

## CLI之后-render

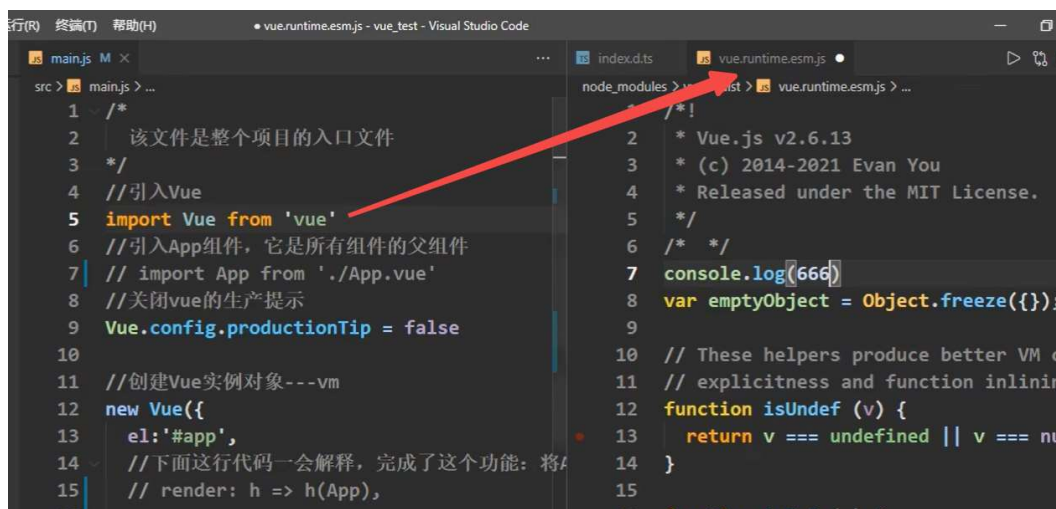
笔记本: Vue

创建时间: 2022/4/20 14:44

更新时间: 2022/4/20 15:17

作者: Chris吕

Vue 默认引入的是精简版的 esm.js 是不带有完整编译器的vue目的是提高效能. 需要render 来帮助模板解析



然后引用App变量




## 例子

白板.bt x main.js x

### 装修--铺瓷砖

第一种：  
买瓷砖（Vue核心）+ 买工人（模板解析器）====> 铺好的瓷砖+工人

第二种：  
买瓷砖（Vue核心）+ 雇工人（模板解析器）====> 铺好的瓷砖



## 小结

```
/*
  关于不同版本的Vue:

  1.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。
*/
```

## 脚手架 & ref标签

笔记本: Vue

创建时间: 2022/4/20 15:23

更新时间: 2022/4/20 17:02

作者: Chris吕



定制化脚手架

## ref

类似原生DOM里面 标签的id="title" 变成 ref='title'  
vue中获取原生dom: this.\$ref.title

ref用在组件上, 是拿的组件的VueComponent实例对象

### ## 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:
  - 打标识: `<h1 ref="xxx">.....</h1>` 或 `<School ref="xxx"></School>`
  - 获取: `this.$refs.xxx`

### ## 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:
  - 打标识: `<h1 ref="xxx">.....</h1>` 或 `<School ref="xxx"></School>`
  - 获取: `this.$refs.xxx`

## Props组件沟通

笔记本: Vue

创建时间: 2022/4/20 17:24

更新时间: 2022/4/20 17:56

作者: Chris吕

```
<Student name="李四" sex="女" age="18"/>
</div>
template>
```

使用组件, 传属性, 得在组件中有props接

age不加':' 都是字符串 改成数字 要用 :age

## 数组props 接法

```
script>
export default {
  name: 'Student',
  data() {
    return {
      msg: '我是一个尚硅谷的学生',
    }
  },
  props: ['name', 'sex', 'age']
}
/script>
```

不可以定制类型,

## 对象接法

```
props: {
  name: String,
  age: Number,
  sex: String
}
```

可以定制类型, 当传字符串过来。控制台报错

**完整接法: 可以给默认值或者输入校验**

```

props:{
  name:{
    type:String, //name的类型是字符串
    required:true //name是必要的
  },
  age:{
    type:Number,
    default:99 //默认值
  },
  sex:{
    type:String,
    required:true
  }
}

```

注意点,1. props在\_data中是没有的, 2. props里面的属性是不可以修改的

```

VueComponent { _uid: 2, _isVue: true, $options: {...}, _renderProxy:
  elf: VueComponent, ...}
  $attrs: (...)
  $children: []
  $createElement: f (a, b, c, d)
  $el: div
  $listeners: (...)
  $options: {parent: VueComponent, _parentVnode: VNode, propsData
  $parent: VueComponent { _uid: 1, _isVue: true, $options: {...}, _r
  $refs: {}
  $root: Vue { _uid: 0, _isVue: true, $options: {...}, _renderProxy:
  $scopedSlots: {$stable: true, $key: undefined, $hasNormal: fals
  $slots: {}
  $vnode: VNode {tag: "vue-component-2-Student", data: {...}, child
    age: "18"
    msg: "我是一个尚硅谷的学生"
    name: "李四"
    sex: "女"
    __VUE_DEVTOOLS_UID__: "1:2"
  }
  _c: f (a, b, c, d)
  _data:
    msg: "我是一个尚硅谷的学生"
    __ob__: Observer {value: {...}, dep: Dep, vmCount: 1}
    get msg: f reactiveGetter()
    set msg: f reactiveSetter(newVal)
    __proto__: Object

```



## mixin

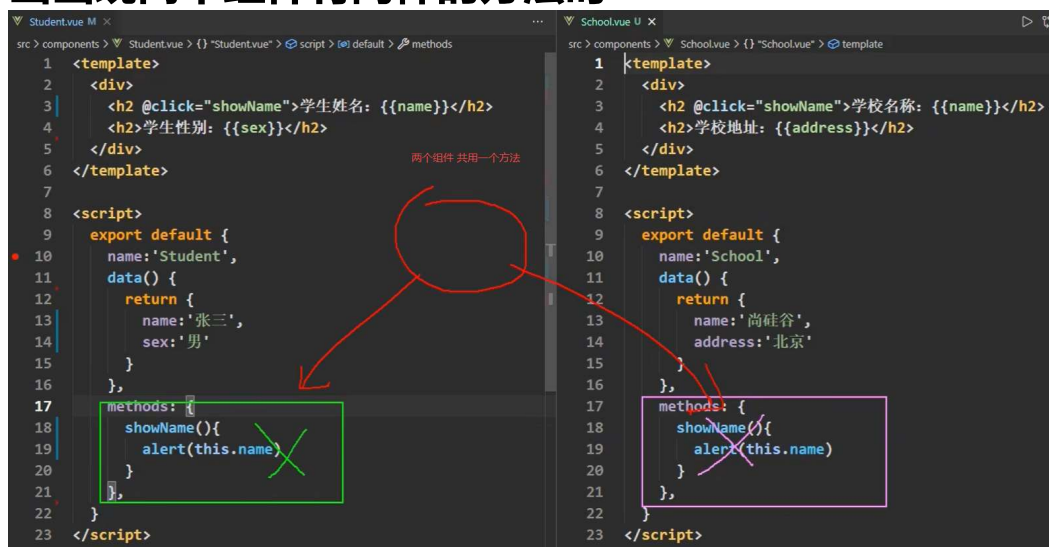
笔记本: Vue

创建时间: 2022/4/20 17:59

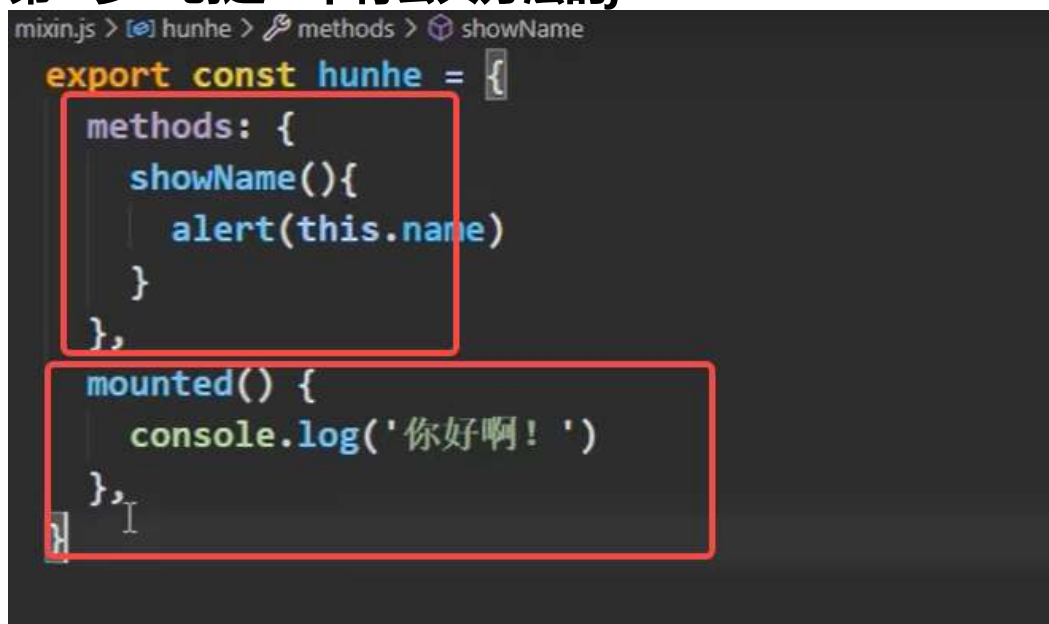
更新时间: 2022/4/20 18:11

作者: Chris吕

## 当出现两个组件有同样的方法时



## 第一步：创建一个有公共方法的js



对象中需要和vue的属性一样

## 第二步：暴露该方法

第三步：在用到该方法的组件中用 `mixins:[]` 数组，1个数据也得装在数组里面

混合优先级：以组件中的属性为主！

全局混合 所有的vm vc都会有混合中的方法，属性

```
> JS main.js > ...
1 //引入Vue
2 import Vue from 'vue'
3 //引入App
4 import App from './App.vue'
5 import {hunhe,hunhe2} from './mixin'
6 //关闭Vue的生产提示
7 Vue.config.productionTip = false
8 Vue.mixin(hunhe)
9 Vue.mixin(hunhe2)
10
11
12 //创建vm
13 new Vue({
14   el:'#app',
15   render: h => h(App)
16 })
```

vc

vm|



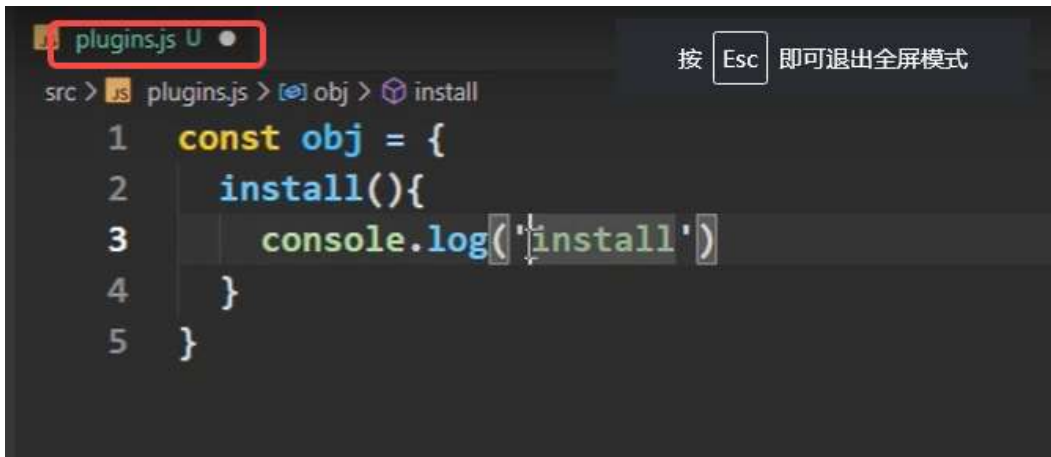
## 插件

笔记本: Vue

创建时间: 2022/4/20 18:13

更新时间: 2022/4/20 18:27

作者: Chris吕



```
src > plugins.js > [obj] > install
1  const obj = {
2    install(){
3      console.log('install')
4    }
5  }
```

1. 对象形式
2. 一定要有 `install(){} 方法`

## 使用插件



```
//应用（使用）插件
Vue.use(plugin)
```

## 插件例子



```
src > plugins.js > [default] > install
1  export default {
2    install(Vue){
3      //全局过滤器
4      Vue.filter('mySlice',function(value){ ...
6    })
7
8      //定义全局指令
9      Vue.directive('fbind',{ ...
22    })
23
24      //定义混入
25      Vue.mixin({ ...
32    })
33
34      //给Vue原型上添加一个方法
35      Vue.prototype.hello = ()=>{alert('你好啊')}
36    }
```

# 小结

```
1  ## 插件
2  功能：用于增强Vue
3  本质：包含install方法的一个对象，install的第一个参数是Vue，第二个以后的参数是插件使用者传递的数据。
4  定义插件：
5  对象.install = function (Vue, options) {
6    // 1. 添加全局过滤器
7    Vue.filter(...)
8
9    // 2. 添加全局指令
10   Vue.directive(...)
11
12   // 3. 配置全局混入(合)
13   Vue.mixin(...)
14
15   // 4. 添加实例方法
16   Vue.prototype.$myMethod = function () {...}
17   Vue.prototype.$myProperty = xxxx
18 }
19 使用插件：Vue.use()
```

## Scoped样式

笔记本: Vue

创建时间: 2022/4/20 18:27

更新时间: 2022/4/20 18:36

作者: Chris吕

---

scoped局部的样式

不适用于App.vue

lang 语法: less css, . . . . .

less: 可以嵌套写

## ToDo案例

笔记本: Vue

创建时间: 2022/4/20 18:37

更新时间: 2022/4/25 15:13

作者: Chris吕

## 待办事项案例

### 1. 拆组件

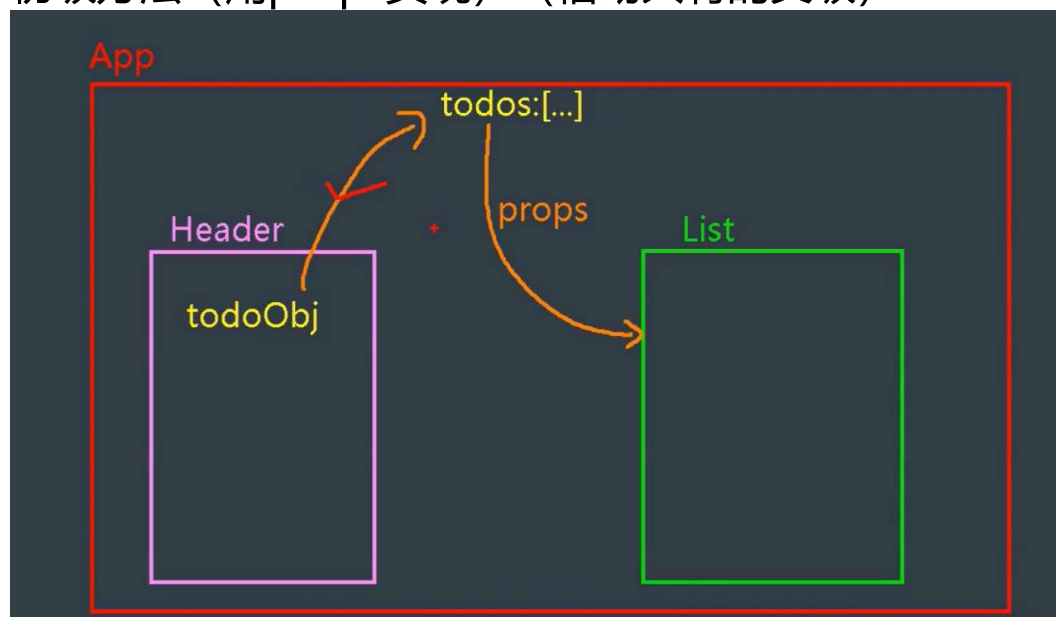
#### 3.5 Todo-list 案例



### 2. 同级组件传值



初级方法（用props实现）（借助共有的父级）



父级定义函数

```

    methods: {
      addToDo(todo) {
        this.todoList.unshift(todo);
      }
    },
  },

```

父级 (app) 定义函数, 并且用组件将函数传给子组件

子组件收取函数并且调用

```

11  props: ["addToDo"],
12  data() {
13    return {
14      title: '',
15    }
16  },
17  methods: {
18    add() {
19      const obj = {id: nanoid(), title: this.title, isDone: false};
20      this.addToDo(obj)
21    }
22  },
23 }

```



不能重名

删除一个todo

用filter

```

    handleDel(id) {
      this.todoList.forEach(item => {
        if (item.id === id) {
          this.todoList = this.todoList.filter(item => item.id !== id);
        }
      })
    }
  }

```

累加已完成

用**reduce**

```
</label>
<span>
  <span>已完成{{totalDone}} / 全部{{todoList.length}}</span>
</span>
<button class="btn btn-danger">清除已完成任务</button>
</div>
</template>

<script>
  // reduce执行累加
  export default {
    name: "EventFooter",
    props: ["todoList"],
    computed: {
      totalDone() {
        // 数组里面遍历的对象
        // 第一次为初始值, 然后下一次为函数return的值
        return this.todoList.reduce((pre, todo) => {return pre + (todo.isDone ? 1 : 0);}, 0);
      }
    }
  }
</script>
```

## 计算属性允许被套娃

```
props: ["todos"],
computed: {
  total() {
    return this.todos.length
  },
  doneTotal() {
    /* const x = this.todos.reduce((pre,current)=>{
      console.log('@',pre,current)
      return pre + (current.done ? 1 : 0)
    },0) */
    return this.todos.reduce((pre,todo)=> pre + (todo.done ? 1 : 0), 0)
  },
  isAll() {
    return this.doneTotal === this.total
  }
}
```

## 添加本地存储

一开始获取是否有存储：有则拿没有则用[]

```
todos: JSON.parse(localStorage.getItem('todos')) || []
```

编辑：如果直接 isEdit=true;没有做数据代理 要用 this.\$set(item, 'isEdit', true)

请输入你的任务名称，按回车键确认

☐ 睡觉

☐ 已完成0 / 全部1

清除已完成任务

vue-devtools

Detected Vue v2.6.13

backend.js

cjs.js?!.node\_modul...=script&lang=js

isEdit: true, ob: Observer

done: (...)

id: (...)

isEdit: true

title: (...)

\_\_ob\_\_: Observer {value: {...}, dep: Dep, vmCount: 0}

get title: f reactiveGetter()

set done: f reactiveSetter(newVal)

get id: f reactiveGetter()

set id: f reactiveSetter(newVal)

get title: f reactiveGetter()

set title: f reactiveSetter(newVal)

\_\_proto\_\_: Object

## 浏览器存储

笔记本: Vue

创建时间: 2022/4/24 17:20

更新时间: 2022/4/24 17:39

作者: Chris吕

---

### localStorage:(浏览器关掉也不会消失)

```
function saveData(){
  localStorage.setItem('msg','hello!!!')
  localStorage.setItem('msg2',666)
  localStorage.setItem('person',JSON.stringify(p))
}
function readData(){
  console.log(localStorage.getItem('msg'))
  console.log(localStorage.getItem('msg2'))

  const result = localStorage.getItem('person4')
  console.log(JSON.parse(result))

  console.log(localStorage.getItem('msg3'))
}
function deleteData(){
  localStorage.removeItem('msg2')
}
function deleteAllData(){
  localStorage.clear()
}
//console.log(localStorage.getItem('msg'))
```

2种方法清除

- 引导用户清除
- 浏览器清除

**sessionStorage:4个方法同上。但是在关闭浏览器的时候清除**

## 组件自定义事件绑定

笔记本: Vue

创建时间: 2022/4/24 17:54

作者: Chris吕

更新时间: 2022/4/25 10:48

# 适用于子给父 \$emit

## 步骤

1. 先要在要传组件那里写一个方法及定义一个事件  
@funcName 在要触发事件的组件上
2. 然后在需要触发事件的方法内 用 this.\$emit('funcName ', parameter)

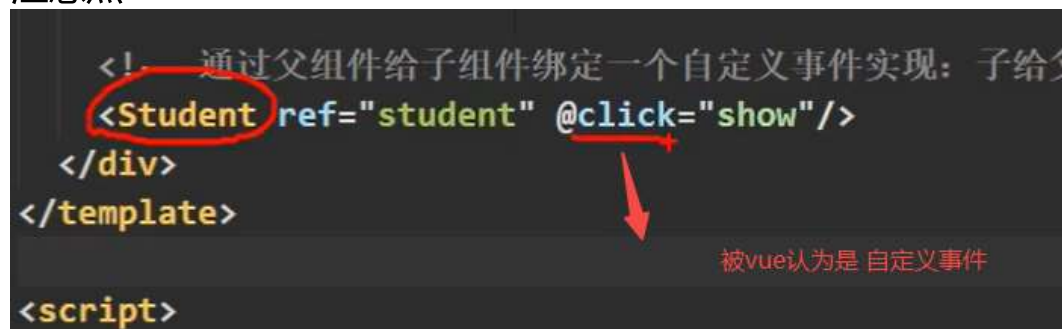
缺点: 不够灵活

## ref配合\$on使用

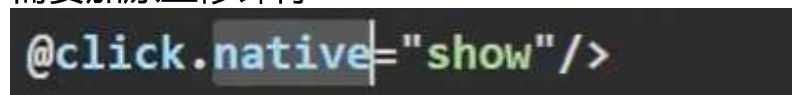
原理: ref= 'refName' 绑定组件后, 在挂载后用  
this.\$refs.refName.\$on('funcName ', parameter)

优点: 灵活运用

## 注意点



需要加原生修饰符





## 组件自定义事件解绑

笔记本: Vue

创建时间: 2022/4/25 10:18

更新时间: 2022/4/25 10:51

作者: Chris吕

## 给哪个组件绑定，就在哪个组件里面解绑

### 解绑一个自定义事件

```
unbind(){  
  this.$off('atguigu')  
}
```

### 解绑多个自定义事件

```
// this.$off('atguigu') //解绑一个自定义事件  
this.$off(['atguigu','demo']) //解绑多个自定义事件
```

用数组包着

### 解绑全部自定义事件

this.\$off()

#### 小结

##### 组件的自定义事件

1. 一种组件间通信的方式，适用于：**子组件 ==> 父组件**
2. 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）。
3. 绑定自定义事件：
  1. 第一种方式，在父组件中：`<Demo @atguigu="test"/>` 或 `<Demo v-on:atguigu="test"/>`
  2. 第二种方式，在父组件中：

```
<Demo ref="demo"/>  
.....  
mounted(){  
  this.$refs.xxx.$on('atguigu',this.test)  
}
```
  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)`
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，回调要么配置在 `methods` 中，要么用箭头函数。

## 组件通信-全局事件总线\_01

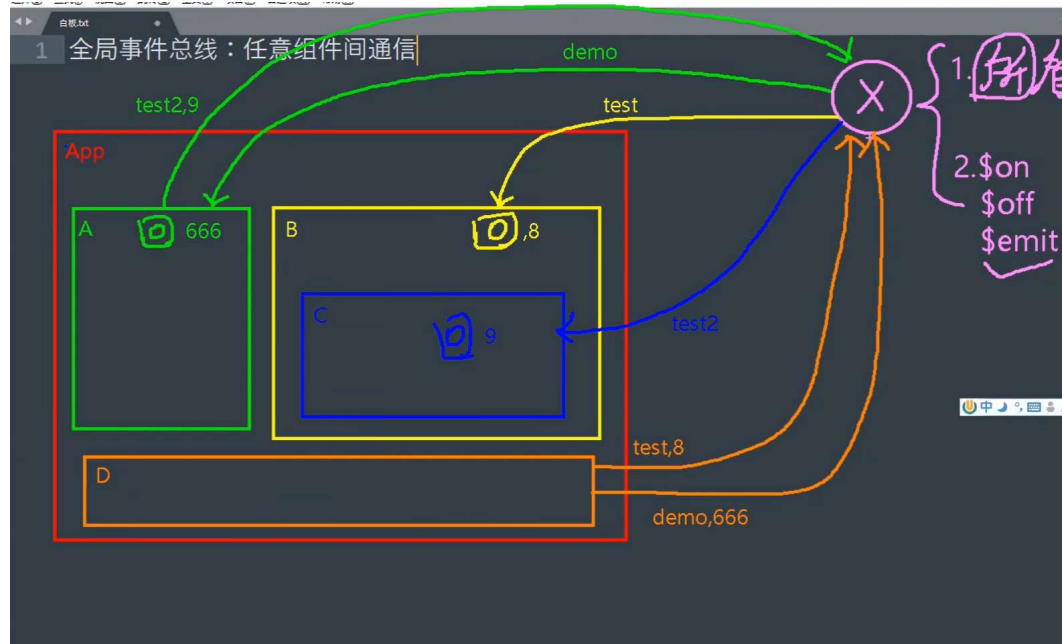
笔记本: Vue

创建时间: 2022/4/25 11:02

更新时间: 2022/4/25 11:20

作者: Chris吕

原理



这个紫色X不能放window，不能放vc(每次缔造都是一个全新的vc)，要放在Vue原型里面

## 组件通信-全局事件总线\_02

笔记本: Vue

创建时间: 2022/4/25 11:21

更新时间: 2022/4/25 14:10

作者: Chris吕

```
beforeCreate() {  
  Vue.prototype.$bus = this //安装全局事件总线  
},
```

在创建Vue的时候  
beforeCreate之前创建实例

在组件 vc

建议: 在vc销毁前解绑

```
beforeDestroy() {  
  this.$bus.$off('hello')  
},
```

小结

### 全局事件总线 (GlobalEventBus)

1. 一种组件间通信的方式, 适用于任意组件间通信。

2. 安装全局事件总线:

```
new Vue({  
  ....  
  beforeCreate() {  
    Vue.prototype.$bus = this //安装全局事件总线, $bus就是当前应用的vm  
  },  
  ....  
})
```

3. 使用事件总线:

1. 接收数据: A组件想接收数据, 则在A组件中给\$bus绑定自定义事件, 事件的回调留在A组件自身。

```
methods(){  
  demo(data){.....}  
}  
.....  
mounted() {  
  this.$bus.$on('xxxx',this.demo)  
}
```

2. 提供数据: `this.$bus.$emit('xxxx',数据)`

4. 最好在beforeDestroy钩子中, 用\$off去解绑当前组件所用到的事件。|

## 消息订阅与发布

笔记本: Vue

创建时间: 2022/4/25 14:22

更新时间: 2022/4/25 14:52

作者: Chris吕

---

1. 订阅消息: 消息名字
2. 发布消息: 消息内容
3. 取消订阅: 订阅id

安装库: `npm i pubsub-js`

引入 `import pubsub from 'pubsub-js'`

订阅:

```
mounted() {  
  // console.log('School',this)  
  /* this.$bus.$on('hello',(data)=>{  
    console.log('我是School组件, 收到了数据',data)  
  }) */  
  this.pubId = pubsub.subscribe('hello',function(msgName,data){  
    console.log('有人发布了hello消息, hello消息的回调执行了',msgName,data)  
  })  
}
```

发布:

```
sendStudentName(){  
  // this.$bus.$emit('hello',this.name)  
  pubsub.publish('hello',666)  
}
```

组件销毁前要取消订阅

```
beforeDestroy() {  
  // this.$bus.$off('hello')  
  pubsub.unsubscribe(this.pubId)  
},
```

## \$nextTick

笔记本: Vue

创建时间: 2022/4/25 15:22

更新时间: 2022/4/25 15:27

作者: Chris吕

---

# \$nextTick指定的回调会在 dom更新后再执行

## | nextTick

1. 语法: `this.$nextTick(回调函数)`
2. 作用: 在下次 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。

## 集成第三方动画

笔记本: Vue

创建时间: 2022/4/25 16:51

更新时间: 2022/4/25 17:33

作者: Chris吕

---

```
<transition-group  
  appear  
  name="animate__animated animate__bounce"  
  enter-active-class="animate__swing"  
  leave-active-class="animate__backOutUp"  
>
```

## 配置代理

笔记本: Vue

创建时间: 2022/4/25 17:44

更新时间: 2022/4/26 18:04

作者: Chris吕

```
✗ 1.xhr new XMLHttpRequest() xhr.open() xhr.send()
```

鼻祖

2.jQuery \$.get \$.post

3.axios

axios是jQuery的1/3

## 4. v-resource 插件 (返回一个 promise)

```
this.$http.get(`https://api.github.com/search/users?q=${this.keyword}`).then(
  response => {
    console.log('请求成功了')
    //请求成功后更新List的数据
    this.$bus.$emit('updateListData',{isLoading:false,errMsg:'',users:response.data.
  },
  error => {
    //请求后更新List的数据
    this.$bus.$emit('updateListData',{isLoading:false,errMsg:error.message,users:[]
  }
)
```

安装: npm i axios

导入:

```
import axios from 'axios'
```

发送请求不能跨域  
满足三个要求

1. 协议 (file/http/...)
2. 主机名
3. 端口号

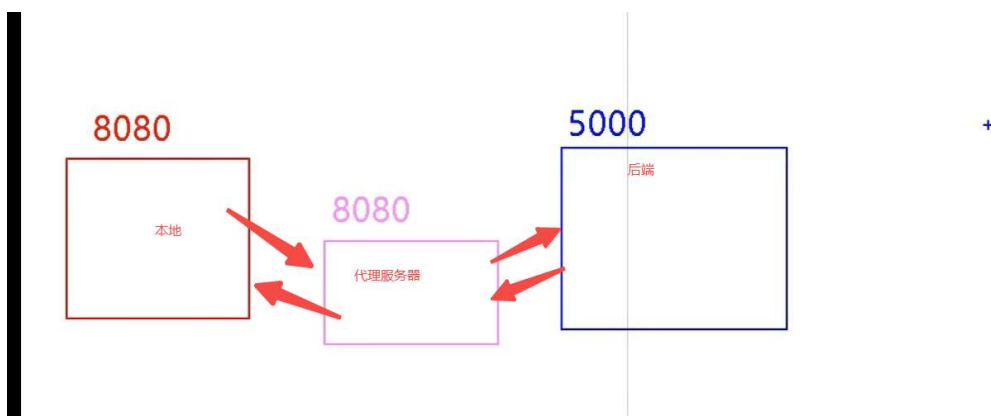
## 解决跨域问题

1. **cors**
2. **jsonp** 用script的src请求GET
3. **代理服务器**

## 代理服务器 开启

1. **Nginx**
2. **vue-cli**

```
vue.config.js > <unknown>
1  module.exports = {
2    pages: {
3      index: {
4        //入口
5        entry: 'src/main.js',
6      },
7    },
8    lintOnSave: false, //关闭语法检查
9    //开启代理服务器
10   devServer: {
11     proxy: 'http://localhost:5000/students'
12   }
13 }
```

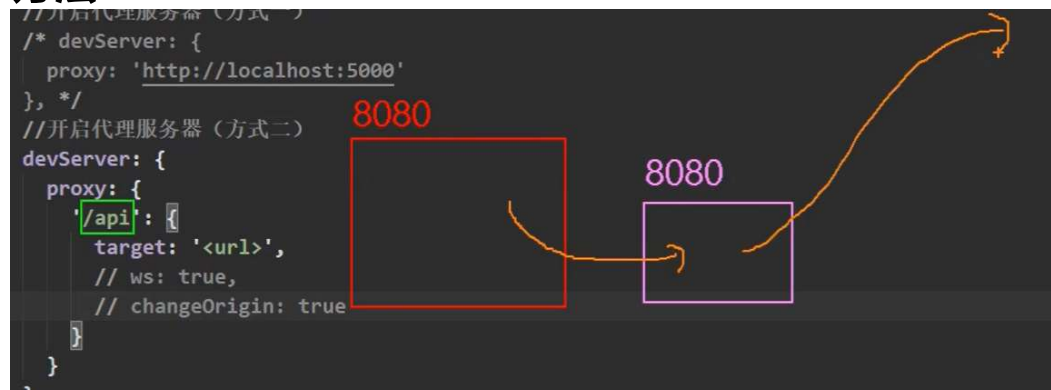


**注意点:**

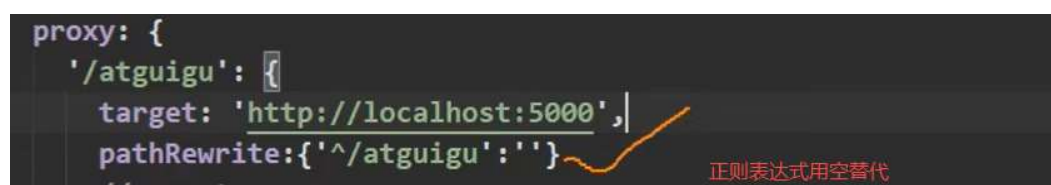


1. 代理服务器（8080）会默认把public文件里的内容放在里面，当请求时发现请求文件名与public的一样，会优先拿本地的
2. 代理服务器只能1个服务器

## 方法2



会先询问api



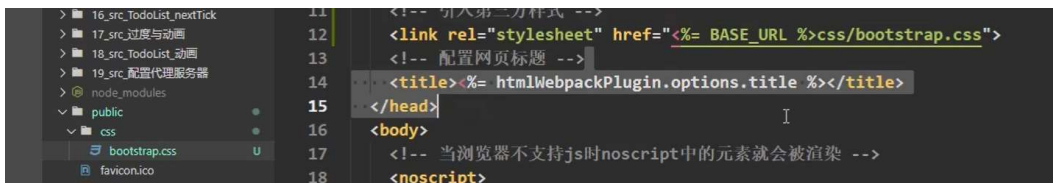
## 网络请求(动态组件)&&静态组件拆分

笔记本: Vue

创建时间: 2022/4/26 17:15

更新时间: 2022/4/26 17:42

作者: Chris吕



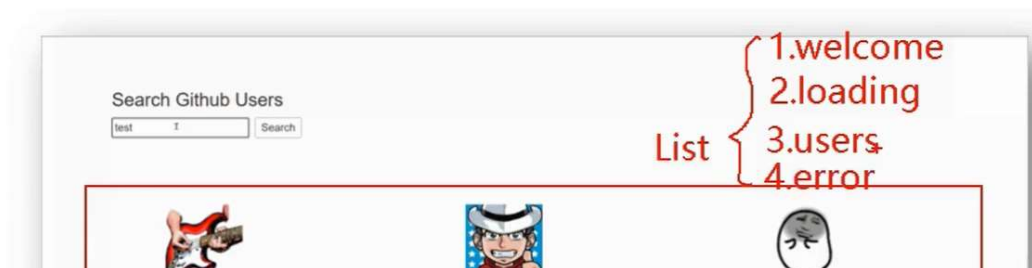
引入第三方库

axios

```
axios.get(`https://api.github.com/search/users?q=${this.keyWord}`).then(
  response => {
    console.log('请求成功了', response.data)
  },
  error => {
    console.log('请求失败了', error.message)
  }
)
```

动态展示

### 4.2.1 效果



## 默认插槽 && 具名插槽

笔记本: Vue

创建时间: 2022/4/26 18:07

更新时间: 2022/4/28 10:11

作者: Chris吕

默认插槽: 先挖着坑, 等你去填(支持单个标签)



需求, 对于相同的每个模块, 需要展示不同的元素, 则需要使用插槽语法.

模块

具名插槽

在APP.vue中

```
<Category title="电影">
  <video slot="center" controls src="..."/>
  <template slot="footer">
    <div class="foot">
      <a href="http://www.atguigu.com">...</a>
      <a href="http://www.atguigu.com">...</a>
      <a href="http://www.atguigu.com">...</a>
    </div>
    <h4>欢迎前来观影</h4>
  </template>
</Category>
```

在 组件中

onens / > category.vue / {} category.vue / > template :

```
<template>
  <div class="category">
    <h3>{{title}}分类</h3>
    <!-- 定义一个插槽（挖个坑，等着组件
    <slot name="center">我是一些默认值
    <slot name="footer">我是一些默认值
  </div>
```

## 作用域插槽

笔记本: Vue

创建时间: 2022/4/28 10:11

更新时间: 2022/4/28 10:24

作者: Chris吕

---

引言: app 是 category的使用者

数据game放在 category里面, 但是app要用数据game, 涉及到作用域的问题

### 用类似组件的做法 把game传回去

```
<slot :games="games">我是默认的一些内容</slot>
</div>
</template>

<script>
export default {
  name: 'Category',
  props: ['title'],
  data() {
    return {
      games: ['红色警戒', '穿越火线', '劲舞团', '超级玛丽'],
    }
  },
}
```

### 使用者app中, 必须在组件中用 template的学法

```
<Category title="游戏">
  <template scope="atguigu">
    <ul>
      <li v-for="(g,index) in
    </li>
    </ul>
  </template>
</Category>
```

## vuev简介

笔记本: Vue

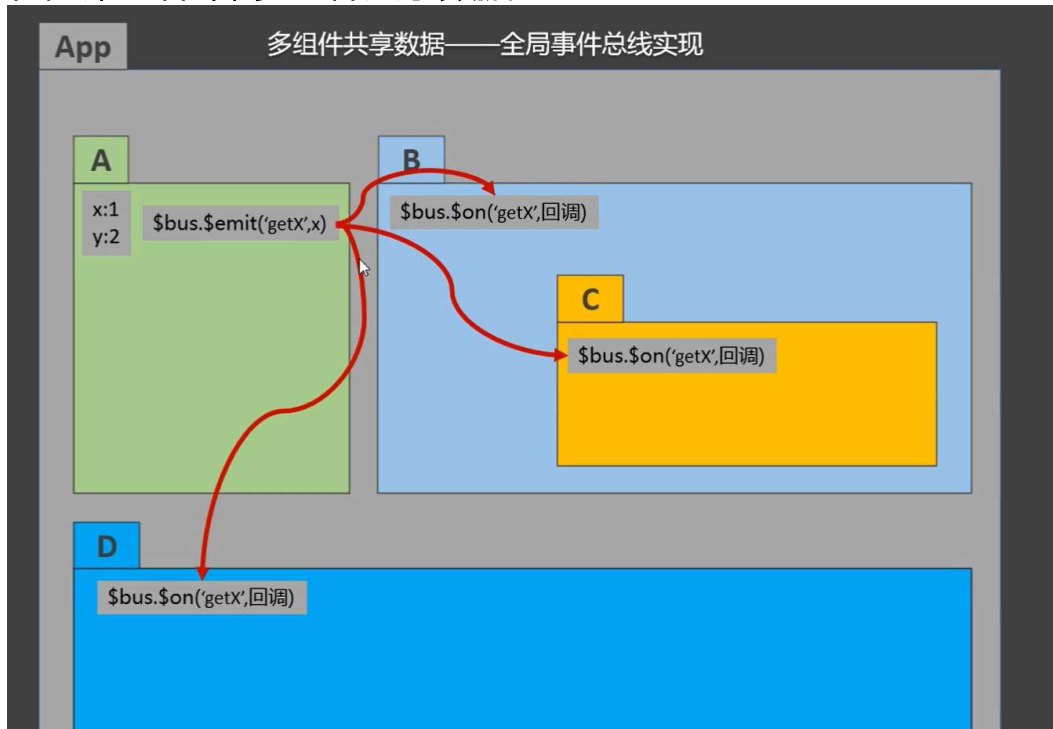
创建时间: 2022/4/28 10:30

更新时间: 2022/4/28 15:31

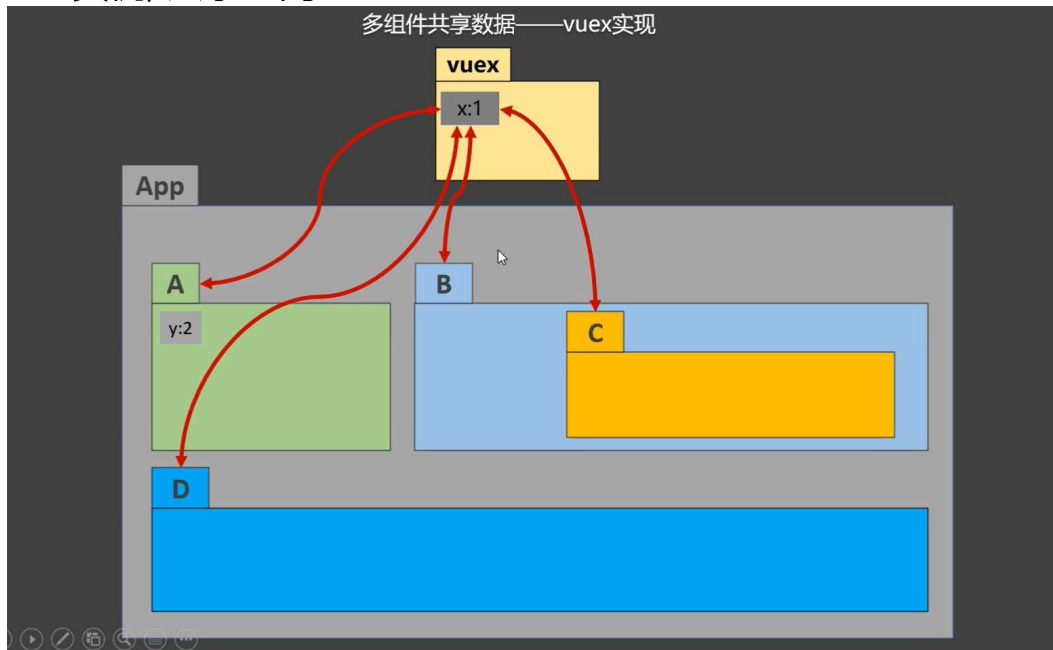
作者: Chris吕

## 集中式(与其相反的是分布式)状态(data)管理的一个vue插件

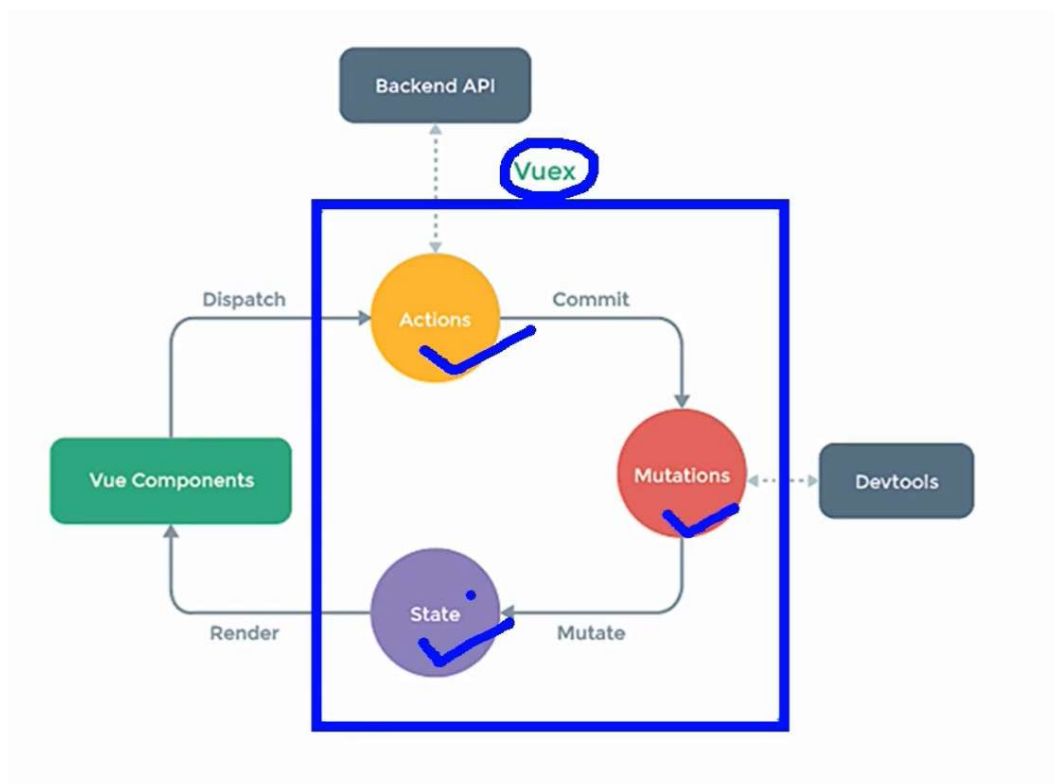
在兄弟组件中, 多组件共享数据



vue实例, 共享空间



vuex 的工作原理

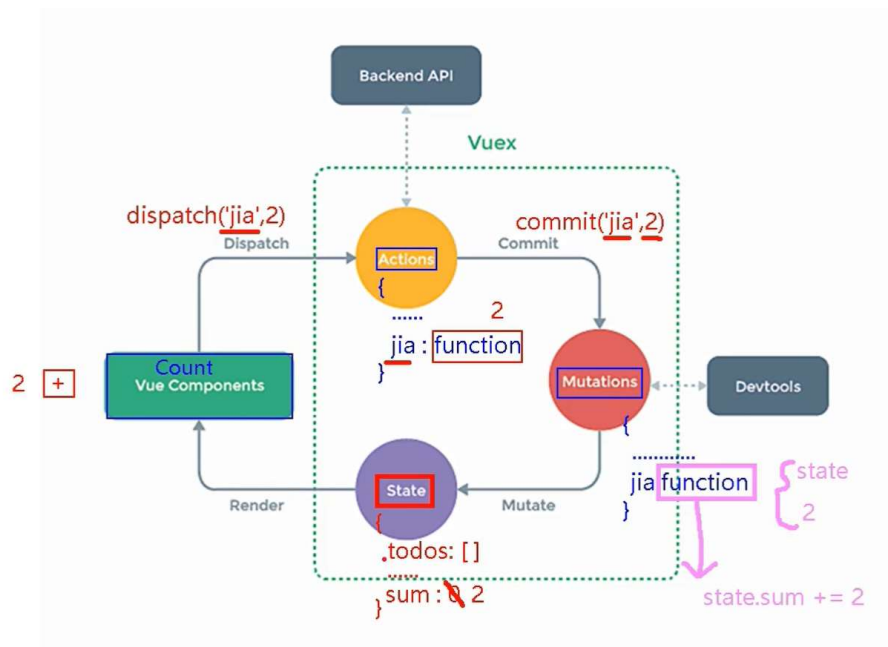


vuex的三个重要组成部分

Action: 动作行为(添加, 减少), 处理异步任务. 当请求中没有数据, 他可以向api请求

Mutations: 修改, 加工

State(对象): 状态, 数据



在没有额外的异步请求或者业务逻辑时, 可以跳过Action环节, 组件直接用commit

## vuex-配置

笔记本: Vue

创建时间: 2022/5/5 8:50

更新时间: 2022/5/5 13:59

作者: Chris吕

---

1. **npm i vuex@3**
2. **Vue.use(Vue)**
3. **store, 管理vuex**
4. **使所有的vc 都能看得见store**

引入后使用 store

```
12 Vue.use(vueResource)
13 Vue.use(Vuex)
14
15 //创建vm
16 const vm = new Vue({
17   el: '#app',
18   render: h => h(App),
19   store: 'hello',
20
21   beforeCreate() {
22     Vue.prototype.$bus = this
23   }
24 }
```

在src下面见到这个文件夹就等于见到了vuex



vuex的主文件一般为 index.js



```

src > store > JS index.js > ...
1  //该文件用于创建vuex 的核心管理
2
3  //先引入Vue 为了在vue中使用vuex
4  import Vue from 'vue';
5
6  //引入vuex 为了创建store实例
7  import Vuex from 'vuex';
8  Vue.use(Vuex);
9
10
11 //该对象用于响应 vuex 的 action 主要做逻辑性处理 或者是 api调用
12 const actions = {};
13
14 //真正帮助我们去操作数组的的对象
15 const mutations = {};
16
17 //用于储存/状态
18 const state = {};
19
20 //创建vuex实例，用法和vue差不多，在构建行数传入一个对象
21 //这里记得Store要大写
22 export default new Vuex.Store({
23
24     // actions:actions的简写
25     actions,
26     mutations,
27     state
28 })

```

## 脚手架中的执行顺序

```

main.js
1  import str1 from './test1'
2  console.log(100)
3  console.log(200)
4  import str2 from './test2'
5

```

脚手架执行顺序: 不管  
import放在哪里, 都是先执行import 再执行代码  
左边执行顺序如序

## vuex-example- 重要!

笔记本: Vue

创建时间: 2022/5/5 14:00

更新时间: 2022/5/6 15:33

作者: Chris吕

### action接到的对象也是一个 mini store 提供了 commit方法

```
3
1 //该对象用于响应 vuex 的 action 主要做逻辑性处理 或者是 api调用
2 const actions = {
3   addOdd(context, val){
4     if(context.state.sum % 2){
5       context.commit('ADD', val);
6     }
7   },
8   addWait(context, val){
9     setTimeout(function(){
10       context.commit('ADD', val);
11     },1000)
12   }
13 };
14
15 //真正帮助我们去操作数组的的对象
16 const mutations = {
17   ADD(state, val){
18     state.sum += val;
19   },
20   SUBSTRACT(state, val){
21     state.sum -= val
22   }
23 };
24
```

### mutation中接的 State 也是跟vue一样的进行数据监测原理

```
▼ {__ob__: Observer} ⓘ
  sum: 0
  ► __ob__: Observer {value: {...}, dep: Dep, vmCount: 0}
  ► get sum: f reactiveGetter()
  ► set sum: f reactiveSetter(newVal)
  ► __proto__: Object
```

### getters使用场景(需要写返回值)

当多个组件需要用同一个计算逻辑时, 使用getter时传的是state

```

4
5 //用于储存/状态
6 const state = {
7   sum:0
8 };
9
10 //配置getter
11 const getters = {
12   bigSum(state){
13     return state.sum*10;
14   }
15 }
16 //创建vuex实例，用法和vue差不多，在构建行数传入一个对象
17 //这里记得Store要大写
18 export default new Vuex.Store({
19
20   // actions:actions的简写
21   actions,
22   mutations,
23   state,
24   getters
25 })

```

## mapState 和 mapGetters

用法:将多个重复 this.\$store.state 用vuex里面的 mapState帮我们批量生成,

**对象写法:**

```

},
computed:{
  ...mapState({sum:'sum',school:'school',subject:'subject'})
}
}
</script>

```

一个对象包着, 前面的 sum是计算属性名, 'sum'是vuex里面 state的属性

## 数组写法

```

//数组写法
...mapState(['sum','school','subject']),

```

需要计算属性和state同名

## mapMutations & mapActions用法

```

@click="increment(n)">+

```

需要在html中传参

```

import { mapState, mapGetters, mapMutations, mapActions } from 'vuex'
export default {
  data() {
    return {
      number:1,
    }
  },
  methods: {
    // increment(){
    //   this.$store.commit('ADD',this.number)
    // },
    // decrement(){
    //   this.$store.commit('SUBSTRACT',this.number)
    // },
    ...mapMutations({increment:'ADD', decrement:'SUBSTRACT'}),
    // incrementOdd(){
    //   this.$store.dispatch('addOdd',this.number)
    // },
    // incrementWait(){
    //   this.$store.dispatch('addWait',this.number)
    // }
    ...mapActions([incrementOdd:'addOdd', incrementWait:'addWait'])
  },
  computed: {

```

数组方法同上, 需要保证同名

## 多组件共用数据

数据都放在state里面, 然后在用的时候导入就可以了

```

computed:{
  ...mapState(['personList','sum'])
},

```

## 模块化配合命名空间

按功能分成几套配置

```

7
8 //求和相关的配置
9 const countOptions = {
10 >   actions:{ ...
13     },
14 >   mutations:{ ...
13     },
14 >   state:{ ...
18     },
19 >   getters:{[ ...
13     ]},
14 }
15
16 //人员管理相关的配置
17 const personOptions = {
18   actions:{},
19 >   mutations:{ ...
14     },
15 >   state:{ ...
19     },
20   getters:{},
21 }
22

```

在引入时用 modules

```
4 import Vuex from 'vuex'
5 //应用Vuex插件
6 Vue.use(Vuex)
7
8 //求和相关的配置
9 > const countOptions = { ...
44 }
45
46 //人员管理相关的配置
47 > const personOptions = { ...
61 }
62
63
64 //创建并暴露store
65 export default new Vuex.Store({
66   modules:{
67     a:countOptions,
68     b:personOptions
69   }
70 })
```



在模块中使用命名空间, 让组件在使用mapState, mapMutations...的时候可以用索引来增加

```
//求和相关的配置
const countOptions = {
  namespaced:true,
  actions:{
    jiaOdd(context,value){ ...
  },
    jiaWait(context,value){ ...
  }
},
  mutations:{ ...
},
  state:{ ...
},
  getters:{ ...
},
}
```

其后可以把各模块分离出独立的js文件

## 路由

笔记本: Vue

创建时间: 2022/5/13 15:57

更新时间: 2022/5/13 18:34

作者: Chris吕

```
{
  path: '/home',
  component: Home,
  children: [
    {
      path: 'news',
      component: News,
    },
    {
      path: 'message',
      component: Message
    }
  ]
}
```

二级路由里面不要写 /

### 路由的props配置

作用: 让路由组件更方便的收到参数

```
{
  name: 'xiangqing',
  path: 'detail/:id',
  component: Detail,

  //第一种写法: props值为对象, 该对象中所有的key-value的组合最终都会通过props传给Detail组件
  // props:{a:900}

  //第二种写法: props值为布尔值, 布尔值为true, 则把路由收到的所有params参数通过props传给Detail组件
  // props:true

  //第三种写法: props值为函数, 该函数返回的对象中每一组key-value都会通过props传给Detail组件
  props(route){
    return {
      id:route.query.id,
      title:route.query.title
    }
  }
}
```

### <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, `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>`

## 路由的导航



```

//$router的两个API
this.$router.push({
  name: 'xiangqing',
  params: {
    id: xxx,
    title: xxx
  }
})

this.$router.replace({
  name: 'xiangqing',
  params: {
    id: xxx,
    title: xxx
  }
})

this.$router.forward() //前进
this.$router.back() //后退
this.$router.go() //可前进也可后退

```

push是可前进后退的 stack原理

replace是直接替代不可前进后退

go传正数是前进, 传负数是后退

```

</ul>
<keep-alive include="News">
  <router-view></router-view>
</keep-alive>
</div>

```

include缓存组件 写 组件名 千万别写路由名



## I.两个新的生命周期钩子

1. 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：
  1. `activated` 路由组件被激活时触发。
  2. `deactivated` 路由组件失活时触发。

## 路由守卫

笔记本: Vue

创建时间: 2022/5/13 18:34

更新时间: 2022/5/25 9:27

作者: Chris吕

---

Learning in work