0. 学习文档

```
https://juejin.im/post/5b0ba2d56fb9a00a1357a334
```

1. 源码目录结构

├── /dist/ # 项目输出目录 ├── /lib/ # 项目源码目录 │ ├── /adapters/ # 定义请求的适配器 xhr、http │ │ ├── http.js # 实现http适配器(包装http包) │ │ └── xhr.js # 实现xhr适配器(包装xhr对象) │ ├── /cancel/ # 定义取消功能 │ ├── /core/ # 一些核心功能 │ │ ├── Axios.js # axios的核心主类 │ │ ├── dispatchRequest.js # 用来调用http请求适配器方法发送请求的函数 │ │ ├── InterceptorManager.js # 拦截器的管理器 │ │ └── settle.js # 根据http响应状态，改变Promise的状态 │ ├── /helpers/ # 一些辅助方法 │ ├── axios.js # 对外暴露接口 │ ├── defaults.js # axios的默认配置 │ └── utils.js # 公用工具 ├── package.json # 项目信息 ├── index.d.ts # 配置TypeScript的声明文件 └── index.js # 入口文件

2. 源码分析

1). axios与Axios的关系

2). axios为什么能有多种发请求的方法?

```
axios函数对应的是Axios.prototype.request方法通过bind(Axiox的实例)产生的函数
axios有Axios原型上的所有发特定类型请求的方法: get()/post()/put()/delete()
axios有Axios的实例上的所有属性: defaults/interceptors
后面又添加了create()/CancelToken()/all()
```

3). axios.create()返回的对象与axios的区别?

```
相同:
    都是一个能发任意请求的函数: request(config)
    都有发特定请求的各种方法: get()/post()/put()/delete()
    都有默认配置和拦截器的属性: defaults/interceptors
不同:
    默认匹配的值很可能不一样
    instance没有axios后面添加的一引起方法: create()/CancelToken()/all()
```

4). axios运行的整体流程

5). Axios.prototype.request()都做了什么?

6). dispatchrequest()都做了什么?

7). xhrAdapter()做了什么?

```
整体流程: request(config)  ===> dispatchRequest(config) ===> xhrAdapter(config)
request(config): 将请求拦截器 / dispatchRequest() / 响应拦截器 通过promise链串连起来，返回promise
dispatchRequest(config): 转换请求数据 ===> 调用xhrAdapter()发请求 ===> 请求返回后转换响应数据. 返回promise
xhrAdapter(config): 创建XHR对象，根据config进行相应设置，发送特定请求，并接收响应数据，返回promise
```

8). axios的请求/响应拦截器是什么?

```
请求拦截器: 在真正发请求前，可以对请求进行检查或配置进行特定处理的函数，包括成功/失败的函数，传递的必须是config
响应拦截器: 在请求返回后，可以对响应数据进行特定处理的函数，包括成功/失败的函数，传递的默认是response
```

9). axios的请求/响应数据转换器是什么?

```
请求转换器: 对请求头和请求体数据进行特定处理的函数
    setContentTypeIfUnset(headers, 'application/json;charset=utf-8');
    return JSON.stringify(data)
响应转换器: 将响应体json字符串解析为js对象或数组的函数
    response.data = JSON.parse(response.data)
```

10). response的整体结构

```
{
    data,
    status,
    statusText,
    headers,
    config,
    request
}
```

11). error的整体结构

```
{
    message,
    request,
    response
}
```

12). 如何取消已经发送的请求?

1.当配置了cancelToken对象时，保存cancel函数
    创建一个用于将来中断请求的cancelPromise
    并定义了一个用于取消请求的cancel函数
    将cancel函数传递出来
2.调用cancel()取消请求
    执行cacel函数，传入错误信息message
    内部会让cancelPromise变为成功，且成功的值为一个Cancel对象
    在cancelPromise的成功回调中中断请求，并让发请求的proimse失败，失败的reason为Cacel对象