有生云

通用方法文档

产品版本：V1.0

文档版本：202401

北京有生博大软件股份有限公司

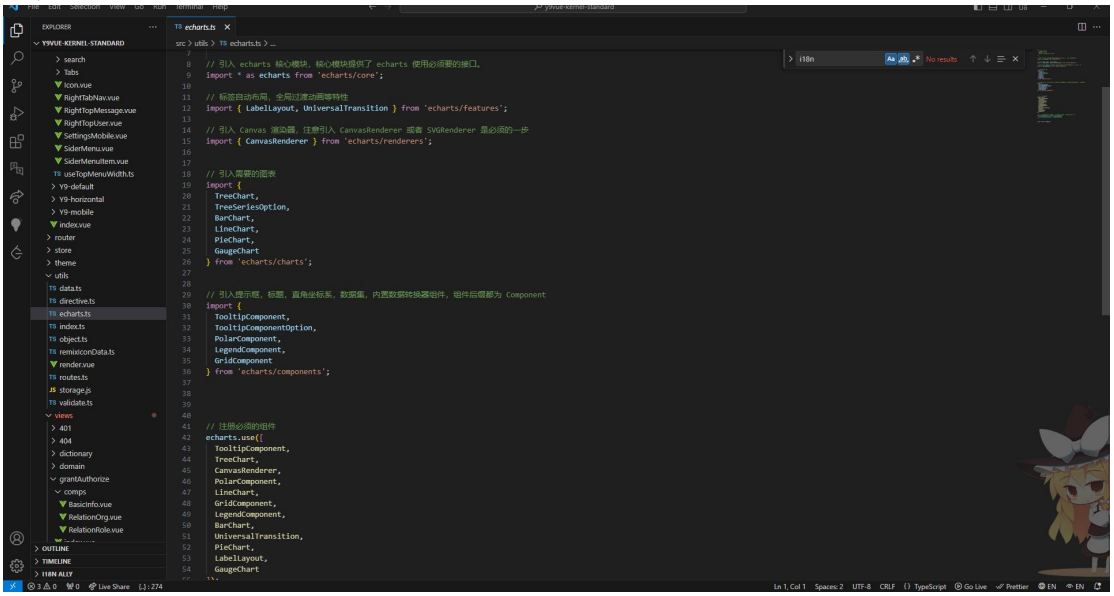
目录

在项目文件中，你可以看到很多地方都是用了 utils 目录下的工具文件。**src/utils** 目录下是封装的一些方法和一些数据整合。

1. echart 图

文件位置：**src/utils/echarts.ts**

如果项目中需要使用到 echarts 图，未防止在每个文件中重复引入，遂在 src/utils 目录下创建了 echarts.ts 文件，用于引入 echars 所需的图表、渲染器等，也有便于统一管理。

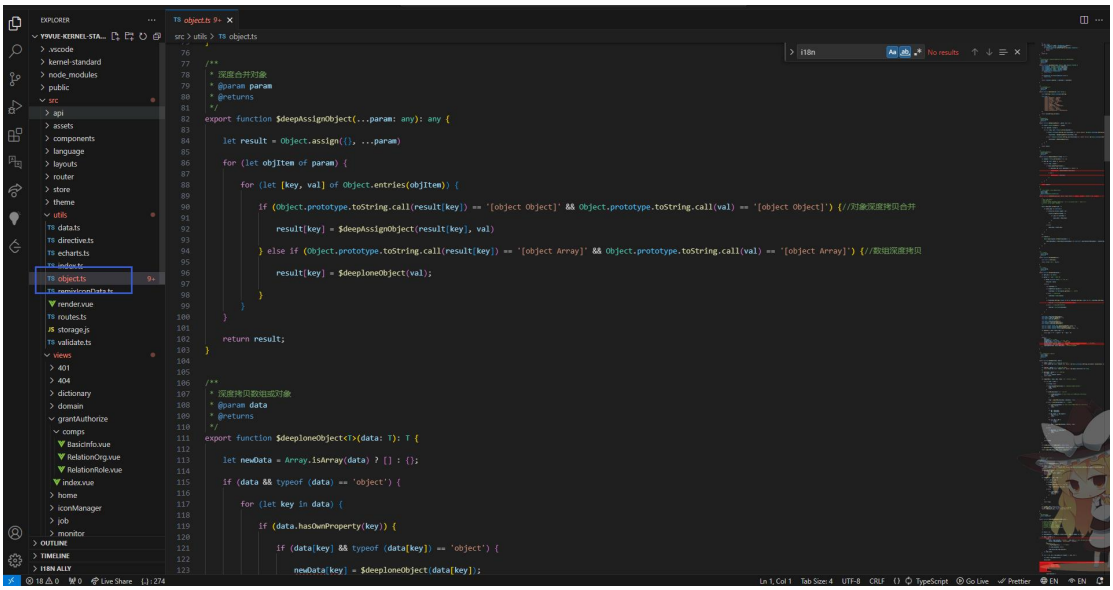


2. 方法封装

2.1. 对象、数组

文件位置：**src/utils/object.ts**

文件内有很多封装的一些方法，比如比较两个对象是否相等、深度合并对象、深度拷贝数组或对象等。

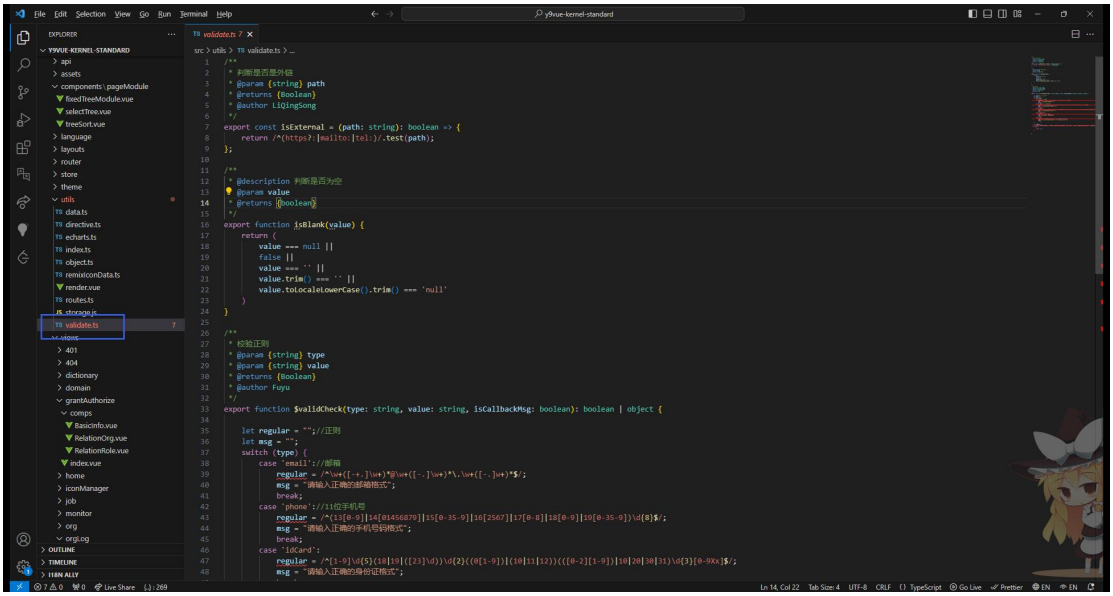


```ts
/**
 * 深度合并对象
 * @param param
 * @returns
 */
export function $deepAssignObject(...param: any): any {

    let result = Object.assign({}, ...param)

    for (let objItem of param) {

        for (let [key, val] of Object.entries(objItem)) {

            if (Object.prototype.toString.call(result[key]) == '[object Object]' && Object.prototype.toString.call(val) == '[object Object]') {//对象深度拷贝合并

                result[key] = $deepAssignObject(result[key], val)

            } else if (Object.prototype.toString.call(result[key]) == '[object Array]' && Object.prototype.toString.call(val) == '[object Array]') {//数组深度拷贝

                result[key] = $deeploneObject(val);

            }
        }
    }

    return result;
}


/**
 * 深度拷贝数组或对象
 * @param data
 * @returns
 */
export function $deeploneObject<T>(data: T): T {

    let newData = Array.isArray(data) ? [] : {};

    if (data && typeof (data) == 'object') {

        for (let key in data) {

            if (data.hasOwnProperty(key)) {

                if (data[key] && typeof (data[key]) == 'object') {

                    newData[key] = $deeploneObject(data[key]);
```

2.2. 检验正则

文件位置：**src/utils/validate.ts**

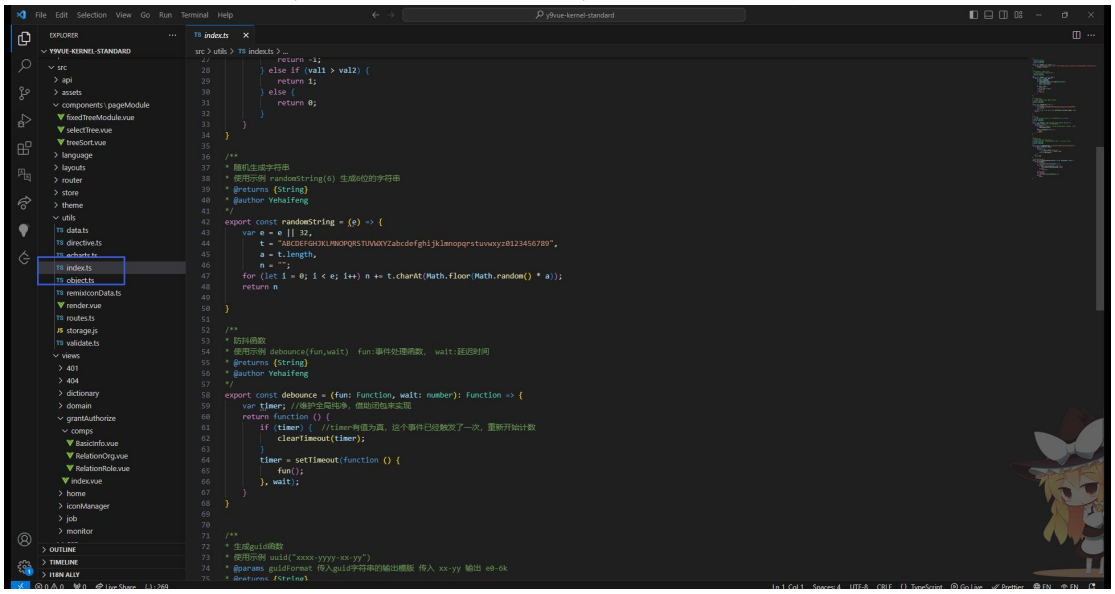
对全文中涉及到的正则进行一个整合，以便统一管理。



```ts
/**
 * 判断是否是外链
 * @param {string} path
 * @returns {Boolean}
 * @author LiQingSong
 */
export const isExternal = (path: string): boolean => {
    return /^(https?:|mailto:|tel:)/.test(path);
};

/**
 * @description 判断是否为空
 * @param value
 * @returns {Boolean}
 */
export function isBlank(value) {
    return (
        value === null ||
        false ||
        value === '' ||
        value.trim() === '' ||
        value.toLocaleLowerCase().trim() === 'null'
    )
}

/**
 * 检验正则
 * @param {string} type
 * @param {string} value
 * @returns {Boolean}
 * @author Fuyu
 */
export function $validCheck(type: string, value: string, isCallbackMsg: boolean): boolean | object {

    let regular = "";//正则
    let msg = "";
    switch (type) {
        case 'email'://邮箱
            regular = /^\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*$/;
            msg = "请输入正确的邮箱格式";
            break;
        case 'phone'://11位手机号
            regular = /^(13[0-9]|14[01456879]|15[0-35-9]|16[2567]|17[0-8]|18[0-9]|19[0-35-9])\d{8}$/;
            msg = "请输入正确的手机号码格式";
            break;
        case 'idCard':
            regular = /^[1-9]\d{5}(18|19|([23]\d))\d{2}((0[1-9])|(10|11|12))(([0-2][1-9])|10|20|30|31)\d{3}[0-9Xx]$/;
            msg = "请输入正确的身份证格式";
```

2.3. 字体大小设置、防抖等

文件位置：**src/utils/index.ts**
封装了防抖函数，字体大中小的字号，随机生成字符串等。



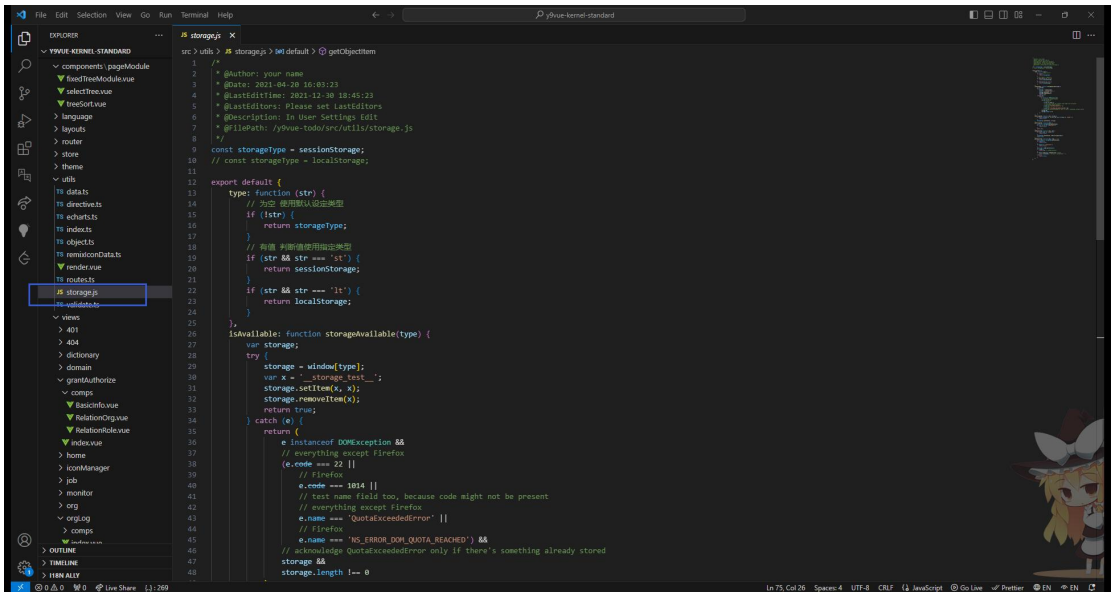
```ts
            return -1;
        } else if (val1 > val2) {
            return 1;
        } else {
            return 0;
        }
    }
}

/**
* 随机生成字符串
* 使用示例 randomString(6) 生成6位的字符串
* @returns {String}
* @author Yehaifeng
*/
export const randomString = (e) => {
    var e = e || 32,
        t = "ABCDEFGHJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789",
        a = t.length,
        n = "";
    for (let i = 0; i < e; i++) n += t.charAt(Math.floor(Math.random() * a));
    return n
}

/**
* 防抖函数
* 使用示例 debounce(fun,wait)  fun:事件处理函数,  wait:延迟时间
* @returns {String}
* @author Yehaifeng
*/
export const debounce = (fun: Function, wait: number): Function => {
    var timer; //维护全局纯净，借助闭包来实现
    return function () {
        if (timer) {  //timer有值为真，这个事件已经触发了一次，重新开始计数
            clearTimeout(timer);
        }
        timer = setTimeout(function () {
            fun();
        }, wait);
    }
}


/**
* 生成guid函数
* 使用示例 uuid("xxxx-yyyy-xx-yy")
* @params guidFormat 传入guid字符串的输出模版 传入 xx-yy 输出 e0-6k
* @returns {String}
```

3. 缓存配置

文件位置：**src/utils/storage.js**

本项目默认采用 sessionStorage 的存储方式，如果更改，可以直接在 storage 文件中将 storageType 改为 localStorage，这样项目中有引入该文件进行的存储操作都会变成 storageType 设置的类型进行数据存储。

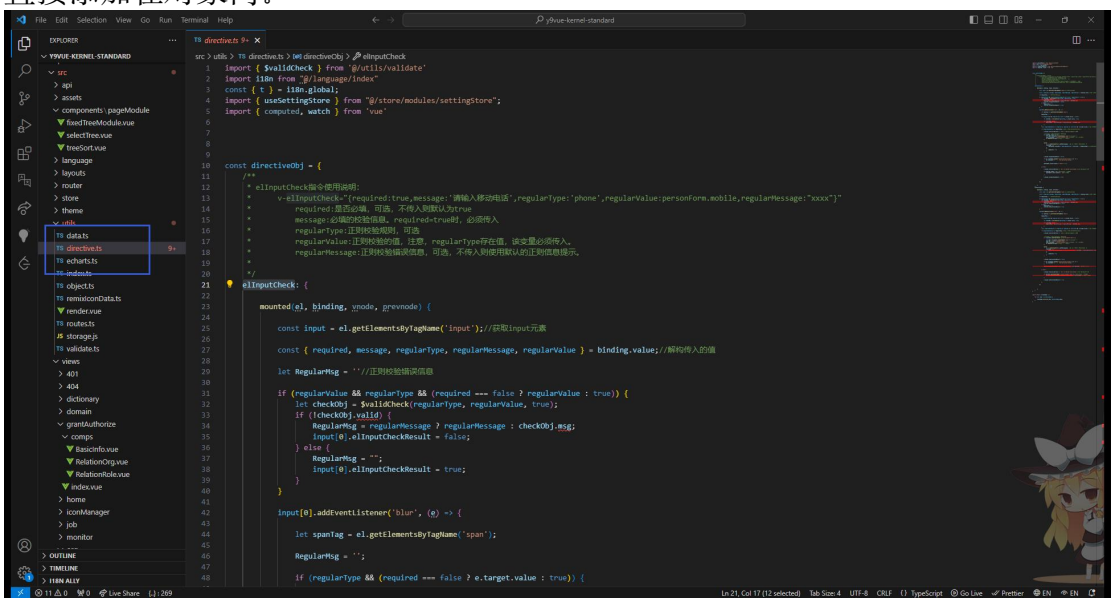
如果想单个更改，可以单独使用 sessionStorage 或 localStorage 去进行操作。



4. 自定义 v-指令

文件位置：**src/utils/directive.ts**

自定义表单验证，与表单错误处理的 v-指令。后续需要添加新的指令可以直接添加在对象内。



```ts
import { $validCheck } from '@/utils/validate'
import i18n from '@/language/index'
const { t } = i18n.global;
import { useSettingStore } from "@/store/modules/settingStore";
import { computed, watch } from 'vue'



const directiveObj = {
    /**
     * elInputCheck指令使用说明:
     *      v-elInputCheck="{required:true,message:'请输入移动电话',regularType:'phone',regularValue:personForm.mobile,regularMessage:'xxxx'}"
     *          required:是否必填，可选，不传入则默认为true
     *          message:必填的校验信息，required=true时，必须传入
     *          regularType:正则校验规则，可选
     *          regularValue:正则校验的值，注意，regularType存在值，该变量必须传入。
     *          regularMessage:正则校验错误信息，可选，不传入则使用默认的正则信息提示。
     *
     */
    elInputCheck: {

        mounted(el, binding, vnode, prevnode) {

            const input = el.getElementsByTagName('input');//获取input元素

            const { required, message, regularType, regularMessage, regularValue } = binding.value;//解构传入的值

            let RegularMsg = '';//正则校验错误信息

            if (regularValue && regularType && (required === false ? regularValue : true)) {
                let checkObj = $validCheck(regularType, regularValue, true);
                if (!checkObj.valid) {
                    RegularMsg = regularMessage ? regularMessage : checkObj.msg;
                    input[0].elInputCheckResult = false;
                } else {
                    RegularMsg = "";
                    input[0].elInputCheckResult = true;
                }
            }

            input[0].addEventListener('blur', (e) => {

                let spanTag = el.getElementsByTagName('span');

                RegularMsg = '';

                if (regularType && (required === false ? e.target.value : true)) {
```

5. 路由数据的处理

文件位置：**src/utils/routes.ts**

文件中包含面包屑类型、路由类型字段，以及一些路由相关的逻辑处理等。