# 【前端面试】你封装过Axios请求吗?

## 前言

axios 是目前最优秀的 HTTP 请求库之一，虽然 axios 已经封装的非常好了，我们可以直接拿过来用。但是在实际的项目中，我们可能还需要对 axios 在封装一下，以便我们更好的管理项目和各个借口。

但是，目前网上有特别多的针对于 axios 在项目中的封装。不得不说，很多大佬封装得非常全面，方方面面都考虑到了。但是我们的每个真的都需要那些封装吗？显然不是的，网上的很多封装其实都显得**有点过度封装了**！

本篇文章实现最简单 Axios 封装，让小伙伴们扩展起来容易一些。

## 1.封装目的

此次进行简单的封装，所以暂时没有考虑**取消重复请求、重复发送请求、请求缓存**等情况！这里主要实现以下目的：

1. 实现请求拦截
2. 实现响应拦截
3. 常见错误信息处理
4. 请求头设置
5. api 集中式管理

## 2.初始化 axios 实例

虽然 axios 可以调用 get、post 等方法发起请求，但是我们为了更好的全局控制所有请求的相关配置，所以我们使用 axios.create()创建实例的方法来进行相关配置，这也是封装 axios 的精髓所在。

**示例代码：**

```
// 创建 axios 请求实例
const serviceAxios = axios.create({
  baseURL: "", // 基础请求地址
  timeout: 10000, // 请求超时设置
  withCredentials: false, // 跨域请求是否需要携带 cookie
});
```

通过 create 方法我们得到了一个 axios 的实例，该实例上有很多方法，比如拦截器等等。我们创建实例的时候可以配置一些基础设置，比如基础请求地址，请求超时等等。

## 3.设置请求拦截

我们在发送请求的时候可能需要携带一些信息在请求头上，比如 token 等，所以说我们就需要将请求拦截下来，处理一些我们的业务逻辑。

**示例代码：**

```
// 创建请求拦截
serviceAxios.interceptors.request.use(
  (config) => {
    // 如果开启 token 认证
    if (serverConfig.useTokenAuthorization) {
      config.headers["Authorization"] = localStorage.getItem("token"); // 请
    }
    // 设置请求头
    if(!config.headers["content-type"]) { // 如果没有设置请求头
      if(config.method === 'post') {
        config.headers["content-type"] = "application/x-www-form-urlencoded'
        config.data = qs.stringify(config.data); // 序列化,比如表单数据
      } else {
        config.headers["content-type"] = "application/json"; // 默认类型
      }
    }
    console.log("请求配置", config);
    return config;
  },
  (error) => {
    Promise.reject(error);
  }
);
```

我们通过调用 axios 的实例方法来进行请求拦截。

## 4.设置响应拦截

axios 请求的返回结果里面包含了很多东西，我们的业务层面通常只需要后端返回的数据即可，所以我们需要设置相应拦截，在响应结果返回给业务层之前做一些操作。

**示例代码：**

```
// 创建响应拦截
serviceAxios.interceptors.response.use(
  (res) => {
    let data = res.data;
    // 处理自己的业务逻辑，比如判断 token 是否过期等等
    // 代码块
    return data;
```

```
  },
  (error) => {
    let message = "";
    if (error && error.response) {
      switch (error.response.status) {
        case 302:
          message = "接口重定向了！";
          break;
        case 400:
          message = "参数不正确！";
          break;
        case 401:
          message = "您未登录，或者登录已经超时，请先登录！";
          break;
        case 403:
          message = "您没有权限操作！";
          break;
        case 404:
          message = `请求地址出错: ${error.response.config.url}`;
          break;
        case 408:
          message = "请求超时！";
          break;
        case 409:
          message = "系统已存在相同数据！";
          break;
        case 500:
          message = "服务器内部错误！";
          break;
        case 501:
          message = "服务未实现！";
          break;
        case 502:
          message = "网关错误！";
          break;
        case 503:
          message = "服务不可用！";
          break;
        case 504:
          message = "服务暂时无法访问，请稍后再试！";
          break;
        case 505:
          message = "HTTP 版本不受支持！";
          break;
        default:
          message = "异常问题，请联系管理员！";
          break;
      }
    }
    return Promise.reject(message);
  }
```

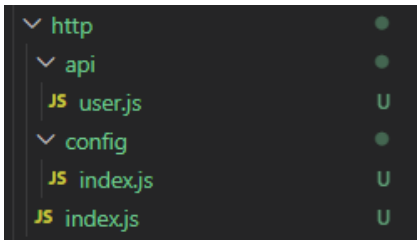
```
);
```

## 5.完整示例

上面直接简单介绍了拦截等相关代码，接下来我们在实际项目中来演练一下，以 Vue 项目为例。

在 src 下面新建 http 文件夹，用来存储关于 axios 请求的一些文件，然后在 http 文件夹下新建 index.js 文件，用于封装我们的 axios，然后在新建 config 文件夹，主要用来创建配置文件，最后新建一个 api 文件夹，用于集中管理我们的接口。

**文件夹目录如下：**




**index.js 代码：**

```
import axios from "axios";
import serverConfig from "./config";
import qs from "qs";

// 创建 axios 请求实例
const serviceAxios = axios.create({
  baseURL: serverConfig.baseURL, // 基础请求地址
  timeout: 10000, // 请求超时设置
  withCredentials: false, // 跨域请求是否需要携带 cookie
});

// 创建请求拦截
serviceAxios.interceptors.request.use(
  (config) => {
    // 如果开启 token 认证
    if (serverConfig.useTokenAuthorization) {
      config.headers["Authorization"] = localStorage.getItem("token"); // 请
    }
    // 设置请求头
    if(!config.headers["content-type"]) { // 如果没有设置请求头
      if(config.method === 'post') {
        config.headers["content-type"] = "application/x-www-form-urlencoded'
        config.data = qs.stringify(config.data); // 序列化,比如表单数据
      } else {
        config.headers["content-type"] = "application/json"; // 默认类型
      }
    }
    console.log("请求配置", config);
    return config;
```

```
  },
  (error) => {
    Promise.reject(error);
  }
);

// 创建响应拦截
serviceAxios.interceptors.response.use(
  (res) => {
    let data = res.data;
    // 处理自己的业务逻辑，比如判断 token 是否过期等等
    // 代码块
    return data;
  },
  (error) => {
    let message = "";
    if (error && error.response) {
      switch (error.response.status) {
        case 302:
          message = "接口重定向了！";
          break;
        case 400:
          message = "参数不正确！";
          break;
        case 401:
          message = "您未登录，或者登录已经超时，请先登录！";
          break;
        case 403:
          message = "您没有权限操作！";
          break;
        case 404:
          message = `请求地址出错: ${error.response.config.url}`;
          break;
        case 408:
          message = "请求超时！";
          break;
        case 409:
          message = "系统已存在相同数据！";
          break;
        case 500:
          message = "服务器内部错误！";
          break;
        case 501:
          message = "服务未实现！";
          break;
        case 502:
          message = "网关错误！";
          break;
        case 503:
          message = "服务不可用！";
          break;
```

```
        case 504:
          message = "服务暂时无法访问，请稍后再试！";
          break;
        case 505:
          message = "HTTP 版本不受支持！";
          break;
        default:
          message = "异常问题，请联系管理员！";
          break;
      }
    }
    return Promise.reject(message);
  }
);
export default serviceAxios;
```

config/index.js 代码：

```
const serverConfig = {
  baseURL: "https://smallpig.site", // 请求基础地址,可根据环境自定义
  useTokenAuthorization: true, // 是否开启 token 认证
};
export default serverConfig;
```

api/user.js 接口调用示例代码：

```
import serviceAxios from "../index";

export const getUserInfo = (params) => {
  return serviceAxios({
    url: "/api/website/queryMenuWebsite",
    method: "post",
    params,
  });
};
export const login = (data) => {
  return serviceAxios({
    url: "/api/user/login",
    method: "post",
    data,
  });
};
```

注意：get 请求需要传 params，post 请求需要传 data。

Vue 文件中调用示例：

```
import { login } from "@/http/api/user"
async loginAsync() {
  let params = {
    email: "123",
    password: "12321"
  }
```

```
  let data = await login(params);
  console.log(data);
}
```

## 6.总结

我们这里只做了最最最基础的 axios 封装，但是可扩展性较高。相较于其它文章的过度封装，这里的封装形式其实可以满足大部分应用场景了。

我们在此基础上可以根据业务场景做一些以下扩展建议：

- 请求拦截里面针对 token 进行处理
- 响应拦截里面判断 token 是否过期等等
- 在 config/index.js 里面动态更改 baseURL
- 在请求拦截里面根据业务场景修改请求头
- 在拦截里面设置全局请求进度条等等