1.JSON.stringify(值) 将js值转换为JSON字符串

2.console.error("报错了",err) 方法用于输出错误信息到控制台

vue中:

axios:1.api.get(url).then(res=>{if(res.data.code===200&&res.data.message=="success"){...}}).catch(err=>{...})

           2.this.api.get()

依赖

拉取了远程分支代码后，npm i 安装依赖

修改组件样式

组件类名 {

/deep/ 父类名 {

子类名{......}

  }

}

不同的组件之间传值

1.在evenrBus中定义事件名

2.传值组件中 this.$fire(this.events.事件名，值)

3.接收组件中 this.$bind(this.events.事件名，值)

可以接收多个传值组件传递过来的值

数组深拷贝 var sz=[...sz2]

使用slice concat [...数组或对象] 进行深拷贝，不能深拷贝出二维、多维数组 (假深拷贝)

最好: JSON.parse( JSON.stringify(数组或对象) )

alpha   beta  gamma

添加事件

window.addEventListener("事件名"，this.callBack, false) //false处，是否捕捉

callBack(e){...}

嵌套页面只读

标签中operate = show

判断对象是否含有某属性

对象.hasOwnProperty("属性名")

媒体查询，同时满足两个条件

@media screen and(min-width:.....) and (min-height:...){  body{....}  }

JSON JS互转

JSON.stringify(...)

JSON.parse(...)

Promise:  写promise一定要写resolve

fun1(){

   return new Promise( (resolve,reject)=>{

     setTimeout( ()=>{

      console.log("1");

       resolve("a")

     } ,500 )

 } )

}

fun2(){

return new Promise( (resolve,reject)=>{

setTimeout( ()=>{

console.log("2");

resolve("b")

} ,500 )

} )

}

this.fun1()

.then(  (data)=>{

  console.log(data)  //a

  return fun2( );

} )

.catch(...)

.then( (data)=>{

console.log(data) //b

return fun3( ).....

})

.catch(...)

Promise.all([this.fun1(), this.fun2()])

.then(   (data)=>{

  console.log(data)  //["a","b"]

})

.catch(  (data)=>{

  console.log(data) //捕捉reject返回的错误

   //只能catch到 reject过来的第一个错误且then得  不到resolve过来的数据

})

Promise.allSettled([this.fun1(),...]).then(data=>...) //[{status"fulfilled",value:"resolve数据"},

    {status"rejected",reason:"reject数据"}]

无论成功与否，都可以then到数据结果

new Set去重

[...new Set(数组)]

字符串中查找

str.indexOf("a")  //找不到返回-1

数组中查找

arr.includes("2")//true false

arr.find((elem,i,arr)=>{return 条件) 满足返回元素 不满足返回undefined

arr.findIndex((elem,i,arr)=>{return 条件) 满足返回元素下标 不满足返回-1

查询字符串中是否包含

str.includes("world") //true false

对象新增方法:

Object.keys(obj)//得到一个数组包含所有下标

Object.values(obj)//得到一个数组包含所有的值

Objecr.entries(obj)//得到大数组包含一个个属性对应值的小数组

查看当前url地址

location.href

截取到端口(包含端口)

location.origin

input对输入的值做校验

onKeyUp="value=value.replace(/[\w]/g,"")"

判断数据类型

Object.prototype.toString.call(  )

过滤器放在methods同层级

filters:{

   filtersA(){

   }

}

子向父传值

子组件: @click="$emit("fuFun",值)"

父组件: @fuFun="fun($event)"   fun(val){...}

转字符串

String(值) 可以检查到null undefined

父子组件访问方法和值

父访问子:this.$refs.refs值.属性或者方法

子访问父:this.$parent.属性或者方法

兄弟组件传值

新建bus.js文件 兄弟组件引入

内容为 import Vue from "vue"

              export default new Vue

发送: bus.$emit.("随便事件名",值)

接收: bus.$on("事件名",data=>{

           对值做处理

            })

es6导入导出语法

导出export var name = "yang"

或 export const a = 1

或 export default {...}

导入import ... from "..."

箭头函数的处理结果需要return出去，除非只有一句

function(a=1,b=2) ab没有传值时，则默认值

延展操作符:

var a = [1,2,3]

var b = {name:"yang",j:20}

var c = [...a] 相当于a打散赋值到c

var d = {...b}相当于b打散赋值到d

指数运算符\*\*

2\*\*2 //4

如果字符串长度不够，则在前或后填充

"a".padStart(4,"\*") a的长度小于4 在前面填充三个\*

"a".padEnd(4,"\*") a的长度小于4 在后面填充三个\*

数组降维，数组去空项

var a = [1,2,3,[4,5,6]]

a.flat(值) 值的大小表示降多少,Infinity表示降到1维

此方法可以去除空项

路由守卫(模拟用户验证)

to-进去哪个路由  from-从哪个路由离开 next-是否进入，或进入哪里

全局守卫: 在const router = new VueRouter..后面

router.beforeEach((to,from,next)={

  let isLogin = window.sessionStorage.getItem("userInfo");

if(isLogin){

  //如果用户信息存在则向下执行

  next()

}else{

  //如果用户token不存在则跳转到login页面

  to.path != "/login" ? next({path:"/login"}) : next()

}

})

组件守卫:

写在组件data同层级处，用法一样\*只在第一组件级有效

到达此组件时:beforeRouteEnter()

离开此组件时:beforeRouteLeave()

to.path去往的路由地址

from.path从哪个路由地址来/从哪里离开

路由独享守卫:

写在index.js文件,每个分配的路由,component后面

beforeEnter:(to,from,next)=>{}

路由懒加载两种方式:

vue异步组件方式:index.js文件中

①不写组件来源(import...from...)

②component: resolve=>(require(["相对路径"],resolve))

ES语法: (常用方法)

①将来源改为:const home2 = ()=>import("相对路径")

②component不变

组件缓存 keep-aluve

<keep-alive  include="a">

 <component>

  这里的组件将被缓存

 </component>

</keep-alive>

include只有匹配的组件被缓存(字符串或者正则)

exclude任何匹配的组件都不会缓存

在index.js分配路由里，某个路由下:

meta:{keepAlive:true} 则表示缓存

监听窗口变化

data..

pcWidth: document.documentElement.clientWidth

mounted..

const that = this;

window.onresize = ()=> {

  return ( ()=>{

     that.pcWidth =              document.documentElement.clientWidth;

      console.log(that.pcWidth)

   } )()

}

watch..

pcWidth(val){

  this.pcWidth = val

}

监听窗口变化2:

mounted中:

jquery写法

$(".class名").resize((e,width,height)=>{

      .......

})

vue中cookie的使用

安装:

npm install vue-cookie

main.js中引用:

import VueCookie from "vue-cookie"

Vue.use(VueCookie)

组件中设置一个cookie:

设置距离当前的过期时间，如一秒钟

var Time1= new Date().getTime()

var Time2 = new Date(Time1 + 1000)

this.$cookie.set("test",values,{expires:Time2})

获取一个cookie:

this.$cookie.get("test")

删除一个cookie:

this.$cookie.delete("test")

去掉字符串两侧空格

去除前面空格 str.trimStart()

去除后面空格 str.trimEnd()

在循环中执行异步操作，通过异步返回的结果来判断循环是否继续执行 async 和 await

await执行完异步操作后，代码才会顺序执行

data:...

randomNum:5

mounted:...

var thiss = this;

function fun(times){

 return new Promise(resolve => {

   setTimeout(()=>{

     thiss.randomNum = Math.floor(Math.random()\* (10-0+1)+0);

     console.log(thiss.randomNum,"随机值")

   },times)

})

}

async function funAsync(){

 for(let i=1;i<thiss.randomNum;i++){

     await fun(1000);

     console.log(i+1)

 }

}

funAsync();

鼠标移入移出事件

@mouseenter和@mouseleave

判断一个值是不是NaN

isNaN(NaN)  //true

通过下标得到字符串对应的值:str[0]

a.indexOf(元素)都可以用于数组，字符串寻找元素对应的下标

将数组排序

arr.sort((a,b)=>a-b)

中文排序

var arr=["鸡","狗","猫"]

//a.localeCompare(b) a,b 谁小 返回谁

arr = arr.sort((a,b)=>{

                return a.localeCompare(b)

          })

arr.splice(2,0,5,6) 从下标2开始往后删除0位，并在此添加入5和6,返回删除的元素，原数组改变

将一个一维数组和一个二维数组concat连接起来，返回一个连接的一维新数组

删除对象的一个属性和值: delete obj.a或

delete obj["a"]

判断对象是否含有某属性: "name" in obj //true false

如果不是自身的而是继承的也会true

判断对象自身的是否含有某属性:

obj.hasOwnProperty("name")

Map、Set都是ES6新增的数据类型

new Map:

var aa = new Map([ ["a",1],["b",2] ])

获取: aa.get("a")  //1

添加: aa.set("c",3)

是否含有: aa.has("d")

删除: aa.delete("a")

Object.protorype.toString.call(aa)//[Object Map]

用for(var [key,item] of aa)遍历Map对象

Set对象中的元素都是唯一的

var bb = new Set([1,2,3,4])

添加: bb.add(1) //一次添加一个

是否含有: bb.has(1)

判断长度: bb.size

删除: bb.delete(1)

数组去重: [ ...new Set(数组a) ]

求集合

var a = new Set([1,2,3])

var b = new Set([2,3,4])

求并集: var c = new Set([...a, ...b]) //{1,2,3,4}

求交集:

var d = new Set( [...a].filter(x=>b.has(x)) )//2 3

求差集:

var e = new Set( [...a].filter(x=>!b.has(x)) )//1

Set对象可以转为数组

var f = [...a] //将set对象转为数组

解构赋值

交换a,b的值

var a = 1 , b = 2;

[ a , b ] = [ b , a ]

将数组中的元素转为字符串:

var a=[1,2,3]

var b = a.map(String)

策略模式/状态模式 减少if else

function fun(type,a,b){

     var obj={

         add(a,b){return ...},

         minus(a,b){return ...}

     };

     return obj[type](a,b)

}

v-if实现页面刷新 或者用key

v-if="num"

data中 num = 0

js中num++

created中页面不会渲染但是在created里写this.$nextTick()可以等待dom生成以后来获取dom对象

directive自定义指令

局部指令 data同一级(全局vue.directives)

directives:{

    test:{

      inserted: (el,binding)=>{...},

      bind: (el,binding)=>{...}

    }

}

更新数据同时更新到视图

this.$set(target, key, value)

target 是要更改的数据源

key 要更改的具体数据 （对象属性或数组下标)

value 重新赋的值

finally无论有无异常都会执行

在请求结果中与then、catch同等级

.finally(()=>{....})

watch监听属性

监听的值需要在data中定义

data:{

  testVal:" "

}

watch: {

   testVal: {

     handler(newVal, oldVal){....},

     或者 handler: 'fun', //调用方法, 需要加引号

     immediate: true   //数据初始化就监听到

    }

}

深度监听,上述不能监听到对象内部的属性值改变

data:{

  testObj: {

    name: "yang"

  }

}

watch: {

   testObj: {  // 监听整个对象

     handler(newVal, oldVal){....},

     或者 handler: 'fun', //调用方法, 需要加引号

     deep: true,

     immediate: true //数据初始化就监听到

    }

或者:

"testObj.name": { //监听对象的某个属性

     handler(newVal, oldVal){....},

     或者 handler: 'fun', //调用方法, 需要加引号

     deep: true,

     immediate: true   //数据初始化就监听到

   }

}

函数防抖，鼠标移动事件，滚动条事件，窗口大小改变事件

function debounce(fn, wait) {

  var timeout = null

  return function( ){

     if(timeout !== null) clearTimeout(timeout);

     timeout = setTimeout(fn, wait)

   }

}

function handle(){

  .....

}

滚动事件:

window.addEventListener("scroll", debounce(handle, 1000))

函数节流，假设用户一直触发一个函数(间隔短)，函数节流就是让他每隔一段时间才触发

function throttle(func, delay) {

  var prev = Date.now();

  return function(){

    var \_this = this; //this指向window

    var args = arguments;

    var now = Date.now();

    if(now - prev >= delay) {

       func.apply(\_this, args);

       prev = Date.now();

    }

  }

}

function handle(){

  .....

}

滚动事件:

window.addEventListener("scroll", throttle(handle, 1000))

v-model.lazy  失去焦点或者回车才会更新数据

v-model.trim  过滤左右两边的空格

重定向路由:

const routes = [

   {

      path: "/",

      redirect: "/home"

   }

]