Ajax

1. 原生 Ajax

1.1. Ajax 简介

Ajax 全称 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>
```

- 现在在 Ajax 中都使用 JSON，而不使用 XML

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
2. 存在跨域问题
3. SEO 不友好

1.4. HTTP

HTTP（hypertext transportprotocol）协议（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互相通信的规则。

1.4.1. 请求报文

重点是格式与参数

```
行        请求方法    url路径    http协议版本
头        Host: atguigu.com
          Cookie: name=guigu
          Content-Type: application/x-www-form-urlencoded
          User-Agent: chrome 83
空行
体        (GET请求，请求体是空的；如果是POST请求的话，请求体可以不为空)
```

POST 请求体例子：

```
username=admin&password=admin
```

请求报文例子：

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

1.4.2. 响应报文

```
行        协议版本(HTTP/1.1)  响应状态码(200)  响应状态字符串(OK)
头        Content-Type: text/html;charset=utf-8
          Content-length: 2048
          Content-encoding: gzip
空行
体        <html>
              <head>
              </head>
              <body>
                  <h1>小赤佬</h1>
              </body>
          </html?
```

1.5. Ajax 操作的基本步骤

```
// 获取 button 元素
const btn = document.getElementByTagName('button')[0]
//获取 div(result) 元素
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...的时候
    //   readystate 是 xhr 对象中的属性，表示状态 有 0(未初始化)、1(open方法调用完毕)、2(send方法调用完毕)、3(服务端返回了部分结果)、4(服务端返回了全部结果) 这几个状态
    //   change 改变
    xhr.onreadystatechange = function () {
        // 判断（4 表示服务端返回了所有结果）
        if (xhr.readystate === 4) {
            // 判断响应状态码 200 400 403 401 500
            // 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.getAllResponseHeader())   // 所有响应头
                // console.log(xhr.response)   // 响应体

                // 设置 result 的内容
                result.innerHTML = xhr.response
            } else {
                //
            }
        }
    }
}
```

1.5.1. 设置请求参数

直接在 url 后面加 ?a=100&b=200&c=300

```
    xhr.open('GET', 'http://127.0.0.1:8000/server?a=100&b=200&c=300')
```

1.5.2. POST 请求传递参数

在 xhr.send() 中直接设置

```
xhr.send('a=100&b=200&c=300')
xhr.send('a:100&b:200:c=300')
// 以上两种方法都可以
```

1.5.3. 设置请求头信息

在 xhr.open() 后面

```
// 初始化 设置类型与 url
xhr.open('POST', 'http://127.0.0.1:3000/server')
// 设置请求头
//xhr.setRequestHeader('请求头参数', '请求头值')
xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded')
```

1.5.4. 服务端响应 JSON 数据

1. 手动转换

```
// 服务端
const data = {
    name: 'xiaochilao'
}

// 对对象进行字符串转换
let str = JSON.stringify(data)
res.send(str)
```

```
// 客户端
xhr.onreadystatechange = function () {
        if (xhr.readystate === 4) {
            if (xhr.status >=200 && xhr.status < 300) {
                let data = JSON.parse(xhr.response)
                result.innerHTML = data.name
            }
        }
    }
```

2. 自动转换

```
// 客户端
xhr.responseType = 'json'

...

xhr.onreadystatechange = function () {
        if (xhr.readystate === 4) {
            if (xhr.status >=200 && xhr.status < 300) {
                result.innerHTML = xhr.response.name
            }
        }
    }
```

1.6. Ajax - IE缓存问题

```
xhr.open('GET', 'http://127.0.0.1:3000/ie?t=' + Date.now())
```

1.7. Ajax 请求超时与网络异常处理

```
const xhr = new XMLHttpRequest()
// 超时设置 2s
xhr.timeout = 2000
// 超时回调
xhr.ontimeout = function () {
    alert('网络异常，请稍后重试')
}
// 网络异常回调
xhr.onerror = function () {
    alert('你的网络似乎出了一些问题！')
}
```

1.8. Ajax 取消请求

```
xhr.abort()  // ajax 对象的 abort 方法可以取消请求
```

1.9. Ajax 请求重复发送问题

```
const btns = document.querySelectorAll('button')
let xhr = null

// 标识变量
let isSending = false  // 是否正在发送AJAX请求
btn[0].onclick = function () {
    // 判断标识变量
    if(isSending) x.bort()  // 如果正在发送，则取消该请求，创建一个新的请求

    xhr = new XMLHttpRequest()
    // 修改标识变量的值
    isSending = true
    xhr.open('GET', 'http://127.0.0.1:3000/delay')
    xhr.send();
    xhr.onreadystatechange = function () {
        if (x.readystate === 4) {
            // 修改标识变量
            isSending = false
        }
    }
}
```

---

2. jQuery 中的 Ajax

2.1. get 请求

```
$.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。

```
$('button').eq(0).click(function () {
    $.get('http://127.0.0.1:3000/jquery-server', {a:100, b:200}, function (data) {
        console.log(data)
    })
}, 'json')
```

2.2. post 请求

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

- url：请求的地址
- 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。

```
$('button').eq(1).click(function () {
    $.post('http://127.0.0.1:3000/jquery-server', {a:100, b:200}, function (data) {
        console.log(data)
    })
})
```

2.3. jQuery 通用方法发送 Ajax 请求

> 参考文档：https://jquery.cuishifeng.cn/jQuery.Ajax.html

```
$('button').eq(2).click(function () {
    $.ajax({
        // url
        url: 'http://127.0.0.1:3000/delay',
        // 参数
        data: {a:100, b:200},
        // 请求类型
        type: 'GET',
        // 响应体结果类型
        dataType: 'json',
        // 成功的回调
        success: function (data) {
            console.log(data)
        },
        // 超时时间
        timeout: 2000,
        // 失败的回调
        error: function () {
            console.log('出错啦！！')
        },
        header: {
            c: 300,
            d: 400
        }
    })
}
```

2.4. 资源跨源请求属性设置

- crossorigin="anonymous"

```
<link crossorigin="anonymous" href="https://cdn.bootcss.com/tritter-bootstrap/3.3.7/css/bootstrap.css" />
 <script crossorigin="anonymous" src="https://cdn.bootcdn.net/ajax/libs/jquery/3.5.1/jquery.js"></script>
```

2.5. 服务端设置允许自定义请求头和设置允许跨域

```
// 服务端
app.all('/delay', function (req, res) {
    // 设置响应头

    res.setHeader('Access-Control-Allow-Origin', '*')  // 设置允许跨域
    res.setHeader('Access-Control-Allow-Header', '*'')  // 设置允许自定义请求头信息
})
```

3. Axios 发送 Ajax 请求

1. 下载

```
npm install axios
```

2. 使用

- GET 请求

```
// 先通过 script 引入

const btns = document.querySelectorAll('button')

// 配置 baseURL
axios.default.baseURL = 'http://127.0.0.1:3000'

btn[0].onclick = function () {
    axios.get('/axios-server', {
        // url 参数
        params: {
            id: 100,
            vip: 7,
        },
         // 请求头信息
        headers: {
        name: 'xiaochilao',
        age: 20,
        }
    }).then(value => {
        // 返回一个对象
        console.log(value)
    })
```

```
}
```

- POST 请求

```
btn[1].onclick = function () {
    // 第二个参数是请求体
    axios.post('/axios-server', {
            username: 'admin',
            password: 'admin'
        },
        {
        // url 参数
        params: {
            id: 200,
            vip: 9,
        },
         // 请求头信息
        headers: {
            height: 180,
            weight: 180,
        },
    })
}
```

Axios 通用方式发请求

```
btn[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)
```

```
    })
}
```

4. 使用 fetch 函数发送 Ajax 请求

fetch 函数属于全局对象，可以直接调用

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

btn.onclick = function () {
    // url 参数可以直接在 url 后面接着写 传入
    fetch('http://127.0.0.1:3000/fetch-server?vip=10', {
        // 请求方法
        method: 'POST',
        // 请求头
        headers: {
            name: 'xiaochilao'
        },
        // 请求体
        body: 'username=admin&password=admin'
    }).then(response => {
        // 获取服务端返回结果，如果服务端返回的是 json 格式的数据，则调用 response.json() 获取
        // return response.text();
        return response.json();
    }).then(response => {
        console.log(response)
    })
}
```

5. 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，纯粹是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

   - 动态的创建一个 script 标签

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

   - 设置 script 的 src，设置回调函数

     `script.src = 'http//localhost:3000/testAJAX?callback=abc'`

5.2.1.1. 原生 JSONP 实践

- 检测用户名是否存在并变化 input 输入框样式

```
// 客户端代码

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
    scirpt.src = 'http://127.0.0.1:3000/check-username'
    // 3. 将 script  插入到文档中
    document.body.appendChild(script)
}
```

```
// 服务端代码 （node）
app.all('/check-username', (req, res) => {
    const data = {
        exist: 1,
        msg: '用户名已经存在',
    }
    // 将数据转化为 json 格式字符串
    let str = JSON.stringify(data)
    // 返回结果（响应）
    res.end(`handle(${str})`)
})
```

5.2.1.2. jQuery 发送 JSONP 请求

```
// 客户端
$('button').eq(0).click(function () {
    $.getJSON('http://127.0.0.1:3000/jquery-jsonp-server?callback=?',
function (data) {
        $('#result').html(`
            名称：  ${data.name},
            校区：  ${data.msg}
        `)
    })
})
```

```
// 服务端
app.all('/jquery-jsonp-server', (req, res) => {
    const data = {
        exist: 1,
        msg: '用户名已经存在',
    }
    // 将数据转化为 json 格式字符串
    let str = JSON.stringify(data)
    // 接收 callback 参数
    let cb = req.query.callback

    // 返回结果（响应）
    res.end(`${cb}(${str})`)
})
```

5.2.2. CORS 解决跨域问题

> 参考：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向访问哪些资源

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：

```
app.all('/cors-server', (req, res) => {
    // 设置响应头  允许跨域
    res.setHeader('Access-Control-Allow-Origin', '*')
    // 设置响应头  允许客户端发送任何响应头（包括自定义的响应头）
    res.setHeader('Access-Control-Allow-Headers', '*')
    // 设置响应头  客户端发送任何方法的请求
    res.setHeader('Access-Control-Allow-Methods', '*')
})
```