AJAX 课程

尚硅谷前端研究院

版本：V 1.0

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 不友好

1.4 AJAX 的使用

HTTP协议

HTTP协议：超文本传输协议，协议详细的规定了浏览器和万维网服务器之间互相通信的规则。

请求报文：

行 ：POST /s?ie=utf-8(url地址) HTTP/1.1（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 （必不可少）

体 ：GET请求无，POST请求有（username=admin&password=admin

响应报文：

行 ：HTTP/1.1(HTTP协议版本) 200(响应状态码) OK(响应状态字符串)

头 ：Content-Type:text/html;charset=utf-8

Content-length:2048

Content-encoding:gzip
空行
体 ：响应内容

Express框架的基本使用：

```JavaScript
// 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
// 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("Helllo Express");
})

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

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

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


xhr.onreadystatechange = function (){
    if(xhr.readyState == 4 && xhr.status == 200){
        var text = xhr.responseText;
        console.log(text);
    }
}
```

Ajax发送GET请求：

服务器：

```
// 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
// request是对请求报文的封装
// response是对响应报文的封装
app.get('/', (request, response) => {
  // 设置响应头，设置允许跨域
  response.setHeader('Access-Control-Allow-Origin','*')
  // 设置响应体
  response.send("Helllo Ajax");
})

// 4.监听端口启动服务
app.listen(3000, () => {
  console.log("服务已启动，3000端口监听中...");
})
```

GET请求：

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

```
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Ajax GET 请求</title>
    <style>
      #result {
        width: 200px;
        height: 100px;
        border: 1px solid #ed7beb;
      }
    </style>
  </head>
  <body>
    <button>点击GET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
        xhr.open('GET', 'http://127.0.0.1:3000/?a=100&b=200')
        // 3.发送
        xhr.send()
        // 4.事件绑定，处理服务端返回的结果
        xhr.onreadystatechange = function () {
          if (xhr.readyState === 4) {
            // 判断响应状态码
            if (xhr.status >= 200 && xhr.status < 300) {
              // 处理结果
              // 行 头 空行 体
              // 1.响应行
              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
            }
          }
        }
      }
    </script>
  </body>
</html>
```

POST请求：

```
<!DOCTYPE html>
```

```html
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Ajax POST 请求</title>
    <style>
      #result {
        width: 200px;
        height: 100px;
        border: 1px solid #aaa;
      }
    </style>
  </head>
  <body>
    <div id="result"></div>

    <script>
      const result = document.getElementById('result')
      // 绑定事件
      result.addEventListener('mouseover', function () {
        // 1.创建对象
        const xhr = new XMLHttpRequest()
        // 2.设置初始化
        xhr.open('POST', 'http://127.0.0.1:3000/')
        // 3.发送
        xhr.send('a=100&b=200&c=300')
        // 4.事件绑定
        xhr.onreadystatechange = function () {
          if (xhr.readyState === 4) {
            if (xhr.status >= 200 && xhr.status < 300) {
              // 处理返回结果
              result.innerHTML = xhr.response
            }
          }
        }
      })
    </script>
  </body>
</html>
```

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

Ajax设置请求头信息：

```
// 设置请求头
        // 设置请求体内容类型
        xhr.setRequestHeader(
          'Content-Type',
          'application/x-www-form-urlencoded'
        )
```

设置响应头信息：

```
// 设置响应头，设置允许跨域
  response.setHeader('Access-Control-Allow-Origin', '*')
  response.setHeader('Access-Control-Allow-Headers','*')
```

json数据转换

```
app.get('/', (request, response) => {
  // 设置响应头，设置允许跨域
  response.setHeader('Access-Control-Allow-Origin', '*')
  response.setHeader('Access-Control-Allow-Headers','*')
  // 响应一个数据
  const data = {
    name: 'guigu',
    age:20
  }
  // 对对象进行字符串转换
  let str = JSON.stringify(data);
  // 设置响应体
  response.send(str);
})
```

JSON.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <style>
      #result {
```

```
        width: 200px;
        height: 100px;
        border: 1px solid #aaa;
      }
    </style>
  </head>
  <body>
    <div id="result"></div>

    <script>
      const result = document.getElementById('result')
      window.onkeydown = function () {
        // 发送请求
        const xhr = new XMLHttpRequest()
        // 设置响应体自动转换类型
        xhr.responseType = 'json'
        xhr.open('GET', 'http://127.0.0.1:3000/')
        xhr.send()
        xhr.onreadystatechange = function () {
          if (xhr.readyState === 4) {
            if (xhr.status >= 200 && xhr.status < 300) {
              // 手动对响应数据进行转换
              let resultText = JSON.parse(xhr.response)
              console.log(resultText)
              // 处理返回结果
              result.innerHTML = resultText.name
            }
          }
        }
      }
    </script>
  </body>
</html>
```

ajax请求超时与网络异常处理：

server.js

```
app.get('/long', (request, response) => {
   // 设置响应头，设置允许跨域
   response.setHeader('Access-Control-Allow-Origin', '*')
   response.setHeader('Access-Control-Allow-Headers','*')
   // 响应一个数据
   const data = {
     name: 'wyb',
     age:24
   }
   // 对对象进行字符串转换
  let str = JSON.stringify(data);
  setTimeout(() => {
   // 设置响应体
   response.send(str+'延时响应');
  },3000)
})
```

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Ajax GET 请求</title>
    <style>
      #result {
        width: 200px;
        height: 100px;
        border: 1px solid #ed7beb;
      }
    </style>
  </head>
  <body>
    <button>点击GET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()
        // 超时设置
        xhr.timeout = 2000
        // 超时回调
        xhr.ontimeout = function () {
          alert('网络异常，请稍后重试！')
        }
        // 网络异常的回调
        xhr.onerror = function () {
          alert('你的网络似乎出了问题！！')
        }

        // 2.初始化
        xhr.open('GET', 'http://127.0.0.1:3000/long')
        // 3.发送
        xhr.send()
        // 4.事件绑定，处理服务端返回的结果
        xhr.onreadystatechange = function () {
          if (xhr.readyState === 4) {
            // 判断响应状态码
            if (xhr.status >= 200 && xhr.status < 300) {
              // 处理结果
              // 行 头 空行 体
              // 1.响应行
              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
          }
        }
      }
    }
  </script>
</body>
</html>
```

ajax取消请求：

```
<script>
    // 发送请求
    const btns = document.getElementsByTagName('button')
    let xhr = null
    btns[0].onclick = function () {
      xhr = new XMLHttpRequest()
      xhr.open('GET', 'http://127.0.0.1:3000/long')
      xhr.send()
    }

    // 取消请求
    btns[1].onclick = function () {
      // abort()方法可以取消发送ajax请求
      xhr.abort()
    }
  </script>
```

ajax请求重复发送问题：

```
<script>
    // 发送请求
    const btns = document.getElementsByTagName('button')
    let xhr = null
    // 标识变量：是否正在发送Ajax请求
    let isSending = false
    btns[0].onclick = function () {
      if (isSending) {
        // 取消上一次的请求
        xhr.abort()
      }
      xhr = new XMLHttpRequest()
      // 修改标识状态
      isSending = true
      xhr.open('GET', 'http://127.0.0.1:3000/long')
      xhr.send()
      xhr.onreadystatechange = function () {
        if (xhr.readyState === 4) {
          // 修改标识状态
          isSending = false
        }
      }
    }
  </script>
```

第 2 章：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。

2.2 post 请求

$.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。

```
<script>
      $(function () {
        window.onload = function () {
          $.ajax({
            // url
            url: 'http://127.0.0.1:3000/long',
            // 参数
            data: { a: 100, b: 200 },
            // 请求类型
            type: 'GET',
            // 响应体结果
            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 (err) {
              console.log(err)
            },
            // 头信息
            headers: {
              c: 300,
              b: 400,
            },
```

```
        })
      }
    })
  </script>
```

axios发送ajax请求：

和jQuery的不同之处：处理数据jQuery采用回调函数，而axios采用的Promise。

发送POST请求和GET请求：

```
axios.defaults.baseURL = 'http://127.0.0.1:3000'
    window.onload = function () {
      axios
        .get('/', {
          // url参数
          params: {
            id: 100,
            vip: 7,
          },
          // 请求头信息
          Headers: {
            name: 'wyb',
            age: 24,
          },
          // GET请求无请求体
        })
        .then((value) => {
          console.log(value.data)
        })

      axios.post(
        '/',
        // 请求体
        { username: 'wyb', password: '127085' },
        {
          // url
          params: {
            id: 200,
            vip: 85,
          },
          // 请求头参数
          Headers: {
            height: 187,
            weight: 62,
          },
        }
      )
    }
```

通用方式：

```
axios({
          // 请求方法
          method: 'POST',
          // url
          url: '/',
          // url参数
          params: {
            vip: 27,
            level: 85,
          },
          // 头信息
          headers: {
            a: 100,
            n: 200,
          },
          // 请求体
          data: {
            username: 'wyb',
            password: 127085,
          },
        }).then((response) => {
          console.log(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)
        })
```

使用fetch函数发送ajax请求

fetch函数用于发起获取资源的请求，返回的是一个Promise，这个promise会在请求响应之后被resolve，并传回response对象。

当遇到网络错误时，fetch()返回的promise会被reject，并传回TypeError。

参数1：url地址

参数2：对象形式；包括：methods（方法）、headers（请求头信息）、body（请求体信息）等。

```
<script>
      const btn = document.querySelector('button')
      btn.onclick = function () {
        fetch('http://127.0.0.1:3000/?vip=27&level=85', {
          // 请求方法
          method: 'POST',
          // 请求头
          headers: {
            name: 'wyb',
          },
          // 请求体
          body: 'username=admin&password=127085',
        })
          .then((response) => {
            // return response.text()
            return response.json()
          })
          .then((value) => {
            console.log(value)
          })
      }
    </script>
```

第 3 章：跨域

3.1 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。

Server.js

```
const { request, response } = require('express');
const express = require('express')

const app = express();

app.get('/home', (request, response) => {
  // 响应一个页面
  response.sendFile(__dirname + '/index.html');
})
app.get('/data', (request, response) => {
  response.send('用户数据')
})
app.listen(9000, () => {
  console.log("服务器启动了...");
})
```

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <h1>Hello 跨域</h1>
    <button>点击获取用户数据</button>

    <script>
      const btn = document.querySelector('button')
      btn.addEventListener('click', function () {
        const xhr = new XMLHttpRequest()
        // 满足同源策略，url可简写
        xhr.open('GET', '/data')
        xhr.send()
        xhr.onreadystatechange = function () {
          if (xhr.readyState === 4) {
            if (xhr.status >= 200 && xhr.status < 300) {
              console.log(xhr.response)
            }
          }
        }
      })
    </script>
  </body>
</html>
```

3.2 如何解决跨域

3.2.1 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

原理：返回函数的调用

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>原理演示</title>
    <style>
      #result {
        width: 300px;
        height: 100px;
        border: 1px solid rgb(153, 177, 227);
      }
    </style>
  </head>
  <body>
    <div id="result"></div>
    <script>
      // 处理数据
      function handle(data) {
        // 获取result元素
        const result = document.querySelector('#result')
        result.innerHTML = data.name
      }
    </script>
    <!-- <script src="http://127.0.0.1:5500/04-hello_react/JSONP/js/app.js"></script> -->
    <script src="http://127.0.0.1:7000/jsonp-server"></script>
  </body>
</html>
```

```
const { request, response } = require('express');
const express = require('express')

const app = express();

app.get('/home', (request, response) => {
  // 响应一个页面
  response.sendFile(__dirname + '/index.html');
})
app.get('/data', (request, response) => {
  response.send('用户数据')
})
```

```
app.get('/jsonp-server', (request, response) => {
  const data = {
    name:'冰雨火'
  }
  // 将数据转化为字符串
  let str = JSON.stringify(data)
  response.end(`handle(${str})`)
})
app.listen(7000, () => {
  console.log("服务器启动了...");
})
```

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.stri ngify(obj)+")");
});
```

原生JSONP的使用：

```
// 用户名检测是否存在
app.get('/check-username', (request, response) => {
  const data = {
    exist: 1,
    msg:'用户名已存在'
  }
  // 将数据转为字符串
  let str = JSON.stringify(data)
  response.end(`handle(${str})`);
})
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <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('#username')
      const p = document.querySelector('p')
      // 声明handle函数
      function handle(data) {
        input.style.border = '1px solid pink'
        // p标签提示文本
        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://127.0.0.1:7000/check-username'
        // 3.将script插入到文档中
        document.body.appendChild(script)
      }
    </script>
  </body>
</html>
```

4) jQuery 中的 JSONP

```
app.all('/jquery-jsonp-server', (request, response) => {
  const data = {
    name: '冰雨火',
    plat:['优酷','芒果','腾讯','爱奇艺']
  }
  let str = JSON.stringify(data)
  // 接收callback参数
  let cb = request.query.callback;
  response.end(`${cb}(${str})`)
})
```

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

```
<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <script
    crossorigin="anonymous"
    src="https://cdn.bootcdn.net/ajax/libs/jquery/3.6.0/jquery.min.js"
  ></script>
  <title>Document</title>
  <style>
    #result {
      width: 300px;
      height: 200px;
      border: 1px solid hotpink;
    }
  </style>
</head>
<body>
  <button>点击发送jsonp请求</button>
  <div id="result"></div>

  <script>
    $(function () {
      $('button').click(function () {
        // ?callback=?是jQuery发送jsonp请求的固定写法
        $.getJSON(
          'http://127.0.0.1:7000/jquery-jsonp-server?callback=?',
          function (data) {
            $('#result').html(`名称：${data.name}<br/>平台：${data.plat}`)
          }
        )
      })
    })
  </script>
</body>
</html>
```

3.2.2 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");
});
```

```
app.post('/', (request, response) => {
  // 设置响应头，设置允许跨域
  // 允许所有网页进行跨域请求
  response.setHeader('Access-Control-Allow-Origin', '*')
  // 允许固定页面进行跨域请求
  response.setHeader('Access-Control-Allow-Origin','http://127.0.0.1:7000')
  // 响应一个数据
  const data = {
    name: 'guigu',
    age:20
  }
  // 对对象进行字符串转换
  let str = JSON.stringify(data);
  // 设置响应体
  response.send(str);
})
```

更多响应头的设置，参考官方文档！