**读懂了Axios源码，你就读懂了Promise**

Axios是基于Promise机制实现的异步的链式请求框架。Promise 在异步中的地位，如同生活水电一般存在。以前对于异步基本都是基于回调函数，现在应该思考使用 promise 去处理。

1. 学习心得
2. 拦截器的实现
3. 取消发送请求

**取消发送请求：**

请求取消：（两个案例）

1. 在一个输入框中输入内容，请求后端，通常前端会做一个防抖比如200ms，但是有时可能会出现这样一种特殊情况，先发送的请求时间比后发送的请求，相应的时间更长，这时如果不能及时取消先发送的请求，那么就会导致先发送的请求结果会替换当前内容的结果。
2. tab页之间的频繁切换，也可能出现上述情况，先发送的请求比后发送的请求响应时间更长，导致页面展示不正确，这时，应该取消先前发送的请求

利用promise实现异步分离，也就是在cancelToken中保存一个 pending 状态的 Promise 对象，然后当我们执行 cancel 方法的时候，能够访问到这个 Promise 对象， 把它从pending状态变成resolved状态，这样我们就可以在then函数中实现取消请求的逻辑



执行取消函数，就是让token的promise的状态变成了成功；



Promise 是一个容器，里面保存着未来才会结束的事件

**拦截器：**

request.interceptors.request.use() // 最后执行

request.interceptors.request.use()

request.interceptors.request.use()



request.interceptors.response.use()

request.interceptors.response.use()

request.interceptors.response.use() // 最后执行



// 遍历实例对象的请求拦截器

this.interceptors.request.handlers.forEach(interceptor=>{

// 将请求拦截器压入数组的最前面

chain.unshift(interceptor.fulfilled,interceptor.rejected)

})

// 遍历实例对象的响应拦截器

this.interceptors.response.handlers.forEach(interceptor=>{

// 将请求拦截器压入数组的最后面

chain.push(interceptor.fulfilled,interceptor.rejected);

})

**学习心得：**

单一职责原则

工厂方法设计模式

适配器设计模式

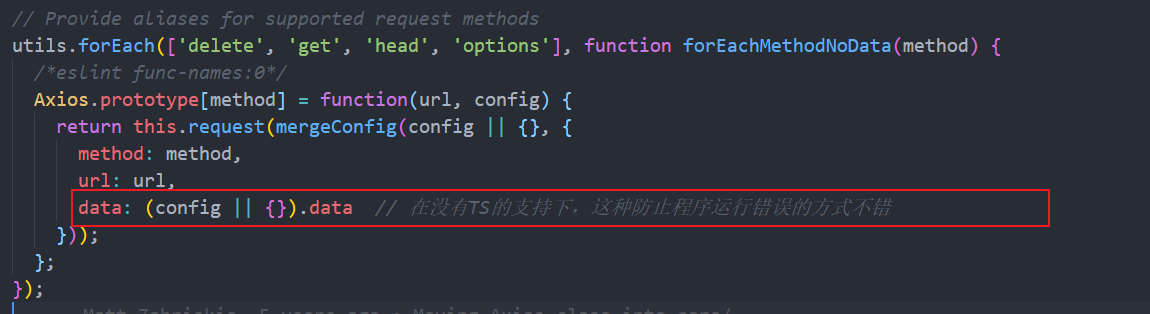
目录结构清晰

// 注释是一种迫不得已的行为，axios 源码就可以看出来，只在必要和关键的地方添加了注释，其它的部分好的变量名已经覆盖了，当然这是一个优秀的程序员才能做到的，毕竟我们大部分时间和精力在业务逻辑上，除非对这些你已经了然于胸，其次是不断的修改和完善。

写注释的形式，函数，变量名起的，没毛病



在没有TS的支持下，这种防止程序运行错误的方式不错



写地址的时候，我不知道大家有没有纠结过下面这种情况：

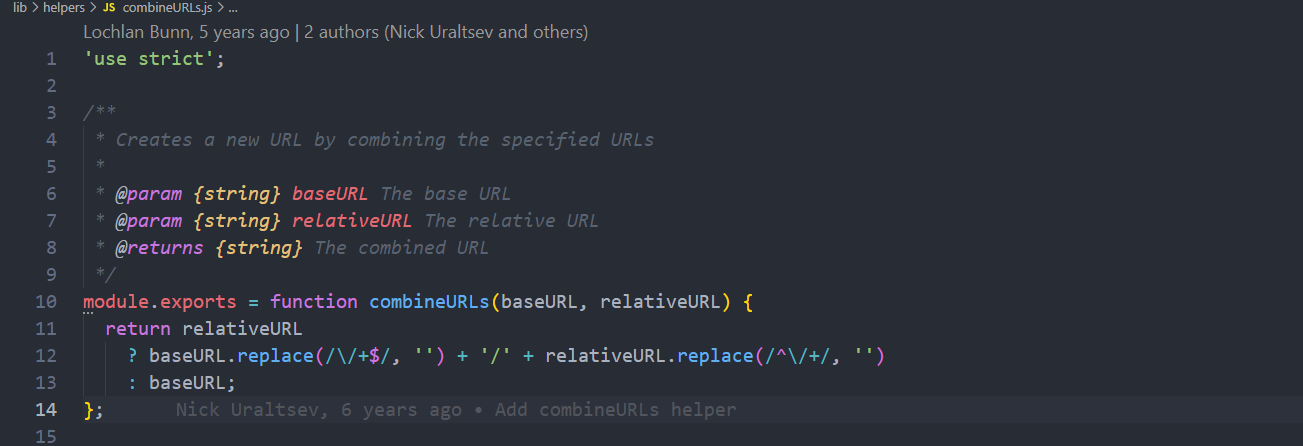
BaseUrl = ‘https://www.baidu.com/’

BaseUrl = ‘https://www.baidu.com’

在写相对url的时候

relativeUrl = ‘/get/list’

relativeUrl = ‘get/list’



一个叫做getUri的实例方法，该方法用于在不发送请求的前提下根据传入的请求配置对象返回一个请求的url

