AJAX 技术

1. Ajax介绍

- Ajax = Asynchronous JavaScript and XML（异步的 JavaScript 和 XML）。
- Ajax 不是新的编程语言，而是一种使用现有标准的新方法。
- 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。
- 起源：在 2005 年，Google 通过其 Google Suggest 使 Ajax 变得流行起来。

2. 创建 XMLHttpRequest 对象

- XMLHttpRequest 是 Ajax 的基础，用于在后台与服务器交换数据。
- 所有现代浏览器均支持 XMLHttpRequest 对象（IE5 和 IE6 使用 ActiveXObject）

```
//创建一个兼容的XMLHttpRequest请求对象
var xmlhttp;
if(window.XMLHttpRequest){
  // code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
}else{
  // code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
```

- 属性：
  - readyState //请求状态：0,1,2,3,4
  - responseText //响应内容
  - responseXML //xml响应对象
  - status //浏览器响应状态：200正常， 404 请求地址不存在 ,,
  - statusText //状态内容
  - onreadystatechange //回调函数属性
- 方法：
  - abort() //取消当前响应，关闭连接并且结束任何未决的网络活动。
  - getAllResponseHeaders() //把 HTTP 响应头部作为未解析的字符串返回。
  - getResponseHeader() //返回指定的 HTTP 响应头部的值
  - open() //初始化 HTTP 请求参数
  - send() //发送 HTTP 请求，使用传递给 open() 方法的参数
  - setRequestHeader() //向一个打开但未发送的请求设置或添加一个 HTTP 请求。
- 模拟POST提交代码：

```
  xmlhttp.open("POST","ajax_test.php",true);
  xmlhttp.setRequestHeader("Content-type","application/x-www-form-
urlencoded");
  xmlhttp.send("fname=Bill&lname=Gates");
```

3. JavaScript版的Ajax实现步骤

- 使用 AJAX 的过程可以类比平常我们访问网页过程

```
// 1. 创建一个 XMLHttpRequest 类型的对象 -- 相当于打开了一个浏览器
var xhr = new XMLHttpRequest();

// 2. 打开与一个网址之间的连接 -- 相当于在地址栏输入访问地址
xhr.open('GET', '/api/demo');

// 3. 通过连接发送一次请求 -- 相当于回车或者点击访问发送请求
xhr.send(null)

// 4. 指定 xhr 状态变化事件处理函数 -- 相当于处理网页呈现后的操作
xhr.onreadystatechange = function () {
  // 通过 xhr 的 readyState 判断此次请求的响应是否接收完成
  if (this.readyState === 4) {
    // 通过 xhr 的 responseText 获取到响应的响应体
    console.log(this)
  }
}
```

- readyState
  - 由于 readystatechange 事件是在 xhr 对象状态变化时触发（不单是在得到响应时），也就意味着这个事件会被 触发多次，所以我们有必要了解每一个状态值代表的含义：

readyState	状态描述	说明
0	UNSENT	代理（XHR）被创建，但尚未调用 open() 方法。
1	OPENED	open() 方法已经被调用，建立了连接。
2	HEADERS_RECEIVED	send() 方法已经被调用，并且已经可以获取状态行和响应头。
3	LOADING	响应体下载中， responseText 属性可能已经包含部分数据。
4	DONE	响应体下载完成，可以直接使用 responseText 。

3.1 GET请求参数处理

```
//创建Ajax请求对象
var xhr = new XMLHttpRequest();

// GET 请求传递参数通常使用的是问号传参
// 这里可以在请求地址后面加上参数，从而传递数据到服务端
xhr.open('GET', './delete.php?id=1');

// 一般在 GET 请求时无需设置响应体，可以传 null 或者干脆不传
xhr.send(null);

//绑定回调函数
xhr.onreadystatechange = function () {
  if (this.readyState === 4) {
    console.log(this.responseText)
  }
}
```

3.2 POST请求参数处理

- POST 请求过程中，都是采用请求体承载需要提交的数据。

```
var xhr = new XMLHttpRequest();

// open 方法的第一个参数的作用就是设置请求的 method
xhr.open('POST', './add.php');

// 设置请求头中的 Content-Type 为 application/x-www-form-urlencoded
// 标识此次请求的请求体格式为 urlencoded 以便于服务端接收数据
xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');

// 需要提交到服务端的数据可以通过 send 方法的参数传递
// 格式：key1=value1&key2=value2
xhr.send('key1=value1&key2=value2');

xhr.onreadystatechange = function () {
  if (this.readyState === 4) {
    console.log(this.responseText);
  }
}
```

3.3. 同步与异步

xhr.open() 方法第三个参数要求传入的是一个 bool 值，其作用就是设置此次请求是否采用异步方式执行，默认 为 true ， 如果需要同步执行可以通过传递 false 实现：

```
console.log('before ajax');

var xhr = new XMLHttpRequest();
```

```
// 默认第三个参数为 true 意味着采用异步方式执行
xhr.open('GET', './time.php', true);
//xhr.open('GET', './time.php', false); //同步方式(不推荐，已废弃)
//(不推荐)主线程上的同步XMLHttpRequest被弃用，因为它会对最终用户体验造成不利影响

xhr.send(null);
xhr.onreadystatechange = function () {
  if (this.readyState === 4) {
    // 这里的代码最后执行
    console.log('request done');
  }
}
console.log('after ajax');
```

3.4 AJAX 请求封装

```
/**
 * 发送一个 AJAX 请求
 * @param {String} method 请求方法
 * @param {String} url 请求地址
 * @param {Object} params 请求参数
 * @param {Function} done 请求完成过后需要做的事情（委托/回调）
 */
function ajax (method, url, params, done) {
  // 统一转换为大写便于后续判断
  method = method.toUpperCase();
  // 对象形式的参数转换为 urlencoded 格式
  var pairs = [];
  for (var key in params) {
    pairs.push(key + '=' + params[key]);
  }
  var querystring = pairs.join('&');
  //创建Ajax请求对象
  var xhr = window.XMLHttpRequest ? new XMLHttpRequest() : new
ActiveXObject('Microsoft.XMLHTTP');
  //设置请求回调处理函数
  xhr.addEventListener('readystatechange', function () {
    if (this.readyState !== 4)
      return; // 尝试通过 JSON 格式解析响应体
    try {
      done(JSON.parse(this.responseText));
    } catch (e) {
      done(this.responseText);
    }
  });

  // 如果是 GET 请求就设置 URL 地址 问号参数
  if (method === 'GET') {
```

```
    url += '?' + querystring
  }

  xhr.open(method, url);
  // 如果是 POST 请求就设置请求体
  var data = null;
  if (method === 'POST') {
    xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded')
    data = querystring;
  }
  xhr.send(data);
}

//测试
ajax('get', './get.php', { id: 123 }, function (data) { console.log(data) });

ajax('post', './post.php', { foo: 'posted data' }, function (data) {
console.log(data) });
```

4. jQuery 中的 Ajax

- jQuery 中有一套专门针对 AJAX 的封装，功能十分完善，经常使用，需要着重注意。
- 在线文档：https://jquery.cuishifeng.cn/index.html

4.1 load( )

- load(url,[data],[callback]) -- 载入远程 HTML 文件代码并插入至 DOM 中。

```
$("#did").load("/test/info");
```

4.2 $.get( )

- $.get(url,[data],[fn],[type]) -- 通过远程 HTTP GET 请求载入信息。
  - url:待载入页面的URL地址
  - data:待发送 Key/value 参数。
  - callback:载入成功时回调函数。
  - type:返回内容格式，xml, html, script, json, text, _default。

```
$.get("/test/info",{name:"John",time:"2pm"},function(data){
  alert("Data Loaded: "+data);
});
```

4.3 $.post( )

- $.post(url,[data],[fn],[type]) -- 通过远程 HTTP POST 请求载入信息。
  - url:待载入页面的URL地址
  - data:待发送 Key/value 参数。

- callback:载入成功时回调函数。
- type:返回内容格式，xml, html, script, json, text, _default。

```
$.post("/test/info",{name:"John",time:"2pm"},function(data){
  alert("Data Loaded: "+data);
});
```

4.4 $.ajax(url,[settings])

- 通过 HTTP 请求加载远程数据,简单易用的高层实现见 .get,.post 等。$.ajax() 返回其创建的 XMLHttpRequest 对象。

```
$.ajax({
   type: "POST",  //请求方式
   url: "some.php", //请求url地址
   data: "name=John&location=Boston", //请求参数
   dataType: "json", //返回的数据类型
   success: function(msg){ //成功回调函数
     alert( "Data Saved: " + msg );
   },
   error: function(){
      //失败回调函数
   }
});
```

5. Fetch

- fetch是一种HTTP数据请求的方式，是XMLHttpRequest的一种替代方案。
- fetch不是ajax的进步封装，而是原生js。
- fetch函数就是原生js，没有使用XMLHttpRequest对象。

5.1 传统的ajax和fetch的区别：

- ajax使用步骤 1.创建XmlHttpRequest对象 2.调用open方法设置基本请求信息 3.设置发送的数据，发送请求 4.注册监听的回调函数 5.拿到返回值，对页面进行更新

```
//1.创建Ajax对象
var xhr=new XMLHttpRequest();
//2.连接服务器（打开和服务器的连接）
xhr.open('GET', url, true);
//3.发送
xhr.send(null);
//4.接收
xhr.onreadystatechange=function (){
  if(xhr.readyState==4){//请求状态
    if(xhr.status==200){//响应状态
      alert('成功了：'+xhr.responseText);
    }else{
```

```
        alert('失败了');
      }
    }
  }
```

- fetch特点 1、第一个参数是URL: 2、第二个是可选参数，可以控制不同配置的 init 对象 3、使用了JavaScript Promises 来处理结果/回调: fetch的配置 Promise fetch(String url [, Object options]); Promise fetch(Request req [, Object options]);

```
fetch(url).then(response => response.json())
  .then(data => console.log(data))
  .catch(e => console.log("Oops, error", e))
```

fetch和ajax 的主要区别 1、fetch()返回的promise将不会拒绝http的错误状态，即使响应是一个HTTP 404或者500 2、在默认情况下 fetch不会接受或者发送cookies

。