Vue

CDN方式：

<script src="https://cdn.jsdelivr.net/npm/vue"></script>引入vue

创建vue实例：var app = new Vue({

El:’#app’,

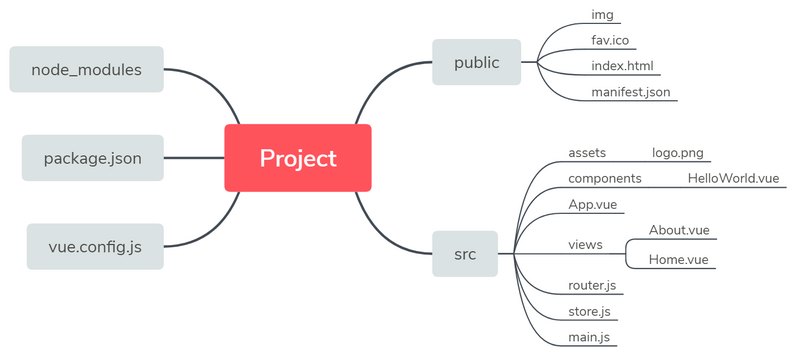
Data:{…},

Methods:{test:function(){}},

})

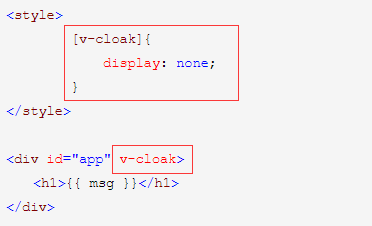
Vue中不兼容原生的绑定事件，如click

Vue中使用click()只触发元素的点击事件



## 小技巧

1. v-cloak --vue页面刷新一瞬间出现样式未加载或者显示vue代码问题



1. 关闭eslint

lintOnSave: false,并重启项目！

1. 打包vue项目资源找不到

新建vue.config.js文件

参考资料：<https://www.cnblogs.com/fqh123/p/10850311.html>

const webpack = require('webpack')

module.exports = {

baseUrl: process.env.NODE\_ENV === 'production' ? './' : '/', *//duild打包*

outputDir: 'dist',

lintOnSave: false,

runtimeCompiler: true, *//关键点在这*

*// 调整内部的 webpack 配置。*

*// 查阅 https://github.com/vuejs/vue-doc-zh-cn/vue-cli/webpack.md*

chainWebpack: () => {},

configureWebpack: {

plugins: [

new webpack.ProvidePlugin({

$: "jquery",

jQuery: "jquery"

})

]

},

*// 配置 webpack-dev-server 行为。*

devServer: {

open: process.platform === 'darwin',

host: '0.0.0.0',

port: 8080,

https: false,

hotOnly: false,

*// 查阅 https://github.com/vuejs/vue-doc-zh-cn/vue-cli/cli-service.md#配置代理*

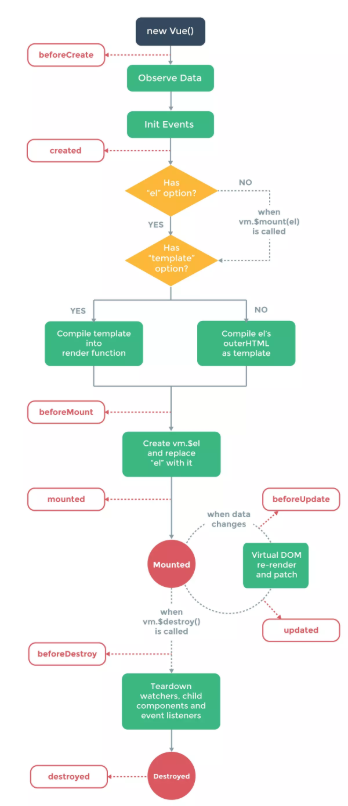
proxy: null, *// string | Object*

before: *app* => {}

}

}

## 生命周期



beforeCreate --组件实例刚被创建，处在组件属性计算之前，如data属性等

因浏览器加载方式，从上到下，创建Vue时，页面本身dom结构已加载，但关于vue还未加载。

created --组件实例创建成功，属性已绑定，DOM还未生成，$el不存在

beforeMount --模板编译/挂载之前

mounted --模板编译/挂载之后

Mounted --模板编译/挂载之后（不保证组件已在document中）

beforeUpdate --组件更新之前

updated --组件更新之后

activated --for keep-alive,组件被激活时调用

deactivated --for keep-alive,组件被移除时调用

beforeDestory --组件销毁前调用

destoryed --组件销毁后调用

beforecreate : 举个栗子：可以在这加个loading事件   
created ：在这结束loading，还做一些初始化，实现函数自执行   
mounted ： 在这发起后端请求，拿回数据，配合路由钩子做一些事情  
beforeDestroy： 你确认删除XX吗？ destroyed ：当前组件已被删除，清空相关内容

## 方法

$mount(dom)【挂载元素，返回自身，如下放回component对象】：



Vue.nextTick(funtion(){}) 【等待执行,当页面DOM**循环更新完毕后**执行此内方法】

input**实时监控**内容事件： v-on:input=“”

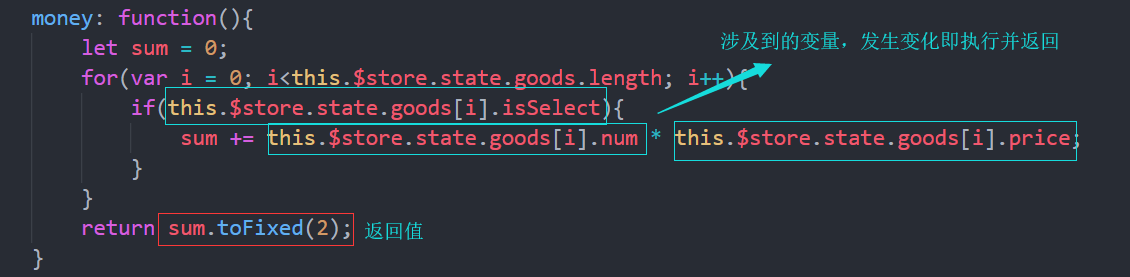
强制重新渲染视图： 进入**updated**生命周期

this.$forceUpdate() | vm.$forceUpdate()

## 数据

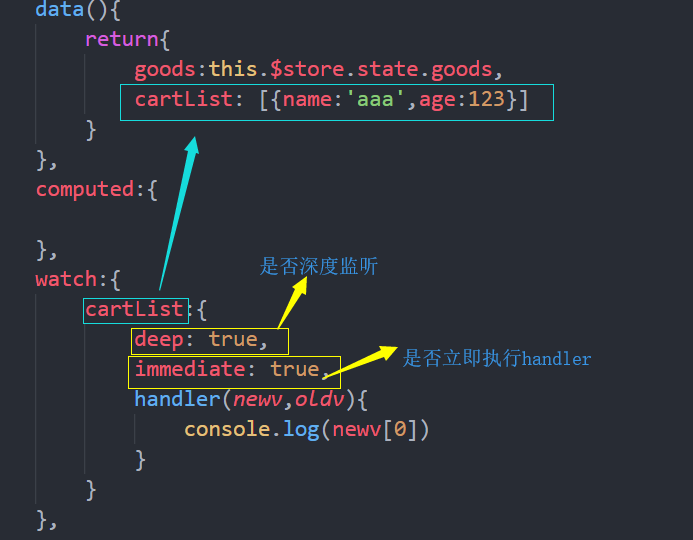
### computed

--自动监听computed使用到的变量，自定义变量可直接使用，类似于动态加载了此变量

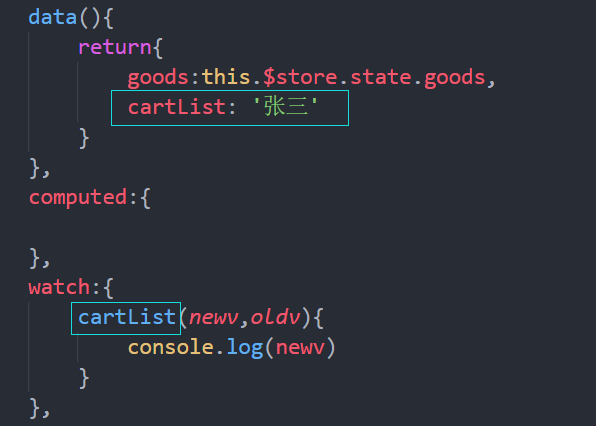


### watch

--监听对象、数组值，发生改变后立即执行其中方法



--监听普通变量



--伪监听其他组件变量

1、在想要监听的组件A中添加watch - $emit()

2、在本组件B data添加新对象

3、本组件B mounted添加$on接收A组件发来的信息并赋值给B data 新变量

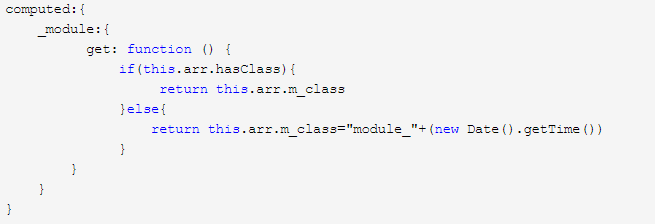
当A组件的监听的变量变化时，B组件也会同步获取到变化的值，达到监听效果

## 赋值

### 绑定类名 – 动态替换类名

1. 三元表达式
2. computed监控替换类名





3、绑定数组

:class=’[name1, name2…]’ name：data中数据

### data对象中添加属性

Vue.set(vm.messageData[0],"messageTest","哈哈哈哈哈")

messageTest: "哈哈哈哈哈"

### data 赋值

vue中不能直接将数组赋值给data

原因：当直接赋值数组时，此时对象中存放的是赋值数组的存储地址，此时使用data.xxx便无法搜索到目标数据

解决方法：将赋值数组每一项push到对象中，也就是将具体的数值传递给对象

## 修饰符

事件修饰符：

.stop --停止事件的触发执行

.prevent --取消默认事件（如链接点击取消链接跳转）

.self --只绑定自身，取消冒泡

.once --只能使用一次

.captrue --正常冒泡为从内向外冒泡，使用此修饰符后，从外向内传导

.passive --相当于给事件添加一个懒加载.lazy,大多数浏览器不会在滚动|触摸事件

中preventDefault(),但是浏览器不知道到底会不会执行此方法，

所以每次等到整个方法执行完毕后再判断是否运行滚动，导致滚

动时出现卡顿甚至无效的情况，使用此修饰符后就等于告诉浏览

器没有preventDefault，放心滚动吧，也就使得滚动流畅许多

.native --对组件事件添加修饰，将组件转化为一个html元素，普通html元素

使用此修饰符会使事件无效

鼠标修饰符

.left --左键点击

.right --右键点击 -暂无效果

.middle --中键点击 -暂无效果

键值修饰符

.keyup | keydown --键盘按键抬起 | 按下

.keyCode --指定按下某键才触发，如：@keyup.alt.13 = ()=>{} 13:enter

键盘各键对应的ascll码：

<https://zhidao.baidu.com/question/266291349.html>

全局config.keyCodes对象自定义按键修饰符别名：Vue.config.keyCodes.f1 = 112

常用的按键别名： .enter .tab .delete .space .esc .up .down .left .right

【.ctrl .alt .meta .shift】系统修饰键

.exact --普通修饰键触发，限制系统修饰键【@click.ctrl.exact】

如：@keyup.65.exact = ()=>{} 65:A

v-bind修饰符

.sync --简化父子组件数据 双向绑定



使用后：



.prop --使绑定的值作为元素属性，而不是被解析为props继承值

.camel --将绑定对象渲染为驼峰式命名

表单修饰符

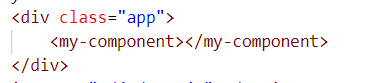
.lazy --懒加载（失去焦点后再更新出来）

.trim --去除空格（末尾空格）

.number --输入的东西转化为数字，需开头为数字，否则无效

## 组件

### 全局组件



//创建组件

Vue.component('my-component',{

template: `

<div class="box">

<ul>

<li>123</li>

<li>234</li>

<li>345</li>

<li>456</li>

</ul>

</div>

`

})

//实例化vue

new Vue({

el: '.app',

data:{}

})

成功挂载至页面：



extend注册方式：

<div class="app">

<div class="box"></div>

</div>

var myComponent = Vue.extend({

template: `

<div class="box">

<ul>

<li v-for='list in arr'>{{list}}</li>

</ul>

</div>

`,

data(){

return {

//此时arr只为myComponent的局部数据

arr: [123,234,345,456]

}

}

})

//创建myComponent实例，并挂载到一个元素 .box

new myComponent().$mount('.box')

效果：



### 局部组件

<div class="app">

<!-- 绑定数组arr -->

<my-component v-bind:arr='arr'></my-component>

</div>

//创建局部组件

var myComponent = {

template: `

<div class="box">

<ul>

<li v-for='list in arr'>{{list}}</li>

</ul>

<input type='text' v-model='arr[0]'>

</div>

`,

props: ['arr']

}

// 实例化vue

new Vue({

el: '.app',

data:{

arr:[123,234,345,456]

},

components:{

//注册至Vue中：my-components 只能在此vue中使用

'my-component': myComponent

}

})

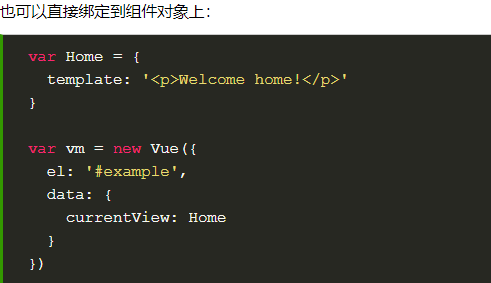
效果：绑定数字、对象可修改父组件中数值



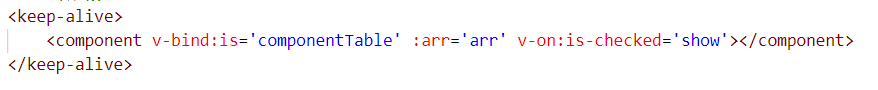
### 动态组件

根据component元素，对其is特性进行动态绑定，可以在同一个挂载点动态切换多个组件：





当需要频繁切换组件时，可以使用<keep-alive></keep-alive>,添加后，当切换组件时，不会直接删除加载的组件，而是以缓存的形式暂时保存，避免重复渲染：







### prop

允许外部环境传递数据给组件

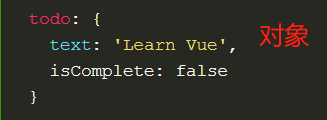
props: [index] index与组件上绑定内容 :index一致



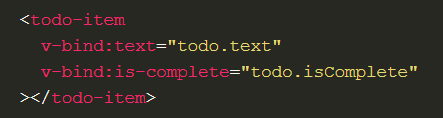


实例1：

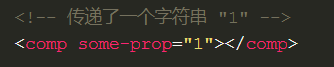
【如果想把一个对象的所有属性作为prop进行传递，可使用不带参数的v-bind】

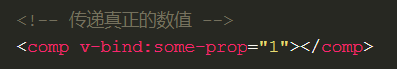
 

等价于：



若想在标签中传递一个javascript数值，使用v-bind绑定，否则其为一个字符串：





**注意：**

**prop是单向数据绑定，当父组件数据变化时，其中子组件数据相应变化，反过来则不会； js中对象和数组是引用类型，若prop值为对象或数组，子组件中改变会导致父组件中数值变化！**

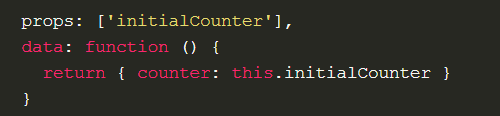
**易修改子组件数据的情况：**

**当prop作为初始值传入后，子组件想把它当作局部变量使用**

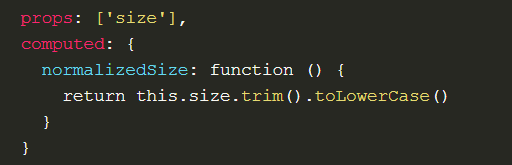
**当prop作为原始数据传入后，有子组件处理成其他数据输出**

**应对方式：**

1. **定义一个局部变量，并用prop的值初始化：**



1. **定义一个计算属性，处理prop值并返回：**



### slot

允许外部环境将额外的内容组合在组件中

**组件插槽，可以让组件内容添加到插槽处，类似于组件的替换效果**

当父组件的引用子组件标签中没有内容时，子组件中的slot的内容才会显示

<slot text='哈哈哈'>测试：</slot>

<my-component v-bind:arr='arr' v-on:is-checked='show'>

<!-- <h2>我是一个测试1 slot-scope</h2>

<h3 slot-scope='prop'>{{prop.text}}</h3> -->

</my-component>



slot-scope可添加数组或对象：

<my-component v-bind:arr='arr' v-on:is-checked='show'>

<h2>我是一个测试1 slot-scope</h2>

<h3 slot-scope='prop'>{{prop.text}}</h3>

</my-component>

//my-component

<div class="box">

<ul>

<li v-for='list in arr'>{{list}}

<slot :text='list'>测试：</slot>

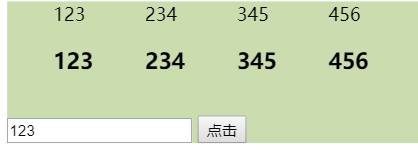
</li>

</ul>

<input type='text' v-model='arr[0]'>

<button @click='clickThis'>点击</button>

</div>



<slot></slot>写于子组件之中，父组件中调用组件的标签之内的内容融合后会将slot替换，

如下图：<my-component>标签中的两个p替换到了了子组件中的slot

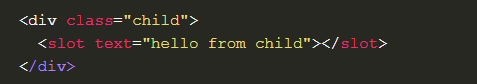


具名插槽：根据slot名称对应插槽



作用域插槽：在父级中，具有特殊性slot-scope的<template>元素必须存在，表明它是作用域插槽的模板。slot-scope的值被用于一个临时变量名，此变量接收从子组件传递过来的prop对象：

child：





### 事件(由内而外触发事件)

允许从组件内触发外部环境的副作用

**一、**

组件之间（不论组件是否有关联）传递数值：

let **pc** new Vue(); 作为公共平台，空也可以，起到搭线作用

one组件: **pc**.$emit(‘sameName’,”**数据**”)

two组件： **pc**.$on(“sameName”,function(**datas**){console.log(**datas**)})

$on事件触发后才能开始接收$emit传递的值

**二、**

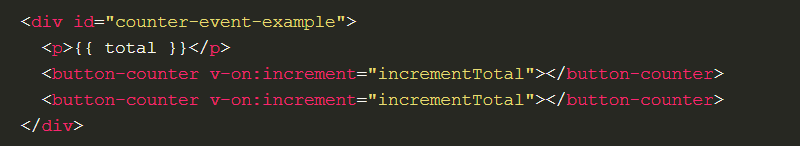
父组件使用prop传递数据给子组件，当子组件想跟父组件通信时，即可使用Vue的自定义事件系统

使用v-on绑定自定义事件：

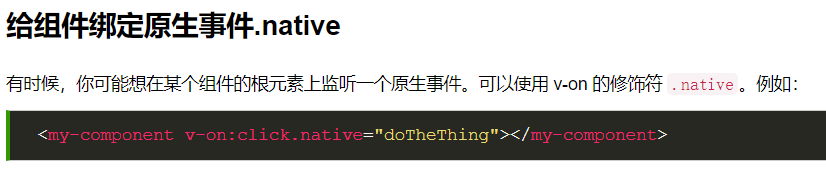
使用$on(eventName)监听事件

使用$emit(eventName)触发事件

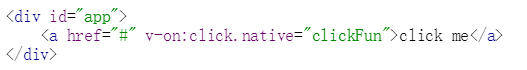
父组件可以在使用子组件的地方直接用v-on监听子组件触发的事件，不能用$on监听子组件按释放的事件，必须在模板里直接用v-on绑定







只在组件根元素上绑定有效，非组件无效：



有效：



数据绑定原理：



子组件触发父组件方法：

子组件中绑定事件 -> 触发方法中绑定父组件触发方法

<div class="app">

<!-- 绑定数组arr -->

<!— v-on:is-checked=’show’ 添加监听事件 -->

<my-component v-bind:arr='arr' v-on:is-checked='show'>

</my-component>

</div>

//创建局部组件

var myComponent = {

template: `

<div class="box">

<ul>

<li v-for='list in arr'>{{list}}

</li>

</ul>

<input type='text' v-model='arr[0]'>

<button @click='clickThis'>点击</button>

</div>

`,

props: ['arr'],

methods: {

//子组件方法

clickThis(){

//触发父组件方法

this.$emit('is-checked',100);

}

}

}

// 实例化vue

new Vue({

el: '.app',

data:{

arr:[123,234,345,456]

},

components:{

//my-components 只能在此vue中使用

'my-component': myComponent,

},

methods:{

//父组件方法

show(value){

alert(value);

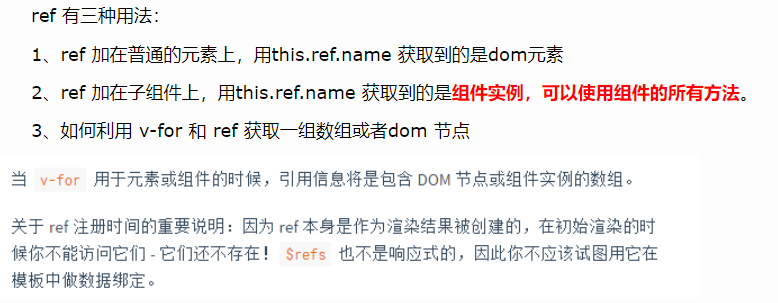
}

}

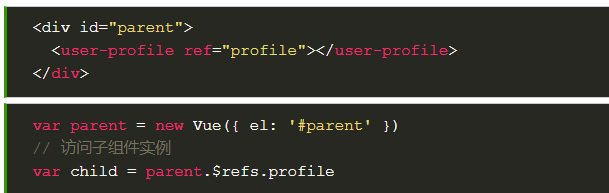
})

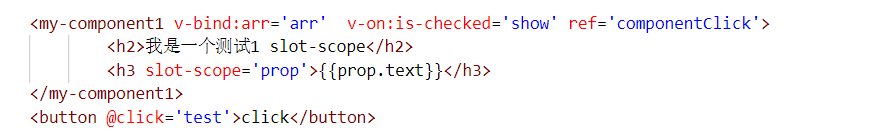
直接访问子组件对象、方法：

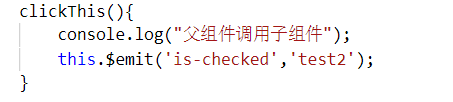
组件名中添加ref：

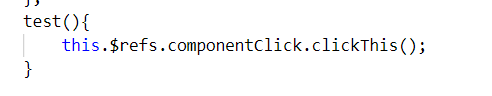


**注意点：如果组件中使用v-if或者v-show等方法，此时$ref.load只会undefind，因为mounted阶段它们不存在！**









## Vuex

核心理解：相当于继承中的父类，所有组件都可以使用公共store，子组件通过获取state来显示或隐藏，达到一种状态管理的模式

导入： npm install vuex –save

main.js引入vuex, 并Vue.user(Vuex)

const store = new Vuex.Store({

state:{

users: [

{name: “123”, age: 22}

]

}

})

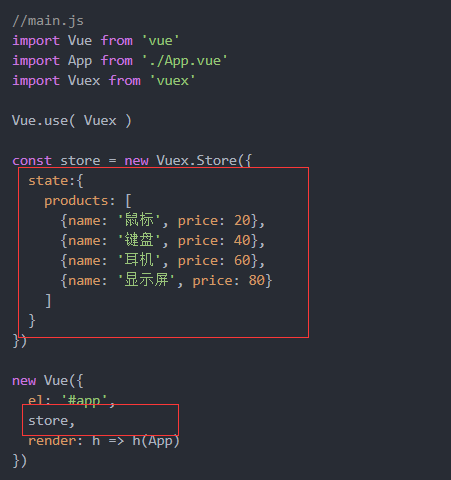
子组件使用：

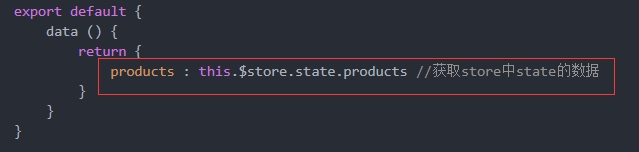
data => datas: this.$store.state.users

概念：

### State：

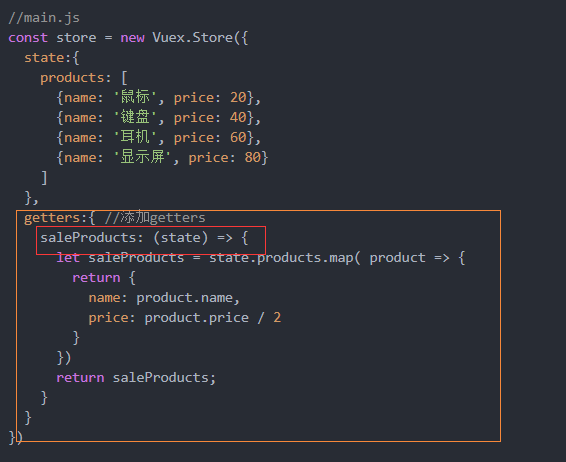
相当于vue -> data

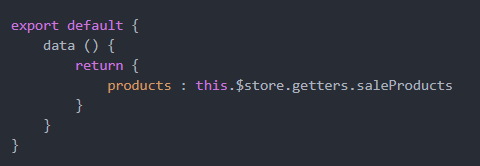




### Getters：

类似于所有组件的computed属性





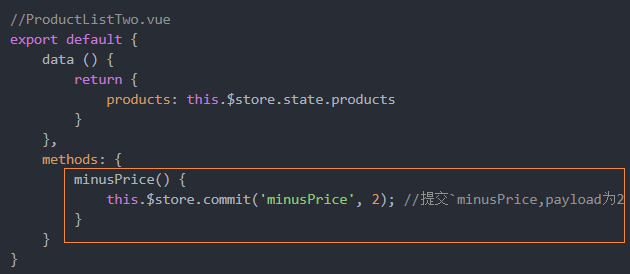
### Mutations：

等于store中的methods，mutations对象中保存这更改数据的回调函数，

官方规定为type,第一个参数是state，第二个参数是payload（自定义参数）

调用：this.$store.commit(‘mutations中的方法名’, 自定义参数)





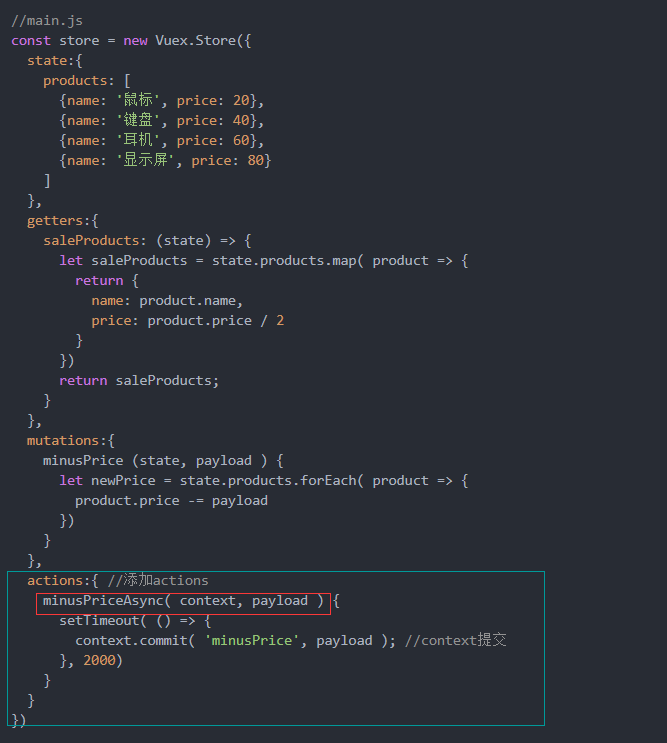
### Actions：

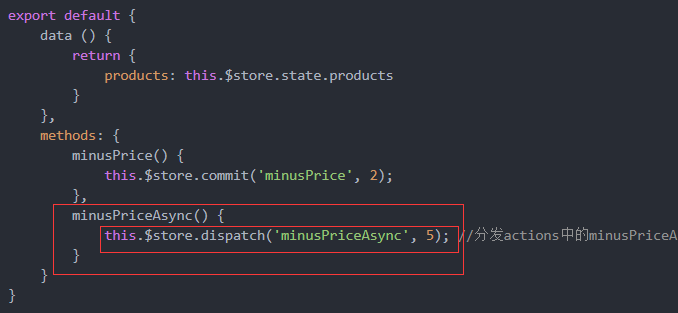
类似于mutations, 【actions提交Mutations而不是状态】

【actions允许异步操作，mutations不允许出现异步】

【actions的回调函数第一个参数是context,是一个与store实例具有

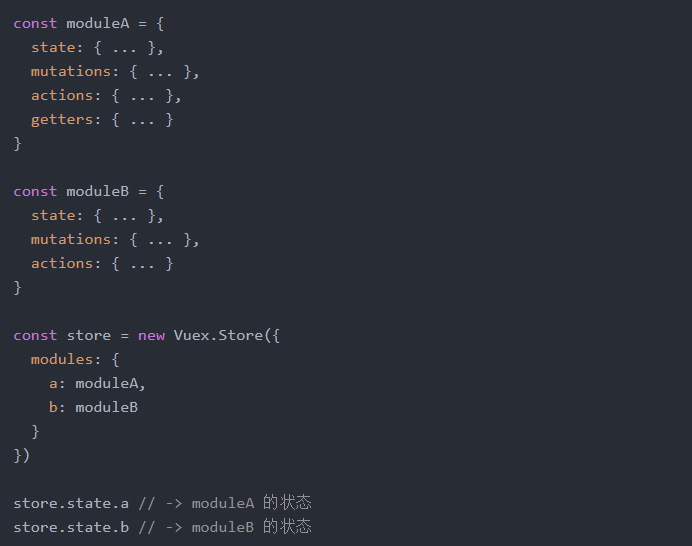
相同属性和方法的方法】





### Modules：

store模块，每个模块拥有自己的state、mutaion、actions、getter、子模块



### 技巧：

#### 简写context：commit

context 对象包含以下属性：

{

state, *// 等同于 `store.state`，若在模块中则为局部状态*

rootState, *// 等同于 `store.state`，只存在于模块中*

commit, *// 等同于 `store.commit`*

dispatch, *// 等同于 `store.dispatch`*

getters, *// 等同于 `store.getters`*

rootGetters *// 等同于 `store.getters`，只存在于模块中*

}

increment: ({ *commit* }) => commit('increment') 等同于：

increment: ({commit: *context*.*commit*}) => context.commit('increment'),

参考资料：<https://blog.csdn.net/m0_37690351/article/details/82764428>

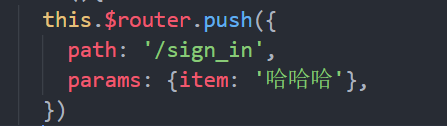
#### vue监控vuex – state数据

1. 在store – getters中添加计算属性 – 监控的值
2. 在vue - computed

## 路由

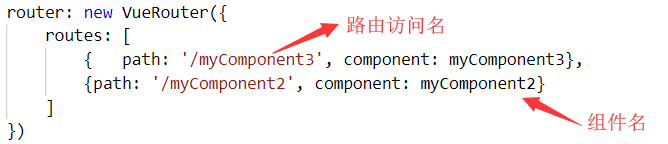
### 添加路由

跳转路由（push仅仅作为跳转路由，类似a -> href）





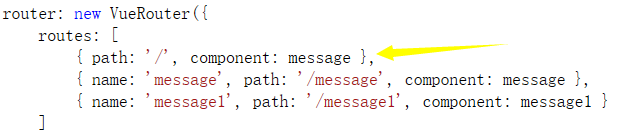
1. 引入vue-router
2. vue中添加router：

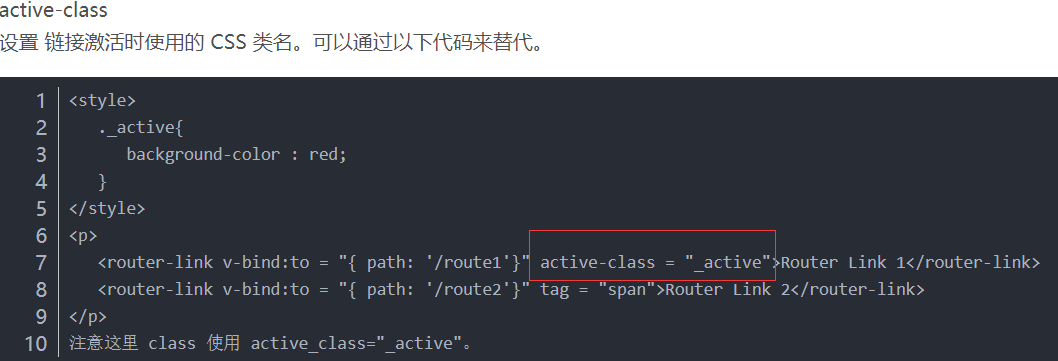


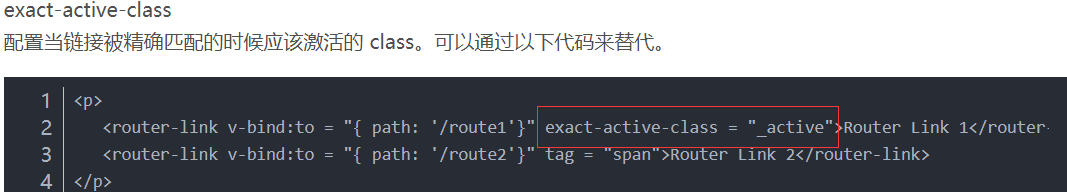
1. 使用路由：



1. 路由默认显示组件：

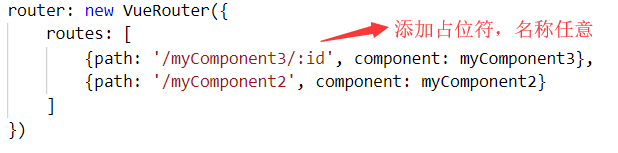


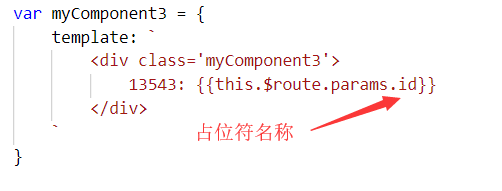




### 路由传参

1. url传参：









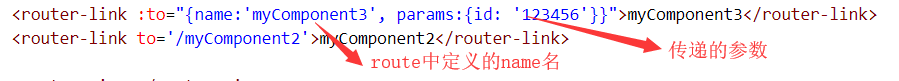
1. 命名传参：

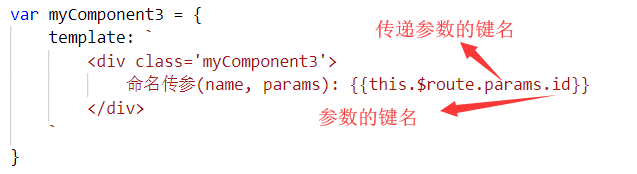
2.1 name、param传参

**缺点：每次点击路由后再点击其他路由需要返回上一级才能点击其他路由**

**优点：链接中不显示参数，类似post**





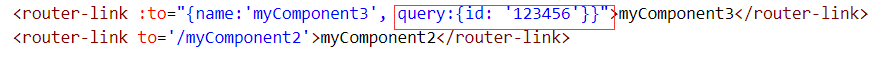


2.2 name、query传参，同上类似

**优点：可以连续点击跳转路由**

**缺点：链接栏中显示参数，类似get**



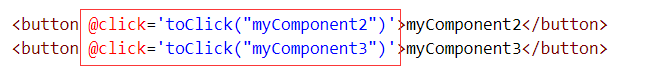


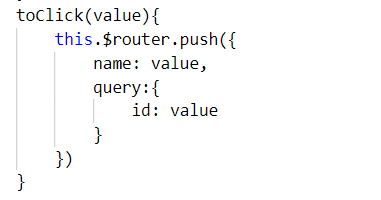
2.3 path、query传参与name传参一致：

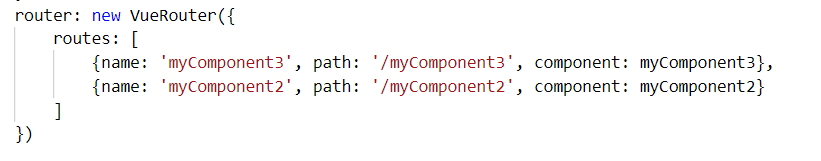
注意：path 不能与params配对



1. 编程式(js)传参

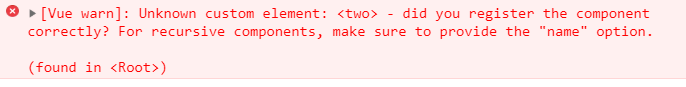






## 常见的小错误

1. 组件注册时显示注册不正确



可能原因： 注册的名字错误

注册的组件中没有根标签

创建组件写于Vue生命之后（js的右上到下运行方式，未加载到此组件）

1. 数组数值变化无法监测

watch中无法监测整个数组情况，--性能消耗和收入不成比例

解决方法：

监测数组的具体的数值，或者computed中自定义一个监测对象，如果此监测对象为true或者>0等条件（自定义，以判断是否改变），再自主的更新数组，达到监听的目的