Ajax 实现与后台服务通信

1、form 表单

2、ajax

3、websocket（不讨论）

首先，最原始的，通过 form 表单以 post/get 方式提交数据。

当你点击 submit 按钮时，浏览器会默认把你在 input 里面输入的数据，以 post 或 get 的方式提交到 form 表单中的 action 这个地址。相当于你提交表单时，就会向服务器发送一个请求，然后服务器会接受并处理提交过来的 form 表单，最后返回一个新的网页。你可以结合以下代码来理解这段话。

```
<form action="/form.html" method="post/get">
    <input type="text" name="username" placeholder="username">
    <input type="password" name="password" placeholder="password">
    <input type="submit">
  </form>
```

1、get 提交数据：请求参数（一般为 input 里的值）拼装成 url，相当于向服务器发 url 请求。

2、post 提交数据：直接向服务器发请求，参数直接发给后台。

但是，这种方法会导致几个问题：

1、在提交时，页面会发生跳转或刷新，导致用户体验不好。

2、单项提交，把数据提交给后台，但是不知道后台会给出怎样的响应，因为提交后页面就发生跳转了。比如：用户登录，那么就不知道到底是注册成功了还是失败了。

3、浪费宽带。因为前后两个页面中的大部分 HTML 代码往往是相同的。但由于每次应用的交互都需要向服务器发送请求，应用的响应时间就依赖于服务器的响应时间，这就导致了用户界面的响应比本地应用慢的多。

为了解决上述问题，2005 年出现了 Ajax。

一、什么是 Ajax

1、Ajax 的全称是 Asynchronous JavaScript and XML，即**异步 JavaScript+XML**。

2、它是一种技术方案，但并不是一种新技术。

3、它依赖的是现有的 CSS/HTML/Javascript，而其中**最核心的依赖是浏览器提供的 XMLHttpRequest 对象。这个对象为向服务器发送请求和解析服务器响应提供了流畅的接口，使得浏览器可以发出 HTTP 请求与接收 HTTP 响应，实现在页面不刷新的情况下和服务端进行数据交互。**

Ajax 和 **XMLHttpRequest** 两者的关系：我们使用 XMLHttpRequest 对象来发送一个 Ajax 请求。

二、怎么实现在页面不刷新的情况下和服务端进行数据交互?

1、XMLHttpRequest 对象

2、fecth（不讨论）

XMLHttpRequest 对象

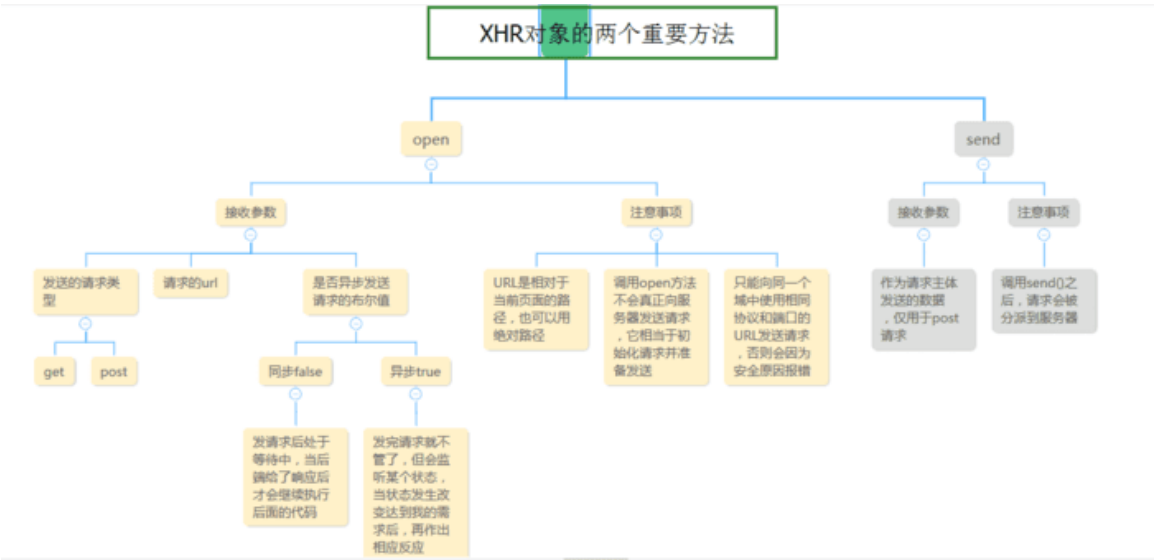
为了便于我们理解怎么使用 XMLHttpRequest 对象实现在页面不刷新的情况下和服务端进行数据交互，我们先来看下下面的代码。

```
<script>
  var xhr = new XMLHttpRequest()  //创建一个对象
  xhr.open('GET','/helloAjax.json',false)  //设置ajax,.open()方法里面的三个参数分别是：要发送的请求类型、请求的url、表示是否异步发送请求的布尔值
  xhr.send()  //发出请求
  var data = xhr.responseText  //当请求到来时，读取请求中的数据
  console.log(data)  //输出
</script>
```

这样就是一个使用 XMLHttpRequest 对象发送的 Ajax 请求了，现在我们来分析分析这段代码。

首先，XMLHttpRequest 构造函数通过 new 的方式构造一个 XHR 对象，并将这个对象赋值给 xhr（可取任意名字）

然后，调用 XMLHttpRequest 对象的方法 open 与 send。XMLHttpRequest 对象的两个重要方法



调用 send 方法之后请求被发往服务器，由于**这次请求是同步的**，JS 代码会在 `xhr.send()`这个步骤暂停掉，一直等到服务器根据请求生成响应（Response），传回给 XHR 对象，再继续执行。

最后，在收到响应后相应数据会填充到 XHR 对象的属性。

有四个相关属性会被填充：

1、responseText —— 从服务器进程作为响应主体被返回的文本。

2、responseXML —— 从服务器进程返回的 DOM 兼容的文档数据对象。

3、status —— 响应的 HTTP 状态。即从服务器返回的数字代码，如 404（未找到）和 200（已就绪）。

4、statusText —— HTTP 状态的说明。伴随状态码的字符串信息。

但多数情况下，我们还是要发送异步请求，才能让 JavaScript 继续执行而不必等待响应。为了更好的理解 **ajax 发送异步请求**，我们来看以下代码

.

```
   <script>
        var xhr = new XMLHttpRequest()
        //请求响应过程的当前活动阶段
        xhr.onreadystatechange = function(){
            console.log('readyState:',xhr.readyState)
        }
        xhr.open('GET','hello.json',true)
        xhr.send()
        //监听请求状态
        xhr.onload = function(){              //onload 相当于 readyState=4
            console.log(xhr.status)
            if((xhr.status >= 200 && xhr.status <= 300) || xhr.status === 304){
                console.log(xhr.responseText)
            }else{
                console.log(error)
            }
        }
    </script>
```

上述代码中，XHR 对象的 readyState 属性，**表示请求响应过程的当前活动阶段**。该属性可取的值如下：

- 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，但尚未接收到响应。
- 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，而且已经可以在客户端使用了。

```
//onload 表示 readyState = 4
```

【注意】

1、只要 **readyState** 属性的值由一个值变成另一个值，就会触发一次 **readystatechange 事件**

2、必须在调用 open()方法之前指定 **readystatechange 事件**处理程序才能确保跨浏览器兼容性。

GET 请求/POST 请求

- 与 POST 相比，GET 更简单也更快，并且在大部分情况下都能用。
- 然而，在以下情况中，请使用 POST 请求：

1、无法使用缓存文件（更新服务器上的文件或数据库）

2、向服务器发送大量数据（POST 没有数据量限制）

3、发送包含未知字符的用户输入时，POST 比 GET 更稳定也更可靠

- **GET 请求/POST 请求使用方法**

一个简单的 GET 请求

```
xhr.open('GET','lazyLoad.html',true)
xhr.send()
```

如果希望通过 GET 方法发送信息，可以向 URL 末尾添加字符串参数

```
xhr.open('GET','/login?username=Iris&password=12345',true)
xhr.send()
```

一个简单的 POST 请求

```
xhr.open('POST','/login',true)
xhr.send('username=Iris&password=12345')
```

如果向 send()里面传递一个对象，可以用函数将该对象拼接成字符串形式

```
        xhr.open('POST','/login',true)
        xhr.send(makeUrl({              //step3、这个时候send()里面就不用发字符串了，直接发传递的对象就好了
            username:'Iris',
            address:'ChangSha',
            age:21
        }))
        //step2、用户传递的是一个对象时    (实参是用户传递的这个对象)
        makeUrl({
```

```
            username:'Iris',
            address:'ChangSha',
            age:21
        })
        //step1、将拼接的过程写成一个函数，向函数makeUrl()里面传递一个形参
        function makeUrl(obj){
            var arr = []            //遍历这个对象
            for(var key in obj){
                arr.push(key + '=' + obj[key])
            }
            return arr.join('&')
}
```