前后端交互03

- Fetct基本用法
  - Request 对象 Headers 对象 Response对象
  - text方法
  - json方法
- Fetch 与 XMLHttpRequest 的差异
- axios 的使用
- 简版axios实现
  - 拦截器实现
  - 混入模式
  - 适配器实现

Fetch

- 基本用法

```
fetch("/test").then(res=>{
    return res.json();
}).then(res=>{
    console.log(res);
}).catch(err=>{
    console.log(err);
})
```

- 参数
  - 第一个参数是请求的url
  - 第二个参数 options对象 常用配置
    - `method`: 请求使用的方法，如 `GET、``POST。`
    - `headers`: 请求的头信息，形式为 `Headers` 的对象或包含 `ByteString` 值的对象字面量。
    - `body`: 请求的 body 信息：可能是一个 `Blob`、`BufferSource`、`FormData`、`URLSearchParams` 或者 `USVString` 对象。注意 GET 或 HEAD 方法的请求不能包含 body 信息。
    - `mode`: 请求的模式，如 `cors、` `no-cors` 或者 `same-origin。`

- `same-origin`表示必须同源，绝对禁止跨域，这个是老版本浏览器默认的安全策略。
- `no-cors`这个就很特殊了，字面意思是禁止以CORS的形式跨域，其实它的效果是，对外域的请求可以发送，外域服务器无论设不设`Access-Control-Allow-Origin: *`都会接收请求并处理请求，但是浏览器不接收响应，即使外域返回了内容，浏览器也当做没接到

Headers对象

你可以通过 `Headers()` 构造函数来创建一个你自己的 headers 对象。

```
var myHeaders = new Headers();
myHeaders.append("Content-Type", "text/plain");
myHeaders.append("Content-Length", content.length.toString());
myHeaders.append("X-Custom-Header", "ProcessThisImmediately")
```

```
myHeaders = new Headers({
  "Content-Type": "text/plain",
  "Content-Length": content.length.toString(),
  "X-Custom-Header": "ProcessThisImmediately",
});
```

Response对象

Response.clone()

创建一个Response对象的克隆

Response.json()

读取 Response对象并且将它设置为已读（因为Responses对象被设置为了 stream 的方式，所以它们只能被读取一次），并返回一个被解析为JSON格式的promise对象

Response.text()

读取 Response对象并且将它设置为已读（因为Responses对象被设置为了 stream 的方式，所以它们只能被读取一次），并返回一个被解析为USVString格式的promise对象

Fetch 与 XMLHttpRequest 的差异

- fetch不能监控进度
- fetch兼容性

axios库

一、axios的主要函数

- 定义
- 调用

二、针对url的一些处理

- 工具类util

```
let util =  {}
```

三、post参数及请求头的处理

- 针对post参数的处理
-
  - data来进行json转换
  - 设置默认头部

四、返还对象的包装

- 返还对象的形式
- 针对返还对象的处理

五、工厂模式函数和混入模式

- axios的使用方式
  - 方式一

```
axios({})
```

- 方式二

```
axios.get({})
```

- 方式三

```
axios.create({})
```

- 工厂模式函数和混入模式
- 继承函数

```
    extends(a, b, thisArg) {
        //只拷贝循环对象自身方法
        for (let key in b) {
            if (b.hasOwnProperty(key)) {
                let val = b[key];
                // console.log("val值",val);
                if (thisArg && typeof val === 'function') {
                    a[key] = val.bind(thisArg);
                } else {
                    a[key] = val;
                }
            }
        }
        return a;
    }
```

六、拦截器

- 拆分dispathXhr触发ajax方法；

- 拦截管理器

```
class InterceptorManager {
    constructor() {
        this.handlers = [];
    }
    use(fulfilled, rejected) {
        this.handlers.push({
            fulfilled: fulfilled,
            rejected: rejected
        });
    }
}
```

- 任务链执行对应函数 (注意this指向)

七、Adapters适配器；

- 在触发dispathXhr函数中判断axios运行的环境

```
 //环境判断  和发送网络请求
        if (typeof process !== 'undefined' && Object.prototype.toString.call(process)
=== '[object process]') {
            return this.adapters.http(this.instanceConfig);
        } else if (typeof XMLHttpRequest !== 'undefined') {
            return this.adapters.xhr(this.instanceConfig);
        }
```

- Adapters中对应不同的处理

```
http(config) {
        this.config = config;
        return new Promise((resolve, reject) => {
            const http = require("http");
            const urls = require("url");
            let { data = null, url, method = 'get', params, headers = {} } =
this.config;
            let pathObj = urls.parse(url)
            let options = {
                host: pathObj.hostname,
                port: pathObj.port,
                path: pathObj.path,
                method: this.config.method.toUpperCase(),
                headers: headers
            };
            // let options = {
            //     host:'localhost',
            //     port:3000,
            //     path:'/atest',
            //     method:'POST',
            //     headers:{
            //         "content-type":"application/json"
            //     }
            // }
            // console.log(options);
            let request = http.request(options, res => {
                let reslut = "";
                res.on("data", chunk => {
```

```
                reslut += chunk;
            })
            res.on("end", () => {
                console.log(reslut.toString())
                resolve(JSON.parse(reslut.toString()));
            })
        })
        request.on("error", err => {
            reject(err);
        })
        request.end();
    })
}
```

总结

- Fetct基本用法
  - Request 对象 Headers 对象 Response对象
  - text方法
  - json方法
- Fetch 与 XMLHttpRequest 的差异
- axios 的使用
- 简版axios实现
  - 拦截器实现
  - 混入模式
  - 适配器实现