**Ajax**

# 1、ajax技术的背景

不可否认，ajax技术的流行得益于google的大力推广，正是由于google earth、google suggest以及gmail等对ajax技术的广泛应用，催生了ajax的流行。而这也让微软感到无比的尴尬，因为早在97年，微软便已经发明了ajax中的关键技术，并且在99年IE5推出之时，它便开始支持XmlHttpRequest对象，并且微软之前已经开始在它的一些产品中应用ajax，比如说MSDN网站菜单中的一些应用。遗憾的是，不知道出于什么想法，当时微软发明了ajax的核心技术之后，并没有看到它的潜力而加以发展和推广，而是将它搁置起来。对于这一点来说，我个人是觉得非常奇怪的，因为以微软的资源和它的战略眼光来说，应该不会看不到ajax技术的前景，唯一的解释也许就是因为当时它的主要竞争对手Netscape的消失反而使它变得麻痹和迟钝，毕竟巨人也有打盹的时候，比如IBM曾经在对微软战略上的失误。正是这一次的失误，成就了它现在的竞争对手google在ajax方面的领先地位，而事实上google目前在ajax技术方面的领先是微软所无法达到的，这一点在后面我讲述ajax缺陷的时候也会提到。现在微软也意识到了这个问题，因此它也开始在ajax领域奋起直追，比如说推出它自己的ajax框架atlas，并且在.NET2.0也提供了一个用来实现异步回调的接口，即ICallBack接口。那么微软为什么对自己在ajax方面的落后如此紧张呢？现在就让我们来分析一下ajax技术后面隐藏的深刻意义。

# 2、ajax技术的意义

我们在平时的开发中都多多少少的接触或者应用到了ajax，谈到ajax技术的意义，我们关注得最多的毫无疑问是提升用户的体验。但是，如果我们结合将来电脑和互联网的发展趋势，我们会发现ajax技术在某些方面正好代表了这种趋势。为什么这样说呢？我们知道，自从电脑出现以来，一直是桌面软件占据着绝对主导的地位，但是互联网的出现和成功使这一切开始发生着微妙的变化。相当一部分的人都相信，迟早有一天，数据和电脑软件将会从桌面转移到互联网。也就是说，将来的电脑有可能抛弃笨重的硬盘，而直接从互联网来获取数据和服务，我记得我念大学的时候，有位教授给我们上课的时候，曾经设想过这样一种情景，也许在将来的电脑桌面上，没有任何多余的软件和程序， 而仅仅只有一个IE，虽然现在看起来我们距离这一天还很遥远，并且这其中还有很多的问题需要解决，但是我觉得这个并非梦想，而是迟早将实现的现实。那么，这其中的主要问题就是互联网的连接不稳定，谁也不愿意看着自己的电脑从服务器一点一滴的下载数据，那么，ajax是不是解决了这个问题呢，说实话，与其说ajax解决了这个问题，倒不如它只是掩盖了这个问题，它只是在服务器和客户端之间充当了一个缓冲器，让用户误以为服务没有中断。精确的说，ajax并不能提高从服务器端下载数据的速度，而只是使这个等待不那么令人沮丧。但是正是这一点就足以产生巨大的影响和震动，它实际上也对桌面软件产生了巨大的冲击。这一点我用一个例子来说明，我们可以比较一下Outlook Express和Gmail，前者是典型的桌面软件，后者是ajax所实现的B/S模式，实际上后者目前已经在慢慢取代前者了，Gmail在收发邮件的时候已经和Outlook Express的功能几乎没有差别了，而且它不需要安装客户端程序。这就是为什么微软对ajax所带来的冲击有着如此的恐惧心理，并且在它前不久所进行的调查之中，将google看做他们未来十年内的主要竞争对手的主要原因之一。当然，这种变化也并不会将桌面软件全部淘汰，现有的浏览器还没有一个能像PhotoShop等桌面程序那样处理复杂的图像。但是我们也不能忽视它带来的影响和冲击。

# 3、关于ajax的名字

ajax 的全称是Asynchronous JavaScript and XML，其中，Asynchronous是异步的意思，它有别于传统web开发中采用的同步的方式。

# 4、关于同步和异步

异步传输是面向字符的传输，它的单位是字符；而同步传输是面向比特的传输，它的单位是桢，它传输的时候要求接受方和发送方的时钟是保持一致的。

具体来说，异步传输是将比特分成小组来进行传送。一般每个小组是一个8位字符，在每个小组的头部和尾部都有一个开始位和一个停止位，它在传送过程中接收方和发送方的时钟不要求一致，也就是说，发送方可以在任何时刻发送这些小组，而接收方并不知道它什么时候到达。一个最明显的例子就是计算机键盘和主机的通信，按下一个键的同时向主机发送一个8比特位的ASCII代码，键盘可以在任何时刻发送代码，这取决于用户的输入速度，内部的硬件必须能够在任何时刻接收一个键入的字符。这是一个典型的异步传输过程。异步传输存在 一个潜在的问题，即接收方并不知道数据会在什么时候到达。在它检测到数据并做出响应之前，第一个比特已经过去了。这就像有人出乎意料地从后面走上来跟你说话，而你没来得及反应过来，漏掉了最前面的几个词。因此，每次异步传输的信息都以一个起始位开头，它通知接收方数据已经到达了，这就给了接收方响应、接收 和缓存数据比特的时间；在传输结束时，一个停止位表示该次传输信息的终止。按照惯例，空闲（没有传送数据）的线路实际携带着一个代表二进制1的信号。步传输的开始位使信号变成0，其他的比特位使信号随传输的数据信息而变化。最后，停止位使信号重新变回1，该信号一直保持到下一个开始位到达。例如在键盘上数字“1”，按照8比特位的扩展ASCII编码，将发送“00110001”，同时需要在8比特位的前面加一个起始位，后面一个停止位。

同步传输的比特分组要大得多。它不是独立地发送每个字符，每个字符都有自己的开始位和停止位，而是把它们组合起来一起发送。我们将这些组合称为数据帧，或简称为帧。

　　数据帧的第一部分包含一组同步字符，它是一个独特的比特组合，类似于前面提到的起始位，用于通知接收方一个帧已经到达，但它同时还能确保接收方的采样速度和比特的到达速度保持一致，使收发双方进入同步。

　　帧的最后一部分是一个帧结束标记。与同步字符一样，它也是一个独特的比特串，类似于前面提到的停止位，用于表示在下一帧开始之前没有别的即将到达的数据了。

　　同步传输通常要比异步传输快速得多。接收方不必对每个字符进行开始和停止的操作。一旦检测到帧同步字符，它就在接下来的数据到达时接收它们。另外，同步传输的开销也比较少。例如，一个典型的帧可能有500字节（即4000比特）的数据，其中可能只包含100比特的开销。这时，增加的比特位使传输的比特总数增加2.5%，这与异步传输中25 %的增值要小得多。随着数据帧中实际数据比特位的增加，开销比特所占的百分比将相应地减少。但是，数据比特位越长，缓存数据所需要的缓冲区也越大，这就限制了一个帧的大小。另外，帧越大，它占据传输媒体的连续时间也越长。在极端的情况下，这将导致其他用户等得太久。

了解了同步和异步的概念之后，大家应该对ajax为什么可以提升用户体验应该比较清晰了，它是利用异步请求方式的。打个比方，如果现在你家里所在的小区因某种情况而面临停水，现在有关部门公布了两种方案，一是完全停水8个小时，在这8个小时内完全停水，8个小时后恢复正常。二是不完全停水10个小时，在这10个小时内水没有完全断，只是流量比原来小了很多，在10个小时后恢复正常流量，那么，如果是你你会选择哪种方式呢？显然是后者。

# 5、ajax所包含的技术

大家都知道ajax并非一种新的技术，而是几种原有技术的结合体。它由下列技术组合而成。

   1.使用CSS和XHTML来表示。

   2.使用DOM模型来交互和动态显示。

   3.使用**XMLHttpRequest来和服务器进行异步通信**。

   4.使用**javascript来绑定和调用**。

在上面几中技术中，除了XmlHttpRequest对象以外，其它所有的技术都是基于web标准并且已经得到了广泛使用的，XMLHttpRequest虽然目前还没有被W3C所采纳，但是它已经是一个事实的标准，因为目前几乎所有的主流浏览器都支持它。

# 6、ajax原理和XmlHttpRequest对象

## AJAX原理

Ajax的原理简单来说通过XmlHttpRequest对象（现在level 2）来向服务器发异步请求，从服务器获得数据，然后用javascript来操作DOM而更新页面。这其中最关键的一步就是从服务器获得请求数据。要清楚这个过程和原理，我们必须对 XMLHttpRequest有所了解。

　XMLHttpRequest是ajax的核心机制，它是在IE5中首先引入的，是一种支持异步请求的技术。简单的说，也就是javascript可以及时向服务器提出请求和处理响应，而不阻塞用户，达到无刷新的效果。所以我们先从XMLHttpRequest讲起，来看看它的工作原理。

## XMLHttpRequest属性

### (1)timeout属性

XHR level2新加的，设置本次请求的最大时间，超过设置时间则停止请求**。**

### (2)readyState属性

请求的五种状态。

0 (**未初始化**)：对象已建立，但是尚未初始化（尚未调用open方法）

　　　　 1 (**初始化**):对象已建立，尚未调用send方法。

　 2 (**发送数据**)： send方法已调用，但是当前的状态及http头未知。

3 (**数据传送中**)：已接收部分数据，因为响应及http头不全，这时通过responseBody和responseText获取部分数据会出现错误。

4 (**完成**)：数据接收完毕,此时可以通过通过responseXml和responseText获取完整的回应数据。

### (3)status属性

该请求的响应状态码 (例如, 状态码200 表示一个成功的请求).**只读.**

### (4)statusText属性

该请求的响应状态信息,包含一个**状态码**和**原因短语** (例如 "200 OK"). **只读.**

### (5)response属性

XHR level2新加的,响应实体的类型由[responseType](https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequest#responseType)来指定，可以是[ArrayBuffer](https://developer.mozilla.org/zh-cn/JavaScript_typed_arrays/ArrayBuffer)， [Blob](https://developer.mozilla.org/zh-cn/DOM/Blob)， [Document](https://developer.mozilla.org/zh-CN/docs/Web/API/Document)，JavaScript对象(即"json")，或者是字符串。如果请求未完成或失败，则该值为null。

### (6)responseType属性

XHR level2新加的,设置该值能够改变响应类型。就是告诉服务器你期望的响应格式。

可以是[ArrayBuffer](https://developer.mozilla.org/zh-cn/JavaScript_typed_arrays/ArrayBuffer)， [Blob](https://developer.mozilla.org/zh-cn/DOM/Blob)， [Document](https://developer.mozilla.org/zh-CN/docs/Web/API/Document)，JavaScript对象(即"json")，字符串或者是空字符串（默认）。

### (7)responseText属性

请求的响应为**文本**，或是当请求未成功或还未发送时为null。**只读。**

### (8)responseXML属性

当responseType设置为"document"，并且请求为异步的，则响应会被当做text/html流来解析。**只读.**

### (9)withCredentials属性

表明在进行跨站(cross-site)的访问控制(Access-Control)请求时，是否使用认证信息(例如cookie或授权的header)。 默认为 false。

**注意:** 这不会影响同站(same-site)请求.

## XMLHttpRequest方法

### (1)open()

初始化一个请求。

open(method,url,async,username,password)

method: 请求所使用的HTTP方法; 例如 "GET", "POST", "PUT", "DELETE"等. 如果下个参数是非HTTP(S)的URL,则忽略该参数.

url:该请求要访问的url。

async：一个可选的布尔值参数，默认为true,意味着是否执行异步操作，如果值为false,则send()方法不会返回任何东西，直到接受到了服务器的返回数据。如果为值为true，一个对开发者透明的通知会发送到相关的事件监听者。

username：用户名,可选参数,为授权使用;默认参数为空string.

password：密码,可选参数,为授权使用;默认参数为空string.

### (2)setRequestHeader ()

给指定的HTTP请求头赋值.在这之前,你必须确认已经调用[open()](https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequest#open)方法打开了一个url。

**xnr.setRequestHeader(‘Content-Type’,’text/html,charaset=utf-8’)**

### (3)send ()

发送请求. 如果该请求是异步模式(默认),该方法会立刻返回. 相反,如果请求是同步模式,则直到请求的响应完全接受以后,该方法才会返回。

send()方法，当method为GET时，没有参数。当method为POST时，参数为data，data的类型可以为ArrayBuffer、Blob、Document、FormData和DOMString。

**补充：**

**ArrayBuffer对象:**

表示**二进制数据的原始缓冲区**，该缓冲区用于存储各种类型化数组的数据。 无法直接读取或写入 ArrayBuffer，但可根据需要将其传递到类型化数组或 DataView 对象来解释原始缓冲区。

他是一个二进制数据的原始缓冲区，虽然 JavaScript 是弱类型语言，但是他本身是对数据的类型和大小都有限制的，我们需要通过某种数据结构将缓冲区的内容有序的读取出来（写进去）。

[**Blob**](https://developer.mozilla.org/zh-cn/DOM/Blob)**对象:**

一个Blob对象就是一个包含有**只读原始数据的类文件对象**。Blob对象中的数据并不一定得是JavaScript中的原生形式。File接口基于Blob，继承了Blob的功能,并且扩展支持了用户计算机上的本地文件。

创建Blob对象的方法有几种，可以调用Bolb构造函数，还可以使用一个已有Blob对象上的slice()方法切出另一个Blob对象，还可以调用Cancas对象上的toBlob方法。

**FormData对象：**

ajax 操作往往用来传递表单数据。为了方便表单处理，HTML 5 新增了一个 FormData 对象，可以**模拟表单**。

　　 首先，新建一个 FormData 对象。

var formData = new FormData ();

　　 然后，为它添加表单项。

formData.append ('username'， '张三');

formData.append ('id'， 123456);

　　 最后，直接传送这个 FormData 对象。这与提交网页表单的效果，完全一样。

xhr.send (formData);

　　 FormData 对象也可以用来获取网页表单的值。

var form = document.getElementById ('myform');

var formData = new FormData (form);

formData.append ('secret'， '123456'); // 添加一个表单项

xhr.open ('POST'， form.action);

xhr.send (formData);

### (4)getAllResponseHeaders ()

返回所有响应头信息(响应头名和值), 如果响应头还没接受,则返回null。

### (5)getResponseHeader ()

返回指定的响应头的值, 如果响应头还没被接受,或该响应头不存在,则返回null。

### (6)abort ()

如果请求已经被发送,则立刻中止请求。

　　但是，由于各浏览器之间存在差异，所以创建一个XMLHttpRequest对象可能需要不同的方法。这个差异主要体现在IE和其它浏览器之间。

## XMLHttpRequest事件

### (1)loadstart事件

XHR level2新加的，当请求**开始时**被触发。

### (2)readystatechange事件

当readyState属性**状态值每次改变时**所触发的事件处理程序。

### (3)load事件

XHR level2新加的，当请求**响应完成后**（即readyState==4）触发的操作。

可以在一定程度上代替onreadystatechange事件使用，但是不能完全代替，要明确onreadystatechange事件是在每次readyState状态值发生变化时都会触发，而onload事件仅在readyState==4时才触发。

### (4)timeout事件

XHR level2新加的，当**超出本次请求设置的时间**后触发该事件。

xhr.timeout=3000;

xhr.ontimeout=function(){alert(‘请求超时’)；}

timeout事件是level2新加的功能，在这之前实现超时的操作原理和步骤为：

function CreateXmlHttp() {  
 //非IE浏览器创建XmlHttpRequest对象  
 if (window.XmlHttpRequest) {  
 xmlhttp = new XmlHttpRequest();  
 }  
 //IE浏览器创建XmlHttpRequest对象  
 if (window.ActiveXObject) {  
 try {  
 xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");  
 }catch (e) {  
 try {  
 xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");  
 }catch (ex) { }  
 }  
 }  
}  
function Ustbwuyi() {  
 var data = document.getElementById("username").value;  
 var xmlhttp = CreateXmlHttp();  
 if (!xmlhttp) {  
 alert("创建xmlhttp对象异常！");  
 return false;  
 }  
 xmlhttp.open("POST", url, false);

**//设置超时时间和超时判断的变量**

**var timeout = 3000;**

**var timeouted = false;**

**//利用setTimeout计时器完成**

**setTimeout(function(){**

**timeouted = true;**

**},timeout);**

xmlhttp.onreadystatechange = function () {  
 if (xmlhttp.readyState == 4 && **!timeouted**) {  
 document.getElementById("user1").innerHTML = "数据正在加载...";  
 if (xmlhttp.status == 200||xmlhttp.status == 304) {  
 document.write(xmlhttp.responseText);  
 }  
 }  
 }  
 xmlhttp.send();  
}

### (5) progress事件

XHR level2新加的,当**请求发送或接收数据期间**，在服务器指定间隔内触发。

xhr.upload.progress = updataProgress;

xhr.progress = updataProgress;

function updataProgress(e){

if(e.lengthComptable){

return e.loaded/e.total\*100;

}

}

### (6)error()

XHR level2新加的，当请求**出错时**被触发。

### (7)abort ()

XHR level2新加的，当请求**被取消时**(即abort（）方法调用时)被触发。

### (8)loadend()

XHR level2新加的，当请求**结束时**被触发。

## AJAX实例

下面是一个比较标准的创建XMLHttpRequest对象的方法。

[复制代码](javascript:void(0);)

function CreateXMLHttp() {  
 //非IE浏览器创建XMLHttpRequest对象  
 if (window.XMLHttpRequest) {  
 xmlhttp = new XMLHttpRequest();  
 }  
 //IE浏览器创建XMLHttpRequest对象  
 if (window.ActiveXObject) {  
 try {  
 xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");  
 }catch (e) {  
 try {  
 xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");  
 }catch (ex) { }  
 }  
 }  
}  
function Ustbwuyi() {  
 var data = document.getElementById("username").value;  
 var xmlhttp = CreateXMLHttp();  
 if (!xmlhttp) {  
 alert("创建xmlhttp对象异常！");  
 return false;  
 }  
 xmlhttp.open("POST", url, false);  
 xmlhttp.onreadystatechange = function () {  
 if (xmlhttp.readyState == 4) {  
 document.getElementById("user1").innerHTML = "数据正在加载...";  
 if (xmlhttp.status == 200||xmlhttp.status == 304) {  
 document.write(xmlhttp.responseText);  
 }  
 }  
 }  
 xmlhttp.send();  
}

[复制代码](javascript:void(0);)

如上所示，函数首先检查XMLHttpRequest的整体状态并且保证它已经完成（readyStatus=4），即数据已经发送完毕。然后根据服务器的设定询问请求状态，如果一切已经就绪（status=200或者status=304），那么就执行下面需要的操作。

对于XmlHttpRequest的两个方法，open和send，其中open方法指定了：

a、向服务器提交数据的类型，即**post还是get**。

b、请求的**url**地址和传递的参数。

  c、传输方式，**false为同步，true为异步**。默认为true。如果是异步通信方式(true)，客户机就不等待服务器的响应；如果是同步方式(false)，客户机就要等到服务器返回消息后才去执行其他操作。我们需要根据实际需要来指定同步方式，在某些页面中，可能会发出多个请求，甚至是有组织有计划有队形大规模的高强度的request，而后一个是会覆盖前一个的，这个时候当然要指定同步方式。

**Send方法用来发送请求。**

　　知道了XMLHttpRequest的工作流程，我们可以看出，**XMLHttpRequest是完全用来向服务器发出一个请求的，它的作用也局限于此**，但它的作用是整个ajax实现的关键，因为ajax无非是两个过程，发出请求和响应请求。并且它完全是一种客户端的技术。而XMLHttpRequest正是处理了服务器端和客户端通信的问题所以才会如此的重要。

现在，我们对ajax的原理大概可以有一个了解了。我们可以把服务器端看成一个数据接口，它返回的是一个纯文本流，当然，这个文本流可以是XML格式，可以是Html，可以是Javascript代码，也可以只是一个字符串。这时候，XMLHttpRequest向服务器端请求这个页面，服务器端将文本的结果写入页面，这和普通的web开发流程是一样的，不同的是，客户端在异步获取这个结果后，不是直接显示在页面，而是先由javascript来处理，然后再显示在页面。至于现在流行的很多ajax控件，比如magicajax等，可以返回DataSet等其它数据类型，只是将这个过程封装了的结果，本质上他们并没有什么太大的区别。

# 7、ajax的优点

   Ajax的给我们带来的好处大家基本上都深有体会，在这里我只简单的讲几点：

    1、最大的一点是**页面无刷新**，在页面内与服务器通信，给用户的体验非常好。

　 2、使用**异步方式与服务器通信**，不需要打断用户的操作，具有更加迅速的响应能力。

　 3、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。并且减轻服务器的负担，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。

  4、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。

# 8、ajax的缺点

下面我着重讲一讲ajax的缺陷，因为平时我们大多注意的都是ajax给我们所带来的好处诸如用户体验的提升。而对ajax所带来的缺陷有所忽视。

　　下面所阐述的ajax的缺陷都是它先天所产生的。

### 1.ajax干掉了back按钮

即对浏览器后退机制的破坏。后退按钮是一个标准的web站点的重要功能，但是它没法和js进行很好的合作。这是ajax所带来的一个比较严重的问题，因为用户往往是希望能够通过后退来取消前一次操作的。

那么对于这个问题有没有办法？答案是肯定的，用过Gmail的知道，Gmail下面采用的ajax技术解决了这个问题，在Gmail下面是可以后退的，但是，它也并不能改变ajax的机制，它只是采用的一个比较笨但是有效的办法，即用户单击后退按钮访问历史记录时，通过创建或使用一个隐藏的iframe来重现页面上的变更。（例如，当用户在Google Maps中单击后退时，它在一个隐藏的iframe中进行搜索，然后将搜索结果反映到Ajax元素上，以便将应用程序状态恢复到当时的状态）。

要解决这个问题必须明确浏览器记录历史纪录的模式，即：**当地址栏里的URL改变时才会存进历史纪录中**。所以，我们可以通过AJAX请求时动态改变URL但不能刷新整个页面的方式来解决这个问题。第一种方式：URL中可以通过**改变hash的值**（不会使页面刷新）来实现(在AJAX请求时调用location,hash属性来配置，hash还有一个配套的onhashchange事件可以监听hash的变化)，第二种方式：利用H5新加的**history.pushState（）方法（**将当前页面的URL添加到浏览器历史记录里）来实现，结合onpopstate事件监听历史纪录的变化（历史纪录条数变化时触发）。

### 2、安全问题

技术同时也对IT企业带来了新的安全威胁，ajax技术就如同对企业数据建立了一个直接通道。这使得开发者在不经意间会暴露比以前更多的数据和服务器逻辑。ajax的逻辑可以对客户端的安全扫描技术隐藏起来，允许黑客从远端服务器上建立新的攻击。还有ajax也难以避免一些已知的安全弱点，诸如跨站点脚步攻击、SQL注入攻击和基于credentials的安全漏洞等。

### 3、对搜索引擎的支持比较弱。

### 4、破坏了程序的异常机制。

至少从目前看来，像ajax.dll，ajaxpro.dll这些ajax框架是会破坏程序的异常机制的。关于这个问题，我曾经在开发过程中遇到过，但是查了一下网上几乎没有相关的介绍。后来我自己做了一次试验，分别采用ajax和传统的form提交的模式来删除一条数据……给我们的调试带来了很大的困难。

### 5、其他

另外，像其他方面的一些问题，比如说违背了**url和资源定位的初衷**。例如，我给你一个url地址，如果采用了ajax技术，也许你在该url地址下面看到的和我在这个url地址下看到的内容是不同的。这个和资源定位的初衷是相背离的。

一些手持设备（如手机、PDA等）现在还不能很好的支持ajax，比如说我们在手机的浏览器上打开采用ajax技术的网站时，它目前是不支持的，当然，这个问题和我们没太多关系。

# 9.ajax缓存

ajax在IE中有一个奇怪的现象，就是open（）方法使用**GET请求**且**参数URL不改变**时，第二次以后的请求得到始终是第一次请求时的数据，这是因为IE将AJAX请求的数据缓存了下来，下次请求时直接从浏览器缓存中取数据。

解决方法，第一种:改用POST请求。第二种：在参数URL后面加上一个时间戳（**new Date().getTime()**）。

# 10.ajax跨域

浏览器中，同源策略使得不同域的地址不能相互访问，这样做是为了安全，但是我们有时必须要通过AJAX跨域获取资源。

要解决跨域的问题，最常用的就是iframe、JSONP、CROS三种方式。

前两种都不是直接的跨域，是通过某种媒介来达到跨域的效果，而CROS是level2中新加的跨域实现方式。

## 1.iframe跨域

是借助于iframe的src属性可以跨域的能力来将a域下的请求资源转移到iframe上（和b同域），然后iframe来请求b域上的资源，最后返回给a域页面的原理实现的。

## 2.JSONP跨域

是利用script标签的src有跨域的能力来实现的。首先在AJAX请求的时候动态创建一个script 标签，将需要跨域请求的资源路径（b域的）写到script标签的src属性上，b域服务器响应该请求后，返回数据给创建的script元素，然后a域就可以获得b域的资源了。最后将创建的script标签清除（已完成使命了）。

**只支持GEI方法。**

## 3.CROS跨域

是level2新添加的功能， 在level2的XMLHttpRequest对象中，可以将open（）方法的参数URL写成任何域下的URL，浏览器发现这是一个非简单请求，就自动发出一个"预检"请求，要求目标域的服务器确认可以这样请求，其中请求可能有两种。

第一次请求称为“预检”，浏览器先询问服务器，当前网页所在的域名是否在服务器的许可名单之中，以及可以使用哪些HTTP动词和头信息字段。只有得到肯定答复，浏览器才会发出正式的XMLHttpRequest请求，否则就报错。设置服务器端许可，tomcat设置方式如下：



一旦服务器通过了"预检"请求，以后每次浏览器正常的CORS请求（第二次），就都跟简单请求一样，会有一个Origin头信息字段。服务器的回应，也都会有一个Access-Control-Allow-Origin头信息字段。

# 11.ajax文件上传

新版 XMLHttpRequest 对象，不仅可以发送文本信息，还可以上传文件。

　　假定 files 是一个"选择文件"的表单元素（<input type="file" id=’file’/>），我们将它装入 FormData 对象。

var files = document.getElementById(‘file’);

var formData = new FormData ();

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

formData.append ('files[]'， files[i]);

}

　　然后，发送这个 FormData 对象。

xhr.send (formData);

# 12.ajax接收二进制文件

从服务器取回二进制数据，较新的方法是使用新增的 responseType 属性。如果服务器返回文本数据，这个属性的值是"TEXT"，这是默认值。较新的浏览器还支持其他值，也就是说，可以接收其他格式的数据。

1.可以把 responseType 设为 **blob**，表示服务器传回的是二进制对象。

var xhr = new XMLHttpRequest ();

xhr.open ('GET'， '/path/to/image.png');

xhr.responseType = 'blob';

　　 接收数据的时候，用浏览器自带的 Blob 对象即可。

var blob = new Blob ([xhr.response], {type: 'image/png'});

**注意，是读取 xhr.response，而不是 xhr.responseText。**

　2.还可以将 responseType 设为 **arraybuffer**，把二进制数据装在一个数组里。

var xhr = new XMLHttpRequest ();

xhr.open ('GET'， '/path/to/image.png');

xhr.responseType = "arraybuffer";

　　 接收数据的时候，需要遍历这个数组。

var arrayBuffer = xhr.response;

if (arrayBuffer) {

var byteArray = new Uint8Array (arrayBuffer);

for (var i = 0; i < byteArray.byteLength; i++) {

// do something

}

}

# 13、ajax的几种框架

目前我们采用的比较多的ajax框架主要有ajax.dll,ajaxpro.dll,magicajax.dll以及微软的atlas框架。Ajax.dll和Ajaxpro.dll这两个框架差别不大，而magicajax.dll只是封装得更厉害一些，比如说它可以直接返回DataSet数据集，前面我们已经说过，ajax返回的都是字符串，magicajax只是对它进行了封装而已。但是它的这个特点可以给我们带来很大的方便，比如说我们的页面有一个列表，而列表的数据是不断变化的，那么我们可以采用magicajax来处理，操作很简单，添加magicajax之后，将要更新的列表控件放在magicajax的控件之内，然后在pageload里面定义更新间隔的时间就ok了，atlas的原理和magicajax差不多。但是，需要注意的一个问题是，这几种框架都只支持IE，没有进行浏览器兼容方面的处理，用反编译工具察看他们的代码就可以知道。

除了这几种框架之外，我们平时用到的比较多的方式是自己创建xmlHttpRequest对象，这种方式和前面的几种框架相比更具有灵活性。另外，在这里还提一下aspnet2.0自带的异步回调接口，它和ajax一样也可以实现局部的无刷新，但它的实现实际上也是基于xmlhttprequest对象的，另外也是只支持IE，当然这是微软的一个竞争策略。