**Ajax简介、优劣势、应用场景以及技术**

**Ajax简介**

* Asynchronous Javascript And XML （异步的JavaScript和XML）
* 它并不是一种单一的技术，而是有机利用一系列交互式网页应用相关的技术所形成的结合体
* AJAX 是一种用于创建快速动态网页的技术。通过在后台与服务器进行少量数据交换，AJAX 可以使网页实现异步更新。这意味着可以在不重新加载整个网页的情况下，对网页的某部分进行更新。

**优点：**

* 页面无刷新，用户体验好。
* 异步通信，更加快的响应能力。
* 减少冗余请求，减轻了服务器负担
* 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

**缺点：**

* ajax干掉了back按钮，即对浏览器后退机制的破坏。
* 存在一定的安全问题。
* 对搜索引擎的支持比较弱。
* 破坏了程序的异常机制。
* 无法用URL直接访问

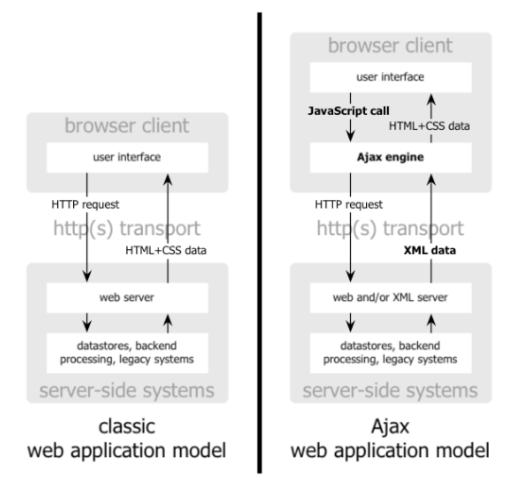
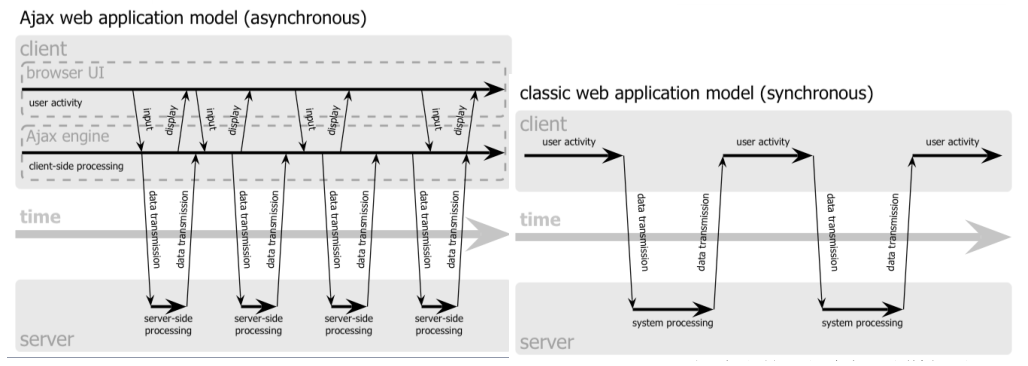
**ajax应用场景**

* 场景 1. 数据验证
* 场景 2. 按需取数据
* 场景 3. 自动更新页面

**AJAX 包含以下五个部分：**

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。

* 使用CSS和XHTML来表示。
* 使用DOM模型来交互和动态显示。
* 数据互换和操作技术，使用XML与XSLT。
* 使用XMLHttpRequest来和服务器进行异步通信。
* 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  
  
第一张图尤其说明了传统Web 应用程序的结构与采用了 AJAX 技术的 Web应用程序的结构上的差别  
主要的差别，其实不是 JavaScript，不是 HTML/XHTML和 CSS，而是采用了 XMLHttpRequest 来向服务器异步的请求 XML 数据  
  
再来看第二张图，传统的 Web 应用模式，用户的体验是割裂的，点击->等待->  
看到新的页面->再点击->再等待。而采用了AJAX技术之后，大部分的计算工作，都是在用户不察觉的情况下，交由服务器去完成了

**创建ajax的步骤**

Ajax的原理简单来说通过XmlHttpRequest对象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。原生创建ajax可分为以下四步

**创建XMLHttpRequest对象**

Ajax的核心是XMLHttpRequest对象，它是Ajax实现的关键，发送异步请求、接受响应以及执行回调都是通过它来完成

所有现代浏览器（IE7+、Firefox、Chrome、Safari 以及 Opera）均内建 XMLHttpRequest 对象。

创建 XMLHttpRequest对象的语法：

|  |
| --- |
| var xhr = new XMLHttpRequest(); |

老版本的 Internet Explorer（IE5 和 IE6）使用ActiveX 对象：

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

为了应对所有的现代浏览器，包括 IE5 和 IE6，请检查浏览器是否支持 XMLHttpRequest对象。如果支持，则创建XMLHttpRequest对象。如果不支持，则创建ActiveXObject：  
兼容各个浏览器的创建Ajax的工具函数

|  |
| --- |
| function createRequest (){      try {          xhr = new XMLHttpRequest();      }catch (tryMS){          try {              xhr = new ActiveXObject("Msxm12.XMLHTTP");          } catch (otherMS) {              try {                  xhr = new ActiveXObject("Microsoft.XMLHTTP");              }catch (failed) {                  xhr = null;              }          }      }      return xhr;  } |

**准备请求**

初始化该XMLHttpRequest对象，接受三个参数：

|  |
| --- |
| xhr.open(method,url,async); |

第一个参数表示请求类型的字符串，其值可以是GET或者POST。

GET请求：

|  |
| --- |
| xhr.open("GET",demo.php?name=tsrot&age=24,true); |

POST请求：

|  |
| --- |
| xhr.open("POST",demo.php,true); |

第二个参数是要作为请求发送目标的URL。

第三个参数是true或false，表示请求是以异步还是同步的模式发出。（默认为true，一般不建议为false）  
false：同步模式发出的请求会暂停所有javascript代码的执行，知道服务器获得响应为止，如果浏览器在连接网络时或者在下载文件时出了故障，页面就会一直挂起。  
true：异步模式发出的请求，请求对象收发数据的同时，浏览器可以继续加载页面，执行其他javascript代码

**发送请求**

|  |
| --- |
| xhr.send(); |

一般情况下，使用Ajax提交的参数多是些简单的字符串，可以直接使用GET方法将要提交的参数写到open方法的url参数中，此时send方法的参数为null或为空。

GET请求：

|  |
| --- |
| xhr.open("GET",demo.php?name=tsrot&age=24,true);  xhr.send(null); |

POST请求：  
如果需要像 HTML 表单那样 POST 数据，请使用 setRequestHeader()来添加 HTTP 头。然后在send()方法中规定您希望发送的数据：

|  |
| --- |
| xhr.open("POST",demo.php,true);  xhr.setRequestHeder("Content-Type","application/x-www-form-urlencoded;charset=UTF-8");  xhr.sen |

**处理响应**

|  |
| --- |
| xhr.onreadystatechange = function(){      if(xhr.readyState == 4 && xhr.status == 200){          console.log(xhr.responseText);      }  } |

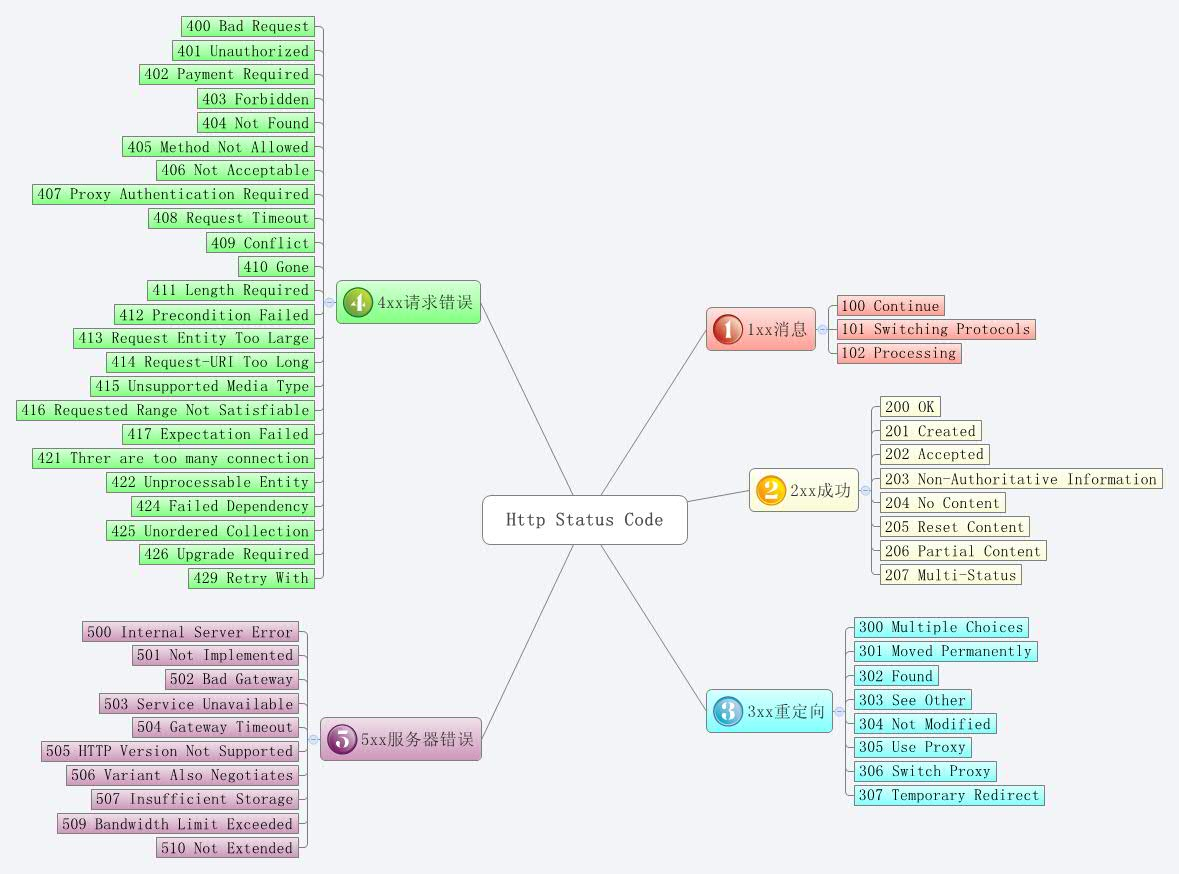
onreadystatechange ：当处理过程发生变化的时候执行下面的函数  
eadyState ：ajax处理过程  
0：请求未初始化（还没有调用 open()）。  
1：请求已经建立，但是还没有发送（还没有调用 send()）。  
2：请求已发送，正在处理中（通常现在可以从响应中获取内容头）。  
3：请求在处理中；通常响应中已有部分数据可用了，但是服务器还没有完成响应的生成。  
4：响应已完成；您可以获取并使用服务器的响应了。

status属性：  
200:”OK”  
404: 未找到页面

responseText：获得字符串形式的响应数据  
responseXML：获得 XML形式的响应数据  
对象转换为JSON格式使用JSON.stringify  
json转换为对象格式用JSON.parse()  
返回值一般为json字符串，可以用JSON.parse(xhr.responseText)转化为JSON对象

从服务器传回的数据是json格式，这里做一个例子说明，如何利用  
1、首先需要从XMLHttpRequest对象取回数据这是一个JSON串，把它转换为真正的JavaScript对象。使用JSON.parse(xhr.responseText)转化为JSON对象  
2、遍历得到的数组，向DOM中添加新元素

|  |
| --- |
| function example(responseText){   var saleDiv= document.getElementById("sales");   var sales = JSON.parse(responseText);   for(var i=0;i<sales.length;i++){    var sale = sales[i];    var div = document.createElement("div");    div.setAttribute("class","salseItem");    div.innerHTML = sale.name + sale.sales;    salseDiv.appendChild(div);   }  } |



**封装例子**

将AJAX请求封装成ajax()方法，它接受一个配置对象params

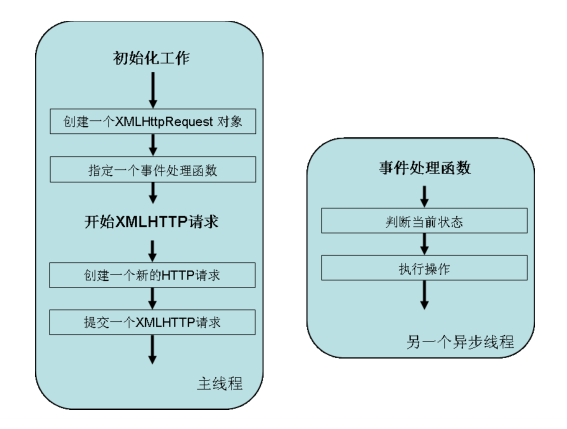
|  |
| --- |
| function ajax(params) {    params = params || {};    params.data = params.data || {};    // 判断是ajax请求还是jsonp请求    var json = params.jsonp ? jsonp(params) : json(params);    // ajax请求    function json(params) {      //  请求方式，默认是GET      params.type = (params.type || 'GET').toUpperCase();      // 避免有特殊字符，必须格式化传输数据      params.data = formatParams(params.data);      var xhr = null;      // 实例化XMLHttpRequest对象      if(window.XMLHttpRequest) {        xhr = new XMLHttpRequest();      } else {        // IE6及其以下版本        xhr = new ActiveXObjcet('Microsoft.XMLHTTP');      }; |

使用实例：

|  |
| --- |
| ajax({    url: 'test.php',   // 请求地址    type: 'POST',   // 请求类型，默认"GET"，还可以是"POST"    data: {'b': '异步请求'},   // 传输数据    success: function(res){   // 请求成功的回调函数      console.log(JSON.parse(res));    },    error: function(error) {}   // 请求失败的回调函数  }); |

这个过程是一定要记在脑子里的

|  |
| --- |
| function ajax(url, success, fail){      // 1. 创建连接      var xhr = null;      xhr = new XMLHttpRequest()      // 2. 连接服务器      xhr.open('get', url, true)      // 3. 发送请求      xhr.send(null);      // 4. 接受请求      xhr.onreadystatechange = function(){          if(xhr.readyState == 4){              if(xhr.status == 200){                  success(xhr.responseText);              } else { // fail                  fail && fail(xhr.status);              }          }      }  } |

  
谈谈JSONP  
要访问web服务器的数据除了XMLHttpRequest外还有一种方法是JSONP  
如果HTML和JavaScript与数据同时在同一个机器上，就可以使用XMLHttpRequest

什么是JSONP？

JSONP(JSON with Padding)是一个非官方的协议，它允许在服务器端集成Script tags返回至客户端，通过javascript callback的形式实现跨域访问（这仅仅是JSONP简单的实现形式）  
JSONP有什么用？  
由于同源策略的限制，XmlHttpRequest只允许请求当前源（域名、协议、端口）的资源，为了实现跨域请求，可以通过script标签实现跨域请求，然后在服务端输出JSON数据并执行回调函数，从而解决了跨域的数据请求  
如何使用JSONP？

在客户端声明回调函数之后，客户端通过script标签向服务器跨域请求数据，然后服务端返回相应的数据并动态执行回调函数  
用XMLHttpRequest时，我们得到一个字符串；要用JSON.parse把字符串转化成对象，使用jsonp时，script标志会解析并执行返回的代码，等我们处理数据时，已经是一个JavaScript对象了

简单实例

|  |
| --- |
| <meta content="text/html; charset=utf-8" http-equiv="Content-Type" />  <script type="text/javascript">      function jsonpCallback(result) {          alert(result.a);          alert(result.b);          alert(result.c);          for(var i in result) {              alert(i+":"+result[i]);//循环输出a:1,b:2,etc.          }      }  </script>  <script type="text/javascript" src="<http://crossdomain.com/services.php?callback=jsonpCallback>"></script>  <!--callback参数指示生成JavaScript代码时要使用的函数jsonpcallback--> |

注意浏览器的缓存问题  
在末尾增加一个随机数可避免频繁请求同一个链接出现的缓存问题

**原生JavaScript实现完整的Ajax、JSONP例子**

|  |
| --- |
| function ajax(params) {      params = params || {};      params.data = params.data || {};      var json = params.jsonp ? jsonp(params) : json(params);      // ajax请求      function json(params) {          params.type = (params.type || 'GET').toUpperCase();          params.data = formatParams(params.data);          var xhr = null;          // 实例化XMLHttpRequest对象          if(window.XMLHttpRequest) {              xhr = new XMLHttpRequest();          } else {              // IE6及其以下版本              xhr = new ActiveXObjcet('Microsoft.XMLHTTP');          };          // 监听事件          xhr.onreadystatechange = function() {              if(xhr.readyState == 4) {                  var status = xhr.status;                  if(status >= 200 && status < 300) {                      var response = '';                      var type = xhr.getResponseHeader('Content-type');                      if(type.indexOf('xml') !== -1 && xhr.responseXML) {                          response = xhr.responseXML; //Document对象响应                      } else if(type === 'application/json') {                          response = JSON.parse(xhr.responseText); //JSON响应                      } else {                          response = xhr.responseText; //字符串响应                      };                      params.success && params.success(response);                  } else {                      params.error && params.error(status);                  }              }          };          // 连接和传输数据          if(params.type == 'GET') {              xhr.open(params.type, params.url + '?' + params.data, true);              xhr.send(null);          } else {              xhr.open(params.type, params.url, true);              //设置提交时的内容类型              xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded; charset=UTF-8');              xhr.send(params.data);          }      }      // jsonp请求      function jsonp(params) {          //创建script标签并加入到页面中          var callbackName = params.jsonp;          var head = document.getElementsByTagName('head')[0];          // 设置传递给后台的回调参数名          params.data['callback'] = callbackName;          var data = formatParams(params.data);          var script = document.createElement('script');          head.appendChild(script);          //创建jsonp回调函数          window[callbackName] = function(json) {              head.removeChild(script);              clearTimeout(script.timer);              window[callbackName] = null;              params.success && params.success(json);          };          //发送请求          script.src = params.url + '?' + data;          //超时处理          if(params.time) {              script.timer = setTimeout(function() {                  window[callbackName] = null;                  head.removeChild(script);                  params.error && params.error({                      message: '超时'                  });              }, time);          }      };      //格式化参数      function formatParams(data) {          var arr = [];          for(var name in data) {              arr.push(encodeURIComponent(name) + '=' + encodeURIComponent(data[name]));          };          // 添加一个随机数，防止缓存          arr.push('v=' + random());          return arr.join('&');      }      // 获取随机数      function random() {          return Math.floor(Math.random() \* 10000 + 500);      }  } |

使用

|  |
| --- |
| ajax({                  url: 'get.php',                  type: 'GET',                  data: {'intro': 'get请求'},                  success:function(res){                      res = JSON.parse(res);                      document.getElementById('a').innerHTML = res.intro;                      console.log(res);                  }              });          ajax({                  url: 'post.php',                  type: 'POST',                  data: {'intro': 'post请求'},                  success:function(res){                      res = JSON.parse(res);                      document.getElementById('b').innerHTML = res.intro;                      console.log(res);                  }              });          ajax({                  url: '<http://music.qq.com/musicbox/shop/v3/data/hit/hit_all.js>',                  jsonp: 'jsonpCallback',                  data: {'callback': 'jsonpCallback'},                  success:function(res){                      JsonCallback(json);                  }              }); |

下面我们就根据以上 封装的例子跨域获取qq音乐的数据

[在线演示–跨域获取qq音乐的数据](https://codepen.io/poetries/pen/oBMKmG)

下面的方法也可以实现

使用jQuery实现

|  |
| --- |
| <script src="jquery-3.1.0.min.js"></script>          <script type="text/javascript">              $.ajax({                    type: "get",                    async: false,                    url: "<http://music.qq.com/musicbox/shop/v3/data/hit/hit_all.js>",                    dataType: "jsonp",                    jsonp: "callback",                    jsonpCallback: "JsonCallback",                    scriptCharset: 'GBK',//设置编码，否则会乱码                    success: function(data) {                      //var result = JSON.stringify(data);                        JsonCallback(data);                    },                    error: function() {                      alert('fail');                    }                  });              function JsonCallback(json){                  var data = json.songlist;                  var html = '';                  for (var i=0;i<data.length;i++) {                      document.write(data[i].url+"<br>");                  }              }          </script>  ```  - 原生js简洁实现  ```javascript  var script = document.createElement("script");              script.src = '<http://music.qq.com/musicbox/shop/v3/data/hit/hit_all.js?callback=JsonCallback>';              document.body.appendChild(script);              function JsonCallback(json){                  var data = json.songlist;                  var html = '';                  for (var i=0;i<data.length;i++) {                      console.log(data[i]);                  }              } |

**jQuery中的Ajax**

jQuery中的ajax封装案例

|  |
| --- |
| //ajax请求后台数据  var btn =  document.getElementsByTagName("input")[0];  btn.onclick = function(){        ajax({//json格式          type:"post",          url:"post.php",          data:"username=poetries&pwd=123456",          asyn:true,          success:function(data){              document.write(data);          }      });  }  //封装ajax  function ajax(aJson){      var ajx = null;      var type = aJson.type || "get";      var asyn = aJson.asyn || true;      var url = aJson.url;        // url 接收 传输位置      var success = aJson.success;// success 接收 传输完成后的回调函数      var data = aJson.data || '';// data 接收需要附带传输的数据        if(window.XMLHttpRequest){//兼容处理          ajx = new XMLHttpRequest();//一般浏览器      }else      {          ajx = new ActiveXObject("Microsoft.XMLHTTP");//IE6+      }      if (type == "get" && data)      {          url +="/?"+data+"&"+Math.random();      }        //初始化ajax请求      ajx.open( type , url , asyn );      //规定传输数据的格式      ajx.setRequestHeader('content-type','application/x-www-form-urlencoded');      //发送ajax请求（包括post数据的传输）      type == "get" ?ajx.send():ajx.send(aJson.data);        //处理请求      ajx.onreadystatechange = function(aJson){        if(ajx.readState == 4){            if (ajx.status == 200 && ajx.status<300)//200是HTTP 请求成功的状态码          {              //请求成功处理数据              success && success(ajx.responseText);          }else{              alert("请求出错"+ajx.status);            }      }        } |

jQuery中的Ajax的一些方法

jquery对Ajax操作进行了封装，在jquery中的$.ajax()方法属于最底层的方法，第2层是load()、$.get()、$.post();第3层是$.getScript()、$.getJSON()，第2层使用频率很高

load()方法  
load()方法是jquery中最简单和常用的ajax方法，能载入远程HTML代码并插入DOM中 结构为：load(url,[data],[callback])  
使用url参数指定选择符可以加载页面内的某些元素 load方法中url语法：url selector 注意：url和选择器之间有一个空格  
传递方式  
load()方法的传递方式根据参数data来自动指定，如果没有参数传递，则采用GET方式传递，反之，采用POST  
回调参数  
必须在加载完成后才执行的操作，该函数有三个参数 分别代表请求返回的内容、请求状态、XMLHttpRequest对象  
只要请求完成，回调函数就会被触发

|  |
| --- |
| $("#testTest").load("test.html",function(responseText,textStatus,XMLHttpRequest){      //respnoseText 请求返回的内容      //textStatus 请求状态 ：sucess、error、notmodified、timeout      //XMLHttpRequest  }) |

load方法参数

| **参数名称** | **类型** | **说明** |
| --- | --- | --- |
| url | String | 请求HTML页面的URL地址 |
| data(可选) | Object | 发送至服务器的key / value数据 |
| callback(可选) | Function | 请求完成时的回调函数，无论是请求成功还是失败 |

$.get()和$.post()方法

load()方法通常用来从web服务器上获取静态的数据文件。在项目中需要传递一些参数给服务器中的页面，那么可以使用$.get()和$.post()或$.ajax()方法

注意：$.get()和$.post()方法是jquery中的全局函数

$.get()方法

$.get()方法使用GET方式来进行异步请求  
结构为：$.get(url,[data],callback,type)  
如果服务器返回的内容格式是xml文档，需要在服务器端设置Content-Type类型 代码如下：header("Content-Type:text/xml:charset=utf-8") //php  
$.get()方法参数解析

| **参数** | **类型** | **说明** |
| --- | --- | --- |
| url | String | 请求HTML页的地址 |
| data(可选) | Object | 发送至服务器的key/ value 数据会作为QueryString附加到请求URL中 |
| callback(可选) | Function | 载入成功的回调函数（只有当Response的返回状态是success才调用该方法） |
| type(可选) | String | 服务器返回内容的格式，包括xml、html、script、json、text和\_default |

$.post()方法

它与$.get()方法的结构和使用方式相同，有如下区别  
GET请求会将参数跟张乃URL后进行传递，而POST请求则是作为Http消息的实体内容发送给web服务器，在ajax请求中，这种区别对用户不可见  
GET方式对传输数据有大小限制（通常不能大于2KB），而使用POST方式传递的数据量要比GET方式大得多（理论不受限制）  
GET方式请求的数据会被浏览器缓存起来，因此其他人可以从浏览器的历史纪录中读取这些数据，如：账号、密码。在某种情况下，GET方式会带来严重的安全问题，而POST相对来说可以避免这些问题  
GET和POST方式传递的数据在服务端的获取也不相同。在PHP中，GET方式用$\_GET[]获取；POST方式用$\_POST[]获取;两种方式都可用$\_REQUEST[]来获取  
总结

使用load()、$.get()和$.post()方法完成了一些常规的Ajax程序，如果还需要复杂的Ajax程序，就需要用到$.ajax()方式  
$.ajax()方法

$.ajax()方法是jquery最底层的Ajax实现，它的结构为$.ajax(options)  
该方法只有一个参数，但在这个对象里包含了$.ajax()方式所需要的请求设置以及回调函等信息，参数以key / value存在，所有参数都是可选的  
$.ajax()方式常用参数解析

| **参数** | **类型** | **说明** |
| --- | --- | --- |
| url | String | (默认为当前页地址)发送请求的地址 |
| type | String | 请求方式（POST或GET）默认为GET |
| timeout | Number | 设置请求超时时间（毫秒） |
| dataType | String | 预期服务器返回的类型。可用的类型如下  **xml**:返回XML文档，可用jquery处理 **html**:返回纯文本的HTML信息，包含的script标签也会在插入DOM时执行 **script**：返回纯文本的javascript代码。不会自动缓存结果，除非设置cache参数。注意：在远程请求时，所有的POST请求都将转为GET请求 **json**:返回JSON数据 **jsonp**:JSONP格式，使用jsonp形式调用函数时，例如：myurl?call back=?,jquery将自动替换后一个？为正确的函数名，以执行回调函数 **text**:返回纯文本字符串 |
| beforeSend | Function | 发送请求前可以修改XMLHttpRequest对象的函数，例如添加自定义HTTP头。在beforeSend中如果返回false可以取消本次Ajax请求。XMLHttpRequest对象是唯一的参数 function(XMLHttpRequest){ this;//调用本次Ajax请求时传递的options参数 } |
| complete | Function | 请求完成后的回调函数（请求成功或失败时都调用） 参数：XMLHttpRequest对象和一个描述成功请求类型的字符串 function(XMLHttpRequest,textStatus){ this;//调用本次Ajax请求时传递的options参数 } |
| success | Function | 请求成功后调用的回调函数，有两个参数 (1)由服务器返回，并根据dataTyppe参数进行处理后的数据 (2)描述状态的字符串 function(data,textStatus){ //data可能是xmlDoc、``jsonObj、html、text等 this;//调用本次Ajax请求时传递的options参数 } |
| error | Function | 请求失败时被调用的函数 |
| global | Boolean | 默认为true。表示是否触发全局Ajax事件，设置为false将不会触发。AjaxStart或AjaxStop可用于控制各种Ajax事件 |

原文地址：<http://blog.poetries.top/2016/11/26/Ajax-summary/>