Ajax 学习笔记

第 1 章：原生 AJAX

1.1 AJAX 简介

- AJAX 全称为 Asynchronous JavaScript And XML，就是异步的 JS 和 XML。
- 通过 AJAX 可以在浏览器中向服务器发送异步请求，最大的优势： `无刷新获取数据` 。
- AJAX 不是新的编程语言，而是一种将现有的标准组合在一起使用的新方式。

1.2 XML 简介

- XML 可扩展标记语言。
- XML 被设计用来传输和存储数据。
- XML 和 HTML 类似，不同的是 HTML 中都是预定义标签，而 XML 中没有预定义标签，全都是自定义标签，用来表示一些数据。

比如说我有一个学生数据：

```
name = "孙悟空" ; age = 18 ; gender = "男" ;
```

用 XML 表示：

```
<student>
<name>孙悟空</name>
<age>18</age>
<gender>男</gender>
</student>
```

现在已经被 JSON 取代了。

用 JSON 表示：

```
{"name":"孙悟空","age":18,"gender":"男"}
```

1.3 AJAX 的特点

1.3.1 AJAX 的优点

1. 可以无需刷新页面而与服务器端进行通信。
2. 允许你根据用户事件来更新部分页面内容。

1.3.2 AJAX 的缺点

1. 没有浏览历史，不能回退
2. 存在跨域问题(同源)
3. SEO（搜索引擎优化） 不友好

1.4 AJAX 的使用

HTTP

HTTP(hypertext transport protocol) 协议，又叫 【超文本传输协议】，协议详细规定了浏览器和万维网服务器之间相互通信的规则。

请求报文

重点是格式与参数

```
行      GET  /s?ie=utf-8  HTTP/1.1
头      Host: atguigu.com
         Coolie: name=guigu
         Content-type: application/x-www-form-urlencoded
         User-Agent: chorme 83
空行
体      username=admin&password=admin
```

响应报文

```
 行      HTTP/1.1  200  OK
 头      Content-type: text/html;charset=utf-8
          Content-length: 2048
          Content-encoding: gzip

 空行
 体      <html>
             <head>
             </head>
             <body>
                 <h1>尚硅谷</h1>
             </body>
          </html>
```

- 404：找不到
- 403：被禁止
- 401：未授权
- 500：内部错误
- 200：OK

Express 基本使用

Express 是基于 Node.js 平台，快速、开放、极简的 Web 开发框架

1. 安装 nodejs
2. 初始化

```
init --yes
```

3. 安装 Express 包

```
npm i express
```

4. 在 js 文件中引入 express

```
const express = require('express');
```

5. 创建应用对象

```
const app = express();
```

6. 创建路由规则

```
// request 是对请求报文的封装
// response 是对响应报文的封装
app.get('/', (request, response) => {
    // 设置响应
    response.send('HELLO EXPRESS');
})
```

7. 监听端口，启动服务

```
app.listen(8000, () => {
    console.log("服务器已经启动，8000 端口监听中")
})
```

```
node xxx.js
```

1.4.1 核心对象

XMLHttpRequest，AJAX 的所有操作都是通过该对象进行的。

1.4.2 使用步骤

1. 创建 XMLHttpRequest 对象

```
var xhr = new XMLHttpRequest();
```

2. 设置请求信息

```
// 设置请求方法和url
xhr.open(method, url);
// 可以设置请求头，一般不设置
xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
```

3. 发送请求

```
xhr.send(body) // get 请求不传 body 参数，只有 post 请求使用
```

4. 接收响应

```
// xhr.responseXML 接收 xml 格式的响应数据
// xhr.responseText 接收文本格式的响应数据
// on 当 ..... 时候
// readystate 是 xhr 对象中的属性，表示状态 0 1 2 3 4
// change 改变
xhr.onreadystatechange = function (){
    // 判断 服务端是否返回了所有结果 以及 响应状态码
    if(xhr.readyState == 4 && xhr.status == 200){
        var text = xhr.responseText;
        console.log(text);
    }
}
```

1.4.3 解决 IE 缓存问题

问题：在一些浏览器中(IE),由于缓存机制的存在，ajax 只会发送的第一次请求，剩余多次请求不会在发送给浏览器而是直接加载缓存中的数据。

解决方式：浏览器的缓存是根据 url 地址来记录的，所以我们只需要修改 url 地址即可避免缓存问题

```
xhr.open("get", "/testAJAX?t="+Date.now());
```

1.4.4 AJAX 请求状态

xhr.readyState 可以用来查看请求当前的状态

https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequest/readyState

- 0: 表示 XMLHttpRequest 实例已经生成，但是 open() 方法还没有被调用。
- 1: 表示 send() 方法还没有被调用，仍然可以使用 setRequestHeader()，设定 HTTP请求的头信息。
- 2: 表示 send()方法已经执行，并且头信息和状态码已经收到。
- 3: 表示正在接收服务器传来的 body 部分的数据。
- 4: 表示服务器数据已经完全接收，或者本次接收已经失败了

GET

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>AJAX GET 请求</title>
    <style>
        #result{
            width:200px;
            height:100px;
            border:solid 1px #90b;
        }
    </style>
</head>
<body>
    <button>点击发送请求</button>
    <div id="result"></div>

    <script>
        //获取button元素
        const btn = document.getElementsByTagName('button')[0];
        const result = document.getElementById("result");
        //绑定事件
        btn.onclick = function(){
            //1. 创建对象
            const xhr = new XMLHttpRequest();
            //2. 初始化 设置请求方法和 url
            xhr.open('GET', 'http://127.0.0.1:8000/server?a=100&b=200&c=300');
            //3. 发送
            xhr.send();
            //4. 事件绑定 处理服务端返回的结果
            // on  when 当....时候
```

```
            // readystate 是 xhr 对象中的属性, 表示状态 0 1 2 3 4
            // change  改变
            xhr.onreadystatechange = function(){
                //判断 (服务端返回了所有的结果)
                if(xhr.readyState === 4){
                    //判断响应状态码 200  404  403 401 500
                    // 2xx 成功
                    if(xhr.status >= 200 && xhr.status < 300){
                        //处理结果  行 头 空行 体
                        //响应
                        // console.log(xhr.status);//状态码
                        // console.log(xhr.statusText);//状态字符串
                        // console.log(xhr.getAllResponseHeaders());//所有响应头
                        // console.log(xhr.response);//响应体
                        //设置 result 的文本
                        result.innerHTML = xhr.response;
                    }else{

                    }
                }
            }


        }
    </script>
</body>
</html>
```

POST

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>AJAX POST 请求</title>
    <style>
        #result{
            width:200px;
            height:100px;
            border:solid 1px #903;
        }
    </style>
</head>
<body>
    <div id="result"></div>
    <script>
        //获取元素对象
        const result = document.getElementById("result");
        //绑定事件
        result.addEventListener("mouseover", function(){
            //1. 创建对象
            const xhr = new XMLHttpRequest();
            //2. 初始化 设置类型与 URL
            xhr.open('POST', 'http://127.0.0.1:8000/server');
            //设置请求头
```

```
            xhr.setRequestHeader('Content-Type','application/x-www-form-urlencoded');
            xhr.setRequestHeader('name','atguigu');
            //3. 发送
            xhr.send('a=100&b=200&c=300');
            // xhr.send('a:100&b:200&c:300');
            // xhr.send('1233211234567');

            //4. 事件绑定
            xhr.onreadystatechange = function(){
                //判断
                if(xhr.readyState === 4){
                    if(xhr.status >= 200 && xhr.status < 300){
                        //处理服务端返回的结果
                        result.innerHTML = xhr.response;
                    }
                }
            }
        });
    </script>
</body>
</html>
```

JSON

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>JSON响应</title>
    <style>
        #result{
            width:200px;
            height:100px;
            border:solid 1px #89b;
        }
    </style>
</head>
<body>
    <div id="result"></div>
    <script>
        const result = document.getElementById('result');
        //绑定键盘按下事件
        window.onkeydown = function(){
            //发送请求
            const xhr = new XMLHttpRequest();
            //设置响应体数据的类型
            xhr.responseType = 'json';
            //初始化
            xhr.open('GET','http://127.0.0.1:8000/json-server');
            //发送
            xhr.send();
            //事件绑定
            xhr.onreadystatechange = function(){
                if(xhr.readyState === 4){
```

```
                    if(xhr.status ≥ 200 && xhr.status < 300){
                        //
                        // console.log(xhr.response);
                        // result.innerHTML = xhr.response;
                        // 1. 手动对数据转化
                        // let data = JSON.parse(xhr.response);
                        // console.log(data);
                        // result.innerHTML = data.name;
                        // 2. 自动转换
                        console.log(xhr.response);
                        result.innerHTML = xhr.response.name;
                    }
                }
            }
        }
    </script>
</body>
</html>
```

第 2 章：jQuery 中的 AJAX

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>jQuery 发送 AJAX 请求</title>
    <link crossorigin="anonymous" href="https://cdn.bootcss.com/twitter-bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">
    <script crossorigin="anonymous" src="https://cdn.bootcdn.net/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
</head>
<body>
    <div class="container">
        <h2 class="page-header">jQuery发送AJAX请求 </h2>
        <button class="btn btn-primary">GET</button>
        <button class="btn btn-danger">POST</button>
        <button class="btn btn-info">通用型方法ajax</button>
    </div>
    <script>
        $('button').eq(0).click(function(){
            $.get('http://127.0.0.1:8000/jquery-server', {a:100, b:200}, function(data){
                console.log(data);
            },'json');
        });

        $('button').eq(1).click(function(){
            $.post('http://127.0.0.1:8000/jquery-server', {a:100, b:200}, function(data){
                console.log(data);
            });
        });
```

```
29
30          $('button').eq(2).click(function(){
31              $.ajax({
32                  //url
33                  url: 'http://127.0.0.1:8000/jquery-server',
34                  //参数
35                  data: {a:100, b:200},
36                  //请求类型
37                  type: 'GET',
38                  //响应体结果
39                  dataType: 'json',
40                  //成功的回调
41                  success: function(data){
42                      console.log(data);
43                  },
44                  //超时时间
45                  timeout: 2000,
46                  //失败的回调
47                  error: function(){
48                      console.log('出错啦!!');
49                  },
50                  //头信息
51                  headers: {
52                      c:300,
53                      d:400
54                  }
55              });
56          });
57
58      </script>
59  </body>
60  </html>
```

2.1 get 请求

```
1   $.get(url, [data], [callback], [type])
```

- url:请求的 URL 地址。
- data:请求携带的参数。
- callback:载入成功时回调函数。
- type:设置返回内容格式，xml, html, script, json, text, _default。

2.2 post 请求

```
1   $.post(url, [data], [callback], [type])
```

- url:请求的 URL 地址。
- data:请求携带的参数。
- callback:载入成功时回调函数。
- type:设置返回内容格式，xml, html, script, json, text, _default。

第 3 章：Axios 函数发送 Ajax 请求 （使用较多）

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>axios 发送 AJAX请求</title>
    <script crossorigin="anonymous"
src="https://cdn.bootcdn.net/ajax/libs/axios/0.19.2/axios.js"></script>
</head>

<body>
    <button>GET</button>
    <button>POST</button>
    <button>AJAX</button>

    <script>
        // https://github.com/axios/axios
        const btns = document.querySelectorAll('button');

        // 配置 baseURL
        axios.defaults.baseURL = 'http://127.0.0.1:8000';

        btns[0].onclick = function () {
            // GET 请求
            axios.get('/axios-server', {
                // url 参数
                params: {
                    id: 100,
                    vip: 7
                },
                // 请求头信息
                headers: {
                    name: 'atguigu',
                    age: 20
                }
            }).then(value => {
                console.log(value);
            });
        }

        btns[1].onclick = function () {
            axios.post('/axios-server', {
                username: 'admin',
                password: 'admin'
            }, {
                // url
                params: {
                    id: 200,
                    vip: 9
                },
                // 请求头参数
                headers: {
                    height: 180,
                    weight: 180,
                }
```

```
            });
        }

        btns[2].onclick = function(){
            axios({
                // 请求方法
                method : 'POST',
                // url
                url: '/axios-server',
                // url参数
                params: {
                    vip:10,
                    level:30
                },
                // 头信息
                headers: {
                    a:100,
                    b:200
                },
                // 请求体参数
                data: {
                    username: 'admin',
                    password: 'admin'
                }
            }).then(response => {
                // 响应状态码
                console.log(response.status);
                // 响应状态字符串
                console.log(response.statusText);
                // 响应头信息
                console.log(response.headers);
                // 响应体
                console.log(response.data);
            })
        }
    </script>
</body>

</html>
```

第 4 章：fetch 函数发送 Ajax 请求 (使用较少)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>fetch 发送 AJAX请求</title>
</head>
<body>
    <button>AJAX请求</button>
    <script>
        // 文档地址
```

```
        // https://developer.mozilla.org/zh-CN/docs/Web/API/WindowOrWorkerGlobalScope/fetch

        const btn = document.querySelector('button');

        btn.onclick = function(){
            fetch('http://127.0.0.1:8000/fetch-server?vip=10', {
                // 请求方法
                method: 'POST',
                // 请求头
                headers: {
                    name:'atguigu'
                },
                // 请求体
                body: 'username=admin&password=admin'
            }).then(response => {
                // return response.text();
                return response.json();
            }).then(response => {
                console.log(response);
            });
        }
    </script>
</body>
</html>
```

第 5 章：跨域

5.1 同源策略

同源策略(Same-Origin Policy)最早由 Netscape 公司提出，是浏览器的一种安全策略。

同源： 协议、域名、端口号 必须完全相同 。

违背同源策略就是跨域 。

5.2 如何解决跨域

5.2.1 JSONP

1. JSONP 是什么?
    - JSONP(JSON with Padding)，是一个非官方的跨域解决方案，纯粹凭借程序员的聪明才智开发出来， 只支持 get 请求 。
2. JSONP 怎么工作的?
    - 在网页有一些标签天生具有跨域能力，比如： img、link、iframe、script 。
    - JSONP 就是利用 script 标签的跨域能力来发送请求的。
3. JSONP 的使用：
    1. 动态的创建一个 script 标签

```
var script = document.createElement("script");
```

    2. 设置 script 的 src，设置回调函数

```
script.src = "http://localhost:3000/testAJAX?callback=abc";
function abc(data) {
    alert(data.name);
};
```

3. 将 script 添加到 body 中

```
document.body.appendChild(script);
```

4. 服务器中路由的处理

```
router.get("/testAJAX" , function (req , res) {
    console.log("收到请求");
    var callback = req.query.callback;
    var obj = {
        name:"孙悟空", age:18
    }
    res.send(callback+"("+JSON.stringify(obj)+")");
});
```

4. jQuery 中的 JSONP：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <button id="btn">按钮</button>
    <ul id="list"></ul>
    <script type="text/javascript" src="./jquery-1.12.3.js"></script>
    <script type="text/javascript">
    window.onload = function () {
        var btn = document.getElementById('btn')
        btn.onclick = function () {
            $.getJSON("http://api.douban.com/v2/movie/in_theaters?
callback=?",function(data) {
                console.log(data);
                //获取所有的电影的条目
                var subjects = data.subjects;
                //遍历电影条目
                for(var i=0 ; i<subjects.length ; i++){
                    $("#list").append("<li>"+
                    subjects[i].title+"<br />"+
                    "<img src=\""+subjects[i].images.large+"\" >"+
                    "</li>");
                }
            });
        }
    }
    </script>
</body>
</html>
```

案例：原生 JSONP 检测用户名是否存在

index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>案例</title>
</head>
<body>
    用户名: <input type="text" id="username">
    <p></p>
    <script>
        //获取 input 元素
        const input = document.querySelector('input');
        const p = document.querySelector('p');

        //声明 handle 函数
        function handle(data){
            input.style.border = "solid 1px #f00";
            //修改 p 标签的提示文本
            p.innerHTML = data.msg;
        }

        //绑定事件
        input.onblur = function(){
            //获取用户的输入值
            let username = this.value;
            //向服务器端发送请求 检测用户名是否存在
            //1. 创建 script 标签
            const script = document.createElement('script');
            //2. 设置标签的 src 属性
            script.src = 'http://127.0.0.1:8000/check-username';
            //3. 将 script 插入到文档中
            document.body.appendChild(script);
        }
    </script>
</body>
</html>
```

serve.js

```js
// 引入express
const express = require('express');

// 创建应用对象
const app = express();

//用户名检测是否存在
app.all('/check-username',(request, response) => {
    // response.send('console.log("hello jsonp")');
    const data = {
        exist: 1,
        msg: '用户名已经存在'
```

```
    };
    //将数据转化为字符串
    let str = JSON.stringify(data);
    //返回结果
    response.end(`handle(${str})`);
});

// 监听端口启动服务
app.listen(8000, () => {
    console.log("服务已经启动, 8000 端口监听中....");
});
```

jQuery 发送 JSONP 请求

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>jQuery-jsonp</title>
    <style>
        #result{
            width:300px;
            height:100px;
            border:solid 1px #089;
        }
    </style>
    <script crossorigin="anonymous"
src='https://cdn.bootcss.com/jquery/3.5.0/jquery.min.js'></script>
</head>
<body>
    <button>点击发送 jsonp 请求</button>
    <div id="result">

    </div>
    <script>
        $('button').eq(0).click(function(){
            $.getJSON('http://127.0.0.1:8000/jquery-jsonp-server?callback=?',
function(data){
                $('#result').html(`
                    名称: ${data.name}<br>
                    校区: ${data.city}
                `)
            });
        });
    </script>
</body>
</html>
```

serve.js

```
//1. 引入express
const express = require('express');

```

```
//2. 创建应用对象
const app = express();

//
app.all('/jquery-jsonp-server',(request, response) => {
    // response.send('console.log("hello jsonp")');
    const data = {
        name:'尚硅谷',
        city: ['北京','上海','深圳']
    };
    //将数据转化为字符串
    let str = JSON.stringify(data);
    //接收 callback 参数
    let cb = request.query.callback;

    //返回结果
    response.end(`${cb}(${str})`);
});

//4. 监听端口启动服务
app.listen(8000, () => {
    console.log("服务已经启动, 8000 端口监听中....");
});
```

5.2.2 CORS

https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Access_control_CORS

1. CORS 是什么?
    - CORS（Cross-Origin Resource Sharing），跨域资源共享。CORS 是官方的跨域解决方案，它的特点是不需要在客户端做任何特殊的操作，完全在服务器中进行处理，支持 get 和 post 请求 。跨域资源共享标准新增了一组 HTTP 首部字段，允许服务器声明哪些源站通过浏览器有权限访问哪些资源。
2. CORS 怎么工作的?
    - CORS 是通过设置一个响应头来告诉浏览器，该请求允许跨域，浏览器收到该响应以后就会对响应放行。
3. CORS 的使用
    - 主要是服务器端的设置:

```
router.get("/testAJAX" , function (req , res) {
    //通过 res 来设置响应头，来允许跨域请求
    //res.set("Access-Control-Allow-Origin","http://127.0.0.1:3000");
    res.set("Access-Control-Allow-Origin","*");
    res.send("testAJAX 返回的响应");
});
```

案例：设置 CORS 响应头实现跨域

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>CORS</title>
    <style>
        #result{
            width:200px;
            height:100px;
            border:solid 1px #90b;
        }
    </style>
</head>
<body>
    <button>发送请求</button>
    <div id="result"></div>
    <script>
        const btn = document.querySelector('button');

        btn.onclick = function(){
            //1. 创建对象
            const x = new XMLHttpRequest();
            //2. 初始化设置
            x.open("GET", "http://127.0.0.1:8000/cors-server");
            //3. 发送
            x.send();
            //4. 绑定事件
            x.onreadystatechange = function(){
                if(x.readyState === 4){
                    if(x.status >= 200 && x.status < 300){
                        //输出响应体
                        console.log(x.response);
                    }
                }
            }
        }
    </script>
</body>
</html>
```

serve.js

```
//1. 引入express
const express = require('express');

//2. 创建应用对象
const app = express();

//3. 创建路由规则
app.all('/cors-server', (request, response)=>{
    //设置响应头
    response.setHeader("Access-Control-Allow-Origin", "*");
    response.setHeader("Access-Control-Allow-Headers", '*');
    response.setHeader("Access-Control-Allow-Method", '*');
    // response.setHeader("Access-Control-Allow-Origin",
"http://127.0.0.1:5500");
    response.send('hello CORS');
});

```

```
17  //4. 监听端口启动服务
18  app.listen(8000, () => {
19      console.log("服务已经启动, 8000 端口监听中....");
20  });
```