前端架构带你 封装axios，后端看了拍手叫绝

前言（为何做）

过去的一段时间，我都认为 **接口请求** 封装是前端的必修课。 只要是写过生产环境前端代码的人，应该都脱离不了异步接口请求，那么 接口请求 的 **封装** 是必经之路。

直到前些天，我们屋某个美团写后台的小姑娘问我前端问题时。我才发现她们代码中的 **接口请求** ，都是没有任何的封装，直接采用以下方式进行：

```
axios.post(`/api/xxxx/xxxx?xxx=${xxx}`, { ...data })
.then((result) => {
    if (result.errno) {
        ....
    } else {
        ....
    }
})
.catch((err) => {
    ....
})
```

这样写也不是说不好，在某种程度上，这增加了代码的可读性。
但是我们大多数页面需要的接口都不止一个，那么我们的组件中极有可能出现 **数十上百** 行重复代码。

那么随着请求的体量越来越大，我们的项目便越来越难以维护。

效果演示

```
const [e, r] = await api.getUserInfo(userid)
if (!e && r) this.userInfo = r.data.userinfo
```

上面是我们最终的实现效果。

接下来，我将带大家一步一步封装一套属于我们自己的 **接口请求工具** ，同时也希望大家分享更好的思路。

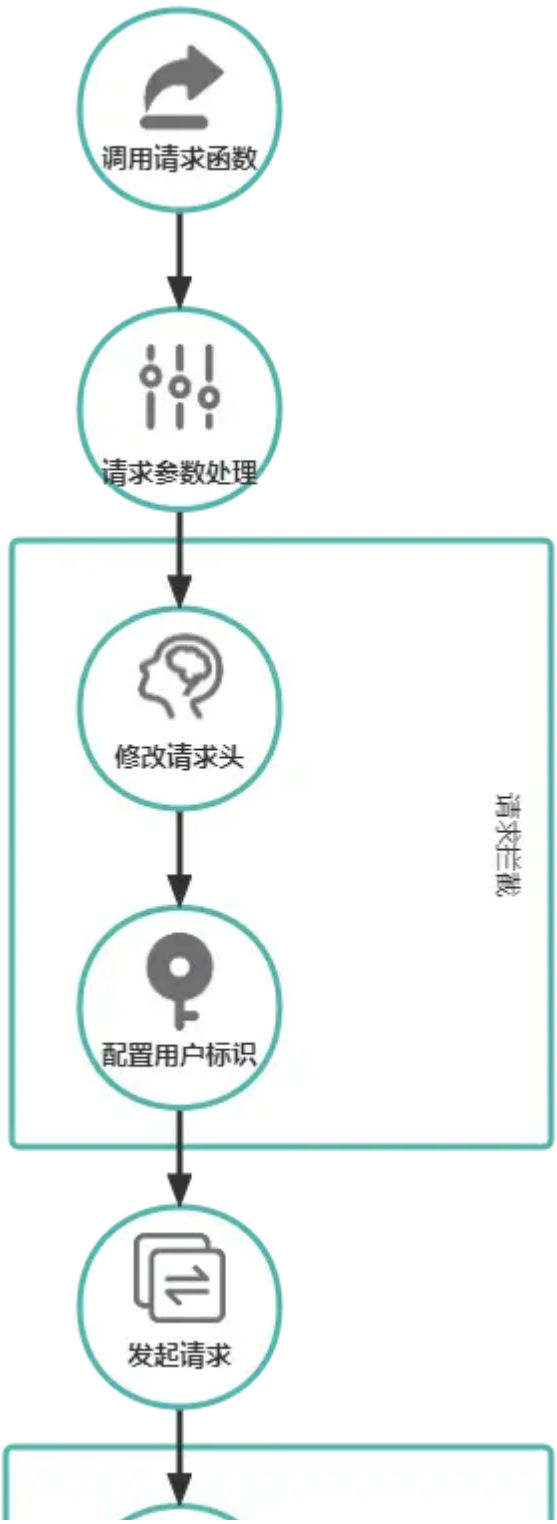
> 注：
>
> - 如果你希望直接看源码，请翻到 《完整代码》

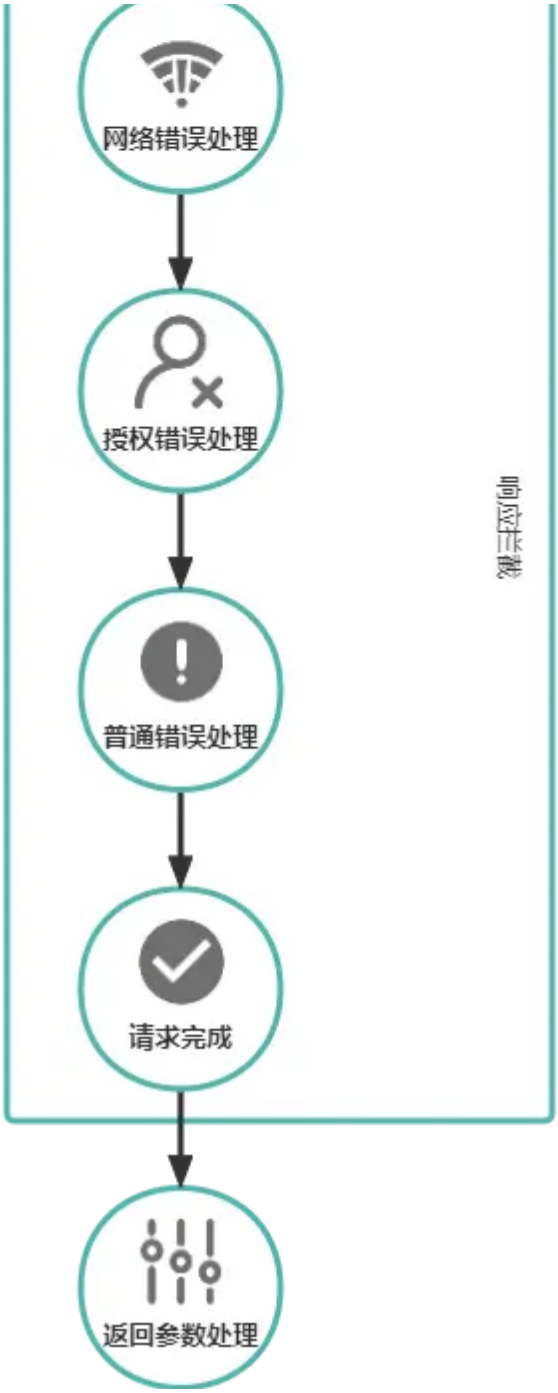
> - 这里以 `axios` 作示范，同样换成 `fetch` 、小程序的 `request` 都是可以的
> - 我将会采用 `typeScript` 书写这段教程，如果你不需要，忽略掉对应的类型即可

思路清晰，先说分析（做什么）

在我们正式开发前，首先需要清楚请求一个接口都做了什么。

为此，消耗了两个小时时间，做了一个请求流程图，以便于我们后续进行需求分析（小声bb：Processon真难用 😢）





有了一个清晰的请求流程图，我们便可以区分出来两块重要的内容来进行拆分： **基础请求流程** 、 **拦截器** 。

接下来我们将两块儿内容展开讲。

基础请求流程

基础请求流程，我们大致可以分为三块， 一是 **请求进入请求拦截前** 、二是 **真正发起的请求** 、三是 **请求从响应拦截出来后** 。

这其中可以归为两类，
一类是 **针对单独接口的处理**
二类是 **针对所有接口需要的内容**

- 针对单独接口的处理

    - 请求前的参数处理
    - 请求后的返回值处理
- 针对所有接口的处理
    - Post
    - Get
    - Put
    - Del

拦截器

拦截器，我们大致可以分为两类， 一类是 **请求接口前的统一处理（请求拦截）** 、 一类是 **请求接口后的统一处理（响应拦截）**

- 请求拦截
    - 请求调整
    - 用户标识
- 响应拦截
    - 网络错误处理
    - 授权错误处理
    - 普通错误处理
    - 代码异常处理

统一调用

随着我们的 `Api` 越来越多，我们可能需要给他们不同的分类，但我们并不希望每次调用都从不同的文件夹引入不同的 `Api` ，因此在 基础请求 + 拦截器 之外，我们还需要一个封包操作。

开发顺序

随着我们要做的内容越来越多，我们希望它有一个顺序以便于我们按部就班的开发（相信大家对开发中出现的不确定性都深恶痛绝）。

以便于我们按照流程，无意外、无惊喜 的完成此次封装。

在我们的开发中，我们基本要遵循先处理通用内容在处理个性化内容的逻辑：

1. 针对所有接口的处理（Get）
2. 请求拦截
3. 响应拦截
4. 针对单独接口的处理

5. 封包处理
6. 针对所有接口的处理（Post、Put、Del）

tips

💡 这里大家可能意外为什么 Post、Put、Del 的处理在最后开发： 因为大多数情况，我们开发中希望所编写的内容有一个及时的回馈。

举个栗子：我在生活中发现 → 我们学习吉他时，大多数人半途而废了。但坚持下来的人基本无一例外的通过吉他在不同的阶段都获得了好处，包括但不限于 **异性** 的夸奖、舍友的鼓掌、 **get女朋友** 。 这也是我们在毕业独处后，很难学会弹吉他的原因（无处炫耀）。

因此，我们需要让所开发的内容尽快达到可用的阶段（MVP）。

万事俱备、只欠东风（怎么做）

按照我们之前定好的顺序，按部就班的开发⑧！

针对所有接口的处理（Get）

我们希望以 `const [e, r] = await api.getUserInfo(id)` 的方式调用，代表着我们需要保证返回值稳定的返回 `[err, result]` ，所以我们需要在请求无论成功失败时，都以 `resolve` 方式调用。

同时，我们希望我们可以处理返回值，因此在这里封装了 `clearFn` 的回调函数。

```
type Fn = (data: FcResponse<any>) => unknown

interface IAnyObj {
  [index: string]: unknown
}

interface FcResponse<T> {
  errno: string
  errmsg: string
  data: T
}

const get = <T,>(url: string, params: IAnyObj = {}, clearFn?: Fn):
Promise<[any, FcResponse<T> | undefined]> =>
  new Promise((resolve) => {
    axios
      .get(url, { params })
      .then((result) => {
        let res: FcResponse<T>
        if (clearFn !== undefined) {
```

```
            res = clearFn(result.data) as unknown as FcResponse<T>
          } else {
            res = result.data as FcResponse<T>
          }
          resolve([null, res as FcResponse<T>])
        })
        .catch((err) => {
          resolve([err, undefined])
        })
    })
```

请求拦截

请求拦截中，我们需要两块内容，一是 **请求的调整** ，二是 **配置用户标识**

```
const handleRequestHeader = (config) => {
    config['xxxx'] = 'xxx'
    return config
}

const handleAuth = (config) => {
    config.header['token'] = localStorage.getItem('token') || token || ''
    return config
}

axios.interceptors.request.use((config) => {
    config = handleChangeRequestHeader(config)
    config = handleConfigureAuth(config)
    return config
})
```

响应拦截

响应错误由三类错误组成：

- 网络错误处理
- 授权错误处理
- 普通错误处理

因此，要优雅的处理响应拦截，我们必须先将三类错误函数写好，以便于我们增强代码扩展性及后期维护。

错误处理函数

```
const handleNetworkError = (errStatus) => {
    let errMessage = '未知错误'
    if (errStatus) {
        switch (errStatus) {
            case 400:
                errMessage = '错误的请求'
                break
            case 401:
                errMessage = '未授权，请重新登录'
                break
            case 403:
                errMessage = '拒绝访问'
                break
            case 404:
                errMessage = '请求错误,未找到该资源'
                break
            case 405:
                errMessage = '请求方法未允许'
                break
            case 408:
                errMessage = '请求超时'
                break
            case 500:
                errMessage = '服务器端出错'
                break
            case 501:
                errMessage = '网络未实现'
                break
            case 502:
                errMessage = '网络错误'
                break
            case 503:
                errMessage = '服务不可用'
                break
            case 504:
                errMessage = '网络超时'
                break
            case 505:
                errMessage = 'http版本不支持该请求'
                break
            default:
                errMessage = `其他连接错误 --${errStatus}`
        }
    } else {
        errMessage = `无法连接到服务器! `
    }

```

```
    message.error(errMessage)
}

const handleAuthError = (errno) => {
        const authErrMap: any = {
          '10031': '登录失效，需要重新登录', // token 失效
          '10032': '您太久没登录，请重新登录~', // token 过期
          '10033': '账户未绑定角色，请联系管理员绑定角色',
          '10034': '该用户未注册，请联系管理员注册用户',
          '10035': 'code 无法获取对应第三方平台用户',
          '10036': '该账户未关联员工，请联系管理员做关联',
          '10037': '账号已无效',
          '10038': '账号未找到',
        }

        if (authErrMap.hasOwnProperty(errno)) {
                message.error(authErrMap[errno])
                // 授权错误，登出账户
                logout()
                return false
        }

        return true
}

const handleGeneralError = (errno, errmsg) => {
        if (err.errno !== '0') {
                meessage.error(err.errmsg)
                return false
        }

        return true
}
```

适配

当我们将所有的错误类型处理函数写完，在 `axios` 的拦截器中进行调用即可。

```
axios.interceptors.response.use(
    (response) => {
        if (response.status !== 200) return Promise.reject(response.data)

        handleAuthError(response.data.errno)
        handleGeneralError(response.data.errno, response.data.errmsg)

```

```
        return response
    },
    (err) => {
        handleNetworkError(err.response.status)
        Promise.reject(err.response)
    }
)
```

针对单独接口的处理

基于上面的几类通用处理，我们这个请求的封装基本已经可用了。

但是我们还有一些额外的操作无处存放（参数处理、返回值处理），且我们并不想将他们耦合在页面中每次调用进行处理，那么我们显然需要一个位置来处理这些内容。

```
import { Get } from "../server"

interface FcResponse<T> {
      errno: string
      errmsg: string
      data: T
}

type ApiResponse<T> = Promise<[any, FcResponse<T> | undefined]>

function getUserInfo<T extends { id: string; name: string; }>(id):
ApiResponse<T> {
      return Get<T>('/user/info', { userid: id })
}
```

封包处理

接口分类封包

用户数据： `api/path/user.ts`

```
import { Get } from "../server"

export function getUserInfo(id) { ... }

export function getUserName(id) { ... }

export const userApi = {
```

```
        getUserInfo,
        getUserName
}
```

订单数据： `api/path/shoporder.ts`

```
import { Get } from "../server"

function getShoporderDetail() { ... }

function getShoporderList() { ... }

export const shoporderApi = {
        getShoporderDetail,
        getShoporderList
}
```

调用点统一

`api/index.ts`

```
import { userApi } from "./path/user"
import { shoporderApi } from "./path/shoporder"

export const api = {
        ...userApi,
        ...shoporderApi
}
```

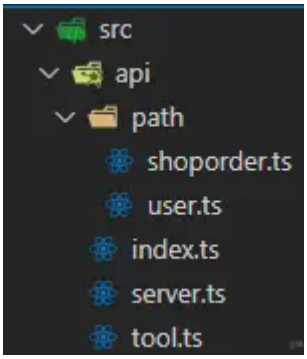
针对所有接口的处理（Post、Put、Del）

```
export const post = <T,>(url: string, data: IAnyObj, params: IAnyObj = {}):
Promise<[any, FcResponse<T> | undefined]> => {
  return new Promise((resolve) => {
    axios
      .post(url, data, { params })
      .then((result) => {
        resolve([null, result.data as FcResponse<T>])
      })
      .catch((err) => {
        resolve([err, undefined])
```

```
        })
    })
}

// Put / Del 同理
```

完整代码



业务处理函数： `src/api/tool.ts`

```
const handleRequestHeader = (config) => {
        config['xxxx'] = 'xxx'

        return config
}

const handleAuth = (config) => {
        config.header['token'] = localStorage.getItem('token') || token || ''
        return config
}
const handleNetworkError = (errStatus) => {
    let errMessage = '未知错误'
    if (errStatus) {
        switch (errStatus) {
            case 400:
                errMessage = '错误的请求'
                break
            case 401:
                errMessage = '未授权，请重新登录'
                break
            case 403:
                errMessage = '拒绝访问'
                break
            case 404:
                errMessage = '请求错误,未找到该资源'
                break
```

```
            case 405:
                errMessage = '请求方法未允许'
                break
            case 408:
                errMessage = '请求超时'
                break
            case 500:
                errMessage = '服务器端出错'
                break
            case 501:
                errMessage = '网络未实现'
                break
            case 502:
                errMessage = '网络错误'
                break
            case 503:
                errMessage = '服务不可用'
                break
            case 504:
                errMessage = '网络超时'
                break
            case 505:
                errMessage = 'http版本不支持该请求'
                break
            default:
                errMessage = `其他连接错误 --${errStatus}`
        }
    } else {
        errMessage = `无法连接到服务器! `
    }

    message.error(errMessage)
}

const handleAuthError = (errno) => {
        const authErrMap: any = {
          '10031': '登录失效，需要重新登录', // token 失效
          '10032': '您太久没登录，请重新登录~', // token 过期
          '10033': '账户未绑定角色，请联系管理员绑定角色',
          '10034': '该用户未注册，请联系管理员注册用户',
          '10035': 'code 无法获取对应第三方平台用户',
          '10036': '该账户未关联员工，请联系管理员做关联',
          '10037': '账号已无效',
          '10038': '账号未找到',
        }

        if (authErrMap.hasOwnProperty(errno)) {
```

```
                message.error(authErrMap[errno])
                // 授权错误，登出账户
                logout()
                return false
        }

        return true
}

const handleGeneralError = (errno, errmsg) => {
        if (err.errno !== '0') {
                meessage.error(err.errmsg)
                return false
        }

        return true
}
```

通用操作封装： `src/api/server.ts`

```
import axios from 'axios'
import { message } from 'antd'

import {
        handleChangeRequestHeader,
        handleConfigureAuth,
        handleAuthError,
        handleGeneralError,
        handleNetworkError
} from './tools'

type Fn = (data: FcResponse<any>) => unknown

interface IAnyObj {
    [index: string]: unknown
}

interface FcResponse<T> {
    errno: string
    errmsg: string
    data: T
}

axios.interceptors.request.use((config) => {
  config = handleChangeRequestHeader(config)
```

```
        config = handleConfigureAuth(config)
        return config
})

axios.interceptors.response.use(
    (response) => {
        if (response.status !== 200) return Promise.reject(response.data)
        handleAuthError(response.data.errno)
        handleGeneralError(response.data.errno, response.data.errmsg)
        return response
    },
    (err) => {
        handleNetworkError(err.response.status)
        Promise.reject(err.response)
    }
)

export const Get = <T,>(url: string, params: IAnyObj = {}, clearFn?: Fn):
Promise<[any, FcResponse<T> | undefined]> =>
  new Promise((resolve) => {
    axios
      .get(url, { params })
      .then((result) => {
        let res: FcResponse<T>
        if (clearFn !== undefined) {
          res = clearFn(result.data) as unknown as FcResponse<T>
        } else {
          res = result.data as FcResponse<T>
        }
        resolve([null, res as FcResponse<T>])
      })
      .catch((err) => {
        resolve([err, undefined])
      })
  })

export const Post = <T,>(url: string, data: IAnyObj, params: IAnyObj = {}):
Promise<[any, FcResponse<T> | undefined]> => {
  return new Promise((resolve) => {
    axios
      .post(url, data, { params })
      .then((result) => {
        resolve([null, result.data as FcResponse<T>])
      })
      .catch((err) => {
        resolve([err, undefined])
      })
```

```
  })
}
```

统一调用点： `src/api/index.ts`

```
import { userApi } from "./path/user"
import { shoporderApi } from "./path/shoporder"

export const api = {
        ...userApi,
        ...shoporderApi
}
```

接口： `src/api/path/user.ts` | `src/api/path/shoporder.ts`

```
import { Get } from "../server"

export function getUserInfo(id) { ... }

export function getUserName(id) { ... }

export const userApi = {
        getUserInfo,
        getUserName
}
```

```
import { Get } from "../server"

function getShoporderDetail() { ... }

function getShoporderList() { ... }

export const shoporderApi = {
        getShoporderDetail,
        getShoporderList
}
```