第 21 章　AJAX开发

AJAX 是随着 Web2.0 时代的到来而产生的，对于传统的 Web 应用开发来说，AJAX 所运用的是更加先进、更加标准化、更加和谐高效以及完整的 Web 开发技术体系。

之前，在使用浏览器访问页面时，当你刷新一个页面的时候（特别是在网站访问很慢的时候），这时浏览器会怎么样？这时的浏览器会出现一片空白，而你只能在浏览器面前白白苦等。传统的 Web 应用常常在页面中添加一个 DIV，来显示“系统正在处理您的请求，请稍等…”类似的提示信息。

这一章将要的介绍的 AJAX 就可以彻底地改变这种窘迫的局面，从而可以大大提高用户体验。如今，随着 Gmail、Google-maps 等产品使用了 AJAX 之后，AJAX 正逐渐吸引着全世界的注意。

本章要点包括以下内容：

- 基于 Ajax 的开发模式介绍
- Ajax 所使用到的技术
- Ajax 详细的开发过程
- Ajax 主要应用范围
- Ajax 尚存在的问题

21.1　Ajax介绍

网络的迅速发展，已经悄悄地改变着人们的生活方式。有越来越多的人习惯从网络中寻找自己想要的信息和知识，不再单纯地从书籍中获取。网络通信的兴起，人们也已经从传统的电话书信联系转变成了使用电子邮件以及及时通信软件（例如 MSN）来和自己的朋友联系。电子商务技术也可以让人们在家中通过网络就可以购买到自己想要到的东西。网络拉近了人们之间的距离，方便了人们的生活。

BS 结构的 Web 应用正是凭借网络的广泛普及性，加上客户端仅仅需要安装了浏览器即可方便浏览站点信息。虽然在很多方面，BS 结构的 Web 应用比桌面应用程序更加优越，但正是由于它拥有客户端广泛适应和灵活性，而失去丰富客户端的控制能力。即之前的 Web 应用属于瘦客户程序，大量的计算和操作都集中在服务器端执行。

Ajax 的出现在一定程度上改变了以往 Web 应用所存在的不足，虽然 Ajax 并非一个革命性的新技术，但是它依靠多个成熟技术的整合并扮演着一个崭新的角色。Ajax 首先在 google 中得到了成功使用，从而使它得到了广泛的关注和推崇。那么 Ajax 到底是什么了？这一章将慢慢地揭开 Ajax 的庐山真面。

Ajax 的全称是 Asynchronous JavaScript And XML（异步 Javascript 和 XML），它可以实现客户端的异步请求操作。首先读者需要明白的，Ajax 并不是一个崭新的技术，它是多个已经非常成熟技术的集合。但是 Ajax 现在能够吸引这么多程序员的关注，是由于它具有独特的魅力以及能在 Web 开发中扮演着一个崭新的角色。它可以实现类似桌面应用程序一样的胖客户端，有更多的控制能力。

但是与 Applet 和 Flash 相比较，Ajax 则是一种轻量级的解决客户端方案。因为 Ajax 操作的基础仍然是 HTML 或者 XHTML 静态页面，它使用 JavaScript 作为脚本语言，这就保证了它的纯文本性质。

简单地来说，Ajax 具有如下多个优势：它具有更好的搜索引擎友好性；设计出色的 Ajax 程序还可以很好的在旧版本的设备上工作；利用 XML 或者 Json 结构数据，Ajax 可以很好地和其他应用程序（一

般为后台动态页面程序，例如 JSP、Servlet 等）进行通信。

传统的 Web 应用请求机制，是用户请求之后，等待后台重新组装整个页面传会给客户端显示。这样的弊端是网络承载了很多无用数据的传递。最为极端的情况，即用户在页面中只需要得到某一条信息，结果，后台服务器最确将插入该条信息的整个页面传递过来，这造成网络负载加重。另外，不管用户在客户端发出什么样的请求，都需要等待页面重新刷新显示，这大大影响了用户的体验。

正是基于传统 Web 应用存在这么多的不足，Ajax 就显得非常及时和受欢迎。Ajax 实现页面异步请求，再也不需要进行页面的刷新，结合 DOM 技术可以只修改页面局部需要修改的信息。并且，Ajax 使得网络只传递有用信息成为了可能，这大大减轻了网络传输的负担。

Ajax 能受到这么多人关注的另一个重要原因是它完全基于成熟的技术，作为异步调用的基础设施 XMLHttpRequest，其实早在 1999 年就被引入到 IE 浏览器中，随后又被其他浏览器所支持。另外，JavaScript、DOM、CSS 也早就是 W3C 标准。所以说，学习 Ajax 就是学习如何有效集合这些技术的使用。

本文通过与传统 Web 应用进行对比来介绍 Ajax 的运行机制。传统 Web 应用一般采取“请求”—“刷新”—“显示”的模式。用户在客户端一般通过单击按钮或者链接来发送一个 Web 请求，服务器接受请求并进行相应处理，处理完成之后，重新组装页面并返回给客户端进行显示。在服务器进行处理的这段时间内，客户端会出现什么情况了？这时客户端浏览器将一直处于 Loading 状态，显示为空白或者无响应状态，用户能做的只有等待。从用户体验来说，传统 Web 执行机制是非常糟糕的。

事实上，有时用户仅仅想获取到某一条信息，但是传统的 Web 执行机制将会为这一个小小的请求不得不重新刷新整个页面，使得页面中的所有图片和数据进行重新下载和运行。这不仅仅加大了网络流量，而且造成用户体验质量的下降。

下面先详细介绍传统 Web 应用的运行模式，然后与 Ajax 进行对比。

21.2　传统Web开发模式

由前面的介绍可知，传统的 Web 执行模式是“请求”—“刷新”—“显示”，每一个请求都会对应一个页面的显示，发送一个请求就会重新获取一个页面，也就是通常所说的刷新。在这样的执行模式下，一个 Web 应用系统一般由多个页面组成，每一个页面对应一个业务处理逻辑，或者成为功能块。而在客户端显示的页面实际上只是一个纯界面性的东西。

从图 21.1 可以更加清晰了解到该传统模式的执行过程。在客户端显示的各个页面都对应着服务器端的一个页面，而整个 Web 应用程序就是由这些页面组成。Web 应用的各个子功能可能对应一个或者多个页面，这取决于 Web 设计者的考虑。虽然传统 Web 应用的开发方式非常灵活，但是它存在的不足也是很明显的，就是当用户需要获取一条信息时，它必须将这一条信息对应到整个页面上，并将整个页面作为整体提供给用户，而不管页面中是否包含冗余信息。

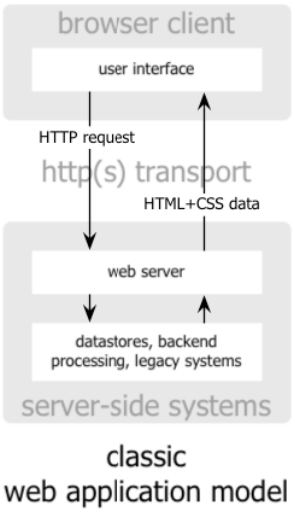

图 21.1　传统 Web 模式

其实一个页面中包含很多静态不变的元素，比如整体框架的 HTML 元素，用户界面格式显示的图片和 CSS 样式表等。这些不变元素在每次请求中也需要重新加载会大大增加网络负载。虽然为了解决这个问题，很多开发者尽量将页面粒度降低，但是还是不如 CS 架构的桌面应用程序。

21.3　基于Ajax的开发模式

和传统的 Web 应用相比较，Ajax 向传统的桌面程序靠拢了一点，但是它又不存在桌面程序存在的不足（需要安装客户端程序，每次升级，客户端同样需要下载升级包）。使用 Ajax 开发的 Web 页面可以从多个接口获取数据，并将它们更新在页面中（通常通过 DOM 来修改页面显示数据，而不需要整体刷新，这样不会出现白屏现象）。

在 Ajax 程序中，每个客户端显示页面不一定对应一个服务器端页面，这时的服务器端页面更多不再是界面显示工具，而是作为提供数据的接口。在 Ajax 中，使用 XMLHttpRequest 对象能够获取到这些页面所返回的信息，然后将其提交给客户端页面的 Ajax 引擎，再由 Ajax 引擎来决定将这些数据插入到页面确定的位置（通常使用 DOM 技术，将在后面重点介绍）。由图 21.2 可以更加清晰地了解到 Ajax 的执行模式。

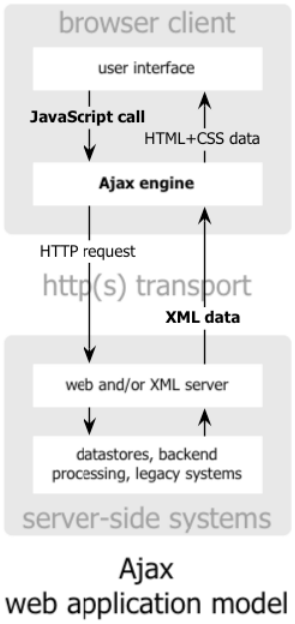

图 21.2　基于 AJAX 的 Web 模式

从图中可以看出，每一个客户端显示页面是由多个服务器端页面程序共同完成的，另外，一个服务器端页面也可以为多个客户端页面服务。在这样的模式下，服务器端的页面可以将粒度划分地非常细。Ajax 引擎根据需要和服务器端的页面程序进行交互，并将需要的数据返回到客户端页面，然后插入到页面相应位置。其实，这时的服务器端页面已经再称为页面了，更类似与接口。

21.4　两种执行模式的比较

之前 XMLHttpRequest 一直被人们所忽视，最近才被重视。其实它可以使服务器端的页面和浏览器之间进行异步通信，而不需要再进行刷新整个页面的操作，大大提高了用户的体验以及降低了网络的流量。下面将对这两种模式的工作流程作进一步的比较。

21.4.1　传统Web模式的工作流程

和 Ajax 的异步概念相比，传统 Web 模式是一种同步。用户必须等待每个请求返回的响应，才能进行下一步操作。下面的时序图 21.3 描述了传统 Web 模式的工作流程图。

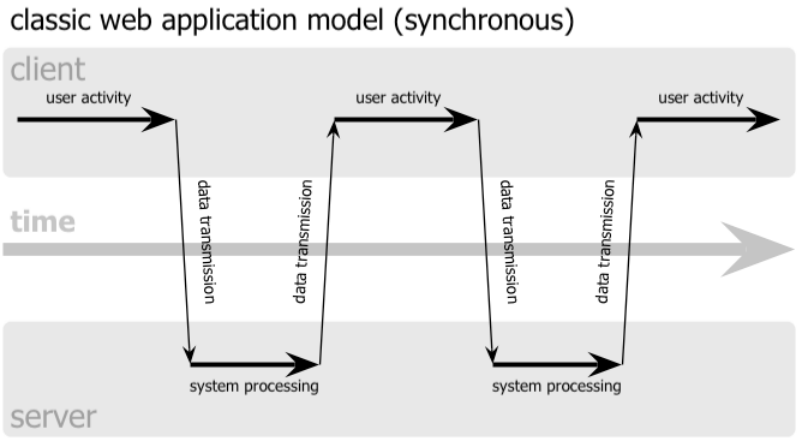

图 21.3　传统 Web 工作流程

从图中可以看出，传统 Web 模式完全遵循“请求”—“刷新”—“显示”模式。用户发出一个请求后，只有等请求处理完后，用户才能进行下一步操作，这期间用户只能等待。用户行为和服务器行为是一种同步的关系，需要相互等待，也正是这样的等待，造成了 BS 结构客户端和 CS 结构客户端的本质区别。

21.4.2　Ajax程序的工作流程

Ajax 最重要的功能就是将 Web 请求和处理从同步转变成了异步。这就意味着客户端和服务器端再也不必相互之间等待，而是可以进行一些并发的执行。形象一点讲，用户在页面进行相关浏览或者其他请求操作时，后台服务器可能正在执行用户前一个请求，用户不需要等待前一个请求处理完就可以进行下一步操作。当后台服务器处理完一个请求之后，会自动将处理结果返回给前台的 Ajax 引擎来更新页面中的部分数据，并且不会像传统 Web 应用模式一下传递大量的冗余数据。

从下面的 Ajax 工作流程图 21.4 可以看出，用户行为和服务器行为之间多了一层 Ajax 引擎，它负责处理用户的行为，并转发给服务器进行响应处理。同时它也接受服务器端返回的数据，经过处理之后在页面相应位置显示。

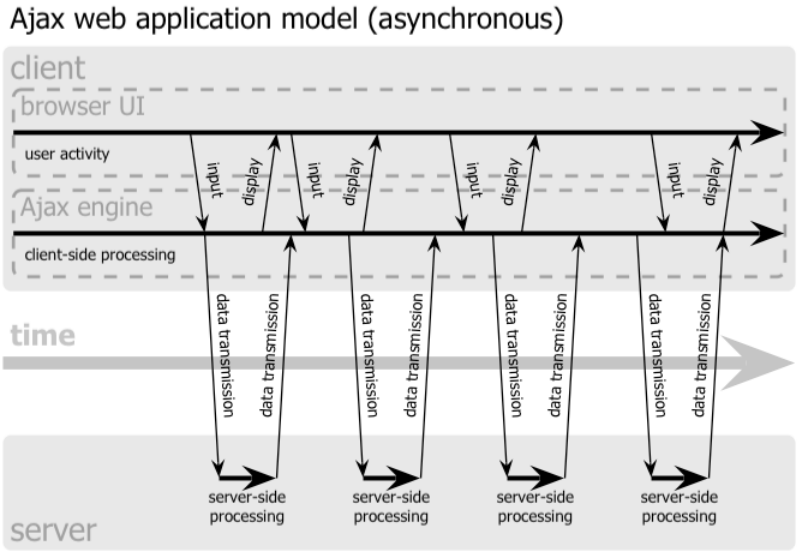

图 21.4　AJAX 程序的工作流程

由于 Ajax 在后台是以异步形式进行工作的，所以用户再也不需要等待服务器的处理，即可以进行下一步操作，这就在用户界面层次上更加地接近了 CS 结构地客户端平台。

注意：**同步和异步只是两种开发模式相区别的一种**，Ajax **还会带来整体** Web **性能的提高**：**由于** Ajax **引擎对用户行为进行了相应处理，所以可以保证客户端只获取需要的数据。**DOM **模型可以实现动态修改页面的层次结构，这样动态获取的数据可以动态地插入的页面相应位置，这就避免了大量冗余数据的重新加载。**

21.5　AJAX带来的好处

总结起来，使用 Ajax，可以为 ISP、开发人员以及终端用户带来如下可见的便捷：

（1）减轻服务器的负担。Ajax 的原则是“按需求获取数据”，它可以最大限度的减少冗余数据的传递，从而可以减轻网络传输负担。

（2）无须再刷新页面，用户再也不会在黑白页面中焦急等待，这样减少了用户心理和实际的等待时间。Ajax 使用 XMLHTTP 对象来发送请求并得到服务器响应，在不需要重新装载整个页面的情况下就可以使用 JavaScript 来操作页面的 DOM 节点，从而可以更新页面数据，即将用户请求结果返回到页面中。

（3）可以给用户带来更好的体验，因为用户再也不需要在服务器进行请求处理时进行无用的等待。

（4）使用 Ajax 可以将以前的一些服务器负担移到客户端来完成，利用客户端闲置的资源来处理，从而可以减轻服务器端和宽带的负担，节约了空间和成本。

（5）Ajax 结合 XML 或者 JSON 可以轻松得获取到外部数据。

（6）Ajax 还保存着完全静态的特性（虽然它已经实现了动态功能），并且它已经被广泛得到其他技术支持，并不需要下载插件或者小程序。

（7）促进了页面显示和数据的进一步分离。

21.6　谁在使用AJAX

在应用 Ajax 来开发 Web 应用方面，Google 是当仁不让的表率。它的 Orkut、Gmail、Google Groups、Google Maps 以及 Google suggest 产品都是使用的 Ajax 技术。Amazon 的 A9.com 搜索引擎也是采用的该类似技术。

典型的，微软也将 Ajax 技术应用到了 MSN Space 上面。刚用 MSN Space 的用户，一定对其服务感到很奇怪，当提交回复评论的时候，浏览器会暂时停顿一下，然后在无刷新的情况下将用户评论提交到后台，并同时显示在页面中。这就是应用了 Ajax 效果，如果真的对一个评论都要重新刷新整个页面，那真的是很费事。

21.7　AJAX使用到的技术

AJAX 中使用到的最为核心的技术是浏览器脚本语言 JavaScript。它像一个强大的黏合剂，综合了

DOM、XML 以及 CSS 样式表等技术并发挥着巨大作用。其中 JavaScript 和 CSS 的基本概念和语法都已经在第五、六章介绍过。另外，读者需要重点了解的是 AJAX 中进行异步请求的 XMLHttpRequest 对象的使用。下面对这些涉及到的技术进行逐一重点介绍。

21.7.1　JavaScript脚本语言

在网页中出现的各种特效，例如字符串组成的时钟、滚动的状态栏、漫天飘舞的雪花等等，这些都是 JavaScript 的功劳。

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的数据校验。在 Ajax 中，JavaScript 确从幕后走到了前台，并发挥着巨大的作用。可以说，它 JavaScript 是 AJAX 的黏合剂，它综合 DOM、XHTML（或者 HTML）、XML（或者 JSON）以及 CSS 等技术，并控制它们的行为。

现在 Web 程序中基于 JavaScript 实现的很多功能已经具有相当大的实用性，不再是一些特效或者表单验证之类的简单应用了，这些高级应用所使用到的 JavaScript 也已经相当专业的。因此，要开发出一个复杂高效的 AJAX 应用程序，就必须对 JavaScript 有深入的了解、应该对待一门新的语言来对待它。

21.7.2　XMLHttpRequest对象

AJAX 实现异步处理，就是通过 XMLHttpRequest 对象实现的，它是 XMLHTTP 组件的一个对象。通过该对象的调用，AJAX 可以实现像桌面应用程序一样同服务器进行数据层面的交换，而不需要每次请求都要刷新整个页面，也不需要每次数据操作都要交付给服务器去完成。AJAX 实现的 Web 应用可以大大减轻服务器负担，而且还加快了响应速度、缩短用户等待时间，因为 HTTP 只传输需要的数据。

IE5.0 开始，开发人员可以在 Web 页面中使用 XMLHTTP ActiveX 组件来扩展自身的功能，不再需要使用 Web 页面导航就可以直接将数据传递到服务器或者从服务器那获取到相应数据。Mozilla1.0 以及 NetScape7 则是创建了继承 XML 的代理类 XMLHttpRequest；在大部分情况下，XMLHttpRequest 对象和 XMLHTTP 组件相类似，方法和属性也非常类似，只是部分属性可能存在差异。

在使用 XMLHttpRequest 进行异步处理之前，需要对该对象初始化操作，根据不同的浏览器，初始化方法是不同的.

（1）IE 浏览器的初始化方法如下：

```
Var xmlHttp = new ActiveXObject(“Msxml2.XMLHTTP”);
```

或者

```
Var xmlHttp = new ActiveXObject(“Microsoft.XMLHTTP”);
```

括号中的不同参数是针对不同版本的浏览器。还有其他参数类型，只是现在最为常见的就是这两种，需要考虑的也就是这两种。

（2）Mozilla 浏览器初始化方法如下：

```
Var xmlHttp = new XMLHttpRequest();
```

总的 XMLHttpRequest 对象初始化程序如下：

```
<script language="javascript" type="text/javascript">
<!--
var xmlhttp ;
try{
   xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
}catch(e){
      try{
```

```
            xmlhttp= new ActiveXObject("Microsoft.XMLHTTP");
        }catch(e){
            try{
                xmlhttp = new XMLHttpRequest();
              }catch(e){}
        }
}
-->
</script>
```

XMLHttpRequest 对象中所包含的一般方法如下表 21.1 所示：

表 21.1　XMLHttpRequest对象方法

方法	描述
Abort()	停止当前的异步请求
getAllResponseHeaders()	以字符串形式返回完整的HTTP头信息
getResponseHeader(“headerLabel”)	使用headerLabel标识来获取指定的HTTP头信息
Open(“method”,”URL”[asyncFlag,][“username”,][“password”]	设置进行异步请求的URL、请求方法以及其他参数信息
Send(content)	发送请求
setRequestHeader(“label”,”value”)	设置HTTP头信息，并和请求一起发送到服务器

其中比较常用的是 open()和 send()两个方法。其中 open()方法在创建完了 XMLHttpRequest 对象之后就可以使用它了，它创建一个请求，并准备向服务器发送，基本语法如下：

```
Xmlhttp.open(method,url,asynchronous,user,password);
```

例如：

```
Xmlhttp.open(“get”,”getData.jsp?key=8”,true);
```

其中参数含义如下：

- Method：指定请求的类型，一般为 get 或者 post。
- url：指定请求的地址，可以使用绝对地址或者相对地址，并且可以附带查询字符串。
- Asynchronous：这是一个可选项，表示请求是同步还是异步，异步请求为 false，同步请求为 true，默认情况下该参数为 true。
- User：可选参数，指定请求用户名，没有则省略。
- Password：可选参数，指定请求的密码，没有则省略。

在使用 open()方法创建了一个请求之后，就可以使用 send()方法向服务器发送该创建的请求，其基本语法如下：

```
Xmlhttp.send(content);
```

其中的 content 变量为记录发送的数据，其格式为查询字符串的形式，如下：

```
Content = “username=Johnson&sex=male&age=25”;
```

其中定义了发送给服务器的三个名/值对，并且在它们之间使用&符号隔开。如果在 open()方法中 method 参数指定为 get，则这些参数是作为字符串提交的，服务器端使用 request.querystring()来获取；如果 method 参数指定为 post，则这些参数作为 HTTP 的 post 方法提交，在服务器端需要使用 request.form()方法来获取。这里的 content 参数是必须要指定，如果没有参数值需要传递，可以设置的 content=null，则 xmlhttp.send(null)。

XMLHttpRequest 对象中所包含的属性如表 21.2 所示：

表 21.2　XMLHttpRequest对象属性

属性	描述
onreadystatechange	状态改变的事件触发器
readyState	对象状态： 0 ＝ 未初始化 1 ＝ 读取中 2 ＝ 已经读取 3 ＝ 交互中 4 ＝ 完成
responseText	服务器进程返回的文本信息
responseXML	服务器进程返回的兼容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

下面对 XMLHttpRequest 对象中的各个属性进行详细介绍：

（1）onreadystatechange 事件捕获请求的状态变化

在向服务器端发送了一个请求后，往往是不知道请求什么时候完成，所以必须使用一种机制来获取请求处理的状态。Onreadystatechange 状态变量就是 XMLHttpRequest 对象用来实现这一功能的。

Onreadystatechange 事件可以指定一个事件处理函数用来处理 XMLHttpRequest 对象的执行结果，例如：

```
Xmlhttp. Onreadystatechange=function(){
    //此处将对异步请求返回的结果进行处理
}
Xmlhttp.open();
Xmlhttp.send();
```

注意：这里将事件绑定在调用了 open()和 send()方法之后进行，是因为这两个方法都会触发 Onreadystatechange 事件。如果该事件在其后面定义，则可能引起该事件指定的处理代码得不到执行。

（2）使用 readyState 属性判断请求状态

Onreadystatechange 事件是在 readyState 属性发生改变的时候触发执行的，该属性记录着当前请求的状态，然后在程序中可以根据不同请求状态进行不同的处理。下表 21.3 列出各种不同属性值的含义。

表 21.3　readyState属性值含义

readyState属性值	描述
0	这是readyState开始具有的值，表示对象已经建立，但还未初始化，这时尚未调用open方法。
1	表示open已经调用，但是尚未调用send方法
2	表示send方法已经调用，但是其他数据还未知
3	表示请求已经发送，正在接受数据过程中
4	表示数据已经接受成功。

在整个过程中，Onreadystatechange 事件在每次 readyState 属性值发生改变的时候都会执行一次，所以通常在事件中判断 readyState 值，然后根据不同状态来作适当的处理。例如：

```
Xmlhttp.onreadystatechange=function(){
  If(xmlhttp.readystate == 4){
   //请求完毕时的执行处理代码
}

}
```

这里的 readyState 仅仅表示请求的状态，并不表示请求的结果，下面介绍的 status 属性时用来判断请求结果。

（3）使用 status 属性判断请求的结果

Status 属性存储服务器端返回的 http 请求响应结果码，下表 21.4 列出了常见的响应结果码含义。

表 21.4　http响应状态码

http响应状态码	描述
200	请求成功
202	请求被接受，但是尚未完成
400	错误的请求
404	请求资源未找到
500	内部服务器错误，例如后台jsp处理文件语法错误

在使用 readyState 属性判断请求完成之后，可以使用 status 属性判断请求处理结果。在 AJAX 开发过程中，最为常见的响应代码为 200，它表示请求处理成功，例如下面的一段 AJAX 代码：

```
Xmlhttp.onreadystatechange=function(){
    If(xmlhttp.readystate == 4){
        //判断请求成功完成
            If(xmlhttp.status == 200){
                //判断请求操作成功
            }else{
             //操作错误或者未完成
            }
    }
}
```

以上代码是一个比较完整的 XMLHttpRequest 对象请求处理代码。

（4）使用 responseText 获得返回的文本

当服务器已经成功处理完请求之后，就可以使用 XMLHttpRequest 对象中的 responseText 属性来获取返回的文本或者 html 页面数据。例如下面的一段 html 页面代码，该页面文件名为 responseText.html：

```
<script language="javascript" type="text/javascript">
var xmlhttp;
try{
    xmlhttp = new ActiveXObjt("Msxml2.XMLHTTP");
}catch(e){
    try{
        xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
    }catch(e){
        try{
            xmlhttp = new XMLHttpRequest();
        }catch(e){}
    }
}
//下面定义 XMLHttpRequest 对象的事件处理程序
xmlhttp.onreadystatechange = function(){
    if(xmlhttp.readyState == 4){
      if(xmlhttp.status == 200){
        var showHtml = xmlhttp.responseText;
        alert(showHtml);
}
```

```
else
   alert(“处理错误”);
}
}
//创建一个请求连接
Xmlhttp.open(“get”,”a.html”,true);
Xmlhttp.send(null);
</script>
```

其中 a.html 静态页面代码如下：

```
<html>
<head>
<title>new Document</title>
</head>
<body>hello,ajax!</body>
</html>
```

运行 responseText.html 页面，弹出的对话框如图 21.5 所示。

图 21.5　responseText 属性返回值

使用 XMLHttpRequest 对象中的 responseText 属性可以获取到请求页面的纯文本信息，并不经过任何处理。并不是每次都要求返回 html 页面代码，有时可以使用 json 数据格式来返回只需要的数据。

（5）使用 responseXML 属性来获取 XML 文档信息

除了可以使用 responseText 属性来获取纯文本信息之后，还可以使用其中的 responseXML 属性来获取服务器返回的 XML 文档对象。在使用 responseText 属性来获取返回的纯文本信息时，后台服务器是不需要进行额外操作的。但是 responseXML 属性不同，它需要服务器返回的必须是一个严格符合 XML 格式的文档，这就要求返回的 HTTP 头的 content-type 设置为“text/XML”。

如果服务器直接获取某个 XML 文档并返回（例如将前面的 a.html 改名为 a.xml），这时因为请求后缀为 xml，所以服务器会自动将返回结果的“content-type”属性设置为“text/xml”。如果是通过后台服务器程序动态生成的 XML，则需要设置 Response[“content－type”]为“text/xml”。

获取的 responseXML 属性实际上是一个 DOM 文档对象模型说规定的文档对象，具有 DOM 模型的操作规范。这里将 a.html 文件改名为 a.xml，然后修改上面实例中的事件触发函数代码，其他部分不需要修改。

```
Xmlhttp.onreadystatechange = function(){
    If(xmlhttp.readyState == 4){
      If(xmlhttp.status == 200){
        Var xmlObj = xmlhttp.responseXML;
        Var title = xmlObj.getElementByTagName(“title”)[0].text;
        Alert(title);
```

```
}
Else
   Alert(“处理错误”);
}
}
```

重新运行该页面，弹出如图 21.6 所示对话框信息。

图 21.6　responseXML 属性返回信息

21.7.3　DOM文档对象模型

DOM 文档对象模型是 Ajax 开发中一个重要的技术，下面从实际运用角度出发来介绍 DOM 的使用方法。

21.7.3.1．什么是 DOM 模型

DOM 模型全称 Document Object Module（文档对象模型），它定义了操作文档对象（例如 XML 文档）的接口。例如 XML 文档，它就像一棵树的结构，树中的每一个结点都对应一个 XML 标记，及一个对象。

实际上，DOM 模型更多的是一个对象模型，它并不依赖于对象的结构，树形结构只是它其中的一个实现。DOM 规定了每个对象具有哪些接口，例如添加结点、删除结点等等。在 AJAX 中，通常使用这些接口来改变文档的状态，从而达到页面动态显示的目的。

现在的 DOM 模型主要有三个部分。

- ❑ 核心：这部分包括最为底层的文档操作接口，适合 HTML 和 XML 文档类型。
- ❑ HTML：这部分内容是针对 HTML 进行操作的接口。
- ❑ XML：该部分是针对 XML 进行操作的接口。

21.7.3.2．DOM 模型在 AJAX 开发中所起的作用

在 AJAX 中，其实 DOM 模型是非常重要的，是所有 AJAX 开发的基础结构。如果没有 DOM 模型，就没有办法在客户端改变页面内容。没有掌握 DOM 模型的相关技术，是不能真正掌握 AJAX 开发精髓的。

21.7.3.3．处理 DOM 中的结点

在 DOM 模型中引用一个结点对象可以多种方法，如下：

（1）document.getElementById()引用指定 id 的结点

在 HTML 或者 XHTML 中，每一个结点对象都可以定义一个 id 属性。但是每个结点定义的 id 属性值必须在整个文档中惟一。另外可以使用 document 中的 getElementById()方法来获取到该结点对象。例如下面的代码：

```
<html>
<head>
<title> document.getElementById()方法 <.title>
```

```
<script language=”javascript”>
var divItem = document.getElementById(“div1”);
alert(divItem.innerHTML);
</script>
</head>
<body>
<div id=”div1”>hello，Ajax！</div>
</body>
</html>
```

代码说明：该段代码将输出 id 属性名为“div1”的 div 结点对象中的文本信息。具体 innerHTML 属性的意义见下面的讲解。

（2）document.getElementByTagName()引用指定标签名的结点对象

使用 document.getElementByTagName()可以返回某文档中所有指定标签名的结点对象组，具体示范实例如下：

```
<html>
<head>
<title> document.getElementById()方法 <.title>
<script language=”javascript”>
var divItems = document.getElementByTagName(“div”);
for(var i=0;i<divItems.length;i++)
    alert(divItems[i].innerHTML);
</script>
</head>
<body>
<div id=”div1”>hello，Ajax！</div>
<div id=”div2”>hello，Ajax2！</div>
</body>
</html>
```

代码说明：由于该页面文档中存在两个标签名为“div”的结点对象，所以使用 document.getElementByTagName()方法返回的是包含两个结点对象的数组，使用 for 循环将这两个结点中的文本信息输出。

21.7.3.4．间接引用结点对象

上面介绍的方法是通过结点对象中的 id 以及结点名称来获取结点对象的。本小节将介绍如何使用间接方法获取相对应的结点对象，具体方法如下：

（1）使用 childNodes 结合属性引用子结点

其实在 JavaScript 中，每个结点都有一个 childNodes 集合属性，是以数组形式存储着该结点下所有的子结点信息。这些子结点的排序顺序是按照在页面文档中出现的先后顺序排列的。这样就可以通过数组索引来引用一个结点下的所有子结点对象。例如获取 HTML 文档根结点，代码可以如下：

```
document.childNodes[0]
```

除了使用 childNodes 属性之后，还可以使用 firstChild 和 lastChild 属性来获取第一子结点和最后一个子结点对象。具体代码如下：

```
Element.firstChild      //获取 element 结点对象中的第一子结点对象
Element.lastChild       //获 element 结点对象中的最后一个子结点对象
```

下面通过一个实例代码综合演示 childNodes 属性以及 firstChild 和 lastChild 属性的使用：

```
<html>
<head>
```

```
<title> document.getElementById()方法 <.title>
<script language=”javascript”>
var divItem = document.getElementById(“div1”);
alert(divItem.firstChild.childNodes[0]);                    //输出“Ajax1”
alert(divItem.firstChild.lastChild);                        //输出“Ajax2”
</script>
</head>
<body>
<div id=”div1”><span><span>Ajax1</span><span>Ajax2</span></span></div>
</body>
</html>
```

（2）使用 parentNode 属性获取父结点

在类似于 HTML 文档结构中，除了根结点之后，每个结点都有一个惟一的父结点，并可以使用 parentNode 属性来获取到，语法代码如下：

```
elemnt.parentNode
```

其中 element 为某个结点对象。

（3）获取兄弟结点对象

定义在同一层次上的所有结点对象称为兄弟结点。使用下面的两个属性可以引用兄弟结点，具体语法代码如下：

```
element.nextSibling                 //引用上一个兄弟结点
element.previousSibling                 //引用下一个兄弟结点
```

21.7.3.5．获取结点信息

其实获取一个结点对象之后，就是为了要获取到该结点下的文本信息，具体方法有如下：

（1）使用 nodeName 属性获取结点名称

该属性语法如下：

```
Node.nodeName
```

但是针对不同的结点类型，nodeName 属性返回不同的值：

- ❑ 元素结点：返回标签名称，例如，则返回“span”；
- ❑ 属性结点：返回属性名称，例如 id=”span1”，则返回“span1”；
- ❑ 文本结点：返回文本内容。

（2）使用 nodeType 属性获取结点类型

该属性使用的语法如下：

```
Node.nodeType
```

对应不同类型结点，nodeType 属性也返回不同值：

- ❑ 元素结点：返回 1；
- ❑ 属性结点：返回 2；
- ❑ 文本结点：返回 2。

（3）使用 nodeValue 属性来获取结点的值

该属性语法代码如下：

```
Node.nodeType
```

不同类型的结点返回不同值：

- ❑ 元素结点：返回 null；
- ❑ 属性结点：undefinded；
- ❑ 文本结点：返回文本内容。

（4）使用 hasChildNodes()属性来判断该结点是否包含子结点

该属性语法代码如下：

```
Node. hasChildNodes()
```

如果 Node 结点有子结点，则返回 true；否则没有子结点则返回 false。

（5）使用 tagName 属性返回某结点的标签名称

该属性仅仅为元素类型的结点独有，和 nodeName 具有相同的返回值，即返回结点标签名。

21.7.3.6. 处理属性结点

下面逐一介绍用于操作属性结点的方法：

（1）获取和设置属性结点的值

下面还是通过一个简单的实例来演示一下，例如存在如下的一个属性结点：

```
<img id=”img1” src=”hello.jpg” alt=”demo” />
```

下面编写 javascript 代码来操作这个结点：

```
<script language=”javascript” type=”text/javascript”>
Var itemImg = document.getElementById(“img1”);          //获取结点对象
Alert(itemImg.src);                                     //输出结点对象中的 src 属性值
itemImg.src = “ok.jpg”;                                 //设置结点对象中的 src 属性值
alert(itemImg.src);
</script>
```

代码说明：由此可以见，当引用了一个属性对象之后，对其属性进行设置或获取是非常方便的。

（2）使用 setAttribute()方法来添加一个新属性

该方法的语法代码如下：

```
element.setAttribute(attributeName,attributeValue);
```

代码说明：其中 element 为某一结点对象；attributeName 为新添加的属性名，attributeValue 为给新添加 attributeName 属性设置的值。

（3）使用 getAttribute()方法来获取一个属性值

该方法的语法代码如下：

```
element.getAttribute(attributeName);
```

代码说明：其中 element 为某结点对象，attributeName 为该结点对象中的某个属性名。

21.7.3.7. 处理文本结点

通过获取一个结点内的文本信息使用 innerHTML 属性，例如对于如下一个结点：

```
<span id=”span1”>hello,Ajax</span>
```

使用 innerHTML 属性获取该结点内的“hello,Ajax”文本信息的方法如下：

```
Document.getElementById(“span1”).innerHTML;
```

其实“hello,Ajax”这个文本信息也可以单独看着一个文本结点，因此可以通过通用的结点处理方法来获取它的值，这种方法已经在前面介绍，具体代码如下：

```
Document.getElementById(“span1”).childNodes[0].nodeValue;
```

代码说明：Document.getElementById(“span1”).childNodes[0]是获取到“hello,Ajax”这个文本结点对象，然后再调用这个结点对象的 nodeValue 属性来获取值。

但是有些情况下，必须使用文本结点处理方法来获取一段文本信息：例如：

```
<div id= “div1”>
    <img src=”xx.gif” alt=”xx” />
    Hello,Ajax!
</div>
```

代码说明：以上的 div 结点内包含了两个结点对象，一个是 img 属性结点，另一个就是“Hello,Ajax!”文本结点。可以使用下面代码来获取该文本信息：

```
Document.getElementById(“div1”).childNodes[1].nodeValue;
```

注意：IE 浏览器会忽略各标签之间的空白文本结点，而 firefox 浏览器则会认为标签之间的空白也是一个文本结点，这一点读者在实际开发过程中要注意。为避免这样的差异，尽量不要在标签之间留空格。

21.7.3.8．使用 innerHTML 属性来修改结点的文本信息

使用 innerHTML 属性可以获取到嵌入标签内（例如开始标签 和结束标签 之间的内容）所有信息，如下代码：

```
<div id=”div1”>
    <span>hello</span>
    <input type=”text” value=”ddd”/>
</div>
```

接下来可以使用 innerHTML 属性来获取到 和 标签之间的所有内容，代码如下：

```
var itemDiv = Document.getElementById(“div1”)
alert(itemDiv.innerHTML);
```

同样可以使用 innerHTML 属性来修改 和 标签之间的内容，代码如下：

```
var itemDiv = Document.getElementById(“div1”)
itemDiv.innerHTML = “修改之后的内容”;
```

这样修改之后，上面的文档内容变为：

```
<div id=”div1”>
    修改之后的内容
</div>
```

21.7.3.9．修改文档的层次结构

（1）使用 document.createElement()方法来创建一个元素结点

创建一个元素结点的语法如下：

```
Document.createElement(NewElement);
其中 NewElement 为要创建的结点标签名称，例如：
Var divElement = document.createElement(“div”);
```

以上代码创建了一个标签名为 div 的结点对象。

（2）使用 document.createTextNode()方法创建文本结点

创建一个文本结点对象的语法如下：

```
Document.createTextNode(text);
```

其中 text 为文本信息，该方法返回的是一个文本结点对象，例如：

```
Var TextElement = document.createTextNode(“this is textNode”);
```

上面创建了一个文本信息为“this is textNode”的文本结点对象，并赋值给 TextElement 变量。

（3）使用 appendChild 方法在文档中添加已经创建好的结点

使用 document.createElement()或者 document.createTextNode()方法创建一个结点对象之后，还需要使用 appendChild()方法来将该结点添加到文档的适当位置。其语法如下：

```
parentElement.appendChild(childElement);
```

其中 parentElement 为新创建结点 childElement 的父结点，即将 childElement 结点添加到 parentElement 结点的子结点中（排在最后）。

（4）使用 insertBefore()方法来插入子结点

另外一个插入结点的方法是 insertBefore()，该方法可以将结点插入到指定结点的前面。语法代码如下：

```
ParentNode.insertBefore(newNode,referenceNode);
```

其中 parentNode 为新插入结点的父结点，newNode 为待插入的新结点；referenceNode 为父结点中已经存在的结点，最终将会把 newNode 结点插入到 referenceNode 结点的前面。

（5）使用 replaceChild()方法来取代子结点

其语法如下：

```
ParentNode.replaceChild(newNode,oldNode);
```

其中 parentNode 为父结点；newNode 为待插入结点；oldNode 是将被取代的结点。该方法将返回被取代的结点对象，但此时这个结点已经不存在该文档中。

（6）使用 cloneNode 来复制结点

在使用 DOM 进行结点操作时，有时会需要将一个结点复制到另外一个位置上，这时就会使用到 cloneNode()方法进行结点的复制操作。该方法语法如下：

```
Node.cloneNode(IsIncludeChildren);
```

其中 node 为被复制结点对象，而 IsIncludeChildren 为一个布尔值类型，true 表示也复制它所有的子结点，false 表示不复制子结点。

（7）使用 removeChild()方法来删除一个子结点

该方法用来删除一个指定的结点对象，其语法如下：

```
parentNode.removeChild(childNode);
```

其中 parentNode 结点需要删除结点的父结点对象，而 childNode 为将被删除的结点。

21.7.3.10．表格操作

虽然表格也对应着一定结构的 XML（HTML）标签，但是使用标准的 DOM 方法并不能正确操作这些表格结点。这时必须使用 DHTML 中定义的接口对其进行操作。另外下拉列表框，虽然在 firefox 浏览器可以识别 DOM 操作方法，但是 IE 浏览器并不支持，所以也必须使用 DOM1（DHTML）方法来进行操作。

（1）创建一个表格对象

创建一个表格对象的方法和生成其他结点对象的方法一样，语句如下：

```
Var table1 = document.createElement(“table”);
```

（2）添加一行

添加一行的方法是调用表格对象（例如上面创建的 table1 对象）的 insertRow 方法，其语法如下：

```
Var row1 = table1.insertRow(index);
```

其中 index 为插入表格 table1 的指定索引。0 表示第一行，1 表示第二行，依次类推。如果没有指定索引，则 IE 浏览器会自动插入到表格的最后一行，而 firefox 浏览器会报错。

（3）添加一个单元格

添加单元格的方法是使用表格行（例如前面创建的 row1）的 insertRow 方法，语法如下：

```
Row1.insertRow(index);
```

其中 row1 为表格行对象，index 为插入表格行中的索引。

（4）引用单元格方法

在 DOM1 中可以根据表格中某个单元格所在行和列的索引位置来引用某一个单元格对象，然后再对其进行相应操作。引用一个单元格的语法如下：

```
Table.rows[i].cells[j];
```

其中 table 为表格对象，i 为行的索引值，j 为单元格的索引位置。I 和 j 都是从 0 开始的，和数组一

样。

当需要使用到单元格中的文本信息时，同样是使用 DOM 中定义的 innerHTML 属性。

（5）删除行和单元格的方法

删除一个表格中的某一行的语法如下：

```
Table.deleteRow(index);
```

其中 index 为要删除行的索引位置。

删除一个表格中的某一单元格的语法如下：

```
Table.deleteCell(index);
```

其中 index 为一个表格中某一单元格的索引位置。

21.7.4　XML/JSON数据表示技术

下面再详细介绍一下如何使用 Ajax 读取 text 文本、JSON 数据以及 XML 数据。

21.7.4.1　用 Ajax 读取 Text 格式的数据

用 Ajax 读取 Text 格式的数据，只需要使用 XMLHttpRequest 对象返回的 responseText 属性即可。代码如下：

（1）创建放置在服务器端的 data.txt 文本文件

```
hello world!
```

（2）创建客户端的 helloworld.htm 文件，详细代码如下：

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

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

function startRequest(){
createXMLHttpRequest();
try{
xmlHttp.onreadystatechange = handleStateChange;
xmlHttp.open("GET", "data.txt", true);
xmlHttp.send(null);
}catch(exception){
alert("您要访问的资源不存在!");
}
}
```

```
function handleStateChange(){
if(xmlHttp.readyState == 4){
if (xmlHttp.status == 200 || xmlHttp.status == 0){
// 显示返回结果
alert("responseText's value: " + xmlHttp.responseText);
}
}
}
</script>
</head>
<body>
<div>
<input type="button" value="return ajax responseText's value"
onclick="startRequest();" />
</div>
</body>
</html>
```

运行该页面程序，可以查看到读取的后台 data.txt 文件中文本信息。

21.7.4.2　用 Ajax 读取 XML 格式的数据

用 Ajax 读取 XML 格式的数据，需要读取 XMLHttpRequest 对象返回的 responseXML 属性。代码如下：

（1）创建放置在服务器端的 XML 文件 data.xml，该文件内容如下：

```
<?xml version="1.0" encoding="GB2312" ?>
<root>
<info>hello world!</info>
</root>
```

（1）创建客户端文件 helloworld.htm，详细代码如下：

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

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

function startRequest(){
createXMLHttpRequest();
try{
```

```
xmlHttp.onreadystatechange = handleStateChange;
xmlHttp.open("GET", "data.xml", true);
xmlHttp.send(null);
}catch(exception){
alert("您要访问的资源不存在!");
}
}

function handleStateChange(){
if(xmlHttp.readyState == 4){
if (xmlHttp.status == 200 || xmlHttp.status == 0){
// 取得 XML 的 DOM 对象
var xmlDOM = xmlHttp.responseXML;
// 取得 XML 文档的根
var root = xmlDOM.documentElement;
try
{
// 取得<info>结果
var info = root.getElementsByTagName('info');
// 显示返回结果
alert("responseXML's value: " + info[0].firstChild.data);
}catch(exception)
{

}
}
}
}
</script>
</head>
<body>
<div>
<input type="button" value="return ajax responseXML's value"
onclick="startRequest();" />
</div>
</body>
</html>
```

21.7.4.3　用 Ajax 读取 JSON 格式的数据

JSON，全名为 JavaScript Object Notation，是一种轻量级的数据交换格式，它是基于 JavaScript 规范开发出来的，和 JavaScript 是完美组合。也可以说，它是 JavaScript 的一个子集，其目的就是形成一种便于阅读和解析的数据交换语言。它实际就是一种结构化的数据，充分利用了 JavaScript 语言的动态性以及对象定义的灵活性，使数据无需额外的解析并能够使用。有关 JSON 数据格式的编写非常简单，读者可以查询相关资料详细了解。

用 Ajax 读取 JSON 格式的数据，也需要先用 XMLHttpRequest 对象的 responseText 属性读取，然后再用 Function 构造返回 JSON 对象的方法。代码如下：

（1）创建服务器端的 JSON 数据格式文件 data.txt，详细内容如下：

```
{
info: "hello world!",
```

```
version: "2.0"
}
```

（2）创建客户端文件 helloworld.htm，详细内容如下：

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

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

function startRequest(){
createXMLHttpRequest();
try{
xmlHttp.onreadystatechange = handleStateChange;
xmlHttp.open("GET", "data.txt", true);
xmlHttp.send(null);
}catch(exception){
alert("您要访问的资源不存在!");
}
}

function handleStateChange(){
if(xmlHttp.readyState == 4){
if (xmlHttp.status == 200 || xmlHttp.status == 0){
// 取得返回字符串
var resp = xmlHttp.responseText;
// 得到 JSON 对象
var json = eval(resp);
// 显示返回结果
alert("JSON's value: " + json.info + "(" + json.version + "v)");
}
}
}
</script>
</head>
<body>
<div>
<input type="button" value="return ajax JSON's value"
onclick="startRequest();" />
</div>
```

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

运行该页面可以得到上面定义的 JSON 数据格式，然后在 JavaScript 中可以类似数组一样的使用它。

21.8　AJAX开发过程

通过上面的讲解，下面给出一个完整的 AJAX 开发过程。Ajax 实质上是遵循 Request/Server 模式的，所以整个开发流程如下：XMLHttpRequest 对象的初始化→发送请求→服务器接收数据→服务器返回相应处理结果数据→客户端接收→使用 DOM 修改客户端页面内容。以上的 AJAX 整个过程都是异步进行的，整个页面是不需要刷新的。

21.8.1　初始化对象并发送XMLHttpRequest请求

使用 JavaScript 实现异步请求，必须使用 XMLHttpRequest 对象，并且在使用之前，必须对它进行初始化，有关初始化方法在前面已经介绍过。针对不同的浏览器，有不同的初始化方法。IE 浏览器使用 ActoveX 控件的形式提供，而 Mozilla 等浏览器则直接一 XMLHttpRequest 类的形式提供。

有些版本的 Mozilla 浏览器在处理服务器返回的包含 XML mime-type 头部信息内容时会出错。所以，为了确保返回内容是 text/xml 信息，需要写上下面的语句：

```
Xmlhttp = new XMLHttpRequest();
xmlHttp.overrideMineType(“text/xml”);
```

21.8.2　定义响应事件处理函数

下面需要对 Onreadystatechange 事件定义一个触发的处理函数。其中有两个定义方法，一是直接将函数名赋给 XMLHttpRequest 对象的 Onreadystatechange 属性就可以，具体方法如下：

```
Xmlhttp.onreadystatechange = processRequest;
```

其中 processRequest 为 JavaScript 中定义的一个函数。这样指定之后，每发生一次触发事件，都会执行一次该函数。

注意：指定的函数不需要加括号，也不指定参数。

另外，还可以直接将一个 JavaScript 方法赋给 Onreadystatechange 属性，具体方法如下：

```
Xmlhttp.onreadystatechange = function(){
  //其中的执行代码
}
```

21.8.3　发送HTTP请求

在指定完 Onreadystatechange 事件的处理函数之后，就可以调用 XMLHttpRequest 对象中的 open()和 send()方法来向服务器发送一个 HTTP 请求。具体方法如下：

```
Xmlhttp.open(‘get’,’http://www.example.com/check.jsp’,true);
Xmlhttp.send(null);
```

其中 open()方法中的第一个参数用来定义 HTTP 请求类型，可以为 get、post 或者 head，前两种是比较常见的使用方法。第二个参数是请求服务器的 URL 地址。基于安全的考虑，这个 URL 只能是同一

个网域的，否则会报“没有权限”的错误。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程序，如果设置为 true，这不会继续执行，直到服务器返回信息。默认设置为 true。

Open 方法执行完之后，紧接着需要执行 send 方法。Send 方法是正式向服务器端发送请求的，它只有一个参数，该参数指定需要传给服务器的数据（前面已经对 send()方法作了重点介绍）。如果跟 form 表单一样传递数据文件时，这时候需要调用 setRequestHeader()方法修改 MIME 类别。具体方法如下：

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

21.8.4　处理服务器返回信息

处理服务器返回信息的代码写在 Onreadystatechange 事件指定的函数体中。在处理之前一般需要通过前面介绍的 readyState 属性以及 status 属性来判断响应状态。其中 readyState 值为 4 时，表示服务器已经返回所有信息，可以开始处理返回数据，语法如下：

```
If(xmlhttp.readyState == 4){
    //信息已经返回，可以正式处理
}else{
    //信息没有返回，等待
}
```

除了需要判断服务器端处理状态之后，还需要使用 status 属性来判断返回的 HTTP 状态码，从而确认返回的页面或者数据没有错误。所有的状态码都可以在 W3C 的官方网站上查到。其中使用得比较多的就是 200 属性值，它表示返回信息正常。基本判断方法如下：

```
If(xmlhttp.status == 200){
    //返回数据正常，可以进行处理
}else{
    //数据有问题
}
```

使用 XMLHttpRequest 对象返回处理信息有两种方法：

- responseText：将传回的信息当成简单的字符串处理，一般需要返回 txt 文件、Json 数据或者 html 页面文件时使用。如果返回的是 Json 数据，则需要使用 eval()方法对返回的数据进行处理一下。
- ReponseXML：将返回的信息作 XML 文档进行处理，这时需要使用 DOM 进行解析。

21.8.5　总体开发框架

结合以上所有的步骤，可以总结出一个开发 AJAX 程序的总体框架，可以供后期调用。具体程序代码如下：

```
<script language="javascript" type="text/javascript">
<!—
Function doAjax(url){
  var xmlhttp ;
  try{
    xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
  }catch(e){
    try{
        xmlhttp= new ActiveXObject("Microsoft.XMLHTTP");
      }catch(e){
        try{
```

```
                xmlhttp = new XMLHttpRequest();
                if(xmlhttp.overrideMimeType){
                   xmlhttp.overrideMimeType("text/xml");
                }
             }catch(e){}
        }
   }
   If(!xmlhttp){
      Window.alert("不能创建 XMLHttpRequest 对象实例");
      Return false;
   }
   Xmlhttp.onreadystatechange = processRequest;
   Xmlhttp.open("get",url,true);
   Xmlhttp.send(null);
}

Function processRequest(){
   If(xmlhttp.readyState == 4){
      If(xmlhttp.status == 200){
         Alert(xmlhttp.responseText);
      }else{
         Alert("请求处理返回的数据有异常");
      }
   }else{
       Alert("处理数据未返回");
   }
}
-->
</script>
```

21.9　AJAX主要应用的范围

AJAX 虽然可以实现无刷新页面功能，但是它也不是无所不能，主要使用在数据交互较多、频繁读取数据、数据分类良好的 Web 应用中。总结起来，可以对如下的内容使用 AJAX 来改进你的 Web 效果。

21.9.1　数据验证

大部分 Web 网站中都需要使用到数据验证的功能，最为突出的就是注册的用户名是否重复的验证。如果提交整个页面来获取用户名是否重复这个信息，这付出的代码将是非常昂贵的。传统的 Web 应用就是打开一个校验窗口，或者又 form 提交到服务器端，再由服务器判断后返回校验信息。前者，一般需要使用 window.open 方法来打开一个新窗口，这比较耗费资源；后台，则需要将整个页面提交到服务器端并由服务器端判断校验，这个过程将非常耗时以及加重服务器端的负担。

而使用了 AJAX，则完全可以使用 XMLHttpRequest 对象来进行异步处理，整个过程是在用户完全不察觉的情况下进行的，不再需要弹出新的窗口，也不需要将整个页面提交到服务器。从而提高了响应速度也减轻服务器负担，用户体验也得到了大大提升。

21.9.2　数据验证实例

首先创建一段进行表单提交的 html 页面 submit.jsp，详细代码如下：

```
<form name=”form1” action=”” method=”post”>
用户名：<input type=”text” name=”username” value=””> <input type=”button” name=”check” value=”惟一性检查” onclick=”dataCheck();”> <input type=”submit” name=”submit” value=”提交”>
</form>
```

编写进行用户名惟一性检验的 js 代码，如下：

```
Function dataCheck(){
   var f = document.form1;
   var username = f.username.value;
   if(username == “”){
      alert(“用户名为空”);
      f.username.focus();
      return false;
   }else{
      doAjax(“check.jsp?username=”+username);
   }

}
```

代码说明：其中进行异步请求操作的 doAjax()方法已经在前面创建，这里直接引用。

下面再创建后台进行惟一性检验的 check.jsp 页面文件，详细代码如下：

```
<%@ page contentType=”text/html;charset=gb2312” %>
<%
   String username = request.getParameter(“username”);
   If(“educhina”.equals(username)) out.print(“用户名已经被注册”);
   Else    out.print(“用户名尚未存在”);
%>
```

运行程序可以直接将后台打印的信息返回到客户端，然后再通过 doAjax()方法中的 alert 输出后台返回的信息。

21.9.3　按需获取数据

针对网站上经常使用的分类导航栏，以前的站点，在点击了一个分类链接之后，需要提交和刷新整个页面，不管你获取的信息有多大，这同样加大了服务器的负担以及降低了用户的体验。后来使用了框架来实现页面导航效果，左侧框架用来装载显示分类导航栏的页面，右侧框架使用的页面显示各个分类中的具体信息。这样的方式需要部分解决以往的不足，但是每次请求还是伴随着大量的冗余数据。

使用 AJAX 就可以完全实现按需获取每个分类中的信息，然后将服务器端返回的数据动态地插入到页面适当的位置。并且这一切过程，AJAX 都是异步进行的，即用户再也不会看到页面出现白屏现象。

另外一个按需获取数据的实例就是菜单功能。传统的 Web 应用是一次性将级联菜单的所有数据全部读取出来并写入到数组中，后来程序实现了根据用户的操作再使用 JavaScript 来控制它的子集项目的显示，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者进行一小部分操作，这是也会出现很大的数据冗余现象。

使用 AJAX 同样可以实现菜单的按需获取数据。初始化页面时，只获取第一级显示菜单，再用户操作其中一级菜单时，会通过 AJAX 异步向后台发出获取当前一级菜单中二级菜单的数据。类似地，如果

用户再操作二级菜单，AJAX 会向后台请求获取当前二级菜单中三级菜单的数据。这样 AJAX 实现了页面用什么取什么，用多少取多少的，不会再出现数据的冗余和浪费现象，减少了数据传输量，也减少了用户等待时间（因为每次都更新很小一部分数据，所以速度上得到了提高）。

21.9.4　按需获取数据实例

首先创建 html 页面文件，在<body></body>体中间的 html 代码如下：

```
<table width=”200” border=”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>
```

接下来创建名为 showRoles(obj)的 js 函数，详细代码如下：

```
Function showRoles(obj){
   Document.getElementById(obj).parentNode.style.display=”block’;
   Document.getElementById(obj).innerHTML = “正在读取数据…”;
   currentPos = obj;
   doAjax(“getData.jsp?playPos=”+obj);
}
```

修改异步请求中所使用到的处理函数 processRequest()，修改之后的代码如下：

```
Function processRequest(){
   If(xmlhttp.readyState == 4){
      If(xmlhttp.status == 200){
         Document.getElementById(currentPos).innerHTML = xmlhttp.responseText;
      }else{
         Alert(“请求处理返回的数据有异常”);
      }
   }else{
       Alert(“处理数据未返回”);
   }
}
```

最后就是创建后台 getData.jsp 页面文件，详细代码如下：

```
<%@ page contentType=”text/html;charset=gb2312”%>
<%
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.print(“  总工程师<br>  软件工程师”);
%>
```

运行一下看看效果。当然读者可以从数据库中获取二级菜单的数据。

21.9.3　读取外部数据

AJAX 还可以调用外部数据，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。其中的 XML 文档数据可以直接从 XML 文档中获取，也可以获取 C、C++或者 Java 等语言生成的具有 XML 结构的数据。

当然 AJAX 并不仅仅只在这三方面进行运用，它的运用很广泛，笔者也相信，AJAX 将会称为以后 Web 开发中的主流。

21.10　AJAX综合实例

这一小节将通过一个综合实例来演示 Ajax 的执行过程以及开发流程。该实例使用 Servlet 作为后台处理程序，这里读者正好可以温习一下上一章所介绍的 Servlet 有关知识。

该实例使用 Ajax 实现导航栏功能，开发步骤如下：

21.10.1　创建前台daohang.html文件

首先创建前台显示页面，其中包含大量 JavaScript 代码，用于进行异步请求以及将后台返回的数据插入到页面适当位置。该页面文件的详细代码如下：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3c.org/TR/1999/REC-html401-19991224/loose.dtd">
<HTML xmlns="http://www.w3.org/1999/xhtml">
<HEAD>
<TITLE>导航栏</TITLE>
<META http-equiv=Content-Type content="text/html; charset=gb2312">
<link rel="stylesheet" href="unFocus History Keeper/oreillymail.css" type="text/css">
<BODY>
<H1>Ajax 实现导航栏</H1>
<DIV id="content">
<table width="90%" border="0" align="left">
  <tr>
    <td width="30%">
       <div id="tags"></div>
     </td>
    <td>
       <div id="content"></div>
     </td>
  </tr>
</table>
```

```
<SCRIPT type=text/javascript>
<!--
function getTags(){
var url = "getTags";
var xmlhttp ;
        try{
           xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
        }catch(e){
              try{
                 xmlhttp= new ActiveXObject("Microsoft.XMLHTTP");
              }catch(e){
                    try{
                       xmlhttp = new XMLHttpRequest();
                       if(xmlhttp.overrideMimeType){
                           xmlhttp.overrideMimeType(“text/xml”);
                       }
                    }catch(e){}
              }
        }
if(!xmlhttp){
   window.alert("不能创建 XMLHttpRequest 对象实例");
   return false;
}
xmlhttp.onreadystatechange = processRequest;
xmlhttp.open("GET",url,true);
xmlhttp.send(null);
}

function processRequest(xmlhttp){
var tagsDiv = document.getElementByTagId("tags");
tagsDiv.innerHTML = '';
If(xmlhttp.readyState == 4){
   If(xmlhttp.status == 200){
        var jsonExpression = "(" + xmlhttp.responseText + ")";
        var tagitems= eval(jsonExpression);
       for(var i=0;i<tagitems.tags.length;i++){
           var tag = tagItems.tags[i];
           var tagDiv = document.createElement("div");
           tagDiv.innerHTML = tag.tagName;
           tagsDiv.appendChild(tagDiv);
           tagDiv.onclick=function(){
                getContent(tag.tagId);
           }
        }
   }else{
      alert("请求处理返回的数据有异常");
   }
}else{
     alert("处理数据未返回");
}
```

```
}

function getContent(tagId){
var url = "getContent?tagId="+tagId;
var xmlhttp ;
        try{
           xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
        }catch(e){
              try{
                 xmlhttp= new ActiveXObject("Microsoft.XMLHTTP");
              }catch(e){
                    try{
                       xmlhttp = new XMLHttpRequest();
                       if(xmlhttp.overrideMimeType){
                           xmlhttp.overrideMimeType(“text/xml”);
                       }
                   }catch(e){}
              }
        }
if(!xmlhttp){
   window.alert("不能创建 XMLHttpRequest 对象实例");
   return false;
}
xmlhttp.onreadystatechange = function(xmlhttp){
        var content = '';
        var contentDiv = document.getElementByTagId("content");
        contentDiv.innerHTML = '';
        If(xmlhttp.readyState == 4){
        If(xmlhttp.status == 200){
            content = xmlhttp.responseText;
            contentDiv.innerHTML = content;
   }else{
      alert("请求处理返回的数据有异常");
   }
}else{
     alert("处理数据未返回");
}
};
xmlhttp.open("GET",url,true);
xmlhttp.send(null);
}
-->
</SCRIPT>
</DIV>
</body>
</html>
```

代码说明：

（1）getTags()方法用来从后台服务器端获取所有的分类信息（返回的数据为 JSON 格式），并将返回数据进行处理。

（2）getContent()方法返回某一个分类下的所有信息，并显示在页面的右侧。该方法获取的后台数据为文本格式，不需要任何的数据解析。

21.10.2　创建tags、contents数据库表

接下来创建用于存放所有分类的 tags 数据库表，该表创建在本书第十二章创建的 mydb 数据库中，操作数据库的类 DBConnect.java 在第十四章已经创建。

21.10.2.1　创建 tags 数据库表

```
DROP TABLE IF EXISTS `mydb`.`tags`;
CREATE TABLE   `mydb`.`tags` (
  `tagid` int(10) unsigned NOT NULL auto_increment,
  `tagname` varchar(45) NOT NULL default '',
  PRIMARY KEY   (`tagid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
Insert into tags value(1,'JavaScript');
Insert into tags value(2,'XML');
Insert into tags value(3,'DOM');
Insert into tags value(4,'Ajax');
Insert into tags value(5,'JSON');
```

21.10.2.2　创建 contents 数据库表

该数据库表存放分类下的具体信息，创建数据库表的 SQL 代码如下：

```
DROP TABLE IF EXISTS `mydb`.`contents`;
CREATE TABLE   `mydb`.`contents` (
  `contentid` int(10) unsigned NOT NULL auto_increment,
  `title` varchar(45) NOT NULL default '',
  `description` varchar(1024) NOT NULL default '',
  `tagid` int(10) unsigned NOT NULL default '0',
  PRIMARY KEY   (`contentid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

说明：具体分类下的信息请读者自己加入。

21.10.3　创建处理类getTags.java和getContent.java

getTags.java 类从数据库中获取所有的分类信息，并组装成 JSON 数据格式返回到前台。getContent.java 类用于返回某一分类下的所有信息，直接一文本字符串格式返回，在前台不需要进行任何数据解析。

21.10.3.1　创建 getTags.java 类

```
package cn.ajax;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import cn.ajax.db.DBConnect;
public class getTags extends HttpServlet{
```

```
    private static final long serialVersionUID = 1L;
    public void doPost(HttpServletRequest request,HttpServletResponse response) throws ServletException,
java.io.IOException{
        PrintWriter out = response.getWriter();
        StringBuffer buf = new StringBuffer();
        String str = "select * from tags";
        try{
            DBConnect dbconnect = new DBConnect();
            dbconnect.excuteQuery(str);
            buf.append("{ tags:[");
            if(dbconnect.next()){

    buf.append("{tagId:'"+dbconnect.getInt("tagid")+"',tagName:'"+dbconnect.getString("tagname")+"'},");
            }
            buf.append("{tagId:'000',tagName:'000'}]}");
        }catch(Exception e)
        {
            e.printStackTrace();
        }
       out.print(buf.toString());
    }
    public void doGet(HttpServletRequest request,HttpServletResponse response) throws ServletException,
java.io.IOException{
        doPost(request,response);
    }
}
```

21.10.3.2 创建 getContent.java 类

```
package cn.ajax;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import cn.ajax.db.DBConnect;

public class getContent extends HttpServlet{
    private static final long serialVersionUID = 1L;
    public void doPost(HttpServletRequest request,HttpServletResponse response) throws ServletException,
java.io.IOException{
        PrintWriter out = response.getWriter();
        StringBuffer buf = new StringBuffer();
        String tagId = request.getParameter("tagId");
        String str = "select * from contents where tagId="+tagId;
        try{
            DBConnect dbconnect = new DBConnect();
            dbconnect.excuteQuery(str);
            buf.append("");
            if(dbconnect.next()){
```

```
        buf.append("<div>title:"+dbconnect.getString("title")+"description:"+dbconnect.getString("description")+"</div><br/>");
                    }
                }catch(Exception e)
                {
                    e.printStackTrace();
                }
            out.print(buf.toString());
        }
        public void doGet(HttpServletRequest request,HttpServletResponse response) throws ServletException, java.io.IOException{
            doPost(request,response);
        }
}
```

21.10.4　在web.xml中声明Servlet类

要让 Web 服务器能够识别到以上创建的 Servlet，还需要在 web.xml 中对以上创建的两个 Servlet 类文件进行声明：

```
<servlet>
  <servlet-name>getTags</servlet-name>
  <servlet-class>cn.ajax.getTags</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>getTags</servlet-name>
  <servlet-pattern>/getTags</servlet-pattern>
</servlet-mapping>
<servlet>
  <servlet-name>getContent</servlet-name>
  <servlet-class>cn.ajax.getContent</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>getContent</servlet-name>
  <servlet-pattern>/getContent</servlet-pattern>
</servlet-mapping>
```

至此，就已经完成了本实例的创建。运行 daohang.html 页面即可看到相应效果。

21.11　AJAX存在的问题

AJAX 虽然带来诸多优点，但本来就是鱼和熊掌不可兼得，它也存在着很多的缺陷和不足。本小节将对 AJAX 存在的问题进行一定的总结。

21.11.1　用户对浏览AJAX网站的方式不习惯

用户已经习惯了传统的页面提交方式，即填写表单、提交、刷新页面、等待处理结果这一过程。即使不是专业的 Web 设计人员，普通用户也大多明白页面的概念，一个页面加上不同的参数就可以定位

到不同的信息，一般 Web 程序还是充当信息发布的手段。

但是 Ajax 确改变了这一切。一个 Web 页面不再单纯的表示同一类信息。用户单击某一个链接或者按钮也不会出现刷新效果，而是在页面的某一局部位置更新数据，并没有明显的提示。Ajax 带来的效果，使得用户根本就不会知道浏览器已经做了什么，毕竟这和传统的 Web 页面区别太大。

要使得用户完全适应这样的浏览方式，还需要循序渐进地进行 Ajax 开发，周全考虑用户的习惯，并时刻提醒用户页面正在做什么或者下一步应该做什么。

21.11.2　对搜索引擎的不友好

一般来讲，Web 信息的发布商都希望自己建立的网站信息能够被更多的人访问到，为更多的人服务。在这样的情形下，搜索引擎起到了很打的作用，人们已经习惯使用搜索引擎来查找自己所需要的信息，但是搜索引擎是基于页面工作的，但是 Ajax 确实基于数据工作，这就让两者无法很好的协调。

Ajax 中进行的通信通常是一个数据片段，例如 xml、html 片段或者纯文本信息，这些信不是完整的 html 文档，而且是通过后台异步获取的，不能够被搜索引擎所发现并缓存起来。最终这会导致用户无法搜索到指定的站点。

21.11.3　前进后退功能实效

传统的 Web 程序是由一个或者多个页面组成，当用户从一个页面跳转到另一个页面上，这时浏览器会自动缓存前一个页面的内容。这样用户可以通过浏览器的后退按钮来直接访问前一个访问的页面。但是 Ajax 在前进、后退功能上确出了麻烦。由于在 Ajax 中，一个页面并不仅仅作为一次数据显示，而且还作为一个用户操作的界面，通过这个界面，用户可以局部获取信息，局部的显示信息。由于所有的操作基本上都是在一个页面上完成，所以浏览器不会缓存从服务器上获取的最新信息，从而浏览器的后退、前进功能就无法应用到用户的操作上。

完全靠浏览器是不可能解决前进、后退功能的，这时需要另找办法。事实上可以发现 Gmail 是可以后退和前进的，它是通过某些手段欺骗了浏览器，并让它可以为 Ajax 程序工作。至于详细的实习方法，读者可以查看相关资料。

21.11.4　刷新定位问题

这个问题通常被称为书签问题或者搜藏夹问题。即当访问一个 Ajax 程序时，如果发现一个感兴趣的页面，于是将其加入到收藏夹中以便日后访问。但是，当下次从收藏夹中访问该页面的时候，却发现它无法像上次那样工作。搜藏夹中的网址只能导航到初次访问该站点的位置，而无法记录下一次所操作的步骤，因而就无法定位到用户本来想收藏的信息。

这样的问题和前进后退一样，不能仅仅通过浏览器来解决，它需要使用某种手段记录下用户操作，为了能够加入到收藏夹，惟一的方法就是将操作步骤的信息加入到 url 中。事实上，很多 Ajax 网站就是通过这种方法来解决刷新问题的。

21.11.5　性能问题

Ajax 的使用，使得大量的运算从服务器端转移到了客户端，这就意味着浏览器将承受更大的负担，

不再是简单的文档显示。Ajax 中的核心语言是 JavaScript，作为一门解释型的脚本语言，其运行效率并不是很高，而且还依赖于不同浏览器的实现。

在 Ajax 客户端的一个页面就是一个程序界面，为了承担这个界面的运行，需要有大量的 javascript 代码为其工作，需要不断利用 Dom 操作文档的结点对象或者用 css 来控制外观。这样使得浏览器要占用很大的内存，并然会容易导致运行速度很慢，如果不及时释放资源将会使浏览器陷入无法响应状态。

要妥善解决这些问题必须在效率和功能上找到平衡点，并可能多的实现按需下载，即使 JavaScript 代码也要如此。

21.11.6　开发难度大

Ajax 不仅仅涉及到大量的客户端技术，还涉及到大量的服务器端技术，一个 Web 开发人员需要了解多项技术。这就使得开发过程变得更加复杂。尽管 Ajax 这么多的问题，但是它的魅力还足以让 Web 开发者无法忽视它的存在。

21.12　本章小结

Ajax 作为 Web2.0 新特性，它存在诸多优点，可以提高用户体验。用户不再需要在页面刷新中苦苦等待数据的显示。Ajax 实现了按需获取数据，并可以动态地修改页面局部数据，这大大降低了网络传输负荷。本章就重点对 Ajax 作了介绍，并具体讲解了 Ajax 的开发过程中所涉及的技术。

读完本章之后，读者应该能够进行基本的 Ajax 开发。有关其他更深一层的内容，请读者自己查看相关资料，例如前进、后退以及刷新功能的实现等。