axios 的简单使用

GET 请求

GET 请求一般用来向服务器查询数据，使用 axios 来发送 GET 请求可以简单地分为三种情况：

1 不带查询参数

即没有要发送给后台的数据，如：

```
axios.get('/user')
  .then(function (response) {
    // 请求成功时执行
    console.log(response)
  })
  .catch(function (error) {
    // 请求失败时执行
    console.log(error)
  })
  .then(function () {
    // 永远执行
  })
```

我常用的是使用**箭头函数**的传参：

```
axios.get('/user')
  .then(resp => {
    // do something
  })
  .catch(err => {
    // ...
  })
  .then(() => {
    // ...
  })
```

2 带查询参数，但查询参数较少

此时可以直接用字符串拼接的方法，将参数拼到 url 上

一个参数：axios.get('/user?id=' + id) // ?代表后面有查询参数

多个参数：axios.get('/user?id=' + id + '&name=' + name) // 用&来连接参数

当然也可以用 Javascript 里的**格式化字符串语法**：

```
axios.get(`/user?id=${id}`) // 注意是反引号``
axios.get(`/user?id=${id}&name=${name}`)
```

3 带查询参数，但查询参数较多

这时用 axios 提供的接口，将参数以字典的形式传入 get

```
axios.get('/user', {
    params: {
      id: 1234,
      age: 18,
      name: 'haha'
    }
  })
```

POST 请求

POST 请求用于向服务器发送数据，一般用来提交一个表单，形式比较简单。即在 POST 函数中提交一个字典，里面是要发送给服务器的数据。

```
axios.post('/user', {
    firstName: 'Fred',
    lastName: 'Flintstone'
  })
  .then(resp => {
    console.log(resp);
  })
  .catch(err => {
    console.log(err);
  });
```

Response

当请求提交给服务器后，服务器会发送一个 Response 对象，里要有我们需要的数据。一般我们请求的数据都位于 data 字段中，即：

```
axios.get('/user')
```

- .then(resp => {
    - // 比如获取服务器发来的 user 数据
    - console.log(resp.data.user)
- })

API 的编写方法

下面解释我们项目中的 API 该如何编写，这里考虑的是 vue 页面中的业务逻辑调用 api 文件中定义的函数请求后端的情况（此即我们项目中请求后端的方法，模块化，低耦合）。一般有两种方法：

1 使用回调函数

使用回调函数是在调用 api 中的函数接口时，将处理请求结果的函数直接传递给 api 中的接口函数。这种方法的结果是在 vue 页面中编写的代码逻辑少，在 api 函数中编写的逻辑多。举例来说：

（api 文件中编写）

其中 success, fail, error 均为回调函数

```
getUserId:function(success, fail, error){
  this.axios.get('/api/user/getUserId')
    .then(resp => {
      if (resp.data.status === 'success') {
        // 成功时调用，将 userId 传递给 success
        success(resp.data.userid)
      } else {
        // 失败时调用，将 errorMsg 传递给 fail
        fail(resp.data.errorMsg)
      }
    })
    // 出错时调用，将"请求错误"字符串传递给 error
    .catch(() => error('请求错误！'))
}
```

（vue 页面中编写）

在 vue 页面要先**编写回调函数**，再调用 api。

可以用传统的定义方式：

```
function success(userId) {
    // 假设这里将接口传回的 userId 赋值给当前 vue 实例的 id 字段
    this.id = userId
}
function fail(errorMsg) {}
function error(err) {}
// 调用接口
getUserId(success, fail, error)
```

也可以用箭头函数定义：

```
var success = userId => this.id = userId
var fail = errorMsg => {}
var error = err => {}
// 调用接口
```

```
getUserId(success, fail, error)
```

将定义与调用写在一起：

```
getUserId(
    userId => this.id = userId,
    errorMsg => {},
    err => {}
)
```

2 使用 Promise

由于 axios 返回的是 Promise 对象，所以可以直接返回，然后在页面上完成业务逻辑的编写。这种方法与第一种相反，在 vue 页面中编写的代码逻辑多，在 api 函数中编写的逻辑少。举例来说：

（api 文件中编写）

```
getUserId:function() {
  // 直接返回 Promise 对象
  return this.axios.get('/api/user/getUserId')
}
```

（vue 页面中编写）

```
// 调用接口
getUserId()
  .then(resp => {
     // 处理成功的情形
     this.id = resp.data.userId;
  })
  .catch(err => {
     // 处理出错的情形
     console.log(err)
  })
```

问题与回答

1. 为什么不直接返回 data 中的数据？

比如，直接在接口函数中返回数据：

```
getUserId: function() {
  this.axios.get('/api/user/getUserId')
    .then(resp => {
       // 成功时调用直接返回数据
       return resp.data.userId
    })
}
```

(未测试，慎用！！）
虽然可以用 axios 提供的特性来弥补这个不足，即用 **async/await** 来同步操作：
（api 文件中编写）

```
async getUserId: function() {
  var resp = await this.axios.get('/api/user/getUserId')
  return resp.data.userId
}
```

（vue 页面中编写）

```
// 调用接口
getUserId().then(userId => this.id = userId)
```

但由于这样会使功能的分布逻辑混乱，所以暂时不采用。

参考

1. axios 官方文档
2. JavaScript Callbacks
3. 理解 JavaScript 的 async/await
4. async/await

其他问题

在后面补充.....