vue基础

Vue是一套渐近式的JavaScript框架，同时是一个用于开发SPA的应用型框架，它采用是MVVM的这一种开发思维，使用数据驱动页面

一、vue.js的使用

在一个项目当中如果要使用vue则必须要导入vue.js的框架，并且建立相应的托管区域。如下代码

```
<!DOCTYPE html>
<html lang="zh">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>vue的第一课</title>
</head>

<body>
    <div id="app">

    </div>
</body>
<script src="./js/vue.min.js"></script>
<script>
    new Vue({
        el: "#app"
    })
</script>
</html>
```

在上面的代码里面，我们在body当中新建了一个标签，并且设置`id="app"`，然后在后面导入了`vue.js`的框架，同时在JS代码当中创建了`vue`的对象，里面的`el:"#app"`代表的是vue框架要接管页面上面`id="app"`的区域

注意事项：接管区域不能是`body`标签，更不能是`html`标签

二、vue接管页面的数据

vue是一个数据驱动页面的框架，那么页面上面所有的数据应该都是由vue来提供的。在vue的框架里面，有很多地方都可以为我提供数据，其中最典型的就是`data`属性

```
new Vue({
    el:"#app",
    data:{
        userName:"软帝·杨标"
    }
})
```

当我们在`data`的属性里面定义了一个`userName`的这一个属性以后，页面的托管区域里面就可以使用这个数据，它这一种数据的使用如下

```
<h2>{{userName}}</h2>
```

打开网页以后，我们就发现这个userName的值已经显示在`<h2>`的标签里面【在这里，我们发现我们已经没有使用DOM操作了】。

在上面的这个数据渲染过程当中，我们官方的说法叫数据绑定【就是把vue的数据与页面进行绑定，既然是绑定，则vue改变页面也改变，同时如果页面改变则vue的数据也改变】

但是在Vue里面，数据绑定的试是有多种多样的，除了上面的花括号的绑定方式以外，我们还其它的几种绑定方式

三、vue的数据绑定方式

1. `{{}}`通过花括号这种方式绑定，这是一种最直观的绑定方式，这一种绑定方式它支持变量，表达式，可以调用方法，属性以及运算符

```
<div id="app">
    <h2>{{userName}}</h2>
    <h2>{{userName.length}}</h2>
    <h2>{{str1.toUpperCase()}}</h2>
    <h2>{{1+1}}</h2>
    <h2>{{sex==0?"男":"女"}}</h2>
    <h2>{{teacherInfo.teacherName}}</h2>
</div>
<script>
    new Vue({
        el: "#app",
        data: {
            userName: "标哥哥",
            str1: "hello world",
            sex: 0,
            teacherInfo: {
                teacherName: "张珊",
                teacherAge: 18
            }
        }
    })
</script>
```

2. `v-text`的数据绑定，这一种绑定方式与上面的绑定方式是相同的效果

```
<body>
    <div id="app">
        <div>{{htmlStr}}</div>
        <div v-text="htmlStr"></div>
        <div v-text="1+1"></div>
    </div>
</body>
<script src="./js/vue.js"></script>
<script>
    new Vue({
        el: "#app",
        data: {
            htmlStr: "<input type='text'>"
        }
    })
```

```
</script>
```

在上面的代码里面，我们运行以后发现这个绑定效果有一个缺点就是无法将html的字符串转换成标签【所以我们内部可以认为这两种绑定方式都是基于`innerText`来完成的】

3. `v-html`的数据绑定，这一种绑定方式可以理解为是基于`innerHTML`来进行数据绑定，如下

```
<body>
    <div id="app">
        <div>{{htmlStr}}</div>
        <div v-text="htmlStr"></div>
        <div v-html="htmlStr"></div>
    </div>
</body>
<script src="./js/vue.js"></script>
<script>
    new Vue({
        el: "#app",
        data: {
            htmlStr: "<input type='text'>"
        }
    })
</script>
```

<input type='text'>
<input type='text'>

4. `v-model`进行表单标签的值的绑定【它也是一个双向绑定的过程】

```
<body>
    <div id="app">
        <input type="text" v-model="userName">
        <h2>{{userName}}</h2>
        <hr>
        <label for="">
            <input type="radio" name="sex" value="男" v-model="sex">男
        </label>
        <label for="">
            <input type="radio" name="sex" value="女" v-model="sex">女
        </label>
        <h2>你是一个{{sex}}学生</h2>
        <hr>
        <input type="range" min="0" max="200" v-model="height">
        <h2>你的身高是{{height}}cm</h2>
    </div>
</body>
<script src="./js/vue.js"></script>
<script>
    new Vue({
        el: "#app",
        data: {
            userName: "张珊",
            sex: "女",
            height:0
```

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

5. `v-bind`属性绑定

在HTML标签当中，如果要对某一些属性进行绑定，这个时候我们就要使用`v-bind`来进行了

```
<body>
    <div id="app">
        <h2>
            <a v-bind:href="linkHref">{{linkName}}</a>
        </h2>
        <img v-bind:src="img2" >
        <img v-bind:src="img3" >
        <img v-bind:src="img4" >
    </div>
</body>
<script src="./js/vue.js"></script>
<script>
    new Vue({
        el: "#app",
        data: {
            linkName: "百度一下",
            linkHref: "https://www.baidu.com",
            img2: "./img/item2.jpg",
            img3: "./img/item3.jpg",
            img4: "./img/item4.jpg"

        }
    })
</script>
```

属性值的绑定是在开发过程当中非常常用的一种操作，所以`vue`对于这一种操作进行了一步简化，直接把前面的`v-bind`省略掉，只留下`:`即可，如下

```
<body>
    <div id="app">
        <h2>
            <a :href="linkHref">{{linkName}}</a>
        </h2>
        <img :src="img2" >
        <img :src="img3" >
        <img :src="img4" >
    </div>
</body>
```

总结：总体上来说，如果属性前面有`:`则代表的是绑定了一个`vue`的值，如果没有则代表的是一个普通的属性值

同时请注意，在属性绑定过程当中，它也支持表达式，运算符以及方法的调用

6. `v-show`隐藏与显示

在vue里面，如果我们想让一个元素隐藏与显示，则可以通过这个指令去完成【想一想：为什么会有这个指令】

```
<body>
    <div id="app">
        <h2 v-show="flag">标哥哥真帅</h2>
        <h2 v-show="1>2">大家好啊</h2>
        <h2 v-show="1+1==2">你得到结果了吗？</h2>
        <h2 v-show="Boolean(123)">哈哈哈</h2>
    </div>
</body>
<script src="./js/vue.js"></script>
<script>
    new Vue({
        el: "#app",
        data: {
            flag: true
        }
    })
</script>
```

在条件渲染的`v-show`的后面，它接一个值，这值最终的结果应该是一个`boolean`类型，如果不是这个类型，则调用`Boolean()`去转；

当`v-show`的条件不成立的时候，这个时候元素上面会多了一个`display:none`将元素进行隐藏

同理，`v-show`的后面也可以跟一个条件表达式，也可以跟方法，还可以跟运算符都行

这个`v-show`的指令在后期的开发过程当中是一个使用频率非常高的一个指令，具体可以参考以下的案例

```
<body>
    <div id="app">
        <p>
            用户名：<input type="text" v-model="userName"><span v-
show="!userName">请输入用户名</span>
        </p>
        <p>
            密码：<input type="text" v-model="pwd"><span v-show="!pwd">请输入
密码</span>
        </p>
        <p>
            <!-- <button type="button" v-show="userName.length&&pwd.length">
登陆</button> -->
            <button type="button" id="btn" :disabled="!
(userName.length&&pwd.length)">登陆</button>
        </p>
    </div>
</body>
<script src="./js/vue.js"></script>
<script>
    new Vue({
        el: "#app",
        data: {
            userName: "",
            pwd: ""
        }
    })
</script>
```

上面的案例就是使用`v-show`配合进行表单的验证

7. `v-if`条件渲染

   这个指令展现出来的效果与`v-show`非常相似，但是原理是绝然不同的

   `v-show`是通过CSS当中的`display:none`让元素隐藏与显示，而`v-if`则是通过**注释**让元素移除或正常显示【这个东西后期在Vue的生命周期里面会有明显的区域】

```
<body>
    <div id="app">
        <h2 v-if="flag">这是标哥哥在给你们的提醒 ， 莫要睡觉</h2>
        <h2 v-else>我就是要睡觉 ， 你咋滴</h2>
        <hr>
        <h2 v-if="age<=18">你是未成年人</h2>
        <h2 v-else-if="age<30">你是青年人</h2>
        <h2 v-else>你是成年人</h2>
    </div>
</body>
<script src="./js/vue.js"></script>
<script>
    new Vue({
        el: "#app",
        data: {
            flag: true,
            age:25
        }
    })
</script>
```

   上面的条件渲染与我们之前在学习JS里面的流程控制原理是一样的，可以进行多条件的判断

8. `v-for`列表渲染

   这是vue当中最重要的一个渲染指令，它可以将列表进行渲染

```
<body>
    <div id="app">
        <h2 v-for="(item,index) in arr" :key="index">{{item}}</h2>
    </div>
</body>
<script src="./js/vue.js"></script>
<script>
    new Vue({
        el: "#app",
        data: {
            arr:["刘光礼","吴景","熊峰","大江哥","凯哥"]
        }
    })
</script>
```

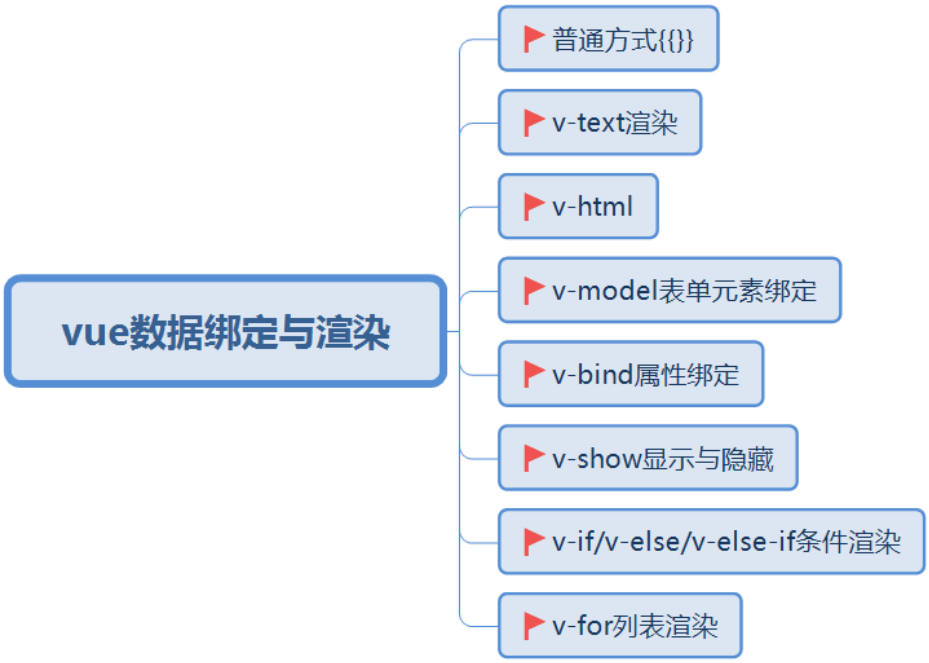
   上面就是最简单的列表渲染，`v-for`是列表渲染的指令，`(item,index)`第一个代表遍历出来的每一项，第二个代表遍历的索引，这个名称是变量名所以是可以更改的

9. `v-once`单次绑定

```
<body>
    <div id="app">
        <h2>{{userName}}</h2>
        <h2 v-text="userName"></h2>
```

```
        <h2 v-once>{{userName}}</h2>
        <hr>
        <button type="button" @click="userName = '李四'">改变userName的值
</button>
    </div>
</body>
<script src="./js/vue.js"></script>
<script>
    new Vue({
        el:"#app",
        data:{
            userName:"张三"
        }
    })
</script>
```

`v-once`只绑定一次，后期这个值如果发生了变化则不会在页面上面进行更改



上面就是我们目前所接触到的渲染方式。同时在这里也在说明一点，在`vue`当中，这些以`v-`开始的东西我们称之为**指令**

四、计算属性

在Vue接管页面的时候，应该是要接管页面上面所有的数据，而这些所有的数据大部分都是由`vue`里面的`data`来提供的，但是data里面提供的数据有一个缺点，它只能直接去使用，不能够进行二次处理，如下数据

```
shopCarList:[
    {goodsName:"冰箱",count:2,price:999},
    {goodsName:"小米10",count:1,price:2999},
    {goodsName:"联想小新Air 14",count:1,price:3899},
    {goodsName:"面包",count:8,price:17.9},
    {goodsName:"牛奶",count:6,price:22}
]
```

如果我们想得到所有商品的总价，这个时候就需要对`shopCarList`去做遍历，然后再求和，最后显示在页面上面。所以我们发现在页面上面有一个值是需要经过计算以后才能去使用的。针对这种情况`vue`是可以实现的

在`vue`里面，它提供了一种叫做**计算属性**的东西，可以完成上面的需要

```
<!DOCTYPE html>
<html lang="zh">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>数据渲染</title>
    <style>
        .table1 {
            width: 900px;
            margin: auto;
            border-collapse: collapse;
        }

        .table1 tr {
            height: 35px;
        }

        .table1 td,
        .table1 th {
            border: 1px solid black;
        }
    </style>
</head>

<body>
    <div id="app">
        <h2>购物车列表</h2>
        <table class="table1">
            <tr>
                <th>商品名称</th>
                <th>商品数量</th>
                <th>商品单价</th>
                <th>商品总价</th>
            </tr>
            <tr v-for="(item,index) in shopCarList" :key="index">
                <td>{{item.goodsName}}</td>
                <td>{{item.count}}</td>
                <td>{{item.price}}</td>
                <td>{{item.count * item.price}}</td>
            </tr>
            <tr>
                <td>总价</td>
                <td colspan="3">
                    {{totalMoney}}
                </td>
            </tr>
        </table>
    </div>
</body>
```

```
<script src="./js/vue.js"></script>
<script>
    new Vue({
        el: "#app",
        data: {
            shopCarList:[
                {goodsName:"冰箱",count:2,price:999},
                {goodsName:"小米10",count:1,price:2999},
                {goodsName:"联想小新Air 14",count:1,price:3899},
                {goodsName:"面包",count:8,price:17.9},
                {goodsName:"牛奶",count:6,price:22}
            ]
        },
        // 这里是计算属性
        computed: {
            // 这里定义了一个方法 ，这个方法必须有一个返回值
            totalMoney() {
                //在这里，我们要把shopCarList这个数组拿到以后去遍历
                //console.log("我要得到data里面的数据");
                // console.log(this.$data.shopCarList);    这是最初的拿值，vue也提供了更简便的拿值方式
                //console.log(this.shopCarList === this.$data.shopCarList);
                let sum = 0;
                this.shopCarList.forEach(item => {
                    sum += item.price * item.count;
                })
                return sum;
            }
        }
    })
</script>

</html>
```

代码分析：

1. 在上面的案例当中，我们可以看到`shopCarList`里面的数据在`totalMoney`这个方法里面计算了一下，最后返回结果，这个时候`totalMoney`写在了`Vue`的`computed`里面，它就叫计算属性
2. 计算属性本身是一个方法，这个方法要返回一个值
3. 在vue的内部，如果要拿`data`里面的数据，可以通过`this.$data`去拿；同时可以更进一步的去简化操作，直接通过`this`调用就可以了，所以你可以看到`this.shopCarList === this.$data.shopCarList`是成立的，我们把中间的`$data`省略掉了

商品名称	商品数量	商品单价	商品总价
冰箱	2	999	1998
小米10	1	2999	2999
联想小新Air 14	1	3899	3899
面包	8	17.9	143.2
牛奶	6	22	132
总价	9171.2		

五、vue事件与事件对象

Vue在接管页面的数据以后，还可以接管页面的方法与事件，如五

```
<!DOCTYPE html>
<html lang="zh">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>vue事件</title>
</head>

<body>
    <div id="app">
        <h2>{{userName}}</h2>
        <button type="button" v-on:click="sayHello">按钮1</button>
        <button type="button" v-on:click="abc('标哥哥')">按钮2</button>
        <button type="button" v-on:click="def($event)">获取事件参数</button>
        <button type="button" v-on:click="aaa">事件方法中的this到底是谁？</button>
        <button type="button" v-on:click="changeUserName">我要改变标哥哥</button>
    </div>
</body>
<script src="./js/vue.js"></script>
<script>
    new Vue({
        el: "#app",
        // 负责接管页面数据
        data: {
            userName:"标锅锅"
        },
        // 负责接管页面计算属性
        computed: {

        },
        // 负责接管页面的事件、方法
        methods: {
            sayHello() {
                alert("今天天气不太好，我心情也不好，因为你们在打瞌睡");
            },
            abc(userName) {
                alert('大家好，我叫' + userName)
            },
            def(event) {
                //在这里，我们要获取事件参数
                console.log(event);
            },
            aaa() {
                //在这里this是指向vue的对象
                console.log(this);
            },
            changeUserName(){
                //把userName换成“标哥哥”
                this.userName = "标哥哥";
                //可以通过下面的方式再调用方法
                this.sayHello();
            }
        }
```

```
    })
</script>
</html>
```

首先通过上面的案例可以总结出以下几个点

1. 所有的事件方法应该都在`methods`里面
2. vue在页面进行事件绑定的时候使用的指令`v-on`
3. 在调用vue的事件方法的时候，如果不需要传递参数，则方法后面的小括号可以省略
4. 如果需要获取事件对象，则在调用方法的时候使用`$event`去进行参数传递
5. vue事件方法中的`this`指向的是vue当前对象
6. 正是因为事件方法里面的`this`是指向vue当前对象的，所以我们可以在事件方法里面去调用数据`data`，也可以再次调用其它的方法
7. 对于一些简单的数据操作，我们可以直接写在事件后面，不必要再次定义方法了，如下

```
<button type="button" v-on:click="userName='我在通过第二种方式把你改成帅气的标哥哥'">第二种方式简便些改userName的值</button>
<script>
    new Vue({
        el: "#app",
        data: {
            userName: "标哥哥"
        },
        methods: {
            changeUserName() {
                this.userName = "无敌帅气的标锅锅";
            }
        }
    })
</script>
```

8. vue在进行事件绑定的时候也提供一种简便的绑定方式，【与之前的属性绑定也是一样的，都有快捷语法】

```
<button type="button" v-on:click="changeUserName">普通绑定</button>
<button type="button" @click="changeUserName">简便版绑定</button>
```

直接把`v-on:`换成`@`即可，如`v-on:click`就会变成`@click`

六、侦听器

侦听器也叫监听器，在Vue里面也叫属性监听

普通侦听

Vue当中的候听器可以对data里面的**属性值** 以及 **计算属性**里面的属性值进行监听，然后再做出相应的处理

```
<body>
    <div id="app">
        <button type="button" @click="age--">-</button>
        <input type="text" v-model="age">
        <button type="button" @click="age++">+</button>
```

```
    </div>
</body>
<script src="./js/vue.js"></script>
<script>
    new Vue({
        el: "#app",
        data: {
            age: 18
        },
        // 侦听器
        watch: {
            //newValue代表新的值，oldValue代表旧的值
            age(newValue, oldValue) {
                if(newValue<14){
                    this.age = 14;
                }
            }
        }
    })
</script>
```

> 普通侦听只适合于基本数据类型，对于对象，我们则无法实现侦听，这个时候，我们就要使用到深度侦听

深度侦听

主要是用于对象的变化的时候的侦听，它的语法与普通侦听器有所不同

```
<body>
    <div id="app">
        <h2>{{age}}</h2>
        <button type="button" @click="age++">年龄+</button>
        <hr>
        <h2>{{userInfo.userName}}----{{userInfo.userAge}}</h2>
        <button type="button" @click="userInfo.userAge++">userInfo年龄+</button>
    </div>
</body>
<script src="./js/vue.js"></script>
<script>
    new Vue({
        el: "#app",
        data: {
            age: 18,
            userInfo: {
                userName: "张三",
                userAge: 18
            }
        },
        watch: {
            //普通侦听
            age(newValue, oldValue) {
                console.log(newValue);
            },
            //深度侦听
            userInfo: {
                handler: function (newValue, oldValue) {
                    console.log("userInfo内部发生了变化");
```

```
                    console.log(newValue);
                },
                deep: true
            }
        }
    });
</script>
```

上面的方式就是深度侦听，这样对象的内部即使发生了变化，我们也能够去侦听到

七、事件修饰符

事件修饰符是主要针对Vue的事件的一个增强功能，它可以让事件执行一些特殊的操作

```
<body>
    <div id="app">
        <div class="div1" @click="a">
            <button type="button" @click="b($event)">按钮</button>
        </div>
    </div>
</body>
<script src="./js/vue.js"></script>
<script>
    new Vue({
        el: "#app",
        methods: {
            a() {
                console.log("我是方法a");
            },
            b(event) {
                console.log("我是方法b");
                event.stopPropagation();
            }
        }
    })
</script>
```

在上面的代码当中，我们可以很清楚 看到DOM里面的事件冒泡行为，在以前的操作当中，如果我们要阻止事件的冒泡，我们需要进行事件对象获取以后，再调用`stopPropagation()`去完在，但是在Vue里面，则没有这么麻烦

我们在绑定事件的时候，可以直接在事件的后面添加修饰符，如下

```
<div class="div1" @click="a">
    <button type="button" @click.stop="b">按钮</button>
</div>
```

我们在内部的按钮的`click`事件后面添加了一个`stop`的东西，这个就是事件修饰符，这个修饰符是阻止事件冒泡

现在我们把一些常用的事件修饰符先列举如下

1. `事件.stop`阻止事件的冒泡
2. `鼠标事件.left`左键
3. `鼠标事件.right`右键
4. `鼠标事件.center`中键

5. `事件.prevent` 阻止事件的默认行为
6. `keydown.up` 只有按键盘的“上键”才会触发
7. `keydown.down` 只要按键盘的"下键"才会触发
8. `keydown.enter` 只要按键盘的"回车键"才会触发
9. `事件.ctrl` 只有按住 `ctrl` 组合才能触发
10. `事件.alt` 只有按信 `alt` 组合才能触发
11. `事件.ctrl.alt` 只有按信 `ctrl+alt` 组合才会触发

八、class动态样式绑定

每个元素其实都有属性，我们之前也学习过 `vue` 里面的属性绑定，如果有属性需要绑定vue里面的值，则使用 `:属性名` 来进行绑定

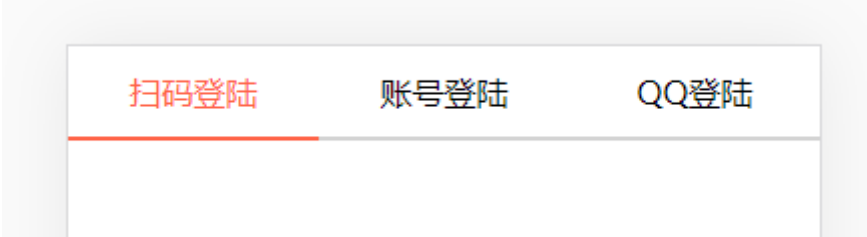
但是 `class` 做为一个特殊的属性，它的绑定是有区别的。而在我们的开发过程当中，我们经常需要使用到动态的class样式绑定，这个时候怎么办呢

对象语法

```
<!DOCTYPE html>
<html lang="zh">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>vue重点 动态样式绑定</title>
    <style>
        * {
            margin: 0;
            padding: 0;
        }

        .login-type {
            width: 100%;

            list-style-type: none;
            display: flex;
        }

        .login-type>li {
            height: 45px;
            flex: 1;
            display: flex;
            flex-direction: row;
            justify-content: center;
            align-items: center;
            border-bottom: 2px solid lightgray;
        }

        .login-type>li.selected {
            color: tomato;
            border-bottom-color: tomato;
        }
    </style>
</head>
```

```
<body>
    <div id="app">
        <ul class="login-type">
            <li :class="{selected:pageType==0}" @click="pageType=0">扫码登陆</li>
            <li :class="{selected:pageType==1}" @click="pageType=1">账号登陆</li>
            <li :class="{selected:pageType==2}" @click="pageType=2">QQ登陆</li>
        </ul>
    </div>
</body>
<script src="./js/vue.js"></script>
<script>
    new Vue({
        el: "#app",
        data: {
            // 我们假0就是扫码，1就是账号
            pageType: 0
        }
    })
</script>

</html>
```

上面采用了一个数组语法，去实现下面的效果

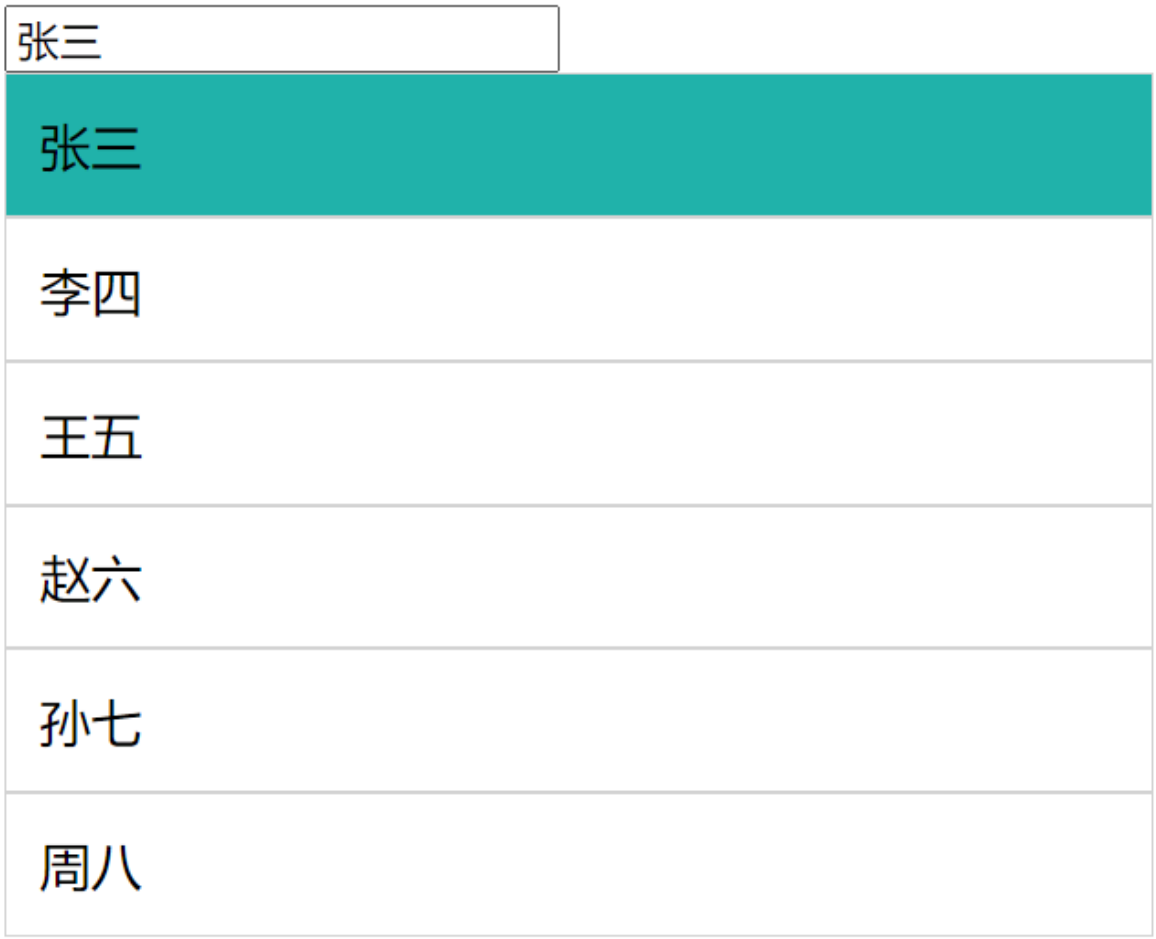


当我们在点击某一个的时候，就会自动的去改变样式

案例

```
<!DOCTYPE html>
<html lang="zh">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>案例</title>
    <style>
        .list-group {
            margin: 0;
            padding: 0;
            list-style-type: none;
        }

        .list-group>li {
            height: 45px;
            display: flex;
            align-items: center;
            border: 1px solid lightgray;
            box-sizing: border-box;
            padding: 0px 10px;
```

```
        }

        .list-group>li.active {
            background-color: lightseagreen;
        }
    </style>
</head>

<body>
    <div id="app">
        <input type="text" v-model="stuList[currentIndex]"
@keydown.up="currentIndex--" @keydown.down="currentIndex++">
        <ul class="list-group">
            <li  v-for="(item,index) in stuList" :class="
{active:currentIndex==index}" :key="index">{{item}}</li>

        </ul>
    </div>
</body>
<script src="./js/vue.js"></script>
<script>
    new Vue({
        el: "#app",
        data: {
            stuList: ["张三", "李四", "王五", "赵六", "孙七", "周八"],
            currentIndex: 0
        },
        watch: {
            currentIndex(newValue, oldValue) {
                if (newValue < 0) {
                    this.currentIndex = this.stuList.length - 1;
                }
                else if(newValue>this.stuList.length-1){
                    this.currentIndex = 0;
                }
            }
        }
    })
</script>

</html>
```



在写数组的对象语法的时候，对象的属性名也就是`class`的名称是可以不加引号的，但是如果这个`class`的名称比较特别的时候，则要注意，如下

```
.a-b{
    color:red;
}
```

在上面的class名称中间有个`-`，这个时候在做对象语法的样式动态绑定的时候，则要注意，应该要加引号，如下

```
<li :class="{'a-b':currentIndex==index}">{{item}}</li>
```

数组语法

数组语法也是`vue`框架中动态样式绑定的一种，它与对象语法很相似，我们现在使用数组语法去完成之前的案例

```
<ul class="login-type">
    <li :class="{selected:pageType==0,abc:pageType==0}" @click="pageType=0">账号登陆</li>
    <li :class="[pageType==1?'selected':null,pageType==1?'abc':null]" @click="pageType=1">扫码登陆</li>
</ul>
<script>
    new Vue({
        el: "#app",
        data: {
            pageType: 1
        }
    })
</script>
```

在上面的代码里面，我们同时使用了数组语法与对象语法，它们最终都是为了实现根据某一个条件去动态的绑定样式，不同的是它们的主法

1. 对象语法是把class名称当成对象的属性名，属性值则是一个布尔值，如果这个值为true则代表这个class存在，否则就不存在
2. 数组语法内部借用的是条件表达式，条件成立给值，条件不成立给另一个值

九、style动态样式绑定

这样原理与我们之前的`class`动态样式绑定的原理差不多

如果页面上面有一个像这样的需求，就是点击某一个按钮，让上面的文字颜色变成相应的颜色，如果我们采用以前的`class`样式动态绑定，这个时候就比较麻烦，代码如下

```
<style>
    .red{
        color: red;
    }
    .blue{
        color: blue;
    }
</style>
<div id="app">
    <h2 :class="[flag?'red':'blue']">标哥哥真帅</h2>
    <button type="button" @click="flag=true">红色</button>
    <button type="button" @click="flag=false">蓝色</button>
    <button type="button">海蓝色</button>
</div>
<script>
    new Vue({
        el: "#app",
        data: {
            flag:true
        }
```

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

`class`样式的动态绑定使用数组语法以后最多只允许两个条件的切换，如果想多个条件的切换实现起来相对困难。所以针对这种情况，我们其实可以使用`style`的动态样式绑定

对象语法

```
<body>
    <div id="app">
        <h2 :style="{color:a}">标哥哥真帅</h2>
        <button type="button" @click="a='red'">红色</button>
        <button type="button" @click="a='blue'">蓝色</button>
        <button type="button" @click="a='lightseagreen'">海蓝色</button>
    </div>
</body>
<script src="./js/vue.js"></script>
<script>
    new Vue({
        el: "#app",
        data: {
            a: "red"
        }
    })
</script>
```

通过上面的案例，我们发现对于这种复杂情况下不固定的样式改变，我们可以使用`style`，这样更加灵活一点

案例


```
<body>
    <div id="app">
        <p :style="cssObj">标哥哥最帅</p>
        <button type="button" @click="cssObj.color='blue'">蓝色字体</button>
        <button type="button" @click="cssObj.color='red'">红色字体</button>
        <button type="button" @click='addFontSize'>字体变大</button>
        <button type="button" @click="cssObj.fontWeight = 'bold'">字体加粗
</button>
    </div>
</body>
<script src="./js/vue.js"></script>
<script>
    new Vue({
        el: "#app",
        data: {
            cssObj: {
                color: "black",
                fontSize: "16px",
                fontWeight: "normal"
```

```
            }
        },
        methods: {
            addFontSize() {
                let f = parseInt(this.cssObj.fontSize);
                f++;
                this.cssObj.fontSize = f + "px";
            }
        }
    })
</script>
```

数组语法

动态的`style`样式绑定里面，也是支持数组语法，它可以将多个数组结合一起，然后同步渲染样式

```
<body>
    <div id="app">
        <p :style="[cssObj,cssObj2]">标哥哥最帅</p>
        <button type="button" @click="cssObj.color='blue'">蓝色字体</button>
        <button type="button" @click="cssObj.color='red'">红色字体</button>
        <button type="button" @click='addFontSize'>字体变大</button>
        <button type="button" @click="cssObj.fontWeight = 'bold'">字体加粗
</button>
        <button type="button" @click="centerText">居中</button>
        <button type="button" @click="setHeight">给一个高度</button>
    </div>
</body>
<script src="./js/vue.js"></script>
<script>
    var app = new Vue({
        el: "#app",
        data: {
            cssObj: {
                color: "black",
                fontSize: "16px",
                fontWeight: "normal",
                border:"2px solid black"
            },
            cssObj2: {
                textDecoration: "underline red wavy"
            }
        },
        methods: {
            addFontSize() {
                let f = parseInt(this.cssObj.fontSize);
                f++;
                this.cssObj.fontSize = f + "px";
            },
            centerText() {
                // this.cssObj2.textAlign="center";
                // 如果以前data对象里面没有这个属性，而后期你又添加了一个属性，但是你希望这
个属性又同步的驱动页面
                // 不能够直接在这个对象上面赋值，要使用vue内部提供给我们一个方法
                this.$set(this.cssObj2, "textAlign", "center");
            },
            setHeight() {
```

```
                // this.cssObj2.height = "100px";
                this.$set(this.cssObj2,"height","100px");
            }
        }
    })
</script>
```

通过上面的案例，我们发现这种方式很灵活，我们后期只用去更改`cssObj`或`cssObj2`这个对象就可以动态的调用这个元素的任何样式

同时，我们还可以通过`this.$set`来向元素上面添加的属性，然后再渲染到页面

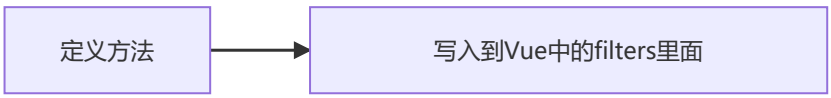
十、过滤器

在我们之前去学习`template`的 模板引擎的时候，我们也学过过滤器的概念【过滤器就是需要将展示的数据做二次处理，所以我们也将过滤器叫格式化工具】

vue在做数据渲染的时候，同样也是支持过滤器的

局部过滤器

局部过滤器本身是定义一个方法，然后再把这个方法 注册 到`vue`内部的`filters`属性当中




```
<div id="app">
    <table class="table table-hover table-bordered table-striped">
        <tr>
            <th>姓名</th>
            <th>性别</th>
            <th>生日</th>
            <th>个性说明</th>
        </tr>
        <tr v-for="(item,index) in stuList" :key="index">
            <td>{{item.userName}}</td>
            <td>{{item.sex}}</td>
            <td>{{item.birthday | formatDate}}</td>
            <td>{{item.desc | showDesc(10)}}</td>
        </tr>
    </table>
</div>
script src="./js/vue.js"></script>
<script>
    //在这里，它还只是一个普通的方法
    function formatDate(d) {
        if (d instanceof Date) {
            let year = d.getFullYear();
            let month = (d.getMonth() + 1).toString().padStart(2, "0");
            let date = d.getDate().toString().padStart(2, "0");
            return [year, month, date].join("-");
        } else {
```

```
            return "";
        }
    }

    function showDesc(str, maxCount = 20) {
        if (str) {
            if (str.length > maxCount) {
                return str.substr(0, maxCount) + "...";
            } else {
                return str;
            }
        } else {
            return "";
        }
    }


    new Vue({
        el: "#app",
        data: {
            stuList: [{
                    userName: "张珊",
                    sex: "女",
                    desc: "我是一个可爱的女孩子，有喜欢的男生快来约",
                    birthday: new Date()
                },
                {
                    userName: "李四",
                    sex: "男",
                    desc: "家穷人瘦，一米六九，初中文化，农村户口，破房三间，薄田两某，一年
四季，药不离口，今日大此，广征女友，革命路上，并肩携手，计划生育，一定遵守",
                    birthday: new Date("1999-9-10")
                },
                {
                    userName: "王五",
                    sex: "男",
                    desc: "阳光帅气的男孩子一枚，期待有缘来领养，共渡余生，小生这厢有礼
了",
                    birthday: new Date("2000-1-5")
                },
            ]
        },
        // 这里就是定义过滤器
        filters: {
            formatDate: formatDate,
            showDesc: showDesc
        }
    })
</script>
```

注意事项:

1. 过滤器本身其实是一个方法，它需要在`Vue`内部的`filters`当中去注册成一个过滤器
2. 过滤器的这个方法当中，它的第一个参数永远都是你要过滤的值，当然，你在定义的时候还可以再定义其它的参数（可能看上面的代码当中的showDesc）这个过滤器
3. 在调用过滤器的时候直接在要显示的值的后面通过`| 过滤器名`来调用即可，如果有参数，则添加括号再加入参数，相当于调用了一个方法，可能看上面代码当中的`shosDesc`

全局过滤器

全局过滤器本质上面也是定义一个方法，但是它不需要注册，可以直接使用

本处还是以上面的日期格式化与长度截取为例子，去注册全局过滤器

```
//定义了第一个全局过滤器
Vue.filter("formatDate", function (d) {
    if (d instanceof Date) {
        let year = d.getFullYear();
        let month = (d.getMonth() + 1).toString().padStart(2, "0");
        let date = d.getDate().toString().padStart(2, "0");
        return [year, month, date].join("-");
    } else {
        return "";
    }
})

//定义了第二个全局过滤器
Vue.filter("showDesc", function (str, maxCount = 20) {
    if (str) {
        if (str.length > maxCount) {
            return str.substr(0, maxCount) + "...";
        } else {
            return str;
        }
    } else {
        return "";
    }
})
```

全局过滤器使用的是`Vue.filter("过滤器名称",过滤器处理的回调函数)`来完成的，通过这种方式定义的过滤器，不需要在`Vue`的内部去注册了，可以直接使用

十一、Vue中的DOM操作

Vue本身其实是不太推荐我们使用原生 的DOM操作的，但是Vue本身又推荐我们使用`virtual DOM`，所以后期如果我们想去操作`virtual DOM`（也就是后期的组件），则我们必须还是要了解一下vue当中的DOM操作的基本点

在vue的内部，如果想获取托管区域内部的DOM元素是有两种方法的

1. 通过事件对象去获取
2. 通过vue内部的方法`$refs`来获取

> 在这里强调一点，切记切记不能使用`document.querySelector`等类似的方式去获取DOM元素

通过事件去获取DOM元素

vue内部是可以使用事件的，竟然有事件，则必然会有**事件绑定者**与**事件触发者**，所以我们可以通过这2者去获取到相应的DOM元素

```
<body>
    <div id="app">
        <div class="box">
            <button type="button" @click="sayHello($event)">按钮</button>
        </div>
```

```
    </div>
</body>
<script src="./js/vue.js"></script>
<script>
    new Vue({
        el:"#app",
        data:{

        },
        methods:{
            sayHello(event){
                console.log("事件的绑定者",event.currentTarget);
                console.log("事件的触发者",event.target);
                console.log(event.currentTarget.parentElement);
            }
        }
    })
</script>
```

通过`$refs`来完成DOM选取

这一种方式是vue内部最常见的一种方式，方便我们要获取需要的DOM元素

```
<body>
    <div id="app">
        <div class="box">
            <button type="button" @click="sayHello">按钮</button>
        </div>
        <div>
            <h2 ref="abc">标哥哥真帅气</h2>
            <button ref="btn1" type="button">哈哈哈</button>
        </div>
    </div>
</body>
<script src="./js/vue.js"></script>
<script>
    new Vue({
        el:"#app",
        data:{

        },
        methods:{
            sayHello(){
                console.log(this.$refs.abc);
                console.log(this.$refs.btn1);
            }
        }
    })
</script>
```

注意事项:

如果在使用`v-for`去循环渲染的时候，这个时候使用`ref`就要注意了，如下

```
<body>
    <div id="app">
        <div>
```

```
            <img  v-for="(item,index) in imgList" :src="item" :key="index"
:ref="'img'+index">
        </div>
        <hr>
        <button type="button" @click="getDom">获取DOM</button>
    </div>
</body>
<script src="./js/vue.js"></script>
<script>
    new Vue({
        el:"#app",
        data:{
            imgList:[
                "./img/item1.jpg",
                "./img/item2.jpg",
                "./img/item3.jpg",
                "./img/item4.jpg"
            ]
        },
        methods:{
            getDom(){
                console.log(this.$refs.img0);
                console.log(this.$refs.img1);
                console.log(this.$refs.img2);
                console.log(this.$refs.img3);
            }
        }
    })
</script>
```

这个时候我们打印出来的每个`ref`都是一个数组了，如下图

这其实是vue提供的一种便捷操作

最后总结一个点：Vue是一个去DOM操作的框架，但是它允许我们拿到DOM以后获取 DOM的属性（但是不要设置DOM的属性），它允许我们通过 ` ref ` 来映射DOM，但不允许我们`query`DOM，因为这么做都是不符合我们`MVVM`数据驱动的原则，也不符合高效开发的原则