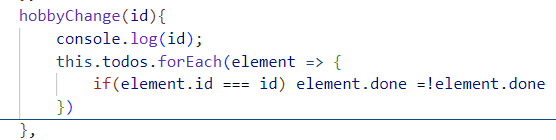
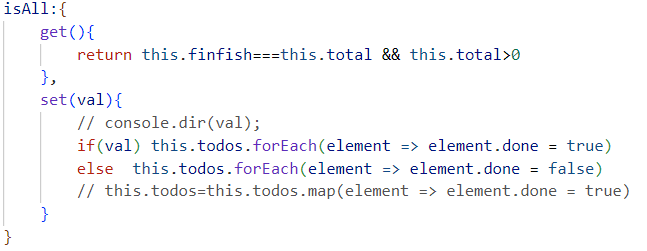
**数组操作(数组去重 求某个字符出现的次数与位置 统计出现次数最多的字符 字符串的全部替换 while循环 indexof判断 repalece替换)**

**（06）全局事件总线**

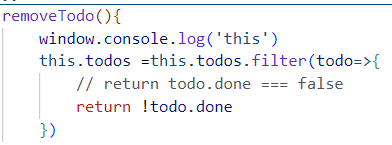
**Foreach（可以修改每个对象的属性值）**



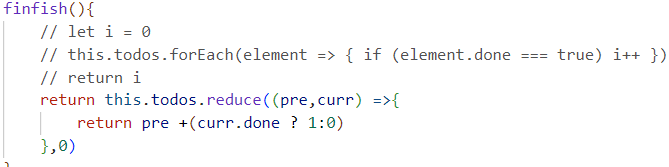


**Filter（删除）返回符合条件的数组元素**

**筛选出符合条件的对象**



**Render 计数 不能指定this对象**



**IndexOf 字符串也可以筛选 比如可以进行模糊查询 (记住 ‘123’.indexOf(‘’) =0 而不是-1) 可以写在计算属性之中 初始化的时候就会加载 不是-1 那么返回整个数组。 列表渲染/列表过滤**

**this.todoList.filter( item => {**

**return item.name.indexOf(this.param) !==-1**

**})**

**findIndex 判断元素（对象）是否在这个数组中 注意如果数组中的元素是对象时 不能通过IndexOf来判断 只能通过findIndex来判断。**

**判断对象中是否有属性 通过 todo.hasOwnProperty(‘item’) （16\_src\_TodoList\_nextTick）**

**Splice（statr,del,insert） 改变原数组**

**Vue模板语法**

Vue模板语法有2大类：

1.插值语法：

功能：用于解析标签体内容。

写法：{{xxx}}，xxx是js表达式，且可以直接读取到data中的所有属性。

2.指令语法：

功能：用于解析标签（包括：标签属性、标签体内容、绑定事件.....）。

举例：v-bind:href="xxx" 或 简写为 :href="xxx"，xxx同样要写js表达式，且可以直接读取到data中的所有属性。

备注：Vue中有很多的指令，且形式都是：v-????，此处我们只是拿v-bind举个例子。

**Vue数据绑定**

单向绑定（V-bind） 数据从data到页面

双向绑定（v-model）双向修改（只应用于表单元素 v-model:value=“” 可以简写成 v-model=“”）

vm.$mount(‘#root’) ===document.querselsetor(‘#root’)

**v-model 在单个复选框时 只有两个结果布尔值 为真是checked=true 此时你设置checked没有用  
v-model 在单选框时 如果不设置value 则单选都可以选.**

**v-moder=“s” 此时s的值为value所设置的值.**

**v-show 不支持 <template> 元素**

**v-moder有修饰符：number layz trim() 表单是多选框时 用数组**

**数据代理**

数据代理：通过一个对象代理对另一个对象中属性的操作（读/写）

**1.Vue中的数据代理：**

通过vm对象来代理data对象中属性的操作（读/写）

**2.Vue中数据代理的好处：**

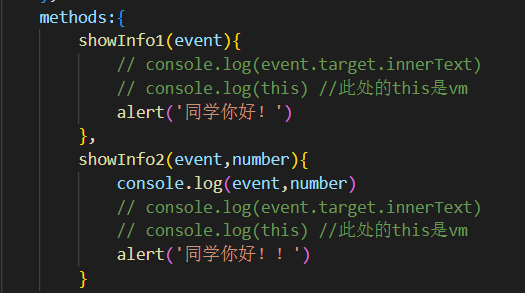
更加方便的操作data中的数据

**3.基本原理：**

通过Object.defineProperty()把data对象中所有属性添加到vm上。为每一个添加到vm上的属性，都指定一个getter/setter。在getter/setter内部去操作（读/写）data中对应的属性。

**事件操作**

@click="demo" 和 @click="demo($event,xxx)" 效果一致，但后者可以传参；



文本

描述已自动生成

文本

描述已自动生成

**计算属性**

1.定义：要用的属性不存在，要通过已有属性计算得来。

2.原理：底层借助了**Objcet.defineproperty**方法提供的getter和setter。

3.get函数什么时候执行？

(1).初次读取时会执行一次。 (2).当依赖的数据发生改变时会被再次调用。

4.优势：与methods实现相比，**内部有缓存机制（复用）**，效率更高，调试方便。

5.备注：

1.**计算属性最终会出现在vm上，直接读取使用即可**。

**2.如果计算属性要被修改，那必须写set函数去响应修改，且set中要引起计算时依赖的数据发生改变。 只有getter时可以简写。**

**属性监视**

**监视属性watch：**

1.当被监视的属性变化时, 回调函数自动调用, 进行相关操作2.监视的属性必须存在，才能进行监视！！（**不存在貌似不报错**）

3.监视的两种写法：

(1).new Vue时传入watch配置 (2).通过vm.$watch监视（‘ishot’）**带引号**

**深度监视：**

(1).Vue中的watch默认不监测对象内部值的改变（一层）。(2).配置deep:true可以监测对象内部值改变（多层）。

备注：

(1).Vue自身可以监测对象内部值的改变，但Vue提供的watch默认不可以！

(2).使用watch时根据数据的具体结构，决定是否采用深度监视。

**computed和watch之间的区别：**

1.computed能完成的功能，watch都可以完成。

2.watch能完成的功能，computed不一定能完成，例如：watch可以进行异步操作（settimeout）。

两个重要的小原则：

1.所被Vue管理的函数，最好写成普通函数，这样this的指向才是vm 或 组件实例对象。

2.所有不被Vue所管理的函数（定时器的回调函数、ajax的回调函数等、Promise的回调函数），最好写成箭头函数，这样this的指向才是vm 或组件实例对象。

**Vue数据监测**

**1.Vue监视数据的原理：**

vue会监视data中所有层次的数据。

**2. 如何监测对象中的数据？**

通过setter实现监视，且要在new Vue时就传入要监测的数据。

(1).对象中后追加的属性，Vue默认不做响应式处理

(2).如需给后添加的属性做响应式，请使用如下API：

Vue.set(target，propertyName/index，value) 或

vm.$set(target，propertyName/index，value)

文本

描述已自动生成

**3. 如何监测数组中的数据？**

通过包裹数组更新元素的方法实现，本质就是做了两件事：

(1).调用原生对应的方法对数组进行更新。

(2).重新解析模板，进而更新页面。

**4.在Vue修改数组中的某个元素一定要用如下方法：**

1.使用这些API:push()、pop()、shift()、unshift()、splice()、sort()、reverse()

2.Vue.set() 或 vm.$set()

特别注意：Vue.set() 和 vm.$set() 不能给vm 或 vm的根数据对象 添加属性！！！

**修改样式 绑定样式**

**1. class样式**

写法:class="xxx" xxx可以是字符串、对象、数组。

字符串写法适用于：类名不确定，要动态获取。

数组写法适用于：要绑定多个样式，个数不确定，名字也不确定。

对象写法适用于：要绑定多个样式，个数确定，名字也确定，但不确定用不用。

**2. style样式**

:style="{fontSize: xxx}"其中xxx是动态值。 **{display:’flex’,fontSize:’40px’}**

:style="[a,b]"其中a、b是样式对象。

**nextTick**

1. 语法：```this.$nextTick(回调函数)```

2. 作用：在下一次 DOM 更新结束后执行其指定的回调。

3. 什么时候用：当改变数据后，要基于更新后的新DOM进行某些操作时，要在nextTick所指定的回调函数中执行。 this.$ref.xxx.focus()



**组件**

Vue.extend 返回的是Vuecomponent函数（构造函数）

**特别注意：每次调用Vue.extend，返回的都是一个全新的VueComponent**

我们只需要写<school/>或<school></school>，Vue解析时会帮我们创建school组件的实例对象,即Vue帮我们执行的：new VueComponent(options)。

**VueComponent.prototype.\_\_proto\_\_ === Vue.prototype**

**让组件实例对象（vc）可以访问到 Vue原型上的属性、方法。**

**图示

描述已自动生成文本

描述已自动生成**

vue.js与vue.runtime.xxx.js的区别：  
(1).vue.js是完整版的Vue，包含：核心功能+模板解析器。  
(2).vue.runtime.xxx.js是运行版的Vue，只包含：核心功能；没有模板解析器。  
2.因为vue.runtime.xxx.js没有模板解析器，所以不能使用template配置项，需要使用render函数接收到的createElement函数去指定具体内容.

**ref属性**

1.被用来给元素或子组件注册引用信息（id的替代者）

2.**应用在html标签上获取的是真实DOM元素，应用在组件标签上是组件实例对象（vc）**

3. 使用方式：

1. 打标识：```<h1 ref="xxx">.....</h1>``` 或 ```<School ref="xxx"></School>```

2. 获取：```this.$refs.xxx```

**props配置项**

**父组件给子组件传递数据，也可传递函数（方法）**

**1.props是只读的**，Vue底层会监测你对props的修改，如果进行了修改，就会发出警告，若业务需求确实需要修改，那么请复制props的内容到data中一份，然后去修改data中的数据。**props的优先级比data配置项更高，因此不能与data数据里面的属性名重名。**

**2. props适用于：**

(1).父组件 ==> 子组件 通信

(2).子组件 ==> 父组件 通信（要求父先给子一个函数）

3. 使用v-model时要切记：**v-model绑定的值不能是props传过来的值**，因为props是不可以修改的！

4.props传过来的若是对象类型的值，修改对象中的属性时Vue不会报错，**但不推荐这样做**。

**mixin(混入)**

可以把多个组件共用的配置提取成一个混入对象

**插件**

Vue.use() 使用插件

**scoped样式** 局部生效 <style scoped>

**组件的自定义事件**

**使用场景：**A是父组件，B是子组件，B想给A传数据，那么就要在A中给B绑定自定义事件（事件的回调在A中）。**子组件把数据传递给父组件。**

**绑定自定义事件：**

第一种方式，在父组件中：```<Demo @atguigu="test"/>``` 或 ```<Demo v-on:atguigu="test"/>```

**给Demo组件的实例对象VC绑定了一个自定义事件，有人触发这个事件，则里面的回调函数将被执行。**

2. 第二种方式，在父组件中（APP）：（灵活性强） **ref属性**

```js

<Demo ref="demo"/>

......

**data（）{**

**return Domename：‘’ 保存Dome组件里面传过来的数据  
}**

**methoed（）{**

**test（name，…para） …para是个数组**

**}，**

mounted(){

console.log(this) ,APP组件的实例对象

this.$refs.xxx.$on('atguigu',**this**.test)

this.$refs.xxx.$once('atguigu',**this**.test) 只触发一次

}

3. 若想让自定义事件只能触发一次，可以使用```once```修饰符，或```$once```方法。

4. **触发**自定义事件：this.$emit('atguigu',数据) **发布**

5. 解绑自定义事件```this.$off('atguigu')```

**6. 组件上也可以绑定原生DOM事件，需要使用```native```修饰符。**

7. 注意：通过this.$refs.xxx.$on('atguigu',回调) **绑定**自定义事件时（**一般在mounted钩子里面 订阅**），**回调要么配置在methods中要么用箭头函数，否则this指向会出问题**！（谁触发了这个回调函数事件，这个this就是指的谁，这里的this是Dome组件的实例对象，而不是父组件APP的实例对象）

**自定义指令**

文本

描述已自动生成

**指令名为多个单词组成 写成 v-big-number 下面的要写成 ”big-number ”用引号包起来(单引号就行) 里面钩子的this都是windows**

**插槽（slot）**

**父组件对子组件里面的内容结构有调整，可以使用插槽 。让父组件可以向子组件指定位置插入html结构，也是一种组件间通信的方式，适用于父组件 ===> 子组件**

子组件中可以给slot定义一个name属性

<slot name="header"></slot>

**父组件中可以通过 slot=‘header’添加内容（官方不推荐）**

<h1 slot=‘header’>Here might be a page title</h1>

**推荐使用下面的 v-slot 只能添加在 <template> 上**

**注意 ：是和name绑定 =是和里面的数据绑定**

<template v-slot:header> <h1>Here might be a page title</h1> </template>

**如果子组件里面有数据 父组件中的插槽想用这个数据 可以在子组件的插槽定义一个属性。然后在父组件中使用 slot-scope=” slotProps”**



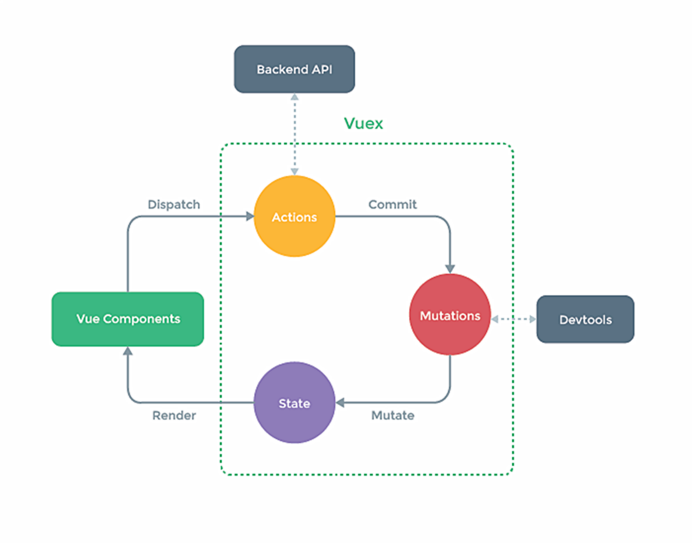
**可以解构插槽（里面是个对象（key：value））**



**在prop对象中没有user这个属性（这个user是个对象）**



**Vuex （VM VC都有store）**



**Store管理着 actions mutations state 这三个对象**

**多个组件对数据都进行操作 而不是简单的读取 要用vuex**

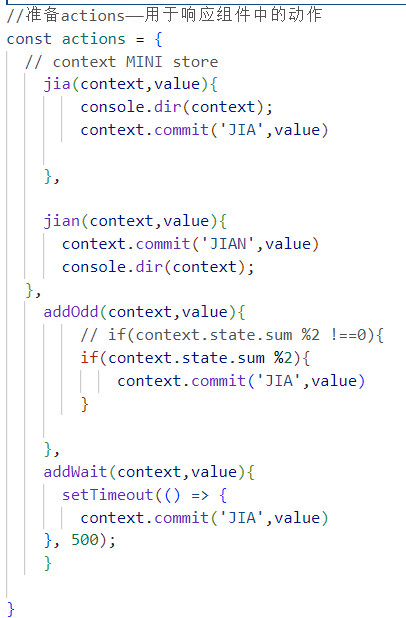
**一般会创建一个store文件夹 里面的index.js存放vuex的配置**

文本

描述已自动生成图形用户界面, 文本, 应用程序

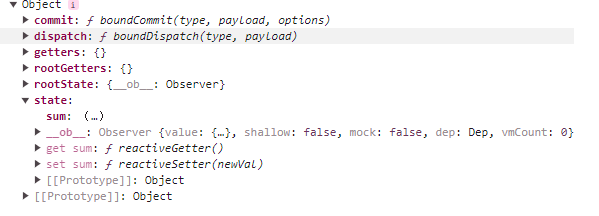
描述已自动生成namespaced:true

**actions——用于响应组件中的动作**



action里面可以设置一些数据处理的逻辑(操作) 比如加个判断 什么时候执行 继续分发dispatch

content是个mini store（上下文） 里面有commit 可以向mutations提交里面的回调函数



**mutations——用于操作数据（state）**

里面包括数据处理的函数，函数的第一个参数是state，第二个参数是组件传过来的数据

文本

中度可信度描述已自动生成 图示

中度可信度描述已自动生成

**组件里面的methods方法**

组件中修改vuex中的数据： this.$store.dispatch('action中的方法名',数据) 或 this.$store.commit('mutations中的方法名',数据)

备注：若没有网络请求或其他业务逻辑，组件中也可以越过actions，即不写dispatch，直接编写commit

如果不需要action这个对象 ，组件中也可以直接写 this.$store.commit('JIA',this.n)

文本, 日程表

描述已自动生成

**getters**

概念：当state中的数据需要经过加工后再使用时，可以使用getters加工。

可以跨组件对这个数据进行加工 相当于每个独立组件的computers

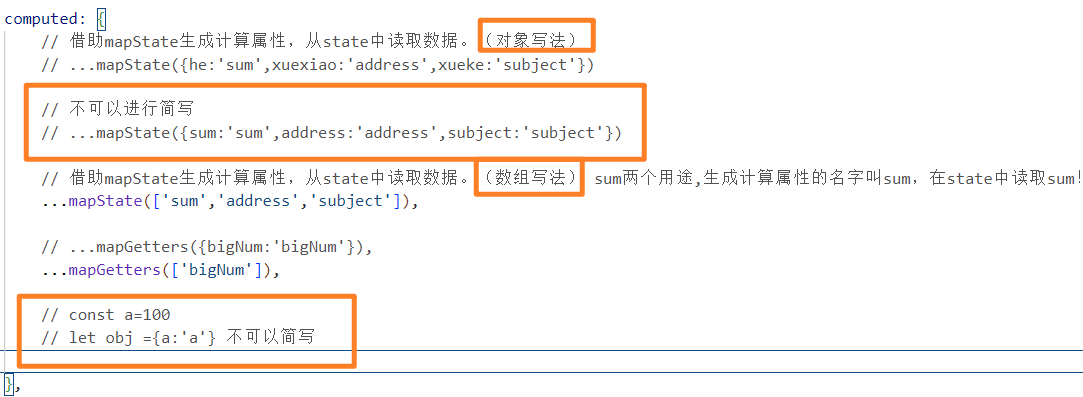


**四个map方法的使用**

可以在组件中更方便的调用Vuex中的数据与方法 不需要 $store.state.sum了 直接sum

1. mapState方法：用于帮助我们映射**state**中的数据为计算属性

2. mapGetters方法：用于帮助我们映射**getters**中的数据为计算属性



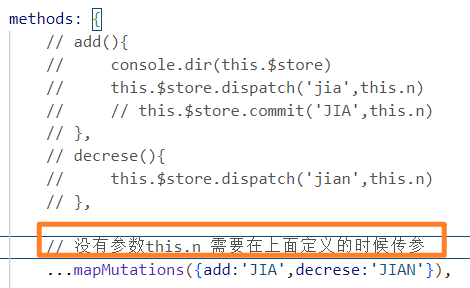
3. mapActions方法：帮助调用**this.$store.dispatch(xxx)**的函数与**actions**对话

4. mapMutations方法：帮助调用**this.$store.commit(xxx)**的函数与**mutations**对话

也有对象写法与数组写法

手机屏幕的截图

描述已自动生成



**备注：mapActions与mapMutations使用时，若需要传递参数需要：在模板中绑定事件时传递好参数，否则参数是事件对象。**

**路由 router**

理解： 一个路由（route）就是一组映射关系（key - value），多个路由需要路由器（router）进行管理。

前端路由：key是路径，value是组件。

基本使用

1. 安装vue-router，命令：npm i vue-router

2. 应用插件：Vue.use(VueRouter)

3.编写router配置项

先在src文件夹下新建router文件夹 在里面的index.js里面配置



App.vue中

<router-link active-class="active" to="/about">About</router-link>

**注意里面的to属性值中的path路径要与router配置项的path路径一致**

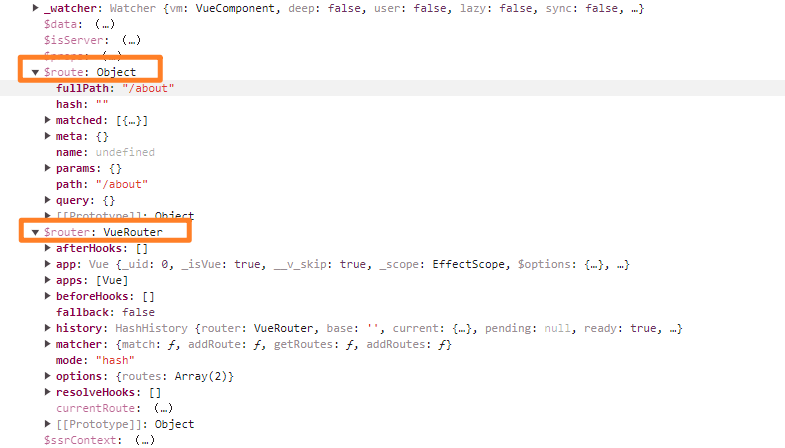
**指定展示位置**

<router-view></router-view>

**注意点**

每个路由组件一般放在src下的**view**文件夹下 且每个路由组件的实例对象都有 **$route（路由规则） 和$router属性**（每个路由组件的router对象相等，也就说整个应用只有一个router） 且**Vue的原型对象**里面也有**$route和$router对象**

**通过切换，“隐藏”了的路由组件，默认是被销毁掉的，需要的时候再去挂载！！！**



**多级路由**

**使用children配置项 //此处一定不要写：/news 但是 <router-link to="/home/news">News</router-link> 要写完整的路径**

手机屏幕的截图

描述已自动生成

**路由传参（这里是Msg组件给Detail组件传参数）**

**（1）query参数**

**在Msg组件里面传参 有两种方式 to的字符串写法与to的对象写法**

**里面的name是router配置项中设置的name**



**2. 接收参数（在Detail路由组件里面）**

this.$route.query.id

this.$route.query.title

**（2）params参数**

**首先要在router配置项中声明接收params参数（给哪个组件传的参数 就在哪个组件添加占位符这里是在Detail）**

文本

描述已自动生成



**2. 接收参数（在Detail路由组件里面）**

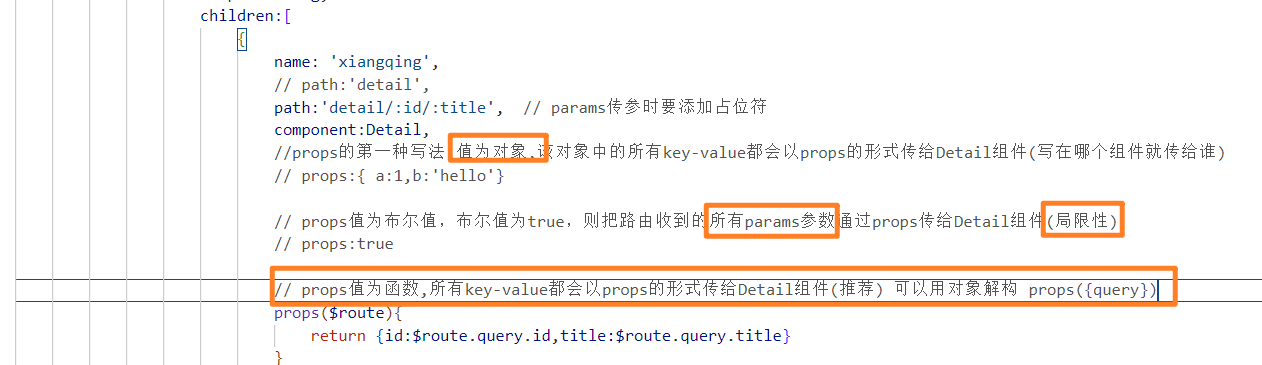
this.$route.params.id

this.$route.params.title

**（3）路由的props配置**

**​作用：让路由组件更方便的收到参数 不需要 this.$route.xxx.id 了 直接id**

**在router配置项中修改，添加props配置项（三种方式）**



**在Detail组件中配置一个props**

图形用户界面, 应用程序, Word

描述已自动生成

**<router-link>的replace属性**

**1. 作用：控制路由跳转时操作浏览器历史记录的模式**

2. 浏览器的历史记录有两种写入方式：分别为**push**和**replace**，**push**是追加历史记录（压栈）， **replace**是替换当前记录。路由跳转时候默认为**push**

**3. 如何开启replace模式：<router-link replace .......>News</router-link> 简写方式**

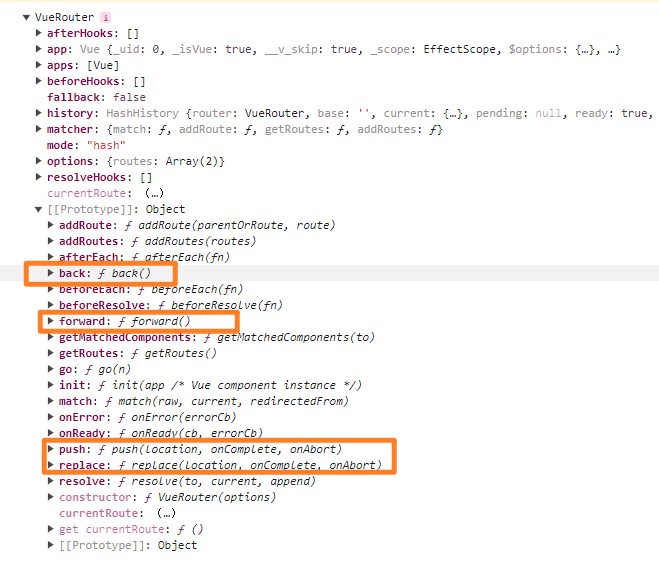
<router-link :replace = ‘true’ .......>News</router-link> 完整写法

**编程式路由导航（\*\*\*\*）**

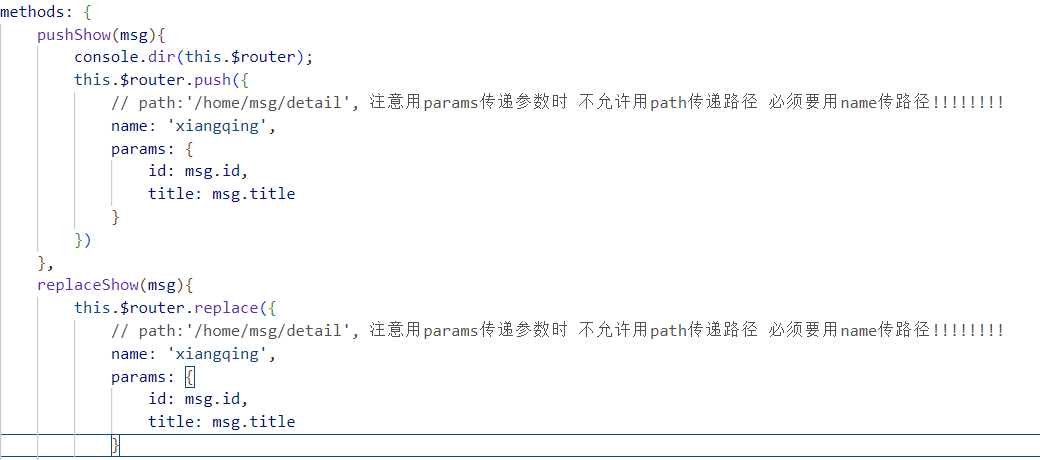
**不借助<router-link> 实现路由跳转，让路由跳转更加灵活**

实现任何标签跳转不只是a标签 常见的就是点击button跳转一个页面，也可以设置定时器

**利用router对象!!!!它的原型对象中有push replace back forward go方法 go需要传递参数**



push replace 是个对象 里面的配置和 **<router-link :to >**里面的配置项一样



this.$router.forward() //前进

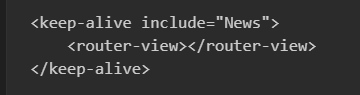
this.$router.back() //后退

this.$router.go() //可前进也可后退 正数就是前进几步 负数就是后退几步

**缓存路由组件**

作用：让不展示的路由组件保持挂载，不被销毁。

**include里面的是组件名！！!多个的话 :include=”[‘News’,’Msg’]”**



**两个生命周期钩子**

作用：路由组件所独有的两个钩子，用于捕获路由组件的激活状态。

2. 具体名字：

**activated路由组件被激活时触发。**

**deactivated路由组件失活时触发**

图示

中度可信度描述已自动生成文本

描述已自动生成

**路由守卫**

作用：对路由进行权限控制

**分类：全局守卫、独享守卫、组件内守卫**



**独享守卫 （只有前置 没有后置 可以搭配全局的后置守卫使用）在router文件中配置**

只想对一个路由进行权限控制（beforeEnter）（进入之前） 使用和前置全局路由守卫一样



**组件内路由守卫（在组件里面配置）**

在编辑文档时不小心关闭了页面可以使用beforRouteLeave来弹出警示信息

后置全局路由守卫是路由配置发生变化之后就给页面标题改名字了。

beforRouteLeave是你离开这个页面之后才给你的页面改名字。

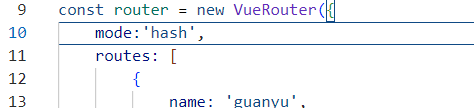


**history模式与hash模式**

**#（hash）从井号开始到最后 不会作为路径的一部分发给服务器的 前端自己玩**

路由器默认开启的就是hash模式（哈西工作模式）（兼容性好）

可以在router配置中设置mode: ‘history’ 路径不带井号（但兼容性略差）



**代理服务器**

**跨域 违背了同源策略（协议 主机名 端口号三者一致才行）**

请求发了 服务器数据也发了 但你拿不到

**解决跨域**

1.cors 后端处理 真正意义的解决跨域

2.jsonp 借助script标签 里面的src属性引入外部资源的时候不受同源策略限制这个特点 **很巧妙** 只能处理get请求 前后端都要配置

3.配置代理服务器

这个代理服务器的端口也是8080和前端请求的端口是一致的 这时就不存在跨域问题，然后代理服务器把前端的请求转发给其他的服务器（5000|5001） 5000|5001给了代理服务器 然后代理再转发给前端

**前端向5000端口号请求学生数据 5001 汽车信息**

图形用户界面, 文本, 应用程序

描述已自动生成

**配置完需要重启这个服务**

1. 优点：配置简单，请求资源时直接发给前端（8080）即可。

2. 缺点：**不能配置多个代理**，不能灵活的控制请求是否走代理。

3. 工作方式：若按照上述配置代理，当请求了前端不存在的资源时，那么该请求会转发给服务器 （**优先匹配前端资源**）



灵活，可以配置多个代理。