ajax基础

Ajax,即Asynchronous JavaScript+XML.该技术能够向服务器请求额外的数据而无需卸载页面.

Ajax技术的核心是XMLHttpRequest对象,简称XHR.

一. XMLHttpRequest对象

IE5最先支持XHR对象.在IE5中,XHR是通过MSXML库中的ActiveX对象实现的.因此IE5创建XHR对象需要编写一个函数.在IE7,Firefox,Opera,Chrome和Safari都支持原生的XHR对象。

在现在的浏览器中创建XHR对象的方法如下,使用XMLHttpRequest构造函数.

var xhr=new XMLHttpRequset();

而IE5，IE6需要使用ActiveXObject。

var xhr=new ActiveXObject("Microsoft.XMLHTTP");

function createXHR(){

if(window.XMLHttpRequest){

return new XMLHttpRequest();

}else{

return new ActiveXObject("Microsoft.XMLHTTP");

}

}

二. 使用XMLHttpRequest对象

1. 向服务器发送请求

如需将请求发送到服务器，使用 XMLHttpRequest 对象的 open() 和 send() 方法：

open(method,url,async) 规定请求的类型、URL 以及是否异步处理请求。调用该方法不会起到真正的请求.而只是启动一个请求以备发送。

method：请求的类型"GET"或"POST"。

url：文件在服务器上的位置.位置是相对于执行代码的当前页面的相对路径,或绝对路径。

async：true(异步)或 false(同步)

异步: javascript继续执行,不必等待响应.

同步:javascript要等到服务器响应后再继续执行.

send(string) 将请求发送到服务器。

string：当使用"POST"请求，参数为请求主体发送的数据。

若不为"POST"请求，则参数应为null，参数对有些浏览器来说是必需的。

例:

var xmlhttp=new XMLHttpRequset();

xmlhttp.open("GET","test1.txt",true);

xmlhttp.send(null);

(1) 发送GET请求

发送GET请求。请求参数添加在请求url之后。

xhr.open("GET","demo\_get.asp?t=" + Math.random(),true);

xhr.send(null);

(2) 发送POST请求

发送POST请求需要设置请求域：Content-type：application/x-www-form-urlencoded

该请求域表示数据模拟了表单的方式提交。

请求参数用send(string)方法发送。

xmlhttp.open("POST","ajax\_test.asp",true);

xmlhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");

xmlhttp.send("fname=Bill&lname=Gates");

2. 服务器响应

当得到服务器响应后,响应数据会自动填充XHR对象的属性：

属性 描述

responseText: 作为响应主体被返回的文本.

responseXML: 如果响应的内容类型是text/xml或application/xml,则这个属性包含转着响应数据的XML DOM文档.

status: 响应的HTTP状态.

statusText HTTP状态的说明.

readyState：

说明：

(1) status属性

若接收到响应后，检查status属性，以确定响应已经成功返回。

若成功,该值应为200.

若该值是304表示,即请求是有效的,同时请求的资源并没有被修改,则可以直接使用浏览器中缓存.

(2) readyState属性

若发送的是异步请求,JavaScript会继续执行而不等待响应.此时,检测应XHR对象的readyState属性.该属性表示请求/响应过程的当前获得阶段.

该属性的可取值：

0：未初始化. 尚未调用open()方法.

1：启动. 已经调用open()方法.

2：发送. 已经调用send ()方法,但尚未调用响应.

3：接收. 已经接收到部分响应数据.

4：完成. 已经接收到全部响应数据,而且数据已经可以在客户端使用.

只要readyState属性的值改变就会触发readStatechange事件。为了保证跨浏览器兼容,需要在调用open()之前指定onreadstatechange事件处理函数。

在接收到响应之前可以调用abort()方法来取消异步请求,该方法会停止XHR对象触发事件,且不在运行访问任何与响应有关的对象属性.如xhr.abort();中止请求后,需要对XHR对象进行解引用操作.由于内存原因.不建议重用XHR对象.

三. JSON,JavaScript Object Notation ,JavaScript对象表示法

例:JSON表示一个person对象.

var person={

"name":"大法师",

"age":236,

"sex":"男"

};

注意:JavaScript不要求给对象的属性加引号。但未加引号的属性在JSON中则被视为语法错误。

Ajax从服务器上接收到的JSON为字符串形式。为了使用这些对象或数组。使用eval()函数。把这些字符串转为本地脚本并执行。

注意:在eval()对JSON求值时.

var obj1=eval("{}"); //错误, eval()相当于执行{}空代码块.

在文本放在圆括号内,即可避免所悟.因为圆括号表示值.

var obj2=eval("({})");

var object3=eval("("+jsonText+")"); //通用的解决方法.

在Ajax中使用JSON

Douglas Crockford自己维护着针对JavaScript的JSON序列化器/解析器.

www.json.org/js.html下载后使用该库.

Crockford的JSON库中,有个全局JSON对象,这个对象有两个方法:parse()和stringify().

其中,parse()方法接受两个参数,JSON文本和可选的过滤函数.

安全

eval()函数用来解析JSON.但是还可以解释任意JavaScript代码.

XSS攻击,Cross-Site Scripting

在不过滤JSON数据就直接将其传递给eval(),即有可能发生这个攻击.

Crockford的JSON库可以妥善解析JSON字符串.确保JSON转换为JavaScript对象.过滤掉其中的恶意代码.

四. 使用示例

var url="test.action" //请求url

var xmlhttp=new XMLHttpRequest(); //创建XMLHttpRequest对象

xmlhttp.onreadystatechange=handler; //设置readystatechange事件处理函数。

//在open()调用之前指定onreadystatechange函数。

xmlhttp.open("GET",url); //请求test.xml

xmlhttp.send(null); //发送请求

function test(data){

//Ajax处理返回数据。

}

//事件处理函数

function handler(){

if(this.readyState==4 && this.status==200){

var data=this.responseText; //...接收数据...

/\*

var data=this.responseXML; //或者接收XML

\*/

test(data); //处理数据

}else if(this.readyState!=4 && this.status!=200){

//响应不成功

}

}

五. JQuery Ajax

一直不太喜欢jquery。因为感觉如果jquery用多了，js本身的语法以及js中各种概念会被抹杀掉。

写js的时候对于jquery我是能不用就不用。可是纯javascript写ajax实在太过麻烦。所以jquery也是必须的。

jquery中ajax主要方法有三种：

$.ajax(); 感觉使用起来还是比较麻烦的。

$.post()是对$.ajax()的post方式提交ajax查询的封装。

$.get()是对$.ajax()的get方式提交ajax查询的封装。

$.post(url,postJson,callbackfunction(data,textStatus){})

第一个参数请求url。

第二个参数是一个json，表示发送的参数。

第三个参数是请求处理函数。该函数要求有两个参数，data是服务器返回的参数，textStatus表示ajax与服务器通信是否成功。textStatus与服务器上是否发生异常没有关系。成功的话，textStatus=="success"

例：$.post("Hello1.ashx",{"id":"2"},function(data,textStatus){})

六. 使用Get方式发送请求的浏览器缓存问题

在使用ajax不断请求服务器的时候可能会因为浏览器缓存而导致无法得到服务器最新的数据。

使用post方式请求，不会有缓存问题。使用get方式请求，会产生缓存问题。

解决get请求的缓存问题的方式是：使请求url每次都不一样。

所以如果不使用jquery或者其他js框架，自己写ajax的get请求的话，可以在url之后加一个时间数值或其他什么东西。这样就保证了每次请求都不一样。

例：url为 "HellAjax.ah?"+"t="+new Date()

附. XMLHttpRequest的接口

以下为W3C定义的XMLHttpRequest的接口，更多详细内容请参考W3C官网。

[NoInterfaceObject]

interface XMLHttpRequestEventTarget : EventTarget {

// event handlers

[TreatNonCallableAsNull] attribute Function? onloadstart;

[TreatNonCallableAsNull] attribute Function? onprogress;

[TreatNonCallableAsNull] attribute Function? onabort;

[TreatNonCallableAsNull] attribute Function? onerror;

[TreatNonCallableAsNull] attribute Function? onload;

[TreatNonCallableAsNull] attribute Function? ontimeout;

[TreatNonCallableAsNull] attribute Function? onloadend;

};

interface XMLHttpRequestUpload : XMLHttpRequestEventTarget {

};

enum XMLHttpRequestResponseType {

"",

"arraybuffer",

"blob",

"document",

"json",

"text"

}

[Constructor]

interface XMLHttpRequest : XMLHttpRequestEventTarget {

// event handler

//事件，当XML对象状态（status）发生改变时触发

[TreatNonCallableAsNull] attribute Function? onreadystatechange;

// 以下常量表示XMLHttpRequest对象的状态。XMLHttpRequest对象的status属性只能是以下值。

//未发送请求

const unsigned short UNSENT = 0;

//执行open方法

const unsigned short OPENED = 1;

//已经发送请求

const unsigned short HEADERS\_RECEIVED = 2;

//正在加载

const unsigned short LOADING = 3;

//已经返回数据

const unsigned short DONE = 4;

//XHR对象的状态

readonly attribute unsigned short readyState;

// request

//打开URL的请求

//method：HTTP请求方式，包括GET，POST，HEAD，PUT，DELETE，OPTIONS

//url：请求的url

//async：是否异步请求

//user：可缺省

//password：可缺省

void open(DOMString method, DOMString url, optional boolean async, optional DOMString? user, optional DOMString? password);

//设置HTTP请求头，关于请求头的设置，参见HTTP协议文档（http://ietf.org/rfc/rfc2616）。

void setRequestHeader(DOMString header, DOMString value);

attribute unsigned long timeout;

attribute boolean withCredentials;

readonly attribute XMLHttpRequestUpload upload;

//发送请求

void send();

void send(ArrayBuffer data);

void send(Blob data);

void send(Document data);

void send(DOMString? data);

void send(FormData data);

//撤销请求

void abort();

// response

//HTTP状态

readonly attribute unsigned short status;

//HTTP状态文档

readonly attribute DOMString statusText;

//获取指定的响应头

DOMString getResponseHeader(DOMString header);

//获取所有的响应头

DOMString getAllResponseHeaders();

void overrideMimeType(DOMString mime);

attribute XMLHttpRequestResponseType responseType;

readonly attribute any response;

//响应的文本内容

readonly attribute DOMString responseText;

//响应的XML文档对象

readonly attribute Document responseXML;

};

[Constructor]

interface AnonXMLHttpRequest : XMLHttpRequest {

};