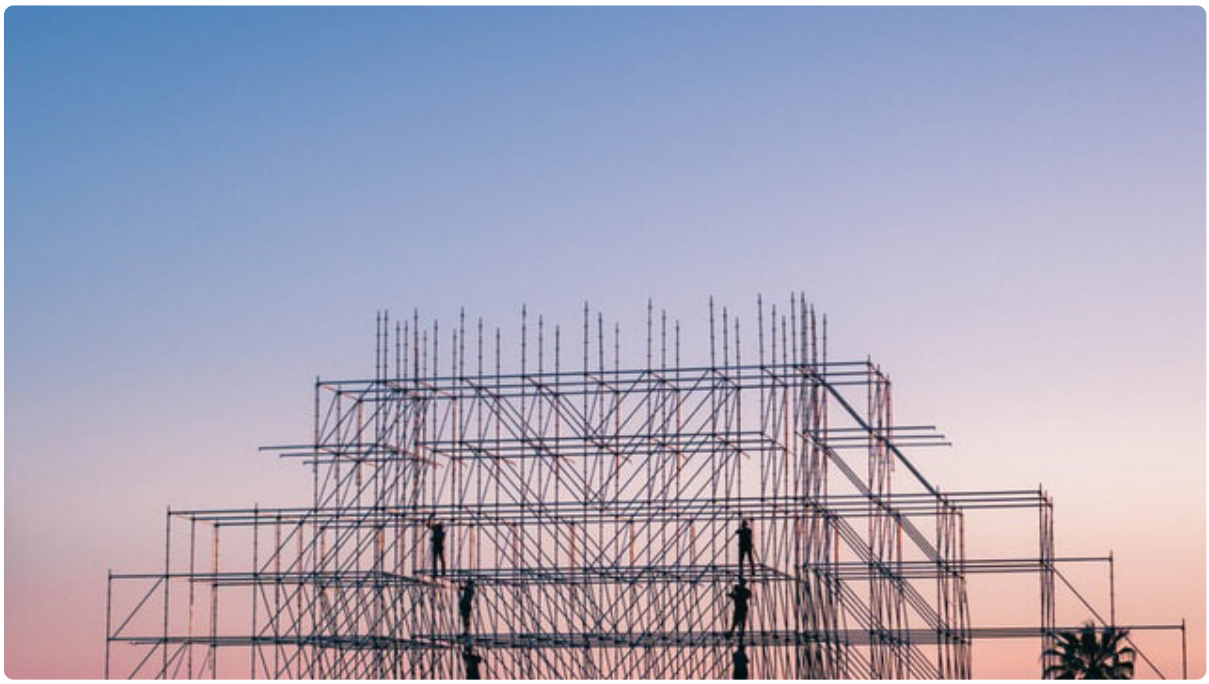
封装接口请求

更新时间：2019-07-25 10:03:43



> 加紧学习，抓住中心，宁精勿杂，宁专勿多。
>
> —— 周恩来

我们在开发中广泛使用的axios调用接口，为了更好地调用，做一些全局的拦截，我们通常会对其进行一下封装，本小节我们就来看下如何使用TypeScript对它进行封装，使其同时能够有很好的类型支持。

首先我们需要安装axios，如何安装依赖我们在1.3小节学习过了，这里直接安装：

```
npm install axios
```

我们在src文件夹下创建*utils*文件夹，然后在utils文件夹创建*axios.ts文件，axios这个插件自带声明文件，所以我们不需要另外安装了。先来看下最基础的封装：

```
import axios, { AxiosInstance, AxiosRequestConfig, AxiosPromise,AxiosResponse } from 'axios'; // 引入axios和定义在node_modules/axios/index.ts文件里的类型声明

class HttpRequest { // 定义一个接口请求类，用于创建一个axios请求实例
  constructor(public baseUrl: string) { // 这个类接收一个字符串参数，是接口请求的基本路径
    this.baseUrl = baseUrl;
  }
  public request(options: AxiosRequestConfig): AxiosPromise { // 我们实际调用接口的时候调用实例的这个方法，他返回一个AxiosPromise
    const instance: AxiosInstance = axios.create() // 这里使用axios.create方法创建一个axios实例，他是一个函数，同时这个函数包含多个属性，就像我们前面讲的计数器的例子
    options = this.mergeConfig(options) // 合并基础路径和每个接口单独传入的配置，比如url、参数等
    this.interceptors(instance, options.url) // 调用interceptors方法使拦截器生效
    return instance(options) // 最后返回AxiosPromise
  }
  private interceptors(instance: AxiosInstance, url?: string) { // 定义这个函数用于添加全局请求和响应拦截逻辑
    // 在这里添加请求和响应拦截
  }
  private mergeConfig(options: AxiosRequestConfig): AxiosRequestConfig { // 这个方法用于合并基础路径配置和接口单独配置
    return Object.assign({ baseURL: this.baseUrl }, options);
  }
}
export default HttpRequest;
```

我们封装了这个类之后，里面的baseUrl一般本地开发环境的基础路径和线上生产环境的基础路径是不一样的，所以可以根据当前是开发环境还是生产环境做判断，应用不同的基础路径。这里我们要写在一个配置文件里，我们在*src*文件夹下创建一个*config*文件夹，然后在这个文件夹创建一个*index.ts*文件，在里面配置生产和开发环境接口基础路径：

```
// src/config/index.ts
export default {
  api: {
    devApiBaseUrl: '/test/api/xxx',
    proApiBaseUrl: '/api/xxx',
  },
};
```

然后我们在axios.ts文件里直接引入接口配置，然后判断当前环境：

```
// ... 其他代码省略
import config from '@/config'; // @代表src一级目录，是我们在vue.config.js文件里配置的
const { api: { devApiBaseUrl, proApiBaseUrl } } = config;
const apiBaseUrl = process.env.NODE_ENV === 'production' ? proApiBaseUrl : devApiBaseUrl;
// process.env.NODE_ENV是vue服务内置的环境变量，有两个值，当本地开发时是development，当打包时是production
// ... 其他代码省略
```

接下来我们就可以将这个apiBaseUrl作为默认值传入HttpRequest的参数：

```
// ...
class HttpRequest { // 定义一个接口请求类，用于创建一个axios请求实例
  constructor(public baseUrl: string = apiBaseUrl) { // 这个类接收一个字符串参数，是接口请求的基本路径
    this.baseUrl = baseUrl;
  }
  // ...
// ...
```

接下来我们来完善拦截器，在类中interceptors方法内添加请求拦截器和响应拦截器，实现对所有接口请求的统一处理：

```
private interceptors(instance: AxiosInstance, url?: string) { // 定义这个函数用于添加全局请求和响应拦截逻辑
  // 在这里添加请求和响应拦截
  instance.interceptors.request.use((config: AxiosRequestConfig) => {
    // 接口请求的所有配置，都在这个config对象中，他的类型是AxiosRequestConfig，你可以看到他有哪些字段
    // 如果你要修改接口请求配置，需要修改 axios.defaults 上的字段值
    return config
  },
  (error) => {
    return Promise.reject(error)
  })
  instance.interceptors.response.use((res: AxiosResponse) => {
    const { data } = res // res的类型是AxiosResponse<any>，包含六个字段，其中data是服务端返回的数据
    const { code, msg } = data // 通常服务端会将响应状态码、提示信息、数据等放到返回的数据中
    if (code !== 0) { // 这里我们在服务端将正确返回的状态码标为0
      console.error(msg) // 如果不是0，则打印错误信息，我们后面讲到UI组件的时候，这里可以使用消息窗提示
    }
    return res // 返回数据
  },
  (error) => { // 这里是遇到报错的回调
    return Promise.reject(error)
  })
 }
```

请求时的拦截通过给`instance.interceptors.request.use`传入回调函数来处理，这个回调函数有一个参数，就是这次请求的配置对象。如果你需要修改请求的配置，需要修改`axios.defaults`上的字段，来修改全局配置。比如你要在header中添加字段"Content-Type"，需要这样添加：

```
axios.defaults.headers.post['Content-Type'] = 'application/x-www-form-urlencoded'
```

到这里我们这个axios请求类就封装好了，上面我们讲了一般服务端会将状态码、提示信息和数据封装在一起，然后作为数据返回，所以我们调用所有api请求返回的数据格式都是一样的，那么我们就可以定义一个接口来指定返回的数据的结构，我们在*src/utils/axios.ts*文件中定义`class HttpRequest`前加一个接口定义，并且导出：

```
export interface ResponseData {
  code: number
  data?: any
  msg: string
}
```

你可以根据你服务端实际返回的情况，来定义这个接口ResponseData，我这里data字段设为可选的。

接下来我们来简单使用一下。在*src*文件夹下创建一个*api*文件夹，然后再这个文件夹创建一个*index.ts*文件，在这里创建请求实例：

```
// src/api/index.ts
import HttpRequest from '@/utils/axios'
export * from '@/utils/axios'
export default new HttpRequest()
```

我们把这个请求类引进来，然后默认导出一个这个类的实例。后面我们要进行接口请求的时候，要对接口进行分类，比如和用户相关的，和数据相关的等等。这里我们先添加一个登陆接口的请求，所以在*api*文件夹下创建一个*user.ts*文件，在这里创建一个登陆接口请求方法：

```
// src/api/user.ts
import axios, { ResponseData } from './index'
import { AxiosPromise } from 'axios'

interface LoginReqArguInterface {
  user_name: string;
  password: number|string
}
export const loginReq = (data: LoginReqArguInterface): AxiosPromise<ResponseData> => {
  return axios.request({
    url: '/api/user/login',
    data,
    method: 'POST'
  })
}
```

这里我们封装登录请求方法*loginReq*，指定他需要一个参数，参数类型要符合我们定义的`LoginReqArguInterface`接口指定的结构，这个方法返回一个类型为`AxiosPromise`的Promise，AxiosPromise这个接口是axios声明文件内置的类型，他可以传入一个泛型变量参数，用于指定api请求返回的结果中data字段的类型。

> 为什么我们要这样封装api的请求，而不是直接在需要调用接口的地方使用`axios.get`或者`axios.request`这些方法直接请求呢？这是因为我们一个api请求可能多个地方需要用到，这里的登录接口复用性很低，但是一些其他接口比如获取数据的接口，可能多个地方会用到，所以我们封装一下，可以统一管理，以后接口升级，就只需要改一个地方就可以了。

接下来我们调用一个这个请求试一下，我们在*src/views/Home.vue*文件里先调用一下，只是为了先感受一下调用请求的方式，登录接口我们会在下个小节放到登录页实现。

打开*Home.vue*文件后，我们首先要引入这个请求方法：

```
import { loginReq } from '@/api/user'
```

然后下面我们在Home这个组件类里添加mounted声明周期钩子函数，在这里调用这个api请求方法：

```
export default class Home extends Vue {
  public mounted() {
    loginReq({ user_name: 'Lison', password: 123 }).then((res) => {
      console.log(res.data.code)
    })
  }
}
```

在编写这些代码的时候，你可以看到，当你书写`loginReq()`之后，编辑器提示你：`Expected 1 arguments, but got 0.`，这是因为我们在定义loginReq的时候指定了一个参数，当你传入一个空对象`{}`的时候，编辑器会提示你：`Type '{}' is missing the following properties from type 'LoginReqArguInterface': user_name, password`，这是因为我们指定了参数必须包含user_name和password两个字段。这样我们封装好api请求方法后，任何人使用我们的请求方法，都可以得到很好的代码提示。

调用这个方法后他的返回值是一个Promise，所以我们在返回值上调用then方法，他传入一个回调函数，这个回调函数有一个参数，类型是`AxiosResponse`，然后你会发现，当我们输入在回调函数中使用console.log()打印东西时，当我们输入`res`然后输入`.`之后，编辑器列出了六个可访问的属性：data、status、statusText、headers、config和request，当我们选中`data`然后输入`.`后，编辑器列出了三个可访问属性：code、data和msg，这三个是我们自己通过接口`ResponseData`定义的。

当然现在你调用这个接口肯定会报404错误，因为我们没有后端服务，下个小节我们会使用**mockjs**来拦截请求，用来在前端模拟响应api请求，然后实现一个登录页，根据api接口返回的结果判断是否登录成功。

搭建基础项目

实现登录页并用Mock响应请求