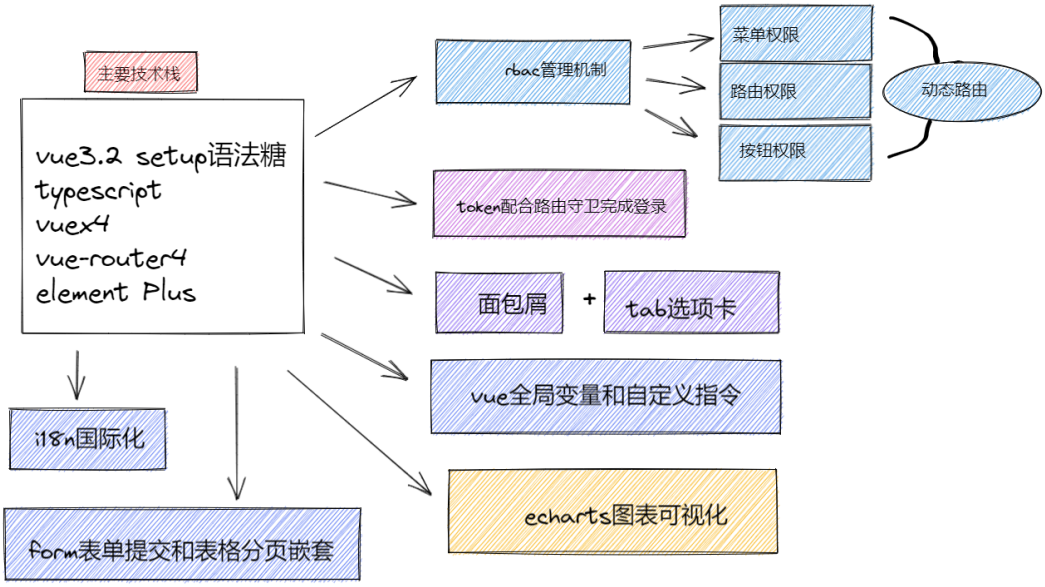
vue3+element plus后台管理ts版

大家都知道前端开发的重点难点都集中于js,现在vue3结合typescript是大势所趋,那么



在学习vue3知识体系前,先学习下typescript基础

Bug总结:

遇到bug不要慌!

bug1.使用select和switch组件的change时报错:

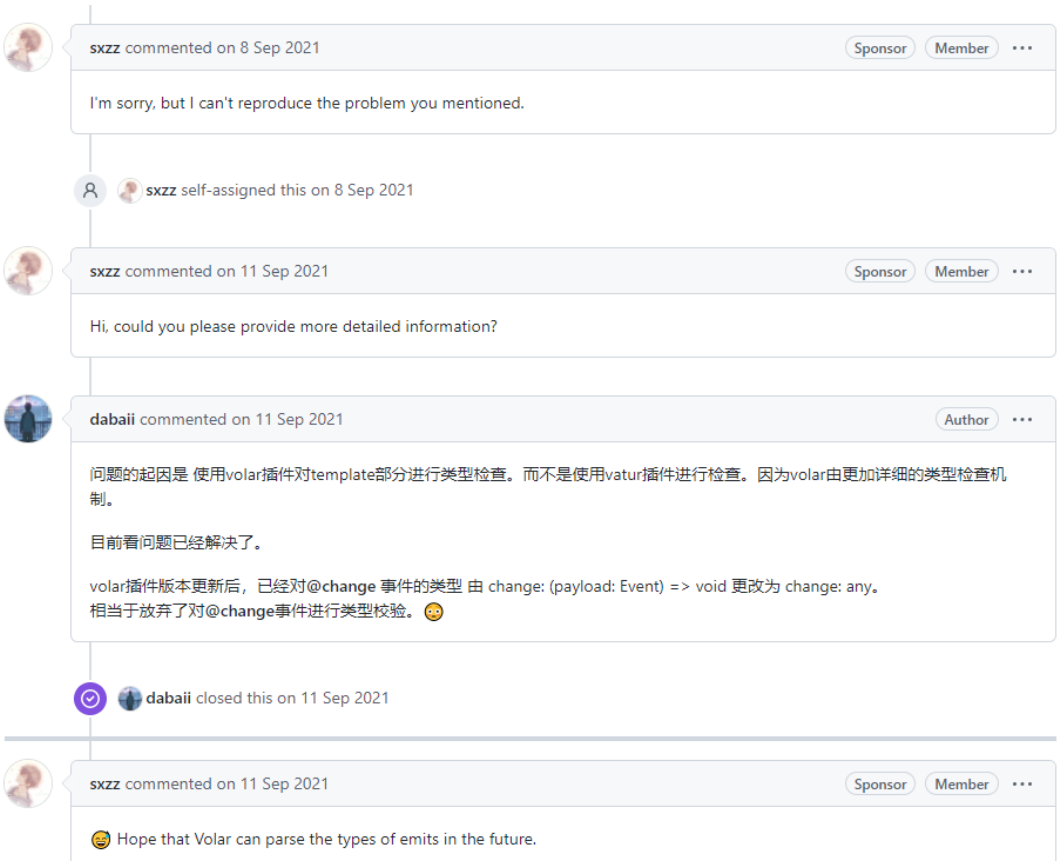
vue的vscode插件(volar)会对template部分进行ts语法分析，会为默认事件添加对应的类型。这样导致出现ts类型校验出错

"不能将类型“(value: string) => void”分配给类型“(payload: Event) => void”。 参数“value”和“payload”的类型不兼容。",

如何解决?,这个时候本来想去去elementplus和volar的issues提出问题

后来发现在2021年12月的时候volar版本就出过类似问题

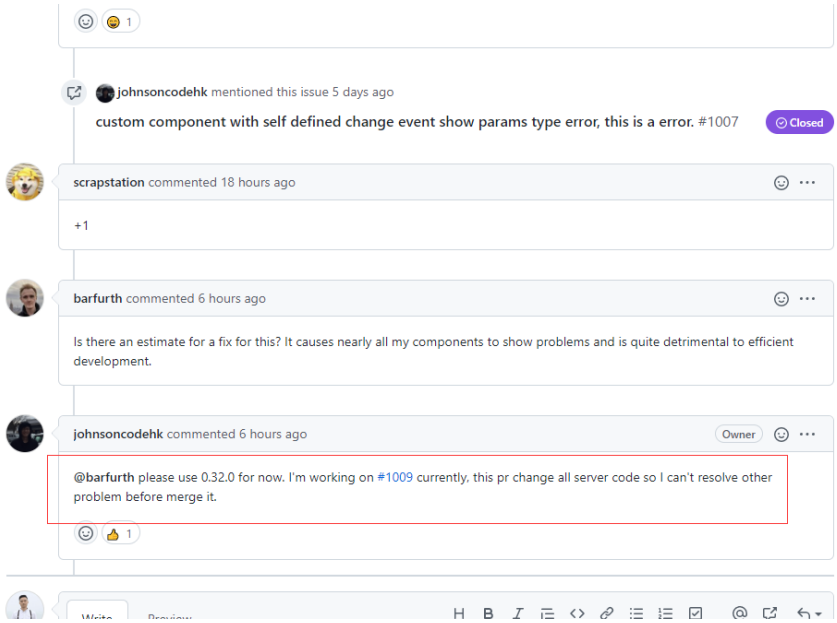
https://github.com/element-plus/element-plus/issues/3264



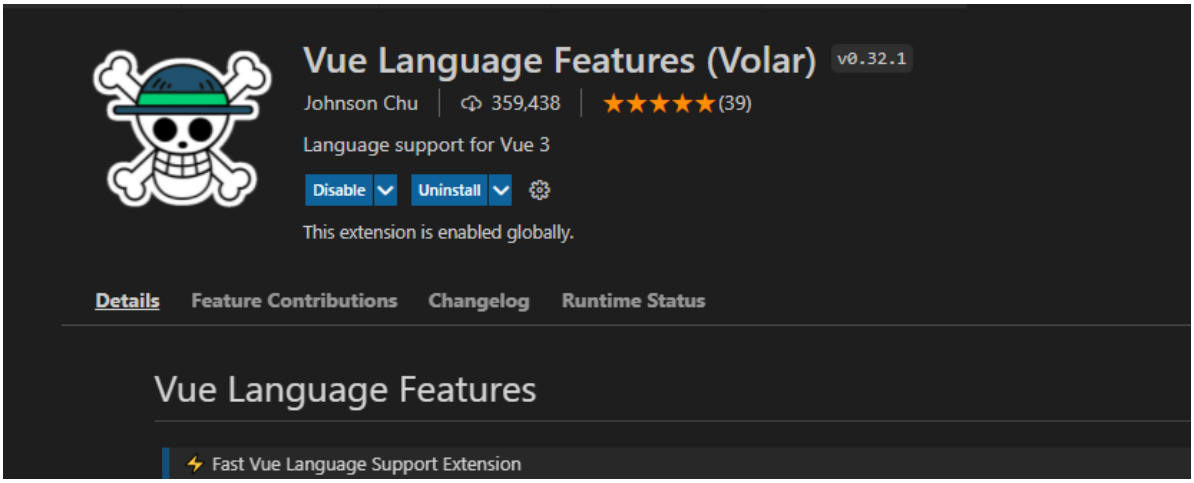
OK,那基本锁定问题是出在了volar版本更新的问题了

然后去volar的issues看看吧,果然volar的作者**johnsoncodehk** 建议先回退版本到0.32.0

https://github.com/johnsoncodehk/volar/issues/1001

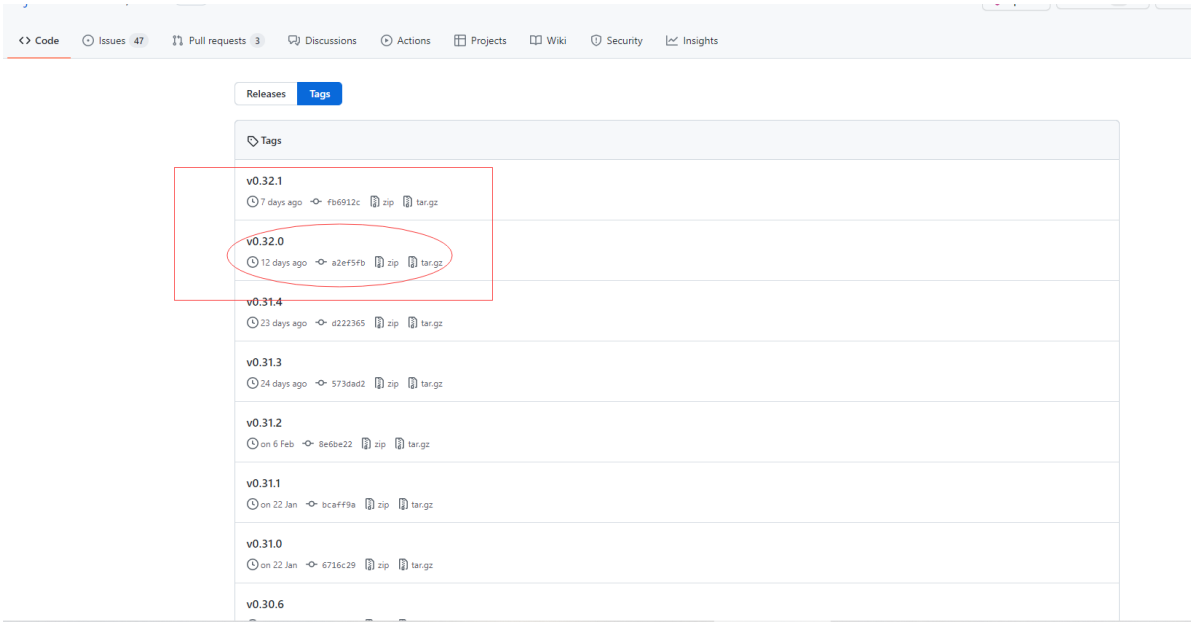


目前我的版本是0.32.1,看来是个过渡版本



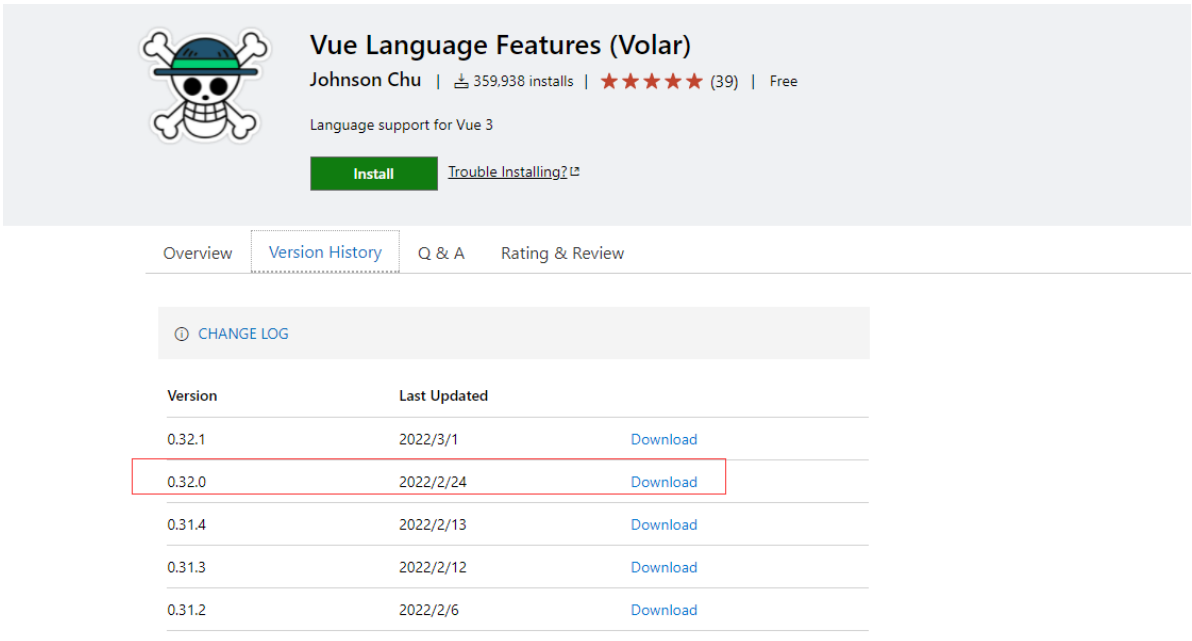
如何版本降级?

找到对应要回退的版本tag

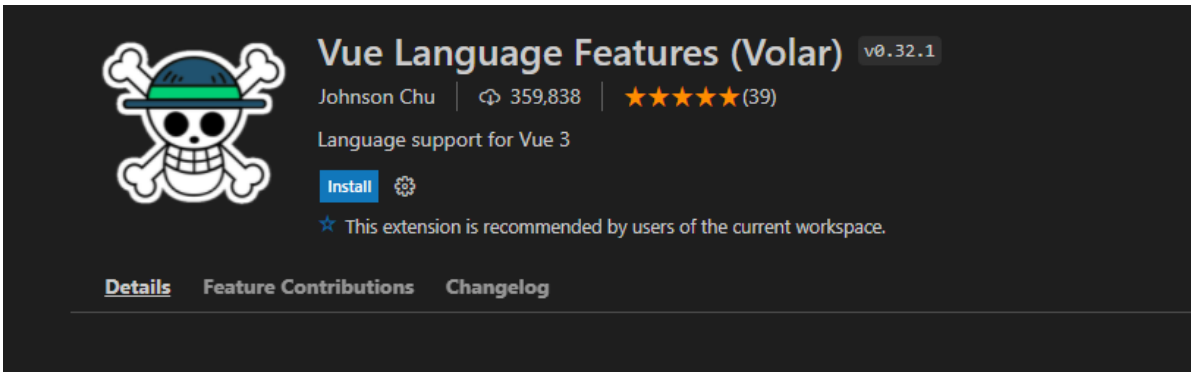


1 打开 VSCode 插件市场网址 https://marketplace.visualstudio.com/vscode，输入你想要的插件名称

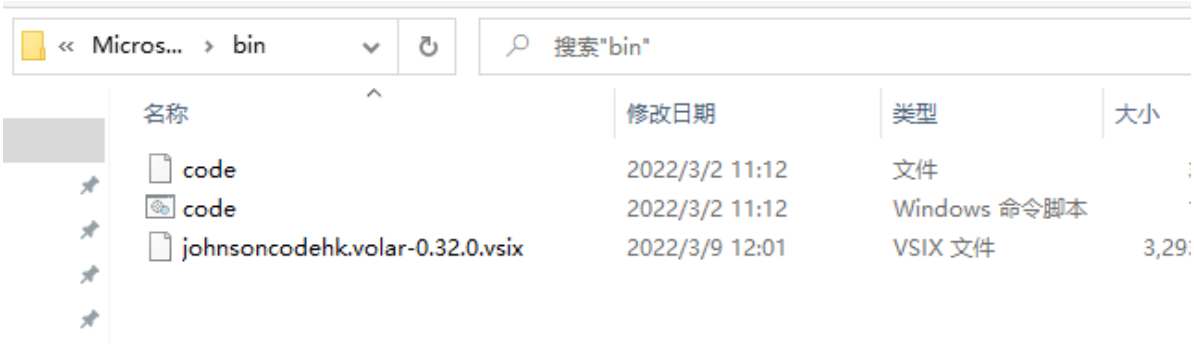
然后下载对应的版本,这里我下载的是volar 0.32.0



2 卸载已经安装的volar

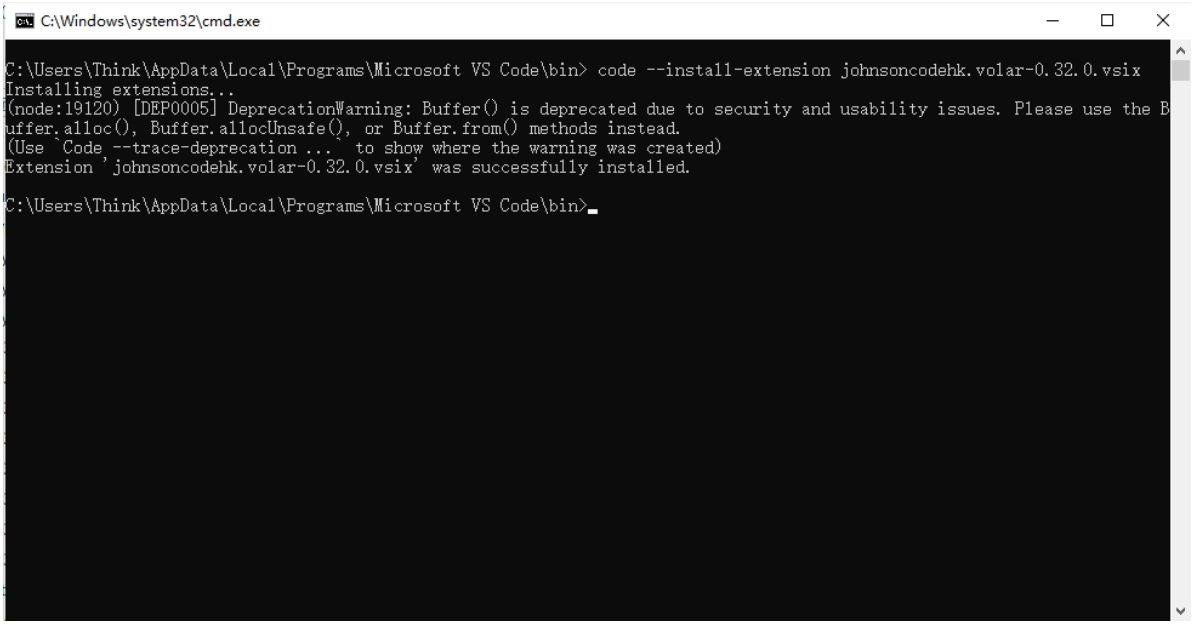


3 把下载下来的离线安装包拷贝到 VSCode 的安装目录下的 bin 目录下，比如我的 VSCode 安装在 `D:\Microsoft VS Code\`，因此这里我应该拷贝到 `C:\Microsoft VS Code\bin` 这个目录下

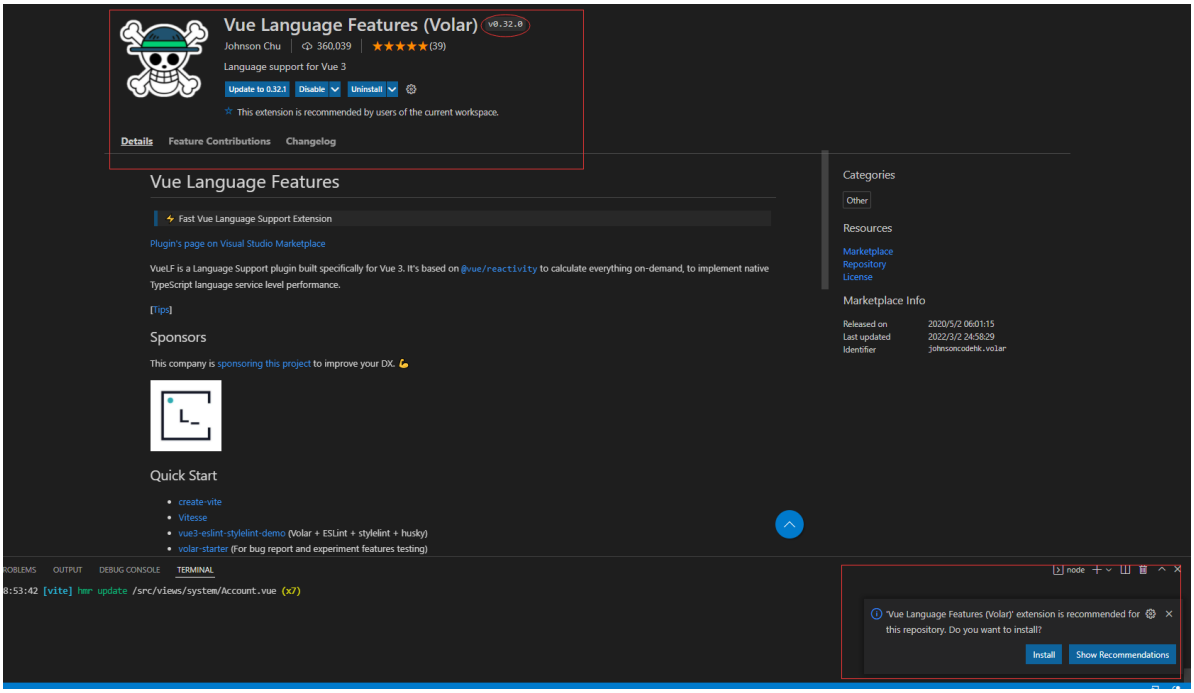


4 输入命令安装

```
code --install-extension johnsoncodehk.volar-0.32.0.vsix
```



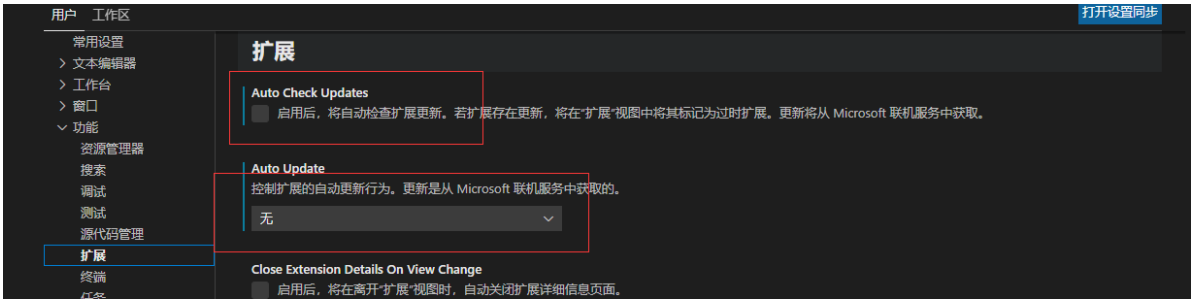
这样版本降级就成功了,提示更新暂时不理,等稳定点的版本再说



这样我的报错信息就消失了

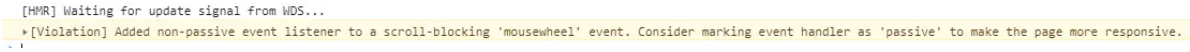
```
      <el-form-item>
        <el-button @click="resetForm">重置</el-button>
        <el-button type="primary" @click="handelConfirm">确定</el-button>
      </el-form-item>
    </el-form>
  </el-dialog>

  <!--选择角色弹窗-->
  <el-dialog title="设置角色" v-model="state.showSetRoleDialog">
    <el-select @change="upRole" value="选择角色" placeholder="选择角色">
      <el-option v-for="item in roles" :key="item.id" :label="item.name" :value="item.id"></el-option>
    </el-select>
  </el-dialog>
  <!-- 用户table表格 -->
  <div style="margin: 0px 10px;text-align: left;">
    <el-table
      :data="state.users.slice((state.currentPage - 1) * state.pageSize, state.currentPage * state.pageSize)"
      stripe
      :default-sort="{ prop: 'username', order: 'ascending' }"
    >
      <el-table-column prop="username" label="用户名" sortable></el-table-column>
      <el-table-column label="角色" prop="roleName"></el-table-column>
      <el-table-column label="状态">
        <template #default="scope">
          <el-switch
            v-model="scope.row.status"
            active-color="green"
            inactive-color="red"
            @change="(value:boolean) => commitStatusChange(value, scope.row)"
          ></el-switch>
        </template>
      </el-table-column>
      <el-table-column label="操作" width="280px">
```

取消扩展更新,『文件』>>『首选项』>>『设置』>>『功能』>>『扩展』，取消勾选下面两项:



bug2 vue中警告[Violation] Added non-passive event listener to a scroll-blocking event.

引入echarts地图组件报错



原因是 Chrome51 版本以后，Chrome 增加了新的事件捕获机制 - Passive Event Listeners；

Passive Event Listeners：就是告诉前页面内的事件监听器内部是否会调用preventDefault函数来阻止事件的默认行为，以便浏览器根据这个信息更好地做出决策来优化页面性能。当属性passive的值为true的时候，代表该监听器内部不会调用preventDefault函数来阻止默认滑动行为，Chrome浏览器称这类型的监听器为被动（passive）监听器。目前Chrome主要利用该特性来优化页面的滑动性能，所以Passive Event Listeners特性当前仅支持mousewheel/touch相关事件。

解决方法

```
1.npm i default-passive-events -S

2.main.ts中加入：import 'default-passive-events'
```

bug3 警告您正在运行 vue-i18n 的 esm-bundler 构建

如果是js

可以引入stackoverflow的解决方法

https://stackoverflow.com/questions/66140411/you-are-running-the-esm-bundler-build-of-vue-i18n-it-is-recommended-to-configur

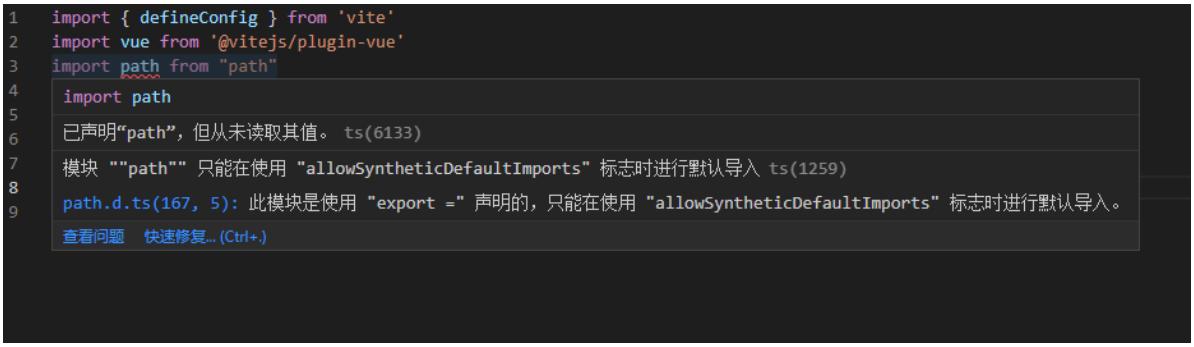
在vite中添加

```
    define: {
        __VUE_I18N_FULL_INSTALL__: true,
        __VUE_I18N_LEGACY_API__: false,
        __INTLIFY_PROD_DEVTOOLS__: false,
    },
```

这显然是一个已知的错误。他们说它将在 9.2 版本中修复。

https://github.com/intlify/vue-i18n-next/issues/789

bug4 模块 “path“只能在使用 “allowSyntheticDefaultImports“ 标志时进行默认导入



解决办法

- 在 `tsconfig.node.json` 文件中
- 添加 `"allowSyntheticDefaultImports": true`

之前提示找不到path,安装@type/node即可,后来vite优化了,多了对node的配置tsconfig.node.json

所以直接配置即可解决

bug5[Violation] Added non-passive event listener to a scroll-blocking ‘mousewheel‘ event.

原因是 Chrome51 版本以后，Chrome 增加了新的事件捕获机制 - Passive Event Listeners；

Passive Event Listeners：就是告诉前页面内的事件监听器内部是否会调用preventDefault函数来阻止事件的默认行为，以便浏览器根据这个信息更好地做出决策来优化页面性能。当属性passive的值为true的时候，代表该监听器内部不会调用preventDefault函数来阻止默认滑动行为，Chrome浏览器称这类型的监听器为被动（passive）监听器。目前Chrome主要利用该特性来优化页面的滑动性能，所以Passive Event Listeners特性当前仅支持mousewheel/touch相关事件。

解决办法：

上线后报错消失,但是开发环境还是存在,尤其elementplus ui存在等待修复

1.安装 default-passive-events

```
npm i default-passive-events -S
```

2.main.ts全局引入

```
import 'default-passive-events';
```

Typescript教程

1typescript简介

什么是typescript?

TypeScript 是一种由微软开发的自由和开源的编程语言。它是 JavaScript 的一个超集，而且本质上向这个语言添加了可选的静态类型和基于类的面向对象编程。

Typescript 为 Javascript 增加类型能力，主要为了避免 JS 弱类型下产生的各种有意无意的问题。或者说，TS 用来限制 JS 的编写，就像一个 lint 一样

如果TypeScript能够解决你的问题，那就用吧,很多人都受益于TypeScript；如果你觉得TypeScript不能带来好处，那不用也行。

为什么要使用typescript?

- 您可以避免经典的错误 'undefined' is not a function.
- 在不严重破坏代码的情况下，重构代码更容易。
- 使大型、复杂的应用程序源码更易阅读。

1

首先要分清楚，强类型和弱类型、静态类型和动态类型是两组不同的概念，类型强弱是针对类型转换是否显示来区分，静态和动态类型是针对类型检查的时机来区分。

TS对JS的改进主要是静态类型检查，静态类型检查有何意义？标准答案是“静态类型更有利于构建大型应用”。

静态类型检查可以做到提前发现错误，即你编写的代码即使没有被执行到，一旦你编写代码时发生类型不匹配，语言在编译阶段（解释执行也一样，可以在运行前）即可发现。针对大型应用，测试调试分支覆盖困难，很多代码并不一定能够在所有条件下执行到。而假如你的代码简单到任何改动都可以从UI体现出来，这确实跟大型应用搭不上关系，那么静态类型检查确实没什么作用。

2

公司现在的前端项目大概有十几万行代码，各种从后端拿到的数据类型有上百种
以前后端接口一改，要改字段，瞬间懵逼。
全局搜索，一个个改，各种牵扯到的东西改下来再测试一顿估计小半天没了。
用了 TypeScript 之后，把数据对应的 interface 改掉，然后重新编译一次，把编译失败的地方全部改掉就好了。

而且在优秀的 TypeScript 架构中，业务开发基本不需要写类型，所有外部输入的类型都可以自动拿到，只需要把一些 local variable 和 output 的类型定义一下就好了，基本跟手写 ES 6 没有区别。

写代码的过程中各种错误在越早期修改的成本就越低。试想没有静态检查跑一遍代码进某个奇怪的 case 才能复现的错误在写代码时期就直接给你的个错误提示，将是多么省时省力省钱。

#

2typescript数据类型声明

首先在电脑安装了node 和typescript

```
npm i -g typescript
```

检测版本

```
tsc -v
```

了解下typescript数据类型和javascript有何不同?

因为typescritpt是javascript的超集

- var
- let
- const

typescript官网建议let 或者const

声明方式

```
let a:string; //直接声明类型
let a:string = "helloword"; //声明类型又声明值
let a = "helloword"
```

函数传参类型声明和函数返回值类型声明

```
function fna(a:number,b:number) :number{
    return a + b
}
```

3typescript数据类型

数据类型	举例
Boolean 类型	let isDone: boolean = false; // ES5：var isDone = false;
Number 类型	let count: number = 10; // ES5：var count = 10;
String 类型	let name: string = "Semliker"; // ES5：var name = 'Semlinker';
Array 类型	let list: number[] = [1, 2, 3]; // ES5：var list = [1,2,3]; let list: Array = [1, 2, 3]; // Array泛型语法 // ES5：var list = [1,2,3];
null	null代表不存在
undefinded	undefined 代表变量未被初始化
字面类型	let a:10; //既限制类型又限制初始值 let a: "hello" | "word"; let a: string | number;// 通常我们用或来使用字面类型
any	在 TypeScript 中，任何类型都可以被归为 any 类型。这让 any 类型成为了类型系统的顶级类型（也被称作全局超级类型）
unknown	就像所有类型都可以赋值给 `any`，所有类型也都可以赋值给 `unknown`。这使得 `unknown` 成为 TypeScript 类型系统的另一种顶级类型（另一种是 `any`）
Tuple	元组是 TypeScript 中特有的类型，其工作方式类似于数组。元组可用于定义具有有限数量的未命名属性的类型。每个属性都有一个关联的类型。使用元组时，必须提供每个属性的值。
Void	某种程度上来说，void 类型像是与 any 类型相反，它表示没有任何类型。当一个函数没有返回值时，你通常会见到其返回值类型是 void
Never	类型表示的是那些永不存在的值的类型。 例如，`never` 类型是那些总是会抛出异常或根本就不会有返回值的函数表达式或箭头函数表达式的返回值类型。
enum	使用枚举我们可以定义一些带名字的常量。 使用枚举可以清晰地表达意图或创建一组有区别的用例。 TypeScript 支持数字的和基于字符串的枚举。

3.1 boolean number string

```
let isDone: boolean = false; // ES5: var isDone = false;
let count: number = 10; // ES5: var count = 10;
let name: string = "Semliker"; // ES5: var name = 'Semlinker';
```

3.2 字面类型

限定类型的同时又限定值

```
let s:10

s =10
let s: "hello"| "hi"
s = "hello"
s = "hi"
```

3.3 any和unknown的区别

any任意定义

unknown是一个类型安全的any

unknown不能直接赋值给其它变量

```
let a:any;
a = "hello";
a = 10
let c:unknown;
c = true;

let b:string;

b = a
b = c
```

##

3.4 Array数组和object对象

array两种表达式

```
let list: number[]
let list: Array<number>
```

object对象

```
let person:object; // 过于any
let person:{name:string,age:number} //对属于限定数据类型
```

对象解构

```
let person = {
  name: "wong",
  gender: "Male",
};

let { name, gender } = person;
```

对象展开运算

```
let person = {
  name: "Semlinker",
  gender: "Male",
  address: "Xiamen",
};

// 组装对象
let personWithAge = { ...person, age: 33 };

// 获取除了某些项外的其它项
let { name, ...rest } = person;
```

3.5 void 和never

void表示函数返回为空

比如console.log或者null和undefinded

返回undefined 代表变量未被初始化

null代表不存在

```
function fnb():void {
    return null
}
```

never表示永远不会有返回值连空都没有,同样顾名思义，表示**永不存在的值**的类型,概念有点绕，什么情况下变量会永远不存在值呢？因为通常情况下，变量一旦申明了，就会被分配值，即使没有特别指定，也会被初始化为 `undefined`，同样一个函数即使有个写返回值，也会默认返回 `undefined`，也不是真正的不存在返回值：

其实确实有一些情形，值会永不存在，比如，从程序运行的维度讲，如果一个函数执行时抛出了**异常**，那么这个函数变永远不会有值了（因为抛出异常会直接中断程序运行，这样程序就运行不到返回值那一步了，即具有不可达的终点，也就永不存在返回了

```
let foo:string;
console.lgo(typeof foo)
```

```
function fnc():never {
    throw new Error("报错停止了")
}
```

或者函数无限循环的(死循环),这样也同样使得程序永远无法运行到函数返回值那一步

```
 while (true) {};
```

3.6 tuple

元组 固定长度的数组

定义元组类型

```
let tupleType: [string, boolean];
tupleType = ["Semlinker", true];
```

可选的元组类型

```
let tup1: [string, number?, number?]; tup1 = ['a']; tup1 = ['a', 1]; tup1 = ['a',
1, 3]
```

操作元组

操作元组可以使用push()和pop()添加和删除元组最后一个元素

```
let mytup:[number, number, string, string] = [7, 36, 'summer', 'xkd'];
mytup.push('mark');
console.log(mytup);

// 输出：[ 7, 36, 'summer', 'xkd', 'mark' ]
mytup.pop();
console.log(mytup);
```

更新或修改元组元素

元组可以修改 索引和运算符 "="来修改元组中的元素

```
let mytup:[number, string] = [7, 'xkd'];
mytup[1] = 'aaa';
console.log(mytup);
```

解构元组

```
let mytup:[number, string, boolean] = [7, 'summer', true];
let [a, b, c] = mytup;
console.log(a);
console.log(b);
console.log(c);
```

如果不需要全部,我们直接取其中一个

```
let mytup:[number, string, boolean] = [7, 'summer', true];
let [a, ...arg] = mytup;
console.log(a);
console.log(arg);
```

3.7枚举类型

枚举就是“各种情况”的语意化，例如我传0表示增加，1表示更新，2表示删除。传012或用012来判断也可以，但是表达意义不明确。所以把它们赋值各一个枚举对象的new update delete等字段，然后用它们来参与比较判断，就会跟清晰一点。再补充一点，后台传枚举的各个情况都对应一个数字。

```
enum orderInfo {
    new = 0,
    update =1,
    delete =2
}

let order:number;
if(order === orderInfo.new) {
```

3.8 类型别名

类型别名会给一个类型起个新名字。 类型别名有时和接口很像，但是可以作用于原始值，联合类型，元组以及其它任何你需要手写的类型。起别名不会新建一个类型 - 它创建了一个新名字来引用那个类型

```
type Name = string;
let a:Name;
a = "sss"
```

使用这个方法我们可以自由定义数据类型

```
type Name = 1|2|3|4;
let a:Name;
a = 5
```

4 typescript编译

4-1 typescript.json配置

之前,我使用的是tsc XXX.ts 编译单个文件和tsc XXX.ts -w监控单个文件.

我们可以使用命令直接创建tscconfig.json

```
tsc --init
```

现在有一个问题,我如果创建的文件多了,

如果全部文件进行编译监控?

在目录下创建tsconfig.json文件,

直接tsc -w就可以编译监控所以ts后缀结尾的ts文件

inculde：对test1下的路径文件进行编译 设置好后只用tsc即可编译一个代表文件 **代表目录*,inculde: ["./src//***"]就是对src下的所有子目录下的文件进行编译

exculde：与inculde相反就是忽略哪些文件 对test1下的index.ts不进行编译

extends: 继承其它ts .json配置选项

files：指定编译的文件列表 对test1下的index2和index3进行文件编译

4-2compilerOptions编译选项

名称	作用
target	是把ts转成什么版本的js 如'es3', 'es5', 'es6', 'es2015', 'es2016', 'es2017', 'es2018', 'es2019', 'es2020', 'esnext'。下列例子就是将ts代码转成es5版本的js代码
lib	选择宿主环境
outdDir	将编译后的js输出到哪个目录 默认与编译的ts文件同级
outFile	将ts文件合并编译成一个文件
module	编译用的模块方式有'none', 'commonjs', 'amd', 'system', 'umd', 'es6', 'es2015', 'es2020', 'esnext'.但是当有outFile属性后 只支持'amd', 'system'方式

4-3 编译语法配置

编译js文件并且有报错提示

- **allowJs**允许javascript文件被编译
- **checkJs**报告.js文件中的错误
-

removeComments控制添加注解

这个用于**指定编译输出文件中是否删除源文件中的注释**，默认为false，即不删除，如果设置为true，则会在输出文件中清除所有注释。

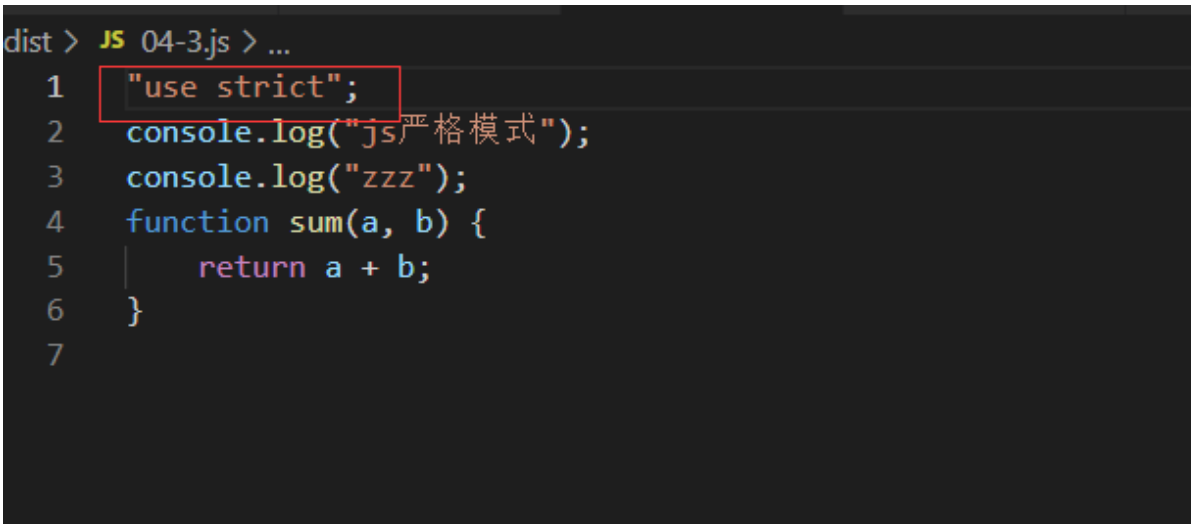
noEmit

不生成输出文件

strict模式名称	用途
alwaysStrict	是否开启严格模式
noImplicitAny	是否开启隐式any
noImplicitThis	能否指定明确的this
strictNullChecks	检测是否为空
strict	严格模式总开关

alwaysStrict

控制strict严格模式的使用



noImplicitAny

设置为true，代表开启any类型检测

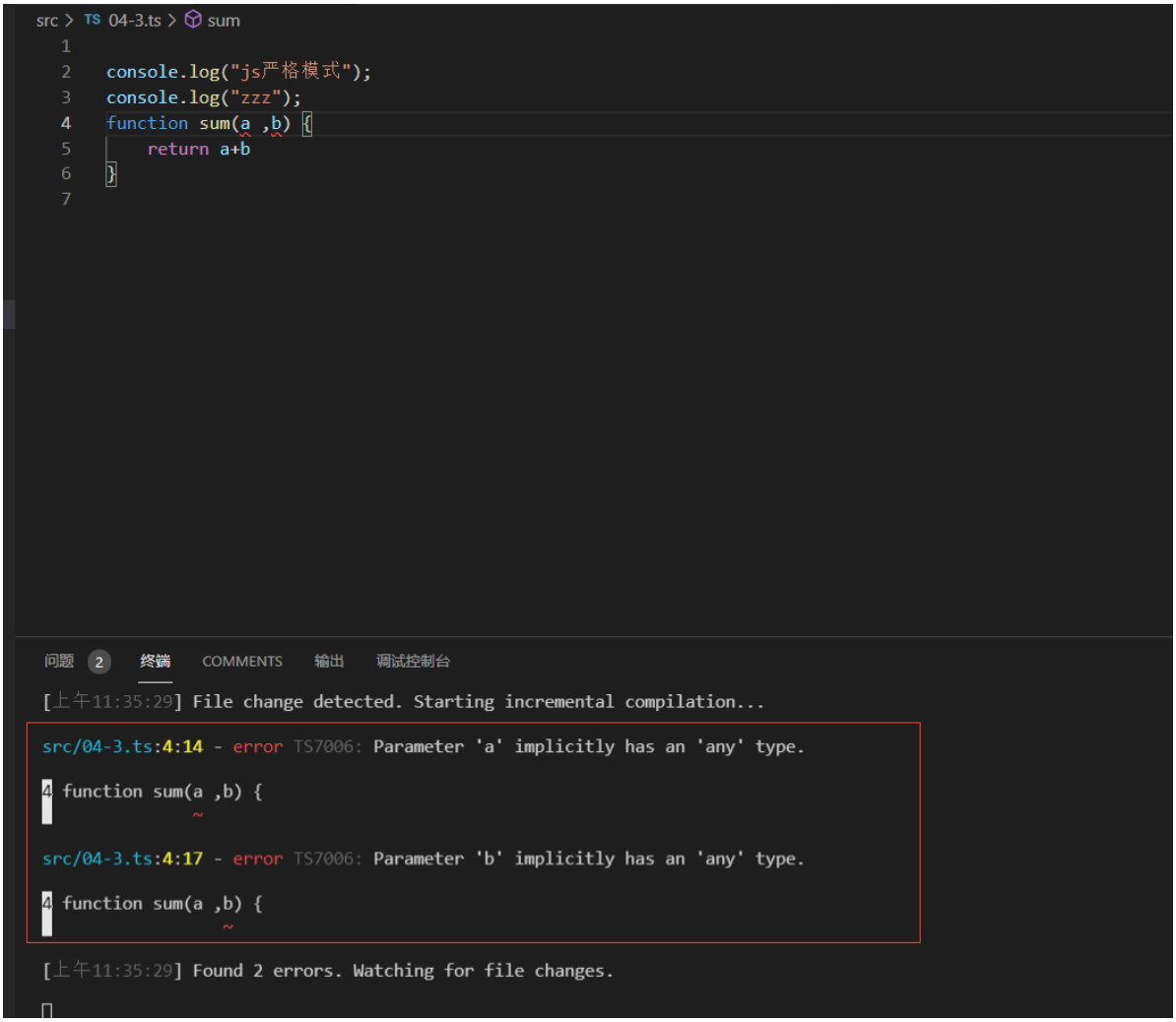
```
/* Strict Type-Checking Options */
"strict": false,                                    /*
"noImplicitAny": true,                            /* Rai
// "strictNullChecks": true,                        /*
// "strictFunctionTypes": true,                     /*
// "strictBindCallApply": true,                     /*
// "strictPropertyInitialization": true,            /*
// "noImplicitThis": true,                          /*
"alwaysStrict": true,                             /* Par
```

当表达式和申明 类型为any时，是否需要发出警告

```
1
2   console.log("js严格模式");
3   console.log("zzz");
4   function sum(a:number ,b:number) {
5       return a+b
6   }
7
```

```
问题  终端  COMMENTS  输出  调试控制台
[上午10:26:42] File change detected. Starting incremental compilation...

[上午10:26:42] Found 0 errors. Watching for file changes.
```

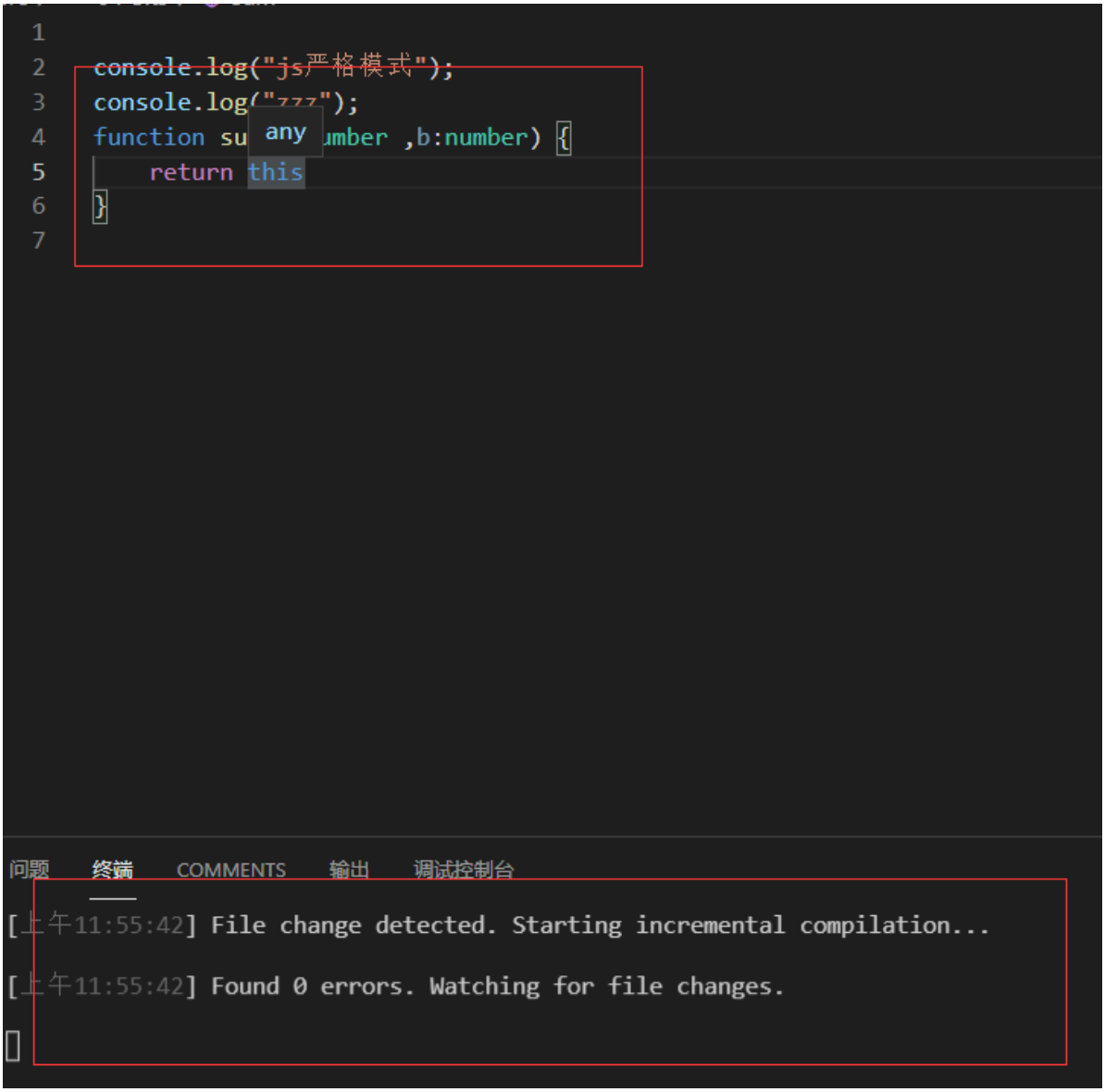
如果未申明类型，终端编译就会报错



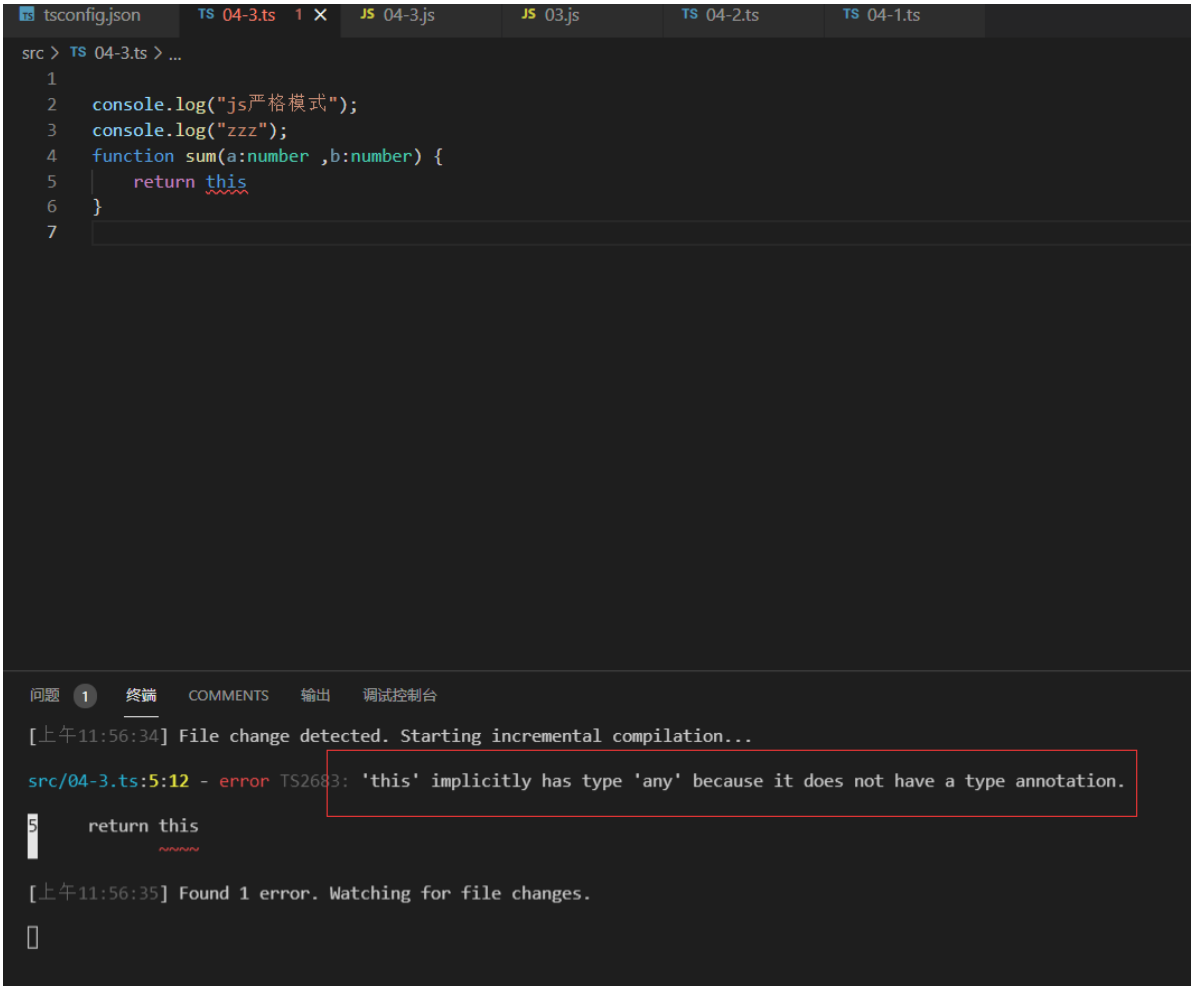
noImplicitThis

这个是用于**控制当源文件中存在this的值是any的时候是否报错**

false不报错


1
2 console.log("js严格模式");
3 console.log("zzz");
4 function su any mber ,b:number) {
5 return this
6 }
7
问题 终端 COMMENTS 输出 调试控制台
[上午11:55:42] File change detected. Starting incremental compilation...
[上午11:55:42] Found 0 errors. Watching for file changes.

true就会报错



strictnullcheck

空指针是最常见的bug之一，而通过`strictnullcheck` TypeScript编译器标志可以在很大程度上避免空指针。

```
function xx(): number {
    return null;
}
```

5 typescript类 接口 泛型

5-1 typescript类

什么是面相对象?

```
对象的概念
        现实世界中客观存在的事物就是对象。


    类的概念
        把一组对象相同的属性和相同的行为抽象出来就形成了类。类是对一组对象相同属性和相同行为的描
述。
```

面向对象是一种 **对现实世界理解和抽象的方法**，是计算机编程技术发展到一定阶段后的产物。

面向过程 (Procedure Oriented) 是一种 **以过程为中心** 的编程思想。这些都是以什么正在发生为主要目标进行编程，不同于面向对象的是谁在受影响。与面向对象明显的不同就是 **封装**、**继承**、**类**。

无论是在软件开发还是在实际工作中，深入地理解软件开发的思想都非常有必要。

类的简介和使用

TypeScript 是面向对象的 JavaScript。

类描述了所创建的对象共同的属性和方法。

```
class Person {
    name:string = "Ak";
    age:number = 18;
    join(b:number):number {
        return this.age + b
    }
}

const ps = new Person()

console.log(ps.name);
console.log(ps.join(2));
```

一般会实例化使用属性和函数，但是也能使用类属性和类函数

5-2构造函数

主要用来在创建对象时初始化对象， 即为对象成员变量赋初始值，总与*new*运算符一起使用在创建对象的语句中

在类中直接加constructor

```
class Person1 {
    name:string;
    age:number;

    constructor(name:string,age:number) {
        console.log("执行构造函数...")
        this.name = name
        this.age = age
    }
}

const ps1 = new Person1("小明",18)
const ps2 = new Person1("小花",17)

console.log(ps2);class Gun {
    bullet:number;
    constructor(b:number) {
        this.bullet = b;
    }

}
const gunAk = new Gun(44);
console.log(gunAk.bullet);
```

5-3 类的继承extends和super

1. 子类通过 extends 继承父类
2. 子类可以重新定义父类中的方法进行重写

首先，为何要使用extends继承，比如我写两个类，

一个dog一个snake,然后实例化引用，这是没有什么问题的，

但是除了蛇 和 狗，还有其它的动物，

如果都写一个类，代码就很臃肿，

而且，代码的同质性很高！

```
class Dog {
    name:string;

    constructor(name:string) {
        this.name = name
    }

    move(distance:number) {
        console.log("奔跑方式:");
        console.log(`${this.name} 移动了 ${distance} M`);


    }
}


class Snake {
    name:string;

    constructor(name:string) {
        this.name = name
    }

    move(distance:number) {
        console.log("爬行方式:");
        console.log(`${this.name} 移动了 ${distance} M`);


    }
}

const dog = new Dog("旺财");
const snake = new Snake("小白");
dog.move(10)
snake.move(5
```

怎么解决这个问题呢?

这时候，extends就能很好的复用，

复用的时候呢，我们使用super就能成功调用父类,

但是调用过程中,派生类中用了constructor,

会取代基类中的构造函数constructor,

这时候报错,提示基类中constructor无法使用,

在派生类中用super(name)重新引用constructor即可

```
class Animal {
    name:string;
    constructor(theName:string) {
        this.name = theName
    }
    move(distance: number = 0) {
        console.log(`${this.name} 移动 ${distance}m`);
    }
}
class Dog extends Animal {


    move(distance:number) {
        console.log('奔跑中。。。');
        super.move(distance)

    }
}

class Snake extends Animal {

    constructor(name:string) {
        super(name)
    }
    move(distance:number) {
        console.log("爬行中。。。");
        super.name = "狗子"
        super.move(distance)

    }
}

let dog = new Dog("旺财")
let snake = new Snake("python")
dog.move(10)
snake.move(20)
```

5-4类修饰符

private:私有 在当前类里面可以访问，子类、类外部都没法访问

public :公有 在当前类里面、 子类 、类外面都可以访问
protected：保护类型 在当前类里面、子类里面可以访问 ， 在类外部没法访问
private ： 私有 在当前类里面可以访问，子类、类外部都没法访问

```
function zzz() {
    //父类
class Animal{
    private name:string;
    protected sex:string;
    age:number;
    constructor(name:string,age:number,sex:string){
        this.name=name;
        this.age=age;
        this.sex=sex;
    }
    say(){
        return `我是${this.name}我${this.age}岁`
    }
}
//子类
class Dog extends Animal{
    constructor(name:string,age:number,sex:string){
        super(name,age,sex)
    }
    say1(){
        console.log(this.name);//报错 私有属性只能在它本类中使用
        console.log(this.sex);//正确 保护类型可以在子类中使用
        console.log(this.age);//正确
    }
}
var p=new Animal("张三",23,"男");
console.log(p.age);//正确，公有可以在本类，子类，类外部访问
console.log(p.name);//报错，私有的属性不能再类外部访问
console.log(p.sex);//报错，受保护类型只能在本类或者子类中访问

}
```

abstract 抽象

定义抽象类和抽象函数,

抽象函数只能定义在抽象类中,

并且不能有实体

子类必须给抽象方法进行重写

```
function zzz() {
    //父类
```

```
   abstract class Animal{
    private name:string;
    protected sex:string;
    age:number;
    constructor(name:string,age:number,sex:string){
        this.name=name;
        this.age=age;
        this.sex=sex;
    }
    abstract say():void
}
//子类
class Dog extends Animal{
    constructor(name:string,age:number,sex:string){
        super(name,age,sex)
    }
    say1(){
        console.log(this.name);//报错 私有属性只能在它本类中使用
        console.log(this.sex);//正确 保护类型可以在子类中使用
        console.log(this.age);//正确

    }

    say() {
        return `我是${this.name}我${this.age}岁`
    }
}

class Snake extends Animal {
    say() {

    }
}
var p=new Dog("张三",23,"男");
console.log(p.age);//正确，公有可以在本类，子类，类外部访问
console.log(p.name);//报错，私有的属性不能再类外部访问
console.log(p.sex);//报错，受保护类型只能在本类或者子类中访问

}
```

static

可以使用class修饰符static

类成员的静态属性我们可以直接调用

调用方式类属性的方式调用，this方法在内部也无法使用了

```
class Gun {
    static bullet:number = 10;

    shot(b:number):number {
        Gun.bullet = b
        return Gun.bullet
    }
}

const gunAk = new Gun();

console.log(Gun.bullet);
```

并且实例化对象中的属性都是可变的

但是如果在属性前添加readOnly就无法改变

5-5接口

interface和type的区别?

都可以描述一个对象或者函数

type

```
type User = {
  name: string
  age: number
};
```

interface

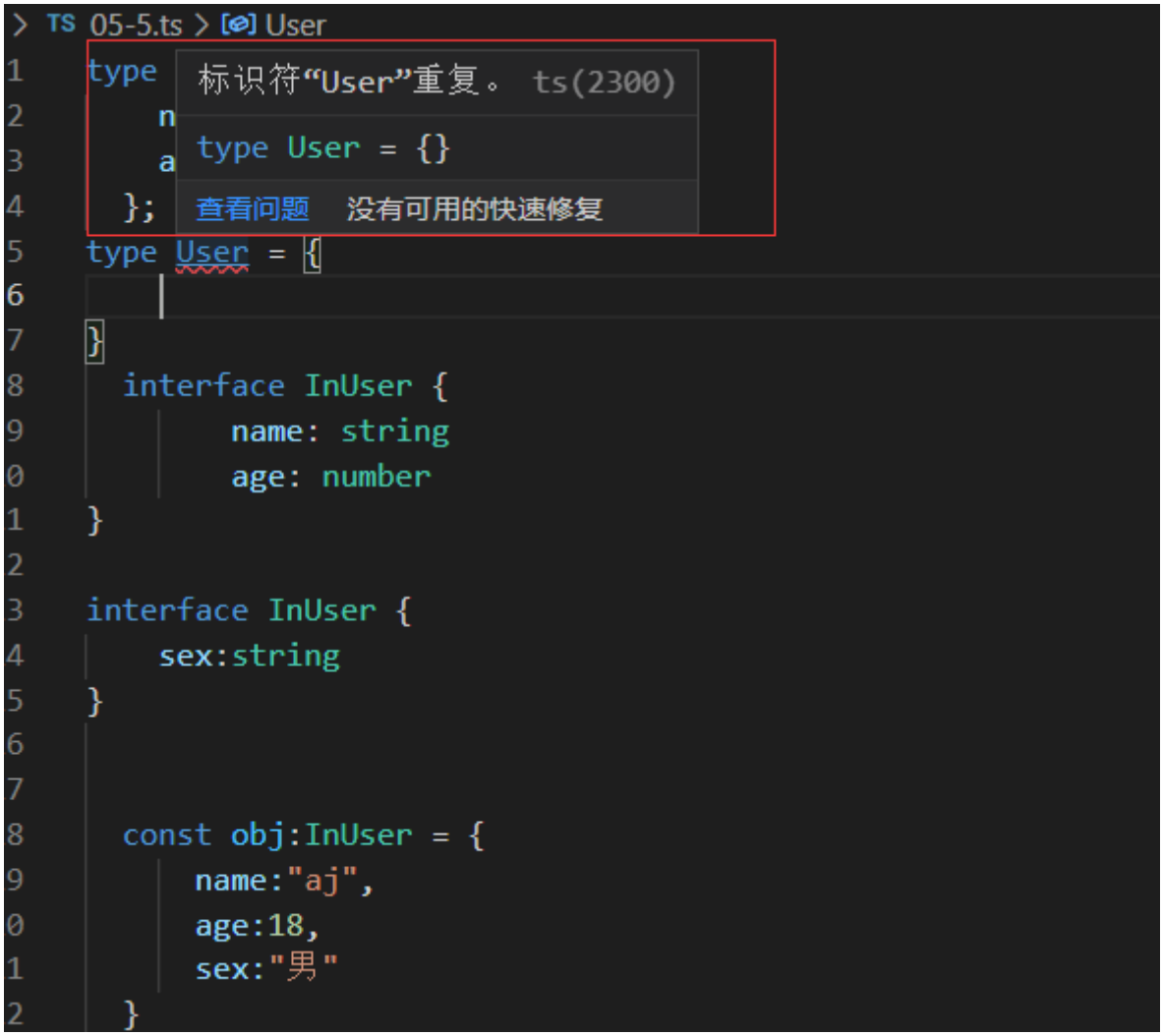
```
  interface InUser {
        name: string
        age: number
}
//用哪种都可以
 const obj:InUser = {
      name:"aj",
      age:18
  }
```

但是又有很多不同点

interface可以声明合并,而type只能类型联合

```
  interface InUser {
        name: string
        age: number
}

interface InUser {
    sex:string
} //此处声明同名称的接口进行合并


  const obj:InUser = {
      name:"aj",
      age:18,
      sex:"男"
  }
```

换成type类型联合,首先type是不能起重复的名称




```
type User = {
    name: string
    age: number
  };
type UserName = User & {
    sex:string
} //只能进行类型联合

  const obj:UserName = {
      name:"aj",
      age:18,
      sex:"男"
  }
```

interface可以继承 ,而type只能通过交叉类型 实现

```
}
  interface Users {
        name: string
        age: number
}

interface InUser extends Users {
    sex:string
} //这里InUser继承了Users
```

```
 const obj:InUser = {
     name:"aj",
     age:18,
     sex:"男"
 }
```

interface一般通过 implements限制类的实现

```
interface Myinter {
     name:string;
     move() :number;
 }

 class MyClass implements Myinter {
     name: string;
     move(): number {
        return 22
     }

 }
```

interface编译后js文件是看不到的

5-6泛型

比如一个普通的函数是这样定义

```
function fnz(arg: number): number {
    return arg;
}

function fnz(arg: any): any {
    return arg;
}
```

为何有数据类型,要使用泛型?

使用any会关闭类型检查,所以类型不明确的时候使用泛型,

如果我们传入一个数字，我们只知道任何类型的值都有可能被返回,

我们需要一种方法使返回值的类型与传入参数的类型是相同的

泛型该怎么用?

添加了类型变量`T`。 `T`帮助我们捕获用户传入的类型（比如：`number`），之后我们就可以使用这个类型。 之后我们再次使用了`T`当做返回值类型。现在我们可以知道参数类型与返回值类型是相同的了。这允许我们跟踪函数里使用的类型的信息。

```
function fnz<T>(arg: T): T {
    return arg;
}
```

1.利用到了我们ts中的类型推论,即编译器会根据传入的参数自动地帮助我们确定T的类型

```
function fnz<T>(arg: T): T {
    return arg;
}

fnz(12) //number类型
```

2.可以使用传入所有的参数，包含类型参数

```
function fnz<T>(arg: T): T {
    return arg;
}

fnz<number>(12) //number类型
```

泛型可以用在函数 接口 类,并且泛型可以继承

```
class funNumber<T> {
    zeroValue: T;
    add: (x: T, y: T) => T;
}

let myNumber = new funNumber<number>();
myNumber.zeroValue = 0;
```

```
myNumber.add = function(x, y) { return x + y; };

interface Length {
    length: number;
}

function fnz<T extends Length>(arg: T): T {
    console.log(arg.length);  // Now we know it has a .length property, so no
more error
    return arg;
}

fnz("19")
```

我们也可以使用不同的泛型参数名，只要在数量上和使用方式上能对应上就可以

```
function createMan<T,K>(name:T,age:K):[T,K]{
    return [name,age];
}
let result=createMan<string,number>("张三",30);
console.log(result[0],result[1]);//结果：张三 30
```

6 vite构建工具

两种打包工具:webpack和vite

> 目前线上项目都还是使用webpack为主,但是vite实在太快太香,基于目前主流vue cli已经很多教程,我这里使用vite2创建项目,构建后台管理系统
>
> Vite 需要 Node.js 版本 >= 12.0.0。

6-1 vite2创建项目

npm:

```
npm create vite
```

yarn:

```
yarn create vite
```

pnpm:

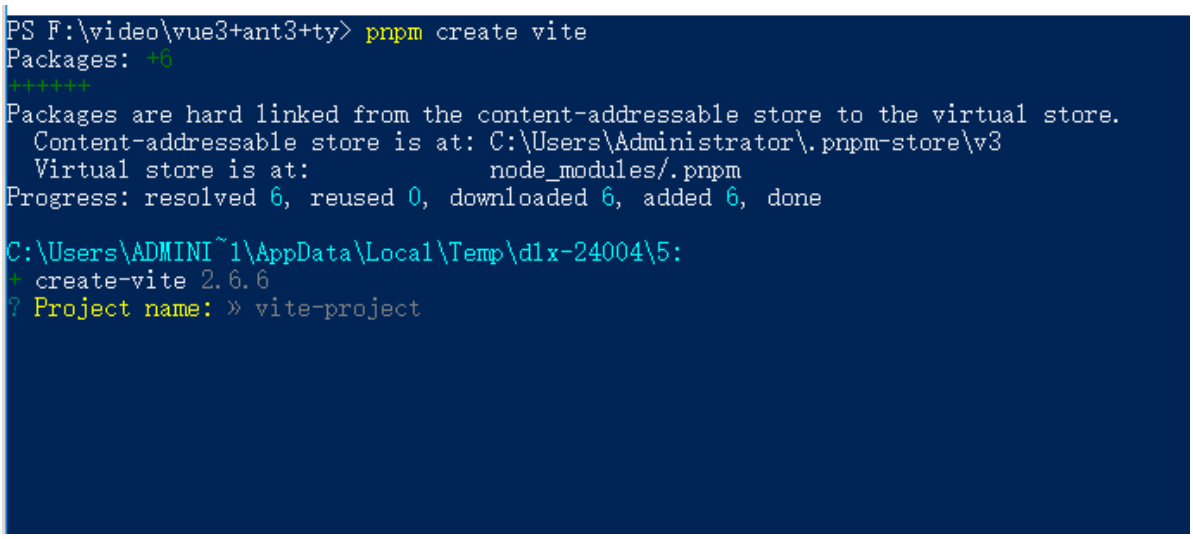
首先安装pnpm,安装完成后查看版本,使用命令与npm差不多

```
npm install -g pnpm
```

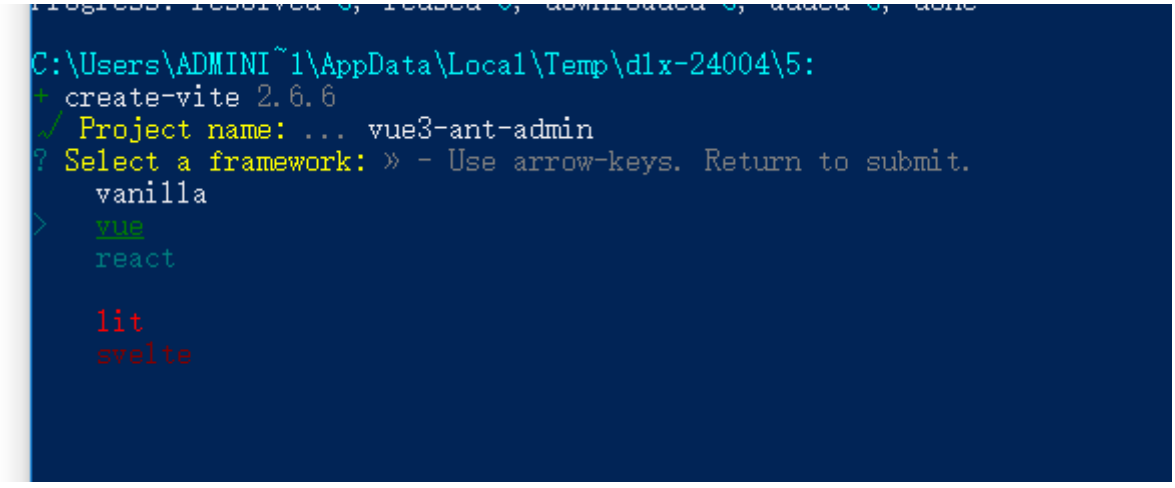
使用npm创建项目

```
npm create vite
```

输入项目名称



选择项目类型为 vue



使用typescript就选vue-ts

```
PS F:\video\vue3+ant3+ty> pnpm create vite
Packages: +6
++++++
Packages are hard linked from the content-addressable store to the virtual store.
  Content-addressable store is at: C:\Users\Administrator\.pnpm-store\v3
  Virtual store is at:             node_modules/.pnpm
Progress: resolved 6, reused 0, downloaded 6, added 6, done

C:\Users\ADMINI~1\AppData\Local\Temp\dlx-24004\5:
+ create-vite 2.6.6
√ Project name: ... vue3-ant-admin
√ Select a framework: » vue
? Select a variant: » - Use arrow-keys. Return to submit.
    vue
>   vue-ts
```

```
npm i

npm run dev
```

```
Scaffolding project in F:\video\vue3+ant3+ty\vue3-ant-admin...

Done. Now run:

  cd vue3-ant-admin
  npm install
  npm run dev
```

6-2 volar替换vetur

什么是volar?

volar是我们vue3开发终极神器,可以说 Volar 是 Vetur 的继任者,

除了给我们提供代码高亮,语法提示!

提示

必须先禁用或者卸载vetur

功能

volar最明显功能具有拆分功能

6-3 vite配置

vite2新特性

别名去斜杆

多框架支持react

全新插件支持

那vite2该如何配置?

首先参考官网:

https://cn.vitejs.dev/config/#config-intellisense

官网的配置 琳琅满目

但是不用担心

我们只需要了解主要几个配置

root

alias

Esbuild

引入path

如果报错是我们编辑器无法识别node

安装下@type/node

它和ts有关该包用于在 node.js 中使用 typescript 时加载所有类型定义。添加其他包时，如果默认情况下不包含它们，则还必须添加它们的类型。

```
npm install @types/node --save-dev
```

这样import path from "path";就不会报错提示找不到path

记得ctrl+shift+p -> reload重新加载编辑器

这时我们可以根据path的绝对路径来配置别名alias,并且配置端口号

```
import { defineConfig } from 'vite'
import vue from '@vitejs/plugin-vue'

import path from "path";

// https://vitejs.dev/config/
export default defineConfig({
  server:{

    port:8001
  },
  resolve: {
    alias: {
```

```
      "@": path.resolve(__dirname, "src"),
      "com": path.resolve(__dirname, "src/components")
    }
  },
  plugins: [vue()]
})
```

如果是vite新版添加了tsconfig.node.json,可以直接添加

```
allowSyntheticDefaultImports": true
```

解决办法

- 在`tsconfig.node.json`文件中
- 添加`"allowSyntheticDefaultImports": true`

7 vue3.2新特性

7-1 script setup语法糖解读

以前写响应式数据,ref reactive computed,我们这样写

```
<template>
    <div>{{ count }}</div>
    <div>{{ obj.a }}</div>
    <div>{{sum}}</div>
</template>

<script lang="ts">

import { ref, reactive, computed } from 'vue'


export default {
  setup() {
    const count = ref(1)
    const obj = reactive({
      a: 1,
      b: 2
    })
    const sum = computed(() => {
    return obj.a + 3;
});
    return {
      count,
      obj,
      sum
    }
  }
}



</script>

<style></style>
```

引入setup语法糖之后,一切变得更加简单

export default \ setup \ return都不需要了,

组件在编译的过程中代码运行的上下文是在 `setup()` 函数中，无需 `return`， `template` 可直接使用。

```
<script setup lang="ts">
```

```
import { ref, reactive, computed } from 'vue'


    const count = ref(5)
    const obj = reactive({
      a: 1,
      b: 2
    })
    const sum = computed(() => {
    return obj.a + count.value;
});




</script>
```

之前我们注册一个component组件

```
<script lang="ts">
import ScriptSetup from './setup/ScriptSetup.vue'
export default {
  components:{
    ScriptSetup
  }
}


</script>

<template>
  <img alt="Vue logo" src="./assets/logo.png" />
  <ScriptSetup  />

</template>
```

并且在语法糖中,引入的组件可以直接使用,无需注册,ScriptSetup.vue被引入后就无需注册

```
<script setup lang="ts">
import ScriptSetup from './setup/ScriptSetup.vue'


</script>

<template>
  <img alt="Vue logo" src="./assets/logo.png" />
  <ScriptSetup  />

</template>
```

7-2 defineProps

首先,之前vue3下props

app.vue 父组件

在父组件赋值

```
<script setup lang="ts">

// import SetupScript from 'com/SetupScript.vue'
import DefineProps from 'com/DefineProps.vue'

</script>

<template>
  <img alt="Vue logo" src="./assets/logo.png" />
  <!-- <SetupScript msg="Hello Vue 3 + TypeScript + Vite" /> -->
  <DefineProps msg="hello,setup" :ageList=[18,19,20] />
</template>

<style>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

ScriptSetup子组件

props是用于状态的父传子,

在子组件中需要申明,才能取值

```
<script lang="ts">
export default {
    props:{
        msg:{
            type:String,
            required:true,
            default:"hello"
        },
        ageList:{
            type:Array,
            required:true
        }
    }
}
</script>

<template>
    <h1>{{msg}}</h1>
    <div v-for = "(item,index) in ageList" :key=index>
```

```
        <li>{{item}}</li>
    </div>
</template>

<style>
</style>
```

`props` 在语法糖中使用 `defineProps` 方法来使用：

子组件 ScriptSetup.vue：

```


import { ref, reactive, computed } from 'vue'

     defineProps({
        msg:{
            type:String,
            requried:true,
            default:"hello"
        },
        ageList:Array
    })
    const count = ref(5)
    const obj = reactive({
      a: 1,
      b: 2
    })
    const sum = computed(() => {
    return obj.a + count.value;
});


```

definePropsl也给了我们纯类型props声明方式

```
defineProps<{
        msg?:String ,
    ageList:Array<number>
    }>()
```

只能是要么使用运行时声明，要么使用类型声明。同时使用两种声明方式会导致编译报错。

仅限类型的 `defineProps` 声明的不足之处在于，它没有可以给 props 提供默认值的方式。为了解决这个问题，提供了 `withDefaults` 编译器宏

```
  const props =   withDefaults(defineProps<{
              msg?:string
        ageList?:Array<number>
  }>(),{
        msg:"hello",
        ageList:() =>[1,2]

    })
```

7-3 defineEmits

子组件向父组件事件传递

因为defineEmits也是宏命令,所以无需导入,

使用方式:

普通声明方式和类型声明方式

普通写法

```
<template>
    <button @click="clickEmits">子传父</button>
</template>

<script setup lang="ts">


    const emits = defineEmits(['parentClick'])

    const clickEmits = ()=>{
        emits('parentClick',"我是子组件")
    }
</script>

<style>
</style>
```

父组件

```
<script setup lang="ts">
import ScriptSetup from './setup/ScriptSetup.vue'
import DefineEmits from './setup/DefineEmits.vue'
const parentClick = (e:string) =>{
  alert(e)
}

const parentChange =(m:string) =>{
  alert(m)
}

</script>

<template>
  <img alt="Vue logo" src="./assets/logo.png" />
  <ScriptSetup    />
  <DefineEmits @parentClick= "parentClick" @parentChange="parentChange" />

</template>

<style>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
```

```
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

类型声明

```
    const emits = defineEmits<{
        (e:"parentClick",data:string):void
        (e:"parentChange",data:string):void
    }>()
```

7-4 useSlots 和 useAttrs

相同点：

都是发生在父子组件之间的关系；

都是为了应对父组件调用子组件的场合；

区别一：设计思想
props的设计思想是传递状态，将数据驱动组件的思想贯彻到底，子组件的渲染取决于父组件传递的数据；
slot的设计思想是传递DOM节点，将父组件的模板代码节点直接传递给子组件的某个slot，来达到最终渲染的目的；
区别二：作用范围
父组件在调用子组件的时候申明并赋值变量，那么子组件内将其加到props列表后，就可以接收并使用，但是一旦传递过来，作用域就发生变化；
子组件虽然为父组件预留slot，但是slot的作用域依然属于父组件，所以可以访问到父组件内的所有状态；
个人认为：
slot是 比props更高级的封装，通常子组件的调用是为父组件服务的，所以使用插槽这么个形象的概念，为父组件预留平台，使得父组件可以直接在子组件中熏染自己的内容；
渲染父组件的内容到子组件，slot比props更直接，更省力，更符合组件的设计思想，既然是父组件的内容表达，为何不在父组件里加工好直接放入子组件渲染，而是通过props变量去传递接收，进而由子组件去代加工渲染呢?
如果使用了一个别人封装好的子组件，那么其内容源码一般是不建议修改的，因为它可能还会复用到其他地方，容易出现一改俱改的局面，所以子组件的封装让其失去了灵活性，所以，使用slot插槽可以很大的发挥父组件的自定义的能力，子组件不会干涉slot存放的任何内容，而props做不到这一点，必须传递子组件内封装好的prop变量，不可更改和自定义；
作用域插槽的使用将父子组件的融合发挥到极致，为了应对有时候组件的渲染需要同时由父子组件分工协作共同完成，slot的默认作用域是调用它的父组件，但是使用slot-scope声明一个作用域变量，通过这个变量可以访问到子组件的所有状态（前提是必须知道），这样父组件在调用子组件的时候不但可以在slot中加工渲染自己的数据和模板，还可以加工渲染子组件的内的数据和模板，而且对子组件的数据是只读的，不会修改；

Vue插槽是一种将内容从父组件注入子组件的绝佳方法

为什么使用插槽?

简而言之，作用域内的插槽允许我们父组件中的插槽内容访问仅在子组件中找到的数据。 例如，我们可以使用作用域限定的插槽来授予父组件访问 `info` 的权限。

可以通过useContext从上下文中获取 slots 和 attrs。不过提案在正式通过后，废除了这个语法，被拆分成了useAttrs和useSlots。示例：

```
<script setup>
  import { useContext } from 'vue'

  const { slots, attrs } = useContext()
</script>

// 新
<script setup lang="ts">
    // 引入useSlots
```

```
    import { onMounted, useSlots} from 'vue'

    const slots = useSlots();
    onMounted(()=>{
        console.log(slots.header());

    })
</script>

<template>
    <h1>slots</h1>
    <slot name="header" />

</template>

<style>
</style>
```

父组件app.vue

```
<script setup lang="ts">
// import ScriptSetup from './setup/ScriptSetup.vue'
// import DefineEmits from './setup/DefineEmits.vue'

import SlotAttrs from './setup/SlotAttrs.vue'

</script>

<template>
  <img alt="Vue logo" src="./assets/logo.png" />
  <!-- <ScriptSetup    />
  <DefineEmits @parentClick= "parentClick" @parentChange="parentChange" /> -->
  <slot-attrs>
      <template #header >
        <text>test</text>
      </template>
  </slot-attrs>
</template>

<style>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

`useAttrs

`attrs` 是父组件传递给子组件的属性中除了`props`外的属性：

父组件

```
 <slot-attrs a="1" b="2" msg="hello" class="child" style="color:red">
      <template #header >
        <text>test</text>
      </template>
  </slot-attrs>
```

子组件

```
<script setup lang="ts">
    // 引入useSlots
   import { onMounted, useAttrs, useSlots} from 'vue'

    const slots = useSlots();
    const attrs = useAttrs();
    defineProps({
        msg:String
    })
    onMounted(()=>{
        console.log(slots.header&&slots.header());
        console.log(attrs)

    })
</script>
```

7-5 对外暴露属性(defineExpose)

在vue3.x的setup语法糖中定义的变量默认不会暴露出去，这时使用`definExpose({ })`来暴露组件内部属性给父组件使用

为了在 组件中明确要暴露出去的属性，使用 defineExpose 编译器宏：

```
<template>
    {{msg}}
</template>

<script setup>
import { ref } from 'vue'

let msg = ref("Child Components");
let num = ref(123);

// defineExpose无需导入，直接使用
defineExpose({
    msg,
    num
});
</script>
```

当父组件通过模板 ref 的方式获取到当前组件的实例，获取到的实例会像这样 `{ msg: number, num: number }` (ref 会和在普通实例中一样被自动解包

```
<script setup lang="ts">

// import SetupScript from 'com/SetupScript.vue'
// import DefineProps from 'com/DefineProps.vue'
// import DefineEmits from 'com/DefineEmits.vue'
// const parentClick = (msg:string) => {
//   alert(msg)
// }
// import SlotsAttrs from './components/SlotsAttrs.vue'

import DefineExpose from './components/DefineExpose.vue'
import { ref,onMounted } from 'vue';

let child = ref<{msg:string, handle:()=>void}|null>(null)

onMounted(() => {
  console.log(child.value?.msg);

  child.value?.handle()

})

</script>

<template>
  <img alt="Vue logo" src="./assets/logo.png" />
  <!-- <SetupScript msg="Hello Vue 3 + TypeScript + Vite" /> -->
  <!-- <DefineProps  :ageList=[18,19,20] /> -->
```

```
  <!-- <DefineEmits @parentClick="parentClick" /> -->
  <!-- <slots-attrs a="1" b="2" msg="hello" class="child" style="color:red">
      <template #header>
          <h1>我是父组件插槽!</h1>
      </template>
  </slots-attrs> -->
  <DefineExpose  ref="child" />

</template>

<style>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

在父组件中直接修改子组件的属性，子组件也会相应更新

8 vue-router4和vuex4

8-1 vue-router4安装和引入

安装vue-router4

```
npm i vue-router@next
```

vue-router是Vue.js官方的路由插件，路由插件是根据不同的url地址来显示不同的页面或内容的功能

它和vue.js是深度集成的，适合用于构建单页面应用。vue的单页面应用是基于路由和组件的，路由用于设定访问路径，并将路径和组件映射起来。传统的页面应用，是用一些超链接来实现页面切换和跳转的。在vue-router单页面应用中，则是路径之间的切换，也就是组件的切换。

1.编写路由配置文件

在src目录创建router目录,新建index.ts

createWebHistory路由模式路径不带#号(生产环境下不能直接访问项目，需要nginx转发)

createWebHashHistory路由模式路径带#号

```
import {createRouter,createWebHashHistory,RouteRecordRaw} from 'vue-router'

const routes:Array<RouteRecordRaw>  = [
    {
        path:'/',
        component: () => import('@/components/HelloWorld.vue')
    }
]

const router = createRouter({
    history: createWebHashHistory(),
    routes: routes
})

export default router
```

2.main.ts中引入

```
import { createApp } from 'vue'
import App from './App.vue'
import router from "./router";

createApp(App).use(router).mount('#app')
```

3.在app.vue入口文件使用router-view,使用router-view承载子路由的容器

```
<router-view />
```

8-2 vuex安装和引入

安装vuex4

```
npm install vuex@next
```

1.src目录下创建 store/index.ts

```
import { createStore } from 'vuex'

 interface State {
    count: number
  }
export const store = createStore<State>({
    state () {
      return {
        count: 0
      }
    },
    mutations: {
      increment (state) {
        state.count++
      }
    }
  })
```

2.在main.ts中注册

```
import { createApp } from 'vue'
import App from './App.vue'
import router from "./router";
import { store } from './store'

createApp(App).use(router).use(store).mount('#app')
```

3.在项目中使用vuex中的状态和函数,直接引入useStore即可使用state和mutation

```
import {computed} from 'vue'
import { useStore } from 'vuex'



const store = useStore()
let count = computed(() => {
  return store.state.count
})
const increment = () =>{
  store.commit('increment')
```

```
  console.log(count);


}
```

8-3 安装sass和element plus

安装element plus

```
npm install element-plus --save
```

安装下sass

```
npm i sass-loader sass -D
```

安装完成后,如何使用element plus提供的ui?

哪种引入方式值得推荐?

完整引入

```
// main.ts
import { createApp } from 'vue'
import ElementPlus from 'element-plus'
import 'element-plus/dist/index.css'
import App from './App.vue'

const app = createApp(App)

app.use(ElementPlus)
app.mount('#app')
```

如果您使用 Volar，请在 `tsconfig.json` 中通过 `compilerOptions.type` 指定全局组件类型。

```
// tsconfig.json
{
  "compilerOptions": {
    // ...
    "types": ["element-plus/global"]
  }
}
```

element plus推荐自动化的按需导入

按需导入

您需要使用额外的插件来导入要使用的组件。

#自动导入 推荐

First you need install `unplugin-vue-components` and `unplugin-auto-import`.

```
npm install -D unplugin-vue-components unplugin-auto-import
```

Then add the code below into your `Vite` or `Webpack` config file.

Vite

```
// vite.config.ts
import AutoImport from 'unplugin-auto-import/vite'
import Components from 'unplugin-vue-components/vite'
import { ElementPlusResolver } from 'unplugin-vue-components/resolvers'

export default {
  plugins: [
    // ...
    AutoImport({
      resolvers: [ElementPlusResolver()],
    }),
```

首先我们导入库包

```
npm install -D unplugin-vue-components unplugin-auto-import
```

然后在vite.config.ts中导入

```
// vite.config.ts
import AutoImport from 'unplugin-auto-import/vite'
import Components from 'unplugin-vue-components/vite'
import { ElementPlusResolver } from 'unplugin-vue-components/resolvers'

export default {
  plugins: [
    // ...
    AutoImport({
      resolvers: [ElementPlusResolver()],
    }),
    Components({
      resolvers: [ElementPlusResolver()],
    }),
  ],
}
```

手动导入

```
<template>
  <el-button>I am ElButton</el-button>
</template>
<script>
  import { ElButton } from 'element-plus'
  export default {
    components: { ElButton },
```

```
  }
</script>
// vite.config.ts
import ElementPlus from 'unplugin-element-plus/vite'

export default {
  plugins: [ElementPlus()],
}
```

9 Layout布局

9-1 页面布局

我们的布局内容主要由3部分组成,侧边栏 ,顶部,页面内容

```
<template>
  <div class="layout">
      <el-container >
        <el-aside class="layot-sider">
            Asider
        </el-aside>

        <el-container>
            <el-header>
                Header
            </el-header>

            <el-main>
                Main
            </el-main>
        </el-container>
    </el-container>
  </div>
</template>

<script  setup lang="ts">

</script>

<style lang="scss">
.layout {
    display: flex;
    height: 100vh;

    .el-header {
        background-color: #b3c0d1;
        color: var(--el-text-color-primary);
         margin: 0;
    }

    .el-aside {
        color: var(--el-text-color-primary);
         margin: 0;
    }
}
</style>
```

这个时候看到有外边距

在根目录index.html中去掉外边距

box-sizing: border-box就是将border和padding数值包含在width和height之内,这样的好处就是修改border和padding数值盒子的大小不变。”

```
<style>
  html,body {
    margin: 0;
    padding: 0;
    height: 100%;
    box-sizing:border-box
  }
</style>
```

9-2侧边栏

menuBar

```
<template>

        <el-menu
        active-text-color="#ffd04b"

        class="el-menu"
        default-active="2"
        text-color="#fff"

      >
        <el-sub-menu index="1">
          <template #title>
            <el-icon><location /></el-icon>
            <span>Navigator One</span>
          </template>
          <el-menu-item-group title="Group One">
            <el-menu-item index="1-1">item one</el-menu-item>
            <el-menu-item index="1-2">item one</el-menu-item>
          </el-menu-item-group>
          <el-menu-item-group title="Group Two">
            <el-menu-item index="1-3">item three</el-menu-item>
          </el-menu-item-group>
          <el-sub-menu index="1-4">
            <template #title>item four</template>
            <el-menu-item index="1-4-1">item one</el-menu-item>
          </el-sub-menu>
        </el-sub-menu>
        <el-menu-item index="2">
          <el-icon><icon-menu /></el-icon>
          <span>Navigator Two</span>
        </el-menu-item>
        <el-menu-item index="3" disabled>
          <el-icon><document /></el-icon>
          <span>Navigator Three</span>
        </el-menu-item>
        <el-menu-item index="4">
          <el-icon><setting /></el-icon>
          <span>Navigator Four</span>
        </el-menu-item>
      </el-menu>
</template>

<script setup lang="ts">

import {
  Location,
  Document,
  Menu as IconMenu,
  Setting,
} from '@element-plus/icons-vue'
```

```
</script>

<style lang="scss">
  .el-menu{
    background-color: $menuBg;
    border: none;
  }
</style>
```

logoBar

```
<template>
  <div class="logo">
    <img src="@/assets/logo.png" >
    <h2 v-show="!collapsed" class="title">Vue3-admin</h2>
  </div>
</template>

<script setup lang="ts">
  defineProps({
    collapsed:{
      type:Boolean
    }
  })
</script>

<style lang="scss" scoped>
.logo {
  display: flex;
  align-items: center;
  padding-left: 14px;
  height: 64px;
  line-height: 64px;
  overflow: hidden;
  white-space: nowrap;

  img {
    height: 32px;
    margin-right: 8px;
  }

  .title {
   font-size: 20px;
    color: white;
    margin-bottom: 0;
  }
}
</style>
```

9-3 vite中添加css全局变量

首先创建一个variables.scss

```
/* Variables */

// Base color
$blue:#324157;
$light-blue:#3A71A8;
$red:#C03639;
$pink: #E65D6E;
$green: #30B08F;
$tiffany: #4AB7BD;
$yellow:#FEC171;
$panGreen: #30B08F;

// Sidebar
$sideBarWidth:300px;
$subMenuBg:#1f2d3d;
$subMenuHover:#001528;
$subMenuActiveText:#f4f4f5;
$menuBg:rgb(50, 65, 87);;
$menuText:#162B64;
$menuActiveText:#435EBE; // Also see settings.sidebarTextTheme

// Login page
$lightGray: #eee;
$darkGray:#889aa4;
$loginBg: #2d3a4b;
$loginCursorColor: #fff;
```

这就是全局样式 变量,为了方便统一管理修改

我们需要引入在全局使用

所以需要在vite中配置

```
 css:{
    //css预处理
    preprocessorOptions:{
      scss:{
        //引入varibles.scss全局预定义变量
        additionalData:`@import "./src/styles/variables.scss";`,
      }
    }
  },
```

9-4控制侧边栏折叠

首先要有个折叠按钮来控制

引入**icon**

```
import {
  Expand,
  Fold
} from '@element-plus/icons-vue'
```

在layout的index.vue中的header添加icon,加is判断

```
  <el-header>
                    <el-row >
                            <!-- 侧边栏折叠按钮 -->
                            <el-icon style="font-size: 20px;" @click="() =>
(collapsed = !collapsed)">
                                <component :is="collapsed ? Expand : Fold" />
                            </el-icon>

                    </el-row>
                </el-header>
```

声明collapsed,并且对子组件传递状态

```
const collapsed = ref<boolean>(false)
```

父传子

```
const collapsed = ref<boolean>(false)
```

子组件接收collapsed,并且绑定到对应元素

```
            <el-aside :style="'width:' + autoWidth">
                <logo-bar :collapsed="collapsed" />
                <menu-bar :collapsed="collapsed" />
            </el-aside>
```

defineProps传值

```
  defineProps({
    collapsed:{
      type:Boolean
    }
  })
```

这个时候menu已经可以正常折叠,但是aside的宽带并没有跟随,

所以我们将aside的width绑定;

```
<el-aside :style="'width:' + autoWidth">
```

使用计算属性计算aside的宽width,并跟随调整

```
const autoWidth = computed(() => {

     if(collapsed.value) {
        return "64px"
    } else {
        return "200px"
    }
})
```

当然,以上只是pc端折叠,

如果在移动端因为aside占屏比太高,

一般选择完全隐藏

所以需要一个函数判断是否是移动端

然后再进行aside的宽带变化

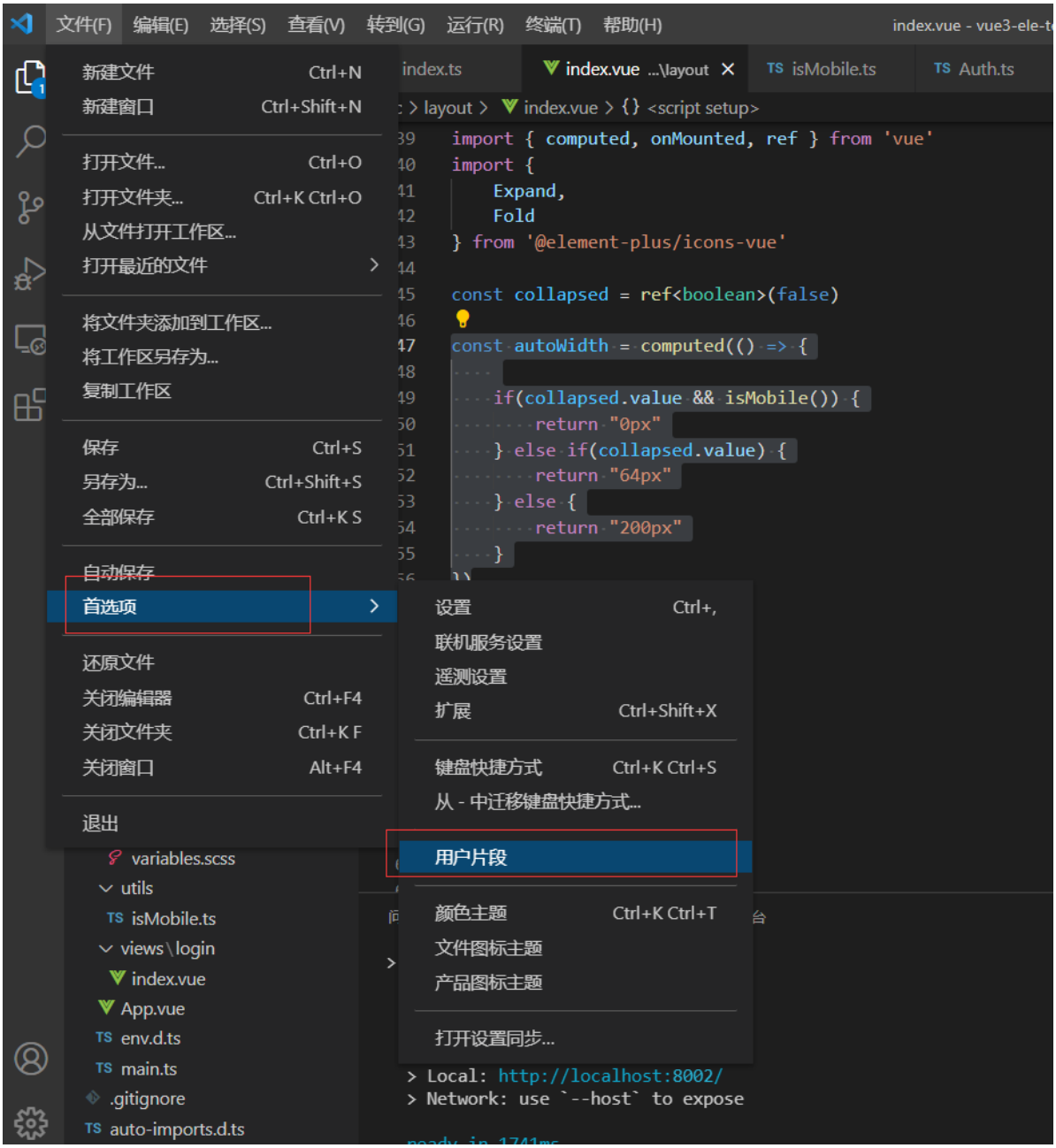
在src目录下创建一个utils的isMobile.ts来判断当前是否属于移动端

```
export const isMobile= ():boolean=>{
    let flag =
navigator.userAgent.match(/(phone|pad|pod|iPhone|iPod|ios|iPad|Android|Mobile|Bl
ackBerry|IEMobile|MQQBrowser|JUC|Fennec|wOSBrowser|BrowserNG|WebOS|Symbian|Windo
ws Phone)/i)
    if(flag === null) {
        return false
    } else {
        return true
    }

}
```

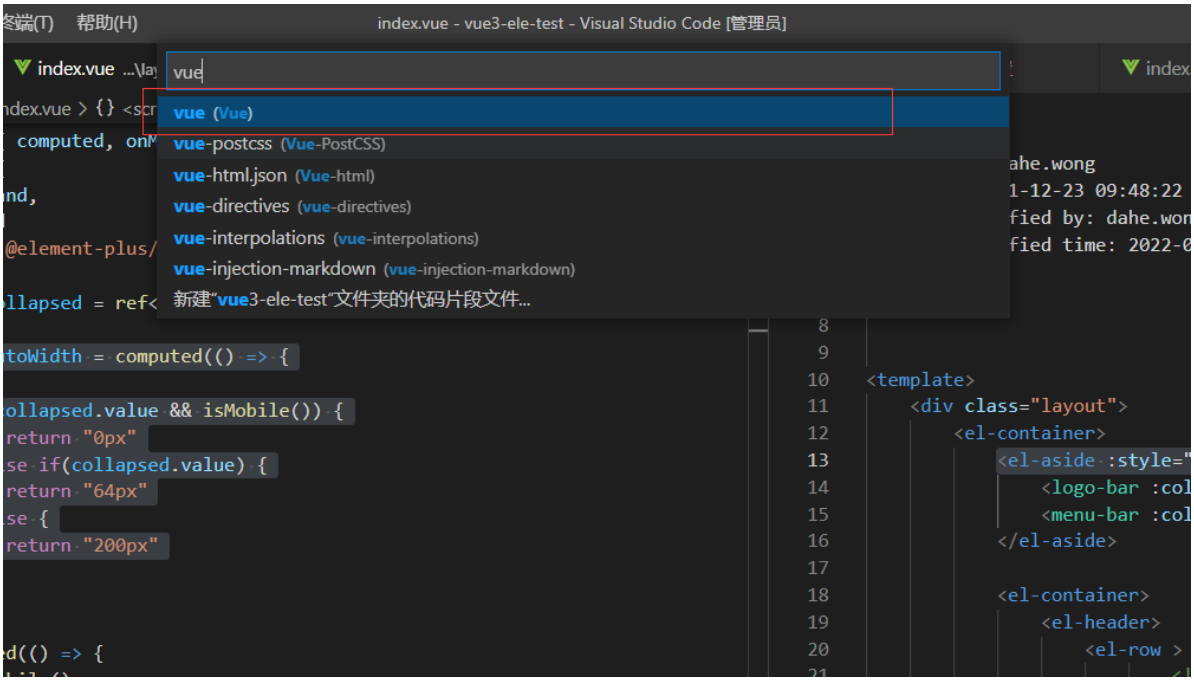
最后,重新计算autoWidth

```
const autoWidth = computed(() => {

    if(collapsed.value && isMobile()) {
        return "0px"
    } else if(collapsed.value) {
        return "64px"
    } else {
        return "200px"
    }
})
```

9-5 vscode 快捷键新建vue模板



首选项-用户片段



选择新建一个代码片段vue3

然后输入模板保存即可

```
    "Print to console": {
        "prefix": "vue3",
        "body": [
          "<!-- $0 -->",
          "<template>",
          "  <div></div>",
          "</template>",
          "",
          "<script setup lang='ts'>",

          "</script>",
          "<style lang='scss' scoped>",
          "</style>",
      ],
        "description": "Log output to console"
      }
```

9-6全局使用动态图标icon

1 安装下icon插件

```
npm install @element-plus/icons
```

2 在main.ts中配置

```
import * as ElIcons from '@element-plus/icons'

const app = createApp(App)
for (const name in ElIcons){
    app.component(name,(ElIcons as any)[name])
}
app.use(router).use(store).mount('#app')
```

3 页面中可以潇洒的使用了

9-7 menu菜单

菜单规划

接着生成对应的路由列表

```
const menus = reactive([
  {
        path: '/',
        redirect: '/index',
        name: 'Index',

        meta: {
         title: '首页',
              icon:'house',
        },
        children: [
          {
            path: 'index',
            name: 'Index',
            component:() => import('@/views/index/Index.vue'),
            meta: {
              title: '首页',
```

```
            icon:'house',

          },
        },
      ],
    },
    {
      path: '/user',
      redirect: '/user',
      name: 'User',

      meta: {
        title: '用户管理',
        affix: true,
        icon:'UserFilled',
      },
      children: [
        {
          path: 'manger',
          name: 'UserManger',
          component:() => import('@/views/user/User.vue'),
          meta: {
            title: '用户管理',
            icon:'UserFilled',

          },
        },
      ],
    },{
      path: '/storesLocation',
      redirect: '/storesLocation',
      name: 'storesLocation',

      meta: {
          title: '门店管理',
          icon:'LocationInformation',
      },
      children: [
        {
          path: 'storesLocation',
          name: 'storesLocation',
          component:() => import('@/views/storesLocation/StoresLocation.vue'),
          meta: {
              title: '门店管理',
              icon:'LocationInformation',

          },
        },
      ],
    },

    {
      path: '/order',
      name: 'Order',

      meta: {
        title: '订单管理',
        icon: 'Notebook',
```

```
        roles: ['admin', 'editor']
      },
      children: [
        {
          path: 'orderQuery',
          name: 'orderQuery',
          component:() => import('@/views/orders/OrderQuery.vue'),
          meta: {
            title: '订单查询',
            icon: 'Notification',
          },
        },
        {
          path: 'orderAction',
          name: 'orderAction',
          component:() => import('@/views/orders/OrderAction.vue'),
          meta: {
            title: '订单处理',
            icon: 'Money',
          },
        },
      ],
    },{
      path: '/good',
      name: 'good',

      meta: {
        title: '商品管理',
        icon: 'TakeawayBox',
      },
      children: [
        {
          path: 'category',
          name: 'category',
          component:() => import('@/views/goods/Goods.vue'),
          meta: {
            title: '商品种类',
            icon: 'ShoppingBag',
          },
        },
        {
          path: 'goodQuery',
          name: 'goodQuery',
          component:() => import('@/views/goods/Category.vue'),
          meta: {
            title: '商品查询',
            icon: 'SoldOut',
          },
        },
      ],
    },

    {
      path:'/system',
      name:'system',

      meta:{
        title:'system',
```

```
          icon:'Wallet',
          roles: ['admin','editor']
        },
        children:[
          {
            path:'account',
            name:'account',
            component:() => import('@/views/system/Account.vue'),
            meta:{
              title:'account',
              icon:'User',
              roles: ['editor']
            }

          },
          {
            path:'group',
            name:'group',
            component:() => import('@/views/system/Group.vue'),
            meta:{
              title:'group',
              icon:'Refrigerator',
              roles: ['admin']
            }
          }, {
            path:'task',
            name:'task',
            component:() => import('@/views/system/Task.vue'),
            meta:{
              title:'account',
              icon:'Clock',
              roles: ['editor']
            }

          },
          {
            path:'Setting',
            name:'Setting',
            component:() => import('@/views/system/Setting.vue'),
            meta:{
              title:'系统设置',
              icon:'Setting',
              roles: ['admin']
            }
          }
        ]

      }
])
```

从路由列表的信息遍历到menuLIst

```
<el-menu
    active-text-color="#409EFF"
    :collapse="collapsed"
    class="el-menu"
```

```
    text-color="#fff"
    v-for="menu in menus" :key="menu.path"
  >
    <el-sub-menu v-if="menu.children && menu.children.length >1 " >
      <template #title>
        <el-icon>
          <component :is="menu.meta?.icon"></component>
        </el-icon>
        <span>{{menu.meta?.title}}</span>
      </template>

    </el-sub-menu>
    <template v-else v-for="item in menu.children" :key="item.path" >
         <el-menu-item >
      <el-icon>
        <component :is="item.meta.icon"></component>
      </el-icon>
      <span>{{item.meta.title}}</span>
    </el-menu-item>
    </template>
  </el-menu>
```

这样一个简单的menu就制作成功,但是我们想下拉二级菜单的时候,就需要重新操作一次

因此我们将menu封装成子组件,并且在子组件中调用自己,这样就能递归

创建一个menu-item的子组件

```
    <template v-for="menu in menus" :key="menu.path" >
         <el-sub-menu v-if="menu.children && menu.children.length >1 " >
      <template #title>
        <el-icon>
          <component :is="menu.meta?.icon"></component>
        </el-icon>
        <span>{{menu.meta?.title}}</span>
      </template>

       <menu-item :menus="menu.children"></menu-item>

    </el-sub-menu>
    <template v-else  >
         <el-menu-item >
      <el-icon>
        <component :is="menu.meta?.icon"></component>
      </el-icon>
      <span>{{menu.meta?.title}}</span>
    </el-menu-item>
    </template>
    </template>
</template>
```

10 面包屑和选项卡

10-1 菜单切换页面

路由信息

```
{
        path:'/login',
        component: () => import('@/views/login/Login.vue')
    }, {
        path: '/',
        redirect: '/index',
        name: 'Index',
        component:Layout,
        meta: {
         title: '首页',
              icon:'house',
        },
        children: [
          {
            path: 'index',
            name: 'Index',
            component:() => import('@/views/index/Index.vue'),
            meta: {
              title: '首页',
              icon:'house',

            },
          },
        ],
      },
      {
        path: '/user',
        redirect: '/user/manger',
        name: 'User',
        component:Layout,
        meta: {
          title: '用户管理',
          affix: true,
          icon:'UserFilled',
        },
        children: [
          {
            path: 'manger',
            name: 'UserManger',
            component:() => import('@/views/user/User.vue'),
            meta: {
              title: '用户管理',
              icon:'UserFilled',

            },
          },
        ],
      },{
        path: '/stores',
        redirect: '/stores/Location',
        name: 'storesLocation',
        component:Layout,
```

```
      meta: {
          title: '门店管理',
          icon:'LocationInformation',
      },
      children: [
        {
          path: 'Location',
          name: 'storesLocation',
          component:() => import('@/views/storesLocation/StoresLocation.vue'),
          meta: {
              title: '门店管理',
              icon:'LocationInformation',

          },
        },
      ],
    },

    {
      path: '/order',
      name: 'Order',
      component:Layout,
      meta: {
        title: '订单管理',
        icon: 'Notebook',
        roles: ['admin', 'editor']
      },
      children: [
        {
          path: 'orderQuery',
          name: 'orderQuery',
          component:() => import('@/views/orders/OrderQuery.vue'),
          meta: {
            title: '订单查询',
            icon: 'Notification',
          },
        },
        {
          path: 'orderAction',
          name: 'orderAction',
          component:() => import('@/views/orders/OrderAction.vue'),
          meta: {
            title: '订单处理',
            icon: 'Money',
          },
        },
      ],
    },{
      path: '/good',
      name: 'good',
      component:Layout,
      meta: {
        title: '商品管理',
        icon: 'TakeawayBox',
      },
      children: [
        {
          path: 'category',
```

```
            name: 'category',
            component:() => import('@/views/goods/Goods.vue'),
            meta: {
              title: '商品种类',
              icon: 'ShoppingBag',
            },
          },
          {
            path: 'goodQuery',
            name: 'goodQuery',
            component:() => import('@/views/goods/Category.vue'),
            meta: {
              title: '商品查询',
              icon: 'SoldOut',
            },
          },
        ],
      },

      {
        path:'/system',
        name:'system',
        component:Layout,
        meta:{
          title:'system',
          icon:'Wallet',
          roles: ['admin','editor']
        },
        children:[
          {
            path:'account',
            name:'account',
            component:() => import('@/views/system/Account.vue'),
            meta:{
              title:'account',
              icon:'User',
              roles: ['editor']
            }

          },
          {
            path:'group',
            name:'group',
            component:() => import('@/views/system/Group.vue'),
            meta:{
              title:'group',
              icon:'Refrigerator',
              roles: ['admin']
            }
          }, {
            path:'task',
            name:'task',
            component:() => import('@/views/system/Task.vue'),
            meta:{
              title:'account',
              icon:'Clock',
              roles: ['editor']
            }
```

```
          },
          {
            path:'Setting',
            name:'Setting',
            component:() => import('@/views/system/Setting.vue'),
            meta:{
              title:'系统设置',
              icon:'Setting',
              roles: ['admin']
            }
          }
        ]

      }
```

创建appMain.vue

```
<template>



   <router-view />



</template>



<script setup lang='ts'>

</script>

<style lang='scss' scoped>

</style>
```

然后在layout的index.vue中引入

```
                <el-main>
                    <app-main></app-main>
                </el-main>
```

最后在menu-item加点击跳转和索引

```
 <el-menu-item @click="toPath(menu.name)" :index="menu.path">
```

这个时候页面就能正常切换了

10-2 面包屑

知识要点

el-breadcrumb面包屑

```
useRoute的应用

useRoute()的api->matched获取嵌套路由的路由信息

watche监控path

Ref配置声明数据的数据类型
```

引入面包屑模板

```
<template>
    <el-breadcrumb >
    <el-breadcrumb-item v-for="(item,index) in breadcrumb" :key="index" >
{{item.meta.title}}</el-breadcrumb-item>

  </el-breadcrumb>
</template>
```

面包屑监听路由path变化,

并且通过route.matched获取当前嵌套路由的路由信息

然后通过过滤得到面包屑展开对应数组

```
<script setup lang="ts">
import { Ref,ref,onMounted,watch } from 'vue'
import {RouteLocationMatched, useRoute} from 'vue-router'
const route = useRoute()

const breadcrumb:Ref<RouteLocationMatched[]> = ref([])
const getBreadCrumb = ()=> {
    let matched =  route.matched.filter((item)=> item.meta && item.meta.title &&
item.children.length !==1)


    const frist = matched[0]
    if(frist.path !== '/index') {
        matched = [{path:'/index', meta:{title:'首页'}} as any ].concat(matched)
    }
    breadcrumb.value = matched
}
// 初始化加载面包屑
onMounted(() => {
    getBreadCrumb()
})

// 监控路由变化,面包屑发生变化
watch(()=>route.path,() => {
    getBreadCrumb()
})
```

```
</script>
```

10-3tab选项卡添加

需求分析

```
添加tab, 并且需要和menu菜单联动
```

注意:

vue文件使用别名引入ts后缀文件,

会报错,找不到模块

需要在typescript.json中配置别名路径识别

```
"paths": {
      "@": ["./src"],
      "@/*": ["./src/*"] // 多加个这个
    }
```

1. 引入ele-tab组件

```
<template>
  <div style="margin-bottom: 20px">
    <el-button size="small" @click="addTab(editableTabsValue)">
      add tab
    </el-button>
  </div>
  <el-tabs
    v-model="editableTabsValue"
    type="card"
    closable
    @tab-remove="removeTab"
  >
    <el-tab-pane
      v-for="item in editableTabs"
      :key="item.name"
      :label="item.title"
      :name="item.name"
    >
      {{ item.content }}
    </el-tab-pane>
  </el-tabs>
</template>
<script lang="ts" setup>
import { ref } from 'vue'

let tabIndex = 2
const editableTabsValue = ref('2')
const editableTabs = ref([
  {
    title: 'Tab 1',
    name: '1',
    content: 'Tab 1 content',
```

```
  },
  {
    title: 'Tab 2',
    name: '2',
    content: 'Tab 2 content',
  },
])

const addTab = (targetName: string) => {
  const newTabName = `${++tabIndex}`
  editableTabs.value.push({
    title: 'New Tab',
    name: newTabName,
    content: 'New Tab content',
  })
  editableTabsValue.value = newTabName
}
const removeTab = (targetName: string) => {
  const tabs = editableTabs.value
  let activeName = editableTabsValue.value
  if (activeName === targetName) {
    tabs.forEach((tab, index) => {
      if (tab.name === targetName) {
        const nextTab = tabs[index + 1] || tabs[index - 1]
        if (nextTab) {
          activeName = nextTab.name
        }
      }
    })
  }

  editableTabsValue.value = activeName
  editableTabs.value = tabs.filter((tab) => tab.name !== targetName)
}
</script>
```

2.配置tab相关的数据为全局状态tabsList

首先创建tab的数据类型type,方便用时引入

```
export type Itab = {
    path: string
    title: string
  }
```

在store中的index.ts中,加入state\mutations\getters

```
import { createStore } from 'vuex'
import {useRoute} from 'vue-router'
import { tabsRootContextKey } from 'element-plus'
import {Itab} from './type'


 interface State {
```

```
    tabsList:Array<Itab>
  }

export const store = createStore<State>({
    state:{

      tabsList:[]
    },

    mutations: {

      addTabe(state:State,tab:Itab) {
        const isSome = state.tabsList.some((item) => item.path == tab.path)
        if(!isSome) {
          state.tabsList.push(tab)
        }
      }
    },

    getters:{
      getAddTab(state:State) {
          return state.tabsList
      }
    }


  })
```

2.创建tabBar子组件,并且在layout的index.vue中引入

使用watche监控route.path,tab进行改变和添加

```
<template>
  <el-tabs

    type="card"
    v-model="activeKey"

  >
    <el-tab-pane
      v-for="item in TabsList"
      :key="item.name"
      :label="item.title"
      :name="item.path"
      closable
    >

    </el-tab-pane>
  </el-tabs>
</template>
<script lang="ts" setup>
import { ref,computed,watch } from 'vue'
import { useStore } from 'vuex'
import {useRoute} from 'vue-router'
```

```
import {Itab} from '../../../store/type'

const store = useStore()
const route = useRoute()
const TabsList = computed(() => {
   return store.getters.getAddTab
})

const activeKey = ref('')


const addTab = ()=>{
  const {meta,path} = route
  const tab:Itab = {
    path:path,
    title:meta.title as string
  }

  store.commit('addTabe',tab)

}
watch(()=>route.path,(to)=>{
  activeKey.value = route.path
    addTab()
})


</script>
```

10-4 点击tab跳转和删除

点击切换页面

tab组件的点击触发事件\删除

事件名	说明	回调参数
tab-click	tab 被选中时触发	tab 被点击的标签
tab-remove	点击 tab 移除按钮后触发	name，被删除的标签的名字
tab-add	点击 tabs 的新增按钮后触发	—
edit	点击 tabs 的新增按钮或 tab 被关闭后触发	(targetName, action)

点击触发切换页面

```
// 点击切换tab
const clickHande = (event:any) =>{
    console.log(event.props);
    router.push({path:event.props.name})
}
```

删除tab

创建removeTab并且在store中创建closeCurrentTab函数

```
  closeCurrentTab(state: State, targetName:string) {
        // 关闭当前页
        const index = state.tabsList.findIndex((item) => item.path ==
targetName)
        state.tabsList.splice(index, 1)
      },
    },
```

removetab

```
const removeTab = (targetName: string) => {
  console.log(targetName)
  if (TabsList.value.length === 1) {
     return      alert('这已经是最后一页!')
  }



  // 如果删除的是当前页
  if (activeKey.value === targetName) {
    TabsList.value.forEach((tab: Itab, index: number) => {
      if (tab.path === targetName) {
        const nextTab = TabsList.value[index + 1] || TabsList.value[index - 1]
        if (nextTab) {
          activeKey.value = nextTab.path
        }
```

```
      }
    })
  }

  store.commit('closeCurrentTab', targetName)



}
```

10-5 tab刷新缓存数据

beforeunload在页面刷新和关闭时触发的事件

通过window.addEventListener进行监听,如果一旦关闭或者刷新,

就将数据缓存到本地

```
// 移除tab
const removeTab = (targetName:string)=>{
    if(tabsList.value.length === 1) {
       return alert('这已经是最后一页')
    }

    // 如果删除的是当前页
    if(activeKey.value === targetName) {
      tabsList.value.forEach((tab:Itab,index:number)=>{
          if(tab.path === targetName) {
            const nextTab = tabsList.value[index + 1] || tabsList.value[index -
1]
            if(nextTab) {
              activeKey.value = nextTab.path
            }
          }
      })
    }

    store.commit('closeCurrentTab',targetName)

}
```

然后在onMounted添加刷新事件

```
onMounted(() => {

  // 初始化页面生成tab
  addTab()
  // 刷新保存数据
  refresh()
})
```

10-6 tab右键菜单关闭列表

如何在el-tabs 上绑定右键事件?

可以使用 @contextmenu.prevent.native="openContextMenu($event)"

```
<template>
  <el-tabs
    type="card"
    v-model="activeKey"
    @tab-click="clickHande"
    @tab-remove="removeTab"
    @contextmenu.prevent.native="openContextMenu($event)"
  >
    <el-tab-pane
      v-for="item in TabsList"
      :key="item.name"
      :label="item.title"
      :name="item.path"
      closable
    ></el-tab-pane>
  </el-tabs>
  <ul v-show="contextMenuVisible" :style="{ left: left + 'px', top: top + 'px' }"
class="contextmenu">
    <li @click="closeAllTabs">关闭所有</li>
    <li @click="closeOtherTabs('left')">关闭左边</li>
    <li @click="closeOtherTabs('right')">关闭右边</li>
    <li @click="closeOtherTabs('other')">关闭其他</li>
  </ul>
</template>
```

怎么获取当前右键点击时的tab

通过查看 e.srcElement.id 发现id的值含有tab的id 值， 因为在el-tab-pane 绑定的name值 :name="item.id" 就是tab的id值，只要获取当前右键点击时的tab的id值就知道当前的tab

```
// 右键点击状态
const contextMenuVisible: Ref<boolean> = ref(false)
const left: Ref<string> = ref('')
const top: Ref<string> = ref('')

// 右键点击事件
const openContextMenu = (e: any) => {
  console.log(e.srcElement.id)
  if (e.srcElement.id) {
    let currentContextTabId = e.srcElement.id.split("-")[1];
    contextMenuVisible.value = true;
    store.commit("saveCurContextTabId", currentContextTabId)
    left.value = e.clientX;
    top.value = e.clientY + 10

  }
}
```

style样式

```
.contextmenu {
  width: 100px;
  margin: 0;
  border: 1px solid #ccc;
  background: #fff;
  z-index: 3000;
  position: absolute;
  list-style-type: none;
  padding: 5px 0;
  border-radius: 4px;
  font-size: 14px;
  color: #333;
  box-shadow: 2px 2px 3px 0 rgba(0, 0, 0, 0.2);
  li {
    margin: 0;
    padding: 7px 16px;
    &:hover {
      background: #f2f2f2;
      cursor: pointer;
    }
  }
}
```

10-7 右键批量关闭tab

关闭事件

```
// 关闭所有tab

const closeAllTabs = () => {

 store.commit('closeAllTabs')

 contextMenuVisible.value = false

 router.push("/index")

}

// 关闭其它tab

const closeOtherTabs = (par: string) => {

 store.commit("closeOtherTabs", par);

 contextMenuVisible.value = false;

}
```

关闭右键菜单，有时候打开右键菜单没有进行其它操作，右键菜单一直显示

```
// 监控右键点开的List,避免右键菜单一直显示
watch(()=>contextMenuVisible.value,()=>{
      if (contextMenuVisible.value) {
        window.addEventListener("click", ()=>contextMenuVisible.value = false);
      } else {
        window.removeEventListener("click", ()=>contextMenuVisible.value =
false);
      }
})
```

最后在vuex添加函数

```
   saveCurContextTabId(state: State,curContextTabId) {
          state.curContextTabId = curContextTabId
      },
      //  关闭所有tab
      closeAllTabs(state: State) {
        state.tabsList = []
        sessionStorage.removeItem('TABS_ROUTES')
      },
      // 关闭其它
      closeOtherTabs(state: State,par:string) {
           if(par == 'other') {
            state.tabsList = state.tabsList.filter((item) => item.path ==
state.curContextTabId)
           } else if(par == 'left') {
```

```
            const index = state.tabsList.findIndex((item) => item.path ==
state.curContextTabId)
            state.tabsList.splice(0, index)
           } else if(par == 'right') {
            const index = state.tabsList.findIndex((item) => item.path ==
state.curContextTabId)
            state.tabsList.splice(index + 1)
           }
      }
```

11 用户登录

11-1登录form组件

第一步:路由当中注册login页面

```
    {
        path:'/',
        component: () => import('@/views/login/index.vue')
    }
```

第二步:构建页面,并且引入element plus form组件

```
<script setup lang="ts">
import { User, Lock } from '@element-plus/icons-vue'
import { ref, reactive } from 'vue'

const loginFormRef = ref(null)
const loginRules = ref(null)
const loginForm = reactive({
  username: '',
  password: ''
})




</script>

<template>
  <div class="login-container">
    <!-- 背景vedio -->
    <video poster="@/assets/login/1.jpg" loop autoplay muted>
      <source src="@/assets/login/Particles.mp4" />
    </video>

    <div class="login-form">
      <!-- 标题 -->
      <header>
        <img src="@/assets/logo.png" />
        <h3>vue3-admin</h3>
      </header>
      <!-- form组件 -->
      <el-form ref="loginFormRef" :model="loginForm" :rules="loginRules">
        <el-form-item>
          <el-icon>
            <user />
          </el-icon>
          <el-input label="username" placeholder="username" v-
model="loginForm.username" type="text" />
        </el-form-item>
        <el-form-item>
          <el-icon>
            <lock />
          </el-icon>
```

```
          <el-input label="password" placeholder="password" v-
model="loginForm.password" type="password" />
        </el-form-item>

        <el-form-item style="border: none; background:none">
          <el-button type="primary" style="width:100%;  margin-bottom:30px;">登录
</el-button>
        </el-form-item>
      </el-form>
    </div>
  </div>
</template>

<style lang="scss">
// 隐藏滚动条
::-webkit-scrollbar {
  width: 0 !important;
}
::-webkit-scrollbar {
  width: 0 !important;
  height: 0;
}
.login-container {
  height: 100%;
  width: 100%;
  overflow: hidden;
  display: flex;
  justify-content: center;
  align-items: center;
  video {
    position: absolute;
    /* Vertical and Horizontal center*/
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    width: 100%;
    height: 100%;
    object-fit: fill;
    z-index: -1;
  }
  .login-form {
    width: 400px;
    height: 380px;
    padding: 4vh;
    margin: 20px;
    background: url("@/assets/login/login_form.png");
    background-size: 100% 100%;
    border-radius: 10px;
    box-shadow: 0 2px 8px 0 rgba(7, 17, 27, 0.06);
    opacity: "0.2";

    header {
      display: flex;
      justify-content: center;
      align-items: center;
      margin-bottom: 20px;
      img {
```

```
        display: inline-block;
        width: 40px;
      }

      h1 {
        margin-bottom: 0;
        font-size: 24px;
        color: #fff;
        text-align: center;
      }
    }
    .el-input {
      display: inline-block;
      height: 47px;
      width: 85%;

      input {
        height: 47px;
        background: transparent;
        border: 0px;
        border-radius: 0px;
        padding: 12px 5px 12px 15px;
        color: $lightGray;
        caret-color: $loginCursorColor;
        -webkit-appearance: none;

        &:-webkit-autofill {
          box-shadow: 0 0 0px 1000px $loginBg inset !important;
          -webkit-text-fill-color: #fff !important;
        }
      }
    }

    .el-form-item {
      border: 1px solid rgba(255, 255, 255, 0.1);
      background: rgba(0, 0, 0, 0.1);
      border-radius: 5px;
      color: #454545;
    }
  }

  .tips {
    font-size: 14px;
    color: #fff;
    margin-bottom: 10px;

    span {
      &:first-of-type {
        margin-right: 16px;
      }
    }
  }

  .svg-container {
    padding: 6px 5px 6px 15px;
    color: $darkGray;
    vertical-align: middle;
    width: 30px;
```

```
    display: inline-block;
  }

  .title-container {
    position: relative;

    .title {
      font-size: 26px;
      color: $lightGray;
      margin: 0px auto 40px auto;
      text-align: center;
      font-weight: bold;
    }

    .set-language {
      color: #fff;
      position: absolute;
      top: 3px;
      font-size: 18px;
      right: 0px;
      cursor: pointer;
    }
  }

  .show-pwd {
    position: absolute;
    right: 10px;
    top: 7px;
    font-size: 16px;
    color: $darkGray;
    cursor: pointer;
    user-select: none;
  }

  .thirdparty-button {
    position: absolute;
    right: 0;
    bottom: 6px;
  }

  @media only screen and (max-width: 470px) {
    .thirdparty-button {
      display: none;
    }
  }
}
</style>
```

form表单校验

element plus提供的校验

rules普通校验和自定义校验,两种方式!

首先都需要先绑定 :rules,并且在ele-item中配置prop

```
    <!-- form组件 -->
    <el-form ref="loginFormRef" :model="loginForm" :rules="loginRules">
      <el-form-item prop="username" >
        <el-icon>
          <user />
        </el-icon>
        <el-input  placeholder="username" v-model="loginForm.username" type="text" />
      </el-form-item>
      <el-form-item prop="password" >
        <el-icon>
          <lock />
        </el-icon>
        <el-input  placeholder="password" v-model="loginForm.password" type="password" />
      </el-form-item>

      <el-form-item style="border: none; background:none">
        <el-button type="primary" style="width:100%;  margin-bottom:30px;">登录</el-button>
      </el-form-item>
    </el-form>
  </div>
</div>
```

普通校验

type
很少用到，input框正常情况的值都是'string',radio, checkbox也用不到这个
required
必填

```
{ required: true, message: '请输入活动名称', trigger: 'blur' }
```

pattern
正则校验,可以写自己需要的各种正则表达式

```
{pattern: /^[a-zA_Z0-9]{2,10}$/, message:'请输入2到10位数字或字母', trigger:
'blur'},
```

max, min, len

```
{max:2,min:10, message:'请输入2到10位数字或字母', trigger: 'blur'},
{ len: 10, message: '请输入10个字符', trigger: 'blur' }
```

enum

```
只能输入在备选数组中的值,一般这种就用select或者radio了，很少用到
{ type: 'enum', enum: ['admin', 'user', 'guest'] },
```

Whitespace
不能全部都是空格

```
{whitespace: true, message: '不能为全空格', trigger: 'blur' }
```

rulesForm这样配置:

```
const loginRules = reactive({
    username: [
          {
```

```
            required: true,
            message: 'Please input Activity name',
            trigger: 'blur',
          },
          {pattern: /^[a-zA_Z0-9]{2,10}$/, message:'请输入2到10位数字或字母',
trigger: 'blur'},
          {
            min: 3,
            max: 15,
            message: 'Length should be 3 to 15',
            trigger: 'blur',
          },
        ],

        password: [
          {
            required: true,
            message: 'Please input Activity password',
            trigger: 'blur',
          },
          {whitespace: true, message: '不能为全空格', trigger: 'blur' },
          {
            min: 3,
            max: 10,
            message: 'Length should be 3 to 10',
            trigger: 'blur',
          },
        ],
})
```

自定义校验:

同步校验

```
const userNameRef = (rule: object, value: string, callback: Function)=>{


}

const passwordRef = (rule: object, value: string, callback: Function)=>{
   if (value === '') {
        callback(new Error('Please input the password'))
      }

}

const loginRules = reactive({
         username: [{ validator: userNameRef, trigger: 'blur' }],
        password: [{ validator: passwordRef, trigger: 'blur' }]
})
```

一般我们使用自定义校验,会配合异步aysnc await 向后端发起请求进行验证

```
const userNameRef = async (rule: object, value: string, callback: Function)=>{
     try {
        await request(value)
        callback()
```

```
      } catch {
        callback(new Error('该用户名已被使用'))
      }

}

const passwordRef = (rule: object, value: string, callback: Function)=>{
   if (value === '') {
        callback(new Error('Please input the password'))
      }

}
```

11-2 配置代理跨域和axios访问

同源

协议

端口

域名

在vite.config.ts中配置

```
 server:{

    port:8002,
    proxy:{
      '/api': {
        target: 'http://106.52.235.252:8101/',
        changeOrigin: true,//允许跨域
        rewrite: (path) => path.replace(/^\/api/, '') //将作为识别用途的/api字符串从
请求路径中去掉
      },
    }
  },
```

我们需要引入axios作为http访问后端接口的工具

```
npm install axios
```

创建/api/axios.ts,

并且使用拦截器

```
import axios from 'axios'

// 创建axios实例
const axiosInstance = axios.create({
    baseURL: '/api',
    // timeout: 1200000 // 请求超时时间，毫秒（默认2分钟
})



// response 拦截器
axiosInstance.interceptors.response.use(response => {
console.log("[请求api] [" + response.config.method + "] ", response.config.url, "
[接口返回数据]", response.data)
if (response.status < 200 || response.status >= 400) {
    alert("未知异常");
    return Promise.reject();
}

const res = response.data
if (res.code === 200)
    return res;
```

```
else if (res.code === 10001) {
    alert('未登录或登录已经过期，请重新登录')
} else if (res.code === 10002) {
    alert('权限不足 错误码[' + res.code + "]");
} else
alert(res.message + ' [错误码' + res.code + ']');
return Promise.reject();
}, () => {
    alert("服务器网络异常")
return Promise.reject();
})

export default axiosInstance
```

在api目录创建Auth.ts,用于放登录事务

```
import axiosInstance from "./axios";

// 获取验证码

export function getCode() {
    return axiosInstance({
      url: '/auth/code',
      method: 'get'
    })
  }


  // 账号密码登录
  export function login(requestUser:object) {
    return axiosInstance({
      url: '/auth/login',
      method: 'post',
      data: requestUser
    })
  }

  // 通过token直接登录
  export function loginByToken(token:string) {
    return axiosInstance({
      url: '/auth/loginByToken?token=' + token,
      method: 'post'
    })
  }
```

11-3 login路由跳转

登录业务

1向后端提交用户登录信息loginform

2返回数据code为200就算成功

在Auth.ts中创建login

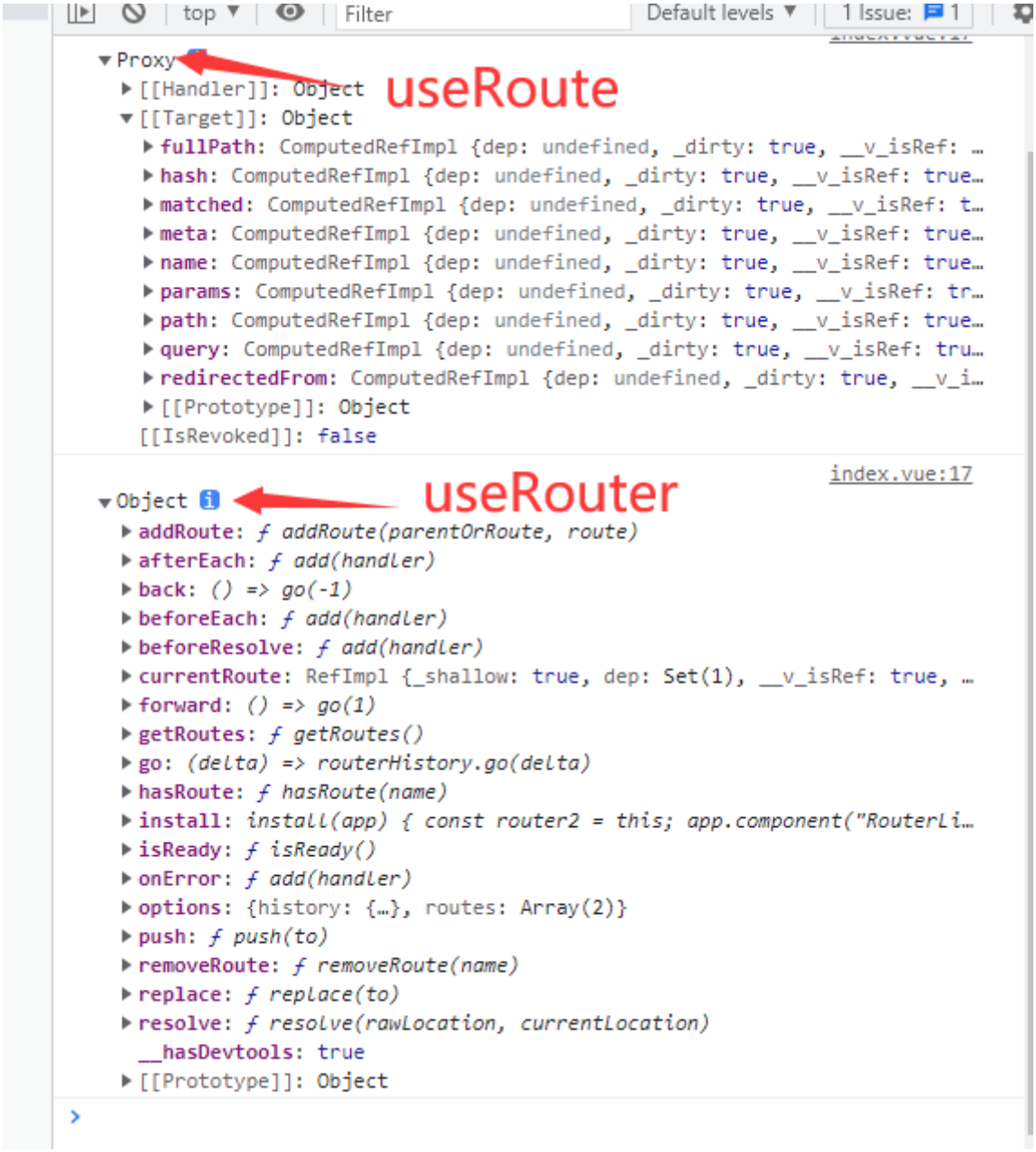
```
// 账号登录
export function login(requestUser:object) {
    return axiosInstance({
        url:'/auth/login',
        method:'post',
        data:requestUser
    })
}
```

在login中调用,通过router.push跳转页面

```
login(loginForm).then(result=>{
    if(result.data.token) {
        router.push({path:'/index'})

    }

  })
```

引入useRoute和useRouter,并且在setup中打印

```
import { useRoute, useRouter } from "vue-router";

console.log(useRoute());
console.log(useRouter());
```



可以看出useRoute是表示当前激活的路由的状态信息，包含了当前 URL 解析得到的信息，还有 URL 匹配到的 route路由记录

而useRouter是全局的路由实例

跳转路由push　go　replace之前的区别

1、router.go(n)
这个方法的参数是一个整数，意思是在 history 记录中向前或者后退多少步，类似 window.history.go(n)

2、router.push(location)
想要导航到不同的 URL，则使用 router.push 方法。这个方法会向 history 栈添加一个新的记录，所以，当用户点击浏览器后退按钮时，则回到之前的 URL。

3、router.replace(location)
跟 router.push 很像，唯一的不同就是，它不会向 history 添加新记录，而是跟它的方法名一样 —— 替换掉当前的 history 记录。

路由守卫

路由守卫分为前置路由守卫和后置路由守卫

11-4vuex实现模块化以及ts支持

当使用组合式 API 编写 Vue 组件时，您可能希望 `useStore` 返回类型化的 store。为了 `useStore` 能正确返回类型化的 store，必须执行以下步骤：

1. 定义类型化的 `InjectionKey`。
2. 将 store 安装到 Vue 应用时提供类型化的 `InjectionKey` 。
3. 将类型化的 `InjectionKey` 传给 `useStore` 方法。

让我们逐步解决这个问题。首先，使用 Vue 的 `InjectionKey` 接口和自己的 store 类型定义来定义 key：

> 本质上，`Vuex` 将`store` 安装到 `Vue` 应用中使用了 `Vue` 的 `Provide/Inject` 特性，这就是 `injection key` 是很重要的因素的原因。

这里官网有提供

https://next.vuex.vuejs.org/zh/guide/typescript-support.html#vue-%E7%BB%84%E4%BB%B6%E4%B8%AD-store-%E5%B1%9E%E6%80%A7%E7%9A%84%E7%B1%BB%E5%9E%8B%E5%A3%B0%E6%98%8E

在store文件夹中放入modules文件夹和index.ts入口文件，modules用于放置单独的模块

index.ts

```
import { InjectionKey } from "@vue/runtime-core"
import { createStore, Store, useStore as baseUseStore } from "vuex"
import { tabStore,TabState } from './modules/tab'


 export interface RootState {

  TabStore:TabState
  }

export const key: InjectionKey<Store<RootState>> = Symbol()//Symbol()函数会返回symbol类型的值

export const store: Store<RootState> = createStore({
  modules: { tabStore },




  })

//通过引入自定义的组合式函数，不用提供 injection key 和类型声明就可以直接得到类型化的 store

  export function useStore() {
    return baseUseStore(key)
  }
```

示例模块**tabStore**

在模块内使用的时候，需要将RootState接口引入

```
import { Module } from "vuex"
import {Itab} from '../type'
import { RootState } from "../index"

export interface TabState {

    tabsList:Array<Itab>
    curContextTabId:string
  }

  export const tabStore: Module<TabState, RootState> = {
    namespaced: true,
    state: (): TabState => ({
        tabsList:[],
        curContextTabId:''
    }),
    mutations: {
        // 添加tab
        addTabe(state:TabState,tab:Itab) {
          const isSome = state.tabsList.some((item) => item.path == tab.path)
          if(!isSome) {
            state.tabsList.push(tab)
          }
        },
        // 删除当前tab
        closeCurrentTab(state: TabState, targetName:string) {
          // 关闭当前页
          const index = state.tabsList.findIndex((item) => item.path ==
targetName)
          state.tabsList.splice(index, 1)
        },
        saveCurContextTabId(state: TabState,curContextTabId) {
            state.curContextTabId = curContextTabId
        },
        //  关闭所有tab
        closeAllTabs(state: TabState) {
          state.tabsList = []
          sessionStorage.removeItem('TABS_ROUTES')
        },
        // 关闭其它
        closeOtherTabs(state: TabState,par:string) {
             if(par == 'other') {
              state.tabsList = state.tabsList.filter((item) => item.path ==
state.curContextTabId)
             } else if(par == 'left') {
              const index = state.tabsList.findIndex((item) => item.path ==
state.curContextTabId)
              state.tabsList.splice(0, index)
             } else if(par == 'right') {
              const index = state.tabsList.findIndex((item) => item.path ==
state.curContextTabId)
```

```
            state.tabsList.splice(index + 1)
           }
      }
    },

    getters:{
      getAddTab(state:TabState) {
          return state.tabsList
      }
    }
  }
```

全局文件引入,将上述 injection key 传入 `useStore` 方法可以获取类型化的 store。

```
import { createApp } from 'vue'
import App from './App.vue'
import router from "./router";
import { store, key } from "./store"
import './styles/index.scss'
import * as ElIcons from '@element-plus/icons'


const app = createApp(App)
for (const name in ElIcons){
    app.component(name,(ElIcons as any)[name])
}
app.use(router).use(store,key).mount('#app')
```

在vuex+ typescript下使用方法方法

```
对模块中的各个属性方法的访问：

//state (当然原来也是这么访问的)
store.state.tabStore.tabsList

//getters
store.getters['tabStore/getTabList']

//mutations
store.commit('tabStore/increment')

//actions
store.dispatch('tabStore/incrementAction')
```

11-5 token登录和路由守卫

总所周知，每次你登录就是跳转到登录界面然后输入用户名密码什么的，登录。

那么如何记住登录呢？难不成我们的用户每次要完成登录后才能进行的操作的时候都需要重新登录一次吗？

将信息存储在本地storage中

当你将用户信息存好之后，以后每次只需要来取一次这个值，如果值存在，那么用户就是登录了，不存在则跳转到登录页面即可

后端token登录接口加入auth 的api

```
// 通过token直接登录
 export function loginByToken(token:string) {
   return axiosInstance({
     url: '/auth/loginByToken?token=' + token,
     method: 'post'
   })
 }
```

store创建auth.ts模块

```
import { Module } from "vuex"
import { RootState } from "../index"
import {login,loginByToken} from '@/api/Auth'


export interface AuthState {

    token: string
    userInfo: object
    roles: string[]

  }

  export const authStore:Module<AuthState, RootState> = {
    namespaced: true,
    state:():AuthState=>({
        token:'',
        userInfo:{},
        roles:[]
    }),

    mutations:{
            addToken(state:AuthState,token:string) {
                state.token = token
            }
    },

    getters:{
        getToken(state:AuthState) {
            return state.token
        }
    },
```

```
    actions:{
        login({commit, state, dispatch}, requestUser){
            login(requestUser).then(result=>{
                state.userInfo = result.data
                // token保存到vuex和localStorage
                commit('addToken',result.data.token)
                localStorage.setItem('token', result.data.token);


            })

        },

        loginByToken({commit, state, dispatch}, token) {
            commit('addToken',token)
            loginByToken(token).then(result=>{
                state.userInfo = result.data;
                localStorage.setItem('token', result.data.token);

            }).catch(()=>{
                localStorage.removeItem("token")
            })
        }
    }

  }npm i nprogress -S
```

store的index.ts添加模块

```
import { InjectionKey } from "@vue/runtime-core"
import { createStore, Store, useStore as baseUseStore } from "vuex"
import { authStore,AuthState } from "./modules/auth"
import { tabStore, TabState } from './modules/tab'


export interface RootState {

  TabStore: TabState,
  authStore:AuthState
}

export const key: InjectionKey<Store<RootState>> = Symbol()

export const store: Store<RootState> = createStore({
  modules: {
    tabStore,
    authStore
  },




})


export function useStore() {
```

```
  return baseUseStore(key)
}
```

在login界面更改登录函数

```
// 登录事件
const handleLogin = () => {


  login(loginForm).then(result=>{
    (loginFormRef.value as any).validate(async(valid: boolean)=>{
        if(valid) {
          loading.value = true
           await store.dispatch('authStore/login',loginForm)
            // 跳转路由

           setTimeout(() => {
              loading.value = false;
              router.push({ path: '/index'})
            }, 1000)

        } else {
          return false
        }
    })

  })
}
```

token登录测试

```
// 初始化
onMounted(() => {
  getValidCode()
  handleToken()
})
// token登录
const handleToken = ()=>{
 const token =  localStorage.getItem('token');
 if(token!= null) {
   store.dispatch('auth/loginByToken',token)

 }
}
```

登录算是完成了，

后端也返回成功code

我们既要验证用户是否登录，又要验证用户登录是否真实，还要保证用户信息安全，所以我们需要有一个东西，它就像一把钥匙一样，锁在我们服务器上的，用户登录了，就把钥匙给他。

11-6 路由守卫配合token登录

首先,我们要理解什么是路由守卫,

路由守卫主要有前置和后置

全局前置守卫beforeEach就是在进入一个路由之前会调用
全局后置钩子afterEach就是在进入一个路由之后会调用

当一个导航触发时，全局前置守卫按照创建顺序调用。守卫是异步解析执行，此时导航在所有守卫 resolve 完之前一直处于 等待中(没有执行next就一直等待)。

每个守卫方法接收三个参数：

to: Route： 即将要进入的目标 路由对象
from: Route： 当前导航正要离开的路由
next: Function： 一定要调用该方法来 resolve 这个钩子。执行效果依赖 next 方法的调用参数。
next(): 进行管道中的下一个钩子。如果全部钩子执行完了，则导航的状态就是 confirmed (确认的)。
next(false)： 中断当前的导航。如果浏览器的 URL 改变了 (可能是用户手动或者浏览器后退按钮)，那么 URL 地址会重置到 from 路由对应的地址。
next('/') 或者 next({ path: '/' })：跳转到一个不同的地址。当前的导航被中断，然后进行一个新的导航。
你可以向 next 传递任意位置对象，且允许设置诸如 replace: true、name: 'home' 之类的选项以及任何用在 router-link 的 to prop 或 router.push 中的选项。
首先login和loginByToken返回数据成功后

将跳转路由auth.ts中添加路由跳转

```
 import router from "@/router";

 if(result.data.status) {
                    router.push({path:'/index'})
                }
```

利用前置路由守卫，配合token一键登录

```
import { store } from '@/store'
import {loginByToken } from '@/api/Auth'

router.beforeEach((to,from,next)=>{
  const token = localStorage.getItem('token')
    if(!store.state.authStore.token && !token ) {
      if(to.path.startsWith("/login"))
       next()
       else {
         console.log("还没有登录")
         next('/login')
       }
    } else if(!store.state.authStore.token && token){
      loginByToken(token).then(res=>{
          if(res.data.status) {

            next()
          }
      })
    }
    else {
```

```
      next()
    }
})
```

11-7用户登出

首先,给用户登出的途径

这里选择header右上角安排头像点击下拉出现退出

首先在store/type中创建type类型

```
export type UserType ={
    avatar:string,
    username:string,
    roleName:string,
    status:number
}
```

首先store的auth.ts,需要对userInfo数据类型进行约束

```
import {UserType} from '../type'

export interface AuthState {
    token: string,
    userInfo: UserType,
    roles: string[]
}


state: (): AuthState => ({
        token: '',
        userInfo:  {
            avatar:'',
            username:'',
            roleName:'',
            status:1
        },
        roles: []
    }),
```

创建UserBar.vue

```
<!--  -->
<template>
    <div class="userInfo">
        <el-dropdown :show-timeout="10" :hide-timeout="10" placement="right"
trigger="click">
            <!--用户信息弹窗开关 下拉菜单-->
            <div>
                <img
                    v-if="userInfo.avatar"
                    style="width: 35px; height: 35px; border-radius: 50%"
                    :src="userInfo.avatar"
                    alt="img"
                />

            </div>
            <template #dropdown>
```

```
                <el-dropdown-menu>

                    <el-dropdown-item @click="handleLogout">退出登录</el-dropdown-
item>
                </el-dropdown-menu>
            </template>
        </el-dropdown>
        <el-row>
            <el-col>
                <span
            style="color: #212121; margin-left: 10px;font-size: 15px"
        >{{ userInfo.username ? userInfo.username : '' }} </span>
            </el-col>
            <el-col>
                <span style="color: gray; margin-left: 10px;font-size: 8px">
            [{{ userInfo.roleName }}]
        </span>
            </el-col>
        </el-row>


    </div>
</template>

<script setup lang='ts'>
import { computed } from 'vue'
import { useStore } from '@/store'
import { useRouter } from 'vue-router';

const store = useStore()
const router = useRouter()

const userInfo = computed(() => {
    return store.state.authStore.userInfo
})

const handleLogout = ():void =>{
    localStorage.removeItem('token')
    location.reload()
}

</script>
<style lang='scss' scoped>
.userInfo {
    display: flex;
    flex-direction: row;
    align-items: center;
    justify-content: center;
}
</style>
```

然后header父组件中,通过el-row和el-col布局

```
  <el-row :gutter="24" style="display: flex; align-items: center">
      <el-col :span="20" :offset="0" >
        <bread-crumb />
      </el-col>
      <el-col :span="4" >
        <user-bar />
      </el-col>


    </el-row>
```

layout的index也加下水平居中布局

```
    <el-header>
                    <el-row :gutter="24" style="width: 100%; display: flex;
align-items: center;">
                            <!-- 侧边栏折叠按钮 -->
                            <el-col :span="1">
                                <el-icon style="font-size: 26px; margin-right:
15px;" @click="() => (collapsed = !collapsed)">
                                <component :is="collapsed ? Expand : Fold" />
                            </el-icon>
                            </el-col>

                            <!-- header组件 -->
                            <el-col :span="23">
                                <header-bar class="row-bg"/>
                            </el-col>

                    </el-row>
                </el-header>
```

其次,目标清空token,跳转登录页

因为我在之前路由守卫中有设置 to.path(login),防止死循环,

所以清空token加上reload刷新就会跳回登录页

```
  localStorage.removeItem('token')
     location.reload()
```

这时候没有数据!

因为在路由守卫中并没有添加数据

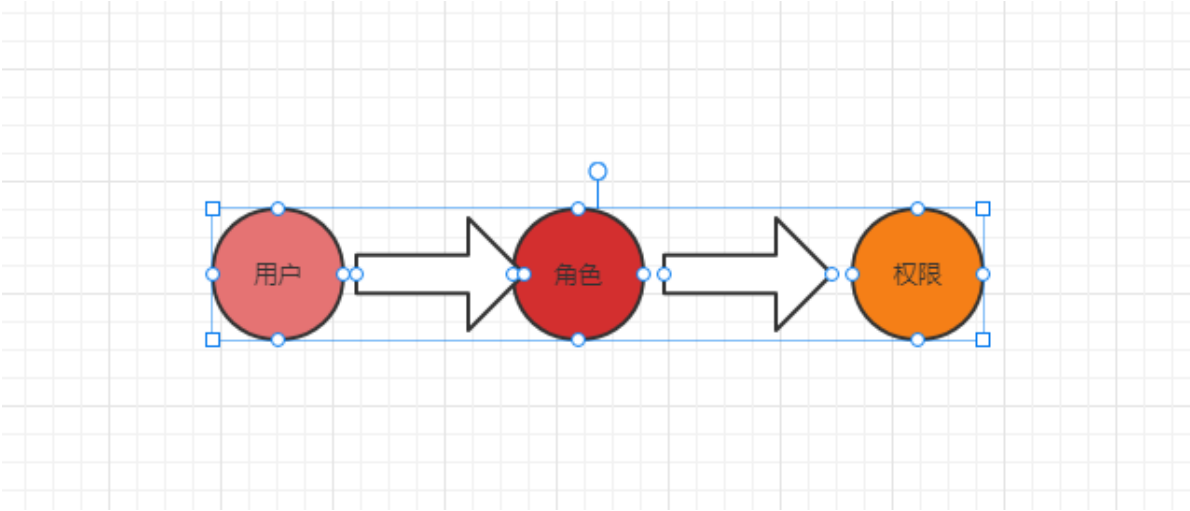
```
  loginByToken(token).then(res=>{
           if(res.data.status) {
             store.commit('authStore/addUserInfo',res.data)
             next()
           }
       })
```

auth.ts的mutations中

```
addUserInfo(state:AuthState,userinfo:object){
          state.userInfo = userinfo
      }
```

12 vue后台项目基于RBAC实现权限管理

12-1rbac权限机制和动态路由



什么是rbac权限管理机制?

“RBAC是一套成熟的权限模型。在传统权限模型中,我们直接把权限赋予用户。而在RBAC中,增加了“角色”的概念,我们首先把权限赋予角色,再把角色赋予用户。这样,由于增加了角色,授权会更加灵活方便。”

1 系统用户

2 角色管理

3 权限管理

后台管理系统的权限管理有有哪些?

菜单权限

路由权限

按钮权限

后台权限

什么是动态路由?

路由需要分成两类，**静态路由**和**动态路由**。静态路由是任何菜单权限下都能查看的界面路由；动态路由是根据菜单权限动态生成的路由集合。我们可能有一个 `User` 组件，它应该对所有用户进行渲染，但用户 ID 不同，我们分配不同的菜单权限

为什么使用匹配动态路由?

能模块的访问，本质上是由路由实现的，点击菜单只是途径之一，所以权限控制放在路由上是合理的,计算出来的路由的访问权限。这个计算出来的路由，并不是可预知的。就没法在全量里面先定义了，这种场景是必须要用动态路由的

vue-router4动态路由API有哪些?

- `router.addRoute(route: RouteRecord)`：动态添加路由
- `router.removeRoute(name: string | symbol)`：动态删除路由
- `router.hasRoute(name: string | symbol)`: 判断路由是否存在

- `router.getRoutes()`：获取路由列表

了解下vue-router4动态路由

getRoutes()就可以获取当前已配置的路由的数组

在路由守卫中添加 console.log(router.getRoutes());打印


```
router.beforeEach((to,from,next)=>{
  const token = localStorage.getItem('token')
```

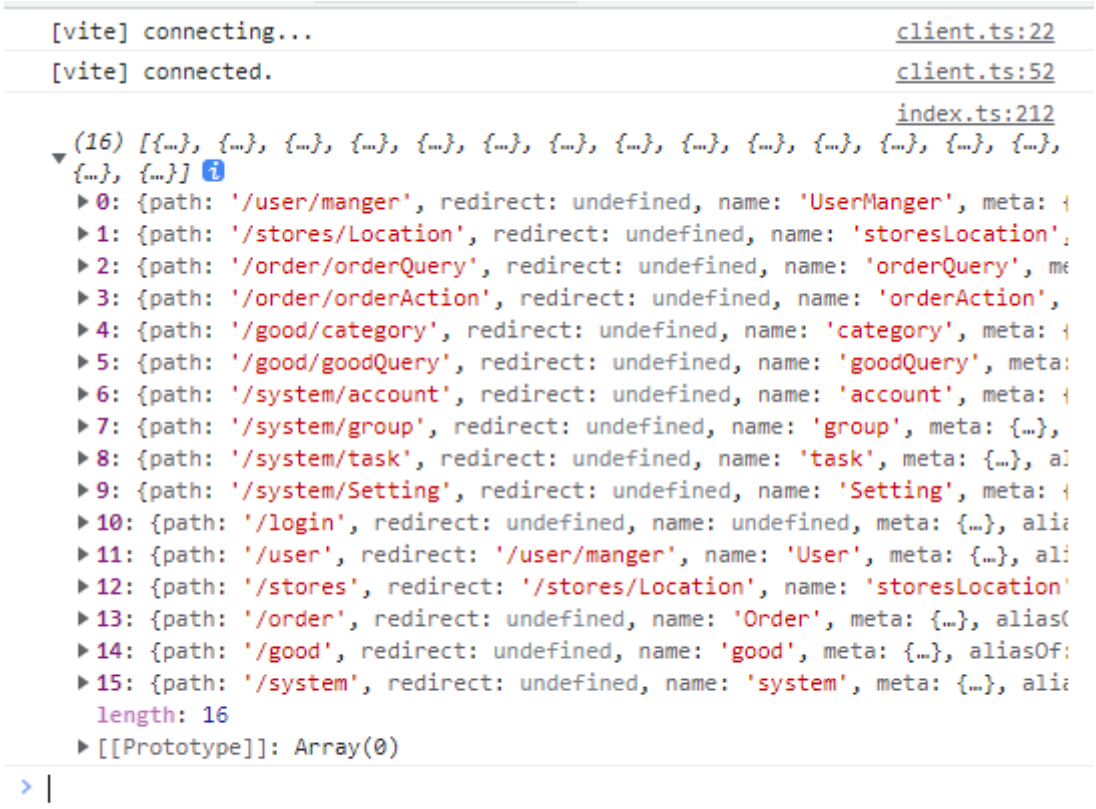
```
  if(!store.state.authStore.token && !token) {
      if(to.path.startsWith('/login'))
      next()
      else {
        next('/login')
      }
  } else if(!store.state.authStore.token && token) {
    loginByToken(token).then(res=>{
      if(res.data.status) {
       store.commit('authStore/addUserInfo',res.data)
       console.log(router.getRoutes());

        next()
      } else{
        next('/login')
      }

    })

  } else {
    next()
  }
})
```

hasRoute方法，它接受一个参数，路由的名称，来判断在已配置的路由里有没有这个路由，返回的是个布尔值

```
console.log(router.hasRoute('Index'))
```

removeRoute移除当前配置的单个路由

```
router.removeRoute("Index")

 console.log(router.getRoutes())
```



router.addRoute是vue-router4的改变之前添加的是数组,现在变成只能添加单条路由了

```
router.addRoute({
  path:'/wong',
  name:'wong',
  component: () => import('@/views/login/Login.vue')
})
  console.log(router.getRoutes())
```

12-2 vue3全局变量

在vue2中全局变量是prototype

在vue3中使用globalProperties

比如引入elementPlus的组件作为全局变量,

```
import * as ElementUI from 'element-plus'


// 声明全局变量属性
declare module '@vue/runtime-core' {
    interface ComponentCustomProperties {
        $confirm : (a:string)=>Promise<void>
        $Alert: any
        $Notify:any
    }
  }

//   声明全局变量
app.config.globalProperties.$confirm= ElementUI.ElMessageBox.confirm
app.config.globalProperties.$Alert = ElementUI.ElMessageBox.alert
app.config.globalProperties.$Notify = ElementUI.ElNotification
```

在main.ts中声明后,如何使用?

报错很好理解 因为 `getCurrentInstance()` 的返回类型存在 `null` 所以在此处添加断言即可

```
import { ComponentInternalInstance, getCurrentInstance } from 'vue';
// 添加断言
const { proxy } = getCurrentInstance() as ComponentInternalInstance
```

使用as定义数据类型,还需要使用proxy?.来排除null

```
proxy?.$confirm
```

12-3 request拦截器

后台管理权限的核心是角色

1 员工页

员工管理->给员工分配角色

2 角色页

角色管理->给角色分配权限

第一步,需要引入页面需要的api

这里已经给大家安排好了

引入即可

```
import {
  addSysUser,
  deleteSysUsers,
  getAllSysUsers,
  updateStatus,
  updateSysUser,
  resetPassword,
  setRole
} from '@/api/system/user'
import { getAllRoles } from '@/api/system/role'
```

我们通过在页面生命周期初始化使用getAllSysUsers()调用数据,

但是回发现报错,因为这是我们服务器在验证用户过程中,没有收到token指令

但是我们获取用户数据的时候,我们需要告诉服务器我们现在是已经登录的状态,

所以我们发送请求的时候需要在header附带token值,来验证我们当前用户

所以我们就需要request拦截器

```
// request拦截器
axiosInstance.interceptors.request.use(requestInfo => {

    if(requestInfo.headers) {
        requestInfo.headers['token'] = localStorage.getItem('token') || "0";
        requestInfo.headers['Content-Type'] = 'application/json;charset=UTF-8';
    return requestInfo
    }
},
error => {
    return Promise.reject(error)
}
)
```

12-4 员工页表格

用户列表

这里需要注意:

自定义表格使用v-slot插槽

default是el-tabel默认自带

并且通过scope.row获取当前行的数据

然后内置一些按钮,

如果按钮有超出一行,我们通过定义width适当提高宽度

```
   <div>
      <el-table :data="state.users">
        <el-table-column prop="username" label="用户名"></el-table-column>
        <el-table-column prop="roleName" label="角色"></el-table-column>
        <el-table-column label="状态">
          <template #default="scope">
            <el-switch v-model="scope.row.status" active-color="green" inactive-
color="red"></el-switch>
          </template>
        </el-table-column>
        <el-table-column label="操作" width="220">
          <template #default="scope">
            <el-button type="text" size="small">授权</el-button>
            <el-button type="text" size="small">重置密码</el-button>
            <el-button type="text" size="small">编辑</el-button>
            <el-button type="text" size="small">删除</el-button>
          </template>
        </el-table-column>
        <el-table-column></el-table-column>
      </el-table>
    </div>
```

script方面声明下state,然后给state.users赋值即可

```
const state = reactive({
  users: []
})


onMounted(() => {
  getUsers()
})

// 定义获取表格数据
const getUsers = () => {
  getAllSysUsers().then(result => {
    state.users = result.data

  })
```

12-5员工页分页

各位同学,

分页我们使用Pagination组件

任何一个组件离不开属性和事件

属性

设置`layout`，表示需要显示的内容

`prev`表示上一页，`next`为下一页，`pager`表示页码列表

`jumper`表示跳页元素，`total`表示总条目数，`size`用于设置每页显示的页码数量。

current-page 当前页数，支持 v-model 双向绑定

total总条目数

page-size 每页显示个数选择器的选项设置

事件

current-change `current-change` 改变时触发

size-change `pageSize` 改变时触发

```
      <el-row style="float: right">
         <el-pagination
           @current-change="handleCurrentChange"
           @size-change="handleSizeChange"
           :current-page="state.currentPage"
           :page-sizes="[5, 10, 20, 30, 50, 100]"
           layout=" total, sizes, prev, pager, next, jumper"
           :total="state.users.length"
         ></el-pagination>
       </el-row>
```

然后分页数据后的数据我们使用slice进行截取

slice可从已有的数组中返回选定的元素

比如slice(4,10)就是从第5个元素开始,第10个元素结束

因此在state中需要设置两个参数

```
 currentPage: 1,//当前页
  pageSize: 10,//每页显示条目个数
```

然后table的data数据这样绑定

```
:data="state.users.slice((state.currentPage - 1) * state.pageSize,
state.currentPage * state.pageSize)"
```

改变页和改变当前页的条数

```
// 改变页
const handelCurrentChange = (val:number)=>{
    state.currentPage =val
}

// 改变当前条数
const handelSizeChange = (val:number)=>{
  state.pageSize = val
}
```

12-6新增删除修改用户

添加用户按钮和弹窗


```
    <div style="text-align: left; margin: 5px 10px;">
          <el-button type="primary" @click="toAddUser">新增系统用户</el-button>
      </div>
      <!-- 新增用户form表单 -->
      <el-dialog :title="state.formTitle" v-model="state.userFormDialogVisible">
          <el-form
          ref="userForm"
          :model="state.userFormData"
          :rules="state.rules"
          label-width="100px"
          >
              <el-form-item label="用户名" prop="username">
                  <el-input v-model="state.userFormData.username" placeholder="请
输入用户名"></el-input>
              </el-form-item>
              <el-form-item>
                <el-button @click="resetForm"> 重置</el-button>
                <el-button type="primary" @click="handelConfirm"> 确定</el-
button>
              </el-form-item>

          </el-form>
      </el-dialog>
```

响应式数据声明

```
const state = reactive({
  users:[],
 currentPage:1,//当前页
 pageSize:10,//每页显示10条
 userFormDialogVisible:false,
 userFormData:{
   username:''
 },
 rules:{
   username:[
     {
       required:true,
       message:'请输入用户名',
       trigger:'blur'
     }
   ]
```

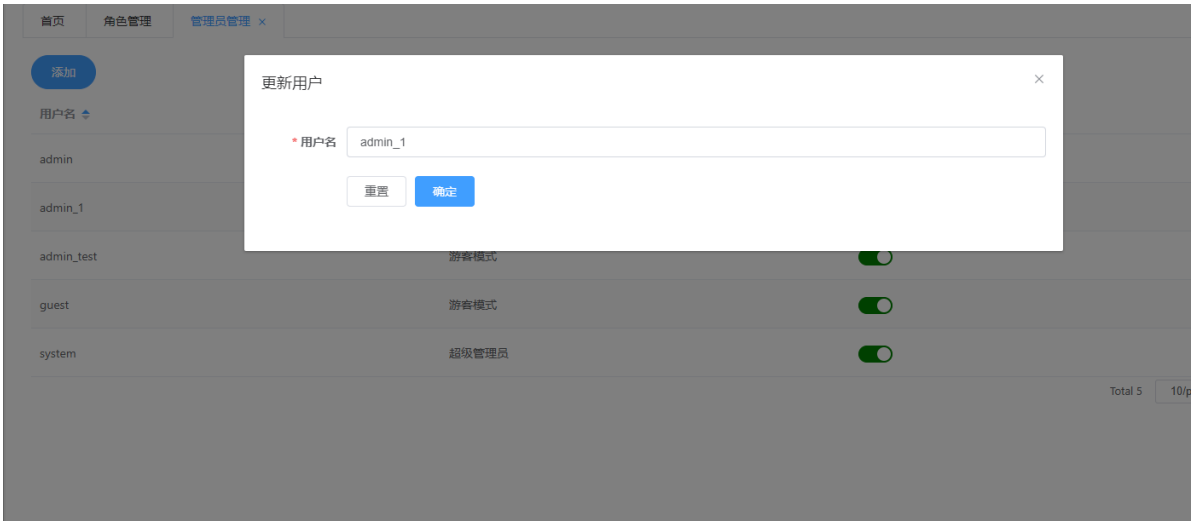
```
 },
 formTitle:''
})
```

新增删除修改用户

```
// 点击添加用户按钮
const toAddUser = ()=>{
    state.formTitle ='添加用户'
    state.userFormDialogVisible = true
}

// 确认添加或修改用户
const handelConfirm = ()=>{
  if(state.formTitle.startsWith('添加用户')) {
    addSysUser(state.userFormData).then(res=>{
      console.log(res);
      if(res.data == 1) {
        proxy?.$Alert('添加成功')
        getUsers()
        state.userFormDialogVisible = false
      }

  })
  } else if(state.formTitle.startsWith('修改用户')) {
    updateSysUser(state.userFormData).then(result =>{
       proxy?.$Alert('更新成功')
        getUsers()
       state.userFormDialogVisible = false
    })
  }
}

// 重置用户信息
const resetForm = ()=>{
  state.userFormData = {
    username:''
  }
}

// 删除用户
const deleteUser = (id:number)=>{
    proxy?.$Confirm("确认要删除当前用户吗?").then(()=>{
      deleteSysUsers(id).then(()=>{
        proxy?.$Notify.success({
          title:'删除成功',
          duration:500
        })
        getUsers()
      })
    })
}

// 修改用户按钮
const toEditUser = (selecteUser:object)=>{
    state.userFormData = JSON.parse(JSON.stringify(selecteUser))
```

```
    state.formTitle = '修改用户'
    state.userFormDialogVisible = true
}
```

12-7 修改用户状态



这里主要使用三目运算法

首先在switch组件写一个change,

通过value这个布尔值的变化触发函数commitStatusChange

```
              <template #default="scope">
                    <el-switch
                    v-model="scope.row.status"
                    active-color="green"
                    inactive-color ="red"
                    @change="(value:boolean)=>
commitStatusChange(value,scope.row)"
                    >

                    </el-switch>
               </template>
```

commitStatusChange函数

```
// 更新用户状态
const commitStatusChange = (value:boolean,user:User) =>{
    proxy?.$Confirm(value===false ? '冻结用户?' : '激活用户?').then(()=>{
      updateStatus(user.id,user.status).then(()=>{
        proxy?.$Notify.success(value===false ? '已冻结' : '已激活')
      })
    })
}
```

12-8 授权角色

完善button按钮

```
              <template #default="scope">
                  <el-button type='text' size="small"
@click="toSetRole(scope.row.id)">授权</el-button>
                  <el-button type='text' size="small"
@click="resetPw(scope.row.id)">重置密码</el-button>
                  <el-button  type='text' size="small"
@click="toEditUser(scope.row)">编辑</el-button>
                  <el-button type='text' size="small"
@click="deleteUser(scope.row.id)">删除</el-button>
              </template>
```

首先,重设密码

```
// 重设密码
const resetPw = (userId:number) =>{
  proxy?.$Confirm('重置该用户密码,请谨慎操作!').then(()=>{
    resetPassword(userId).then(()=>{
      proxy?.$Notify.success('密码已经重置成功')
    })
  })
}
```

授权dialog弹窗

这里引入select和option两个组件

```
       <el-dialog title="设置角色" v-model="state.showSetRoleDialog">
            <el-select @change="upRole" value="选择角色" placeholder="选择角色">
                <el-option v-for="item in roles" :key="item.id"
:label="item.name" :value="item.id"></el-option>
            </el-select>
       </el-dialog>
```

整体授权业务

```
// 设置角色按钮
const toSetRole = (userId:number) =>{
  getAllRoles().then(result=>{
```

```
    roles.value = result.data
    state.currentUserId = userId
    state.showSetRoleDialog = true
  })
}

// 确认设置角色
const upRole = (roleId:number) =>{
  setRole(state.currentUserId,roleId).then(result=>{
     if(result.data === 1) {
       proxy?.$Notify.success('角色设置成功')
       getUsers()
       state.showSetRoleDialog = false

     }

  })
}
```

13 角色页

13-1 角色表单

角色页非常简单,页面table数据渲染,然后是角色的增删改查

引入接口

```
import { deleteRoles, getAllRoles, updateRole, updateRolePermission, addRole,
getPermissionsOfRole } from '@/api/system/role'
```


```
   <!-- 然后开发表格用户角色列表 -->
    <div style="margin: 5px 0px;text-align: left;">
      <el-table :data="state.roles" stripe :max-height="700">
        <el-table-column prop="name" label="角色名称" sortable></el-table-column>
        <el-table-column prop="description" label="描述"></el-table-column>
        <el-table-column label="操作" width="150">
          <template #default="scope">
            <el-button @click="toSetPermissions(scope.row)" type="text"
size="small">授权</el-button>
            <el-button @click="edit(scope.row)" type="text" size="small">编辑
</el-button>
            <el-button @click="delRoles(scope.row.id)" type="text" size="mini">删
除</el-button>
          </template>
        </el-table-column>
      </el-table>
    </div>
```

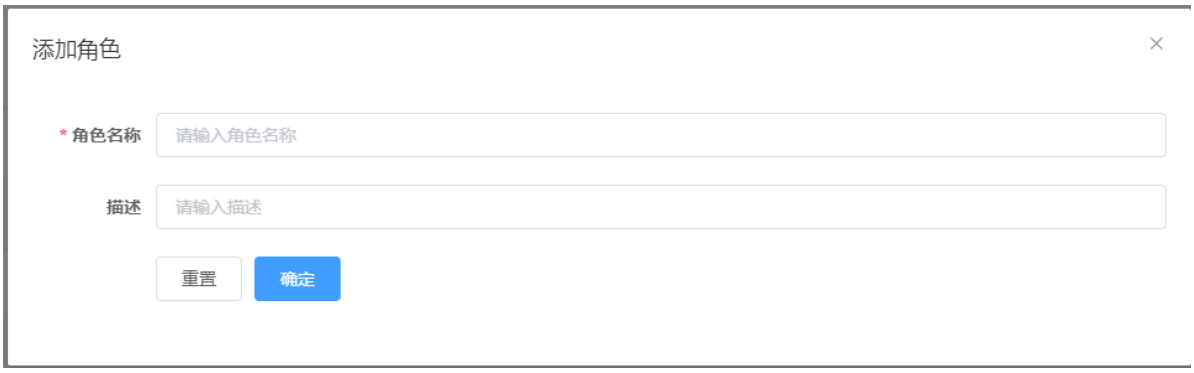
在script中,生命周期中添加查询

```
import { deleteRoles, getAllRoles, updateRole, updateRolePermission, addRole,
getPermissionsOfRole } from '@/api/system/role'
import { onMounted, reactive } from 'vue';

const state = reactive({
  roles: [],

})

// 初始化
onMounted(() => {
  _getAllRoles()
})

// 获取所有角色
const _getAllRoles = () => {
```

```
  getAllRoles().then(result => {
    console.log(result);

    state.roles = result.data
  })
}
```

13-2 角色增删改

然后添加和修改角色弹窗




```
    <el-dialog :title="state.formTitle" v-model="state.dialogVisible">
      <el-form
        ref="elForm"
        :model="state.formData"
        :rules="state.rules"
        size="medium"
        label-width="100px"
      >
        <el-form-item label="角色名称" prop="name">
          <el-input
            v-model="state.formData.name"
            placeholder="请输入角色名称"
            :maxlength="20"
            :style="{ width: '100%' }"
          ></el-input>
        </el-form-item>
        <el-form-item label="描述">
          <el-input
            v-model="state.formData.description"
            placeholder="请输入描述"
            :style="{ width: '100%' }"
          ></el-input>
        </el-form-item>

        <el-form-item>
          <el-button @click="resetForm">重置</el-button>
          <el-button type="primary" @click="handelConfirm">确定</el-button>
        </el-form-item>
      </el-form>
    </el-dialog>
```

增删改查业务

```
// 页面初始化
onMounted(() => {
  _getAllRoles()
})

// 获取角色数据
const _getAllRoles = () => {
  getAllRoles().then(result => {
```

```
    state.roles = result.data
  })
}

// 添加角色
const toAdd = () => {
  state.dialogVisible = true;
  resetForm()
  state.formTitle = '添加角色';
}

// 重置表单
const resetForm = () => {
  state.formData = {
    id:1,
    name: "",
    description: ""
  };
}
// 确认角色信息
const handelConfirm = () => {
  state.dialogVisible = false;
  if (state.formTitle === "添加角色") {
    addRole(state.formData).then(() => {
      _getAllRoles();
    })
  } else if (state.formTitle === "更新角色") {
    updateRole(state.formData).then(() => {
      _getAllRoles();
    })
  }
}


// 角色编辑
const edit = (selectedRole: object) => {
  // 深拷贝一个对象 不然在表格显示的数据会受到印象
  state.formData = JSON.parse(JSON.stringify(selectedRole));
  state.dialogVisible = true
  state.formTitle = '更新角色'
}
// 角色删除
const delRoles = (id: number) => {

  proxy?.$confirm("是否删除角色").then(() => {
    deleteRoles([id]).then(() => {
      proxy?.$Notify.success('删除成功')
      _getAllRoles();
    })
  }).catch(() => {
  })
}
```

13-3 树形控件组件

树形控件有很多种,

我们的需求就是有两种

1 可选择

2 默认选中已有权限

3 选中后改变后端数据

官网https://element-plus.gitee.io/zh-CN/component/tree.html

首先我们引入最简单的el-tree

并且给数据

```
const data = [
  {
    label: 'Level one 1',
    children: [
      {
        label: 'Level two 1-1',
        children: [
          {
            label: 'Level three 1-1-1',
          },
        ],
      },
    ],
  },
  {
    label: 'Level one 2',
    children: [
      {
        label: 'Level two 2-1',
        children: [
          {
            label: 'Level three 2-1-1',
          },
        ],
      },
      {
        label: 'Level two 2-2',
        children: [
          {
            label: 'Level three 2-2-1',
          },
        ],
      },
    ],
  },
  {
    label: 'Level one 3',
    children: [
      {
        label: 'Level two 3-1',
```

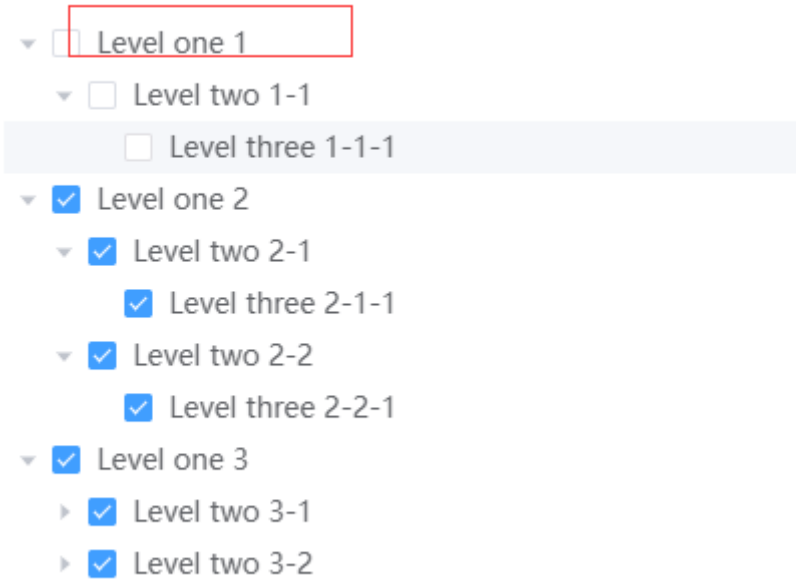
```
        children: [
          {
            label: 'Level three 3-1-1',
          },
        ],
      },
      {
        label: 'Level two 3-2',
        children: [
          {
            label: 'Level three 3-2-1',
          },
        ],
      },
    ],
  },
]
```

data可以绑定数据

那么我们需要的选择框该如何展示?

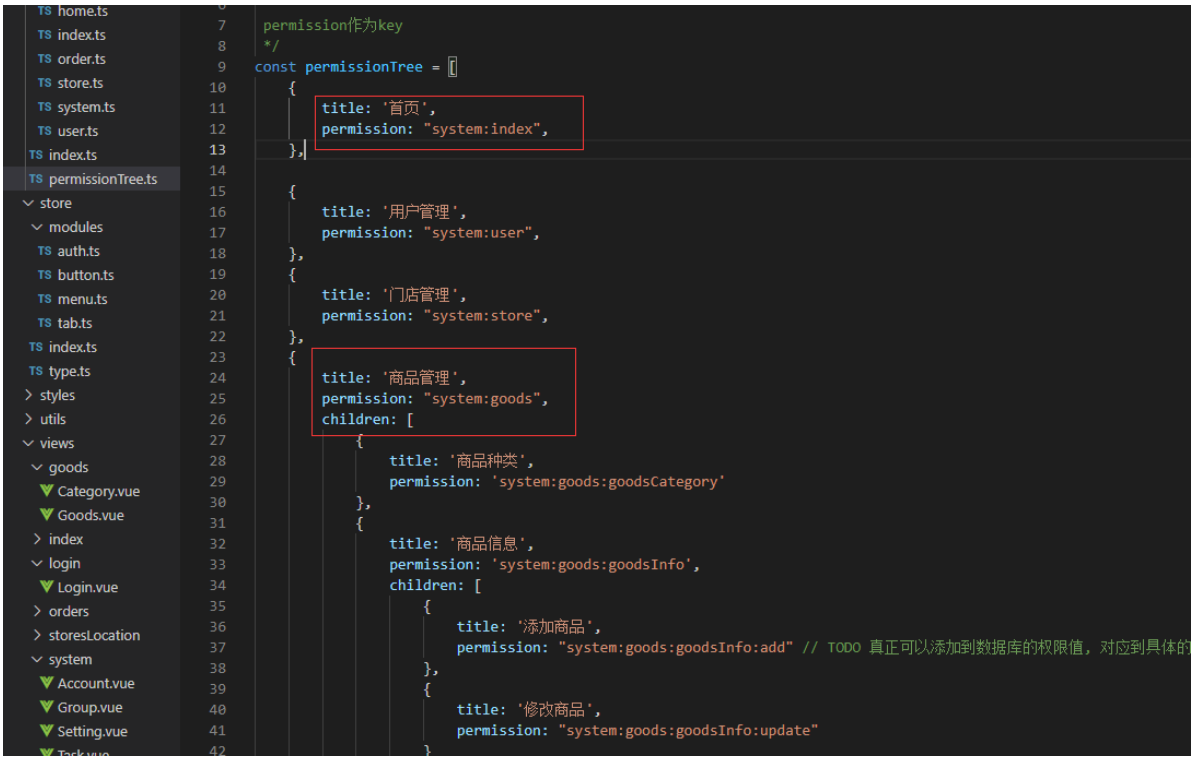
通过添加show-checkbox

然后如果我们给的数据不是id label children这样的数据展示实例该如何?



```
// 虚拟数据
const data = [
  {
    label: 'Level one 1',
    children: [
      {
        label: 'Level two 1-1',
        children: [
          {
            label: 'Level three 1-1-1',
          },
        ],
      },
    ],
  },
  {
    label: 'Level one 2',
```

比如我们给的数据对象节点属性名称为title permission



这个时候我们就是要props来定义树形控件的映射规则

```
defaultProps: { // 树形控件值映射规则
    id: 'title', // 作为key
    label: 'title',
    children: 'children'
  },
```

通过模板声明,很快就能得到所需数据展示

```
</el-dialog>
<!-- 角色授权弹窗 -->
<el-dialog title="角色授权" v-model="state.showSetPermissionDialog">
    <el-tree
    :data="state.permissionTree"
    :props="state.defaultProps"
    show-checkbox>

    </el-tree>
</el-dialog>
```

必须确认当前用户的权限是属于哪些勾选的节点,并且让当前用户已经有权限的必须勾选出来

使用default-checked-keys绑定permissions

```
default-checked-keys="state.permissions"
```

在点击授权的时候自动向后端获取用户的权限,并使checkbox选择状态

```
// 点击授权按钮
const toSetPermissions = (selectedRole:any) => {
     state.permissions = []
    state.formData = JSON.parse(JSON.stringify(selectedRole));
  state.showSetPermissionDialog = true

  getPermissionsOfRole(selectedRole.id).then(result => {


      state.permissions = result.data;
```

```
  })
}
```

属性	作用	数据类型
data	展示数据	array
node-key	每个树节点用来作为唯一标识的属性，整棵树应该是唯一的	string
props	配置选项，具体看下表	object
show-checkbox	节点是否可被选择	boolean
default-checked-keys	默认勾选的节点的 key 的数组	array

Props	说明	类型	可选值	默认值
label	指定节点标签为节点对象的某个属性值	string, function(data, node)	—	—
children	指定子树为节点对象的某个属性值	string	—	—
disabled	指定节点选择框是否禁用为节点对象的某个属性值	string, function(data, node)	—	—
isLeaf	指定节点是否为叶子节点，仅在指定了 lazy 属性的情况下生效	string, function(data, node)	—	—
class	自定义节点类名	string, function(data, node)	—	—

13-4 获取拥有权限的默认节点

模板部分,我们使用default-checked-keys绑定默认选择的节点,

并且通过node-key确认permission属性为key

```
  <el-dialog title="角色授权" v-model="state.showSetPermissionDialog"
    destroy-on-close>
          <el-tree
          :data="state.permissionTree"
          show-checkbox
          :props="state.defaultProps"
          node-key="permission"
          :default-checked-keys="state.permissions">

          </el-tree>
    </el-dialog>
```

然后在点击权限按钮进行赋值即可

```
// 点击授权按钮
const toSetpermissions= (id:number)=>{
   state.permissions=[]
  state.showSetPermissionDialog = true
  getPermissionsOfRole(id).then(result=>{
        console.log(result)
        state.permissions = result.data
  })
}
```

13-5 确认和取消授权

角色授权对话框

```
   <!--角色授权对话框-->
    <el-dialog title="角色授权" v-model="state.showSetPermissionDialog"
    destroy-on-close>
      <el-form>
        <el-form-item>
          <!--树形控件 配置角色菜单-->
          <el-tree
            :data="state.permissionTree"
            show-checkbox
            node-key="permission"
            :props="state.defaultProps"
            ref="permissionRef"
            :default-checked-keys="state.permissions"

          ></el-tree>
        </el-form-item>
        <el-form-item>
          <el-button @click="resetChecked">清空</el-button>
          <el-button type="primary" @click="setPermissions">确定</el-button>
        </el-form-item>
      </el-form>
    </el-dialog>
```

角色事件,主要包括 点击 取消 确认

```
// 点击授权按钮
const toSetPermissions = (selectedRole:any) => {
     state.permissions = []
    state.formData = JSON.parse(JSON.stringify(selectedRole));
  state.showSetPermissionDialog = true

  getPermissionsOfRole(selectedRole.id).then(result => {


      state.permissions = result.data;


  })
}


// 取消授权
const resetChecked = () => {

    permissionRef.value!.setCheckedKeys([],false)


}

// 确认授权
const setPermissions = () => {
```

```
  let nodes = permissionRef.value!.getCheckedNodes(false, false)
  console.log(nodes)
  let permissions: InstanceType<typeof ElTree>[] = [];
  nodes.forEach(node => {
    if (node.permission) {
      permissions.push(node.permission);
    }
  })
  let vo = {
    roleId: state.formData.id,
    permissions: permissions
  }

  type resType = {
    code :number,
    message:string,
    data:any[]
  }
  updateRolePermission(vo).then(() => {

     proxy?.$Notify.success('操作成功')
     state.showSetPermissionDialog = false;


  })
}
```

14 菜单权限和按钮权限

14-1 menu模块

每个账号登录对应的动态路由和动态菜单都不一样

动态菜单通过vuex创建modules/menu.ts模块

```
import {Module} from 'vuex'
import {RootState} from '../index'
import { asyncRoutes } from '@/router/modules'
import { RouteRecordRaw } from 'vue-router'

export interface MenuState {
    menuList:RouteRecordRaw[]
}

export const menuStore: Module<MenuState,RootState> ={
    namespaced:true,
    state:():MenuState =>({
        menuList:[]
    }),

    getters:{
        getMenus:state => state.menuList
    },
    mutations:{
        setMenus(state,systemMenu) {
            state.menuList = systemMenu
        }
    },

    actions: {
        generateSystemMenus({commit,state},perm:string[]) {

        }
    }
}
```

在store/index.ts中注册

```
import { createStore,Store,useStore as baseUseStore} from "vuex"
import { InjectionKey } from 'vue'
import {tabStore,TabState} from './modules/tabs'
import { authStore,AuthState } from "./modules/auth"
import { menuStore,MenuState } from "./modules/menu"


export interface RootState {
    tabStore:TabState,
    authStore:AuthState,
    menuStore:MenuState

}

export const key: InjectionKey<Store<RootState>> = Symbol()
```

```
export const store:Store<RootState> = createStore({
    modules:{
        tabStore,
        authStore,
        menuStore
    }
})

// 自定义组合式函数
export function useStore() {
    return baseUseStore(key)
}
```

14-2 路由赋值menu菜单

首先permissions权限对比路由meta.permission,

找到相同的保留并且push

```
 function hasPermission(permissions:string[], needPermission:string) {
    for (let i = 0; i < permissions.length; i++)
        if (permissions[i].startsWith(needPermission))
            return true;
    return false;
}


function filterRouter(routes:RouteRecordRaw[], perms:string[]) {
    const res:RouteRecordRaw[]= []
    routes.forEach(route => {
        if (route.children) {
            route.children = filterRouter(route.children, perms) // 递归处理孩子节
点
            if (route.children && route.children.length > 0)
                res.push(route)
        } else {
            if (route.meta!.permission) { // 需要权限

                if (hasPermission(perms, route.meta!.permission ))
                    res.push(route);
            } else {
                res.push(route)
            }
        }
    })
    return res
}
```

在actions中引入对比函数

```
 generateSystemMenus({ commit ,state},perm:string[]) {
            let routers = filterRouter(asyncRoutes, perm);
           routers.forEach(route=>{
               if(route.redirect == null&&route.children?.length==1) {
                    route.redirect = route.path+ '/' +route.children[0].path
                    route.meta = route.children[0].meta
               }else {
                router.addRoute(route)
               }

           })
           console.log(routers)
           commit('setMenus',routers)
        }
```

在menuBar/index.vue中引入store

并且给响应式数据menus赋值为store.getters['menuStore/getMenus'],

切记,使用getters计算属性的时候要用 -中括号-

```
import {useStore} from '@/store'

const store = useStore()
const menus = store.getters['menuStore/getMenus']
```

解决登录和刷新时state中menuList添加数据

在auth.ts中给账号密码登录login和loginBytToken登录添加menuStore/generateSystemMenus

```
store.dispatch("menuStore/generateSystemMenus", result.data.permissions)
```

这样登录就会自动加载动态菜单

不过,如果刷新vuex中的menuList数据就会消失

所以最后在前置路由守卫刷新中也添加menuStore/generateSystemMenus

不过这里要注意:在配置路由的时候typescript需要给路由的meta配置属性

注意,提示meta类型为RouteMeta||undefine

需要**补充的类型**

```
declare module 'vue-router' {
  interface RouteMeta {
    // 是可选的标题
    title: string
    // 每个路由都必须声明
    icon?: string
    permission: string
  }

}
```

其次,路由守卫获取动态路由需要重定向

```
if(to.matched.length == 0) {
          router.push(to.path)
       }
```

最后如果有二级菜单,当二级菜单中只剩下一条菜单数据,我们让二级菜单变成一级菜单

```
   // 添加到动态路由
                routers.forEach(route=>{
                    // 二级menu跳成一级menu
                    if(route.redirect == null && route.children?.length == 1) {
                        route.redirect = route.path +'/' +
route.children[0].path
                        route.meta = route.children[0].meta
                    }
                    router.addRoute(route)
                })


                // 添加动态菜单
                commit('setMenus',routers)
```

14-3 vue自定义指令directive

除了核心功能默认内置的指令 (例如 `v-model` 和 `v-show`)，Vue 也允许注册自定义指令。注意，在 Vue 中，代码复用和抽象的主要形式是组件。然而，有的情况下，你仍然需要对普通 DOM 元素进行底层操作，这时候就会用到自定义指令

在vue3中通过app.directive声明自定义指令

官网

https://v3.cn.vuejs.org/guide/custom-directive.html#%E7%AE%80%E4%BB%8B

在main.ts中

```
// 按钮权限-自定义指令
app.directive('btn',{
// 当被绑定的元素被挂载到dom中时
    mounted(el, binding) {
        let buttonList=['a','b']
      // 判断是否有某个按钮权限，没有的话就删除该按钮
      if (!buttonList.includes(binding.value)) {
        // 原生方法 找父级 parentNode，删除子节点 removeChild
        el.parentNode.removeChild(el)
      }
    }
  })
```

按钮中引入

```
<div style="text-align: left; margin: 5px 10px">
      <el-button type="primary" v-btn="'c'" round>添加</el-button>
    </div>
```

按钮设置为c 而 buttonList中只有 a b所以这个按钮就不显示

我们在上面定义指令的时候，会发现其中包含了 `mounted` 钩子函数，指令还提供了如下钩子函数，我们用代码的形式来给大家列出来。

```
app.directive('directiveName', {
  // 指令绑定元素挂载前
  beforeMount(el) {},
  // 指令绑定元素挂载后
  mounted(el, binding) {},
  // 指令绑定元素因为数据修改触发修改前
  beforeUpdate(el) {},
  // 指令绑定元素因为数据修改触发修改后
  updated(el) {},
  // 指令绑定元素销毁前
  beforeUnmount(el) {},
  // 指令绑定元素销毁后
  unmounted(el) {},
});
```

14-4 按钮权限

为了方便控制按钮权限在vuex中

定义一个新模块button.ts

```
import { Module } from "vuex";
import { RootState } from "../index";

export interface ButtonState {
    buttonList:string[]
}



export const buttonStore:Module<ButtonState, RootState> = {
    namespaced: true,
    state: (): ButtonState => ({
        buttonList:[],

    }),
    mutations:{
        addButton(state:ButtonState,buttons) {
                state.buttonList = buttons
        }
    },
    getters: {
        getbuttons: state => state.buttonList
    },
    actions:{
        generateButtons({commit,state},buttons:string[]) {
            let bList:string[] = []
            buttons.forEach(button=>{

                if(button.match(/:/g)?.length === 3){
                    bList.push(button)
                }



            })
            commit('addButton',bList)
        }
    }
}
```

index.ts中引入

```
import { InjectionKey } from "@vue/runtime-core"
import { createStore, Store, useStore as baseUseStore } from "vuex"
import { authStore,AuthState } from "./modules/auth"
import { tabStore, TabState } from './modules/tab'
import { menuStore,MenuState } from "./modules/menu"
import {buttonStore,ButtonState} from './modules/button'
```

```
export interface RootState {

  TabStore: TabState,
  authStore:AuthState,
  menuStore:MenuState,
  buttonStore:ButtonState
}

export const key: InjectionKey<Store<RootState>> = Symbol()

export const store: Store<RootState> = createStore({
  modules: {
    tabStore,
    authStore,
    menuStore,
    buttonStore
  },




})


export function useStore() {
  return baseUseStore(key)
}
```

在main.ts中定义directive

```
// 按钮权限-自定义指令
app.directive('btn',{
    mounted(el, binding) {
      // 判断是否有某个按钮权限，没有的话就删除该按钮
      if (!store.state.buttonStore.buttonList.includes(binding.value)) {
        // 原生方法 找父级 parentNode，删除子节点 removeChild
        el.parentNode.removeChild(el)
      }
    }
  })
```

在按钮添加v-btn来控制按钮显示,如果没有权限就不显示

```
 <!-- 商品添加按钮 -->
    <div style="text-align: left; margin: 5px 10px">
      <el-button type="primary" v-btn="'system:goods:goodsInfo:add'" round>添加
</el-button>
    </div>
```

15 i18n国际化和全屏

15-1 初次使用vue-i18n

3.2版本使用安装需要加@next

```
npm i vue-i18n@next
```

在src目录中创建i18n/index.ts

```
import { createI18n } from "vue-i18n";

const i18n = createI18n({
    locale: 'chs', // set locale
    legacy: false,
    globalInjection:true,
    messages: {
      chs: {
        message: {
          hello: '你好世界',
        },
      },
      en: {
        message: {
          hello: 'hello world',
        },
      },
    },
  })
```

```
  export default i18n
```

在main.ts中引入

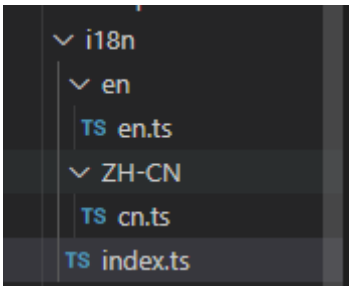
```
import i18n from './i18n'
const app = createApp(App)
for (const name in ElIcons){
    app.component(name,(ElIcons as any)[name])
}
app.use(router).use(store,key).use(ElementUI).use(i18n).mount('#app')
```

在模板中引入message

```
<template>
  <div>Task</div>
  <div>
    <p>{{ $t('message.hello') }}</p>
  </div>
</template>

<script setup lang='ts'>

</script>
<style lang='scss' scoped>
</style>
```

15-2 i18n三种使用场景

首先引入中英文数据源



引入完成后在i18n/index.ts中配置数据源,这里创建一个message的对象是未来使用的时候缩短链式


```
import { createI18n } from "vue-i18n";
import CN from "./ZH-CN/cn";
import EN from "./en/en";

const message = {
  chs:{
    ...CN
  },
  en:{
    ...EN
  }
}
const i18n = createI18n({
    locale: 'chs', // set locale
    legacy: false, //支持compositionApi
    globalInjection:true,// 全局注册 $t方法
    messages: message
  })
```

```
  export default i18n
```

1 之前我们已经了解了在template模板中使用的方法

$t

```
<p>{{ $t('menus.wGoodsManger') }}</p>
```

2 在ts后缀的文件中要如何使用呢?这里我们在路由的配置信息页面为例

这里通过引入i18n

import i18n from '@/i18n'

然后通过

i18n.global.t('menus.wIndex')

以上的方法使用

```
import Layout from '@/layout/index.vue'
import i18n from '@/i18n'
import {
  RouteRecordRaw,
} from "vue-router";
const homeRouter:RouteRecordRaw= {
  path: '/',
  redirect: '/index',
  name: 'Index',
  component: Layout,
  meta: {
    title: i18n.global.t('menus.wIndex'),
    icon: 'house',
    permission: "system:index",
  },
  children: [
    {
      path: 'index',
      name: 'Index',
      component: () => import('@/views/index/Index.vue'),
      meta: {
        title: i18n.global.t('menus.wIndex'),
        icon: 'house',
        permission: "system:index",
      },

    },
  ],
}

export default homeRouter
```

3在script中使用

引入

import { useI18n } from "vue-i18n";

提取t

const {t} = useI18n()

使用

t('menus.wIndex')

这里以面包屑为例

```
<!--  -->
<template>
  <el-breadcrumb >

    <el-breadcrumb-item    v-for="(item, index) in breadcrumbs"
        :key="index">
        {{item.meta.title}}

        </el-breadcrumb-item>
  </el-breadcrumb>
</template>

<script setup lang="ts">
import { useRoute, RouteLocationMatched,useRouter} from 'vue-router'
import { watch,ref,Ref,onMounted } from 'vue'
import { useI18n } from "vue-i18n";

const route = useRoute()
const router = useRouter()
const {t} = useI18n()
const  breadcrumbs:Ref<RouteLocationMatched[]> = ref([])


  // 获取面包屑数据
 const getBreadcrumb = () => {
    let matched = route.matched.filter((item) => item.meta && item.meta.title &&
item.children.length !== 1)
    const frist = matched[0]
    if (frist.path !== '/index') {
      matched = [{ path: '/index', meta: { title: t('menus.wIndex') } } as
any].concat(matched)

    }
      breadcrumbs.value  = matched
  }




// 初始化启动
onMounted(() => {
  getBreadcrumb()
})

watch(()=>route.path,() => {
  getBreadcrumb()
})
```

```
</script>
<style lang='scss' scoped>
</style>
```

15-3 切换中英文

首先,在header模块创建LanguageBar.vue

```
<!--  -->
<template>
    <div>
        <el-dropdown :show-timeout="10" :hide-timeout="10" placement="right"
trigger="click">
            <div>
                <img src="../../../assets/images/lang.png" style="width: 25px;
height: 25px;">
            </div>
            <template #dropdown>
                <el-dropdown-menu>
                    <el-dropdown-item @click="toLang('zh')"
:disabled="getCurrentLang == 'zh'">中文</el-dropdown-item>
                    <el-dropdown-item
                        @click="toLang('en')"
                        :disabled="getCurrentLang == 'en'"
                    >English</el-dropdown-item>
                </el-dropdown-menu>
            </template>
        </el-dropdown>
    </div>
</template>

<script setup lang='ts'>
import { computed } from 'vue';
import { useI18n } from 'vue-i18n';
import { useStore } from '@/store';

import { asyncRoutes } from '@/router/modules';
const i18n = useI18n()
const store = useStore()

const getCurrentLang = computed(() => {
    return useI18n().locale.value
})
const toLang = (lang: string) => {


    i18n.locale.value = lang
    localStorage.setItem('language', lang)

    console.log(asyncRoutes)

}
</script>
<style lang='scss' scoped>
</style>
```

在index.vue中引入LanguageBar

```
<!--  -->
<template>
```

```
    <el-row :gutter="24" style="display: flex; align-items: center">
      <el-col :span="18" :offset="0" >
        <bread-crumb />
      </el-col>

      <el-col :span="2">
        <language-bar />
      </el-col>
      <el-col :span="4" >
        <user-bar />
      </el-col>


    </el-row>



</template>

<script setup lang="ts">
import BreadCrumb from './BreadCrumb.vue'
import UserBar from './UserBar.vue'
import LanguageBar from './LanguageBar.vue';

</script>
<style lang='scss' scoped>
.headerBox{
  display: flex;
  flex-direction: row;
  justify-content: space-between;


}
</style>
```

这样就能成功切换中英文,

但是,问题来了,之前ts文件和script 中配置的t变量只有刷新才能切换中英文,

也就是,menu 面包屑 tab都需要手动更改,才能完成中英文切换

OK,没办法顺便把大部分buton按钮也都转成中英文

en.ts

```
export default {
   menus: {
    wIndex:'Index',
    wUserManger:'UserManger',
    wGoodsManger:'GoodsManger',
    wGoodsCategory:'GoodsCategory',
    wGoodsInfo:'GoodsInfo',
    wOrderManger:'OrderManger',
    wOrderQuery:'OrderQuery',
    wOrderAction:'OrderAction',
    wStoreManger:'StoreManger',
    wSystemAdmin:'SystemAdmin',
```

```
    wSystemManger:'SystemManger',
    wSystemRole:'SystemRole',
    wSystemTask:'SystemTask',
    wSystemSetting:'SystemSetting'
   },
   button:{
         wLogin:'login',
         wLogout:'logout',
         wcloseCurrentTab: "Close CurrentTab",
         wcloseLeftTabs: "Close LeftTabs",
         wcloseRightTabs: "Close RightTabs",
         wcloseOtherTabs: "Close OtherTabs",
         wcloseAllTabs: "Close AllTabs",
         wadd:'add',
         wdelete:'delete',
         wedit:'edit',
         wempower:'empower',
         wresetPassWord:'resetPassWord'
   }

}
```

cn.ts

```
export default {
    menus:{
        wIndex:'首页',
    wUserManger:'用户管理',
    wGoodsManger:'商品管理',
    wGoodsCategory:'商品种类',
    wGoodsInfo:'商品查询',
    wOrderManger:'订单管理',
    wOrderQuery:'订单查询',
    wOrderAction:'订单处理',
    wStoreManger:'门店管理',
    wSystemAdmin:'系统管理',
    wSystemManger:'管理员管理',
    wSystemRole:'角色管理',
    wSystemTask:'定时任务',
    wSystemSetting:'系统设置'
    },
    button:{
          wLogin:'登录',
          wLogout:'退出登出',
          wcloseCurrentTab: "关闭所有",
          wcloseLeftTabs: "关闭左边",
          wcloseRightTabs: "关闭右边",
          wcloseOtherTabs: "关闭其它",
          wadd:'添加',
          wdelete:'删除',
          wedit:'编辑',
          wempower:'授权',
          wresetPassWord:'重置密码'
    }

}
```

15-4用screenfull实现全屏

本身全屏用原生docment操作也可以,但是有插件用,代码更加方便,省了很多事

首先安装下screenfull第三方插件

```
npm i screenfull
```

看看screenfull如何使用

在header创建screenFull.vue

```
<!--  -->
<template>
  <div @click="onToggle">
        <img v-if="isFullscreen" src="../../../assets/images/notFull.png"
style="width: 25px; height: 25px;">
        <img v-else src="../../../assets/images/isFull.png" style="width: 25px;
height: 25px;">
  </div>
</template>

<script setup lang='ts'>
import { onMounted, onUnmounted, ref } from 'vue'
import screenfull from 'screenfull'
// 是否全屏
const isFullscreen = ref(false)

// 监听变化
const change = () => {
  isFullscreen.value = screenfull.isFullscreen
}

// 切换事件
const onToggle = () => {
  screenfull.toggle()
}

// 设置监听器
onMounted(() => {
  screenfull.on('change', change)
})

// 删除监听器
onUnmounted(() => {
  screenfull.off('change', change)
})

</script>
<style lang='scss' scoped>
</style>
```

同样在,header的index引入即可

```
<!--  -->
<template>

    <el-row :gutter="24" style="display: flex; align-items: center">
      <el-col :span="18" :offset="0" >
        <bread-crumb />
      </el-col>
      <el-col :span="1">
          <screenfull-vue />
      </el-col>
      <el-col :span="1">
        <language-bar />
      </el-col>
      <el-col :span="4" >
        <user-bar />
      </el-col>


    </el-row>



</template>

<script setup lang="ts">
import BreadCrumb from './BreadCrumb.vue'
import UserBar from './UserBar.vue'
import LanguageBar from './LanguageBar.vue';
import screenfullVue from './screenfull.vue';
</script>
<style lang='scss' scoped>
.headerBox{
  display: flex;
  flex-direction: row;
  justify-content: space-between;


}
</style>
```

16 echart5基础

16-1初步使用echarts5

随着 Vue 更新到了 V3 版本，ECharts 更新到了 V5 版本，它们均新增加了对typescript的开发方式与原先 Vue V2 版本仍然具有一些区别，所以本文在此给出一个新版的开发样例。

安装

```
npm install echarts --save
```

使用方法一

基于准备好的dom,初始化实例

```
<!--  -->
<template>
  <div>
      <div id="main" style="width: 35%; height: 300px;"></div>
  </div>
</template>

<script setup lang='ts'>
  import * as echarts from 'echarts'
import { onMounted } from 'vue';

  onMounted(()=>{
    init()
  })
  const init = ()=>{

    let mychart = echarts.init(document.getElementById('main') as HTMLElement)
      var option = {
    title: {
      text: '产品销量排行'
    },
    tooltip: {},
    legend: {
      data: ['销量']
    },
    xAxis: {
      data: ['衬衫', '羊毛衫', '雪纺衫', '裤子', '高跟鞋', '袜子']
    },
    yAxis: {},
    series: [
      {
        name: '销量',
        type: 'bar',
        data: [5, 20, 36, 10, 10, 20]
      }
    ]
  };
  // 给予数据源
  mychart.setOption(option)
  }
</script>
<style lang='scss' scoped>
</style>
```

方法二 通过ref虚拟化dom

```
<!--  -->
<template>
  <div>
    <div style="margin: 15px 25px;">Echart5可视化</div>
    <div ref="main" style="width: 33%; height: 300px"></div>
  </div>
</template>

<script setup lang='ts'>
import * as echarts from "echarts";
import { onMounted, ref } from "vue";

const main = ref()
onMounted(
  () => {
    init()
  }
)
function init() {
  // 使用ref创建虚拟DOM引用，使用时用main.value
  var myChart = echarts.init(main.value);
  // 指定图表的配置项和数据
  var option = {
    title: {
      text: '产品销量排行'
    },
    tooltip: {},
    legend: {
      data: ['销量']
    },
    xAxis: {
      data: ['衬衫', '羊毛衫', '雪纺衫', '裤子', '高跟鞋', '袜子']
    },
    yAxis: {},
    series: [
      {
        name: '销量',
        type: 'bar',
        data: [5, 20, 36, 10, 10, 20]
      }
    ]
  };
  // 使用刚指定的配置项和数据显示图表。
  myChart.setOption(option);
}

</script>
<style lang='scss' scoped>
</style>
```

总结:使用echart步骤

引入

注意这里使用`import * as echarts from 'echarts'`

- 先准备一个DOM元素，定义好高度和宽度
- 使用echarts.init()方法，初始化一个ECharts的实例,这里也可以使用ref虚拟dom
- 使用setOption设置配置项

16-2 Echars5的title属性

首先看属性

```
   title: {
          // 标题
          show: true, //是否显示
          text: "4545",
          link:'http://www.baidu.com',
          target:'self', //'self' 当前窗口打开 'blank' 新窗口打开
          x:'center',

          textStyle: {
            color: "#fff", // 主标题文字的颜色。
            fontStyle: "normal", // 主标题文字字体的风格。 'normal'  'italic'
'oblique'
            fontWeight: "normal", // 主标题文字字体的粗细。 'normal' 'bold'
'bolder'  'lighter' 500|600
            fontFamily: "sans-serif", // 主标题文字的字体系列。
            fontSize: 18, // 字体大小

          },
          subtext: "bb", // 副标题文本

          subtextStyle:{

        }
        },
```

16-3 legend属性

legend是图表的图例

以下是单独配置的方法

```
legend:{

     show:true,
       icon:'circle',
     top:'10%',
     itemWidth:10,

     textStyle:{
       color:'red',
       fontSize:20,
       backgroundColor:'yellow'
     }

   },
```

legend配置多name样式,

data能控制显示的系列名,比show:true控制更加精确

```
 legend: {
     data: [{
       name: '南山',
       // 强制设置图形为圆。
       icon: 'circle',
       // 设置文本为红色
       textStyle: {
         color: 'red'
       }
     }, {
       name: '福田',
       // 强制设置图形为圆。
       icon: 'circle',
       // 设置文本为红色
       textStyle: {
         color: 'blue'
       }
     },
     ]


   },
```

16-4 tooltip属性

tooltip是什么?

是我们**提示框组件**

鼠标滑过,或者点击显示的信息

z注意?

trigger 触发方式

item axis

triggerOn 触发时机

'mousemove|click' 分别代表移动到触发 和点击触发

formatter 格式化

- 折线（区域）图、柱状（条形）图、K线图：`{a}`（系列名称），`{b}`（类目值），`{c}`（数值），`{d}`（无）
- 散点图（气泡）图：`{a}`（系列名称），`{b}`（数据名称），`{c}`（数值数组），`{d}`（无）
- 地图：`{a}`（系列名称），`{b}`（区域名称），`{c}`（合并数值），`{d}`（无）
- 饼图、仪表盘、漏斗图: `{a}`（系列名称），`{b}`（数据项名称），`{c}`（数值），`{d}`（百分比）

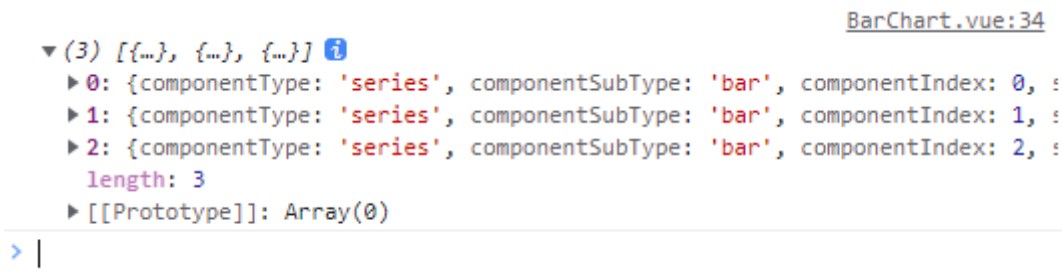
```
tooltip: {
    trigger:'axis',
    triggerOn:'click',
    formatter:'{a}{b}{c}'
  },
```

formatter的格式化从 {a}{b}{c}的顺序往下,无需去记

主要是formatter的格式化回调函数格式:

```
 tooltip: {
    trigger:'axis',
    triggerOn:'click',
    formatter:(params:Array<any>)=>{
        console.log(params)
    }
  },
```

回调格式,是表单数据,这样的目标是为了能更加自由的定制回调显示



16-5 toolbox属性

内置有导出图片，数据视图，动态类型切换，数据区域缩放，重置五个工具

```
   toolbox: {
    feature: {
      saveAsImage: {},
      dataView:{},
      restore:{},
      dataZoom:{},
      magicType:{
        type:['bar','line']
      }
    }
  },
```

16-6 水平柱状图

柱状图的配置主要series中进行配置,

1是水平柱状图

柱状图的配置很简单,有水平的垂直的,

控制垂直还是水平,在于对xaxis和yaxis的配置.

默认data给到哪,哪行就是属性轴,默认的type:'category'就是配置在哪行,另一行就是type:'value'

```
  xAxis: {
      data: ['衬衫', '羊毛衫', '雪纺衫', '裤子', '高跟鞋', '袜子']
    },
    yAxis: {},
```

2 聚合柱状图

经常用到聚合柱状图，以更直观地比较各维度信息,不同维度数据配置在series中,

一个对象就为一个维度数据

```
{
        name: '销量1',
        type: 'bar',
        stack:'111',
         label:{
              show:true,
              position:'top'
           },
           itemStyle:{
             color:'red'
           },
        data: [5, 20, 36, 10, 10, 20]
      },
      {
        name: '销量2',
        type: 'bar',
        stack:'112',
        data: [5, 20, 36, 10, 10, 20]
      }
```

3堆叠柱状图

一般对比各个维度数据,可以考虑堆叠,这样更加清晰

在series中配置stack的值,如果值相同则柱状可以堆叠在一起

```
stack:'销量',
```

4柱状图样式

自定义颜色

itemStyle

color为颜色配置

16-7 折线图

普通折线图,在系列中设置

```
type:'line'
```

既可以从柱状图改为折线图

直线变曲线

```
smooth :true //平滑过渡 ,将直线变成曲线
```

内容填充

```
areaStyle:{

}
```

标记

分别使用markPoint和markLine可以标记

折线图的最大值和最小值,还有平均线

```
 markPoint:{
          data:[
            {type:'max',name:'最大值'},
            {type:'min',name:'最小值'}
          ]
       },
       markLine:{
         data:[
           {type:'average',name:'平均值'}
         ]
       },
```

如果要达到折线的堆叠效果也是一样,stack设置的值一样即可

然后,我们在选择折线图的时候可以给予选择高亮

```
emphasis:{
    focus:'series'
}
```

16-8 饼图

饼图

type:'pie',则可以设置charts图表为简单饼图

如果需要效果特色更好看点,可以选择

环形饼图和玫瑰饼图

环形

```
radius:['40%','70%'] //第一个参数内半径,第二个参数外半径
```

也可以通过label配置饼图的标签

```
label:{
    show:true,
    position:'' //这里和折线图不同,这里配置参数是outside和inside center 代表 外 里 中心
}
```

玫瑰pie

```
roseType:'area'
```

如果饼图想单独给属性设置color,相对麻烦,

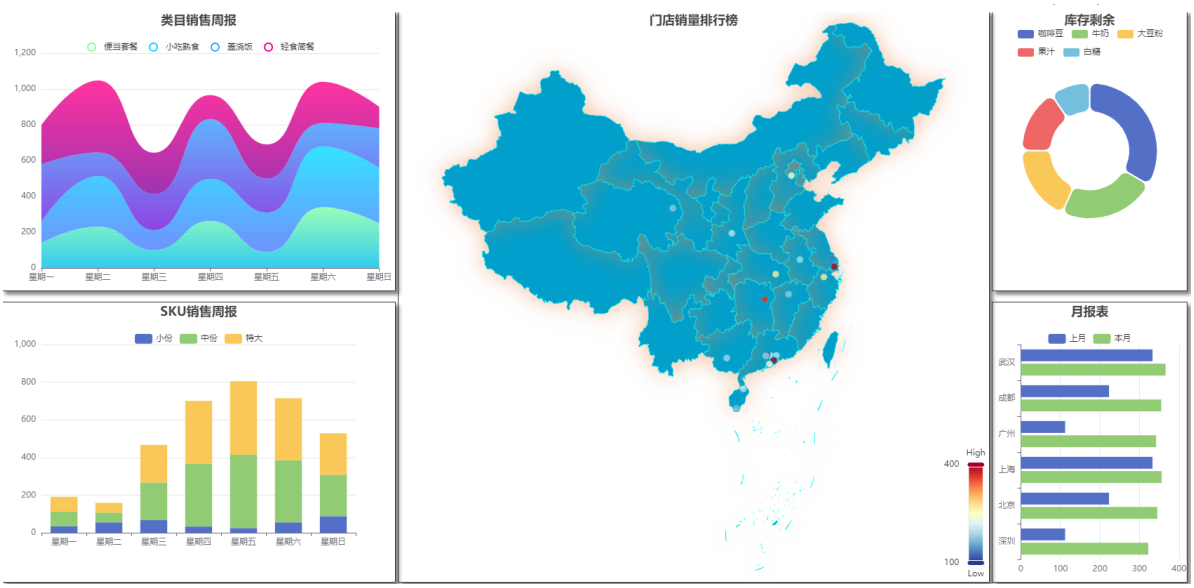
之前的折线和柱状都是在series中有多个不同对象可以在对象中配置itemStyle

而pie图是数组组成,需要配置的话要在data映射的数组中配置itemStyle

17 后台管理数据可视化

17-1页面布局

预览数据可视化




```
<el-row :gutter="3">
   <el-col :span="8" >
     <line-chart   />
     <bar-chart />
   </el-col>
   <el-col :span="12">
     <map-chart  :mapList="state.map"/>
   </el-col>
   <el-col :span="4">
     <pie-chart  :pieList="state.pie" />
     <bar-to-chart  :barToList="state.barTo" />
   </el-col>

 </el-row>
```

17-2 异步加载数据展示pie库存

首先echarts图表 通过setoptions加载,

常用的可是化一般是折线图 柱状图 饼图 地图和散点图

echarts画图，DOM渲染的时候，echarts实现先从后端获取数据再渲染

async await

获取数据API

pie 库存剩余

https://c4156a34-94b2-4302-969f-e96f6277625a.bspapp.com/pie

line 销售数据

https://c4156a34-94b2-4302-969f-e96f6277625a.bspapp.com/chart

bar 销售数据

https://c4156a34-94b2-4302-969f-e96f6277625a.bspapp.com/bar

barTo 销售

https://c4156a34-94b2-4302-969f-e96f6277625a.bspapp.com/barTo

地图

https://c4156a34-94b2-4302-969f-e96f6277625a.bspapp.com/map

我们通过api完成环形库存剩余饼图

```
<!--  -->
<template>
  <div>
     <div ref="main" style="width: 100%; height: 400px; margin: 15px 0;
border:1px solid #333; box-shadow: 5px 5px 5px #888888;"></div>
  </div>
</template>

<script setup lang='ts'>
import * as echarts from 'echarts'
import { onMounted, ref, } from 'vue';
import axios from 'axios'

const main = ref()
const data =ref([])
onMounted(()=>{
  init()
})
const init= async ()=>{
    await axios.get('https://c4156a34-94b2-4302-969f-
e96f6277625a.bspapp.com/pie').then(res=>{
        data.value = res.data
    })
  // 1通过dom初始化echarts 2ref虚拟化dom
  let mychart =echarts.init(main.value)
  // 数据源
  var option = {
```

```
        title:{
            text:'库存剩余',
            x:'center'
        },
        tooltip:{
            trigger:'item'
        },
        legend:{
            top:'5%',
            left:'center'
        },
        series:[
            {
                name:'库存',
                type:'pie',
                radius:['40%','70%'],
                itemStyle:{
                    borderRadius:10,
                    borderColor:'#fff',
                    borderWidth:2
                },
                label:{
                    show:false,
                    position:'center'
                },
                emphasis:{
                    label:{
                        show:true,
                        fontSize:'40',
                        fontWeight:'bold'
                    }
                },
                data:data.value
            }
        ]
  };

  // 数据源给予
  mychart.setOption(option)
}

</script>
<style lang='scss' scoped>
</style>
```

17-3 类目销售折线图

折线图,

需要注意

areaStyle折线图堆叠

有设置透明度和颜色渐变

```
<!--  -->
<template>
    <div>
        <div
            ref="main"
            style="width: 100%; height: 400px; margin: 15px 0; border:1px solid
#333; box-shadow: 5px 5px 5px #888888;"
        ></div>
    </div>
</template>

<script setup lang='ts'>
import * as echarts from 'echarts'
import { onMounted, reactive, ref, } from 'vue';
import axios from 'axios'

const main = ref()
const lineList = reactive({
    xdata: [],
    Bento: [],
    snack: [],
    bowl: [],
    dining: []
})
onMounted(() => {
    init()
})
const init = async () => {
    await axios.get('https://c4156a34-94b2-4302-969f-
e96f6277625a.bspapp.com/chart').then(res => {

        lineList.xdata = res.data.xdata
        lineList.Bento = res.data.Bento
        lineList.snack = res.data.snack
        lineList.bowl = res.data.bowl
        lineList.dining = res.data.dining
    })
    // 1通过dom初始化echarts 2ref虚拟化dom
    let mychart = echarts.init(main.value)
    // 数据源
    var option = {
        title: {
            text: '类目销售周报',
            x: 'center'
        },
        color: ['#80FFA5', '#00DDFF', '#37A2FF', '#FF0087', '#FFBF00'],
        tooltip: {
```

```
            trigger: 'axis',
            axisPointer: {
                type: 'cross',
                label: {
                    backgroundColor: '#6a7985'
                }
            }
        },
        legend: {
            top: '10%'
        },
        xAxis: {
            type: 'category',
            //   刻度线boundary
            boundaryGap: false,
            data: lineList.xdata
        },
        yAxis: {
            type: 'value'
        },
        series: [
            {
                name: '便当套餐',
                type: 'line',
                //    堆叠
                stack: '销量',
                // 平滑
                smooth: true,
                lineStyle: {
                    width: 0
                },
                // 隐藏所有数据点
                showSymbol: false,
                areaStyle: {
                    // 透明度
                    opacity: 0.8,
                    // 颜色渐变
                    color: new echarts.graphic.LinearGradient(0, 0, 0, 1, [
                        {
                            offset: 0,
                            color: 'rgb(128,255,165)'
                        },
                        {
                            offset: 0,
                            color: 'rgb(1,191,236)'

                        }

                    ])
                },
                emphasis: {
                    focus: 'series'
                },
                data: lineList.Bento

            },{
                name: '小吃熟食',
                type: 'line',
```

```
            //    堆叠
            stack: '销量',
            // 平滑
            smooth: true,
            lineStyle: {
                width: 0
            },
            // 隐藏所有数据点
            showSymbol: false,
            areaStyle: {
                // 透明度
                opacity: 0.8,
                // 颜色渐变
                color: new echarts.graphic.LinearGradient(0, 0, 0, 1, [
                    {
                        offset: 0,
                        color: 'rgb(0,211,255)'
                    },
                    {
                        offset: 0,
                        color: 'rgb(77,119,255)'

                    }

                ])
            },
            emphasis: {
                focus: 'series'
            },
            data: lineList.snack

        },{
            name: '盖浇饭',
            type: 'line',
            //    堆叠
            stack: '销量',
            // 平滑
            smooth: true,
            lineStyle: {
                width: 0
            },
            // 隐藏所有数据点
            showSymbol: false,
            areaStyle: {
                // 透明度
                opacity: 0.8,
                // 颜色渐变
                color: new echarts.graphic.LinearGradient(0, 0, 0, 1, [
                    {
                        offset: 0,
                        color: 'rgb(55,162,255)'
                    },
                    {
                        offset: 0,
                        color: 'rgb(116,21,219)'

                    }
```

```
                    ])
                },
                emphasis: {
                    focus: 'series'
                },
                data: lineList.bowl

            },{
                name: '轻食简餐',
                type: 'line',
                //    堆叠
                stack: '销量',
                // 平滑
                smooth: true,
                lineStyle: {
                    width: 0
                },
                // 隐藏所有数据点
                showSymbol: false,
                areaStyle: {
                    // 透明度
                    opacity: 0.8,
                    // 颜色渐变
                    color: new echarts.graphic.LinearGradient(0, 0, 0, 1, [
                        {
                            offset: 0,
                            color: 'rgb(255,0,135)'
                        },
                        {
                            offset: 0,
                            color: 'rgb(135,0,157)'

                        }

                    ])
                },
                emphasis: {
                    focus: 'series'
                },
                data: lineList.dining

            }
        ]

    };

    // 数据源给予
    mychart.setOption(option)
}

</script>
<style lang='scss' scoped>
</style>
```

17-4sku销售周报

柱状图

```
<!--  -->
<template>
  <div>
     <div ref="main" style="width: 100%; height: 400px; margin: 15px 0;
border:1px solid #333; box-shadow: 5px 5px 5px #888888;"></div>
  </div>
</template>

<script setup lang='ts'>
import * as echarts from 'echarts'
import { onMounted, reactive, ref, } from 'vue';
import axios from 'axios'

const main = ref()
const state =reactive({
    xdata:[],
    small:[],
    medium:[],
    large:[]
})
onMounted(()=>{
  init()
})
const init= async ()=>{
    await axios.get('https://c4156a34-94b2-4302-969f-
e96f6277625a.bspapp.com/bar').then(res =>{
         state.xdata = res.data.xdata
         state.small = res.data.small
         state.medium = res.data.medium
         state.large = res.data.large
    })
  // 1通过dom初始化echarts 2ref虚拟化dom
  let mychart =echarts.init(main.value)
  // 数据源
  var option = {
      title:{
          text:'SKU销售周报',
          x:'center'
      },
      tooltip:{
          trigger:'item'
      },
      legend:{
          show:true,
          top:'10%'
      },
      xAxis:{
          type:'category',
          data:state.xdata
      },
      yAxis:{
```

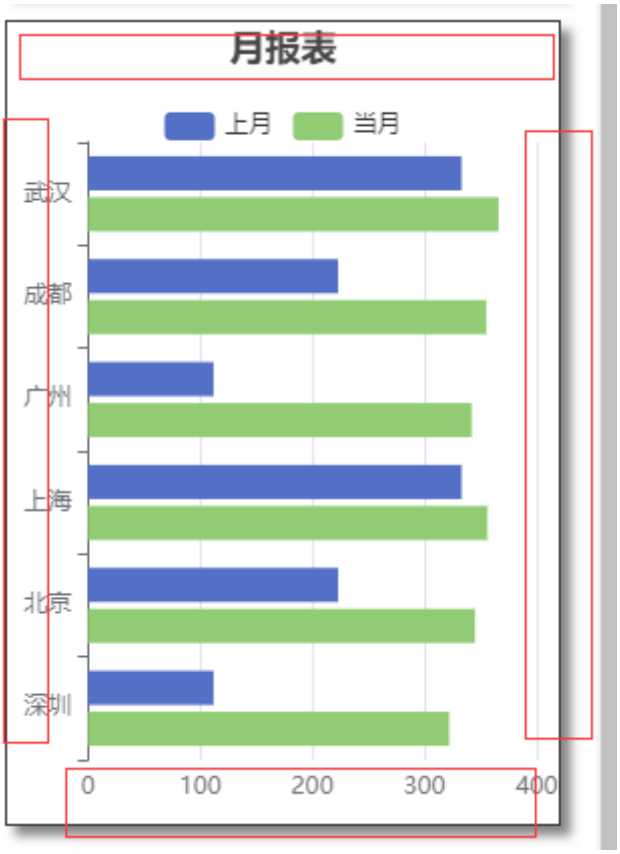
```
      },
      series:[
          {
              name:'小份',
              type:'bar',
              stack:'销量',
              barWidth:'60%',
              data:state.small

          },{
              name:'中份',
              type:'bar',
              stack:'销量',
              barWidth:'60%',
              data:state.medium

          },{
              name:'大份',
              type:'bar',
              stack:'销量',
              barWidth:'60%',
              data:state.large

          }
      ]

  };

  // 数据源给予
  mychart.setOption(option)
}

</script>
<style lang='scss' scoped>
</style>
```

17-5 月报表

学过之前的报表

月报表没什么太大技术含量,

唯独布局需要根据自己业务情况



grid,设置

控制上下左右的边距

```
<!--  -->
<template>
  <div>
     <div ref="main" style="width: 100%; height: 400px; margin: 15px 0;
border:1px solid #333; box-shadow: 5px 5px 5px #888888;"></div>
  </div>
</template>

<script setup lang='ts'>
import * as echarts from 'echarts'
import { onMounted, reactive, ref, } from 'vue';
import axios from 'axios'

const main = ref()
const state = reactive({
    ydata:[],
    last:[],
    current:[]

})
onMounted(()=>{
```

```
  init()
})
const init= async ()=>{
    await  axios.get('https://c4156a34-94b2-4302-969f-
e96f6277625a.bspapp.com/barTo').then(res=>{
        state.ydata = res.data.ydata
        state.last = res.data.last
        state.current = res.data.current
    })
  // 1通过dom初始化echarts 2ref虚拟化dom
  let mychart =echarts.init(main.value)
  // 数据源
  var option = {
      title:{
          text:'月报表',
          x:'center'
      },
      tooltip:{
          trigger:'axis',
          axisPointer:{
              type:'shadow'
          }
      },
      legend:{
          show:true,
          top:'10%'
      },
      grid:{
          left:'3%',
          right:'4%',
          bottom:'3%',
          containLabel:true
      },
      xAxis:{
          type:'value',
        //   boundaryGap:[0,0.01]
      },
      yAxis:{
          type:'category',
          data:state.ydata
      },
      series:[
          {
              name:'上月',
              type:'bar',
              data:state.last
          },
          {
              name:'当月',
              type:'bar',
              data:state.current
          }
      ]
  };

  // 数据源给予
  mychart.setOption(option)
}
```

```
</script>
<style lang='scss' scoped>
</style>
```

17-6显示地图

地图 + 散点图

显示地图

1 获取中国地图数据

这里数据我们到阿里云可视化平台获取

https://datav.aliyun.com/portal/school/atlas/area_selector

2 注册地图数据

```
echarts.registerMap('chinaMap', chinaMap as any)
```

3 geo配置就可以显示地图

```
geo: {
            map: "chinaMap",
            type: 'map',
            zoom: 1.2,
            itemStyle: {
                areaColor: '#009fcc',
                borderColor: '#00ffff',
                shadowColor: 'rgba(230,130,70,0.5)',
                shadowBlur: 30,
                emphasis: {
                    focus: 'self',
                    areaColor: 'yellow',//鼠标选择区域颜色
                    shadowBlur: 20,
                    borderWidth: 0,
                    shadowColor: 'rgba(0, 0, 0, 0.5)'
                }
            }
```

```
        },
```

17-7散点图配合地图展示门店信息

获取到对应的接口,

生成散点图和散点图的视觉映射

```
<!--  -->
<template>
    <div>
        <div
            ref="mains"
            style="width: 100%; height: 815px; margin:15px 0 ; border: 1px solid
#333 ;box-shadow: 5px 5px 5px #888888;"
        ></div>
    </div>
</template>

<script setup lang='ts'>
import * as echarts from "echarts";
import axios from 'axios'
import { onMounted, ref,watch } from "vue";
import { chinaMap } from '../../../assets/map/china'

const mains = ref()
onMounted(
    () => {
        init()
    }
)

const props = defineProps({
    mapList:{
        type:Array,
        required:true,
        default:[]
    }
})
const data = ref([])

const init = async () => {
   await axios.get('https://c4156a34-94b2-4302-969f-
e96f6277625a.bspapp.com/map').then(res=>{
        data.value = res.data
   })
    console.log(props.mapList)
    let myChart = echarts.init(mains.value);
    echarts.registerMap('chinaMap', chinaMap as any)
    let option = {
        title:{
            text:'门店销量排行榜',
            x:'center'
        },
        tooltip: {
            trigger: 'item'
        },
        geo: {
```

```
            map: "chinaMap",
            type: 'map',
            zoom: 1.2,
            itemStyle: {
                areaColor: '#009fcc',
                borderColor: '#00ffff',
                shadowColor: 'rgba(230,130,70,0.5)',
                shadowBlur: 30,
                emphasis: {
                    focus: 'self',
                    areaColor: 'yellow',//鼠标选择区域颜色
                    shadowBlur: 20,
                    borderWidth: 0,
                    shadowColor: 'rgba(0, 0, 0, 0.5)'
                }
            }

        },
        visualMap: {
            left: 'right',
            min: 100,
            max: 400,
            inRange: {
                color: [
                    '#313695',
                    '#4575b4',
                    '#74add1',
                    '#abd9e9',
                    '#e0f3f8',
                    '#ffffbf',
                    '#fee090',
                    '#fdae61',
                    '#f46d43',
                    '#d73027',
                    '#a50026'
                ]
            },
            text: ['High', 'Low'],
            calculable: true
        },
        series: [{
            type: 'scatter', // 特效散点图，也可用普通散点图scatter
            coordinateSystem: 'geo', // 坐标系使用geo，以地图作为底图
            itemStyle: {
                borderColor: '#06b9d1',
                color: 'red'
            },
            data: data.value,
            encode: {
                value: 2
            },

        }]

    };
    myChart.setOption(option)
}
</script>
```

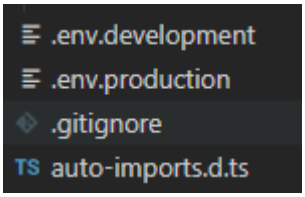
```
<style lang='scss' scoped>
</style>
```

18 打包上线

18-1 env环境配置

开发环境和生成环境之下

我们需要变量展示不同的信息



.env.development

.env.production

分别代表开发和生产两种环境的配置

axios之前访问的api是'api',

所以我们在.env.development配置

```
VITE_URL = '/api'
```

.env.production中配置对应的后端接口

```
VITE_URL = 'http://106.52.235.252:8101/'
```

最后在axios.ts中通过import.meta.env.VITE_URL配置不同环境的变量

```
const axiosInstance = axios.create({
    baseURL:import.meta.env.VITE_URL,

})
```

因为是ts,如果报无类型,在env.d.ts中配置类型

```

interface ImportMetaEnv {
  VITE_URL: string

}
```

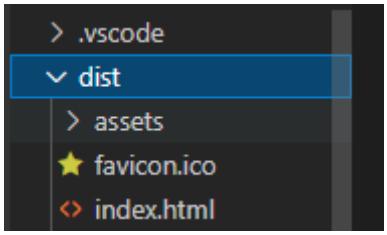
18-2 网站性能优化实战 gzip

使用

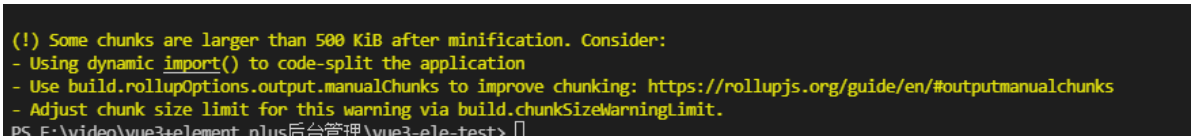
```
npm run build
```

即可打包

打包后默认放在dist文件夹中



如果一些chuck很大会保错



解决方法一:

加大限制的大小将500kb改成1000kb或者更大,默认限制红线是500kb

```
  build: {
    chunkSizeWarningLimit:1500,

        }
```

```
build.chunkSizeWarningLimit
类型： number

默认： 500

块大小警告的限制（以 kbs 为单位）。
```

解决方法二:

分解块，将大块分解成更小的块

```
rollupOptions: {
        output:{
            manualChunks(id) {
              if (id.includes('node_modules')) {
                  return id.toString().split('node_modules/')[1].split('/')
[0].toString();
              }
          }
        }
    }
```

不过这样分解后,你部署的时候就知道会有引入顺序的问题，部署会出错的，

上线会提示分割后提示 Cannot access 'S' before initialization

因为部分库会引用相同的依赖，导致最后的引入顺序可能会有问题。不像webpack那样，可以设置优先级以及根据引入次数进行打包,但实际这是只是视觉上的减小，其实他只不过是把大文件拆开来分成很多分了而已， 没什么实质性的作用

##

我们打包会遇到块包过大的问题,

官网提供拆分方案,只是视觉上的减少

什么是gzip?

Gzip 是在 Unix 和类 Unix 系统上使用的一种文件格式和软件应用程序，用于在 HTTP 内容提供给客户端之前对其进行压缩。众所周知，该过程最多可将文件缩小 80%，从而 缩短页面加载时间、减少带宽消耗并减少 SSL 开销（由于SSL 握手期间的往返次数减少 ）。

与 gzip 相关的文件类型包括：

.gz – 表示由 gzip 算法压缩的文件扩展名。
.tar 文件，tarball – 一种用于存储多个文件以供存档但不用于压缩的格式。Gzip 可用于压缩 .tar 文件。
.tgz、.tar.gz、.gz 文件 – 表示已被 gzip 压缩的 .tar 文件。

优化的目的?

最大限度地减少了交付给用户的代码文件的大小，从而显着减少了页面加载时间

如何优化?

这里我们使用

vite-plugin-compression

首先安装

```
npm i vite-plugin-compression -D
```

配置

```
import viteCompression from 'vite-plugin-compression';
plugins: [
    viteCompression()
  ],
```

再来使用

```
npm run build
```

然后nginx配置下gzip即可

```
#开启gzip功能
gzip on;
#开启gzip静态压缩功能
gzip_static on;
#gzip缓存大小
gzip_buffers 4 16k;
#gzip http版本
gzip_http_version 1.1;
#gzip 压缩级别 1-10
gzip_comp_level 5;
#gzip 压缩类型
gzip_types text/plain application/javascript text/css application/xml
text/javascript application/x-httpd-php image/jpeg image/gif image/png;
# 是否在http header中添加Vary: Accept-Encoding，建议开启
gzip_vary on;
```

nginx中配置gzip_static on提示nginx: [emerg] unknown directive “gzip_static“ i

此时可用在nginx的安装目录的sbin中使用`./nginx -V`查看当前nginx的配置信息，看有没有配置`--with-http_gzip_static_module`

通过以上信息中的configure arguments看出我们没有配置该信息。这时需要我们进入原来的nginx解压的目录中，进行配置并重新安装

```
## 配置
./configure --prefix=/usr/local/nginx --with-http_gzip_static_module
## 重新安装
make && make install
```

18-3 build 配置和nginx配置

vite打包编译上线,

nginx配置

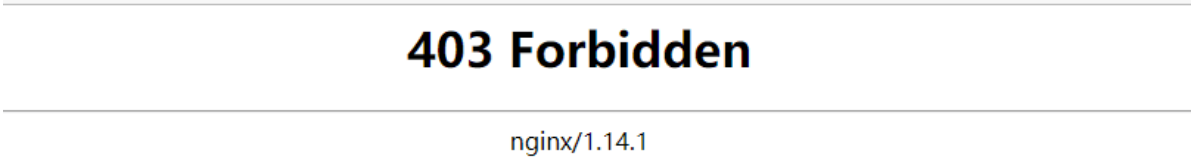
build编译打包配置

```
 build: {
      target: 'es2015',
      outDir: 'dist',
      assetsDir: 'assets',
      //小于此阈值的导入或引用资源将内联为 base64 编码，以避免额外的 http 请求。设置为 0 可以完全禁用此项。
      assetsInlineLimit: 2048,
      //css代码拆分
      cssCodeSplit: true,
      // Terser 相对较慢，但大多数情况下构建后的文件体积更小。ESbuild 最小化混淆更快但构建后的文件相对更大。
     // 设置为 false 可以禁用最小化混淆，或是用来指定使用哪种混淆器。默认为 Esbuild，它比 terser 快 20-40 倍，压缩率只差 1%-2%。
      minify:  'terser',
      terserOptions: {
        compress: {
          // 生产环境去除console
          drop_console: true
        }
      }
    }
```

nginx配置

因为我放到root这个目录,

首先我看到了403,这是目录没有权限




```
cd /root
ls -lt
chmod -R 777 el-admin
```

主要vite需要try_files重定向,否则会报错404 not fount

```
            root /root/el_admin/;
            index  index.html index.htm;
             try_files $uri $uri/ /index.html;
```

18-4 elementPlus切换为正式版打包问题

更新成正式版后两个问题

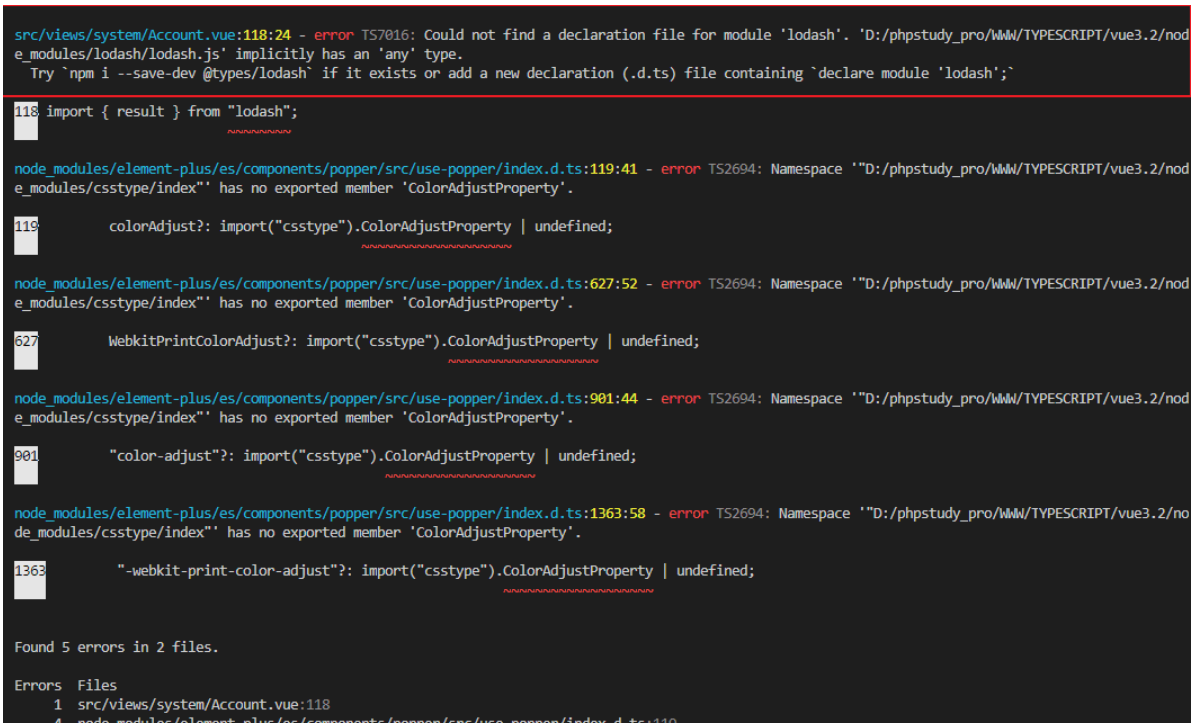
1 使用的时候发现ui有些许改变,input多了边框渐变,

2 打包后发现很多类型检测错误,

解决第一个问题:

input边框

在其他组件使用还好



但是这些错误是elementPlus这个library的问题,

我们不可能还去改第三方lib,

我们只需要怎么排除错误,又能成功打包

这个时候element plus官网也给了方法

在package.json中

build加一段--skipLibCheck,

这样就只检测自身,不检测第三方library

```
"build": "vue-tsc --noEmit --skipLibCheck && vite build",
```

19 页面切换动画

19-1 过渡动画Transition

transition用于过渡动画

设置位置appMain.vue,属于layout

```
<template>
  <div>
    <router-view />
  </div>
</template>

<script setup lang='ts'>
</script>
<style lang='scss' scoped>
</style>
```

目标就是给router-view添加过渡动画

vue-router官网

```
<router-view v-slot="{ Component }">
  <transition name="fade">
    <component :is="Component" />
  </transition>
</router-view>
```

```
v-enter-from:定义进入过渡的开始状态。在元素被插入之前生效，在元素被插入之后的下一帧移除
v-enter-active:定义进入过渡生效时的状态。在整个进入过渡的阶段中应用，在元素被插入之前生效，在过渡/动
画完成之后移除。这个类可以被用来定义进入过渡的过程时间，延迟和曲线函数
v-enter-to:定义进入过渡的结束状态。在元素被插入之后下一帧生效 (与此同时 v-enter-from 被移除)，在过渡/
动画完成之后移除。
v-leave-from:定义离开过渡的开始状态。在离开过渡被触发时立刻生效，下一帧被移除.
v-leave-active:定义离开过渡生效时的状态。在整个离开过渡的阶段中应用，在离开过渡被触发时立刻生效，在
过渡/动画完成之后移除。这个类可以被用来定义离开过渡的过程时间，延迟和曲线函数。
v-leave-to:离开过渡的结束状态。在离开过渡被触发之后下一帧生效 (与此同时 v-leave-from 被删除)，在过渡/
动画完成之后移除
```

name

transition name随意命名用来取代css中的v

所以我们appMain过渡动画可以这样,

mode

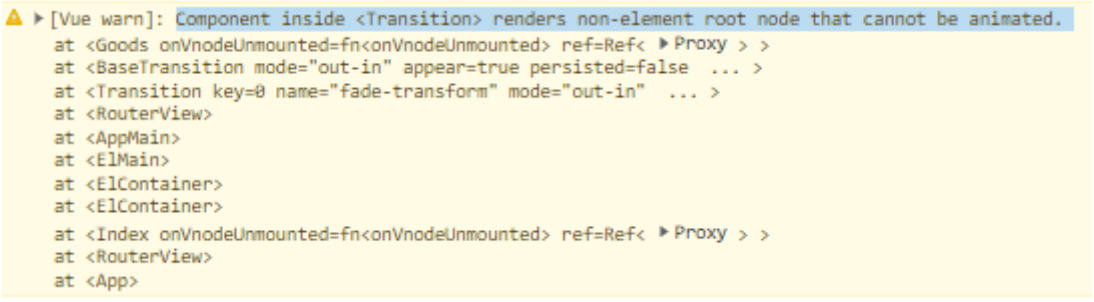
总共有 3 种过渡模式：

- default：淡入和淡出过渡同时发生
- in-out：新元素首先淡入。 然后当前元素淡出。
- out-in：当前元素先淡出。 然后新元素开始淡入。

```
<!--  -->
<template>
  <div>
       <router-view v-slot="{ Component }">
        <template v-if="Component">
          <transition :name="transitionName" mode="out-in" appear>

              <component :is="Component" />

          </transition>
        </template>
      </router-view>
  </div>
</template>

<script setup lang='ts'>
import { ref } from 'vue';
import { useRoute } from 'vue-router';
const transitionName= ref('fade-transform')
const route = useRoute()
</script>
<style lang='scss' scoped>
/* fade-transform */
.fade-transform-leave-active,
.fade-transform-enter-active {
  transition: all 0.3s;
}

.fade-transform-enter-from {
  opacity: 0;
  transform: translateX(-30px);
}

.fade-transform-leave-to {
  opacity: 0;
  transform: translateX(30px);
}

</style>
```

这样能够过渡,但是当访问到user.vue这个页面就会抱死

```
//报错信息
Component inside <Transition> renders non-element root node that cannot be
animated.
```



提示这个的原因是vite现在页面不需要加根目录标签

user.vue是这样的

```
<!--  -->
<template>
  <div>user</div>
</template>

<script setup lang='ts'>
</script>
<style lang='scss' scoped>
</style>
```

多加一层标签

```
<template>
  <div>
    <div>user</div>
  </div>
</template>
```

看看还会报错吗?

再访问user.vue页面就不报错了

结论:vite使用过渡动画需要页面有根标签,这因为组件中包裹的不是一个单节点元素。 transition 中不能有多个root元素

解决: **把我们的组件都包裹成单节点**

当然,那样我们过渡动画只能往一个方向,

如果想要两方向过渡,

我们使用watch监控一个自定义的route.meta属性,

这里我使用index,给每个动态路由meta下加一个index索引,当然要安装顺序有大小

并且RouteMeta的数据类型定义为number

然后我们使用watch监控,

很多同学会说为何不使用unbeforeRouteupdate,

vue-router监控路由的api只能监控同级路由,

在我这个后台管理系统不会触发

```
watch(() => route.meta.index, (to, from) => {


  if (to && from) {
    transitionName.value = to < from ? "slide-right" : "slide-left";
    console.log(transitionName.value)
  }




})
```

19-2 动态缓存keep-alive

vue3使用keep-alive用于动态缓存

直接包裹即可

```
<div>
   <router-view v-slot="{ Component }">
     <template v-if="Component">
       <transition :name="transitionName" mode="out-in" appear>
         <keep-alive>
             <component :is="Component" />
         </keep-alive>

       </transition>
     </template>
   </router-view>
 </div>
```

keep-alive属性“include，exclude”的使用

使用include，exclude 属性需要给所有vue类的name赋值，否则 include，exclude将不生效

include 值为字符串或者正则表达式匹配的组件name不会被销毁

exclude 值为字符串或正则表达式匹配的组件name会被销毁

包裹动态组件时，会缓存不活动的组件实例，而不是销毁它们。和 相似， 是一个抽象组件：它自身不会渲染一个 DOM 元素，也不会出现在组件的父组件链中。

当组件在 内被切换时，它的setup 和 onmounted 生命周期钩子不会被调用，取而代之的是 activated 和 deactivated。(这会运用在 的直接子节点及其所有子孙节点。)

```
<!-- regex (使用 `v-bind`) -->
<keep-alive :include="/a|b/">
  <component :is="view"></component>
</keep-alive>

<!-- Array (使用 `v-bind`) -->
<keep-alive :include="['a', 'b']">
  <component :is="view"></component>
</keep-alive>
```

使用keep-alive简单,

如果需要加include,

需要在vue组件中name赋值

因为setup的原因,

所以需要重新包裹script

对name赋值

```
<!-- 定义name,让keep-alive的include检索 -->
<script lang='ts'>
export default {
  name: "account",
}
</script>
```

然后在appMain中

v-bind绑定includeList

```
    <transition :name="transitionName" mode="out-in" appear>
          <keep-alive :include="includeList">
              <component :is="Component" :key="route.fullPath" />
          </keep-alive>

        </transition>
```

然后声明

```
const includeList = ref<string[]>([])
```

监听路由的变化,让route.meta中keep-alive为true

注意:routeMeta需要添加数据类型;

```
keep-alive:boolean
```

最后我们需要使用watch的深度监听,

```
// 监听meta中keepAlive为true的页面,如果为true,则动态缓存
    watch(() => route,(newVal:any,oldVal)=>{

        if(newVal.name && newVal.meta.keepAlive &&
includeList.value.indexOf(newVal.name) === -1){
          includeList.value.push(newVal.name);
          sessionStorage.setItem('KEEP_ALIVE',
JSON.stringify(includeList.value))
        }

      },{deep:true}) // 开启深度监听
```

19-3 nprogress进度条

安装

```
npm install --save nprogress
npm i --save-dev @types/nprogress
```

使用

在router的index中

进度条开始

```
router.beforeEach((to,from,next)=>{
    NProgress.start()
  const token = localStorage.getItem('token')
  if(!store.state.authStore.token && !token) {
      if(to.path.startsWith('/login'))
      next()
      else {
        next('/login')
      }
  } else if(!store.state.authStore.token && token) {
    loginByToken(token).then(res=>{
      if(res.data.status) {

       store.commit('authStore/addUserInfo',res.data)
       store.dispatch('menuStore/generateSystemMenus',res.data.permissions)
       store.dispatch('buttonStore/generateButtons',res.data.permissions)
       if(to.matched.length == 0) {

          router.push(to.path)
       }
        next()
      } else{
        next('/login')
      }

    })

  } else {
    next()
  }
})
```

进度条结束

```
router.afterEach(() => {
  NProgress.done();
});
```

nprogress配置和修改颜色

在route的index中,直接可配置progress

```
NProgress.configure({
   // 动画方式
   easing: "ease",
   // 递增进度条的速度
   speed: 500,
   // 是否显示加载ico
   showSpinner: false,
   // 自动递增间隔
   trickleSpeed: 200,
   // 初始化时的最小百分比
   minimum: 0.3
 }); //progress配置
```

如果要修改progress颜色,需要在app.vue的style中添加样式

```
#nprogress .bar { background: red !important;}
```
