同的是，HTML 文档的树形主要包含表示元素、标记的节点和表示文本串的节点。

7.4.2、HTML 文档的节点

DOM 下，HTML 文档各个节点被视为各种类型的 Node 对象。每个 Node 对象都有自己的属性和方法，利用这些属性和方法可以遍历整个文档树。由于 HTML 文档的复杂性，DOM 定义了 nodeType 来表示节点的类型。这里列出 Node 常用的几种节点类型：

接口	nodeType 常量	nodeType 值	备注
Element	Node.ELEMENT_NODE	1	元素节点
Text	Node.TEXT_NODE	3	文本节点
Document	Node.DOCUMENT_NODE	9	document
Comment	Node.COMMENT_NODE	8	注释的文本
DocumentFragment	Node.DOCUMENT_FRAGMENT_NODE	11	document 片断
Attr	Node.ATTRIBUTE_NODE	2	节点属性

DOM 树的根节点是个 Document 对象，该对象的 documentElement 属性引用表示文档根

元素的 Element 对象（对于 HTML 文档，这个就是<html>标记）。Javascript 操作 HTML 文档的时候，document 即指向整个文档，<body>、<table>等节点类型即为 Element。Comment 类型的节点则是指文档的注释。具体节点类型的含义，请参考《Javascript 权威指南》，在此不赘述。

Document 定义的方法大多数是生产型方法，主要用于创建可以插入文档中的各种类型的节点。常用的 Document 方法有：

方法	描述
createAttribute()	用指定的名字创建新的 Attr 节点。
createComment()	用指定的字符串创建新的 Comment 节点。
createElement()	用指定的标记名创建新的 Element 节点。
createTextNode()	用指定的文本创建新的 TextNode 节点。
getElementById()	返回文档中具有指定 id 属性的 Element 节点。
getElementsByTagName()	返回文档中具有指定标记名的所有 Element 节点。

对于 Element 节点，可以通过调用 getAttribute()、setAttribute()、removeAttribute()方法来查询、设置或者删除一个 Element 节点的性质，比如<table>标记的 border 属性。下面列出 Element 常用的属性：

属性	描述
tagName	元素的标记名称，比如<p>元素为 P。HTML 文档返回的 tabName 均为大写。

Element 常用的方法：

方法	描述
getAttribute()	以字符串形式返回指定属性的值。
getAttributeNode()	以 Attr 节点的形式返回指定属性的值。
getElementsByTabName()	返回一个 Node 数组，包含具有指定标记名的所有 Element 节点的子孙节点，其顺序为在文档中出现的顺序。
hasAttribute()	如果该元素具有指定名字的属性，则返回 true。
removeAttribute()	从元素中删除指定的属性。
removeAttributeNode()	从元素的属性列表中删除指定的 Attr 节点。
setAttribute()	把指定的属性设置为指定的字符串值，如果该属性不存在则添加一个新属性。
setAttributeNode()	把指定的 Attr 节点添加到该元素的属性列表中。

Attr 对象代表文档元素的属性，有 name、value 等属性，可以通过 Node 接口的 attributes 属性或者调用 Element 接口的 getAttributeNode()方法来获取。不过，在大多数情况下，使用 Element 元素属性的最简单方法是 getAttribute()和 setAttribute()两个方法，而不是 Attr 对象。

7.4.3、使用 DOM 操作 HTML 文档

Node 对象定义了一系列属性和方法，来方便遍历整个文档。用 parentNode 属性和 childNodes[]数组可以在文档树中上下移动；通过遍历 childNodes[]数组或者使用 firstChild 和 nextSibling 属性进行循环操作，也可以使用 lastChild 和 previousSibling 进行逆向循环操作，也可以枚举指定节点的子节点。而调用 appendChild()、insertBefore()、removeChild()、replaceChild()方法可以改变一个节点的子节点从而改变文档树。

需要指出的是，childNodes[]的值实际上是一个 NodeList 对象。因此，可以通过遍历 childNodes[]数组的每个元素，来枚举一个给定节点的所有子节点；通过递归，可以枚举树

中的所有节点。下表列出了 Node 对象的一些常用属性和方法：

Node 对象常用属性：

属性	描述
attributes	如果该节点是一个 Element，则以 NamedNodeMap 形式返回该元素的属性。
childNodes	以 Node[]的形式存放当前节点的子节点。如果没有子节点，则返回空数组。
firstChild	以 Node 的形式返回当前节点的第一个子节点。如果没有子节点，则为 null。
lastChild	以 Node 的形式返回当前节点的最后一个子节点。如果没有子节点，则为 null。
nextSibling	以 Node 的形式返回当前节点的兄弟下一个节点。如果没有这样的节点，则返回 null。
nodeName	节点的名字，Element 节点则代表 Element 的标记名称。
nodeType	代表节点的类型。
parentNode	以 Node 的形式返回当前节点的父节点。如果没有父节点，则为 null。
previousSibling	以 Node 的形式返回紧挨当前节点、位于它之前的兄弟节点。如果没有这样的节点，则返回 null。

Node 对象常用方法：

方法	描述
appendChild()	通过把一个节点增加到当前节点的 childNodes[]组，给文档树增加节点。
cloneNode()	复制当前节点，或者复制当前节点以及它的所有子孙节点。
hasChildNodes()	如果当前节点拥有子节点，则将返回 true。
insertBefore()	给文档树插入一个节点，位置在当前节点的指定子节点之前。如果该节点已经存在，则删除之再插入到它的位置。
removeChild()	从文档树中删除并返回指定的子节点。
replaceChild()	从文档树中删除并返回指定的子节点，用另一个节点替换它。

接下来，让我们使用上述的 DOM 应用编程接口，来试着操作 HTML 文档。

A、遍历文档的节点

DOM 把一个 HTML 文档视为树，因此，遍历整个树是应该是家常便饭。跟之前说过的一样，这里我们提供两个遍历树的例子。通过它，我们能够学会如何使用 childNodes[]和 firstChile、lastChild、nextSibling、previousSibling 遍历整棵树。

例子 1-- sample3_1.htm：

这个例子使用了 childNodes[]和递归方式来遍历整个文档，统计文档中出现的 Element 元素总数，并把 Element 标记名全部打印出来。需要特别注意的是，在使用 DOM 时，必须等文档被装载完毕再执行遍历等行为操作文档。sample3_1.htm 具体代码如下：

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>无标题文档</title>
<script language="javascript">
var elementName = ""; //全局变量，保存 Element 标记名，使用完毕要清空
function countTotalElement(node) { //参数 node 是一个 Node 对象
    var total = 0;
    if(node.nodeType == 1) { //检查 node 是否为 Element 对象
        total++;                //如果是，计数器加 1
        elementName = elementName + node.tagName + "\r\n"; //保存标记名
```

```
    }
    var childrens = node.childNodes;          //获取 node 的全部子节点
    for(var i=0;i<childrens.length;i++) {
        total += countTotalElement(childrens[i]); //在每个子节点上进行递归操作
    }
    return total;
}
</script>
</head>
<body>
<a href="javascript:void(0)"
onClick="alert('标记总数：' + countTotalElement(document) + '\r\n 全部标记如下：\r\n' +
elementName);elementName='';">开始统计</a>
</body>
</html>
```

运行效果如下：

例子 2 – sample3_2.htm：

接下来使用 firstChile、lastChild、nextSibling、previousSibling 遍历整个文档树。修改一下 countTotalElement 函数，其他跟 sample3_1.htm 一样：

```
function countTotalElement(node) { //参数 node 是一个 Node 对象
    var total = 0;
    if(node.nodeType == 1) { //检查 node 是否为 Element 对象
        total++;                //如果是，计数器加 1
        elementName = elementName + node.tagName + "\r\n"; //保存标记名
    }
    var childrens = node.childNodes;          //获取 node 的全部子节点
    for(var m=node.firstChild; m!=null;m=m.nextSibling) {
        total += countTotalElement(m); //在每个子节点上进行递归操作
    }
    return total;
}
```

B、搜索文档中特定的元素

在使用 DOM 的过程中，有时候需要定位到文档中的某个特定节点，或者具有特定类型的节点列表。这种情况下，可以调用 Document 对象的 getElementsByTagName()和

getElementById()方法来实现。

document.getElementsByTagName()返回文档中具有指定标记名的全部 Element 节点数组（也是 NodeList 类型）。Element 出现在数组中的顺序就是他们在文档中出现的顺序。传递给 getElementsByTagName()的参数忽略大小写。比如，想定位到第一个<table>标记，可以这样写：document.getElementsByTagName(“table”)[0]。例外的，可以使用 document.body 定位到<body>标记，因为它是唯一的。

getElementsByTagName()返回的数组取决于文档。一旦文档改变，返回结果也立即改变。相比，getElementById()则比较灵活，可以随时定位到目标，只是要实现给目标元素一个唯一的 id 属性值。这个我们在《AJAX 开发简略》的“ 级联菜单 ”例子中已经使用过了。

Element 对象也支持 getElementsByTagName()和 getElementById()。不同的是，搜索领域只针对调用者的子节点。

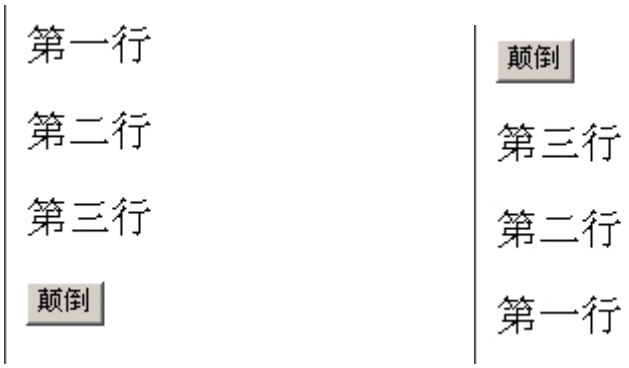
C、修改文档内容

遍历整棵文档树、搜索特定的节点，我们最终目的之一是要修改文档内容。接下来的三个例子将使用 Node 的几个常用方法，来演示如何修改文档内容。

例子 3 -- sample4_1.htm：

这个例子包含三个文本节点和一个按钮。点击按钮后，三个文本节点和按钮的顺序将被颠倒。程序使用了 Node 的 appendChild()和 removeChild()方法。

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>无标题文档</title>
<script language="javascript">
    function reverseNode(node) { //  颠倒节点 node 的顺序
        var kids = node.childNodes; //获取子节点列表
        var kidsNum = kids.length; //统计子节点总数
        for(var i=kidsNum-1;i>=0;i--) { //逆向遍历子节点列表
            var c = node.removeChild(kids[i]); //删除指定子节点，保存在 c 中
            node.appendChild(c); //将 c 放在新位置上
        }
    }
</script>
</head>
<body>
<p>第一行</p>
<p>第二行</p>
<p>第三行</p>
<p><input type="button" name="reverseGo" value="颠倒"
onClick="reverseNode(document.body)"></p>
</body>
</html>
```



例子 4-- sample4_2.htm：

例子 1 通过直接操作 body 的子节点来修改文档。在 HTML 文档中，布局和定位常常通过表格<table>来实现。因此，例子 4 将演示操作表格内容，将表格的四个单元行顺序颠倒。如果没有使用<tbody>标签，则<table>把全部的<tr>当做是属于一个子节点<tbody>，所以我们采用数组缓存的方式，把行数据颠倒一下。这个例子同时也演示了如何使用 DOM 创建表格单元行。

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>无标题文档</title>
<script language="javascript">
function reverseTable() {
	var node = document.getElementsByTagName("table")[0]; //第一个表格
	var child = node.getElementsByTagName("tr"); //取得表格内的所有行
	var newChild = new Array(); //定义缓存数组，保存行内容
	for(var i=0;i<child.length;i++) {
		newChild[i] = child[i].firstChild.innerHTML;
	}
	node.removeChild(node.childNodes[0]); //删除全部单元行
	var header = node.createTHead(); //新建表格行头
	for(var i=0;i<newChild.length;i++) {
		var headerrow = header.insertRow(i); //插入一个单元行
		var cell = headerrow.insertCell(0); //在单元行中插入一个单元格
		//在单元格中创建 TextNode 节点
		cell.appendChild(document.createTextNode(newChild[newChild.length-i-1]));
	}
}
</script>
</head>
<body>
<table width="200" border="1" cellpadding="4" cellspacing="0">
	<tr>
		<td height="25">第一行</td>
	</tr>
	<tr>
```

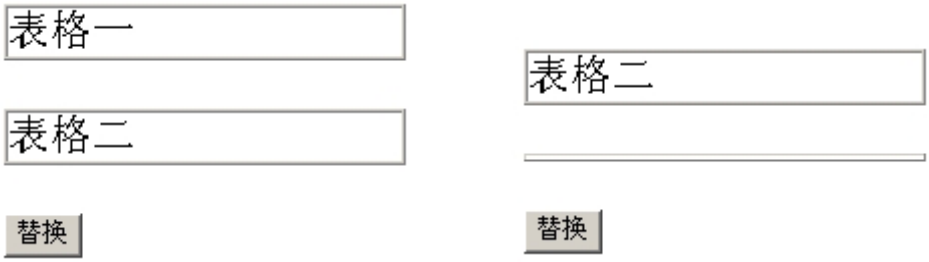
```
        <td height="25">第二行</td>
    </tr>
    <tr>
        <td height="25">第三行</td>
    </tr>
    <tr>
        <td height="25">第四行</td>
    </tr>
</table>
<br>
<input type="button" name="reverse" value="开始颠倒" onClick="reverseTable()">
</body>
</html>
```

例子 5 -- sample4_3.htm：

正如我们在 Node 节点介绍部分所指出的那样，appendChild()、replaceChild()、removeChild()、insertBefore()方法会立即改变文档的结构。下面的例子包含两个表格，我们试着把表格二的内容替换表格一的内容。

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>无标题文档</title>
<script language="javascript">
function replaceContent() {
    var table1 = document.getElementsByTagName("table")[0];
    var table2 = document.getElementsByTagName("table")[1];
    var kid1 = table1.firstChild.firstChild.firstChild; //定位到<td>节点
    var kid2 = table2.firstChild.firstChild.firstChild; //定位到<td>节点
    var repKid = kid2.firstChild; //定位到表格二<td>内含的 TextNode 节点
    try {
        //用表格二的单元格内容替换表格一的单元格内容，表格二变成没有单元格内容
        kid1.replaceChild(repKid,kid1.firstChild);
        //下面注释如果开放，将出现 object error，因为表格二已经被改变
        //kid2.replaceChild(kid1.firstChild,kid2.firstChild);
    }catch(e){
```

```
            alert(e);
        }
}
</script>
</head>
<body>
<table width="200" border="1" cellspacing="0" cellpadding="0">
<tbody>
        <tr>
            <td>表格一</td>
        </tr>
</tbody>
</table>
<br>
<table width="200" border="1" cellspacing="0" cellpadding="0">
<tbody>
        <tr>
            <td>表格二</td>
        </tr>
</tbody>
</table>
<br>
<input type="button" name="replaceNode" value="替换" onClick="replaceContent()">
</body>
</html>
```



注意，当执行 kid1.replaceChild(repKid,kid1.firstChild);的时候，table2 的子节点已经被转移到 table1 了，table2 已经没有子节点，不能再调用 table2 的子节点。看看代码的注释，试着运行一下，应该就知道文档是怎么改变的了。

D、往文档添加新内容

在学会遍历、搜索、修改文档之后，我们现在试着网文档添加新的内容。其实没有什么新意，只是利用我们上述提到的 Node 的属性和方法而已，还是操作<table>标记的内容。有新意的是，我们要实现一个留言簿。是的，留言簿，你可以往里面留言，只是不能刷新噢。

例子 6 – sample5_1.htm：

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>无标题文档</title>
```

```
<script language="javascript">
function insertStr() {
	var f = document.form1;
	var value = f.str.value;
	if(value!="") {
		//  从最终的 TextNode 节点开始，慢慢形成<tbody>结构
		var text = document.createTextNode(value); //新建一个 TextNode 节点
		var td = document.createElement("td"); //新建一个 td 类型的 Element 节点
		var tr = document.createElement("tr"); //新建一个 tr 类型的 Element 节点
		var tbody = document.createElement("tbody"); //新建一个 tbody 类型的 Element 节点
		td.appendChild(text); //将节点 text 加入 td 中
		tr.appendChild(td); //将节点 td 加入 tr 中
		tbody.appendChild(tr); //将节点 tr 加入 tbody 中
		var parNode = document.getElementById("table1"); //定位到 table 上
		parNode.insertBefore(tbody,parNode.firstChild); //将节点 tbody 插入到节点顶部
		//parNode.appendChild(tbody); //将节点 tbody 加入节点尾部
	}
}
</script>
</head>
<body>
<form name="form1" method="post" action="">
	<input name="str" type="text" id="str" value="">
	<input name="insert" type="button" id="insert" value="留言" onClick="insertStr()">
</form>
<table width="400" border="1" cellspacing="0" cellpadding="0" id="table1">
<tbody>
	<tr>
		<td height="25">网友留言列表：</td>
	</tr>
</tbody>
</table>
</body>
</html>
```

我们之前说过，<table>的子节点是<tbody>，<tbody>的子节点才是<tr>，<tr>是<td>的父节点，最后<td>内部的 TextNode 节点。所以，往<table>增加单元格行要逐级形成，就像往树里面添加一个枝桠一样，要有叶子有径。看看，这个留言簿是不是很简单啊。这个例子同时也演示了往<table>表格标记里面增加内容的另一种方法。

加油加油 留言

网友留言列表：
这个留言簿不错
好像很简单噢
加油加油

E、使用 DOM 操作 XML 文档

在数据表示方面，XML 文档更加结构化。DOM 在支持 HTML 的基础上提供了一系列的 API，支持针对 XML 的访问和操作。利用这些 API，我们可以从 XML 中提取信息，动态的创建这些信息的 HTML 呈现文档。处理 XML 文档，通常遵循“ 加载 XML 文档→提取信息→加工信息→创建 HTML 文档 ”的过程。下面的例子演示了如何加载并处理 XML 文档。

这个例子包含两个 JS 函数。loadXML()负责加载 XML 文档，其中既包含加载 XML 文档的 2 级 DOM 代码，又有实现同样操作的 Microsoft 专有 API 代码。需要提醒注意的是，文档加载过程不是瞬间完成的，所以对 loadXML()的调用将在加载文档完成之前返回。因此，需要传递给 loadXML()一个引用，以便文档加载完成后调用。

例子中的另外一个函数 makeTable()，则在 XML 文档加载完毕之后，使用最后前介绍过的 DOM 应用编程接口读取 XML 文档信息，并利用这些信息形成一个新的 table 表格。

例子 7 -- sample6_1.htm：

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>无标题文档</title>
<script language="javascript">
function loadXML(handler) {
	var url = "employees.xml";
	if(document.implementation&&document.implementation.createDocument) {
		var xmldoc = document.implementation.createDocument("", "", null);
		xmldoc.onload =    handler(xmldoc, url);
		xmldoc.load(url);
	}
	else if(window.ActiveXObject) {
		var xmldoc = new ActiveXObject("Microsoft.XMLDOM");
		xmldoc.onreadystatechange = function() {
			if(xmldoc.readyState == 4) handler(xmldoc, url);
		}
		xmldoc.load(url);
	}
}
function makeTable(xmldoc, url) {
	var table = document.createElement("table");
	table.setAttribute("border","1");
```

```
        table.setAttribute("width","600");
        table.setAttribute("class","tab-content");
        document.body.appendChild(table);
        var caption = "Employee Data from " + url;
        table.createCaption().appendChild(document.createTextNode(caption));
        var header = table.createTHead();
        var headerrow = header.insertRow(0);
        headerrow.insertCell(0).appendChild(document.createTextNode("姓名"));
        headerrow.insertCell(1).appendChild(document.createTextNode("职业"));
        headerrow.insertCell(2).appendChild(document.createTextNode("工资"));
        var employees = xmldoc.getElementsByTagName("employee");
        for(var i=0;i<employees.length;i++) {
                var e = employees[i];
                var name = e.getAttribute("name");
                var job = e.getElementsByTagName("job")[0].firstChild.data;
                var salary = e.getElementsByTagName("salary")[0].firstChild.data;
                var row = table.insertRow(i+1);
                row.insertCell(0).appendChild(document.createTextNode(name));
                row.insertCell(1).appendChild(document.createTextNode(job));
                row.insertCell(2).appendChild(document.createTextNode(salary));
        }
}
</script>
<link href="css/style.css" rel="stylesheet" type="text/css">
</head>

<body onLoad="loadXML(makeTable)">
</body>
</html>
```

供读取调用的 XML 文档 – employees.xml：

```
<?xml version="1.0" encoding="gb2312"?>
<employees>
        <employee name="J.Doe">
                <job>Programmer</job>
                <salary>32768</salary>
        </employee>
        <employee name="A.Baker">
                <job>Sales</job>
                <salary>70000</salary>
        </employee>
        <employee name="Big Cheese">
                <job>CEO</job>
                <salary>100000</salary>
        </employee>
```

```
</employees>
```

Employee Data from employees.xml

姓名	职业	工资
J. Doe	Programmer	32768
A. Baker	Sales	70000
Big Cheese	CEO	100000

7.5、处理 XML 文档

脱离 XML 文档的 AJAX 是不完整的。在本部分未完成之前，有读者说 AJAX 改名叫 AJAH（H 应该代表 HTML 吧）比较合适。应该承认，XML 文档在数据的结构化表示以及接口对接上有先天的优势，但也不是所有的数据都应该用 XML 表示。有些时候单纯的文本表示可能会更合适。下面先举个 AJAX 处理返回 XML 文档的例子再讨论什么时候使用 XML。

7.5.1、处理返回的 XML

例子 8 -- sample7_1.htm：

在这个例子中，我们采用之前确定的 AJAX 开发框架，稍微修改一下 body 内容和 processRequest 的相应方式，将先前的 employees.xml 的内容读取出来并显示。

body 的内容如下：

```
<input type="button" name="read"
 value="读取 XML" onClick="send_request('employees.xml')">
```

processRequest()方法修改如下：

```
    // 处理返回信息的函数
    function processRequest() {
        if (http_request.readyState == 4) { // 判断对象状态
            if (http_request.status == 200) { // 信息已经成功返回，开始处理信息
                var returnObj = http_request.responseXML;
                var xmlobj = http_request.responseXML;
                var employees = xmlobj.getElementsByTagName("employee");
                var feedbackStr = "";
                for(var i=0;i<employees.length;i++) { // 循环读取 employees.xml 的内容
                    var employee = employees[i];
                    feedbackStr += "员工：" + employee.getAttribute("name");
                    feedbackStr +=
                    " 职位：" + employee.getElementsByTagName("job")[0].firstChild.data;
                    feedbackStr +=
                  " 工资：" + employee.getElementsByTagName("salary")[0].firstChild.data;
                    feedbackStr +=   "\r\n";
                }
                alert(feedbackStr);
```

```
                } else { //页面不正常
                    alert("您所请求的页面有异常。");
                }
            }
        }
```

运行一下，看来效果还不错：

7.5.2、选择合适的 XML 生成方式

现在的 web 应用程序往往采用了 MVC 三层剥离的设计方式。XML 作为一种数据保存、呈现、交互的文档，其数据往往是动态生成的，通常由 JavaBean 转换过来。由 JavaBean 转换成 XML 文档的方式有好几种，选择合适的转换方式往往能达到事半功倍的效果。下面介绍两种常用的方式，以便需要的时候根据情况取舍。

A、类自行序列化成 XML

类自行序列化成 XML 即每个类都实现自己的 toXML()方法，选择合适的 API、适当的 XML 结构、尽量便捷的生成逻辑快速生成相应的 XML 文档。显然，这种方式必须要求每个类编写专门的 XML 生成代码，每个类只能调用自己的 toXML()方法。应用诸如 JDOM 等一些现成的 API，可以减少不少开发投入。例子 9 是一个利用 JDOM 的 API 形成的 toXML()方法。

例子 9 -- toXml() 的 JDOM 实现 -- Employ 类的 toXml()方法：

```
public Element toXml() {
    Element employee = new Element(“employee”);
    Employee.setAttribute(“name”,name);
    Element jobE = new Element(“job”).addContent(job);
    employee.setContent(jobE);
    Element salaryE = new Element(“salary”).addContent(salary);
    employee.setContent(salaryE);
    return employee;
}
```

JDOM 提供了现成的 API，使得序列化成 XML 的工作更加简单，我们只需要把 toXML()外面包装一个 Document，然后使用 XMLOutputter 把文档写入 servlet 就可以了。toXml()允许递归调用其子类的 toXML()方法，以便生成包含子图的 XML 文档。

使用类自行序列化成 XML 的方式，要每个类都实现自己的 toXML()方法，而且存在数据模型与视图耦合的问题，即要么为每个可能的视图编写独立的 toXML()方法，要么心甘情愿接收冗余的数据，一旦数据结构或者文档发生改变，toXML()就要做必要的修改。

B、页面模板生成 XML 方式

一般的，可以采用通用的页面模板技术来生成 XML 文档，这个 XML 文档可以符合任

何需要的数据模型，供 AJAX 灵活的调用。另外，模板可以采用任何标记语言编写，提高工作效率。下面是一个采用 Struts 标签库编写的 XML 文档，输出之前提到的 employees.xml：

Sample8_2.jsp：

```
<%@ page contentType="application/xml; charset=gb2312" import="Employee"%>
<%@ page import="java.util.Collection,java.util.ArrayList"%>
<?xml version="1.0"?>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>
<%
Employee em1 = new Employee();
em1.setName("J.Doe");
em1.setJob("Programmer");
em1.setSalary("32768");
Employee em2 = new Employee();
em2.setName("A.Baker");
em2.setJob("Sales");
em2.setSalary("70000");
Employee em3 = new Employee();
em3.setName("Big Cheese");
em3.setJob("CEO");
em3.setSalary("100000");
Collection employees = new ArrayList();
employees.add(em1);
employees.add(em2);
employees.add(em3);
pageContext.setAttribute("employees",employees);
%>
<employees>
<logic:iterate name="employees" id="employee">
    <employee name="<bean:write name='employee' property='name'/>">
        <job><bean:write name="employee" property="job"/></job>
        <salary><bean:write name="employee" property="salary"/></salary>
    </employee>
</logic:iterate>
</employees>
```

```
<?xml version="1.0" ?>
- <employees>
  - <employee name="J.Doe">
      <job>Programmer</job>
      <salary>32768</salary>
    </employee>
  - <employee name="A.Baker">
      <job>Sales</job>
      <salary>70000</salary>
    </employee>
  - <employee name="Big Cheese">
      <job>CEO</job>
      <salary>100000</salary>
    </employee>
  </employees>
```

采用页面模板生成 XML 方式，需要为每个需要的的数据模型建立一个对立的 JSP 文件，用来生成符合规范的 XML 文档，而不能仅仅在类的 toXML()方法中组织对象图来实现。不过，倒是可以更加方便的确保标记匹配、元素和属性的顺序正确以及 XML 实体正确转义。

参考资料中 Philip McCarthy 的文章还描述了一种 Javascript 对象标注的生成方式，本文在此不赘述。有兴趣的读者可以自行查看了解。

7.5.3、如何在使用 XML 还是普通文本间权衡

使用 XML 文档确实有其方便之处。不过 XML 文档的某些问题倒是要考虑一下，比如说延迟，即服务器不能立即解析 XML 文档成为 DOM 模型。这个问题在一定程度上会影响 AJAX 要求的快速反应能力。另外，某些情况下我们并不需要使用 XML 来表示数据，比如说数据足够简单成只有一个字符串而已。就好像我们之前提到的数据校验和级联菜单的例子一样。所以，个人认为在下面这些情况下可以考虑使用 XML 来作为数据表示的介质：

- 数据比较复杂，需要用 XML 的结构化方式来表示
- 不用考虑带宽和处理效率支出
- 与系统其他 API 或者其他系统交互，作为一种数据中转中介
- 需要特定各式的输出视图而文本无法表示的

总之，要认真评估两种表示方式的表示成本和效率，选择合适的合理的表示方式。

在关于 AJAX 的系列文章的下一篇，我们将综合使用 DOM 和 XML，来实现一个可以持久化的简单留言簿。另外，还将试着模拟 MSN Space 的部分功能，来体会 AJAX 的魅力。

参考文章：

作者：	fanscial	标题：	《AJAX 简介》
网址：	http://www.blogjava.net/fanscial/archive/2005/08/31/11628.html		
作者：	Amour GUO	标题：	《AJAX 内部交流文档》

网址：	http://www.dragonson.com/doc/ajax.html		
作者：	MoztwWiki	标题：	《AJAX 上手篇》
网址：	http://wiki.moztw.org/index.php/AJAX_%E4%B8%8A%E6%89%8B%E7%AF%87		
作者：	Philip McCarthy	标题：	面向 Java 开发人员的 Ajax:构建动态的 Java 应用程序
网址：	http://kb.csdn.net/java/Articles/200510/bed0423e-5297-49c9-9464-0e3eb733c8b5.html		
作者：	Philip McCarthy	标题：	面向 Java 开发人员的 Ajax:Ajax 的 Java 对象序列化
网址：	http://kb.csdn.net/java/Articles/200510/a5b630cf-a2c2-46f1-8e3b-eadde723e734.html		
作者：	David Flanagan	书名：	《Javascript 权威指南》