axiso 学习笔记

一、axiso 说明

- axios 是一个基于 promise 的 HTTP 库，可以用在浏览器和 node.js 中。
- axios 既可以发送请求，也可以接收请求。
- 当 axios 使用在浏览器中时，是通过 XMLHttpRequests 来发送请求和接收请求的。
- 当 axios 使用在 node.js 中时，是通过 http 来发送请求和接收请求的。
- 支持 promise API
- 拦截请求和响应
- 转换请求数据和响应数据
- 取消请求
- 自动转换 JSON 数据
- 客户端支持防御 XSRF，跨站请求伪造攻击漏洞

二、axiso 安装引用

- 使用 npm 安装

```
 $ npm i axios
```

- 直接 script 标签引用

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

- ES6 import 引用

```
import axios from 'axios'
axios.get()
```

三、axiso 使用

- 使用 GET 请求

```
//为给定ID的user创建请求
axiso.get('/user?ID=12345')
.then(function (res) {
    console.log(res)
})
.catch(function (err) {
    console.log(err)
})
//可选地，上面的请求可以这样做
axios.get('/user', {
    params: {
        ID: 12345,
    },
```

```
})
.then(function (res) {
    console.log(res)
})
.catch(function (err) {})
```

- 使用 POST 请求

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

- 合并请求，.then 以后使用 axios.spread 方法分割请求的数据，一一对应。

```
function getUserAccount() {
    return axios.get('/user/12345')
}
function getUserPermissions() {
    return axios.get('/user/12345/permissions')
}
axios.all([getUserAccount(), getUserPermissions()]).then(
    axios.spread(function (acct, perms) {
        //两个请求现在都执行完成
    })
)
```

- 通过调用 axios 的 API，传递一个配置对象来发起请求
- axios(config)

```
axios({
    method: 'post',
    url: '/user/12345',
    data: {
        first: 1,
        last: 2,
    },
}).then((res) => {
    console.log('请求结果：', res)
})
```

- axios 为所有请求方式都提供了别名

```
-   axios.request(config)
-   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]])
```

- 创建一个 axios 实例 axios.create([config])

```
var instance = axios.create({
    baseUrl: 'https://some-domain.com/api/',
    timeout: 1000,
    headers: { 'X-Custom-Header': 'foobar' },
})
```

四、axios 库

- post 请求方法：通过传递 url 和 data 请求，创建 new XMLHttpRequest()完成请求

```
post(url, data={}) {
    /**/
    let xhr = new XMLHttpRequest();
}
```

- 考虑到 get 请求也会有同样的处理，统一封装一个 request 请求方法，请求的参数通过配置 config 进行传递

```
request(config) {

}
```

- request(config) 参考的是 axios 的一个默认配置项，config 传递进去后，与默认配置项进行合并。不同类型的属性采用不同的合并方式，例如：url 采用的是覆盖合并，header 采用的是新增合并。axios 提供 mergeConfig 对配置项进行了合并处理。

- 配置会以一个优先顺序进行合并。这个顺序是：在 lib/defaults.js 找到库的默认值，然后是实例的 defaults 属性，最后是请求的 config 参数。后者将优先于前者。

```
//使用由库提供的配置的默认值来创建实例
//此时超时配置的默认值是0
var instance = axios.create()
//覆写库的超时默认值
//现在，在超时前，所有的请求都会等待2.5秒
instance.defaults.timeout = 2500
//为已知需要花费很长时间的请求覆写超时设置
instance.get('/url', {
    timeout: 5000,
})
```

- 合并配置之后，通过适配器 Adapter 进行请求的处理
- 通过 getAdapter() 判断是否浏览器环境，发送 XMLHttpRequest 和 http 请求

```
function getAdapter(config) {
    // 如果是浏览器环境
    if (typeof XMLHttpRequest !== 'undefined') {
        return xhrAdapter(config);
    } else {
        return httpAdapter(config);
    }
}
```

- XMLHttpRequest 请求实现，针对 post 参数的处理，data 来进行 json 转换

```
function xhrAdapter(config) {
    return new Promise((resove, reject) => {
        let xhr = new XMLHttpRequest();
        xhr.onload = function() {
            let data = JSON.parse(xhr.responseText);
            resove(data);
        }
        let fd = new FormData();
        for (let key in config.data) {
            fd.append(key, config.data[key]);
        }
        xhr.open(config.method, config.url, true);
        xhr.send(fd);
        return xhr;
    });
}
```

- Interceptor 拦截器，主要用于在请求或响应被 then 或 catch 处理前拦截它们
- 根据 Promise 返回状态 fulfilled 和 rejected 判断请求成功还是失败，将状态保存在 handlers 数组中进行事件管理

```
use(fulfilled, rejected) {
    this.handlers.push({
        fulfilled,
        rejected
    });
}
```

- 在通过 getAdapter() 发起请求时，需求对其进行包装，防止请求立马执行。

```
let chain = [function(config) {
    let adapter = getAdapter( config );
    return adapter;
}, undefined];
```

axiso