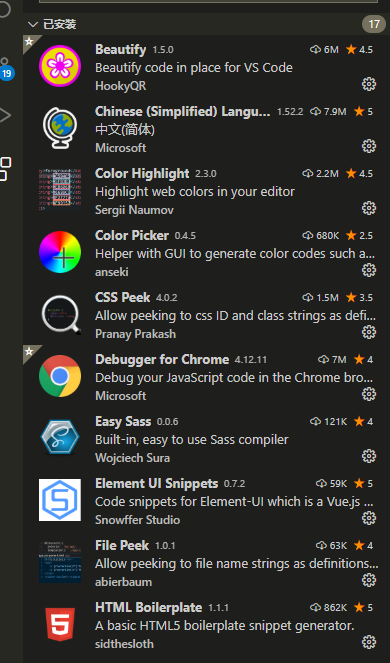
**Vscode所用的插件**



1. **父传子，子组件获取数据的不同方式**

**1.props:[“op”]**

**2..props:{**

**Op:{**

**Type:‘string’(数据类型)**

**}**

**}**

**使用时候，直接在钩子函数中，this.op就可以获得**

**二、**

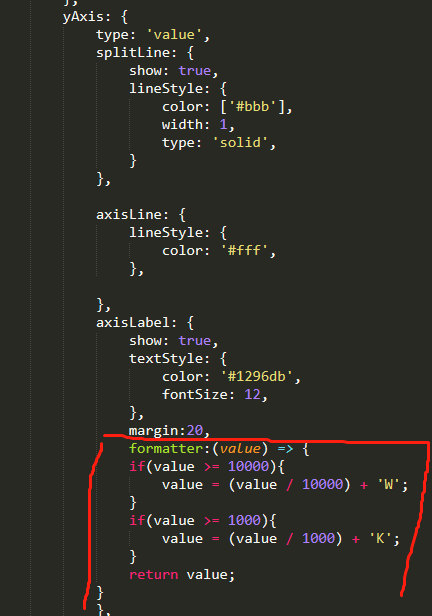
**localStorage.getItem(key):获取指定key本地存储的值**

**localStorage.setItem(key,value)：将value存储到key字段**

**localStorage.removeItem(key):删除指定key本地存储的值**

**localStorage与sessionStorage的唯一一点区别就是localStorage属于永久性存储，而sessionStorage属于当会话结束的时候，sessionStorage中的键值对会被清空**

1. **echart 参数**
2. **y轴数字过长导致没显示全：解决方法：**



**formatter:(value) => {**

**if(value >= 10000){**

**value = (value / 10000) + 'W';**

**}**

**if(value >= 1000){**

**value = (value / 1000) + 'K';**

**}**

**return value;**

**}**

1. **X轴数据过多影响显示效果：**
   * 1. dataZoom: {
2. realtime: true, //拖动滚动条时是否动态的更新图表数据
3. height: 20, //滚动条高度
4. start: 100, //滚动条开始位置（共100等份）
5. end: 98 //结束位置（共100等份）
6. }

**添加滑动条**

**3. y轴左右偏移，添加margin：20**



**4.在x轴上画上基线 在**series下添加markLine 在data中要与x轴的某个值对应上

 markLine: {

                        symbol: ['none', 'none'], //去掉箭头

                        itemStyle: {

                            normal: {

                                lineStyle: {

                                    type: 'solid',

                                    color: '#fff'

                                },

                                label: {

                                    show: true,

                                    position: 'middle'

                                }

                            }

                        },

                        data: [

                            {

                                xAxis: '' //这里设置false是隐藏不了的，可以设置为-1

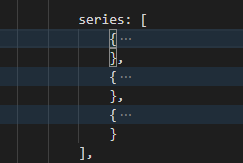
                            }

                        ]

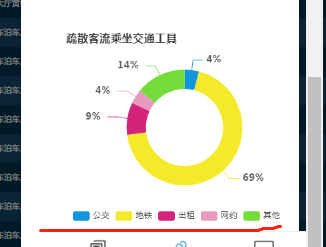
                    }

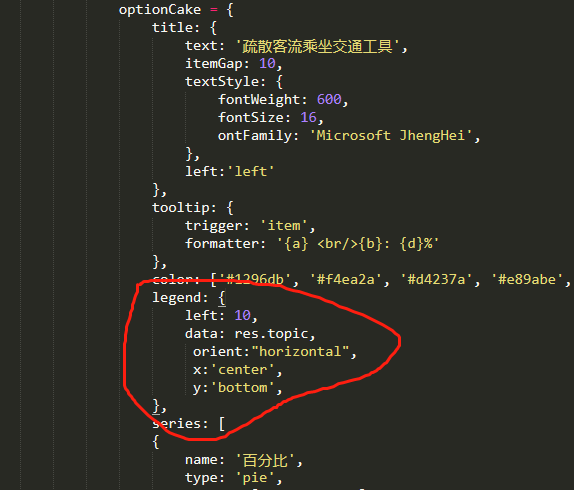
**5.添加多个折线图**

**在series中多些几个对象**



**6.饼状图 控制它的位置**





**legend: {**

**left: 10,**

**data: res.topic,**

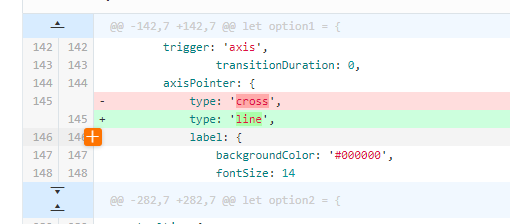
**orient:"horizontal",**

**x:'center',**

**y:'bottom',**

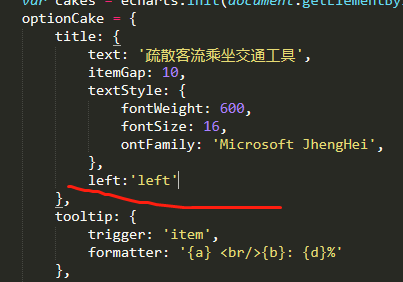
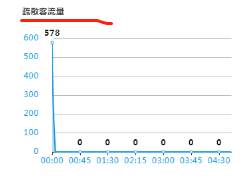
**},**

1. **十字标线去掉横坐标的**



type: 'cross', 为十字标线 type: 'line', 竖向标线

1. **标题位置 left取值 left center right**

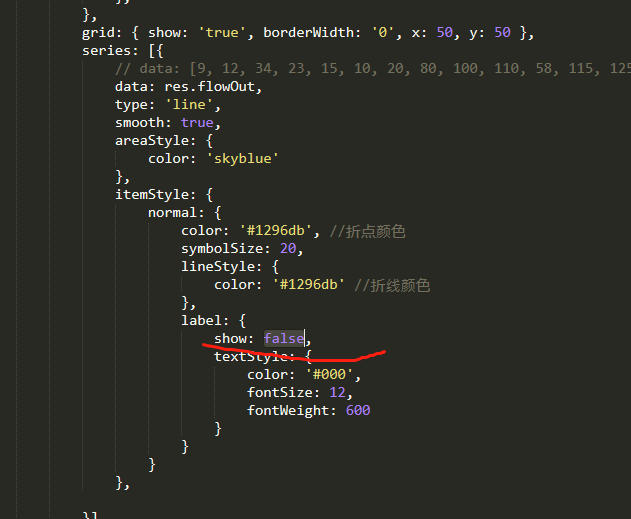


**10，改变饼状图的块儿颜色**



**color: ['#1296db', '#f4ea2a', '#d4237a', '#e89abe','#74dc38'],**

**11.折线图图中原点上的数字**



**12.扇形折线文字防止折叠**



**avoidLabelOverlap: true, 文字不折叠**

**minAngle: 1, 最小角度1**

**13.去掉时间获取封装**

**// 当前时间**

**function renderTime(date) {**

**let data = new Date(date).toJSON();**

**if (data == null) {**

**return null;**

**}**

**let dt = new Date(data);**

**let Y = dt.getFullYear();**

**let M = dt.getMonth() < 9 ? '0' + (dt.getMonth() + 1) : dt.getMonth() + 1**

**let D = dt.getDate();**

**let h = dt.getHours();**

**let m = dt.getMinutes() < 10 ? '0' + dt.getMinutes() : dt.getMinutes();**

**let s = dt.getSeconds() < 10 ? '0' + dt.getSeconds() : dt.getSeconds();**

**return Y + '-' + M + '-' + D + ' ' + h + ':' + m + ':' + s;**

**}**

**14.转换时分秒**

**function currents() {**

**let date = new Date();**

**let near = renderTime(date).substring(0, 10).replace(/-/g, '');**

**let timesClock = parseInt(renderTime(date).substring(11, 13))**

**console.log("timesClock", timesClock)**

**if (timesClock < 10) {**

**var timesClocks = '0' + timesClock**

**} else {**

**var timesClocks = timesClock**

**}**

**console.log("时间=============>", near, timesClocks)**

**request(near, timesClocks)**

**}**

**时间转换插件 moment.Js**

**四.普通页面之间跳转传值**

**window.location.href = 'newsListDetailss.html?id='+id 跳转页面传参数**

**其他页面取参数**

**function getUrlParam(name) {**

**var reg = new RegExp("(^|&)" + name + "=([^&]\*)(&|$)"); //构造一个含有目标参数的正则表达式对象**

**var r = window.location.search.substr(1).match(reg); //匹配目标参数**

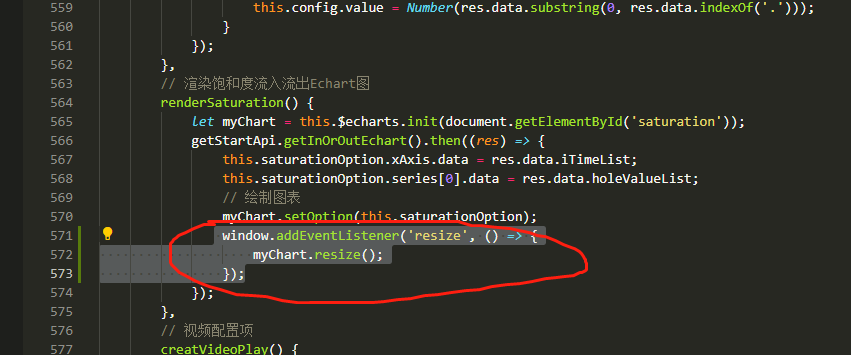
**if (r != null) return unescape(r[2]);**

**return null; //返回参数值**

**}**

**let id = getUrlParam("id")**

**15.echarts自适应**



 window.addEventListener('resize', () *=>* {

                    myChart.resize();

                });

**五、改变标签的属性值 $("#fam").attr('src',res.infoPath);**

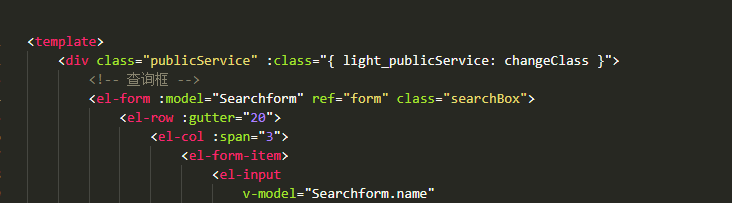
**六、**

**1.获取Vuex的store**

this.$store.state.style

**可以在周期和watch中拿到**

**2.动态绑定class**



:class="{ light\_publicService: changeClass }"



:class="[changeClass == false ? 'table1366' : 'table']"



**Class 复合语法使用**

**3.**

**Vue 延迟一段时间执行**

this.$nextTick(() *=>* {

                this.$refs.uploadTitle.clearFiles();

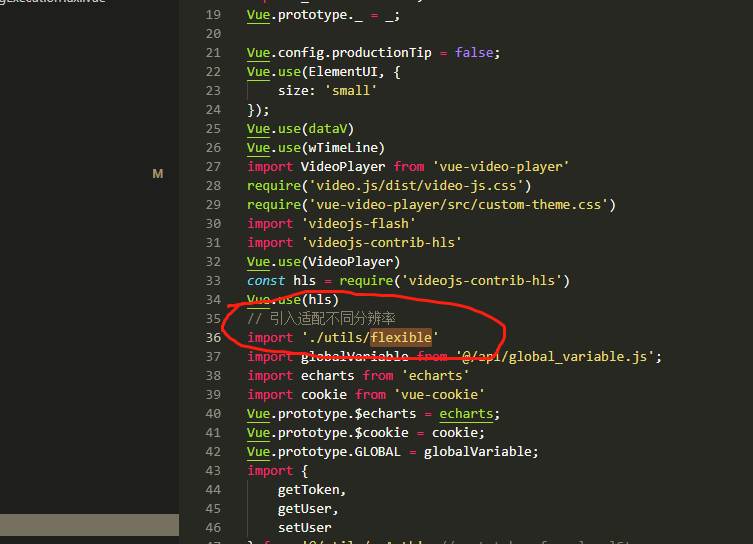
            });

**4.插槽**



**标签的class要与插槽名相同**

**5.vue根据窗口自适应**



**Main.js中引入js**

**Style第一行写上**

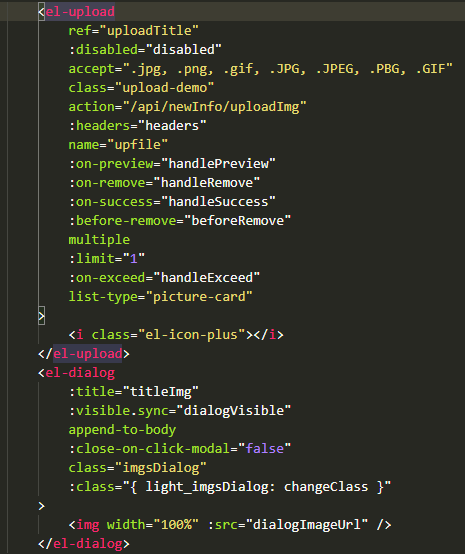
@function px2rem($px) {

    $rem: 190px;

    @return ($px/ $rem) + rem;

}

**4.图片上传**



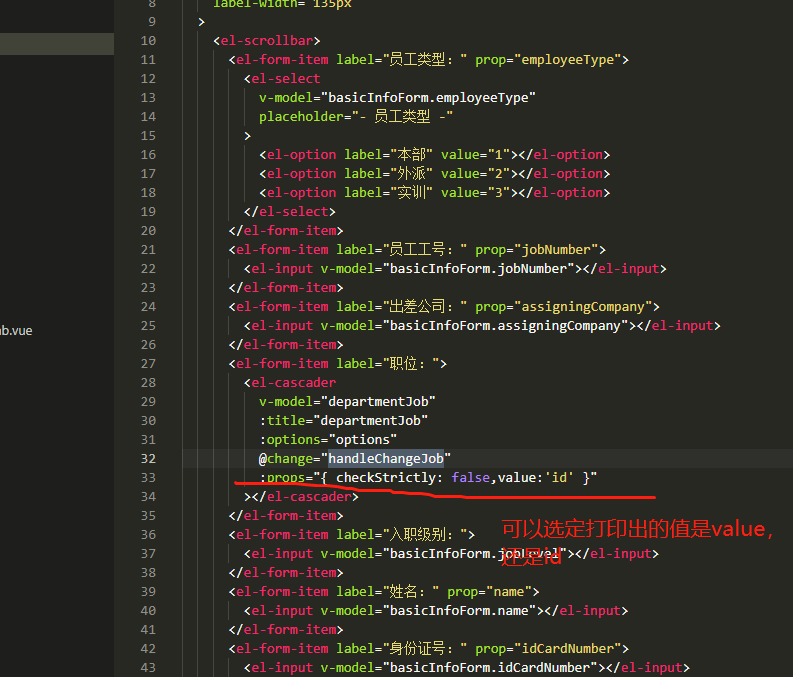
**文件**

**3.html 声明**

<meta name="referrer" content="always">

不改变http header 中的 referer 的值，注意：这种情况下，如果当前页面使用了 https 协议，而要加载的资源使用的是 http 协议，加载资源的请求头中也会携带 referer。

4.elementUI input





**获取级联框的所有值**

**6.element 的input框校验**



  qq: [{pattern:/^[1-9][0-9]{0,19}$/,message:'qq号只能是0-20位数字',trigger:'blur'}],

**7.正则校验**

   {pattern:/^[1-9][0-9]{4,14}$/,message:'qq号只能是5-15位数字',trigger:'blur'}

              ],

              ICBCCard: [

                {pattern:/^6[2]\d{17}$/,message:'卡号只能是以62为开头的19位数字',trigger:'blur'}

              ],

**8.ref作痛**

1.ref 加在普通的元素上，用this.ref.name 获取到的是dom元素

2. ref 加在子组件上，用this.ref.name 获取到的是**组件实例，可以使用组件的所有方法**。

3利用 v-for 和 ref 获取一组数组或者dom 节点

　注意：

　　1、**ref 需要在dom渲染完成后才会有**，在使用的时候确保dom已经渲染完成。比如在生命周期 **mounted(){} 钩子中调用**，或者**在 this.$nextTick(()=>{}) 中调用**。

　　2、如果ref 是循环出来的，**有多个重名，那么ref的值会是一个数组** ，此时要拿到单个的ref 只需要循环就可以了。

**9.** *event*.preventDefault()阻止元素发生默认事件

**10.vue获取当前路由**

完整url可以用 window.location.href  
路由路径可以用 this.$route.path  
路由路径参数 this.$route.params 例如：/user/:id → /user/2044011030 → this.$route.params.id  
路由查询参数 this.$route.query 例如：/user/search?name=sf → this.$route.query.name

**十一、浏览器本地存储**

**webStorang分为sessionStorang和localStorang**

**1、生命周期：localStorage:localStorage的生命周期是永久的，关闭页面或浏览器之后localStorage中的数据也不会消失。localStorage除非主动删除数据，否则数据永远不会消失。**

**sessionStorage的生命周期是在仅在当前会话下有效。sessionStorage引入了一个“浏览器窗口”的概念，sessionStorage是在同源的窗口中始终存在的数据。只要这个浏览器窗口没有关闭，即使刷新页面或者进入同源另一个页面，数据依然存在。但是sessionStorage在关闭了浏览器窗口后就会被销毁。同时独立的打开同一个窗口同一个页面，sessionStorage也是不一样的。**

**2、存储大小：localStorage和sessionStorage的存储数据大小一般都是：5MB**

**3、存储位置：localStorage和sessionStorage都保存在客户端，不与服务器进行交互通信。**

**4、存储内容类型：localStorage和sessionStorage只能存储字符串类型，对于复杂的对象可以使用ECMAScript提供的JSON对象的stringify和parse来处理**

**5、获取方式：localStorage：window.localStorage;；sessionStorage：window.sessionStorage;。**

**6、应用场景：localStoragese：常用于长期登录（+判断用户是否已登录），适合长期保存在本地的数据。sessionStorage：敏感账号一次性登录；**

**WebStorage的优点：**

**（1）存储空间更大：cookie为4KB，而WebStorage是5MB；**

**（2）节省网络流量：WebStorage不会传送到服务器，存储在本地的数据可以直接获取，也不会像cookie一样美词请求都会传送到服务器，所以减少了客户端和服务器端的交互，节省了网络流量；**

**（3）对于那种只需要在用户浏览一组页面期间保存而关闭浏览器后就可以丢弃的数据，sessionStorage会非常方便；**

**（4）快速显示：有的数据存储在WebStorage上，再加上浏览器本身的缓存。获取数据时可以从本地获取会比从服务器端获取快得多，所以速度更快；**

**（5）安全性：WebStorage不会随着HTTP header发送到服务器端，所以安全性相对于cookie来说比较高一些，不会担心截获，但是仍然存在伪造问题；**

**（6）WebStorage提供了一些方法，数据操作比cookie方便；**

**setItem (key, value) ——  保存数据，以键值对的方式储存信息。**

**getItem (key) ——  获取数据，将键值传入，即可获取到对应的value值。**

**removeItem (key) ——  删除单个数据，根据键值移除对应的信息。**

**clear () ——  删除所有的数据**

**key (index) —— 获取某个索引的key**

# ****使用方法****



**删除数据**

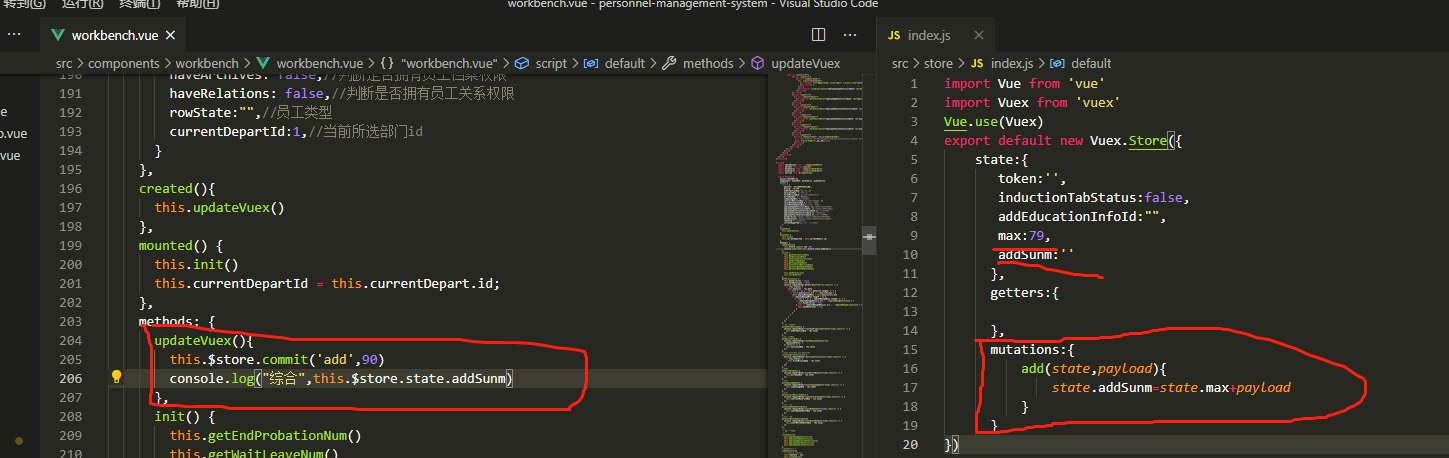
**localStorage.removeItem(key)**

**sessionStorage.removeItem(key)**

**十二、vuex mutations**

**①Mutations 传值 this.$store.commit(store,payload)**

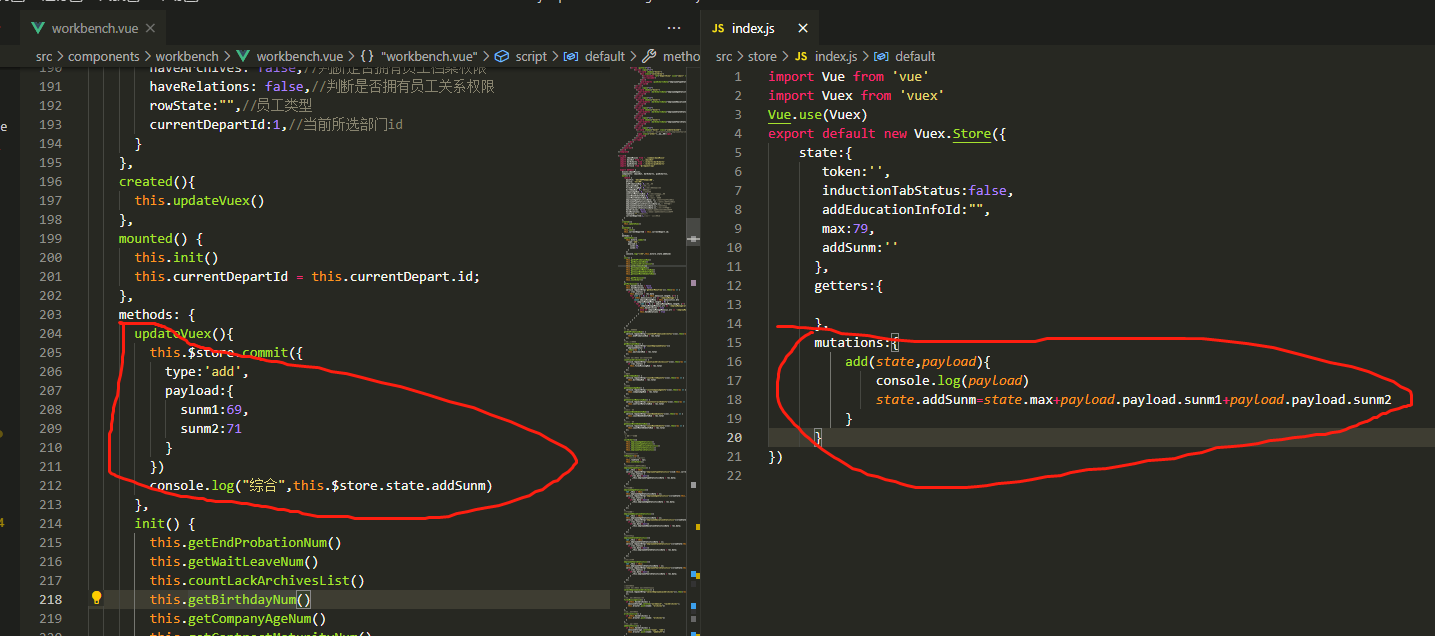
**Store为vuex中store的值，payload为调用mutations时候传的值**



**②**mutations**传输多个对象**

**mutation 必须同步执行**

**一、**



this.$store.commit({

          type:'add',

          payload:{

            sunm1:69,

            sunm2:71

          }

        })

  add(*state*,*payload*){

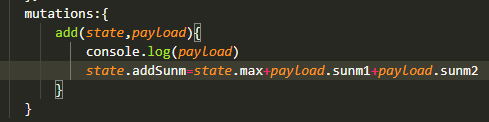
              console.log(*payload*)

*state*.addSunm=*state*.max+*payload*.payload.sunm1+*payload*.payload.sunm2

          }

**二、**





**③mutations 中增删数据**

Vue.set(state,"age",15) 添加

Vue.delete(state,'age') 删除

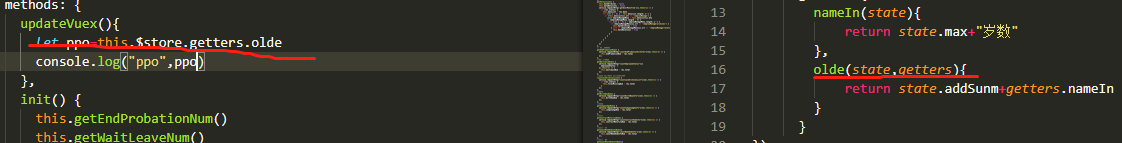
# ****Vuex getters****

**可以对state中的成员加工后传递给外界**

**Getters中的方法有两个默认参数**

**state 当前VueX对象中的状态对象**

**getters 当前getters对象，用于将getters下的其他getter拿来用**



# ****Actions 可以异步操作****

# ****Actions调用的是mutation里的方法****

Actions中的方法有两个默认参数

* context 上下文(相当于箭头函数中的this)对象
* payload 挂载参数

# 

# 

使用方法

this.$store.state.addSunm

# ****actions用法****

  caozuo(*context*,*payload*){

*context*.commit('add',*payload*)

          }

      }

**十三、axios传参数**

**①Get、delete是 params{ } post和put { }**

**原始写法：**//原始的Axios请求方式

axios({

method: 'post',

url: '/user/12345',

data: {

firstName: 'Fred',

lastName: 'Flintstone'

},

timeout: 1000,

...//其他相关配置

});

**②Vue-resouce请求**

**\*引入Vue框架\*/**

**import Vue from 'vue'**

**/\*引入资源请求插件\*/**

**import VueResource from 'vue-resource'**

**/\*使用VueResource插件\*/**

**Vue.use(VueResource)**

**语法：**引入vue-resource后，可以基于全局的Vue对象使用http，也可以基于某个Vue实例使用http

**// 基于全局Vue对象使用http**

**Get传参数**,{params : {a:1,b:2}}

**Vue.http.get('/someUrl', [options]).then(successCallback, errorCallback);**

**Vue.http.post('/someUrl', [body], [options]).then(successCallback, errorCallback);**

**// 在一个Vue实例内使用$http**

**this.$http.get('/someUrl', [options]).then(successCallback, errorCallback);**

**this.$http.post('/someUrl', [body], [options]).then(successCallback, errorCallback);**

**十四、怪异盒子模型的宽高包含了content border padding content**

**标准 的高宽只有content**

**十五、Vue路由按需加载**

component: () *=>* import('@/components/workbench/workbench')

# ****子组件向父组件传参数****

**子组件中写一个函数包裹住**

this.$emit('childFn', this.message);

**childfn是父组件的事件名 this.message是要穿的参数，**

**父组件中**

<test-com @childFn="parentFn"></test-com>

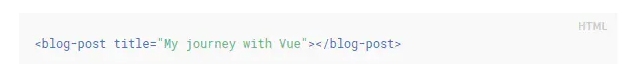
parentFn(payload) {

this.message = payload;

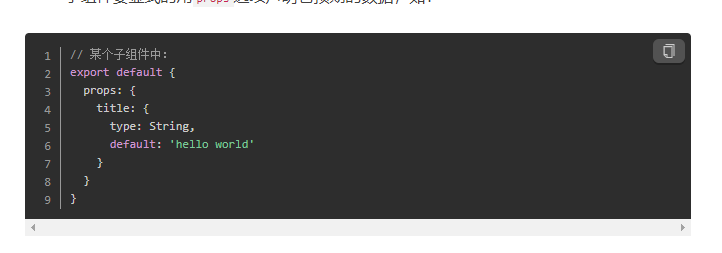
}

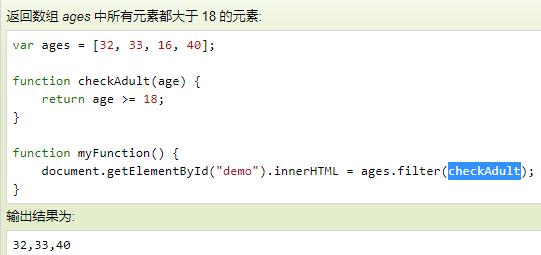
**@childFn是emit中的事件名 parentFn是父组件的函数用来处理接受的值**

**父组件箱子组件穿参数**



**在父组件中引入子组件标签名，在标签名上绑定要穿的值，在子组件中用props接受**



**十六、**

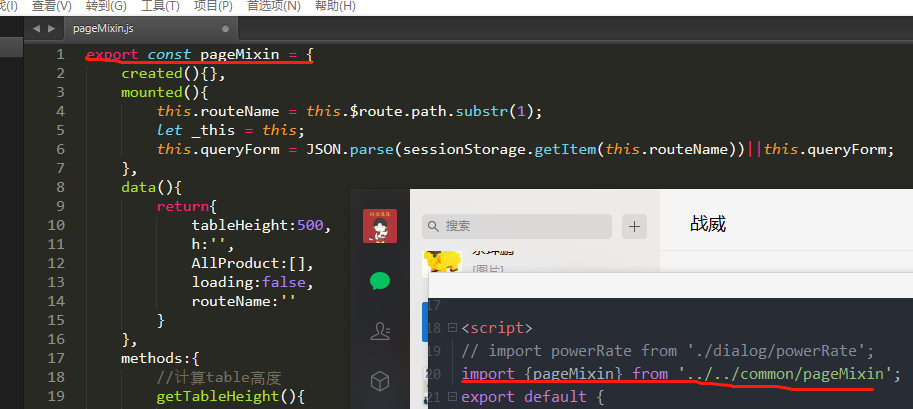
**filter过滤数组**

**reverse() 将字符串倒序排放**

**join() 方法用于把数组中的所有元素放入一个字符串**

**trim()去除前后空格**

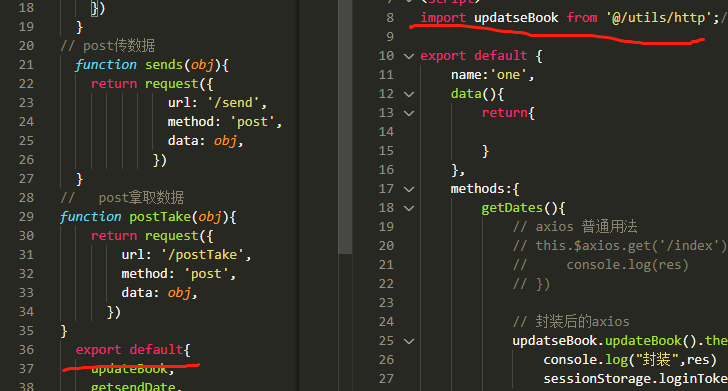
**十七、Mixin**

****

**Vue中引入其他js文件的方法，用**

**Export 暴露用**

**import { fasda} from “./components/page”引入**



**export default 暴露出用**

**import fasda from “./components/page”引入**

**mixin：**

**写在js文件中，用import引入，在组件中写上** mixins:[minixx.pageMinix],

**即可使用，若mixins的方法名和数据名与组件的冲突，那么mixins的·就会被覆盖**

**十八、自定义组件和自定义指令**

* **自定义指令：通过通过directive方法，配合钩子函数及参数定义指令**

**钩子函数**

* **inserted: 被绑定元素插入父节点时调用**
* **bind: 指令被绑定时调用**
* **unbind: 指令解绑时调用，如 $destory()**
* **update: 组件更新时调用**
* **componentUpdated: 组件更新后完成调用**

**自定义指令格式**



**局部指令：**

**directives: {**

**focus: {**

**// 指令的定义**

**inserted: function (el，binding，vnode) {**

**el.focus()**

**}**

**}**

**}**

**调用：v-focus**

**全局指令：**

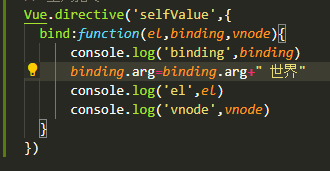
Vue.directive（‘demo’,{

Bind:function(el,binding,vnode){

}

}）

**调用：**

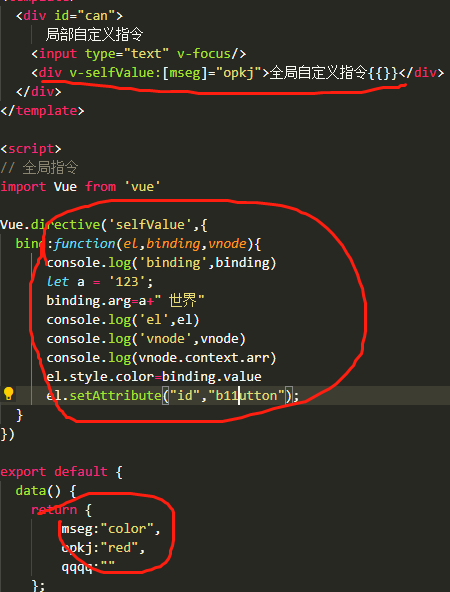


**写在Main.js中**



**SelfValue是方法名，binding中含有arg:hello value:world**

setAttribute("type","button"); 给标签设置属性名

**实例**

**十九、插槽**

1. **用法：子组件写<slot>内容</slot>，父组件引入子组件后，在子组件标签内写内容，slot标签内的不会显示，不写内容，slot内容会显示**
2. **具名插槽：**

**子组件中<slot name=”名字”></slot>**

**父组件引用 <template v-slot:名字></template>**

**父子组件的名字要保持一直**

1. **作用域插槽：**

**子组件中：<slot :data=”mse”><slot> mse=”破”**

**父组件：<div slot-scope=”nui”>{{nui.data}}</div>**

**二十：抗锯齿渲染**

*-webkit-font-smoothing*: antialiased;

*-moz-osx-font-smoothing*: grayscale;

**二十一、**

**Vue3 跨域代理代理配置**

**在项目根目录创建 vue.config.js文件，文件内写上**

*module*.*exports* = {

    devServer: {

      proxy:"http://127.0.0.1:3333",

      port: 8080

    }

  }

**proxy:**[**http://127.0.0.1:3333**](http://127.0.0.1:3333) **后台服务地址**

**port 项目端口地址**

**封装axios**

**二十二 路由传参**

**传参数**

**1.params传参数 路由中一定要写上name**

this.$router.push({

        name: "two",

         params: { userId: 123 }

      });

**接收参数**

this.$route.params.userId

**2.query传参数 一定写路径**

**传输**

**this.$router.push({ path: '/news', query: { userId: 123 }});**

**接受**

**this.$route.query.userId**

**二十三 watch侦听器**

**普通侦听：**

**Watch:{**

**watchtv（val,oldVal）{**

**写上要监听的变化**

**}**

**}**

**Watchtv：是data中的数据，val是变化之前的值，oldVal是变化后的值**

**深度监听：**

**数据层级较高，平通监听无法监控到变化，需要用到deep:true,但是会出现val与oldVal值相同的问题，所以要用到计算属性computed，计算属性中写：**

**Op:{**

**Pp:[**

**{name:98}**

**]**

**}**

**Tab(){**

**Return JSON.parse(JSON.stringify(this.op))**

**}**

**watch:{**

**Tab:{**

**handler(val,oldVal){**

**Console.log(val,oldVal)**

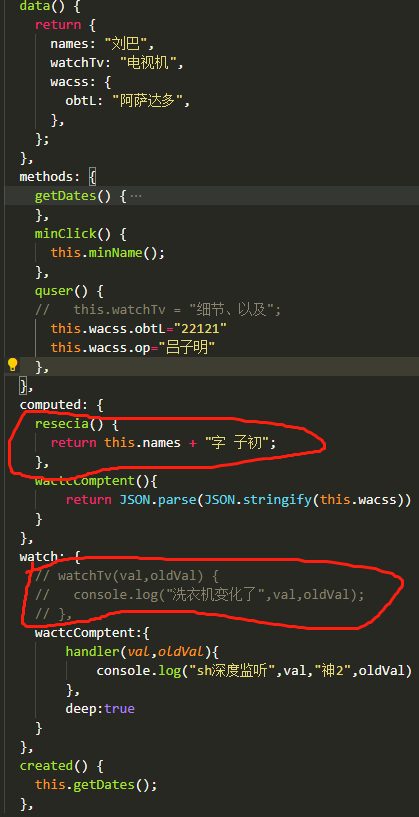
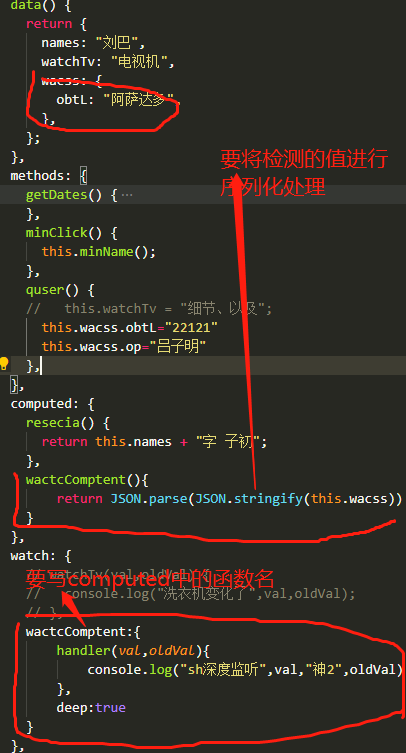
**}，**

**deep:true**

**}**

**}**

**handler是深度监听的函数必须这样写，Tab是指计算属性中写的函数名**

**二十四：路由**

**获取路由相关的信息**

this.$router this.$route

 返回上一层

this.$router.go(-1)

**router也可以插传输数据**



**跳转页面**

this.$router.push('/nagivation/two')

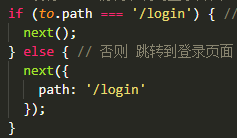
**写了 path，params 会被忽略，**



**Redirect 路由重定向**

**1.路由全局守卫导航**

**进入任何页面前，先判断时候存在token，如果存在跳入首页，否则进入登录页面**

**to即将跳转的页面，from来自哪个页面，**

**next（） 进行下一步**

**next(‘/login’) 跳到下一个界面**

**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。**

**2.全局解析守卫**

**router.beforeResolve 注册一个全局守卫。这和 router.beforeEach 类似，区别是在导航被确认之前，同时在所有组件内守卫和异步路由组件被解析之后，解析守卫就被调用**

**3.全局后置钩子**

**outer.afterEach((to, from) => {**

**// ...**

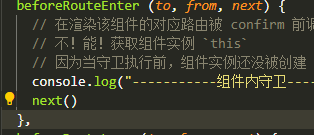
**})**

**你也可以注册全局后置钩子，然而和守卫不同的是，这些钩子不会接受 next 函数也不会改变导航本身：**

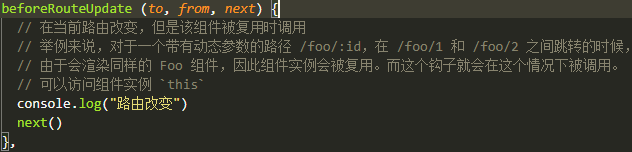
**4.路由独享守卫**



**5.组件守卫**

**组件使用前调用**

**离开组件前被调用**



**二十五、路由权限配置**

**addRoutes（）添加路由**

* 1. **利用meta对象控制权限**



**在main.js中写上全局导航并进行判断，roles：【‘admin’】，只有角色是admin的能够访问**



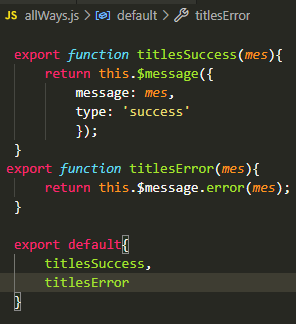
if(*to*.meta.roles.indexOf(roles)>-1){

      next()

    }

**对比roles中是否与登录角色人员的权限匹配**

**二十六、自定义方法封装**



**二十七、数组循环方法**

