**第33章 Ajax**

学习要点：

1. XMLHttpRequest
2. GET与POST
3. 封装Ajax

2005年Jesse James Garrett发表了一篇文章，标题为：“Ajax：A new Approach to Web Applications”。他在这篇文章里介绍了一种技术，用他的话说，就叫：Ajax，是Asynchronous JavaScript + XML的简写。这种技术能够想服务器请求额外的数据而无须卸载页面(即刷新)，会带来更好的用户体验。一时间，席卷全球。

1. **XMLHttpRequest**

Ajax技术核心是XMLHttpRequest对象(简称XHR)，这是由微软首先引入的一个特性，其他浏览器提供商后来都提供了相同的实现。在XHR出现之前，Ajax式的通信必须借助一些hack手段来实现，大多数是使用隐藏的框架或内嵌框架。

XHR的出现，提供了向服务器发送请求和解析服务器响应提供了流畅的接口。能够以异步方式从服务器获取更多的信息，这就意味着，用户只要触发某一事件，在不刷新网页的情况下，更新服务器最新的数据。

虽然Ajax中的x代表的是XML，但Ajax通信和数据格式无关，也就是说这种技术不一定使用XML。

IE7+、Firefox、Opera、Chrome和Safari都支持原生的XHR对象，在这些浏览器中创建XHR对象可以直接实例化XMLHttpRequest即可。

var xhr = new XMLHttpRequest();

alert(xhr); //XMLHttpRequest

如果是IE6及以下，那么我们必须还需要使用ActiveX对象通过MSXML库来实现。在低版本IE浏览器可能会遇到三种不同版本的XHR对象，即MSXML2.XMLHttp、MSXML2.XMLHttp.3.0、MSXML2.XMLHttp.6.0。我们可以编写一个函数。

function createXHR() {

if (typeof XMLHttpRequest != 'undefined') {

return new XMLHttpRequest();

} else if (typeof ActiveXObject != 'undefined') {

var versions = [

'MSXML2.XMLHttp.6.0',

'MSXML2.XMLHttp.3.0',

'MSXML2.XMLHttp'

];

for (var i = 0; i < versions.length; i ++) {

try {

return new ActiveXObject(version[i]);

} catch (e) {

//跳过

}

}

} else {

throw new Error('您的浏览器不支持XHR对象！');

}

}

var xhr = new createXHR();

在使用XHR对象时，先必须调用open()方法，它接受三个参数：要发送的请求类型(get、post)、请求的URL和表示是否异步。

xhr.open('get', 'demo.php', false); //对于demo.php的get请求，false同步

PS：demo.php的代码如下：

<?php echo Date('Y-m-d H:i:s')?> //一个时间

open()方法并不会真正发送请求，而只是启动一个请求以备发送。通过send()方法进行发送请求，send()方法接受一个参数，作为请求主体发送的数据。如果不需要则，必须填null。执行send()方法之后，请求就会发送到服务器上。

xhr.send(null); //发送请求

当请求发送到服务器端，收到响应后，响应的数据会自动填充XHR对象的属性。那么一共有四个属性：

|  |  |
| --- | --- |
| 属性名 | 说明 |
| responseText | 作为响应主体被返回的文本 |
| responseXML | 如果响应主体内容类型是"text/xml"或"application/xml"，则返回包含响应数据的XML DOM文档 |
| status | 响应的HTTP状态 |
| statusText | HTTP状态的说明 |

接受响应之后，第一步检查status属性，以确定响应已经成功返回。一般而已HTTP状态代码为200作为成功的标志。除了成功的状态代码，还有一些别的：

|  |  |  |
| --- | --- | --- |
| HTTP状态码 | 状态字符串 | 说明 |
| 200 | OK | 服务器成功返回了页面 |
| 400 | Bad Request | 语法错误导致服务器不识别 |
| 401 | Unauthorized | 请求需要用户认证 |
| 404 | Not found | 指定的URL在服务器上找不到 |
| 500 | Internal Server Error | 服务器遇到意外错误，无法完成请求 |
| 503 | ServiceUnavailable | 由于服务器过载或维护导致无法完成请求 |

我们判断HTTP状态值即可，不建议使用HTTP状态说明，因为在跨浏览器的时候，可能会不太一致。

addEvent(document, 'click', function () {

var xhr = new createXHR();

xhr.open('get', 'demo.php?rand=' + Math.random(), false); //设置了同步

xhr.send(null);

if (xhr.status == 200) { //如果返回成功了

alert(xhr.responseText); //调出服务器返回的数据

} else {

alert('数据返回失败！状态代码：' + xhr.status + '状态信息：' + xhr.statusText);

}

});

以上的代码每次点击页面的时候，返回的时间都是时时的，不同的，说明都是通过服务器及时加载回的数据。那么我们也可以测试一下在非Ajax情况下的情况，创建一个demo2.php文件，使用非Ajax。

<script type="text/javascript" src="base.js"></script>

<script type="text/javascript">

addEvent(document, 'click', function () {

alert("<?php echo Date('Y-m-d H:i:s')?>");

});

</script>

同步调用固然简单，但使用异步调用才是我们真正常用的手段。使用异步调用的时候，需要触发readystatechange事件，然后检测readyState属性即可。这个属性有五个值：

|  |  |  |
| --- | --- | --- |
| 值 | 状态 | 说明 |
| 0 | 未初始化 | 尚未调用open()方法 |
| 1 | 启动 | 已经调用open()方法，但尚未调用send()方法 |
| 2 | 发送 | 已经调用send()方法，但尚未接受响应 |
| 3 | 接受 | 已经接受到部分响应数据 |
| 4 | 完成 | 已经接受到全部响应数据，而且可以使用 |

addEvent(document, 'click', function () {

var xhr = new createXHR();

xhr.onreadystatechange = function () {

if (xhr.readyState == 4) {

if (xhr.status == 200) {

alert(xhr.responseText);

} else {

alert('数据返回失败！状态代码：' + xhr.status + '状态信息：'

+ xhr.statusText);

}

}

};

xhr.open('get', 'demo.php?rand=' + Math.random(), true);

xhr.send(null);

});

PS：使用abort()方法可以取消异步请求，放在send()方法之前会报错。放在responseText之前会得到一个空值。

1. **GET与POST**

在提供服务器请求的过程中，有两种方式，分别是：GET和POST。在Ajax使用的过程中，GET的使用频率要比POST高。

在了解这两种请求方式前，我们先了解一下HTTP头部信息，包含服务器返回的响应头信息和客户端发送出去的请求头信息。我们可以获取响应头信息或者设置请求头信息。我们可以在Firefox浏览器的firebug查看这些信息。

//使用getResponseHeader()获取单个响应头信息

alert(xhr.getResponseHeader('Content-Type'));

//使用getAllResponseHeaders()获取整个响应头信息

alert(xhr.getAllResponseHeaders());

//使用setRequestHeader()设置单个请求头信息

xhr.setRequestHeader('MyHeader', 'Lee'); //放在open方法之后，send方法之前

PS：我们只可以获取服务器返回回来响应头信息，无法获取向服务器提交的请求头信息，自然自定义的请求头，在JavaScript端是无法获取到的。

GET请求

GET请求是最常见的请求类型，最常用于向服务器查询某些信息。必要时，可以将查询字符串参数追加到URL的末尾，以便提交给服务器。

xhr.open('get', 'demo.php?rand=' + Math.random() + '&name=Koo', true);

通过URL后的问号给服务器传递键值对数据，服务器接收到返回响应数据。特殊字符传参产生的问题可以使用encodeURIComponent()进行编码处理，中文字符的返回及传参，可以讲页面保存和设置为utf-8格式即可。

//一个通用的URL提交函数

function addURLParam(url, name, value) {

url += (url.indexOf('?') == -1 ? '?' : '&'); //判断的url是否有已有参数

url += encodeURIComponent(name) + '=' + encodeURIComponent(value);

alert(url);

return url;

}

PS：当没有encodeURIComponent()方法时，在一些特殊字符比如“&”，会出现错误导致无法获取。

POST请求

POST请求可以包含非常多的数据，我们在使用表单提交的时候，很多就是使用的POST传输方式。

xhr.open('post', 'demo.php', true);

而发送POST请求的数据，不会跟在URL的尾巴上，而是通过send()方法向服务器提交数据。

xhr.send('name=Lee&age=100');

一般来说，向服务器发送POST请求由于解析机制的原因，需要进行特别的处理。因为POST请求和Web表单提交是不同的，需要使用XHR来模仿表单提交。

xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');

PS：从性能上来讲POST请求比GET请求消耗更多一些，用相同数据比较，GET最多比POST快两倍。

上一节课的JSON也可以使用Ajax来回调访问。

var url = 'demo.json?rand=' + Math.random();

var box = JSON.parse(xhr.responseText);

1. **封装Ajax**

因为Ajax使用起来比较麻烦，主要就是参数问题，比如到底使用GET还是POST；到底是使用同步还是异步等等，我们需要封装一个Ajax函数，来方便我们调用。

function ajax(obj) {

var xhr = new createXHR();

obj.url = obj.url + '?rand=' + Math.random();

obj.data = params(obj.data);

if (obj.method === 'get') obj.url = obj.url.indexOf('?') == -1 ?

obj.url + '?' + obj.data : obj.url + '&' + obj.data;

if (obj.async === true) {

xhr.onreadystatechange = function () {

if (xhr.readyState == 4) callback();

};

}

xhr.open(obj.method, obj.url, obj.async);

if (obj.method === 'post') {

xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');

xhr.send(obj.data);

} else {

xhr.send(null);

}

if (obj.async === false) {

callback();

}

function callback () {

if (xhr.status == 200) {

obj.success(xhr.responseText); //回调

} else {

alert('数据返回失败！状态代码：' + xhr.status + '，

状态信息：' + xhr.statusText);

}

}

}

//调用ajax

addEvent(document, 'click', function () { //IE6需要重写addEvent

ajax({

method : 'get',

url : 'demo.php',

data : {

'name' : 'Lee',

'age' : 100

},

success : function (text) {

alert(text);

},

async : true

});

});

//名值对编码

function params(data) {

var arr = [];

for (var i in data) {

arr.push(encodeURIComponent(i) + '=' + encodeURIComponent(data[i]));

}

return arr.join('&');

}

PS：封装Ajax并不是一开始就形成以上的形态，需要经过多次变化而成