Axios网络请求

教学内容

- 第一节 Axios的使用
- 第二节 与Vue整合
- 第三节 跨域

Axios简介

- 在实际项目开发中，前端页面所需要的数据往往需要从服务器端获取，这必然涉及与服务器的通信。
- Axios 是一个基于 promise 网络请求库，作用于node.js 和浏览器中。
- Axios 在浏览器端使用XMLHttpRequests发送网络请求，并能自动完成JSON数据的转换 。
- 安装：npm install axios
- 地址：https://www.axios-http.cn/

发送网络请求

- 发送GET请求。

```
// 向给定ID的用户发起请求
axios.get('/user?ID=12345')
  .then(function (response) {
    // 处理成功情况
    console.log(response);
  })
  .catch(function (error) {
    // 处理错误情况
    console.log(error);
  })
  .then(function () {
    // 总是会执行
  });
```

```
// 上述请求也可以按以下方式完成（可选）
axios.get('/user', {
    params: {
      ID: 12345
    }
  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  })
  .then(function () {
    // 总是会执行
  });
```

发送网络请求

- 发送POST请求。

```
axios.post('/user', {
    firstName: 'Fred',
    lastName: 'Flintstone'
  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });
```

异步回调问题

- async/await

```
// 支持async/await用法
async function getUser() {
  try {
    const response = await axios.get('/user?ID=12345');
    console.log(response);
  } catch (error) {
    console.error(error);
  }
}
```

其他请求方式

- 参考：https://axios-http.com/zh/docs/req_config

```
// 发起一个post请求
axios({
  method: 'post',
  url: '/user/12345',
  data: {
    firstName: 'Fred',
    lastName: 'Flintstone'
  }
});
```

axios.get(url[, config])

axios.delete(url[, config])

axios.head(url[, config])

axios.options(url[, config])

axios.post(url[, data[, config]])

axios.put(url[, data[, config]])

axios.patch(url[, data[, config]])

与Vue整合

- 在实际项目开发中，几乎每个组件中都会用到 axios 发起数据请求。此时会遇到如下两个问题：
- 每个组件中都需要导入 axios
- 每次发请求都需要填写完整的请求路径
- 可以通过全局配置的方式解决上述问题：

```
// 配置请求根路径
axios.defaults.baseURL = 'http://api.com'

// 将axios作为全局的自定义属性，每个组件可以在内部直接访问 (Vue3)
app.config.globalProperties.$http = axios

// 将axios作为全局的自定义属性，每个组件可以在内部直接访问 (Vue2)
Vue.prototype.$http = axios
```

为什么会出现跨域问题

- 为了保证浏览器的安全，不同源的客户端脚本在没有明确授权的情况下，不能读写对方资源，称为同源策略，同源策略是浏览器安全的基石
- 同源策略（Sameoriginpolicy）是一种约定，它是浏览器最核心也最基本的安全功能
- 所谓同源（即指在同一个域）就是两个页面具有相同的协议（protocol），主机（host）和端口号（port）
- 当一个请求url的协议、域名、端口三者之间任意一个与当前页面url不同即为跨域，此时无法读取非同源网页的 Cookie，无法向非同源地址发送 AJAX 请求

跨域问题解决

- CORS（Cross-Origin Resource Sharing）是由W3C制定的一种跨域资源共享技术标准，其目的就是为了解决前端的跨域请求。
- CORS可以在不破坏即有规则的情况下，通过后端服务器实现CORS接口，从而实现跨域通信。
- CORS将请求分为两类：简单请求和非简单请求，分别对跨域通信提供了支持。

简单请求

满足以下条件的请求即为简单请求：

- 请求方法：GET、POST、HEAD
- 除了以下的请求头字段之外，没有自定义的请求头：
- Accept、Accept-Language、Content-Language、Last-Event-ID、Content-Type
- Content-Type的值只有以下三种：
- text/plain、multipart/form-data、application/x-www-form-urlencoded

简单请求的服务器处理

- 对于简单请求，CORS的策略是请求时在请求头中增加一个Origin字段，

```
Host: localhost:8080
Origin: http://localhost:8081
Referer: http://localhost:8081/index.html
```

- 服务器收到请求后，根据该字段判断是否允许该请求访问，如果允许，则在HTTP头信息中添加Access-Control-Allow-Origin字段。

```
Access-Control-Allow-Origin: http://localhost:8081
Content-Length: 20
Content-Type: text/plain;charset=UTF-8
Date: Thu, 12 Jul 2018 12:51:14 GMT
```

非简单请求

- 对于非简单请求的跨源请求，浏览器会在真实请求发出前增加一次OPTION请求，称为预检请求（preflight request）
- 预检请求将真实请求的信息，包括请求方法、自定义头字段、源信息添加到HTTP头信息字段中，询问服务器是否允许这样的操作。
- 例如一个GET请求：

```
OPTIONS /test HTTP/1.1
Origin: http://www.test.com
Access-Control-Request-Method: GET
Access-Control-Request-Headers: X-Custom-Header
Host: www.test.com
```

- Access-Control-Request-Method表示请求使用的HTTP方法，Access-Control-Request-Headers包含请求的自定义头字段

非简单请求

- 服务器收到请求时，需要分别对Origin、Access-Control-Request-Method、Access-Control-Request-Headers进行验证，验证通过后，会在返回HTTP头信息中添加：

```
Access-Control-Allow-Origin: http://www.test.com
Access-Control-Allow-Methods: GET, POST, PUT, DELETE
Access-Control-Allow-Headers: X-Custom-Header
Access-Control-Allow-Credentials: true
Access-Control-Max-Age: 1728000
```

- Access-Control-Allow-Methods、Access-Control-Allow-Headers：真实请求允许的方法、允许使用的字段
- Access-Control-Allow-Credentials: 是否允许用户发送、处理cookie
- Access-Control-Max-Age: 预检请求的有效期，单位为秒，有效期内不会重复发送预检请求。

非简单请求

- 当预检请求通过后，浏览器才会发送真实请求到服务器。这样就实现了跨域资源的请求访问。

Spring Boot中配置CORS

- 在传统的Java EE开发中，可以通过过滤器统一配置，而Spring Boot中对此则提供了更加简洁的解决方案

```
@Configuration
public class CorsConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")   // 允许跨域访问的路径
        .allowedOrigins("*") // 允许跨域访问的源
        .allowedMethods("POST", "GET", "PUT", "OPTIONS", "DELETE")  // 允许请求方法
        .maxAge(168000)  // 预检间隔时间
        .allowedHeaders("*")  // 允许头部设置
        .allowCredentials(true); // 是否发送 cookie
    }

}
```