

hello Vue

```
4   <meta charset="UTF-8" />
5   <title>初识Vue</title>
6   <!-- 引入Vue -->
7   <script type="text/javascript" src="../js/vue.js"></script>
8   </head>
9   <body>
10  </body>
11  </html>
```

Vue

```
</head>
<body>
    <!-- 准备好一个容器 -->
    <div id="root">
        <h1>Hello, {{name}}</h1>
    </div>

<script type="text/javascript" >
    Vue.config.productionTip = false //阻

    //创建Vue实例
    const x = new Vue({
        el: '#root', //el用于指定当前Vue实例
        data: {
            name: '尚硅谷'
        }
    })

```

```
9 <body>
10 <!--
11 初识Vue:
12     1.想让Vue工作，就必须创建一个Vue实例，且要传入一个配置对象；  

13     2.root容器里的代码依然符合html规范，只不过混入了一些特殊的Vue语法；  

14     3.root容器里的代码被称为【Vue模板】；  

15 -->
16
```

好一个容器 -->

```
'root">
11o, {{name}}, {{address}}, {{Date.now()}}</h1>
```

4.Vue实例和容器是一对应的；

5.真实开发中只有一个Vue实例，并且会配合着组件一起使用；

6.{{xxx}}中的xxx要写js表达式，且xxx可以自动读取到data中的所有属性；

7.一旦data中的数据发生改变，那么模板中用到该数据的地方也会自动更新；

指令语法V-bind

```
<!-- 准备好一个容器-->
<div id="root">
  <h1>插值语法</h1>
  <h3>你好, {{name}}</h3>
  <hr/>
  <h1>指令语法</h1>
  <a v-bind:href="url">点我去尚硅谷学习1</a>
</div>
</body>

<script type="text/javascript">
  Vue.config.productionTip = false //阻止 vue 在启动时

  new Vue({
    el: '#root',
    data: {
      name: 'jack',
      url: 'http://www.atguigu.com' // 直接写字符串
    }
  })
</script>
<a v-bind:href="url" x="hello">点我去尚硅谷学习1</a>
</div>
```

V-bind的简写

v-bind: ===> :

```
<!--  
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。  
  
24   <h1>插值语法</h1>  
25   <h3>你好, {{name}}</h3>  
26   <hr/>  
27   <h1>指令语法</h1>  
28   <a v-bind:href="school.url.toUpperCase()" x="hello">点我去{{school.name}}学习1</a>  
29   <!-- <a :href="url" x="hello">点我去{{name2}}学习2</a> -->  
30   </div>  
31 </body>  
32  
33 <script type="text/javascript">  
34     Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。  
35  
36     new Vue({  
37       el: '#root',  
38       data:{  
39         name:'jack',  
40         school:{  
41           name:'尚硅谷',  
42           url:'http://www.atguigu.com',  
43         }  
44     })  
45   </script>
```

英 音 画

数据绑定

v-bind是单向绑定

v-model双向绑定

```
<div id="root">
    单向数据绑定: <input type="text" v-bind:value="name"><br/>
    双向数据绑定: <input type="text" v-model:value="name"><br/>

    <!-- 如下代码是错误的, 因为v-model只能应用在表单类元素(输入类元素)上 -->
    <h2 v-model:x="name">你好啊</h2>
</div>
</body>

<script type="text/javascript">
    Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。

    new Vue({
        el: '#root',
        data: {
            name: '尚硅谷'
        }
    })

```

简写方式

备注:

1. 双向绑定一般都应用在表单类元素上(如: `input`、`select`等)
2. `v-model:value` 可以简写为 `v-model`, 因为`v-model`默认收集的就是`value`值。

```
-->
<!-- 准备好一个容器-->
<div id="root">
    <!-- 普通写法 -->
    <!-- 单向数据绑定: <input type="text" v-bind:value="name"><br/>
    双向数据绑定: <input type="text" v-model:value="name"><br/> -->

    <!-- 简写 -->
    单向数据绑定: <input type="text" :value="name"><br/>
    双向数据绑定: <input type="text" v-model="name"><br/>

    <!-- 如下代码是错误的, 因为v-model只能应用在表单类元素(输入类元素)上 -->
    <h2 v-model:x="name">你好啊</h2>
</div>
</body>
```

el和data的两种语法

```
const v = new Vue({
  //el:'#root',
  data:{
    name:'尚硅谷'
  }
})
console.log(v)
v.$mount('#root')
```

这种方式比较灵活

```
//data的两种写法
new Vue({
  el:'#root',
  //data的第一种写法：对象式
  /* data:{
    name:'尚硅谷'
  } */
```

```
//data的第二种写法：函数式
```

```
data:function(){
  return{
    name:'尚硅谷'
  }
}
```

简写方式

```
//data的第二种写法：函数式
data(){
  console.log('@@@',this) //此处的this是Vue实例对象
  return{
    name:'尚硅谷'
  }
}
```

组件都用这种方式

箭头函数没有自己的this

```
<!--
  data与el的2种写法
  1.el有2种写法
    (1).new Vue时候配置el属性。
    (2).先创建Vue实例，随后再通过vm.$mount('#root')指定el的值。
  2.data有2种写法
    (1).对象式
    (2).函数式
    如何选择：目前哪种写法都可以，以后学习到组件时，data必须使用函数式，否则会报错。
  3.一个重要的原则：由Vue管理的函数，一定不要写箭头函数，一旦写了箭头函数，this就不再是Vue实例了。
-->
<!-- 准备好一个容器-->
```

原文地址: <https://www.cnblogs.com/hailun/p/6279029.html>

ES6标准新增了一种新的函数: Arrow Function (箭头函数)

```
x => x * x
```

相当于:

```
function (x) {
  return x * x;
}
```

 // 两个参数:
(x, y) => x * x + y * y

理解MVVM模型

```
19     <!-- 准备好一个容器-->
20     <div id="root">
21         <h1>学校名称: {{name}}</h1>
22         <h1>学校地址: {{address}}</h1>
23         <h1>测试一下1: {{1+1}}</h1>
24         <h1>测试一下2: {{$options}}</h1>
25         <h1>测试一下3: {{$emit}}</h1>
26     </div>
27 </body>
28
29 <script type="text/javascript">
30     Vue.config.productionTip = false //阻止 vue 在启动时生成生产提
31
32     const vm = new Vue({
33         el: '#root',
34         data: {
35             name: '尚硅谷',
36             address: '北京',
37             a: 1
38         }
39     })

```

原型上有的东西可以直接在模板上使用

数据代理

```
<script type="text/javascript" >
    let person = {
        name: '张三',
        sex: '男',
    }
    // 给对象添加属性
    Object.defineProperty(person, 'age', {
        value: 18
    })

```

```
}

Object.defineProperty(person, 'age', {
    value:18,
    enumerable:true, | I
})                                原本的方式是不参加遍历的，但是增加了enumerable:true 以后
                                    就可以遍历了

13 }
14
15 Object.defineProperty(person,'age',{
16     value:18,
17     enumerable:true, //控制属性是否可以枚举，默认值是false
18     writable:true, //控制属性是否可以被修改，默认值是false
19     configurable:true //控制属性是否可以被删除，默认值是false
20 })
21

<script type="text/javascript" >
let number = 18
let person = {
    name:'张三',
    sex:'男',
}
Object.defineProperty(person,'age',{
    // value:18,
    // enumerable:true, //控制属性是否可以枚举，默认值是false
    // writable:true, //控制属性是否可以被修改，默认值是false
    // configurable:true //控制属性是否可以被删除，默认值是false

    //当有人读取person的age属性时，get函数(getter)就会被调用，且返回
    get(){
        console.log('有人读取age属性了')
        return number
    },
}

//当有人修改person的age属性时，set函数(setter)就会被调用，且会收
set(value){
    console.log('有人修改了age属性，且值是',value)
    number = value
}

})
```

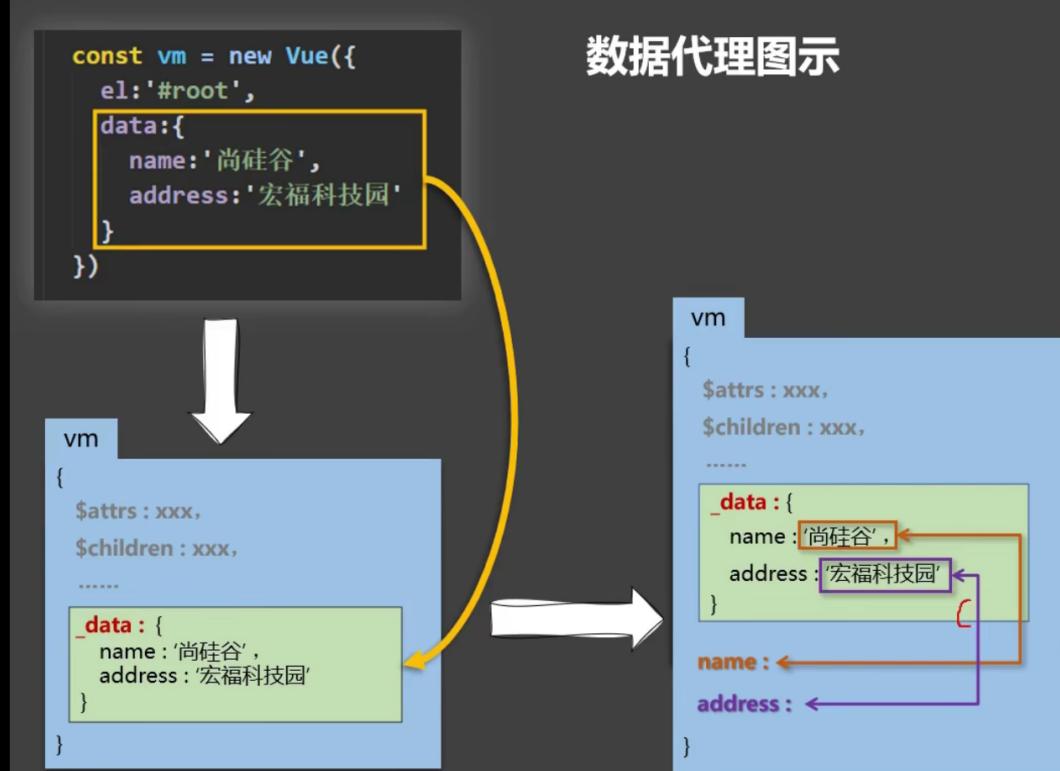
```

6  </head>
7  <body>
8      <!-- 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）-->
9
10 </body>
11 </html>
<body>
    <!-- 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）-->
    <script type="text/javascript" >
        let obj = {x:100}
        let obj2 = {y:200}

        Object.defineProperty(obj2, 'x',{
            get(){
                return obj.x
            },
            set(value){
                obj.x = value
            }
        })
    </script>

```

数据代理图示



这里的`_data`=上面的`data`, 而数据代理就是做了一个操作: 将`_data`中的数据给了`vm`, `vm`就代理了其中的`name`和`address`

事件处理

```
9   <body>
10  >    <!-- ...
18      <!-- 准备好一个容器-->
19      <div id="root">
20          <h2>欢迎来到{{name}}学习</h2>
21          <button v-on:click="showInfo">点我提示信息</button>
22      </div>
23  </body>
24
25  <script type="text/javascript">
26      Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。
27
28      new Vue({
29          el:'#root',
30          data:{
31              name:'尚硅谷'
32          },
33          methods:{
34              showInfo(){
35                  +
36                  alert('同学你好')
37              }
38          })
39      </script>
40  </html>
```

简写方式

```
<!-- <button v-on:click="showInfo">点我提示信息</button> -->
<button @click="showInfo1">点我提示信息1</button>
<button @click="showInfo">点我提示信息</button>
</div>
```

```
<n>欢迎来到{{name}}子组件</n>
<button v-on:click="showInfo">点我提示信息</button>
</div>
</body>

<script type="text/javascript">
Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。

new Vue({
  el:'#root',
  data:{
    name:'尚硅谷'
  },
  methods:{
    showInfo(event){
      console.log(event.target.innerText)
      // alert('同学你好')
    }
  }
})
</script>
</html>
```

这里可以拿到上面的内容：点我提示信息

```
<button @click="showInfo1">点我提示信息1</button>
<button @click="showInfo2(66)">点我提示信息2</button>
</div>
</body>
```

上面的66通过参数传到了下面

```
<script type="text/javascript">
Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。

const vm = new Vue({
  el:'#root',
  data:{
    name:'尚硅谷'
  },
  methods:{
    showInfo1(event){ ... },
    showInfo2(number){
      console.log(number)
      // console.log(event.target.innerText)
      // console.log(this) //此处的this是vm
      // alert('同学你好！！')
    }
  }
})
```

上面这种方式会导致event丢失

```
<button @click="showInfo2(66,$event)">点我提示信息2</button>
</div>
</body>

<script type="text/javascript">
Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。

const vm = new Vue({
  el:'#root',
  data:{
    name:'尚硅谷'
  },
  methods:{
    showInfo1(event){ ... },
    showInfo2(number,a){
      console.log(number,a)
      // console.log(event.target.innerText)
      // console.log(this) //此处的this是vm
      // alert('同学你好！！')
    }
  }
})

```

这里的\$event 作为占位符保证可以将event传给下面的a

事件的基本使用：

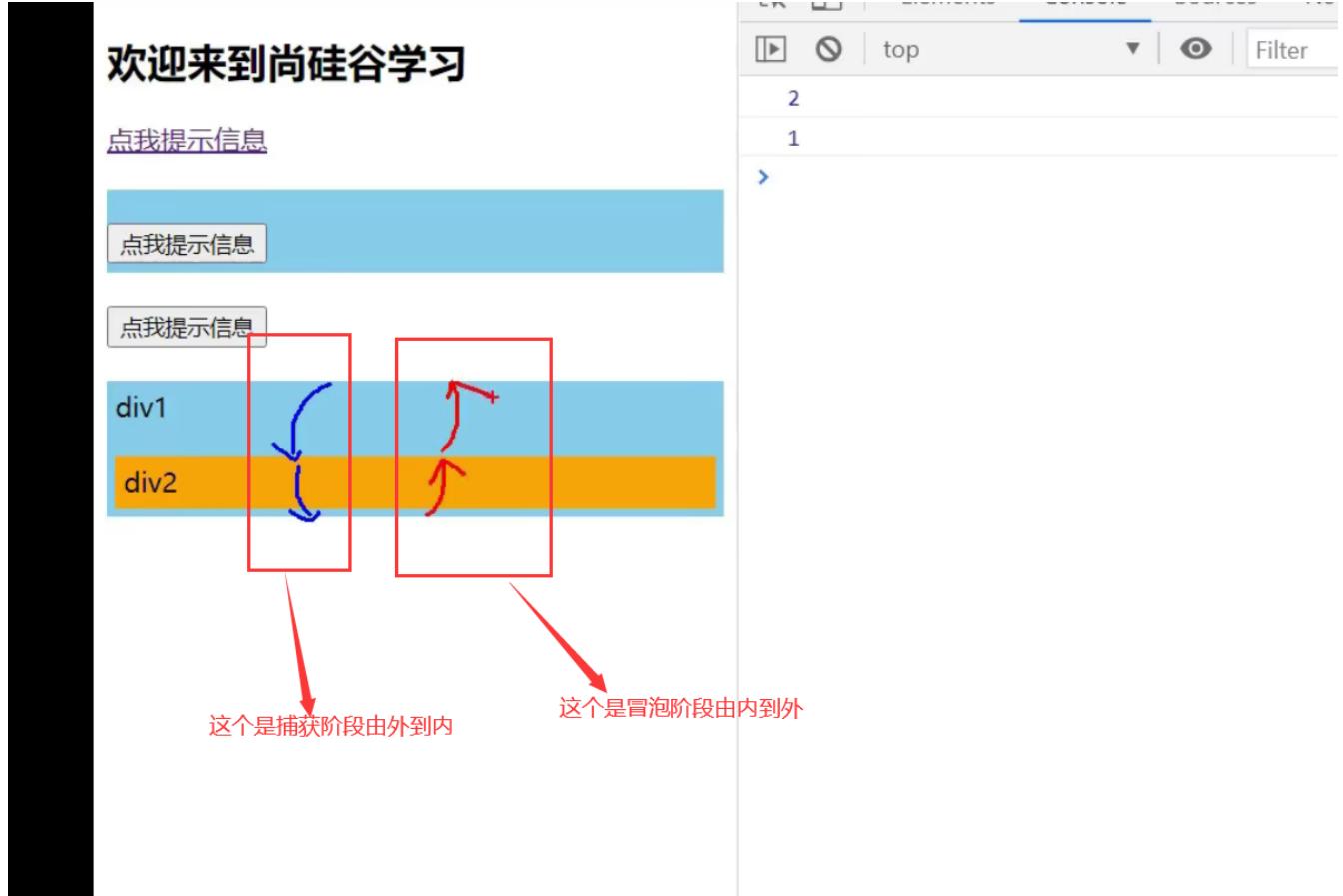
1. 使用v-on:xxx 或 @xxx 绑定事件，其中xxx是事件名；
2. 事件的回调需要配置在methods对象中，最终会在vm上；
- 3.methods中配置的函数，不要用箭头函数！否则this就不是vm了；
- 4.methods中配置的函数，都是被Vue所管理的函数，this的指向是vm 或 组件实例对象；
- 5.@click="demo" 和 @click="demo(\$event)" 效果一致，但后者可以传参；

-->

```
20     <div id="root">
21         <h2>欢迎来到{{name}}学习</h2>
22         <a href="http://www.atguigu.com" @click="showInfo">点我提示信息</a>
23     </div>
24 </body>
25
26 <script type="text/javascript">
27     Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。
28
29     new Vue({
30         el:'#root',
31         data:{
32             name:'尚硅谷'
33         },
34         methods:{
35             showInfo(e){
36                 e.preventDefault() //这个阻止模式事件及上面的网址的跳转
37                 alert('同学你好！')
38             }
39         }
40     })
41 </h2>从这里看不到{{name}}子句</h2>
42     <a href="http://www.atguigu.com" @click.prevent="showInfo">点我提示信息</a>
43 </div>
44 </body>
45
46 <script type="text/javascript">
47     Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。
48
49     new Vue({
50         el:'#root',
51         data:{
52             name:'尚硅谷'
53         },
54         methods:{
55             showInfo(e){
56                 alert('同学你好！')
57             }
58         }
59     })
60 }
```

```
</head>
<body>
  <!-- |
    Vue中的事件修饰符:
      1.prevent: 阻止默认事件（常用）;
      2.stop: 阻止事件冒泡（常用）;
      3.once: 事件只触发一次（常用）;
      4.capture: 使用事件的捕获模式;
      5.self: 只有event.target是当前操作的元素是才触发事件;
      6.passive: 事件的默认行为立即执行，无需等待事件回调执行完毕;
  -->
  <!-- 准备好一个容器-->
  <div id="root">
    <h2>欢迎来到{{name}}学习</h2>
    <a href="http://www.atguigu.com" @click.prevent="showInfo">点我提示信息</a>
  </div>
</body>

<script type="text/javascript">
  Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。
  new Vue({
    el: '#root',
    data: {
```



键盘事件

```
10 >     <!-- ... -->
32     <!-- 准备好一个容器-->
33     <div id="root">
34         <h2>欢迎来到{{name}}学习</h2>
35         <input type="text" placeholder="按下回车提示输入" @keyup.enter="showInfo">
36     </div>
37 </body>
38
39 <script type="text/javascript">
40     Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。
41
42     new Vue({
43         el:'#root',
44         data:{
45             name:'尚硅谷'
46         },
47         methods: [
48             showInfo(e){
49                 console.log(e.target.value)
50             }
51         ],
52     })
53 </script>
7       <script type="text/javascript" src=" ../../js/vue.js" ></script>
8   </head>
9   <body>
10      <!-- |
11          1.Vue中常用的按键别名:
12              回车 => enter
13              删 除 => delete (捕获“删除”和“退格”键)
14              退 出 => esc
15              空 格 => space
16              换 行 => tab
17              上    => up
18              下    => down
19              左    => left
20              右    => right
21
22          2.Vue未提供别名的按键，可以使用按键原始的key值去绑定，但
23
24          3.系统修饰键（用法特殊）: ctrl、alt、shift、meta
-->
<!-- 准备好一个容器-->
<div id="root">
    <h2>欢迎来到{{name}}学习</h2>
    <input type="text" placeholder="按下回车提示输入" @keyup.caps-lock="showInfo">
</div>
</body>
<script type="text/javascript">
    Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。

```

当按下enter键并起来时会触发

当需要按下^{按住}CAPSLOCK这个键，这个键比较特殊
是由两个单词组成，需要使用“-”来连接

```
-->
```

```
e}}学习</h2>
t" placeholder="按下回车提示输入" @keydown.tab="showInfo">
```

tab比较特殊需要和keydown配合使用

```
avascript">
```

```
22      2.Vue未提供别名的按键，可以使用按键原始的key值去绑定，但注意要转为kebab-case（短横线命名）
23
24      3.系统修饰键（用法特殊）：ctrl、alt、shift、meta
25          (1).配合keyup使用：按下修饰键的同时，再按下其他键，随后释放其他键，事件才被触发。
26          (2).配合keydown使用：正常触发事件。
27
28      4.也可以使用keyCode去指定具体的按键（不推荐）
29
30      5.Vue.config.keyCode.自定义键名 = 键码，可以去定制按键别名
31  -->
40      Vue.config.productionTip = false //禁用 vue 提交的生产环境提示
41  Vue.config.keyCode.huiche = 13 //定义了一个别名按键
42
43  new Vue({
    <!-- <button @click.stop="showInfo">点我提示信息</button> -->
    <a href="http://www.atguigu.com" @click.stop.prevent="showInfo">点我提示信息</a>
    </div>                                先阻止冒泡再阻止默认事件

```



```
<h2>欢迎来到{{name}}学习</h2>
<input type="text" placeholder="按下回车提示输入" @keyup.ctrl.y="showInfo">
    按下Ctrl+y的时候提示
</div>
```

计算属性

```
3  <head>
4      <meta charset="UTF-8" />
5      <title>姓名案例_插值语法实现</title>
6      <!-- 引入Vue -->
7      <script type="text/javascript" src="../js/vue.js"></script>
8  </head>
9  <body>
10     <!-- 准备好一个容器-->
11     <div id="root">
12         姓: <input type="text" v-model="firstName"> <br/><br/>
13         名: <input type="text" v-model="lastName"> <br/><br/>
14         全名: <span>{{firstName}}-{{lastName}}</span>
15     </div>
16 </body>
17
18 <script type="text/javascript">
19     Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。
20
21     new Vue({
22         el:'#root',
23         data:{
24             firstName:'张',
25             lastName:'三'
26         }
27     })
28 </script>
```

```

<!-- 准备好一个容器-->
<div id="root">
    姓: <input type="text" v-model="firstName"> <br/><br/>
    名: <input type="text" v-model="lastName"> <br/><br/>
    全名: <span>{{fullName()}}</span>
</div>
</body>

<script type="text/javascript">
Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。

```

这里的括号一定要，因为这里不是触发事件的回调函数而是直接调用函数。

```

new Vue({
    el:'#root',
    data:{
        firstName:'张',
        lastName:'三'
    },
    methods: {
        fullName(){
            return this.firstName + '-' + this.lastName
        }
    },
})

```

全名: {{fullName}}

全名: {{fullName}}

```

</div>
</body>

<script type="text/javascript">
Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。

```

```

const vm = new Vue({
    el:'#root',
    data:{
        firstName:'张',
        lastName:'三'
    },
    computed:{ // 计算属性
        fullName:{
            //get有什么作用? 当有人读取fullName时, get就会被调用, 且返回值就作为fullName
            get(){
                console.log('get被调用了')
                // console.log(this) //此处的this是vm
                return this.firstName + '-' + this.lastName
            }
        }
    }
})

```

```
computed:{  
    fullName:{  
        //get有什么作用? 当有人读取fullName时, get就会被调用, 且返回值就作为fullName的值  
        //get什么时候调用? 1. 初次读取fullName时。2. 所依赖的数据发生变化时。  
        get(){  
            console.log('get被调用了')  
            // console.log(this) //此处的this是vm  
            return this.firstName + '-' + this.lastName  
        }  
    }  
},  
//get什么时候调用? 当fullName被修改时。  
set(value){  
    console.log('set', value)  
    const arr = value  
}  
25     lastName:'三',  
26 },  
27     computed:{  
28         //完整写法  
29 >     /* fullName:{ ... */  
41         //简写  
42     fullName(){  
43         console.log('get被调用了')  
44         return this.firstName + '-' + this.lastName  
45     }  
46 }
```

简写方式调用时括号不用

监视属性

```
<div id="root">
  <h2>今天天气很{{info}}</h2>
  <button @click="changeWeather">切换天气</button>
</div>
</body>

<script type="text/javascript">
Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示

new Vue({
  el:'#root',
  data:{
    isHot:true
  },
  computed:{
    info(){
      return this.isHot ? '炎热' : '凉爽'
    }
  },
  methods: {
    changeWeather(){
      this.isHot = !this.isHot
    }
  }
})
```

```
监视属性 > 2.天气案例_监视属性.html > html > script > vm > watch > isHot > handler

9 <body>
10    <!-- 准备好一个容器-->
11    <div id="root">
12        <h2>今天天气很{{info}}</h2>
13        <button @click="changeWeather">切换天气</button>
14    </div>
15 </body>
16
17 <script type="text/javascript">
18     Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。
19
20     const vm = new Vue({
21         el:'#root',
22         data:{
23             isHot:true,
24         },
25         computed:{ ... },
26         methods: { ... },
27         watch:{}
28             isHot:{
29                 //handler什么时候调用? 当isHot发生改变时。
30                 handler(newValue,oldValue){
31                     console.log('isHot被修改了',newValue,oldValue)
32                 }
33             }
34         })
35
36     watch:{}
37         isHot:{
38             immediate:true, //初始化时让handler调用一下
39                 //handler什么时候调用? 当isHot发生改变时。
40                 handler(newValue,oldValue){
41                     console.log('isHot被修改了',newValue,oldValue)
42                 }
43             }
44
45         }
46     }
47 }
```

```
5
6     vm.$watch('isHot', {
7         immediate:true, //初始化时让handler调用一下
8         //handler什么时候调用? 当isHot发生改变时。
9         handler(newValue,oldValue){
10             console.log('isHot被修改了',newValue,oldValue)
11         }
12     })
13 
```

以上这个是监视的另一种写法

深度监视

```
09_监视属性 > 3.大气案例_深度监视.html > html > script > vm > watch > 'numbers.a'
18     </body>
19
20     <script type="text/javascript">
21         Vue.config.productionTip = false //阻止 vue 在启动时生成生产
22
23         const vm = new Vue({
24             el:'#root',
25             data:{
26                 isHot:true,
27                 numbers:{
28                     a:1,
29                     b:1
30                 }
31             },
32             computed:{ ... },
33             methods: { ... },
34             watch:{
35                 isHot:{ ... },
36                 ['numbers.a']:{
37                     handler(){
38                         console.log('a被改变了')
39                     }
40                 }
41             }
42         })
43     </script>
```

深度监视，监视的是a，注意需要引号括起来

```
29         isHot: true,
30         numbers:{  
31             a:1,  
32             b:1  
33         }  
34     },  
35     computed:{ ...  
36     methods: { ...  
37     watch:{  
38         isHot:{ ...  
39             //监视多级结构中某个属性的变化  
40             /* 'numbers.a':{  
41                 handler(){  
42                     console.log('a被改变了')  
43                 }  
44             } */  
45             //监视多级结构中所有属性的变化  
46             numbers:{  
47                 deep:true,  
48                 handler(){  
49                     console.log('numbers改变了')  
50                 }  
51             }  
52         }  
53     }  
54 }
```

<body>

<!--

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（多层次）。

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。

-->

 谁没有一个空里、

简写

```
3     }, */
4     //简写
5     /* isHot(newValue, oldValue){
6         console.log('isHot被修改了', newValue, oldValue)
7     } */
8 }
9 
```

简写的三个方式，简写的代价是不能有以下两种配

```
1 //正常写法
2 /* vm.$watch('isHot',{
3     immediate:true, //初始化时让handler调用一下
4     deep:true, //深度监视
5     handler(newValue, oldValue){
6         console.log('isHot被修改了', newValue, oldValue)
7     }
8 }) */
9 
```

```
10 (3) vm.$watch(['isHot'],function(newValue, oldValue){
11     console.log('isHot被修改了', newValue, oldValue)
12 })
```

计算属性和监视属性的对比

```

0     <!-- 准备好一个容器--> 
1     <div id="root">
2         姓: <input type="text" v-model="firstName"> <br/><br/>
3         名: <input type="text" v-model="lastName"> <br/><br/>
4         全名: <span>{{fullName}}</span> <br/><br/>
5     </div>
6 </body>
7
8 <script type="text/javascript">
9     Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。
0
1     const vm = new Vue({
2         el: '#root',
3         data: {
4             firstName: '张',
5             lastName: '三',
6             fullName: '张-三'
7         },
8         watch: {
9             firstName(val){
10                 this.fullName = val + '-' + this.lastName
11             },
12             lastName(val){
13                 this.fullName = this.firstName + '-' + val
14             }
15         }
16     })
17     //输出: 全名: 张-三
18
19     //输出: 全名: 张-三
20     //输出: 全名: 张-三
21     //输出: 全名: 张-三
22     //输出: 全名: 张-三
23     //输出: 全名: 张-三
24     //输出: 全名: 张-三
25     //输出: 全名: 张-三
26     //输出: 全名: 张-三
27     //输出: 全名: 张-三
28     //输出: 全名: 张-三
29     //输出: 全名: 张-三
30     //输出: 全名: 张-三
31     //输出: 全名: 张-三
32     //输出: 全名: 张-三
33     //输出: 全名: 张-三
34     //输出: 全名: 张-三
35     //输出: 全名: 张-三
36 </body>
<!-- |
    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。
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), 最好写成箭头函数,
       这样this的指向才是vm 或 组件实例对象。
-->
<!-- 准备好一个容器-->

```

watch可以开启异步任务让其等
以下在执行

样式绑定

```
35      }
36    </style>
37    <script type="text/javascript" src="../js/vue.js"></script>
38  </head>
39  <body>
40 >    <!-- ...
51    <!-- 准备好一个容器-->
52    <div id="root">
53      <div class="basic" :class="a" @click="changeMood">{{name}}</div>
54    </div>
55  </body>
56
57 <script type="text/javascript">
58   Vue.config.productionTip = false
59
60   new Vue({
61     el:'#root',
62     data:{
63       name:'尚硅谷',
64       a:'normal'
65     },
66     methods: {
67       changeMood(){
68
69     }
70   <body>
71 >    <!-- ...
72    <!-- 准备好一个容器-->
73    <div id="root">
74      <!-- 绑定class样式--字符串写法，适用于：样式的类名不确定，需要动态指定 -->
75      <div class="basic" :class="mood" @click="changeMood">{{name}}</div>
76    </div>
77  </body>
78
79 <script type="text/javascript">
80   Vue.config.productionTip = false
81
82   new Vue({
83     el:'#root',
84     data:{
85       name:'尚硅谷',
86       mood:'normal'
87     },
88     methods: {
89       changeMood(){
90         this.mood = 'happy'
91       }
92     }
93   }
```

```
<div class="basic" :class="mood" @click="changeMood">{{name}}</div> <br/><br/>

<!-- 绑定class样式--数组写法，适用于：模式的类名不确定，需要动态指定 -->
<div class="basic" :class="classArr">{{name}}</div>
</div>
</body>

<script type="text/javascript">
Vue.config.productionTip = false

const vm = new Vue({
  el:'#root',
  data:{
    name:'尚硅谷',
    mood:'normal',
    classArr:['atguigu1','atguigu2','atguigu3']
  },
  methods: ...
})
</script>

<!-- 绑定class样式--对象写法，适用于：要绑定的样式个数确定、名字也确定，但要动态决定用不用 -->
<div class="basic" :class="classObj">{{name}}</div>
</div>
</body>

<script type="text/javascript">
Vue.config.productionTip = false

const vm = new Vue({
  el:'#root',
  data:{
    name:'尚硅谷',
    mood:'normal',
    classArr:['atguigu1','atguigu2','atguigu3'],
    classObj:{
      atguigu1:false,
      atguigu2:false,
    }
  },
})
```

```
<div class="basic" :style="{fontSize: fsize+'px'}">{{name}}</div>
</div>
</body>

<script type="text/javascript">
  Vue.config.productionTip = false

  const vm = new Vue({
    el: '#root',
    data: {
      name: '尚硅谷',
      mood: 'normal',
      classArr: ['atguigu1', 'atguigu2', 'atguigu3'],
      classObj: {
        atguigu1: false,
        atguigu2: false,
      },
      fsize: 40
    },
    methods: {
    })
  </script>
```

第一这里的样式要用对象
即用{}进行管理，还有font-size要改为驼峰命名法，这样才能动态绑定内嵌的样式

条件渲染

```
<!-- 静态幻灯片准备 -->
<div id="root">

  <!-- 使用v-show做条件渲染 -->
  <!-- <h2 v-show="false">欢迎来到{{name}}</h2> -->
  <!-- <h2 v-show="1 === 1">欢迎来到{{name}}</h2> -->

  <!-- 使用v-if做条件渲染 -->
  <!-- <h2 v-if="false">欢迎来到{{name}}</h2> -->
  <!-- <h2 v-if="1 === 1">欢迎来到{{name}}</h2> -->

</div>
</body>

<script type="text/javascript">
```

```
7   </head>
8   <body>
9 >     <!-- ...
27     <!-- 准备好一个容器-->
28     <div id="root">
29       <h2>当前的n值是:{{n}}</h2>
30       <button @click="n++">点我n+1</button>
31       <!-- 使用v-show做条件渲染 -->
32       <!-- <h2 v-show="false">欢迎来到{{name}}</h2> -->
33       <!-- <h2 v-show="1 === 1">欢迎来到{{name}}</h2> -->
34
35       <!-- 使用v-if做条件渲染 -->
36       <!-- <h2 v-if="false">欢迎来到{{name}}</h2> -->
37       <!-- <h2 v-if="1 === 1">欢迎来到{{name}}</h2> -->
38
39       <div v-show="n === 1">Angular</div>
40       <div v-show="n === 2">React</div>
41       <div v-show="n === 3">Vue</div>
42
43     </div>
44   </body>
45
```

```
9 >     <!-- ... -->
27     <!-- 准备好一个容器-->
28     <div id="root">
29         <h2>当前的n值是:{{n}}</h2>
30         <button @click="n++">点我n+1</button>
31         <!-- 使用v-show做条件渲染 -->
32         <!-- <h2 v-show="false">欢迎来到{{name}}</h2> -->
33         <!-- <h2 v-show="1 === 1">欢迎来到{{name}}</h2> -->
34         +
35         <!-- 使用v-if做条件渲染 -->
36         <!-- <h2 v-if="false">欢迎来到{{name}}</h2> -->
37         <!-- <h2 v-if="1 === 1">欢迎来到{{name}}</h2> -->
38
39         <div v-if="n === 1">Angular</div>
40         <div v-if="n === 2">React</div>
41         <div v-if="n === 3">Vue</div>
42
43     </div>
44     </body>
45
46     <script type="text/javascript">
<!-- <h2 v-if="false">欢迎来到{{name}}</h2>
<!-- <h2 v-if="1 === 1">欢迎来到{{name}}</h2>

<div v-if="n === 1">Angular</div>
<div v-else-if="n === 2">React</div>
<div v-else-if="n === 3">Vue</div>

</div>
body>
```

```
43 <div>
44   MDN Reference
45   <template v-if="n === 1">
46     <h2>你好</h2>
47     <h2>尚硅谷</h2>
48     <h2>北京</h2>
49   </template>
50
51 </div>
52 </body>
<!-- | -->
<template v-if="n === 1">
  <h2>你好</h2>
  <h2>尚硅谷</h2>
  <h2>北京</h2>
</template>

</div>
</body>
```

template只能配合v-if使用
不能配合v-show使用

<!--

条件渲染:

1.v-if

写法:

- (1).v-if="表达式"
- (2).v-else-if="表达式"
- (3).v-else="表达式"

适用于: 切换频率较低的场景。

特点: 不展示的DOM元素直接被移除。

注意: v-if可以和v-else-if、v-else一起使用, 但要求结构不能被“打断”。

2.v-show

写法: v-show="表达式"

适用于: 切换频率较高的场景。

特点: 不展示的DOM元素未被移除, 仅仅是使用样式隐藏掉

3.备注: 使用v-if的时候, 元素可能无法获取到, 而使用v-show一定可以获取到。

-->

<!-- 准备好一个容器-->

列表渲染

```
8 <body>
9 >     <!-- ...
5     <!-- 准备好一个容器-->
6     <div id="root">    [
7     |
8         <ul>
9             <li v-for="p in persons">
0                 {{p.name}}-{{p.age}}
1             </li>
2         </ul>
3     </div>
4
5     <script type="text/javascript">
6         Vue.config.productionTip = false
7
8         new Vue({
9             el:'#root',
0             data:{
1                 persons:[
2                     {id:'001',name:'张三',age:18},
3                     {id:'002',name:'李四',age:19},
4                     {id:'003',name:'王五',age:20}
5                 ]
6             }
7         })

```

```
> 1.基本列表.html > html > body > div#root > ul > li
<!-- 遍历数组 -->
<h2>人员列表</h2>
<ul>
  <li v-for="(p,index) of persons" :key="index">
    {{p.name}}-{{p.age}}
  </li>
</ul>
<!!-- 遍历对象 -->
<h2>汽车信息</h2>
<ul>
  <li v-for="(value,key) of car" :key>
    {{a}}--{{b}}
  </li>
</ul>
</div>

<script type="text/javascript">
  Vue.config.productionTip = false

  new Vue({
    el: '#root',
    data: {
      persons: [ ... ],
      car: {
        name: '奥迪A8',
        price: '70万',
        color: '黑色'
      }
    }
  })
</script>
```

```
<h2>测试遍历字符串</h2>
<ul>
  <li v-for="(char,index) of str" :key="index">
    {{a}}-{{b}}
  </li>
</ul>
</div>

<script type="text/javascript">
  Vue.config.productionTip = false

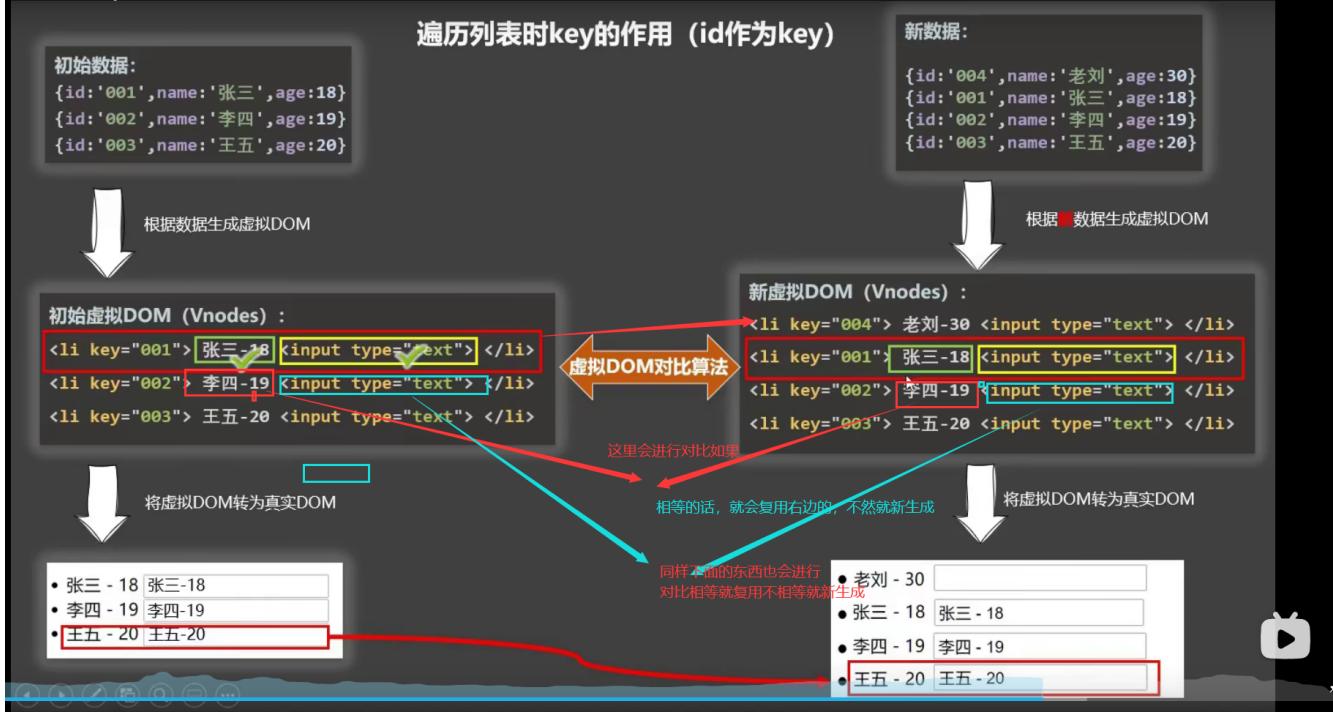
  new Vue({
    el:'#root',
    data:{
      persons:[ ... ],
      car:{ ... },
      str:'hello'
    }
  })
</script>
```

```
<!-- 遍历指定次数 -->
<h2>测试遍历指定次数</h2>
<ul>
  <li v-for="(number,index) of 5" >
    {{a}}-{{b}}
  </li>
</ul>
</div>
```

v-for指令：

1. 用于展示列表数据
2. 语法: `v-for="(item, index) in xxx" :key="yyy"`
3. 可遍历: 数组、对象、字符串(用的很少)、指定次数(用的很少)

key的原理



```
persons:[
  {id:'001',name:'张三',age:18},
  {id:'002',name:'李四',age:19},
  {id:'003',name:'王五',age:20}
],
methods: {
  add(){
    const p = {id:'004',name:'老刘',age:40}
    this.persons.unshift(p)
  }
}
```

这个指令是在**persons**的前面添加一条数据

```

el: '#root',
data: {
  persons: [
    {id: '001', name: '张三', age: 18},
    {id: '002', name: '李四', age: 19},
    {id: '003', name: '王五', age: 20}
  ]
},
methods: {
  add(){
    const p = {id: '004', name: '老刘', age: 40}
    this.persons.unshift(p)
  }
},
]

```

这个是在persons的后面
添加一条数据

<!--
面试题：react、vue中的key有什么作用？（key的内部原理）

1. 虚拟DOM中key的作用：

key是虚拟DOM对象的标识。当状态中的数据发生变化时，Vue会根据【新数据】生成【新的虚拟DOM】，随后Vue进行【新虚拟DOM】与【旧虚拟DOM】的差异比较，比较规则如下：

2. 对比规则：

(1). 旧虚拟DOM中找到了与新虚拟DOM相同的key:

①. 若虚拟DOM中内容没变，则直接使用之前的真实DOM！

②. 若虚拟DOM中内容变了，则生成新的真实DOM，随后替换掉页面中之前的真实DOM。

(2). 旧虚拟DOM中未找到与新虚拟DOM相同的key

创建新的真实DOM，随后渲染到到页面。

3. 用index作为key可能会引发的问题：

1. 若对数据进行：逆序添加、逆序删除等破坏顺序操作：

会产生没有必要的真实DOM更新 => 界面效果没问题，但效率低。

2. 如果结构中还包含输入类的DOM：

会产生错误DOM更新 => 界面有问题。

4. 开发中如何选择key？：

1. 最好使用每条数据的唯一标识作为key，比如id、手机号、身份证号、学号等唯一值。

2. 如果不存在对数据的逆序添加、逆序删除等破坏顺序操作，仅用于渲染列表用于展示，使用index作为key是没有问题的。

列表过滤

3.列表过滤.html > html > body > div#root > ul > li

```
<li v-for="(p,index) of filPerons" :key="index">
    {{p.name}}-{{p.age}}-{{p.sex}}
</li>
</ul>
</div>

<script type="text/javascript">
  Vue.config.productionTip = false

  new Vue({
    el: '#root',
    data: {
      keyword: '',
      persons: [
        {id: '001', name: '马冬梅', age: 19, sex: '女'},
        {id: '002', name: '周冬雨', age: 20, sex: '女'},
        {id: '003', name: '周杰伦', age: 21, sex: '男'},
        {id: '004', name: '温兆伦', age: 22, sex: '男'}
      ],
      filPerons: []
    },
    watch: {
      keyword(val){
        1 this.filPerons = this.persons.filter((p)=>{
          return p.name.indexOf(val) != -1
        })
      }
    }
  })
</script>
```

① 这里如果包含会⁴回所包含值的索引，其中-1就代表不包含

```
12.列表渲染 > 3.列表过滤.html > html > body > script > watch > keyWord > immediate
  20      <script type="text/javascript">
  21          Vue.config.productionTip = false
  22
  23          new Vue({
  24              el:'#root',
  25              data:{
  26                  keyword:'',
  27                  persons:[
  28                      {id:'001',name:'马冬梅',age:19,sex:'女'},
  29                      {id:'002',name:'周冬雨',age:20,sex:'女'},
  30                      {id:'003',name:'周杰伦',age:21,sex:'男'},
  31                      {id:'004',name:'温兆伦',age:22,sex:'男'}
  32                  ],
  33                  filPerons:[]
  34              },
  35              watch:{
  36                  keyword:{}
  37                      immediate:true,
  38                  handler(val){
  39                      this.filPerons = this.persons.filter((p)=>{
  40                          return p.name.indexOf(val) !== -1
  41                      })
  42                  }
  43              }
  44          })
  45      </script>
  46  </html>
```

解决不可折叠问题

```
//#region 头块
//#region
/* new Vue({
  el:'#root',
  data:{
    keyWord:'',
    persons:[
      {id:'001',name:'马冬梅',age:19,sex:'女'},
      {id:'002',name:'周冬雨',age:20,sex:'女'},
      {id:'003',name:'周杰伦',age:21,sex:'男'},
      {id:'004',name:'温兆伦',age:22,sex:'男'}
    ],
    filPerons:[]
  },
  watch:{
    keyword:{
      immediate:true,
      handler(val){
        this.filPerons = this.persons.filter((p)=>{
          return p.name.indexOf(val) !== -1
        })
      }
    }
  }
}) */
//#endregion|
```

就算下面写代码也可以折叠

```
12_列表渲染 > 3.列表过滤.html > html > body > script > computed > filPerons  
vue.config.js - 1.1.12  
22 //用watch实现  
23 //用computed实现  
24 > //用computed实现  
25  
26 //用computed实现  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71
```

这个使用计算属性实现的

```
new Vue({  
    el: '#root',  
    data:{  
        keyWord:'',  
        persons:[  
            {id:'001',name:'马冬梅',age:19,sex:'女'},  
            {id:'002',name:'周冬雨',age:20,sex:'女'},  
            {id:'003',name:'周杰伦',age:21,sex:'男'},  
            {id:'004',name:'温兆伦',age:22,sex:'男'}  
        ]  
    },  
    computed:{  
        filPerons(){  
            return this.persons.filter((p)=>{  
                return p.name.indexOf(this.keyWord) != -1  
            })  
        }  
    }  
})  
</script>  
</html>
```

这个return是filPerons的

这个return是filter的

列表排序

0 为**false**, 1 为**true**。 **bool**表示布尔型变量，也就是逻辑型变量的定义符，以英国数学家、布尔代数的奠基人乔治·布尔 (George Boole) 命名。布尔型变量**bool**的取值只有**false**和**true**，0 为**false**，非 0 为**true**。（例如-1 和 1 都是**true**）。 转载于:<https://www.cnblogs.com/zhouj850/p/10430774.html...>

```
keyword: '',
sortType: 0, //0原顺序 1降序 2升序
persons:[
  {id:'001',name:'马冬梅',age:19,sex:'女'},
  {id:'002',name:'周冬雨',age:20,sex:'女'},
  {id:'003',name:'周杰伦',age:21,sex:'男'},
  {id:'004',name:'温兆伦',age:22,sex:'男'}
],
computed:{
  fillPersons(){
    const arr = this.persons.filter((p)=>{
      return p.name.indexOf(this.keyWord) !== -1
    })
    //判断一下是否需要排序
    if(this.sortType){
      }
  }
},
```

```
let arr = [1,3,2,6,4,5]
arr.sort((a,b)=>{
  return a-b  [ 这个是升序 ]
})
```

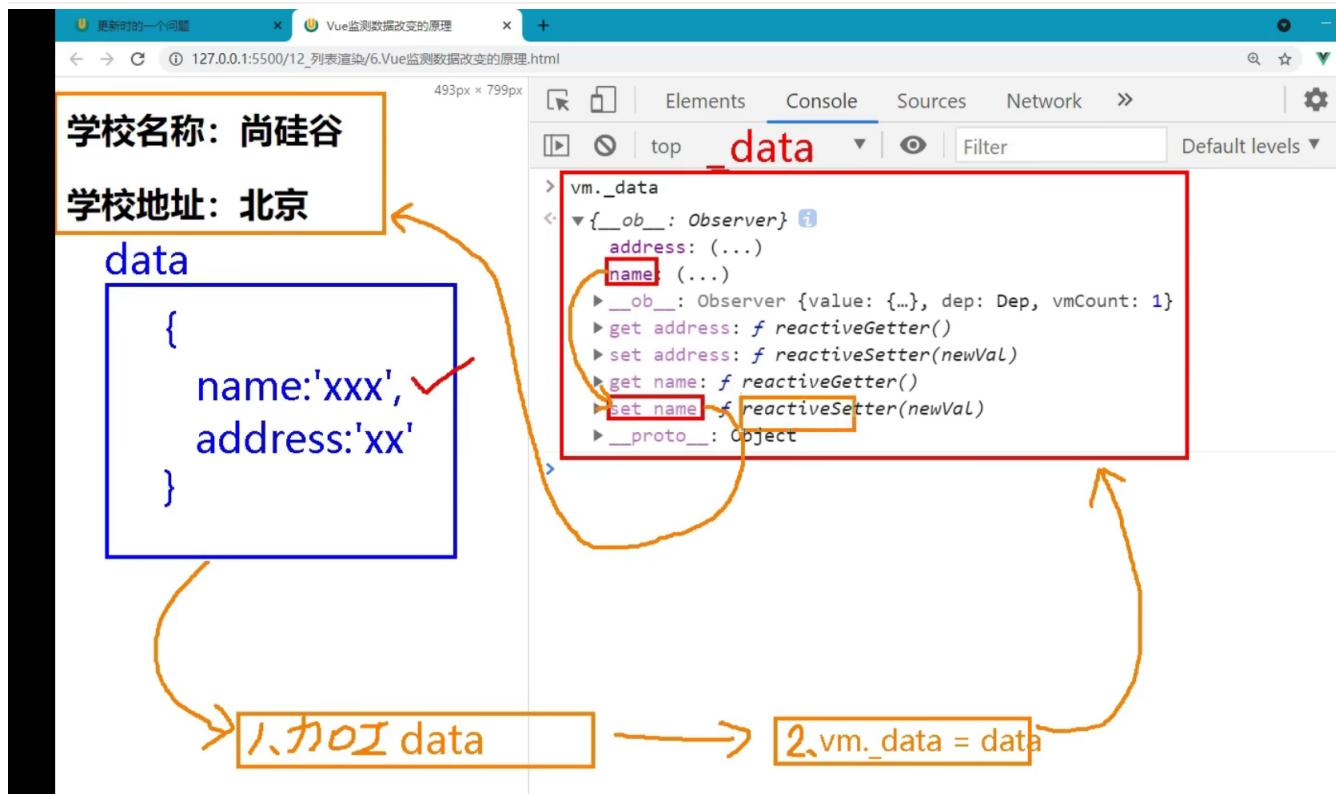
```
let arr = [1,3,2,6,4,5]
arr.sort((a,b)=>{
  return b-a  降序
})
```

```

el: '#root',
data: {
  keyword: '',
  sortType: 0, // 0原顺序 1降序 2升序
  persons: [
    {id: '001', name: '马冬梅', age: 19, sex: '女'},
    {id: '002', name: '周冬雨', age: 20, sex: '女'},
    {id: '003', name: '周杰伦', age: 21, sex: '男'},
    {id: '004', name: '温兆伦', age: 22, sex: '男'}
  ]
},
computed: {
  filPerons() {
    const arr = this.persons.filter((p) => {
      return p.name.indexOf(this.keyword) !== -1
    })
    // 判断一下是否需要排序
    if(this.sortType){
      arr.sort((p1,p2)=>{
        return this.sortType === 1 ? p2.age-p1.age : p1.age-p2.age
      })
    }
    return arr
  }
}

```

数据检测对象原理



push → 最后面加一个

pop → 最后面减一个

shift → ~~最前面加一个~~ 删除第一个

unshift ← ~~最前面减一个~~ 最前面添加一个

splice →

sort →

reverse →

```
> vm._data.student.hobby.splice(0,1,'打台球')
< ▶ ["喝酒"]
>
```

第一个 删除 替换为打台

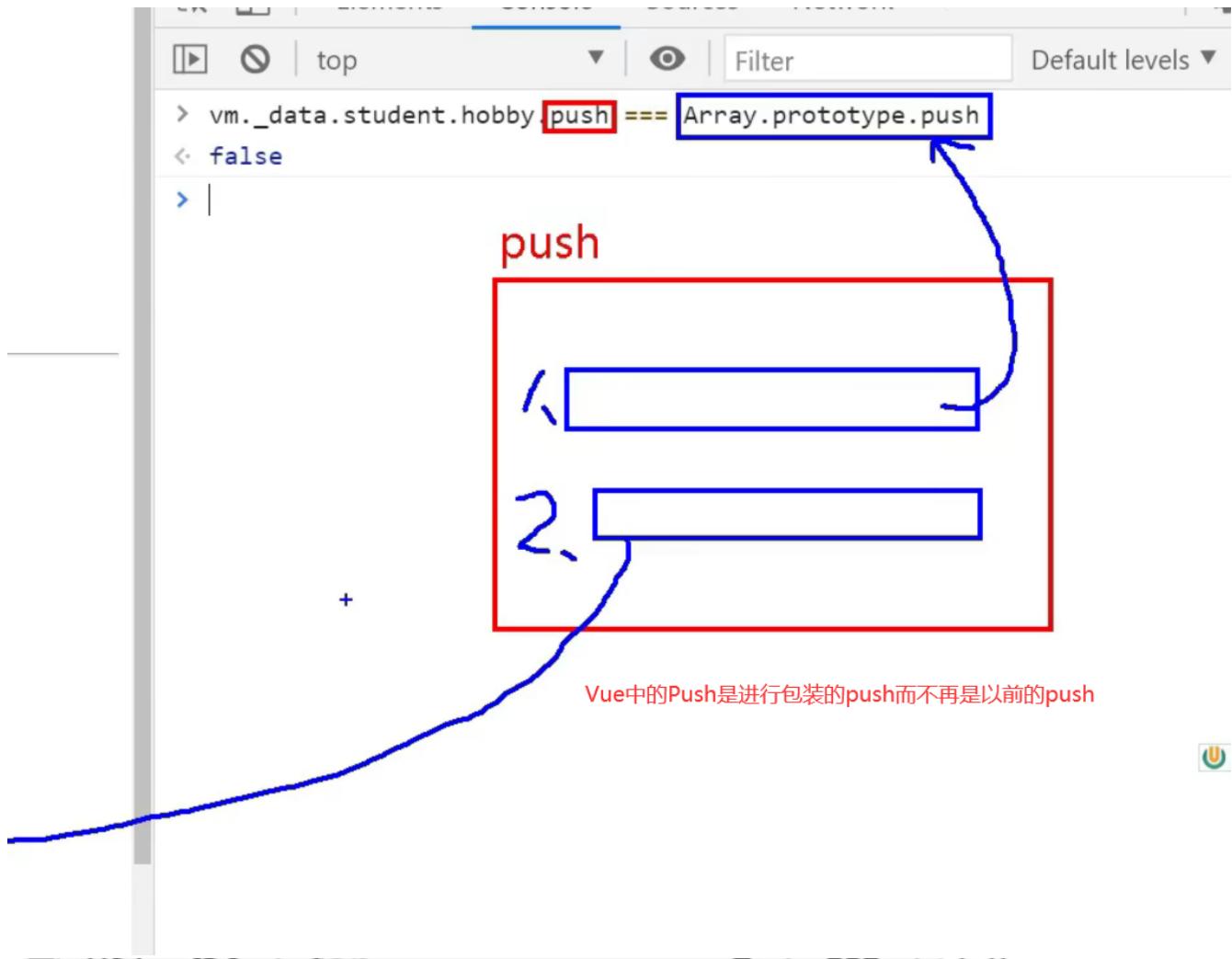
```
> vm._data.student.hobby.push === Array.prototype.push
< false
> |
```

push

1.

2.

Vue中的Push是进行包装的push而不是以前的push



```
> Vue.set(vm._data.student.hobby, 1, '打台球')
  // Vue.set(this.student, 'sex', '男')
  this.$set(this.student, 'sex', '男')
```

<!--

Vue监视数据的原理：

1. 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)

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！！！



收集表单数据

```
<form>
  <label for="demo">账号: </label>
  <input type="text" id="demo">
</div>
<div id="sex">
  <form>
    账号: <input type="text" v-model="account"> <br/><br/>
    密码: <input type="password" v-model="password"> <br/><br/>
    性别:
      男<input type="radio" name="sex" v-model="sex" value="male">
      女<input type="radio" name="sex" v-model="sex" value="female"> <br/><br/>
    爱好:
      学习<input type="checkbox" v-model="hobby" value="study">
      打游戏<input type="checkbox" v-model="hobby" value="game">
      吃饭<input type="checkbox" v-model="hobby" value="eat">
    <br/><br/>
  </div>
```

```
new Vue({
  el:'#root',
  data:{
    account:'',
    password:'',
    sex:'female',
    hobby:[],
  },
  methods: {
    demo(){
      console.log(JSON.stringify(this._data))
    }
  }
})
sex: 'female',
hobby:[],
city:'beijing',
other:'',
agree:''
},
methods: {
  demo(){
    console.log(JSON.stringify(this.userInfo))
  }
}
})
</script>
  agree.
}
},
methods: {
  demo(){
    console.log(JSON.stringify(this.userInfo))
  }
}
})
</script>
</html>
```

```

    年龄: <input type="password" v-model="userInfo.password"> <br/><br/>
    年龄: <input type="number" v-model.number="userInfo.age" /> 收集为数字类型 <br/><br/>

    其他信息: <input type="text" v-model.lazy="userInfo.other" /> 失去焦点收集 <br/><br/>
    账号: <input type="text" v-model.trim="userInfo.account" /> 去空格 <br/><br/>
<!--
    收集表单数据:
        若: <input type="text"/>, 则v-model收集的是value值, 用户输入的就是value值。
        若: <input type="radio"/>, 则v-model收集的是value值, 且要给标签配置value值。
        若: <input type="checkbox"/>
            1. 没有配置input的value属性, 那么收集的就是checked (勾选 or 未勾选, 是布尔值)
            2. 配置input的value属性:
                (1)v-model的初始值是非数组, 那么收集的就是checked (勾选 or 未勾选, 是布尔值)
                (2)v-model的初始值是数组, 那么收集的就是value组成的数组
    备注: v-model的三个修饰符:
        lazy: 失去焦点再收集数据
        number: 输入字符串转为有效的数字
        trim: 输入首尾空格过滤
-->
```

过滤器

```

    <h3>现在是: {{time | timeFormater}}</h3>
    </div>
</body>

<script type="text/javascript">
    Vue.config.productionTip = false

    new Vue({
        el: '#root',
        data: {
            time: 1621561377603 // 时间戳
        },
        computed: { ... },
        methods: { ... },
        filters: {
            timeFormater(value) {
                return value
            }
        }
    })
</script>
```

这里的time作为参数传给value

```
<h3>现在是: {{time | timeFormater('YYYY_MM_DD')}}</h3>
</div>
</body>

script type="text/javascript">
  Vue.config.productionTip = false

  new Vue({
    el:'#root',
    data:{
      time:1621561377603 //时间戳
    },
    computed: { ... },
    methods: { ... },
    filters:[
      timeFormater(value,str){
        // console.log('@',value)
        return dayjs(value).format(str)
      }
    ]
  </body>
  37
  38  <script type="text/javascript">
  39    Vue.config.productionTip = false
  40    //全局过滤器
  41    Vue.filter('mySlice',function(value){
  42      return value.slice(0,4)
  43    })
  44
  45  >    new Vue({ ...
  68
  69  >    new Vue({ ...
  75  </script>
  76  </html>
```

参数对应表

```

methods: { ... },
//局部过滤器
filters: {
  timeFormater(value,str='YYYY年MM月DD日 HH:mm:ss'){
    // console.log('@',value)
    return dayjs(value).format(str)
  }
}
})

```

9 <body>

10 <!--

11 过滤器：
 定义：对要显示的数据进行特定格式化后再显示（适用于一些简单逻辑的处理）。
 语法：
 1.注册过滤器：Vue.filter(name,callback) 或 new Vue{filters:{}}
 2.使用过滤器：{{ xxx | 过滤器名}} 或 v-bind:属性 = "xxx | 过滤器名"
 备注：
 1.过滤器也可以接收额外参数、多个过滤器也可以串联
 2.并没有改变原本的数据，是产生新的对应的数据

9 -->

内置指令

V-text

```

<div id="root">
  <div>{{name}}</div>
  <div v-text="name"></div>
</div>
</body>

<script type="text/javascript">
  Vue.config.productionTip = false //阻止 vue 在启动时生成生产

  new Vue({
    el:'#root',
    data:{
      name:'尚硅谷'
    }
  })

```

V-Html

```

<!--
    v-html指令:
        1.作用: 向指定节点中渲染包含html结构的内容。
        2.与插值语法的区别:
            (1).v-html会替换掉节点中所有的内容, {{xx}}则不会。
            (2).v-html可以识别html结构。
        3.严重注意: v-html有安全性问题!!!!
            (1).在网站上动态渲染任意HTML是非常危险的, 容易导致XSS攻击。
            (2).一定要在可信的内容上使用v-html, 永不要用在用户提交的内容上!
-->
<!-- 准备好一个容器-->

```

V-cloak:

作用: 在网速过慢的情况下防止未经解析的元素跑到页面上去

```

... 3.v-cloak指令.html M X
15_内置指令 > 3.v-cloak指令.html > html > head > style > [v-cloak]
1   <!DOCTYPE html>
2   <html>
3     <head>
4       <meta charset="UTF-8" />
5       <title>v-cloak指令</title>
6       <style>
7         [v-cloak]{}
8           display:none;
9         }
10      </style>
11      <!-- 引入Vue -->
12    </head>
13    <body>
14      <!-- 准备好一个容器-->
15      <div id="root">
16        <h2 v-cloak>{{name}}</h2>
17      </div>
18      <script type="text/javascript" src="http://localhost:8080/resource/5s/vue.js"></script>
19    </body>
20
21    <script type="text/javascript">
22      console.log(1)
23      Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。
24
25      new Vue({
26        el:'#root',
27        data:{
28          name:'尚硅谷'

```

v-cloak 指令 (没有值):

- 本质是一个特殊属性, Vue实例创建完毕并接管容器后, 会删掉v-cloak属性。
- 使用css配合v-cloak可以解决网速慢时页面展示出{{xxx}}的问题。

V-once

```

9  <body>
10 <!--
11   v-once指令:
12     1.v-once所在节点在初次动态渲染后，就视为静态内容了。
13     2.以后数据的改变不会引起v-once所在结构的更新，可以用于优化性能。
14   -->
15   <!-- 准备好一个容器-->
16   <div id="root">
17     <h2 v-once>初始化的n值是:{{n}}</h2>
18     <h2>当前的n值是:{{n}}</h2>
19     <button @click="n++">点我n+1</button>
20   </div>
21 </body>
22
23 <script type="text/javascript">
24   Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。
25
26   new Vue({
27     el:'#root',
28     data:{
```



V-pre

```

7   <script type="text/javascript" src="../js/vue.js"></script>
8 </head>
9 <body>
10 <!--
11   v-pre指令:
12     1.跳过其所在节点的编译过程。
13     2.可利用它跳过：没有使用指令语法、没有使用插值语法的节点，会加快编译。
14   -->
15   <!-- 准备好一个容器-->
16   <div id="root">
17     <h2 v-pre>Vue其实很简单</h2>
18     <h2 v-pre>当前的n值是:{{n}}</h2>
19     <button v-pre @click="n++">点我n+1</button>
20   </div>
21 </body>
22
23 <script type="text/javascript">
24   Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。
25
26   new Vue({
27     el:'#root',
28     data:{
```



自定义指令

```
自定义指令 > 自定义指令.html > html > script > directives
0     需求1：定义一个v-big指令，和v-text功能类似，但会把绑定的数值放大10倍。
1     需求2：定义一个v-fbind指令，和v-bind功能类似，但可以让其所绑定的input元素默认获取焦点。
2     -->
3     <!-- 准备好一个容器-->
4     <div id="root">
5         <h2>{{name}}</h2>
6         <h2>当前的n值是：<span v-text="n"></span> </h2>
7         <h2>放大10倍后的n值是：<span v-big="n"></span> </h2>
8         <button @click="n++">点我n+1</button>
9     </div>
0     </body>
1
2     <script type="text/javascript">
3         Vue.config.productionTip = false
4         new Vue({
5             el:'#root',
6             data:{
7                 name:'尚硅谷',
8                 n:1
9             },
10            directives:[
11                //big函数何时会被调用？1.指令与元素成功绑定时（一上来）。2.指令所在的模板被重新解析时。
12                big(element,binding){
13                    console.log('big')
14                    element.innerText = binding.value * 10
15                }
16            }
17        })
18    </script>
```

```
1 <!DOCTYPE html>
2 <html>
3     <head>
4         <meta charset="UTF-8" />
5         <title>Document</title>
6     </head>
7     <body>
8         <button id="btn">点我创建一个输入框</button>
9         <script type="text/javascript" >
10            const btn = document.getElementById('btn')
11            btn.onclick = ()=>{
12                const input = document.createElement('input')
13                document.body.appendChild(input)
14                input.focus()
15            }
16        </script>
17     </body>
18 </html>
```

```
directives:{  
    //big函数何时会被调用? 1.指令与元素成功绑定时(一上来)。2.指令所在的模板被重新解析  
    big(element,binding){  
        // console.log('big')  
        element.innerText = binding.value * 10  
    },  
    fbind:{  
        //指令与元素成功绑定时(一上来)  
        bind(){  
            console.log('bind')  
        },  
        //指令所在元素被插入页面时  
        inserted(){  
            //  
        },  
        //指令所在的模板被重新解析时  
        update(){  
            //  
        },  
        //  
    },  
    directives:{  
        //big函数何时会被调用? 1.指令与元素成功绑定时(一上来)。2.指令所在的模板被重新解析  
        big(element,binding){  
            // console.log('big')  
            element.innerText = binding.value * 10  
        },  
        fbind:{  
            //指令与元素成功绑定时(一上来)  
            bind(element,binding){  
                element.value = binding.value  
            },  
            //指令所在元素被插入页面时  
            inserted(element,binding){  
                element.focus()  
            },  
            //指令所在的模板被重新解析时  
            update(element,binding){  
                element.value = binding.value  
            }  
        },  
    }  
}
```

总结

```
14     <div id="root">
15         <h2>{{name}}</h2>
16         <h2>当前的n值是: <span v-text="n"></span> </h2>
17         <h2>放大10倍后的n值是: <span v-big-number="n"></span> </h2>
18         <button @click="n++">点我n+1</button>
19         <hr/>
20         <input type="text" v-fbind:value="n">
21     </div>
22 </body>
23
24 <script type="text/javascript">
25     Vue.config.productionTip = false
26
27     new Vue({
28         el:'#root',
29         data:{
30             name:'尚硅谷',
31             n:1
32         },
33         directives:{
34             //big函数何时会被调用? 1.指令与元素成功绑定时(一上来)。2.指令所在的模板被重新解析时。
35             'big-number'(element,binding){
36                 // console.log('big')
37                 element.innerText = binding.value * 10
38             },
39             fbind:{

```

这里出现了，下面就不能用简写模式需要用单引号括起来

自定义指令里面的this是window

```
<script type="text/javascript">
  Vue.config.productionTip = false

  Vue.directive('fbind'), {
    //指令与元素成功绑定时（一上来）
    bind(element,binding){
      element.value = binding.value
    },
    //指令所在元素被插入页面时
    inserted(element,binding){
      element.focus()
    },
    //指令所在的模板被重新解析时
    update(element,binding){
      element.value = binding.value
    }
  })

  new Vue({
```

```

29
30   <script type="text/javascript">
31     Vue.config.productionTip = false
32
33   Vue.directive('fbind',{
34     //指令与元素成功绑定时(一上来)
35     bind(element,binding){
36       element.value = binding.value
37     },
38     //指令所在元素被插入页面时
39     inserted(element,binding){
40       element.focus()
41     },
42     //指令所在的模板被重新解析时
43     update(element,binding){
44       element.value = binding.value
45     }
46   })
47
48   new Vue({
49     el:'#root',
50     data:{
51       name:'尚硅谷',
52       n:1
53     }

```

自定义指令总结：

一、定义语法：

(1).局部指令：

```

new Vue({
  directives:{指令名:配置对象} 或
})

```

```

new Vue({
  directives{指令名:回调函数}
})

```

(2).全局指令：

```

Vue.directive(指令名,配置对象) 或 Vue.directive(指令名,回调函数)

```

二、配置对象中常用的3个回调：

(1).bind： 指令与元素成功绑定时调用。

(2).inserted： 指令所在元素被插入页面时调用。

(3).update： 指令所在模板结构被重新解析时调用。

三、备注：

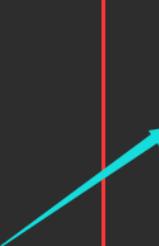
1.指令定义时不加v-，但使用时要加v-；

2.指令名如果是多个单词，要使用kebab-case命名方式，不要用camelCase命名。

```

>
-- 准备好一个容器--> I

```

这个是一个全局的指令

生命周期

```
<div id="#root">
  <h2 :style="{opacity}">欢迎学习Vue</h2>
</div>
</body>          重名的简写形式
```



```
<script type="text/javascript">
  Vue.config.productionTip = false //阻止 vue 在启动时
```



```
new Vue({
  el: '#root',
  data: {
    opacity:1
  }
})
```

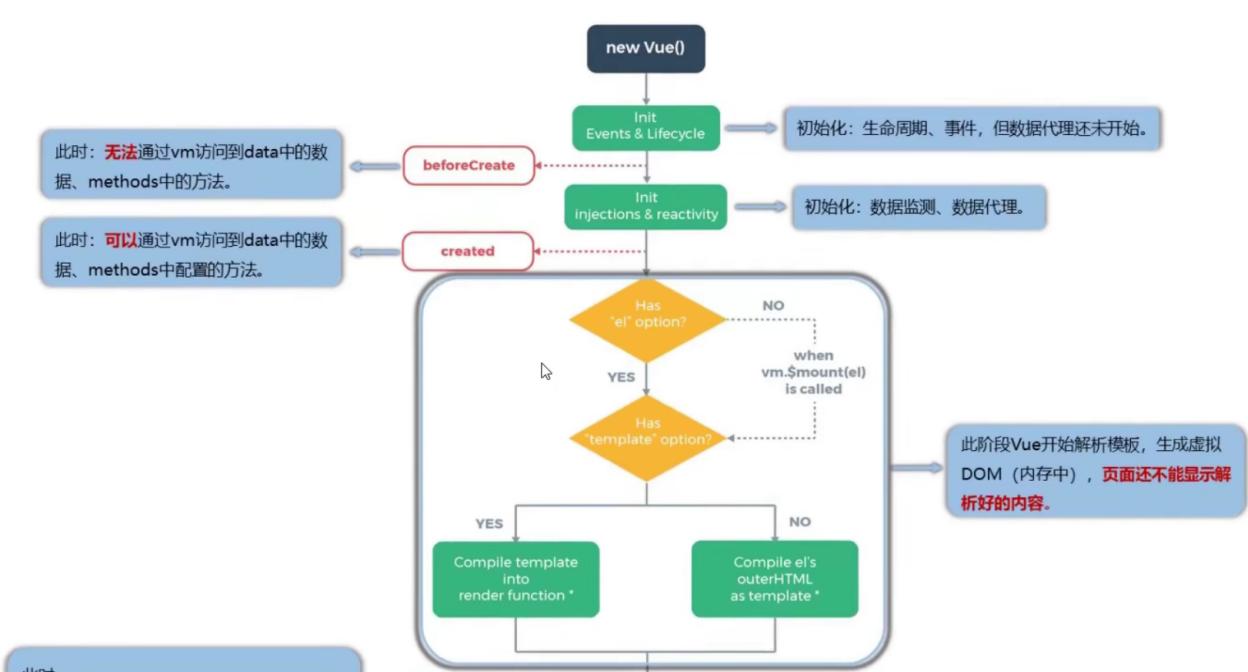
```
19 |   <h2 :style="{opacity}">欢迎学习Vue</h2>
20 |   </div>
21 | </body>
22 |
23 <script type="text/javascript">
24   Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。
25 |
26   new Vue({
27     el: '#root',
28     data: {
29       opacity:1
30     },
31     methods: {
32
33   },
34   //Vue完成模板的解析并把真实的DOM元素放入页面后（挂载完毕）调用mounted
35   mounted(){
36     I
37   }
38 })
39
40 //通过外部的定时器实现（不推荐）
41 /* setInterval(() => {
42   vm.opacity -= 0.01
43   if(vm.opacity <= 0) vm.opacity = 1
44 }) */
45 
```

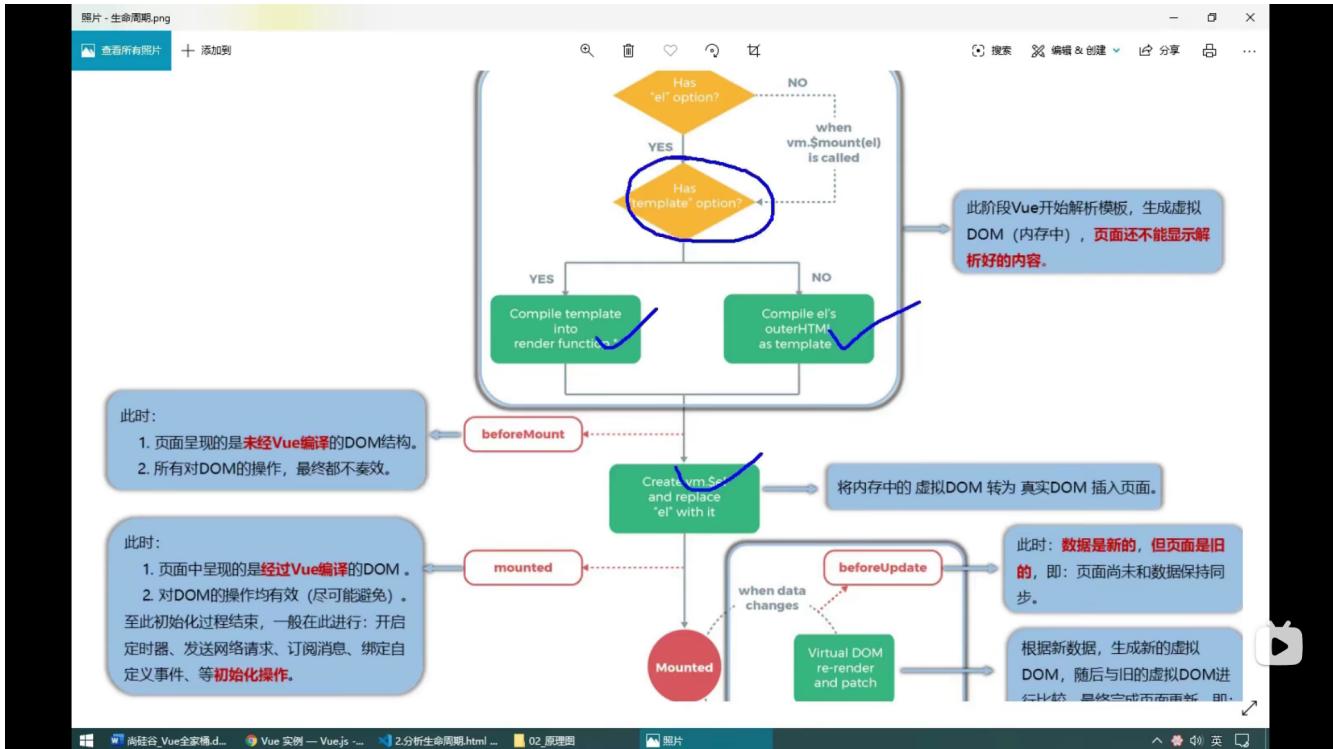
```
// Vue完成模板的解析并把初始的真实DOM元素放入页面后（挂载完毕）调用mounted
mounted(){
  console.log('mounted')
  setInterval(() => {
    this.opacity -= 0.01
    if(this.opacity <= 0) this.opacity = 1
  }, 16)
}

```

生命周期：

1. 又名：生命周期回调函数、生命周期函数、生命周期钩子。
2. 是什么：Vue在关键时刻帮我们调用的一些特殊名称的函数。
3. 生命周期函数的名字不可更改，但函数的具体内容是程序员根据需求编写
4. 生命周期函数中的this指向是vm 或 组件实例对象





```

<body>
  <!-- 准备好一个容器 -->
  <div id="root" :x="n">
    <h2>当前的n值是: {{n}}</h2>
    <button @click="add">点我n+1</button>
  </div>
</body>

<script type="text/javascript">
  Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。

```

```

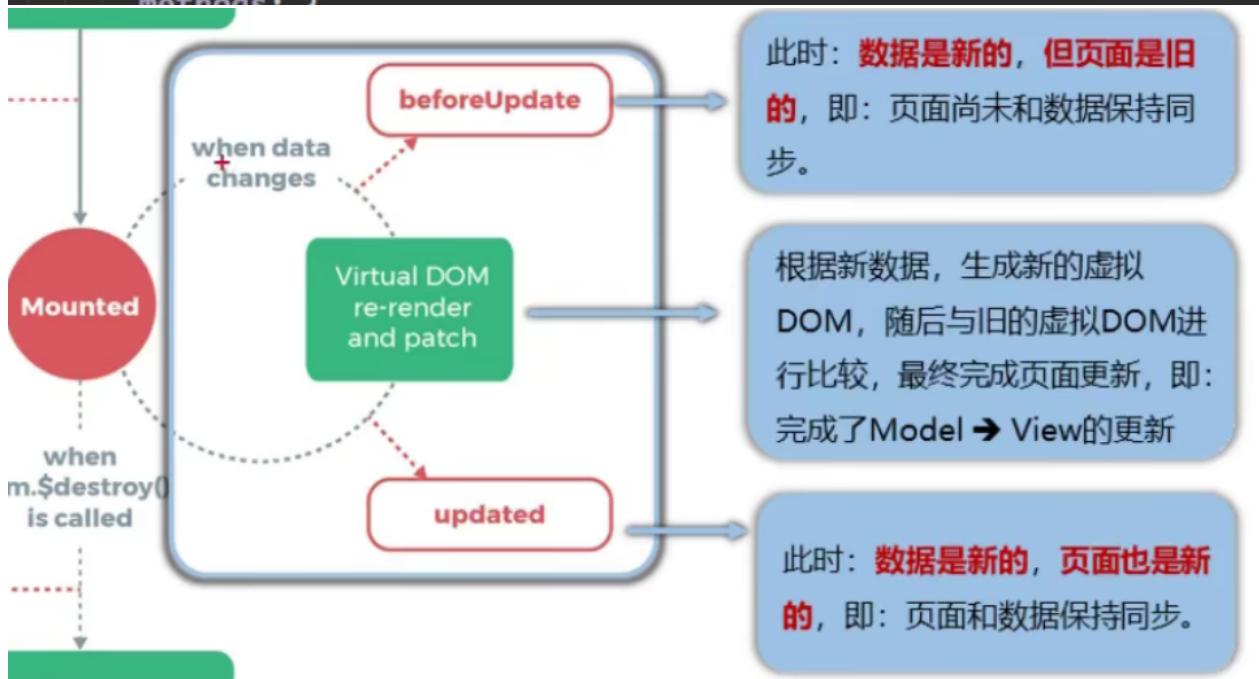
<body>
  <!-- 准备好一个容器-->
  <div id="root" :x="n">

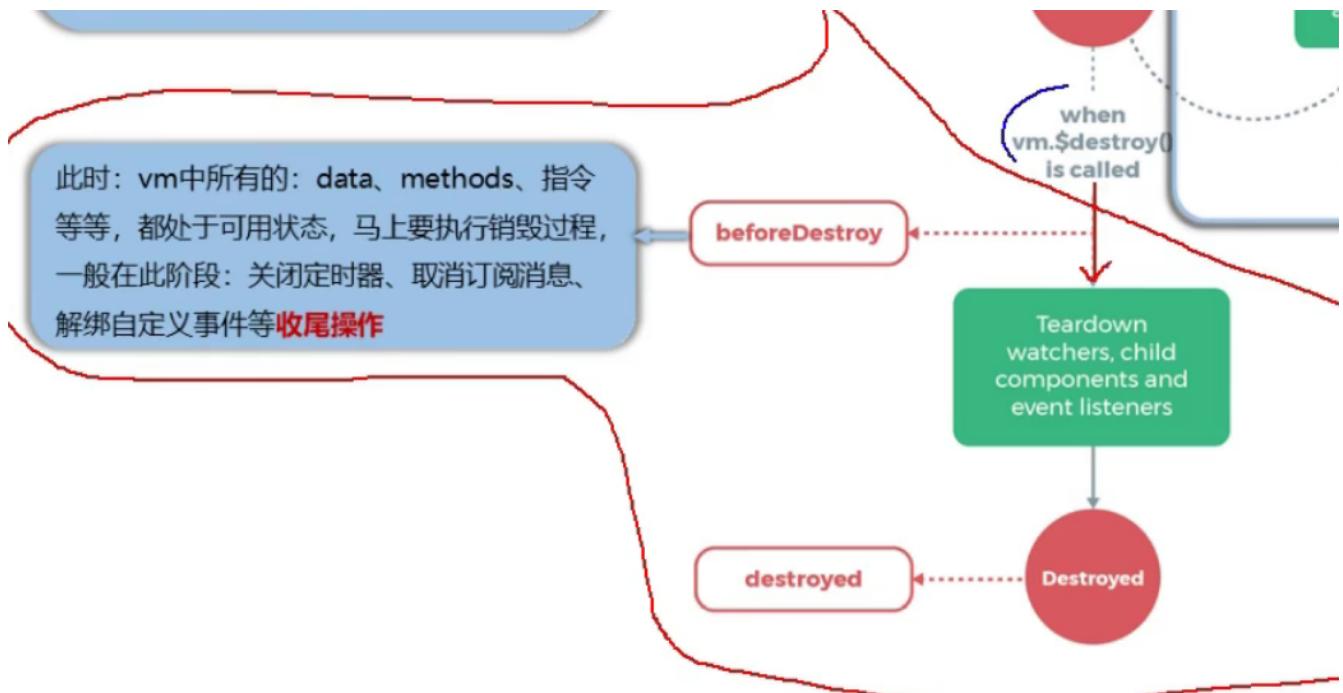
  </div>
</body>

<script type="text/javascript">
  Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。

  new Vue({
    el: '#root',
    template: `
      <div>
        <h2>当前的n值是: {{n}}</h2>
        <button @click="add">点我n+1</button>
      </div>
    `,
    data: {
      n: 1
    },
    methods: {
      add() {
        this.n++
      }
    }
  })
</script>

```





```

1 张三的一生(张三的生命周期):
2     将要出生
3     (重要) 呱呱坠地 ==> 检查身体各项指标。
4     学会说话
5     学会走路
6     .....
7     .....
8     (重要) 将要永别 ==> 交代后事
9     已经永别
10
11
12 vm的一生(vm的生命周期):
13     将要创建 ==> 调用beforeCreate函数。
14     创建完毕 ==> 调用created函数。
15     将要挂载 ==> 调用beforeMount函数。
16     (重要) 挂载完毕 ==> 调用mounted函数。 =====>【重要的钩子】 ✓
17     将要更新 ==> 调用beforeUpdate函数。
18     更新完毕 ==> 调用updated函数。
19     (重要) 将要销毁 ==> 调用beforeDestroy函数。 =====>【重要的钩子】 ✓
20     销毁完毕 ==> 调用destroyed函数。
21
22

```

```
20      new Vue({
21        el:'#root',
22        data:{
23          opacity:1
24        },
25        methods: {
26          stop(){
27            clearInterval(this.timer)
28          }
29        },
30        //Vue完成模板的解析并把初始的真实DOM元素放入页面后（挂载完毕）调用mounted
31        mounted(){
32          console.log('mounted',this)
33          this.timer = setInterval(() => {
34            this.opacity -= 0.01
35            if(this.opacity <= 0) this.opacity = 1
36          },16)
37        },
38      })
39
```

<!--

常用的生命周期钩子：

1. **mounted**: 发送ajax请求、启动定时器、绑定自定义事件、订阅消息等【初始化操作】。
2. **beforeDestroy**: 清除定时器、解绑自定义事件、取消订阅消息等【收尾工作】。

关于销毁Vue实例

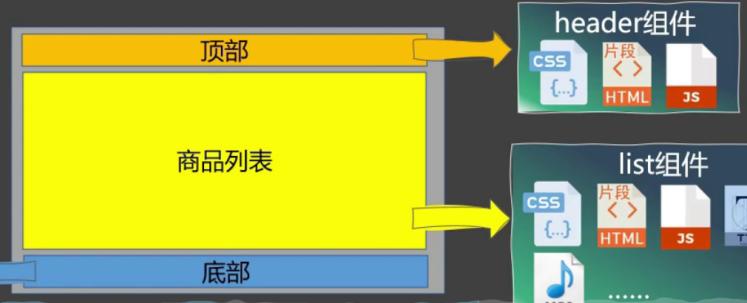
1. 销毁后借助Vue开发者工具看不到任何信息。
2. 销毁后自定义事件会失效，但原生DOM事件依然有效。
3. 一般不会再**beforeDestroy**操作数据，因为即便操作数据，也不会再触发更新流程了。

关于销毁Vue实例

1. 销毁后借助Vue开发者工具看不到任何信息。
2. 销毁后自定义事件会失效，但原生DOM事件依然有效。
3. 一般不会再**beforeDestroy**操作数据，因为即便操作数据，也不会再触发更新流

组件化编程

使用组件方式编写应用



```

<script type="text/javascript">
  Vue.config.productionTip = false

  //创建school组件
  const school = Vue.extend({
    // el:'#root', //组件定义时，一定不要写el配置项，因为最终所有的组件都要被一个vm管理，由vm决定服务
    data:{
      schoolName:'尚硅谷',
      address:'北京昌平',
      studentName:'张三',
      age:18
    }
  })

  //创建vm
  /* new Vue({
    el:'#root',
    data:{
      schoolName:'尚硅谷',
      address:'北京昌平',
      studentName:'张三',
      age:18
    }
  }) */

```

```
37    </body>
38
ml 39    <script type="text/javascript">
40        Vue.config.productionTip = false
41
42    //第一步：创建school组件
43    > const school = Vue.extend({ ...
52
53    //第一步：创建student组件
54    > const student = Vue.extend({ ...
62
63    //创建vm
64    new Vue({
65        el:'#root',
66        //第二步：注册组件（局部注册）
67        components:{
68            xuexiao:school,
69            xuesheng:student
70        }
71    })
```

```
//第一步：创建school组件
const school = Vue.extend({ …

//第一步：创建student组件
const student = Vue.extend({
  template: `
    <div>
      <h2>学生姓名: {{studentName}}</h2>
      <h2>学生年龄: {{age}}</h2>
    </div>
  `,
  data(){
    return {
      studentName:'张三',
      age:18
    }
  }
})
```

```
<!-- 准备好一个容器-->
<div id="root">
  <!-- 第三步：编写组件标签 -->
  <xuexiao></xuexiao>
  <hr>
  <!-- 第三步：编写组件标签 -->
  <xuesheng></xuesheng>
</div>
```

```
//第一步：创建hello组件
const hello = Vue.extend({ ... })

//第二步：全局注册组件
Vue.component('hello', hello)

//创建vm
```

<!--

Vue中使用组件的三大步骤：

- 一、定义组件(创建组件) ✓
- 二、注册组件
- 三、使用组件(写组件标签)

一、如何定义一个组件？

使用`Vue.extend(options)`创建，其中`options`和`new Vue(options)`时传入的那个`options`几乎一样，但区别如下：

- 1.`el`不要写，为什么？ —— 最终所有的组件都要经过一个`vm`的管理，由`vm`中的`el`决定服务哪个容器。
- 2.`data`必须写成函数，为什么？ —— 避免组件被复用时，数据存在引用关系。

备注：使用`template`可以配置组件结构。

二、如何注册组件？

1. 局部注册：靠`new Vue`的时候传入`components`选项
2. 全局注册：靠`Vue.component('组件名', 组件)`



三、编写组件标签：

```
<school></school>
```

<!-- |

几个注意点：

1. 关于组件名：

一个单词组成：

- 第一种写法(首字母小写): `school`
- 第二种写法(首字母大写): `School`

多个单词组成：

- 第一种写法(`kebab-case`命名): `my-school`
- 第二种写法(`CamelCase`命名): `MySchool` (需要`Vue`脚手架支持)

备注：

- (1). 组件名尽可能回避HTML中已有的元素名称，例如：`h2`、`H2`都不行。
- (2). 可以使用`name`配置项指定组件在开发者工具中呈现的名字。



2. 关于组件标签：

第一种写法: `<school></school>`

第二种写法: `<school/>`

备注：不用使用脚手架时，`<school/>`会导致后续组件不能渲染。

3. 一个简写方式：

`const school = Vue.extend(options)` 可简写为: `const school = options`

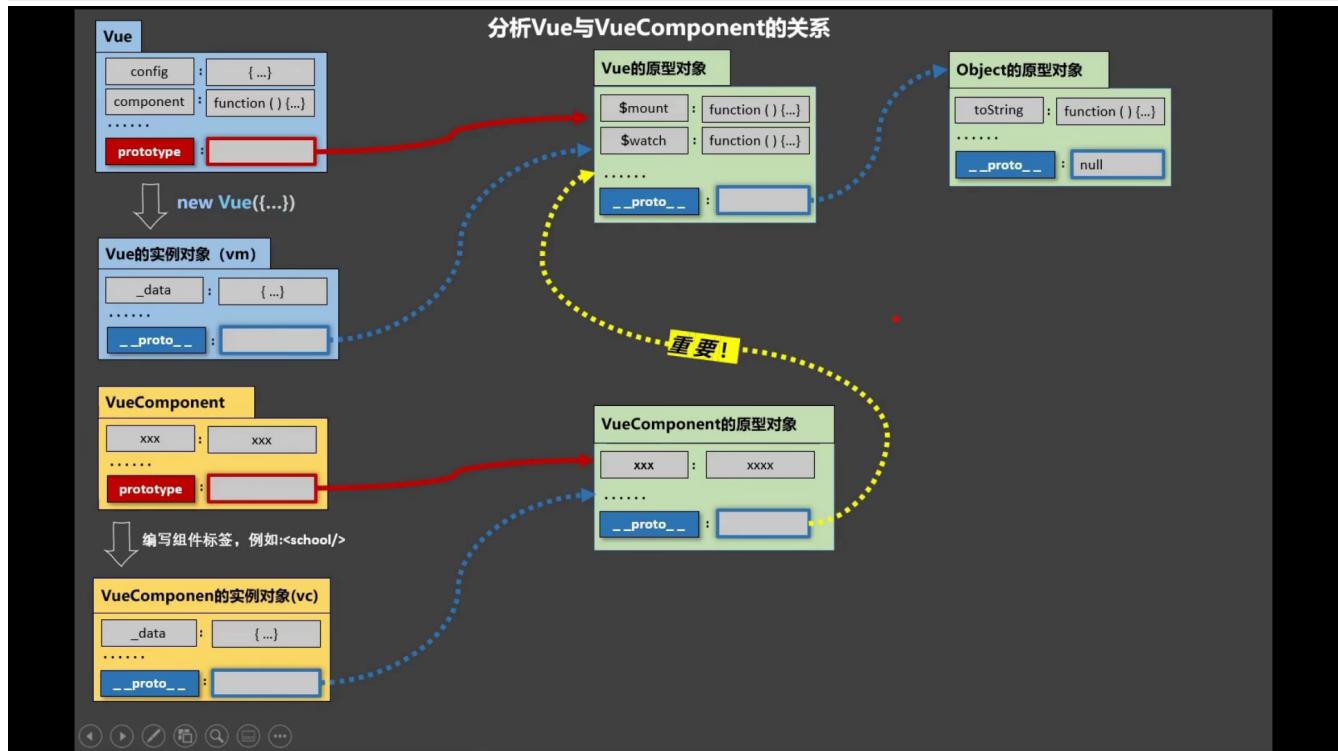
-->

<!-- 准备好一个容器-->

可简写为: `const school = options`

```
<!--  
关于VueComponent:  
1. school组件本质是一个名为VueComponent的构造函数，且不是程序员定义的，是Vue.extend生成的。  
2. 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)。  
3. 特别注意：每次调用Vue.extend，返回的都是一个全新的VueComponent！！！  
4. 关于this指向：  
    (1). 组件配置中：  
        data函数、methods中的函数、watch中的函数、computed中的函数 它们的this均是【VueComponent实例对象】。  
    (2). new Vue()配置中：  
        data函数、methods中的函数、watch中的函数、computed中的函数 它们的this均是【Vue实例对象】。  
5. VueComponent的实例对象，以后简称vc（也可称之为：组件实例对象）。  
    Vue的实例对象，以后简称vm。  
-->  
<!-- 准备好一个空壳 -->  
<!--  
1. 一个重要的内置关系：VueComponent.prototype.__proto__ === Vue.prototype  
2. 为什么要有这个关系：让组件实例对象（vc）可以访问到 Vue原型上的属性、方法。  
-->  
<!-- 准备好一个空壳 -->
```

vue与VueComponent



单文件组件

```
▼ School.vue U X
19_单文件组件 > ▼ School.vue > {} "School.vue" > script > [✖] default
  1  <template>
  2    <div class="demo">
  3      <h2>学校名称: {{schoolName}}</h2>
  4      <h2>学校地址: {{address}}</h2>
  5      <button @click="showName">点我提示学校名</button>
  6    </div>
  7  </template>
  8
  9  <script>
10    export default {
11      name: 'School',
12      data() {
13        return {
14          schoolName: '尚硅谷',
15          address: '北京昌平'
16        }
17      },
18      methods: {
19        showName() {
20          alert(this.schoolName)
21        }
22      },
23    }
24  </script>
25
26  <style>
27    .demo{
28      background-color: #orange;
```

```
1 <template>
2
3 </template>    [
4
5 <script>
6   export default { 快速显示的快捷键
7     <+v即可
8   }
9 </script>
10
11 <style>
12
13 </style>
```

```
▽ App.vue ×
19_单文件组件 > ▽ App.vue > {} "App.vue"
1  <template>
2    <div>
3      <School></School>
4      <Student></Student>
5    </div>
6  </template>
7
8  <script>
9    //引入组件
10   import School from './School'
11   import Student from './Student'
12
13  export default {
14    name: 'App',
15    components: {
16      School,
17      Student
18    }
19  }
20  </script>
21
```



![image-20211021193024995](C:\Users\ZY\AppData\Roaming\Typora\typora-user-images\image-20211021193024995.png)

19_单文件组件 > index.html > html > head > title

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="UTF-8" />
5      <title>练习一下单文件组件的语法</title>
6    </head>
7    <body>
8      <!-- 准备一个容器 -->
9      <div id="root"></div>
10     <script type="text/javascript" src="../js/vue.js"></script>
11     <script type="text/javascript" src="./main.js"></script>
12   </body>
13 </html>
```

19_单文件组件 > main.js > ...

```
1  import App from './App.vue'
2
3  new Vue({
4    el:'#root',
5    template:`<App></App>`,
6    components:{App},
7  })
```

```
JS main.js > render
1  /*
2   |   该文件是整个项目的入口文件
3   */
4  //引入Vue
5  import Vue from 'vue'
6  //引入App组件，它是所有组件的父组件
7  import App from './App.vue'
8  //关闭vue的生产提示
9  Vue.config.productionTip = false
0
1  //创建Vue实例对象---vm
2  new Vue({
3      |r any :: h => h(App),
4  }).$mount('#app')
```

```
> index.html > html > body > noscript
3  <head>
4      <meta charset="utf-8">
5      <!-- 针对IE浏览器的一个特殊配置，含义是让IE浏览器以最高的渲染级别渲染页面 --&gt;
6      &lt;meta http-equiv="X-UA-Compatible" content="IE=edge"&gt;
7      <!-- 开启移动端的理想视口 --&gt;
8      &lt;meta name="viewport" content="width=device-width,initial-scale=1.0"&gt;
9      <!-- 配置页签图标 --&gt;
10     &lt;link rel="icon" href="<%= BASE_URL %&gt;favicon.ico"&gt;
11     <!-- 配置网页标题 --&gt;
12     &lt;title&gt;&lt;%= htmlWebpackPlugin.options.title %&gt;&lt;/title&gt;
13 &lt;/head&gt;
14 &lt;body&gt;
15     <!-- 当浏览器不支持js时noscript中的元素就会被渲染 --&gt;
16     &lt;noscript&gt;
17         &lt;strong&gt;We're sorry but &lt;%= htmlWebpackPlugin.options.title %&gt; doesn't work properly without JavaScript&lt;/strong&gt;
18     &lt;/noscript&gt;
19     <!-- 容器 --&gt;
20     &lt;div id="app"&gt;&lt;/div&gt;
21     <!-- built files will be auto injected --&gt;
22 &lt;/body&gt;
23 &lt;/html&gt;</pre>
```

```
src > main.js > ...
3  */
4 //引入Vue
5 import Vue from 'vue'
6 //引入App组件，它是所有组件的父组件
7 import App from './App.vue'
8 //关闭vue的生产提示
9 Vue.config.productionTip = false
10
11 //创建Vue实例对象---vm
12 new Vue({
13   el: '#app',
14   //下面这行代码一会解释，完成了这个功能：将App组件放入容器中
15   render: h => h(App),
16   // render:h=> q('h1','你好啊')
17
18   // template:`<h1>你好啊</h1>`,
19   // components:{App},
20 })
```

因为引入了残缺板的vue所以要使用render
函数将APP组件放入到容器中

```
/*
```

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。

```
*/
```

```
28 ## vue.config.js配置文件
```

```
29 > 使用vue inspect > output.js可以查看到Vue脚手架的默认配置。
```

```
30 > 使用vue.config.js可以对脚手架进行个性化定制，详情见：https://cli.vuejs.org/zh
```

ref属性

```
Src > App.vue > App.vue > script > default > methods > showDOM
```

```
1 <template>
2   <div>
3     <h1 v-text="msg" ref="title"></h1>✓
4     <button ref="btn" @click="showDOM">点我输出上方的DOM元素</button>
5     <School ref="sch"/>
6   </div>
7 </template>
8
9 <script>
10 //引入School组件
11 import School from './components/School'
12
13 export default {
14   name:'App',
15   components:{School},
16   data() {
17     return {
18       msg:'欢迎学习Vue!'
19     }
20   },
21   methods: {
22     showDOM(){
23       console.log(this.$refs)
24     }
25   },
26 }
```

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属性

转到(G) 运行(R) 调试(T) 帮助(H) App.vue - vue_test - Visual Studio Code

App.vue M X Student.vue U

```
src > App.vue > {} "App.vue" > template > div > Student
1 <template>
2   <div>
3     <Student name="李四" sex="女" age="18"/>
4     <Student name="王老五" sex="男" age="19"/>
5   </div>
6 </template>
7
8 <script>
9   import Student from './components/Student'
10
11  export default {
12    name: 'App',
13    components: {Student}
14  }
15 </script>
16
```

终端 问题 输出 调试控制台 1: node + -

App.vue M Student.vue U

```
src > components > Student.vue > {} "Student.vue" > template
1 <template>
2   <div>
3     <h1>{{msg}}</h1>
4     <h2>学生姓名: {{name}}</h2>
5     <h2>学生性别: {{sex}}</h2>
6     <h2>学生年龄: {{age}}</h2>
7   </div>
8 </template>
9
10 <script>
11   export default {
12     name: 'Student',
13     data() {
14       return {
15         msg: '我是一个尚硅谷的学生',
16       }
17     },
18     props: ['name', 'age', 'sex']
19   }
20 </script>
```

```
App.vue > {} "App.vue" > template > div > Student
1 <template>
2   <div>
3     <Student name="李四" sex="女" :age="18"/>           ↓
4   </div>
5 </template>
6
7 <script>
8   import Student from './components/Student'
9
10 export default {
11   name:'App',
12   components:{Student}
13 }
14 </script>
15
```

这里的18就是数字18

```
// props:['name','age','sex']
```

```
props: [
  name:String,
  age:Number,
  sex:String
]
```

```

27     } */
28
29     props:{
30         name:{
31             type:String, //name的类型是字符串
32             required:true, //name是必要的
33         },
34         age:{
35             type:Number,
36             default:99 //默认值
37         },
38         sex:{
39             type:String,
40             required:true
41         }
42     }
43 }

## 配置项props
功能：让组件接收外部传过来的数据
(1).传递数据：
<Demo name="xxx"/>
(2).接收数据：
第一种方式（只接收）：
props:['name']

第二种方式（限制类型）：
props:{
    name:Number
}

第三种方式（限制类型、限制必要性、指定默认值）：
props:{
    name:{
        type:String, //类型
        required:true, //必要性
        default:'老王' //默认值
    }
}
备注：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。

```

Mixin混入

两个组件共享一个配置

```
src > mixin.js > ...
1 | export const hunhe = {
2 |   methods: {
3 |     showName(){
4 |       alert(this.name)
5 |     }
6 |   },
7 | }
8
```

```
<script>
//引入一个hunhe
import {hunhe} from '../mixin'

export default {
  name: 'School',
  data() {
    return {
      name: '尚硅谷',
      address: '北京'
    }
  },
  mixins: [hunhe]
}
</script>
```

功能：可以把多个组件共用的配置提取成一个混入对象
使用方式：

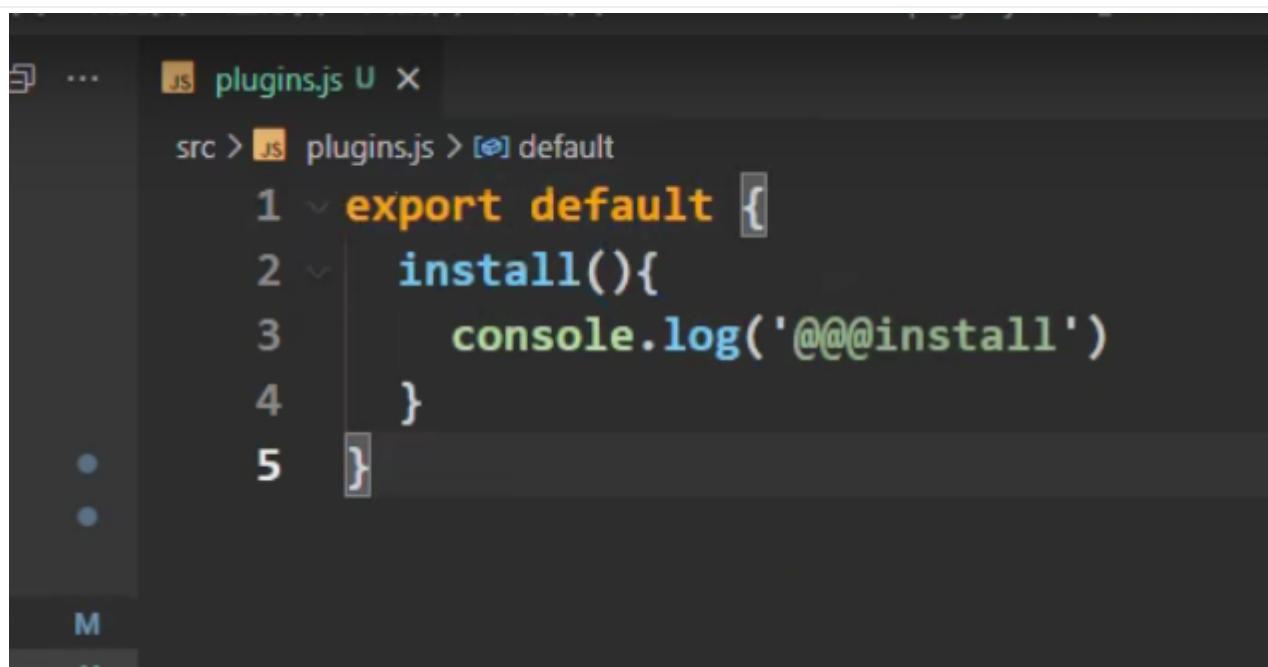
第一步定义混合，例如：

```
{  
  data(){....},  
  methods:{....}  
  ....  
}
```

第二步使用混入，例如：

- (1).全局混入: `Vue.mixin(xxx)`
- (2).局部混入: `mixins:['xxx']`

插件



```
... plugins.js U X  
src > plugins.js > [e] default  
1  export default {  
2    install(){  
3      console.log('@@@install')  
4    }  
5  }
```

```
main.js > ...
//引入Vue
import Vue from 'vue'
//引入App
import App from './App.vue'
//引入插件
import plugins from './plugins'
//关闭Vue的生产提示
Vue.config.productionTip = false

//应用(使用)插件
Vue.use(plugins)
//创建vm
new Vue({
  el: '#app',
  render: h => h(App)
})
```

```
## 插件
功能: 用于增强Vue
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。
定义插件:
对象.install = function (Vue, options) {
  // 1. 添加全局过滤器
  Vue.filter(...)

  // 2. 添加全局指令
  Vue.directive(...)

  // 3. 配置全局混入(合)
  Vue.mixin(...)

  // 4. 添加实例方法
  Vue.prototype.$myMethod = function () {...}
  Vue.prototype.$myProperty = xxxx
}

使用插件: Vue.use()
```

scoped样式

作用：让样式在局部生效，防止冲突。

写法：`<style scoped>`

localStorage

```
<body>
  <h2>localStorage</h2>
  <button onclick="saveData()">点我保存一个数据</button>
  <button onclick="saveData()">点我保存一个数据</button>

  <script type="text/javascript" >
    let p = {name:'张三',age:18}

    function saveData(){
      localStorage.setItem('msg','hello!!!')
      localStorage.setItem('msg2',666)
      localStorage.setItem('person',JSON.stringify(p))
    }
  </script>
</body>
```

1. 存储内容大小一般支持5MB左右（不同浏览器可能还不一样）
2. 浏览器端通过 `Window.sessionStorage` 和 `Window.localStorage` 属性来实现本地存储机制。
3. 相关API:

1. `xxxxxStorage.setItem('key', 'value');`
该方法接受一个键和值作为参数，会把键值对添加到存储中，如果键名存在，则更新其对应的值。
2. `xxxxxStorage.getItem('person');`
该方法接受一个键名作为参数，返回键名对应的值。
3. `xxxxxStorage.removeItem('key');`
该方法接受一个键名作为参数，并把该键名从存储中删除。
4. `xxxxxStorage.clear()`
该方法会清空存储中的所有数据。

4. 备注:
 1. `SessionStorage`存储的内容会随着浏览器窗口关闭而消失。
 2. `LocalStorage`存储的内容，需要手动清除才会消失。
 3. `xxxxxStorage.getItem(xxx)` 如果xxx对应的value获取不到，那么getItem的返回值是null。
 4. `JSON.parse(null)` 的结果依然是null。

组件的自定义事件

```
3   <h1>{{msg}}</h1>
4
5   <!-- 通过父组件给子组件传递函数类型的props实现：子给父传递数据 -->
6   <School :getSchoolName="getSchoolName"/>
7
8   <!-- 通过父组件给子组件绑定一个自定义事件实现：子给父传递数据（第一种写法，使用@或v-on） -->
9   <Student @atguigu="getStudentName"/>
10
11  <!-- 通过父组件给子组件绑定一个自定义事件实现：子给父传递数据（第二种写法，使用ref） -->
12  <!-- <Student ref="student"/> -->
13 </div>
```

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**，否则`this`指向会出现问题！

全局事件总线

找到(G) 运行(R) 终端(T) 帮助(H) • main.js - vue_test - Visual Studio Code

```
main.js M ● School.vue U Student.vue U
src > main.js > beforeCreate > Vue > x
1 //引入Vue
2 import Vue from 'vue'
3 //引入App
4 import App from './App.vue'
5 //关闭Vue的生产提示
6 Vue.config.productionTip = false
7
8 //创建vm
9 new Vue({
10   el:'#app',
11   render: h => h(App),
12   beforeCreate() {
13     Vue.prototype.$x = this //安装全局事件总线
14   },
15 })
```

消息订阅与发布

```
C:\Users\tianyu\Desktop\vue_test>npm i pubsub-js
```

```
7 <script>
8   import pubsub from 'pubsub-js'
9   export default {
10     name:'School',
11     data() {
12       return {
13         name:'尚硅谷',
14         address:'北京',
15       }
16     },
17   },
18   mounted() {
19     // console.log('School',this)
20     /* this.$bus.$on('hello',(data)=>{
21       console.log('我是School组件, 收到了数据',data)
22     }) */
23     pubsub.subscribe('hello',function(){
24       console.log('有人发布了hello消息, hello消息的回调执行了')
25     })
26   },
27   beforeDestroy() {
28     // this.$bus.$off('hello')
29   },
30 }
31 </script>
32 <script>
33   import pubsub from 'pubsub-js'
34   export default {
35     name:'School',
36     data() {
37       return {
38         name:'尚硅谷',
39         address:'北京',
40       }
41     },
42   },
43   mounted() {
44     // console.log('School',this)
45     /* this.$bus.$on('hello',(data)=>{
46       console.log('我是School组件, 收到了数据',data)
47     }) */
48     pubsub.subscribe('hello',function(a,b){↑ 订阅名
49       console.log('有人发布了hello消息, hello消息的回调执行了',a,b)
50     })
51   },
52   beforeDestroy() {
53     // this.$bus.$off('hello')
54   },
55 }
```

消息订阅与发布 (pubsub)

1. 一种组件间通信的方式，适用于任意组件间通信。

2. 使用步骤：

1. 安装pubsub: `npm i pubsub-js`

2. 引入: `import pubsub from 'pubsub-js'`

3. 接收数据：A组件想接收数据，则在A组件中订阅消息，订阅的回调留在A组件自身。

```
methods(){
  demo(data){.....}
}
.....
mounted() {
  this.pid = pubsub.subscribe('xxx',this.demo) //订阅消息
}
```

4. 提供数据: `pubsub.publish('xxx',数据)`

5. 最好在beforeDestroy钩子中，用`PubSub.unsubscribe(pid)`去取消订阅。

nextTick

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。

动画效果

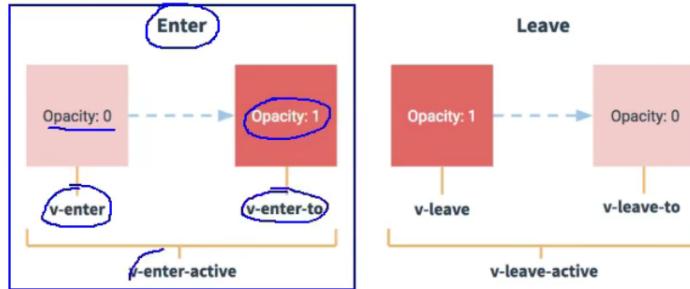
Vue封装的过度与动画

1. 作用：在插入、更新或移除 DOM元素时，在合适的时候给元素添加样式类名。

2. 图示：

Vue

文件



列

3. 写法：

件

us)

与动画

1. 准备好样式：

- 元素进入的样式：

1. `v-enter`: 进入的起点
2. `v-enter-active`: 进入过程中
3. `v-enter-to`: 进入的终点

- 元素离开的样式：

3. 写法：

1. 准备好样式：

- 元素进入的样式：

1. `v-enter`: 进入的起点
2. `v-enter-active`: 进入过程中
3. `v-enter-to`: 进入的终点

- 元素离开的样式：

1. `v-leave`: 离开的起点
2. `v-leave-active`: 离开过程中
3. `v-leave-to`: 离开的终点

2. 使用 `<transition>` 包裹要过度的元素，并配置name属性：

```
<transition name="hello">
  <h1 v-show="isShow">你好啊！</h1>
</transition>
```



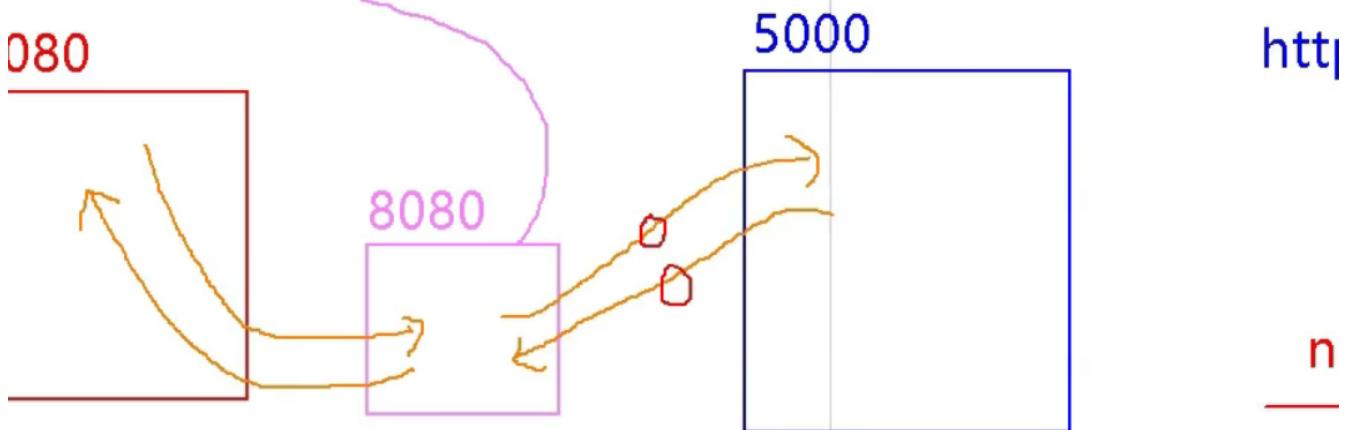
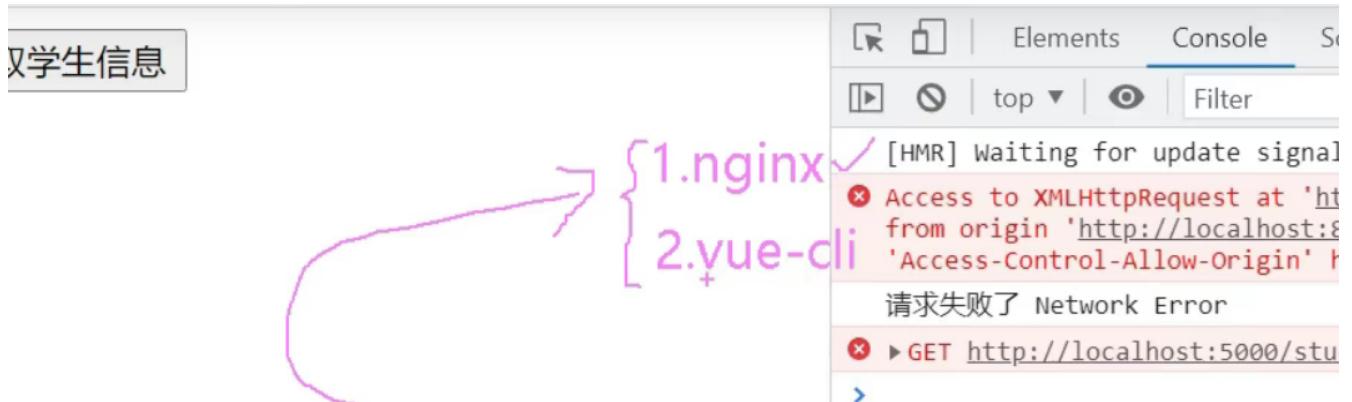
3. 备注：若有多个元素需要过度，则需要使用：`<transition-group>`，且每个元素都要指定 `key` 值。



配置代理方法一

```
C:\Users\tianyu\Desktop\vue_test>npm i axios
```

```
> App.vue > {} "App.vue" > script > default
1  <template>
2    <div>
3      <button @click="getStudents">获取学生信息</button>
4    </div>
5  </template>
6
7  <script>
8    import axios from 'axios'
9    export default {
10      name:'App',
11      methods: {
12        getStudents(){
13          axios.get('http://localhost:5000/students').then(
14            response => {
15              console.log('请求成功了', response.data)
16            },
17            error => {
18              console.log('请求失败了', error.message)
19            }
20          )
21        }
22      },
23    }
24  </script>
25
```



```
    },
    lintOnSave:false, //关闭语法检查
    //开启代理服务器
    devServer: {
      proxy: 'http://localhost:5000'
    }
}
```

配置代理方法二

```
2 }, /*  
3 //开启代理服务器（方式二）  
4 devServer: {  
5   proxy: {  
6     '/atguigu': {  
7       target: 'http://localhost:5000'  
8       // ws: true,  
9       // changeOrigin: true  
0     }  
1   }  
2 }  
3 }  
  
 axios.get('http://localhost:8080/atguigu/students').then(  
  response => {
```

总结代理方法

方法一

在 vue.config.js 中添加如下配置：

```
devServer:{  
  proxy:"http://localhost:5000"  
}
```

说明：

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）

文件

方法二

编写vue.config.js配置具体代理规则：

```
module.exports = {
  devServer: {
    proxy: {
      '/api1': { // 匹配所有以 '/api' 开头的请求路径
        target: 'http://localhost:5000', // 代理目标的基础路径
        changeOrigin: true,
        pathRewrite: {'^/api1': ''}
      },
      '/api2': { // 匹配所有以 '/api' 开头的请求路径
        target: 'http://localhost:5001', // 代理目标的基础路径
        changeOrigin: true,
        pathRewrite: {'^/api2': ''}
      }
    }
  }
}

/* 
  changeOrigin 设置为 true 时，服务器收到的请求头中的 host 为： localhost:5000
  changeOrigin 设置为 false 时，服务器收到的请求头中的 host 为： localhost:8080
  changeOrigin 默认值为 true
*/
}

/*
  changeOrigin 设置为 true 时，服务器收到的请求头中的 host 为： localhost:5000
  changeOrigin 设置为 false 时，服务器收到的请求头中的 host 为： localhost:8080
  changeOrigin 默认值为 true
*/
```

说明：

- 优点：可以配置多个代理，且可以灵活的控制请求是否走代理。
- 缺点：配置略微繁琐，请求资源时必须加前缀。

```
C:\Users\tianyu\Desktop\vue_test>npm i vue-resource
```

VUEX

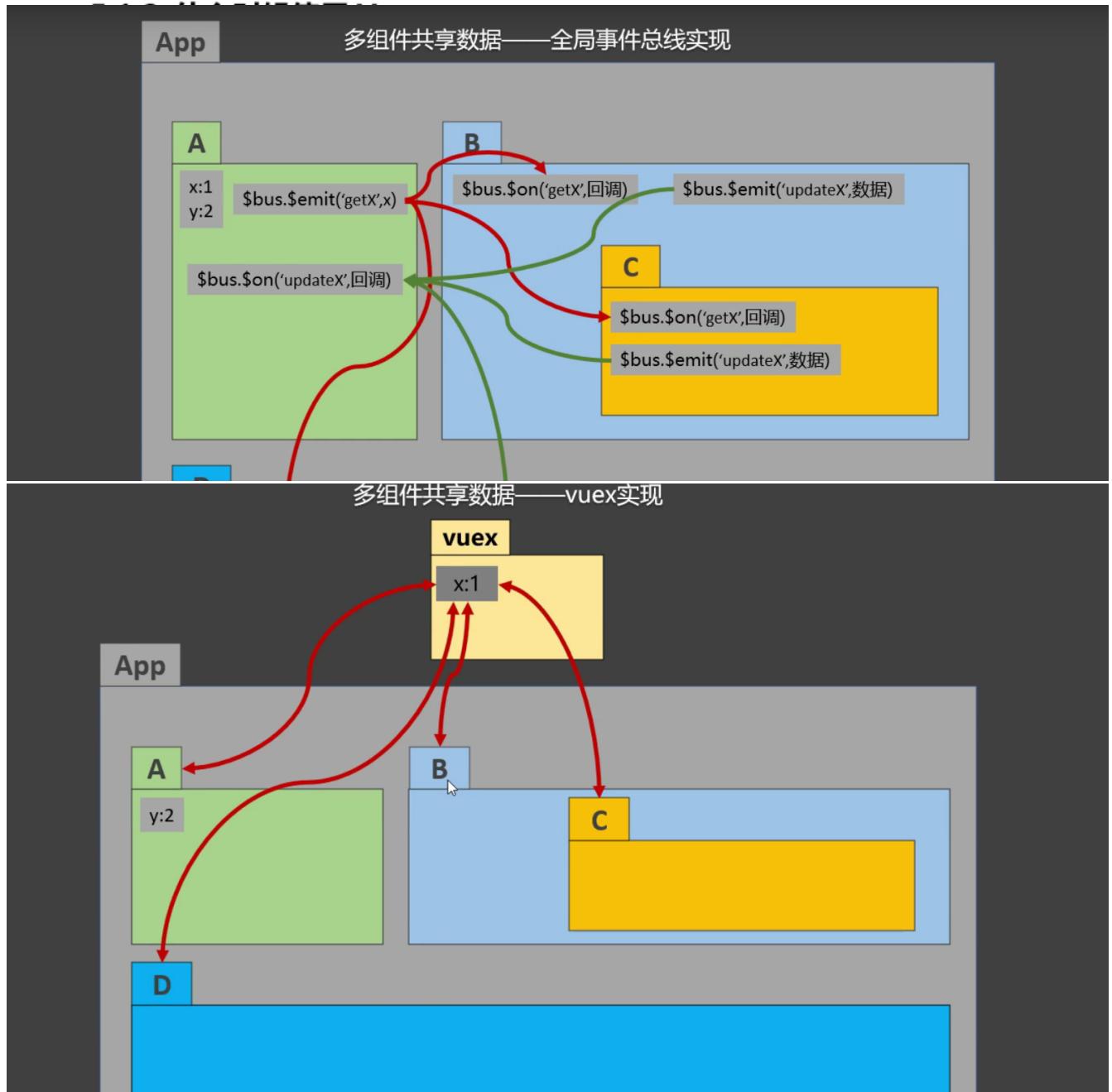
5.1 进阶 vuex ↵

5.1.1 vuex 是什么 ↵

- 概念：专门在 Vue 中实现集中式状态（数据）管理的一个 Vue 插件，对 vue 应用中

多个组件的共享状态进行集中式的管理（读/写），也是一种组件间通信的方式，且适用于任意组件间通信。↵

- Github 地址：<https://github.com/vuejs/vuex> ↵



```
1 //引入Vue
2 import Vue from 'vue'
3 //引入App
4 import App from './App.vue'
5 //引入插件
6 import vueResource from 'vue-resource'
7 //引入vuex
8 import Vuex from 'vuex'
9 //关闭Vue的生产提示
10 Vue.config.productionTip = false
11 //使用插件
12 Vue.use(vueResource)
13 | Vue.use(Vuex)
14
15 //创建vm
16 new Vue({
17   el: '#app',
18   render: h => h(App),
19 |   store: 'hello',
20   beforeCreate() {
21     Vue.prototype.$bus = this
22   }
23 })
24 | // console.log(vm)
```

![image-20211022003926993](C:\Users\ZY\AppData\Roaming\Typora\typora-user-images\image-20211022003926993.png)

```
JS index.js U ● JS main.js M
src > store > JS index.js > ...
1 //该文件用于创建Vuex中最为核心的store
2
3 //引入Vuex
4 import Vuex from 'vuex'
5 //准备actions—用于响应组件中的动作
6 const actions = {}
7 //准备mutations—用于操作数据 (state)
8 const mutations = {}
9 //准备state—用于存储数据
10 const state = {}
11
12 //创建store
13 const store = new Vuex.Store({
14   actions,
15   mutations,
16   state,
17 })
18
19 //store
20 export default store
```

```
2 import Vue from 'vue'
3 //引入Vuex
4 import Vuex from 'vuex'
5 //应用Vuex插件
6 Vue.use(Vuex)
7
8 //准备actions—用于响应组件中的动作
9 const actions = {
10   jia(context,value){
11     // console.log('actions中的jia被调用了',context,value)
12     context.commit('JIA',value)
13   }
14 }
15 //准备mutations—用于操作数据 (state)
16 const mutations = {
17   JIA(state,value){
18     // console.log('mutations中的JIA被调用了',state,value)
19     state.sum += value
20   }
21 }
22 //准备state—用于存储数据
23 const state = {
24   sum:0 //当前的和
25 }
26
27 //创建并暴露store
28 export default new Vuex.Store({
29   actions,
30   state,
31   mutations,
32   getters
33 })
34
35 //准备getters—用于将state中的数据进行加工
36 const getters = {
37   bigSum(state){
38     return state.sum*10
39   }
40 }
41
42 //创建并暴露store
43 export default new Vuex.Store({
44   actions,
45   state,
46   mutations,
47   getters
48 })
49
50 //准备actions—用于响应组件中的动作
51 const actions = {
52   jia(context,value){
53     // console.log('actions中的jia被调用了',context,value)
54     context.commit('JIA',value)
55   }
56 }
57
58 //准备mutations—用于操作数据 (state)
59 const mutations = {
60   JIA(state,value){
61     // console.log('mutations中的JIA被调用了',state,value)
62     state.sum += value
63   }
64 }
65
66 //准备state—用于存储数据
67 const state = {
68   sum:0 //当前的和
69 }
70
71 //创建并暴露store
72 export default new Vuex.Store({
73   actions,
74   state,
75   mutations,
76   getters
77 })
```

路由



1. 单页 Web 应用 (single page web application, SPA) 。 ↵
2. 整个应用只有**一个完整的页面**。 ↵
3. 点击页面中的导航链接**不会刷新**页面，只会做页面的**局部更新**。 ↵
4. 数据需要通过 ajax 请求获取。 ↵

2. key 为路径, value 可能是 function 或 component ↵

2. 路由分类 ↵

1. 后端路由: ↵



1) 理解: value 是 function, 用于处理客户端提交的请求。 ↵

2) 工作过程: 服务器接收到一个请求时, 根据**请求路径**找到匹配的**函数**来处理请求, 返回响应数据。 ↵

2. 前端路由: ↵

1) 理解: value 是 component, 用于展示页面内容。 ↵

2) 工作过程: 当浏览器的路径改变时, 对应的组件就会显示。 ↵

• 6.2 其他路由 ↵

vue UI组件库

· 第7章：Vue UI 组件库

· 7.1 移动端常用 UI 组件库

1. Vant <https://youzan.github.io/vant>
2. Cube UI <https://didi.github.io/cube-ui>
3. Mint UI <http://mint-ui.github.io>

CSS
JS

· 7.2 PC 端常用 UI 组件库

1. Element UI <https://element.eleme.cn>
2. IView UI <https://www.iviewui.com>

dockfile

```
age / docker / Dockerfile / ...
FROM 133.38.35.206/k8s_18883991817_15411-nginx/k8s_18883991817_15411-nginx:1.14.2-alpine
```

FROM: 指定基础镜像，必须为第一个命令



格式：

```
FROM <image>
FROM <image>:<tag>
FROM <image>@<digest>
```

示例：

```
FROM mysql:5.6
```

注：

tag或digest是可选的，如果不使用这两个值时，会使用latest版本的基础镜像



```
RUN rm -f /usr/share/nginx/html/index.html
```

 RUN用于在镜像容器中执行命令，其有以下两种命令执行方式：

shell执行

格式：
RUN <command>

exec执行

格式：
RUN ["executable", "param1", "param2"]

示例：

```
RUN ["executable", "param1", "param2"]
RUN apk update
RUN ["-/etc/execfile", "arg1", "arg1"]
```

注：

RUN指令创建的中间镜像会被缓存，并会在下次构建中使用。如果不使用这些缓存镜像，可以在构建时指定--no-cache参数，如：docker build --no-cache

ADD nginx.conf /etc/nginx/

ADD：将本地文件添加到容器中，tar类型文件会自动解压（网络压缩资源不会被解压），可以访问网络资源，类似wget

 格式：
ADD <src>... <dest>
ADD [<src>,... "<dest>"] 用于支持包含空格的路径

示例：

```
ADD hom* /mydir/          # 添加所有以"hom"开头的文件
ADD hom?.txt /mydir/       # ? 替代一个单字符,例如: "home.txt"
ADD test relativeDir/      # 添加 "test" 到 `WORKDIR`/relativeDir/
ADD test /absoluteDir/     # 添加 "test" 到 /absoluteDir/
```



copy

资源访问地址：<https://www.cnblogs.com/panwenbin-logs/p/8007348.html>

nginx.conf文件

```
#####Nginx配置文件nginx.conf中文详解#####
#定义Nginx运行的用户和用户组
user www www;
#nginx进程数，建议设置为等于CPU总核心数。
worker_processes 8;
#全局错误日志定义类型，[ debug | info | notice | warn | error | crit ]
error_log /usr/local/nginx/logs/error.log info;
#进程pid文件
pid /usr/local/nginx/logs/nginx.pid;
#指定进程可以打开的最大描述符：数目
#工作模式与连接数上限
#这个指令是指当一个nginx进程打开的最多文件描述符数目，理论值应该是最多打开文件数（ulimit -n）与nginx进程数相除，但是nginx分配请求并不是那么均匀，所以最好与ulimit -n 的值保持一致。
#现在在linux 2.6内核下开启文件打开数为65535，worker_rlimit_nofile就应该填写65535。
#这是因为nginx调度时分配请求到进程并不是那么的均衡，所以假如填写10240，总并发量达到3-4万时就有进程可能超过10240了，这时会返回502错误。
worker_rlimit_nofile 65535;
```

```
events
{
    #参考事件模型, use [ kqueue | rtsig | epoll | /dev/poll | select | poll ]; epoll模型
    #是Linux 2.6以上版本内核中的高性能网络I/O模型, linux建议epoll, 如果跑在FreeBSD上面, 就用kqueue模型。
    #补充说明:
    #与apache相类, nginx针对不同的操作系统, 有不同的事件模型
    #A) 标准事件模型
    #Select、poll属于标准事件模型, 如果当前系统不存在更有效的方法, nginx会选择select或poll
    #B) 高效事件模型
    #Kqueue: 使用于FreeBSD 4.1+, OpenBSD 2.9+, NetBSD 2.0 和 MacOS X. 使用双处理器的MacOS X系统使用kqueue可能会造成内核崩溃。
    #Epoll: 使用于Linux内核2.6版本及以后的系统。
    #/dev/poll: 使用于Solaris 7 11/99+, HP/UX 11.22+ (eventport), IRIX 6.5.15+ 和 Tru64 UNIX 5.1A+。
    #Eventport: 使用于Solaris 10。为了防止出现内核崩溃的问题, 有必要安装安全补丁。
    use epoll;
```

```
#单个进程最大连接数 (最大连接数=连接数*进程数)
#根据硬件调整, 和前面工作进程配合起来用, 尽量大, 但是别把cpu跑到100%就行。每个进程允许的最大连接数, 理论上每台nginx服务器的最大连接数为。
worker_connections 65535;

#keepalive超时时间。
keepalive_timeout 60;

#客户端请求头部的缓冲区大小。这个可以根据你的系统分页大小来设置, 一般一个请求头的大小不会超过1k, 不过由于一般系统分页都要大于1k, 所以这里设置为分页大小。
#分页大小可以用命令getconf PAGESIZE 取得。
#[root@web001 ~]# getconf PAGESIZE
#4096
#但也有client_header_buffer_size超过4k的情况, 但是client_header_buffer_size该值必须设置为“系统分页大小”的整倍数。
client_header_buffer_size 4k;

#这个将为打开文件指定缓存, 默认是没有启用的, max指定缓存数量, 建议和打开文件数一致, inactive是指经过多长时间文件没被请求后删除缓存。
open_file_cache max=65535 inactive=60s;

#这个是指多长时间检查一次缓存的有效信息。
#语法:open_file_cache_valid time 默认值:open_file_cache_valid 60 使用字段:http, server, location 这个指令指定了何时需要检查open_file_cache中缓存项目的有效信息。
open_file_cache_valid 80s;

#open_file_cache指令中的inactive参数时间内文件的最少使用次数, 如果超过这个数字, 文件描述符一直在缓存中打开的, 如上例, 如果有一个文件在inactive时间内一次没被使用, 它将被移除。
#语法:open_file_cache_min_uses number 默认值:open_file_cache_min_uses 1 使用字段:http, server, location 这个指令指定了在open_file_cache指令无效的参数中一定的时间范围内可以使用的最小文件数, 如果使用更大的值, 文件描述符在cache中总是打开状态。
open_file_cache_min_uses 1;
```

```
#语法:open_file_cache_errors on | off 默认值:open_file_cache_errors off 使用字段:http, server, location 这个指令指定是否在搜索一个文件是记录cache错误。
open_file_cache_errors on;

}

#设定http服务器，利用它的反向代理功能提供负载均衡支持
http
{
    #文件扩展名与文件类型映射表
    include mime.types;

    #默认文件类型
    default_type application/octet-stream;

    #默认编码
    charset utf-8;

    #服务器名字的hash表大小
    #保存服务器名字的hash表是由指令server_names_hash_max_size 和server_names_hash_bucket_size所控制的。参数hash bucket size总是等于hash表的大小，并且是一路处理器缓存大小的倍数。在减少了在内存中的存取次数后，使在处理器中加速查找hash表键值成为可能。如果hash bucket size等于一路处理器缓存的大小，那么在查找键的时候，最坏的情况下在内存中查找的次数为2。第一次是确定存储单元的地址，第二次是在存储单元中查找键 值。因此，如果Nginx给出需要增大hash max size 或 hash bucket size的提示，那么首要的是增大前一个参数的大小。
    server_names_hash_bucket_size 128;

    #客户端请求头部的缓冲区大小。这个可以根据你的系统分页大小来设置，一般一个请求的头部大小不会超过1k，不过由于一般系统分页都要大于1k，所以这里设置为分页大小。分页大小可以用命令getconf PAGESIZE取得。
    client_header_buffer_size 32k;

    #客户请求头缓冲大小。nginx默认会用client_header_buffer_size这个buffer来读取header值，如果header过大，它会使用large_client_header_buffers来读取。
    large_client_header_buffers 4 64k;

    #设定通过nginx上传文件的大小
    client_max_body_size 8m;

    #开启高效文件传输模式，sendfile指令指定nginx是否调用sendfile函数来输出文件，对于普通应用设为 on，如果用来进行下载等应用磁盘IO重负载应用，可设置为off，以平衡磁盘与网络I/O处理速度，降低系统的负载。注意：如果图片显示不正常把这个改成off。
    #sendfile指令指定 nginx 是否调用sendfile 函数 (zero copy 方式) 来输出文件，对于普通应用，必须设为on。如果用来进行下载等应用磁盘IO重负载应用，可设置为off，以平衡磁盘与网络I/O处理速度，降低系统uptime。
    sendfile on;

    #开启目录列表访问，合适下载服务器，默认关闭。
    autoindex on;

    #此选项允许或禁止使用socket的TCP_CORK的选项，此选项仅在使用sendfile的时候使用
    tcp_nopush on;

    tcp_nodelay on;
```

```

#长连接超时时间，单位是秒
keepalive_timeout 120;

#FastCGI相关参数是为了改善网站的性能：减少资源占用，提高访问速度。下面参数看字面意思都能理解。
fastcgi_connect_timeout 300;
fastcgi_send_timeout 300;
fastcgi_read_timeout 300;
fastcgi_buffer_size 64k;
fastcgi_buffers 4 64k;
fastcgi_busy_buffers_size 128k;
fastcgi_temp_file_write_size 128k;

#gzip模块设置
gzip on; #开启gzip压缩输出
gzip_min_length 1k;      #最小压缩文件大小
gzip_buffers 4 16k;      #压缩缓冲区
gzip_http_version 1.0;   #压缩版本（默认1.1，前端如果是squid2.5请使用1.0）
gzip_comp_level 2;       #压缩等级
gzip_types text/plain application/x-javascript text/css application/xml;      #压缩类型，默认就已经包含textml，所以下面就不用再写了，写上去也不会有问题，但是会有一个warn。
gzip_vary on;

#开启限制IP连接数的时候需要使用
#limit_zone crawler $binary_remote_addr 10m;

```

```

#负载均衡配置
upstream jh.w3cschool.cn {
    #upstream的权重，weight是权重，可以根据机器配置定义权重。weight参数表示权值，权值越高被分配到的几率越大。
    server 192.168.80.121:80 weight=3;
    server 192.168.80.122:80 weight=2;
    server 192.168.80.123:80 weight=3;

    #nginx的upstream目前支持4种方式的分配
    #1、轮询（默认）
    #每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器down掉，能自动剔除。
    #2、weight
    #指定轮询几率，weight和访问比率成正比，用于后端服务器性能不均的情况。
    #例如：
    #upstream bakend {
    #    server 192.168.0.14 weight=10;
    #    server 192.168.0.15 weight=10;
    #}
    #2、ip_hash
    #每个请求按访问ip的hash结果分配，这样每个访客固定访问一个后端服务器，可以解决session的问题。
    #例如：
    #upstream bakend {
    #    ip_hash;
    #    server 192.168.0.14:88;
    #    server 192.168.0.15:80;

```

```

    #}
    #3、fair (第三方)
    #按后端服务器的响应时间来分配请求，响应时间短的优先分配。
    #upstream backend {
    #    server server1;
    #    server server2;
    #    fair;
    #}
    #4、url_hash (第三方)
    #按访问url的hash结果来分配请求，使每个url定向到同一个后端服务器，后端服务器为缓存时比较有效。
    #例：在upstream中加入hash语句，server语句中不能写入weight等其他的参数，hash_method是使用的hash算
法
    #upstream backend {
    #    server squid1:3128;
    #    server squid2:3128;
    #    hash $request_uri;
    #    hash_method crc32;
    #}

#tips:
#upstream bakend{#定义负载均衡设备的Ip及设备状态}{
#    ip_hash;
#    server 127.0.0.1:9090 down;
#    server 127.0.0.1:8080 weight=2;
#    server 127.0.0.1:6060;
#    server 127.0.0.1:7070 backup;
#}
#在需要使用负载均衡的server中增加 proxy_pass http://bakend/;

#每个设备的状态设置为：
#1.down表示单前的server暂时不参与负载
#2.weight为weight越大，负载的权重就越大。
#3.max_fails: 允许请求失败的次数默认为1.当超过最大次数时，返回proxy_next_upstream模块定义的错误
#4.fail_timeout:maxfails次失败后，暂停的时间。
#5.backup: 其它所有的非backup机器down或者忙的时候，请求backup机器。所以这台机器压力会最轻。

#nginx支持同时设置多组的负载均衡，用来给不用的server来使用。
#client_body_in_file_only设置为on 可以将client post过来的数据记录到文件中用来做debug
#client_body_temp_path设置记录文件的目录 可以设置最多3层目录
#location对URL进行匹配.可以进行重定向或者进行新的代理 负载均衡
}

```

```

#虚拟主机的配置
server
{
    #监听端口
    listen 80;

    #域名可以有多个，用空格隔开
    server_name www.w3cschool.cn w3cschool.cn;

```

```
index index.html index.htm index.php;
root /data/www/w3cschool;
```

```
#对**进行负载均衡
location ~ .(php|php5)?$ {
    fastcgi_pass 127.0.0.1:9000;
    fastcgi_index index.php;
    include fastcgi.conf;
}
```

```
#图片缓存时间设置
location ~ .*.(gif|jpg|jpeg|png|bmp|swf)$ {
    expires 10d;
}

#JS和CSS缓存时间设置
location ~ .*.(js|css)?$ {
    expires 1h;
}

#日志格式设定
##$remote_addr与$http_x_forwarded_for用以记录客户端的ip地址;
##$remote_user: 用来记录客户端用户名;
##$time_local: 用来记录访问时间与时区;
##$request: 用来记录请求的url与http协议;
##$status: 用来记录请求状态; 成功是200,
##$body_bytes_sent : 记录发送给客户端文件主体内容大小;
##$http_referer: 用来记录从那个页面链接访问过来的;
##$http_user_agent: 记录客户浏览器的相关信息;
#通常web服务器放在反向代理的后面,这样就不能获取到客户的IP地址了,通过$remote_addr拿到的IP地址是反向代理服务器的IP地址。反向代理服务器在转发请求的http头信息中,可以增加x_forwarded_for信息,用以记录原有客户端的IP地址和原来客户端的请求的服务器地址。
log_format access '$remote_addr - $remote_user [$time_local] "$request" '
'$status $body_bytes_sent "$http_referer" '
'"$http_user_agent" $http_x_forwarded_for';

#定义本虚拟主机的访问日志
access_log  /usr/local/nginx/logs/host.access.log  main;
access_log  /usr/local/nginx/logs/host.access.404.log  log404;

#对 "/" 启用反向代理
location / {
    proxy_pass http://127.0.0.1:88;
    proxy_redirect off;
    proxy_set_header X-Real-IP $remote_addr;

    #后端的web服务器可以通过X-Forwarded-For获取用户真实IP
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

```
#以下是一些反向代理的配置，可选。
proxy_set_header Host $host;

#允许客户端请求的最大单文件字节数
client_max_body_size 10m;

#缓冲区代理缓冲用户端请求的最大字节数,
#如果把它设置为比较大的数值, 例如256k, 那么, 无论使用firefox还是IE浏览器, 来提交任意小于256k的图片, 都很正常。如果注释该指令, 使用默认的client_body_buffer_size设置, 也就是操作系统页面大小的两倍, 8k或者16k, 问题就出现了。
#无论使用firefox4.0还是IE8.0, 提交一个比较大, 200k左右的图片, 都返回500 Internal Server Error错误
client_body_buffer_size 128k;

#表示使nginx阻止HTTP应答代码为400或者更高的应答。
proxy_intercept_errors on;

#后端服务器连接的超时时间_发起握手等候响应超时时间
#nginx跟后端服务器连接超时时间(代理连接超时)
proxy_connect_timeout 90;

#后端服务器数据回传时间(代理发送超时)
#后端服务器数据回传时间_就是在规定时间之内后端服务器必须传完所有的数据
proxy_send_timeout 90;

#连接成功后, 后端服务器响应时间(代理接收超时)
#连接成功后_等候后端服务器响应时间_其实已经进入后端的排队之中等候处理 (也可以说是后端服务器处理请求的时间)
proxy_read_timeout 90;

#设置代理服务器 (nginx) 保存用户头信息的缓冲区大小
#设置从被代理服务器读取的第一部分应答的缓冲区大小, 通常情况下这部分应答中包含一个小的应答头, 默认情况下这个值的大小为指令proxy_buffers中指定的一个缓冲区的大小, 不过可以将其设置为更小
proxy_buffer_size 4k;

#proxy_buffers缓冲区, 网页平均在32k以下的设置
#设置用于读取应答 (来自被代理服务器) 的缓冲区数目和大小, 默认情况也为分页大小, 根据操作系统的不同可能是4k或者8k
proxy_buffers 4 32k;

#高负荷下缓冲大小 (proxy_buffers*2)
proxy_busy_buffers_size 64k;

#设置在写入proxy_temp_path时数据的大小, 预防一个工作进程在传递文件时阻塞太长
#设定缓存文件夹大小, 大于这个值, 将从upstream服务器传
proxy_temp_file_write_size 64k;
}

#设定查看Nginx状态的地址
location /NginxStatus {
    stub_status on;
```

```

access_log on;
auth_basic "NginxStatus";
auth_basic_user_file confpasswd;
#htpasswd文件的内容可以用apache提供的htpasswd工具来产生。
}

#本地动静分离反向代理配置
#所有jsp的页面均交由tomcat或resin处理
location ~ .(jsp|jspx|do)?$ {
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_pass http://127.0.0.1:8080;
}

#所有静态文件由nginx直接读取不经过tomcat或resin
location ~ .*.(htm|html|gif|jpg|jpeg|png|bmp|swf|ioc|rar|zip|txt|flv|mid|doc|ppt|
pdf|xls|mp3|wma)$ {
    expires 15d;
}

location ~ .*(js|css)?$ {
    expires 1h;
}
}

#####
##Nginx配置文件nginx.conf中文詳解#####

```

CreateEdgeInteractor.js

我们通过 ht.Default.def 自定义了 CreateEdgeInteractor 类，然后通过 graphView.setInteractors([new CreateEdgeInteractor(graphView, 'points')]) 这种方式来添加 graphView 拓扑图中的交互器，可以实现创建连线的交互功能。

在 CreateEdgeInteractor 类中通过监听 touchend 放手后事件向 graphView 拓扑图中添加一个 edge 连线，可以通过在 CreateEdgeInteractor 函数中传参来绘制不同的连线类型，比如 “ortho” 则为折线类型：

```

var CreateEdgeInteractor = function (graphView, type) {
    CreateEdgeInteractor.superClass.constructor.call(this, graphView);
    this._type = type;
};

ht.Default.def(CreateEdgeInteractor, DNDInteractor, { //自定义类，继承 DNDInteractor，此交互器有一些基本的交互功能
    handlewindowTouchEnd: function (e) {
        this.redraw();
        var isPoints = false;
    }
});

```

```

if(this._target){
    var edge = new ht.Edge(this._source, this._target); //创建一条连线，传入起始点和终点
    edge.s({
        'edge.type': this._type //设置连线类型 为传入的参数 type 类型 参考 HT for Web 连线
    });
    isPoints = this._type === 'points'; //如果没有设置则默认为 points 连线方式
    if(isPoints){
        edge.s({
            'edge.points': [
                {
                    'x: (this._source.p().x + this._target.p().x)/2,
                    'y: (this._source.p().y + this._target.p().y)/2
                }
            ]
        });
    }
    edge.setParent(this._graphView.getCurrentSubGraph()); //设置连线的父亲节点为当前子网
    this._graphView.getDataModel().add(edge); //将连线添加到拓扑图的数据容器中
    this._graphView.getSelectionModel().setSelection(edge); //设置选中该节点
}

}
this._graphview.removeTopPainter(this); //删除顶层Painter
if(isPoints){
    resetDefault(); //重置toolbar导航栏的状态
}
}
});

```

CreateNodeInteracto节点原生遍历

HT

你好，刚好昨天看到一篇博客，是讲这个东西。“ht是基于HTML5标准的企业应用图形界面一站式解决方案，其包含通用组件、拓扑组件和3D渲染引擎等丰富的图形界面开发类库，提供了完全基于HTML5的矢量编辑器、拓扑编辑器及3D场景编辑器等多套可视化设计工具，和完善的类库开发手册、工具使用手册、及针对HTML5技术如何进行大规模团队开发的客户深度培训手册。”摘自它的官网，本来想照着博客的样子也做一个类似的东西，不过后面发现这个东西并不开源，如果需要使用的话，需要（请发送邮件至service@hightopo.com（为了避免被当作垃圾邮件，请将邮件标题指定为：申请试用），注明您的称呼、公司名称、联系方式（邮箱及电话）等，我们会尽快联系您

作者：疯一般的小猪仔

链接：<https://www.zhihu.com/question/265185156/answer/343719496>

来源：知乎

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

一套丰富的 JavaScript 界面类库，

提供完整的基于 HTML5 图形界面组件库。使用 HT for Web 您可以轻松构建现代化的、跨桌面和移动终端的企业应用，无需担忧跨平台兼容性，及触屏手势交互等棘手问题。



一套监控可视化解决方案，

可用于快速创建和部署，高度可定制化，并具有强大交互功能的拓扑图形及仪表图表等应用。HT for Web 非常适用于实时监控系统的界面呈现，广泛应用于电信网络拓扑和设备管理，以及电力、燃气等工业自动化 (HMI/SCADA) 领域。



一套强大的基于 WebGL 技术的 3D 图形引擎。

HT for Web 提供了一套独特的 WebGL 层抽象，将 Model–View–Presenter (MVP) 的设计模型延伸应用到了 3D 图形领域。使用 HT for Web 您可更专注于业务逻辑功能，不必将精力投入复杂 3D 渲染和数学等非业务核心的技术细节。

hover

实例

选择鼠标指针浮动在其上的元素，并设置其样式：

```
a:hover
{
background-color:yellow;
}
```

Mixin

两个文件公用一个配置，提高代码的复用性

注意：如果被混合的地方有数据或者其他东西，那么混合的时候不会替换只会合并，如果你有的则不会破坏你的。

```
mixins: [hunhe]
import {hunhe} from '../mixin'
```

```
//关闭Vue的生产提示  
Vue.config.productionTip = false  
Vue.mixin(hunhe)          全局的混合  
Vue.mixin(hunhe2)
```

//创建vm

mixin(混入)

功能：可以把多个组件共用的配置提取成一个混入对象

使用方式：

第一步定义混合，例如：

```
{  
    data(){....},  
    methods:{....}  
    ....  
}
```

第二步使用混入，例如：

```
(1).全局混入: Vue.mixin(xxx)  
(2).局部混入: mixins: ['xxx']
```

插件Vue.use(install)

```
1 import Dict from './Dict'
2
3 const install = function(Vue) {
4   Vue.mixin({
5     data() {
6       if (this.$options dicts instanceof Array) {
7         const dict = {
8           dict: {},
9           label: {}
10      }
11      return {
12        dict
13      }
14    }
15    return {}
16  },
17  created() {
18    if (this.$options dicts instanceof Array) {
19      new Dict(this.dict).init(this.$options dicts, () => {
20        this.$nextTick(() => {
21          this.$emit('dictReady')
22        })
23      })
24    }
25  }
26}
27
28
```

插件

功能: 用于增强Vue

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。

定义插件:

```
对象.install = function (Vue, options) {
  // 1. 添加全局过滤器
  Vue.filter(...)

  // 2. 添加全局指令
  Vue.directive(...)

  // 3. 配置全局混入(合)
  Vue.mixin(...)

  // 4. 添加实例方法
  Vue.prototype.$myMethod = function () {...}
  Vue.prototype.$myProperty = xxxx
}
```

使用插件: `Vue.use()`

组件的简写方式

3. 一个简写方式:

```
const school = Vue.extend(options) 可简写为: const school = options
```

```
<!--
关于VueComponent:
1.school组件本质是一个名为VueComponent的构造函数，且不是程序员定义的，是Vue.extend生成的。
2.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)。
3.特别注意：每次调用Vue.extend，返回的都是一个全新的VueComponent！！！
4.关于this指向：
(1).组件配置中：
    data函数、methods中的函数、watch中的函数、computed中的函数 它们的this均是【VueComponent实例】
(2).new Vue()配置中：
    data函数、methods中的函数、watch中的函数、computed中的函数 它们的this均是【Vue实例对象】。
5.VueComponent的实例对象，以后简称vc（也可称之为：组件实例对象）。
    Vue的实例对象，以后简称vm。
-->
<!-- 准备好一个容器-->
```

vue组件的本质是一个vuecomponent构造函数。

```
关于VueComponent:
1.school组件本质是一个名为VueComponent的构造函数，且不是程序员定义的，是Vue.extend生成的。
2.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)。
3.特别注意：每次调用Vue.extend，返回的都是一个全新的VueComponent！！！
4.关于this指向：
(1).组件配置中：
    data函数、methods中的函数、watch中的函数、computed中的函数 它们的this均是【VueComponent实例】
(2).new Vue(options)配置中：
    data函数、methods中的函数、watch中的函数、computed中的函数 它们的this均是【Vue实例对象】。
5.VueComponent的实例对象，以后简称vc（也可称之为：组件实例对象）。
    Vue的实例对象，以后简称vm。
-->
<!-- 准备好一个容器-->
<body>
0   <!--
1     1.一个重要的内置关系：VueComponent.prototype.__proto__ === Vue.prototype
2     2.为什么要有这个关系：让组件实例对象（vc）可以访问到 Vue原型上的属性、方法。
3   -->
```

render函数

使用render函数的原因是因为在脚手架中是不完整版的没有模板解析器需要使用render函数代替模板解析器。

```
13
14   1.vue.js与vue.runtime.xxx.js的区别：
15     (1).vue.js是完整版的Vue，包含：核心功能+模板解析器。
16     (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。
17
18   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
19     render函数接收到的createElement函数去指定具体内容。
20   */
21
```

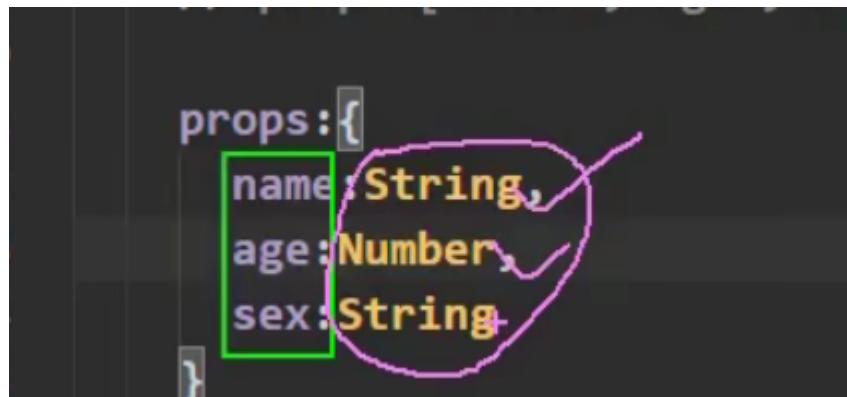
ref

id的替代者，可以给组件打标识

props

```
<div>
  <Student name="李四" sex="女" :age="18+1"/>
```

在组件传值的时候加冒号后传的值是引号里面的表达式



对传入的值进行限制

scoped

所有组件的样式到最后都会汇总到一起

```
<style scoped> .demo{
  background-color: skyblue;
}
</style>
```

写了scoped后就标明其写的样式只适用于此组件中

路由

```
) 帮助(H)
import VueRouter from 'vue-router'
//引入Luyou 组件
import About from '../components/About'
import Home from '../components/Home'

//创建router实例对象，去管理一组一组的路由规则
const router = new VueRouter({
  routes:[
    {
      path:'/about',
      component:About
    },
    {
      path:'/home',
      component:Home
    }
  ]
})

//暴露router
export default router
```

js

4. 实现切换 (active-class可配置高亮样式)

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

英 鸟 日

I

5. 指定展示位置

```
<router-view></router-view>
```



路由传值

```
<ul>
  <li v-for="m in messageList" :key="m.id">
    <router-link :to="`/home/message/detail?id=${m.id}&title=${m.title}`">{{m.title}}</router-link>&ampnbsp&ampnbsp
  </li>
</ul>
<hr>
<router-view></router-view>
</div>
</template>

<script>
  export default {
    name:'Message',
    data() {
      return {
        messageList:[
          {id:'001',title:'消息001'},
          {id:'002',title:'消息002'},
          {id:'003',title:'消息003'}
        ]
      }
    }
  }

```

第二种写法

```
<li v-for="m in messageList" :key="m.id">
  <!-- 跳转路由并携带query参数, to的字符串写法 -->
  <!-- <router-link :to="/home/message/detail?id=${m.id}&title=${m.title}">{{m.title}}</router-link>&nb
  <!-- 跳转路由并携带query参数, to的对象写法 -->
  <router-link :to="{
    path: '/home/message/detail',
    query: {
      id:m.id,
      title:m.title
    }
  }">
    {{m.title}}
  </router-link>
```

(V) 主题(T) 帮助(H)

4. 路由的query参数

1. 传递参数

```
<!-- 跳转并携带query参数, to的字符串写法 -->
<router-link :to="/home/message/detail?id=666&title=你好">跳转</router-link>

<!-- 跳转并携带query参数, to的对象写法 -->
<router-link
  :to="{
    path: '/home/message/detail',
    query: {
      id:666,
      title:'你好'
    }
  }"
>跳转</router-link>
```

2. 接收参数:

```
$route.query.id
$route.query.title
```

命名路由

```
export default new VueRouter({
  routes:[
    {
      name:'guanyu',
      path:'/about',
      component:About
    },
    {
      path: '/home',
      component:Home,
      children:[
        {
          path: 'news',
          component:News,
        },
        {
          path: 'message',
          component:Message,
          children:[
            {
              name:'xiangqing'
            }
          ]
        }
      ]
    }
  ]
})
```

```

7
8      <!-- 跳转路由并携带query参数, to的对象写法 -->
9      <router-link :to="{
10         name:'xiangqing',
11         query:{
12             id:m.id,
13             title:m.title
14         }
15     }">
16         {{m.title}}
17     </router-link>
18
19     </li>
20     </ul>
21     <hr>
22     <router-view></router-view>
23   </div>
24 </template>
25
26 <script>
27   export default {
28     name:'Message',

```

5. 命名路由

1. 作用: 可以简化路由的跳转。

2. 如何使用

1. 给路由命名:

```
{
  path:'/demo',
  component:Demo,
  children:[
    {
      path:'test',
      component:Test,
      children:[
        {
          name:'hello' //给路由命名
          path:'welcome',
          component:Hello,
        }
      ]
    }
  ]
}
```

2. 简化跳转:

```
<!-- 简化前, 需要写完整的路径 -->
<router-link to="/demo/test/welcome">跳转</router-link>

<!-- 简化后, 直接通过名字跳转 -->
<router-link :to="{name:'hello'}">跳转</router-link>
```

2. 简化跳转:

```
<!-- 简化前, 需要写完整的路径 -->
<router-link to="/demo/test/welcome">跳转</router-link>

<!-- 简化后, 直接通过名字跳转 -->
<router-link :to="{name:'hello'}">跳转</router-link>

<!-- 简化写法配合传递参数 -->
<router-link
  :to="{
    name:'hello',
    query:{
      id:666,
      title:'你好'
  }
}>
```

params参数

```
3   <ul>
4     <li v-for="m in messageList" :key="m.id">
5       <!-- 跳转路由并携带params参数, to的字符串写法 -->
6       <router-link :to="`/home/message/detail/${m.id}/你好啊`">{{m.title}}</router-link>&ampnbsp&nbs
7
8       <!-- 跳转路由并携带query参数, to的对象写法 -->
9       <!-- <router-link :to="{
10         name:'xiangqing',
11         query:{
12           id:m.id,
13           title:m.title
14       }"
15       >
16         children:[
17           {
18             name:'xiangqing',
19             path:'detail/:id/:title',
20             component:Detail,
21           }
22         ]
23       </router-link>
24     </li>
25   </ul>
```

6. 路由的params参数

1. 配置路由，声明接收params参数

```
{  
  path: '/home',  
  component: Home,  
  children:[  
    {  
      path: 'news',  
      component: News  
    },  
    {  
      component: Message,  
      children:[  
        {  
          name: 'xiangqing',  
          path: 'detail/:id/:title', // 使用占位符声明接收params参数  
          component: Detail  
        }  
      ]  
    }  
  ]  
}
```

2. 传递参数

2. 传递参数

```
<!-- 跳转并携带params参数，to的字符串写法 -->  
<router-link :to="/home/message/detail/666/你好">跳转</router-link>  
  
<!-- 跳转并携带params参数，to的对象写法 -->  
<router-link  
  :to="{  
    name: 'xiangqing',  
    params:{  
      id:666,  
      title:'你好'  
    }  
}">跳转</router-link>
```

特别注意：路由携带params参数时，若使用to的对象写法，则不能使用path配置项，必须使用name配置！

路由的props

The screenshot shows a code editor window in Visual Studio Code with a dark theme. The file being edited is 'index.js' located at 'src/router/index.js'. The code defines a router configuration:

```
16 component:About
17
18
19 path: '/home',
20 component:Home,
21 children:[
22   {
23     path:'news',
24     component:News,
25   },
26   {
27     path:'message',
28     component:Message,
29     children:[
30       {
31         name:'xiangqing',
32         path:'detail/:id/:title',
33         component:Detail,
34       }
35     //props的第一种写法，值为对象，该对象中的所有key-value都会以props的形式传给Detail组件。
36     // props:{a:1,b:'hello'}
37   }
38   //props的第二种写法，值为布尔值，若布尔值为真，就会把该路由组件收到的所有params参数，以props的形式传给Detail组件。
39   props:true
40 ]
41 ]
42 }
43 }
```

Line 38 contains a note: //props的第二种写法，值为布尔值，若布尔值为真，就会把该路由组件收到的所有params参数，以props的形式传给Detail组件。

//props的第三种写法，值为函数
props({query:{id,title}}){
 return {id,title}
}

7. 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  
    }  
  }  
}
```

js

router-link的replace属性

```
12  
13      <!-- Vue中借助router-link标签实现路由的切换 -->  
14      <router-link replace class="list-group-item" active-class="active" to="/about">About</router-link>  
15      <router-link replace class="list-group-item" active-class="active" to="/home">Home</router-link>  
16      </div>  
17      </div>  
18      <div class="col-xs-6">  
19        <div class="panel">  
20          <div class="panel-body">
```

8. <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>`

编程式路由导航

Vue Router Demo

About

Home

Home组件内容

News Message

- 消息001 push查看 replace查看
- 消息002 push查看 replace查看
- 消息003 push查看 replace查看

• 消息编号: 002
• 消息标题: 消息002

+

[HMR] Waiting for update signal from log.js:24

VueRouter {app: Vue, apps: Array(1), options: {...}, beforeHooks: Array(0), resolveHooks: Array(0), ...}

▶ afterHooks: []
▶ app: Vue {_uid: 0, _isVue: true, \$options: {...}, ...}
▶ apps: [Vue]
▶ beforeHooks: []
▶ fallback: false
▶ history: HashHistory {router: VueRouter, base: ...}
▶ matcher: {match: f, addRoute: f, getRoutes: f, ...}
▶ mode: "hash"
▶ options: {routes: Array(2)}
▶ resolveHooks: []
▶ currentRoute: (...)
▶ __proto__: Object

```
    },
  methods: {
    pushShow(m){
      this.$router.push({
        name: 'xiangqing',
        query: {
          id:m.id,
          title:m.title
        }
      })
    },
  },

```

```
12 <script>
13   export default {
14     name:'Banner',
15     methods: {
16       back(){
17         // this.$router.back()
18         console.log(this.$router)
19       },
20       forward(){
21         this.$router.forward()
22       },
23       test(){
24         this.$router.go(3)
25       }
26     },

```

1. 作用：不借助 `<router-link>` 实现路由跳转，让路由跳转更加灵活

2. 具体编码：

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

this.$router.replace({
  name:'xiangqing',
  params:{
    id:xxx,
    title:xxx
  }
})
this.$router.forward()
this.$router.back()
this.$router.go(3)
```

js

缓存路由组件

```
es > Home.vue > {} "Home.vue" > template > div > div > ul.nav.nav-tabs


- <router-link class="list-group-item" active-class="active" to="/home/news">News</router-link>
- <router-link class="list-group-item" active-class="active" to="/home/message">Message</router-link>


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

其中News是组件名

10.缓存路由组件

1. 作用：让不展示的路由组件保持挂载，不被销毁。

2. 具体编码：

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

vue

```
</ul>
<keep-alive :include="['News', 'Message']">
    <router-view></router-view>
</keep-alive>
</div>
```

路由专有的生命周期钩子

```
•      28     }, */
•      29     activated() {
•      30     |
•      31     },
•      32     deactivated() {
•      33     |
•      34     },
•      35     }
```

11.两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

```
router.beforeEach( guard: (to :Route , from :Route , next) => {
  if (to.meta.title) {
    document.title = to.meta.title + ' - ' + Config.title
  }
})
```

meta 路由元信息，程序员自定义的信息



```
{ path: '/wellSections/wellSection2D/:operationType/:specId/:id/:regionId/:label',
  component: (resolve) => require(['@rcc/views/qis/wellSections/wellSection2D']),
  name: '2D人手井截面图',
  meta: { title: '2D人手井截面图', needTranslate: true, noCache: false } }
```

后置路由守卫

```
//全局后置路由守卫——初始化的时候被调用、每次路由切换之后被调用
router.afterEach((to,from)=>{
  console.log('后置路由守卫',to,from)
})
```

独享路由守卫

```
c:\> router > index.js > router > routes > children > beforeEnter  
26     name:'xinwen',  
27     path:'news',  
28     component:News,  
29     meta:{isAuth:true,title:'新闻'},  
30     beforeEnter: (to, from, next) => {  
31         // ...  
32     }  
33 },  
34 {  
35     name:'xiaoxi',  
36     path:'message',  
37     component:Message,  
38     meta:{isAuth:true,title:'消息'},  
39     children:[ ...  
40     ]  
41 },  
42 ]  
43 ]
```

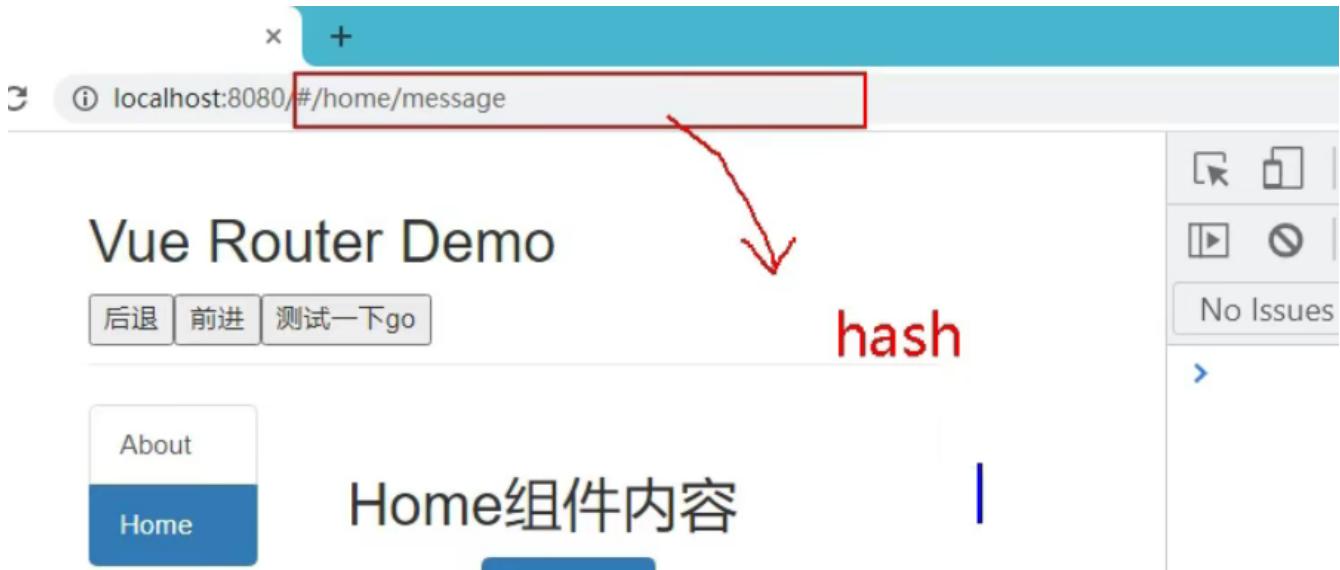
独享路由守卫只有前置，没有后置

组件内路由守卫

```
//通过路由规则，进入该组件时被调用  
beforeRouteEnter (to, from, next) {  
    // ...  
},
```

```
//通过路由规则，离开该组件时被调用  
beforeRouteLeave (to, from, next) {  
    // ...
```

history模式与hash模式



hash值不会随着http请求发给服务器、

hash有#，history没有#号

this.\$router.go()、back、push、replace

this.\$router.go(-1)

原页面表单中的内容会丢失；

向前或者向后跳转n个页面，n可为正整数或负整数

this.\$router.go(-1): 后退+刷新

this.\$router.go(0): 刷新；

this.\$router.go(1) : 前进

this.\$router.back()

原页表单中的内容会保留

this.\$router.back():后退；

this.\$router.back(0) 刷新；

this.\$router.back(1): 前进

this.\$router.push

跳转到指定url路径，并在history栈中添加一个记录，点击后退会返回到上一个页面

1. 不带参数

this.\$router.push('/home')

this.\$router.push({name:'home'})

this.\$router.push({path:'/home'})

2. query传参

this.\$router.push({name:'home',query: {id:'1'}})

this.\$router.push({path:'/home',query: {id:'1'}})

html 取参 \$route.query.id

script 取参 this.\$route.query.id

3. params传参

```
this.$router.push({name:'home',params: {id:'1'}}) // 只能用 name  
路由配置 path: "/home/:id" 或者 path: "/home:id",  
不配置path ,第一次可请求,刷新页面id会消失  
配置path,刷新页面id会保留  
html 取参 $route.params.id  
script 取参 this.$route.params.id
```

4. query和params区别

query类似 get, 跳转之后页面 url后面会拼接参数,类似?id=1, 非重要性的可以这样传,
params类似 post, 跳转之后页面 url后面不会拼接参数 ,但是刷新页面id 会消失,密码之类还是用params刷新页面
this.\$router.replace
跳转到指定url路径, 但是history栈中不会有记录, 点击返回会跳转到上上个页面 (就是直接替换了当前页面) 【A----->B----->C 结果B被C替换 A----->C】

版权声明：本文为CSDN博主「半生过往」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文地址：<https://blog.csdn.net/estrusKing/article/details/123675632>

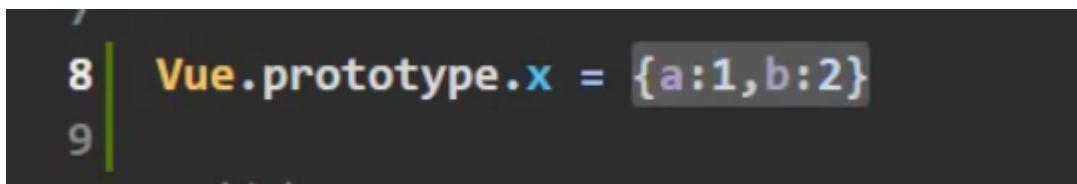
\$nextTick

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。

全局事件通讯

任意组件之间通信



```
8 |  Vue.prototype.x = {a:1,b:2}
9 |
```

A screenshot of a code editor showing a line of code. The line is `Vue.prototype.x = {a:1,b:2}`. The number '8' is to the left of the first '|'. The number '9' is to the left of the second '|'. The code is written in a light-colored font on a dark background.

登录验证码login.js

```
login.js
```

rcc-page > src > comm > api > **JS** login.js > ...

```
1 import request from '@comm/utils/request'
2
3 √ export function login(loginName, loginPwd, code, uuid)
4 √   return request({
5     url: 'auth/login',
6     method: 'post',
7     data: {
8       loginName,
9       loginPwd,
10      code,
11      uuid
12    }
13  })
14}
15
16 √ export function getInfo() {
17 √   return request({
18     url: 'auth/info',
19     method: 'get'
20   })
21 }
22
23 √ export function getCodeImg() {
24 √   return request({
25     url: 'auth/code',
26     method: 'get'
27 })
```

The screenshot shows a code editor interface with the following details:

- EXPLORER** sidebar:
 - OPEN EDITORS**: login.vue (rcc-page|src\system... 1)
 - RCC-PAGE**: mixins, plugins, router, store, utils, views (components, dashboard, features, nested, outerChain, props, system), example.vue, home.vue, login.vue (highlighted with a red border), App.vue, main.js, settings.js, static, .dockerignore, .editorconfig, .env.development, .env.production.
- login.vue** editor tab:

File path: rcc-page > src > system > views > login.vue > {} "login.vue" > template > div.login > el-form.login-form > h1

```
77 watch: {
78   $route: {
79     handler: function(route) {
80       this.redirect = route.query && route.query.redirect
81     },
82     immediate: true
83   }
84 },
85 created() {
86   // 获取验证码
87   this.getCode()
88   // 获取用户名密码等Cookie
89   this.getCookie()
90   // token 过期提示
91   this.point()
92   this.getDomainTitle()
93 },
94 methods: {
95   getCode() {
96     getCodeImg().then(res => {
97       this.codeUrl = res.img
98       this.loginForm.uuid = res.uuid
99     })
100   },
101   getCookie() {
102     const loginName = Cookies.get('loginName')
103     let loginPwd = Cookies.get('loginPwd')
104     const rememberMe = Cookies.get('rememberMe')
105     // 保存cookie里面的加密后的密码
106     this.cookiePass = loginPwd === undefined ? '' : loginPwd
107   }
108 }
```

A red box highlights the `getCodeImg()` function definition.

登录页面布局

1. 登录页面的布局

通过 Element-UI 组件实现布局

- el-form
- el-form-item
- el-input
- el-button
- 字体图标



```
PS C:\Users\iMac\Desktop\3.项目实战day1\code\vue_shop> git checkout -b login
Switched to a new branch 'login'
PS C:\Users\iMac\Desktop\3.项目实战day1\code\vue_shop>
```

git checkout -b login 创建字分支

路由中的懒加载

```
meta: { title: '登录', noCache: true },
component: (resolve) => require(['@system/views/login'], resolve),
```

require: 运行时调用，理论上可以运用在代码的任何地方，

import: 编译时调用，必须放在文件开头

懒加载: component: resolve => require('@/view/index.vue'], resolve)

用require这种方式引入的时候，会将你的component分别打包成不同的js，加载的时候也是按需加载，只用访问这个路由网址时才会加载这个js

非懒加载: component: index

如果用import引入的话，当项目打包时路由里的所有component都会打包在一个js中，造成进入首页时，需要加载的内容过多，时间相对比较长

vue的路由配置文件(routers.js)，一般使用import引入的写法，当项目打包时路由里的所有component都会被打包在一个js中，在项目刚进入首页的时候，就会加载所有的组件，所以导致首页加载较慢，而用require会将component分别打包成不同的js，按需加载，访问此路由时才会加载这个js，所以就避免进入首页时加载内容过多。

版权声明：本文为CSDN博主「余_小凡」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文地址：https://blog.csdn.net/weixin_37380784/article/details/99671933

路由重定向

```
path: '/',
component: layout,
redirect: '/dashboard',
path: 'attr/:specId', 使用占位符声明接收params参数
```

如果这里是/就重定向到/dashboard

CSS伪元素

/定义滚动条高宽及背景 高宽分别对应横竖滚动条的尺寸/

::-webkit-scrollbar

```
{  
width: 16px;  
height: 16px;  
background-color: #F5F5F5;  
}
```

/定义滚动条轨道 内阴影+圆角/

::-webkit-scrollbar-track

```
{  
-webkit-box-shadow: inset 0 0 6px rgba(0,0,0,0.3);  
border-radius: 10px;  
background-color: #F5F5F5;  
}
```

/定义滑块 内阴影+圆角/

::-webkit-scrollbar-thumb

```
{  
border-radius: 10px;  
-webkit-box-shadow: inset 0 0 6px rgba(0,0,0,.3);  
background-color: #555;  
}
```

Scss/Scass

Scss是一种兼容CSS并且比CSS功能更强大的一种样式

Scass中引入了变量，变量用\$来标识，这样使得样式属性可以被复用。

在scss中重复写样式是特别烦恼的事情

如下

```
#content article h1 { color: #333 }  
#content article p { margin-bottom: 1.4em }  
#content aside { background-color: #EEE }
```

面这种是SCSS采用的嵌套CSS方法来使得CSS样式中除去了冗余的部分

```
#content {  
  article {  
    h1 { color: #333 }  
    p { margin-bottom: 1.4em }  
  }  
  aside { background-color: #EEE }  
}
```

父选择器的标识符&

```
article a {  
  color: blue;  
  &:hover { color: red }  
}
```

用了&后被解析为

```
article a { color: blue }  
article a:hover { color: red }
```

群组选择器的嵌套

```
.container h1, .container h2, .container h3 { margin-bottom: .8em }
```

```
.container {  
  h1, h2, h3 {margin-bottom: .8em}  
}
```

子组合选择器和同层组合选择器：>、+和~;

嵌套属性

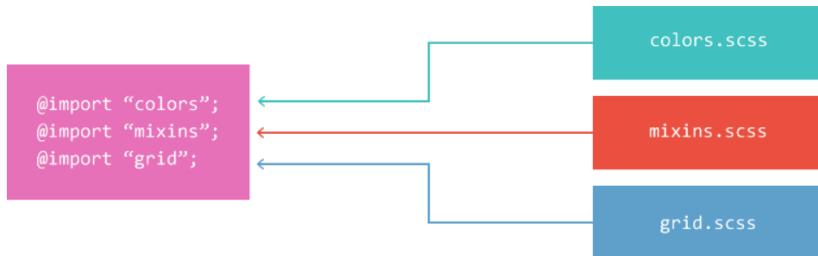
```
nav {  
  border: {  
    style: solid;  
    width: 1px;  
    color: #ccc;  
  }  
}
```

```
nav {  
  border: 1px solid #ccc {  
    left: 0px;  
    right: 0px;  
  }  
}
```

@import

使用 `sass` 的 `@import` 规则并不需要指明被导入文件的全名。你可以省略 `.sass` 或 `.scss` 文件后缀（见下图）。这样，在不修改样式表的前提下，你完全可以随意修改你或别人写的被导入的 `sass` 样式文件语法，在 `sass` 和 `scss` 语法之间随意切换。举例来说，

`@import "sidebar";` 这条命令将把 `sidebar.scss` 文件中所有样式添加到当前样式表中。



本节将介绍如何使用 `sass` 的 `@import` 来处理多个 `sass` 文件。首先，我们将学习编写那些被导入的 `sass` 文件，因为在一个大型 `sass` 项

`css` 有一个特别不常用的特性，即 `@import` 规则，它允许在一个 `css` 文件中导入其他 `css` 文件。然而，后果是只有执行到 `@import` 时，浏览器才会去下载其他 `css` 文件，这导致页面加载起来特别慢。

`sass` 也有一个 `@import` 规则，但不同的是，`sass` 的 `@import` 规则在生成 `css` 文件时就把相关文件导入进来。这意味着所有相关的样式被归纳到了同一个 `css` 文件中，而无需发起额外的下载请求。另外，所有在被导入文件中定义的变量和混合器（参见2.5节）均可在导入文件中使用。

@mixin

如果你的整个网站中有几处小小的样式类似（例如一致的颜色和字体），那么使用变量来统一处理这种情况是非常不错的选择。但是当你的样式变得越来越复杂，你需要大段大段的重用样式的代码，独立的变量就没办法应付这种情况了。你可以通过 `sass` 的混合器实现大段样式的重用。

混合器使用 `@mixin` 标识符定义。看上去很像其他的 `css` `@` 标识符，比如说 `@media` 或者 `@font-face`。这个标识符给一大段样式赋予一个名字，这样你就可以轻易地通过引用这个名字重用这段样式。下边的这段 `sass` 代码，定义了一个非常简单的混合器，目的是添加跨浏览器的圆角边框。

```
@mixin rounded-corners {  
  -moz-border-radius: 5px;  
  -webkit-border-radius: 5px;  
  border-radius: 5px;  
}
```

```
notice {  
  background-color: green;  
  border: 2px solid #00aa00;  
  @include rounded-corners;  
}
```

```
.notice {  
  background-color: green;  
  border: 2px solid #00aa00;  
  -moz-border-radius: 5px;  
  -webkit-border-radius: 5px;  
  border-radius: 5px;  
}
```

混入器中的CSS规则

```
@ mixin no-bullets {  
  list-style: none;  
  li {  
    list-style-image: none;  
    list-style-type: none;  
    margin-left: 0px;  
  }  
}
```

```
ul.plain {  
  color: #444;  
  @include no-bullets;  
}
```

```
ul.plain {  
  color: #444;  
  list-style: none;  
}  
ul.plain li {  
  list-style-image: none;  
  list-style-type: none;  
  margin-left: 0px;  
}
```

给混合器传参

混合器并不一定总得生成相同的样式。可以通过在 `@include` 混合器时给混合器传参，来定制混合器生成的精确样式。当 `@include` 混合器时，参数其实就是可以赋值给 `css` 属性值的变量。如果你写过 `JavaScript`，这种方式跟 `JavaScript` 的 `function` 很像：

```
@ mixin link-colors($normal, $hover, $visited) {  
  color: $normal;  
  &:hover { color: $hover; }  
  &:visited { color: $visited; }  
}
```

```
a {  
  @include link-colors(blue, red, green);  
}  
  
//Sass最终生成的是：  
  
a { color: blue; }  
a:hover { color: red; }  
a:visited { color: green; }
```

当你@include混合器时，有时候可能会很难区分每个参数是什么意思，参数之间是一个什么样的顺序。为了解决这个问题，`sass`允许通过语法`$name: value`的形式指定每个参数的值。这种形式的传参，参数顺序就不必再在乎了，只需要保证没有漏掉参数即可：

```
a {  
  @include link-colors(  
    $normal: blue,  
    $visited: green,  
    $hover: red  
  );  
}
```

before/after选择器

台词：我是唐老鸭。

台词：我住在 Duckburg。

台词：**注释：**对于在 IE8 中工作的`:before`，必须声明 DOCTYPE。

this.\$set(1,2,3)

- 1、对象
- 2、向对象中添加某个字段属性或向或个字段设置值
- 3、对应2字段的值

vue关闭当前路由返回上一页

```

back() {
  this.$store.state.tagsView.visitedViews.splice(this.$store.state.tagsView.visitedViews.findIndex(item => item.path === this.$route.path), 1)

  this.$router.push(this.$store.state.tagsView.visitedViews[this.$store.state.tagsView.visitedViews.length - 1].path)
}

```

Vue中导出到Excel时显示为科学计数法怎么解决

```

'二维码编码': {
  field: 'qrcode',
  callback: value => {
    return '&nbsp;' + value
  }
},

```

Z-Index

菜鸟上路有点区分不了z-index和! important的区别，所以就研究了一下：

z-index的用法就是：

样式中有position属性值为absolute、relative或baifixed导致了页面重叠把想显示的页面给遮住了，这个是后就可以设置z-index值（想显示在上面的值要设置比被遮盖的页面z-index值大）就可以解决了。

!important的用法是：

提高指定样式的优先级，有的时候你的样式是会被其他的样式给遮盖，这是因为你的样式权重不够，那这个时候有两个解决方法，

一：是通过选择器来添加权重，不过这个太过于麻烦，代码太长了。

二：可以在你想要的显示的样式后面加 ! important ，这样代码少简单方便

版权声明：本文为CSDN博主「干净洁」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。
原文链接：<https://blog.csdn.net/ganjingjie/article/details/106709256>

嵌套按钮

```

<el-dropdown trigger="click" placement>
  <el-button class="filter-item" type="primary" size="mini" icon="el-icon-download"
@click="editEquipment">
    导出
    <i class="el-icon-arrow-down el-icon--right" />
  </el-button>
  <el-dropdown-menu slot="dropdown" style="margin-left:30px">
    <el-dropdown-item>

```

```

<el-button
    v-if="operationType !== 'view'"
    id="portPanelviewBtn"
    :disabled="viewPortFlag"
    :loading="loading"
    class="filter-item"
    size="mini"
    type="primary"
    icon="el-icon-download"
    @click="viewPort($event)"
>
    导出光缆
</el-button>
<el-dropdown-item>
    <el-button
        v-if="operationType !== 'view'"
        id="portPanelviewBtn"
        :disabled="viewPortFlag"
        :loading="loading"
        class="filter-item"
        size="mini"
        type="primary"
        icon="el-icon-download"
        @click="viewPort($event)"
    >
        导出电缆
    </el-button>
</el-dropdown-item></el-dropdown-item></el-dropdown-menu>
</el-dropdown>

```

1. 0:

2. 1. **CODE**: "XSQ.081/GJ082060/PXG00015"
2. **CREATE_DATE**: "2016-05-31 15:57:19"
3. **ID**: "53100000000002277998661"
4. **MODIFY_DATE**: "2016-05-31 15:57:19"
5. **NAME**: "081/GJ082060/PXG00015"
6. **ORDER_NUM**: "100"
7. **RELATIONS**: {CREATE_DATE: '2022-04-23 03:09:12', CODE: "", MODIFY_DATE: '2022-04-23 03:09:12', SPEC_ID: '1210111210000', ID: '53100010000006589882923', ...}
8. **R_ID**: "53100010000006589882923"
9. **SPEC_ID**: "1211200002"
10. **SPEC_ID_NAME**: "光缆"
11. **TIME_STAMP**: "2022-04-23 02:42:51.52515"
12. **USING_STATE_ID**: "100723"
13. **USING_STATE_ID_CODE**: "3"
14. **USING_STATE_ID_ID**: "100723"
15. **USING_STATE_ID_NAME**: "占用"
16. **USING_STATE_ID_OBJECT**: {USING_STATE_ID-code: '3', USING_STATE_ID-name: '占用', USING_STATE_ID-id: '100723'}
17. [[Prototype]]: Object

el-tree默认展开第一级

```
<el-scrollbar :style="`height: fullHeight-290 +  
    <el-tree  
        ref="tree"  
        class="filter-tree"  
        :props="defaultProps"  
        :data="treeData"  
        node-key="id"  
        highlight-current  
        :expand-on-click-node="false"  
        :default-expanded-keys="treeExpandData"  
        :filter-node-method="filterNode"  
        @node-contextmenu="treeDataLeftClick"  
    />
```

JS中对对象数组

```
const removeDuplicateObj = (arr) => {  
    const obj = {}  
    arr = arr.reduce((newArr, next) => {  
        obj[next.entityId] ? '' : (obj[next.entityId] = true && newArr.push(next))  
        return newArr  
    }, [])  
    return arr  
}  
console.log('过滤后的数据', removeDuplicateObj(wellInfo))
```

JS中的排序函数

```
data.sort(this.sortBy('start_port'))
```