AJAX

1. AJAX 简介

1. 什么是 AJAX?
   - AJAX 是一种在网页中使用的技术，它允许浏览器与服务器进行**异步通信**，从而在**不刷新整个页面**的情况下**更新页面的部分内容**
   - AJAX 全称为 "Asynchronous JavaScript And XML"，但实际上，**XML 并不是必须的**，因为数据的传输格式可以是 XML、JSON 或其他形式
   - AJAX 并不是一种新的编程语言，而是一种**技术或方法**，利用了现有的标准组合在一起使用
   - AJAX 主要使用 JavaScript 和 XMLHttpRequest 对象来实现**浏览器与服务器的通信**
2. 什么是 XML?
   - XML（可扩展标记语言）是一种被设计用来**传输和存储数据**的**标记语言**
   - XML 是一种文本格式，通过使用**自定义标签**来描述数据的结构和含义
   - XML Vs. HTML
     - HTML 是一种用于创建网页结构的标记语言，它的标签是**预定义的**，用于表示页面中的各种元素（如标题、段落、链接等）
     - XML 中的标签则**没有预定义**，用户可以根据自己的需求创建**自定义**的标签和数据结构，用来表示任何类型的数据
   - AJAX 中的 XML 这种描述数据的方式**已经被 JSON 取代**
   - 示例: 表示一个学生数据（XML 和 JSON 实现）

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

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

3. AJAX 的优缺点
    - 优点
        - **无需刷新页面而与服务器端进行通信**
        - **根据用户事件来更新部分页面内容**
    - 缺点
        - **没有浏览历史，不能回退**（因为 AJAX 动态加载内容时不会改变浏览器的 URL）
        - **存在跨域问题（同源策略）**：AJAX 的请求受同源策略的限制，即只能向与当前页面具有相同协议、域名和端口的服务器发送请求，因此跨域请求需要通过其他方式来处理
        - **SEO 不友好**：搜索引擎爬虫通常不会执行 JavaScript，因此如果网站的内容依赖于 AJAX 动态加载，那么这部分内容可能无法被搜索引擎索引，从而影响网站的 SEO 排名

2. 原生发送 AJAX 请求

2.1 基本使用

1. 创建 XMLHttpRequest 对象 `const xhr = new XMLHttpRequest();`
2. 设置请求报文
    - 请求行 `xhr.open(请求方式, URL)`
    - 请求头 `xhr.setRequestHeader(键, 值)`
    - 请求体
        - 对于 GET 请求，在 open 方法中通过 URL 的查询字符串传递参数
        - 对于 POST 请求，在 send 方法中可通过任意方式传递参数
3. 发送请求 `xhr.send()`
4. 处理响应报文
    - 响应行
        - `xhr.status` 响应状态码
        - `xhr.statusText` 响应状态信息
    - 响应头
        - `xhr.getAllResponseHeaders()` 响应头
    - 响应体
        - `xhr.response` 响应体
        - `xhr.responseXML` xml 格式的响应体
        - `xhr.responseText` 文本格式的响应体

2.2 几个注意点

1. 什么时候处理响应报文?
   - 条件：`xhr.readyState === 4 && (xhr.status >= 200 && xhr.status < 300)`
   - 前者条件解释：服务端是否返回了全部结果
     - `readyState` 是 xhr 对象的属性，取值 0、1、2、3、4 表示不同的状态
       - 0 表示 XMLHttpRequest 实例已经生成，但是 open 方法还没有被调用
       - 1 表示 open 方法调用完毕，此时 send 方法还没有被调用
       - 2 表示 send 方法调用完毕
       - 3 表示正在接收服务器传来的部分数据
       - 4 表示已经接收服务器传来的全部数据 or 本次数据接受失败
     - `readystatechange` 事件共计会触发四次，我们选择 readyState = 4 时再对响应结果进行处理
   - 后者条件解释：响应是否属于成功响应（响应状态码是否为 2xx）
2. 什么是预检请求?
   - 当使用非简单请求（例如带有自定义头部或使用非 GET、POST、HEAD 方法的请求）进行跨域请求时，浏览器会先发送一个 `OPTIONS` 请求给目标服务器，以**确定是否允许实际请求的跨域访问**。
   - 服务器会在响应中包含 `CORS` 相关的**头部信息**，如 Access-Control-Allow-Methods（允许的请求方法）、Access-Control-Allow-Headers（允许的请求头）、Access-Control-Allow-Origin（允许的来源域）等，以便浏览器判断**是否可以继续发送实际请求**。
   - 因此，`OPTIONS` 请求在跨域请求中起到了**确定是否允许跨域访问**的作用，并且可以**设置允许的自定义请求头**。
3. 怎么响应预检请求?

```
res.setHeader('Access-Control-Allow-Origin', '*'); // 设置允许跨域
res.setHeader('Access-Control-Allow-Headers', '*'); // 设置允许自定义的响应头
```

2.3 JSON 响应

1. 方式一
   - 服务器返回 JSON 对象字符串 `JSON.stringify(JSON 对象) => JSON 字符串`，`res.send(JSON 字符串)`
   - 然后浏览器将字符串解析为 JSON 对象 `JSON.parse(JSON 字符串) => JSON 对象(此时 response 类型为 json)`
2. 方式二
   - 服务器返回 JSON 对象字符串
   - 然后浏览器设置响应体类型为 json `xhr.responseType = 'json'(此时 response 类型为 json)`
3. 注：服务器也可以直接用 `res.json(JSON 对象)` 的方式返回 JSON 对象字符串

2.4 IE 缓存问题

2.5 请求超时与网络异常

2.6 取消请求

2.7 重复请求问题

3. jQuery 发送 AJAX 请求

- GET 请求 `$.get(请求 URL，请求参数对象，回调函数[，响应类型])`
- POST 请求 `$.post(请求 URL，请求参数对象，回调函数[，响应类型])`
- 自定义请求

```
$.ajax({
    url: 'xxx', // 表示请求 URL
    data: {xxx}, // 表示请求参数对象
    type: 'xxx', // 表示请求类型，如 GET、POST 等
    dataType: 'xxx', // 表示服务端返回的内容类型的字符串，如 'json' 等
    success: (res) => {}, // 回调函数，AJAX 请求成功得到响应后该回调函数自动调用，接收一个参数 res，表示服务端返回的内容
    timeout: xxx, // 表示最长请求时间，单位为毫秒
    error: () => {}, // 回调函数，请求超时或网络异常时自动调用，
    headers: {xxx} // 表示请求头对象
})
```

4. axios 发送 AJAX 请求

- GET 请求

```
axios.get(请求 URL, {
    params: {xxx}, // 表示 URL 参数
    headers: {xxx}, // 表示请求头
}).then(res => {}) // 当 AJAX 请求成功得到响应后自动调用 then 中的回调函数，接收一个参数 res，表示封装的响应报文对象
```

- POST 请求

```
axios.post(请求 URL, requestBody, { // 其中 requestBody 表示设置请求体内容
    params: {xxx}, // 表示 URL 参数
    headers: {}, // 表示请求头
}).then(res => {}) // 当 AJAX 请求成功得到响应后自动调用 then 中的回调函数，接收一个参数 res，表示封装的响应报文对象
```

- 自定义请求

```
axios({
    method: 'xxx', // 表示请求类型，如 GET、POST 等
    url: 'xxx', // 表示请求 URL
    params: {xxx}, // 表示 URL 参数
    headers: {xxx}, // 表示请求头
    data: {xxx} // 表示请求体
 }).then(res => {}) // 当 AJAX 请求成功得到响应后自动调用 then 中的回调函数，接收一个参数 res，表示封装的响应报文对象
```

5. fetch 发送给 AJAX 请求

```
fetch(请求 URL, {
  method: 'xxx', // 请求方法，如 'GET', 'POST' 等
  headers: {xxx}, // 请求头
  body: 'xxx' 请求体内容
}).then(res => { // 对请求返回的响应进行处理的部分，用于解析响应体中的 JSON 数据
  return res.text();
}).then(res => { // 处理上一个 .then() 方法返回的结果
  xxx // 这里的 res 才是响应体内容
})
```

6. 同源策略与跨域请求

1. 同源策略：即 Same-Origin Policy，由 Netscape 公司提出，是浏览器的一种**安全**策略，这里的**同源**指的是**协议、域名、端口号必须完全相同**。
2. 跨域：即**违背同源策略**的行为，如 AJAX 发送请求默认需要遵守同源策略，但很多时候需要向其他服务器发出请求，此时就需要跨域。
3. 跨域请求的两种实现方式
    - 非官方策略 jsonp
    - 官方策略 cors

7. 跨域策略之 jsonp

1. 什么是 JSONP?
    - JSONP，即 JSON with Padding，是一个**非官方**的跨域解决方案，只支持 GET 请求
    - 因为网页中有一些标签**天生具有跨域的能力**，如 `img`、`link`、`iframe`、`script`，而 JSONP 就是利用 `script` 标签的跨域能力来发送请求的
2. 原生实现 jsonp
3. jQuery 实现 jsonp

8. 跨域策略之 cors

1. 什么是 CORS?
    - CORS，即 Cross-Origin Resource Sharing，跨域资源共享
    - CORS 是**官方**提供的跨域解决方案，其特点是不需要在客户端进行任何特殊的操作，完全在**服务器中进行处理**
    - CORS 支持 GET 和 POST 请求
    - CORS 标准新增了一组 HTTP 首部字段，允许服务器声明**哪些资源允许被哪些源站所访问**

- CORS 是通过**设置响应头**来告诉浏览器该请求允许跨域，浏览器在收到该响应后，以后就会对该响应放行

2. 常用响应头举例

```
response.setHeader("Access-Control-Allow-Origin", "*"); // 允许跨域
response.setHeader("Access-Control-Allow-Headers", '*'); // 允许自定义头
response.setHeader("Access-Control-Allow-Method", '*'); // 允许所有方法
```

跳转到顶部