TS&OOP day02

发现了一个大`bug`，**不登录也可以对数据进行增删改查。**实际上，应实现如下业务场景：

1. 只有登录后，才可以看到除登录之外的页面。
2. 只有登录后，才可以发送其他的`http`增删改查的请求，否则发送的所有请求都应该被驳回：不登录没有操作数据的权限。

只有登录后，才可以看到除登录之外的页面

实现思路：在项目的`VueRouter`对象中添加路由的**全局前置守卫。**前置守卫将会在跳转路由页面之前先执行，在此处判断，若用户没有登录，则直接跳转到登录页面。若用户已经登录，则可以通过验证，看到后续页面。

完整的导航解析流程

1. 导航被触发。
2. 在失活的组件里调用`beforeRouteLeave`守卫。
3. 调用全局的`beforeEach`守卫。
4. 在重用的组件里调用`beforeRouteUpdate`守卫 (2.2+)。
5. 在路由配置里调用`beforeEnter`。
6. 解析异步路由组件。
7. 在被激活的组件里调用`beforeRouteEnter`。
8. 调用全局的`beforeResolve`守卫 (2.5+)。
9. 导航被确认。
10. 调用全局的`afterEach`钩子。
11. 触发 DOM 更新。
12. 调用`beforeRouteEnter`守卫中传给`next`的回调函数，创建好的组件实例会作为回调函数的参数传入。

只有登录后，才可以发送其他的`http`增删改查的请求，否则发送的所有请求都应该被驳回：不登录没有操作数据的权限。

如果想要实现该功能，则需要服务端进行用户验证，验证当前请求中是否包含登录用户的身份（服务端需要完成用户的**鉴权**），如果该用户有访问该资源的权限，则执行相应业务；如果用户没有处理该请求的权限，则直接打回去，返回一个错误消息：您没有操作该模块的权限。

基于`Token`机制实现上述鉴权请求

1. 当登录成功后，从服务端获取登录成功后返回的结果，包含有：登录成功的用户以及`token`字符串。将这些信息保存在客户端供以后使用。

2. 以后同一个客户端发送后续所有请求时，都需要携带该 `token` 字符串一起发送请求，这样服务端才知道该客户端的登录账号是谁，拥有什么权限。因为所有的信息都在这个加了密的 `token` 字符串里，服务端可以对该字符串进行解密，获取 `token` 中保存的信息，完成相关业务。

   在axios中封装请求拦截器，使得每一个axios请求都会携带authorization消息头，将token字符串发给服务端：

   ```
   https://www.axios-http.cn/docs/interceptors
   ```

3. 为axios添加请求拦截器，在拦截器中，为请求数据包添加authorization消息头，把token放在此处发给服务端。

   ```
   /** 添加请求拦截器 */
   instance.interceptors.request.use(function (config) {
       // 在发送请求之前，为config配置authorization消息头
       let token = store.state.token  // token字符串
       if(token){
           config.headers.authorization = token
       }
       return config;
     }, function (error) {
       // 对请求错误做些什么
       return Promise.reject(error);
     });
   ```

TypeScript

`Typescript` 是 `Javascript` 的一个超集。 `Typescript` 在原有 `js` 的基础之上又添加了编译期的类型检查功能，意味着如果使用 `ts` 进行前端开发，会对变量的数据类型进行较为严格的验证，防止程序员写出可能出错的代码，规范编程习惯，适合大型项目开发使用。

`vue3.x`

`Typescript` 代码的编写及运行方式

`typescript` 代码写在后缀为 `.ts` 的文件中，这种文件可以被 `typescript` 编译期编译解析，最终转换为一套功能相同的 `js` 代码，输出到一个 `.js` 文件中，这个过程称为：**编译**。ts代码是无法直接运行的，它只是提供了一套编译环境，将 `ts` 转成 `js` ，真正运行的还是 `js` 代码。

如果在编译过程中， `ts` 的语法检查出现了错误，则会中断编译，报错提示。

全局安装 `Typescript` 编译器

```
npm install -g typescript    # 安装ts编译器
npm install -g ts-node       # 安装ts-node后可以使用vscode插件来执行ts文件
npm install -g tslib @types/node
```

安装成功后，就可以使用tsc命令，对ts文件进行编译：

```
tsc  helloworld.ts
```

将会编译`ts`文件，如果编译通过，生成`helloworld.js`，项目中使用的永远都是`js`文件。

`vscode`中有一个插件可以使用，自动编译`ts`文件，直接运行编译后的`js`：

```
code Runner
```

typescript语言官方文档：

https://www.tslang.cn/docs/home.html

`Typescript`的数据类型

https://www.tslang.cn/docs/handbook/basic-types.html

案例：`01_basictype.ts`测试ts的基本数据类型

```
// 测试基本数据类型  demo/01_basictype.ts

// 声明变量描述一个人，声明的语法中通过冒号:指定变量的类型
// 固定类型的变量不能保存不匹配的其他类型的数据
let pName:string = '亮亮';
let pAge:number = 28;
let pMarried:boolean = true;

// 声明变量，描述数组类型数据.   string[]表示字符串数组类型（数组中只能存放字符串）
let pHobby :string[]       = ['摊煎饼', '玩单杠', '擦玻璃']
let pFav   :Array<string>  = ['健身', '游泳', '瑜伽', '拉丁']

// 声明变量，描述一个元组（表示一个已知元素数量和类型的数组）
let params:[string, number, number] = ['杀手', 1, 20]
// params = ['功夫', 2, 30, 1]  错误，数量不匹配
params = ['功夫', 2, 30]
params = ['熊猫', 1, 100]

// 人为指定元素中保存数据的含义，即可使用元组方便的描述一个具体事物
//           阶数      材质
let mofang:[number, string] = [3, '塑料']
mofang = [4, '合金']
```

```
// 测试枚举类型
// 声明一个枚举类型Color，该类型数据的取值一共只有3类：
// Color.Red  Color.Green  Color.Blue
// 在使用的过程中不关心这3类值具体是多少，只关心值种类的不同即可
enum Color {Red, Green, Blue}
console.log(Color.Red)   //  -> 0
console.log(Color.Green) //  -> 1
console.log(Color.Blue)  //  -> 2

// 声明一个元组，描述亮亮今天穿的衣服
let cl = ['皮夹克', Color.Red]
cl = ['棉裤', Color.Blue]

// 声明一个枚举类型，描述性别：Male   Female
enum Gender {Male, Female}
// 描述亮亮：
pName = '成小亮'
// pGender为Gender枚举类型，意味着取值只能是Gender枚举中两个取值之一
let pGender:Gender = Gender.Male

// 定义一个枚举，指定可能会发送请求的目标请求地址
enum BMDURL {
    ACTOR_ADD = '/actor/add',
    MOVIE_LIST = '/movie-infos',
    CINEMA_ADD = '/cinema/add'
}
// 通过枚举，定义大量的常量值，通过友好的名称方便访问
let url1 = 'http://localhost:8080' + BMDURL.ACTOR_ADD
let url2 = 'http://localhost:8080' + BMDURL.MOVIE_LIST
let url3 = 'http://localhost:8080' + BMDURL.CINEMA_ADD
console.log(url1)
console.log(url2)
console.log(url3)

// 何时可以声明枚举类型并合理使用呢？
// 当发现一个变量保存的值，仅仅可能是有限的几个种类中的其中之一时
// 例如：
// 枚举变量描述性别：  男   女
// 枚举变量描述答案：  正确   错误
// 枚举变量描述年龄段：  未成年   青年   中年   老年
// 枚举变量描述成绩：  优秀   良好   及格    不及格

enum Category {Hot=1, Wait=2, Classic=3}
console.log(Category.Hot) // 热映类别的值
console.log(Category[1])  // 类别值1 所对应的类别名称


// 类型断言
/**
 * myaxios.get("/actor/list", params).then(res=>{
 *   //  res.data是一个对象：  {code:200, msg:ok, data:[{},{}..]}
 *   //  ts编译器会认为res.data为any类型
```

```
 *    //  而具体res.data是什么类型？只有程序员知道，所以程序员
 *    //  可以在此处对any类型的数据进行：类型断言，人为干预ts对代码的解析
 * })
 */

let param:any = 'name=zs&pwd=1234&age=15'

// 调用split方法拆分字符串时，在此没有split方法的提示
// 因为ts人为param是个any类型的数据，没有当字符串来看
console.log(param.split('&'))

// 在此处如果人为确定param就是字符串类型，就可以做一次类型断言
let p2:string = <string>param     // 类型断言语法一    尖括号
let p3:string = param as string   // 类型断言语法二    as
console.log(p2.split('&')) // 在此就有split方法的提示了
console.log(p3.split('&')) // 在此就有split方法的提示了
```

`Typescript`中的函数

```
function aaa(a, b){
    return a+b
}
let r = aaa(100, 200)
```

和`javascript`的函数一样，`Typescript`函数也可以创建有名字的函数与匿名函数，并且在创建这些函数的过程中，明确函数的参数、返回值的数据类型。满足工程化开发的需要。

函数的类型

在`typescript`中声明函数时，需要指定函数的类型（参数的类型与返回值的数据类型）：

```
// 函数的类型 demo/02_func.ts
// 如下指定函数的类型：
// x:number  参数x为number类型
// y:number  参数y为number类型
// :number   该方法的返回值类型也是number
function add(x:number, y:number):number {
    return x + y
}
console.log(add(10, 20))
// console.log(add(10, 'a'))  错误：调用时传递的参数不符合函数的要求


// 声明一个函数，接收一个数字n，返回n以内的随机小数
function getNum(n:number):number{
    return Math.random() * n
}
console.log(getNum(25).toFixed(2))
```

```
let myadd: (x:number, y:number)=>number =
    function(a:number, b:number):number{
        return a+b
    }
console.log(myadd(10, 20))

// 报错，myadd只能指向一个 接收两个number，返回一个number的函数
// myadd = function(a){}

// 调用ts的函数时，参数列表必须与ts函数声明的参数列表一一对应
// console.log(myadd(10))   对应不上，则报错


// 如果有需求，可能传递一个，也可能传两个或多个参数时，该如何定义?
function sum(a:number, b:number, c?:number):number{
    if(c != undefined)
        return a + b + c
    else
        return a + b
}
console.log(sum(10, 20))
console.log(sum(10, 20, 30))
```

Typescript中自定义类型

```
let n = 100;        number类型
let s = 'hello';    string类型
let p = {name:'zs', age:15, gender:'男'};    object类型
let card = {suit:'红桃', rank:'K'}    object类型
```