VUE-DAY 01

1.原生JS写法

2.VUE写法

{{内容}}，这里的{{}}是插值表达式，里面包含变量名或者简单的表达式，例如三元表达式；

`<h2>{{count%2===0?'偶数':'奇数'}}</h2>`

可以在button里面写事件，例如 `@click='fn'` ,传入函数名称； `@click="count++"` ，进行一些简单的数据操作；

`<button @click="fn">点我自增</button>`

标准写法格式举例如下：

HTML | 复制代码

```html
<body>
    <div id="app">
        <!-- <h2>{{count}}</h2>  这里的双{{}}叫做插值表达式，里面不仅可以写普通的变量名，也可以写成三元表达式 -->
        <h2>{{count%2===0?'偶数':'奇数'}}</h2>
        <button @click="fn">点我自增</button>
        <!-- 之前原生写法里button里面用onclick='fn()' -->
        <!-- vue里面直接写函数名称 -->
    </div>
    <!-- 开发环境版本，包含了有帮助的命令行警告 -->
<script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
<script>
    new Vue({  //记住这里的是Vue，而不是VUE，后两个字母要小写
        el:'#app',  //理解为vue关联的视图区域
        data:{
            count:0
        },
        methods:{
            fn(){
                this.count++
            }
        }
    })
</script>
</body>
```

需要注意的细节有：

1，可以将插值表达式配合数据操作作为标签体的内容；

2，在 `@click="fn"` 中，里面的就是函数名，不加括号；

3，new Vue不是new VUE，也不是new vue；

4，记得methods方法调用data里面的数据时要用this，此时this的指向是创建的实例对象

3.事件绑定的指令及细节

HTML | 复制代码

```html
<body>
    <div id="app">
        <!-- <h2>{{count}}</h2>  这里的双{{}}叫做差值表达式 -->
        <h2 v-on:mouseover='fn1'>{{count%2===0?'偶数':'奇数'}}</h2>
        <button @click="fn">点我自增</button>
        <!-- 这里两种写法都可以，因为 @ 是 v-on: 的简写形式，一般用简写形式比较多 -->
        <!-- 之前原生写法里button里面用onclick='fn()' -->
        <!-- vue里面直接写函数名称，不用加括号 -->
    </div>
    <!-- 开发环境版本，包含了有帮助的命令行警告 -->
    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
    <script>
        const vm = new Vue({
            el: '#app',  //理解为vue关联的视图区域，element的缩写形式，代表挂载点
            data: { //理解为存储所有视图区域的数据的对象
                count: 0
            },
            methods: {  //理解为存储所有视图区域事件回调的对象
                //第一个指令：  v-on:  用来绑定事件的，它有一个简写的形式 @
                //如果在methods配置选项中访问到data中的数据 必须通过this调用
                //vue管理的函数不能写成箭头函数，必须写成普通函数
                //定时器的回调函数不属于vue管理的函数，如果希望内部访问this必须写成箭头函数
                fn() {  //不能写成箭头函数，箭头函数里面的this指向父级元素，这里写成fn:()=>{}的形式，里面的this就是指向window了


                    // setTimeout(function () {
                    //     console.log(this);
                    //     this.count++
                    // }, 1000)


                        // setTimeout 里面的this默认的是指向window，所以这里要写成箭头函数，这样就会去找外层的父级元素也就是new出来的实例化对象vm


                    setTimeout(()=> {
                        this.count++
                    }, 1000)
                },
                fn1() {
                    console.log('鼠标移入了');
                }
```

```
            }


            /* 这里的el data methods 等键名都是固定的，不能随意更改名称也不能拼错 */
        })
        console.log(vm);


    </script>
</body>
```

1，点击事件的绑定有两种写法

HTML | 复制代码

```
<h2 v-on:mouseover='fn1'>{{count%2===0?'偶数':'奇数'}}</h2>
        <button @click="fn">点我自增</button>
```

2，如果在methods配置选项中访问到data中的数据 必须通过this调用

3，vue管理的函数不能写成箭头函数，必须写成普通函数

4，定时器的回调函数不属于vue管理的函数，如果希望内部访问this必须写成箭头函数

HTML | 复制代码

```html
<body>
    <div id="app">
        <h1>{{count}}</h1>
        <button @click="count++">点我自增</button> <!-- 这里不需要在methods里面写this.count++，是因为已经在data中定义了count的数值为0，所以是可以访问到的，那么++就是可以的 -->

        <!-- 在模板内直接访问vm不需要加this -->
        <!-- <button @click="alert('这是提示内容')">点我自增</button> -->
        <button @click="alert()">提示</button>
        <!-- 这里alert就是一个函数名，而不是一个类似于++的命令，所以要在methods里面定义 -->
    </div>
    <!-- 开发环境版本，包含了有帮助的命令行警告 -->
<script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
<script>
    new Vue({
        el:'#app',
        data:{
            count:0
        },
        methods:{
            alert(){
                alert('这是提示按钮')
            }
        }
    })
</script>
</body>
```

这里的

4.事件传参

HTML | 复制代码

```html
<body>
    <div id="app">
        <!-- 如果希望给事件传递参数，就必须加上小括号 -->
        <button @click="say('hi')">say hi</button>
        <button @click="say('hello')">say hello</button>
    </div>
    <!-- 开发环境版本，包含了有帮助的命令行警告 -->
<script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
<script>
    new Vue({
        el:'#app',
        data:{

        },
        methods:{
            //如果希望显示地接受事件对象地话 比如在事件触发时手动地传入$event的参数
            say(msg){
                alert('say'+msg)
            }
        }
    })
</script>
</body>
```

5.绑定动态属性

HTML | 复制代码

```html
<body>
    <div id="app">
        <!-- 点击a标签跳转到百度 -->
        <!-- 如果希望动态地绑定标签内部的属性 必须通过v-bind:属性名  这个指令  可以简写为 : -->

        <!-- <a v-bind:href="url">{{count}}</a> -->
        <a :href="url">{{count}}</a>
        <!-- 这两种写法是一样的，一个没有简写一个简写 -->
    </div>
    <!-- 开发环境版本，包含了有帮助的命令行警告 -->
<script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
<script>
    new Vue({
        el:'#app',
        data:{
            count:'百度',
            url:'http://www.baidu.com'
        }
    })
</script>
</body>
```

用 `v-bind:` 或者直接用 `:` 实现动态属性的绑定，同样利用插值表达式也可以实现内容的动态绑定；

6.条件渲染指令v-if

HTML | 复制代码

```html
<body>
    <div id="app">
        <!-- v-if后面是通过布尔值来确认元素的显示与否 -->
        <!-- 第三个指令： v-if后面的布尔值如果为真 元素显示 否则元素隐藏 -->
        <h1 v-if="show">这是一个动态显示的标题</h1>

        <button @click="show = !show">点我切换标题的显示隐藏</button>
    </div>
    <!-- 开发环境版本，包含了有帮助的命令行警告 -->
<script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
<script>
    new Vue({
        el:'#app',
        data:{
            show:true
        },
    })
</script>
</body>
```

`v-if` 其实可以区别于之前的 `v-bind:` 和 `v-on:` ，只有这两个指令是后面还要跟冒号的，而 `v-if/v-else/v-for/v-html/v-text` 等都不需要后面加冒号，直接跟等号；

v-if后面是通过布尔值来确认元素的显示与否，v-if后面的布尔值如果为真 元素显示 否则元素隐藏

7.条件渲染指令v-else

HTML | 复制代码

```html
<body>
    <div id="app">
        <!-- v-else是需要紧跟在相应的v-if后面的，表示和v-if相反的显示 -->
        <h1 v-if="show">标题1</h1>
        <h1 v-else>标题2</h1>


        <button @click="show = !show">点击切换标题</button>
    </div>
    <!-- 开发环境版本，包含了有帮助的命令行警告 -->
    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
    <script>
        new Vue({
            el: '#app',
            data: {
                show: true
            },
        })
    </script>
</body>
```

v-else是需要紧跟在相应的v-if后面的，表示和v-if相反的显示

8.多个元素的条件渲染

HTML | 复制代码

```html
<div id="app">
        <!-- template是一个vue提供的空白标签 -->
        <button @click="show = !show">点击切换标题的显示隐藏</button>
        <template v-if="show">
            <h1>标题1</h1>
            <h1>标题2</h1>
            <h1>标题3</h1>
        </template>
    </div>
    <!-- 开发环境版本，包含了有帮助的命令行警告 -->
    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
    <script>
        new Vue({
            el: '#app',
            data: {
                show: true
            },
        })
    </script>
```

这里引入了template这个标签，其实算作一个空标签，在DOM中不占用空间，在元素中找不到它的位置，点击隐藏后也不会有位置的显示

9.v-show条件展示的指令

不同的是带有 v-show 的元素始终会被渲染并保留在 DOM 中。v-show 只是简单地切换元素的 一个CSS属性display。

v-if 是“真正”的条件渲染，因为它会确保在切换过程中条件块内的事件监听器和子组件适当地被销毁和重建。

v-if 也是**惰性的**：如果在初始渲染时条件为假，则什么也不做——直到条件第一次变为真时，才会开始渲染条件块。

相比之下，v-show 就简单得多——不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 CSS 进行切换。

一般来说，v-if 有更高的切换开销，而 v-show 有更高的初始渲染开销。因此，如果需要非常频繁地切换，则使用 v-show 较好；如果在运行时条件很少改变，则使用 v-if 较好

11.v-for循环渲染的指令

HTML | 复制代码

```html
<body>
    <div id="app">
         <ul>
            <!-- 这里写in或者of都可以实现数组遍历 -->
            <li v-for="(item,index) in friends">
                序号:{{item.id}}--姓名：{{item.name}}--年龄：{{item.age}}--性别：{{item.gender}}--<button @click="del(index)">点我删除这条数据</button>
            </li>
         </ul>
    </div>
    <!-- 开发环境版本，包含了有帮助的命令行警告 -->
<script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
<script>
    new Vue({
        el:'#app',
        data:{
            show:true,
            friends:[
                {
                    id:Math.random().toFixed(1),//生成随机0——1之间的数值并保留一位小数
                    name:'peiqi',
                    age:4,
                    gender:'girl'
                },
                {
                    id:Math.random().toFixed(1),
                    name:'qiaozhi',
                    age:3,
                    gender:'boy'
                },
                {
                    id:Math.random().toFixed(1),
                    name:'suxi',
                    age:5,
                    gender:'girl'
                }
            ]
        },
        methods:{
            del(index){
                //删除对应的那条数据
                //在一个数组中删除某个索引值的数组元素
                this.friends.splice(index,1)
            }
```

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

HTML | 复制代码

```
<body>
    <div id="app">
        <ul>
            <!-- 对象数据也可以渲染，普通数字也可以渲染，字符串也可以渲染，但实际渲染的数据都是数组 -->
            <li v-for="(value,key,index) in person">
                序号:{{index}}--姓名：{{item.name}}--年龄：{{item.age}}--性别：{{item.gender}}
            </li>
        </ul>
    </div>
    <!-- 开发环境版本，包含了有帮助的命令行警告 -->
<script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
<script>
    new Vue({
        el:'#app',
        data:{
            person:{
                name:'peiqi'
            }
        },
    })
</script>
</body>
```

12.v-html指令及v-text指令

HTML | 复制代码

```html
<body>
    <div id="app">
        <!-- v-html可以将指令后的内容渲染到页面上（和插值表达式{{}}作用一样），而且可以解析标签（{{}}不能解析标签） -->
        <h1 v-html="count"></h1>
    </div>
    <!-- 开发环境版本，包含了有帮助的命令行警告 -->
<script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
<script>
    new Vue({
        el:'#app',
        data:{
            count:'<em>0</em>'
        },
    })
</script>
</body>
```

HTML | 复制代码

```html
<body>
    <div id="app">
        <!-- v-text  和插值表达式{{}}的作用一样，都不能解析标签 -->
        <h1 v-text="count"></h1>
    </div>
    <!-- 开发环境版本，包含了有帮助的命令行警告 -->
<script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
<script>
    new Vue({
        el:'#app',
        data:{
            count:'<em>0</em>'
        },
    })
</script>
</body>
```

13.计算属性及其方法

HTML | 复制代码

```html
<body>
    <div id="app">
        <!-- 如果模板中能直接使用一个变量 那么这个变量一定会被挂载到vue实例身上 -->
        <!-- data，method，computed配置选项中的数据都会挂载到模板上 -->
        <h2>{{reverse}}</h2>  <!-- 尽量避免在模板中写过于复杂的表达式，这不是vue所推崇的方式 -->
    </div>
    <!-- 开发环境版本，包含了有帮助的命令行警告 -->
<script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
<script>
    new Vue({
        el:'#app',
        data:{
            msg:'hello'
        },
        computed:{
         reverse(){
             //计算属性必须通过返回值的形式表达
             return this.msg.split('').reverse().join('')
         }
        }
    })
</script>
</body>
```

HTML | 复制代码

```html
<body>
    <div id="app">
        <!-- data，method，computed配置选项中的数据都会挂载到模板上 -->
        <!-- 计算属性在调用时不要加() -->


        <!-- 1.methods方法需要在引用的时候加括号，computed计算则不需要加括号 -->


        <!-- <h2>{{reverse()}}</h2>
        <h2>{{reverse()}}</h2>
        <h2>{{reverse()}}</h2>
        <h2>{{reverse()}}</h2> -->


        <h2>{{reverse}}</h2>
        <h2>{{reverse}}</h2>
        <h2>{{reverse}}</h2>
        <h2>{{reverse}}</h2>
    </div>
    <!-- 开发环境版本，包含了有帮助的命令行警告 -->
    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
    <script>
        new Vue({
            el: '#app',
            data: {
                msg: 'hello'
            },
            // methods:{
            // reverse(){
            //     console.log('方法被调用了');
            //     //计算属性必须通过返回值的形式表达
            //     return this.msg.split('').reverse().join('')
            // }
            // }
            computed: {
                // 2.计算属性它有一个缓存功能，大大提高页面性能，多次调用只计算一次
                reverse() {
                    console.log('方法被调用了');
                    //计算属性必须通过返回值的形式表达
                    return this.msg.split('').reverse().join('')
                }
            }
        })
    </script>
```

```
</body>
```

HTML | 复制代码

```
<body>
    <div id="app">
        <!-- data，method，computed配置选项中的数据都会挂载到模板上 -->
        <!-- 计算属性在调用时不要加() -->
        <h2>{{reverse}}</h2>
        <h3>{{count}}</h3>
        <button @click="msg+=1">自增按钮</button>  <!-- 按钮点一次，修改一次数值，页面就会加载一次，如果用的是methods方法那么每加载一次页面都会调用一次方法，如果用的是计算机属性那么如果计算属性相关的数据没有发生改变的话也就不会调用相应的方法，可以实现局部加载的性能 -->
    </div>
    <!-- 开发环境版本，包含了有帮助的命令行警告 -->
    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
    <script>
        new Vue({
            el: '#app',
            data: {
                count:0,
                msg: 'hello'
            },
            // methods:{
            //     reverse(){
            //         console.log('方法被调用了');
            //         //计算属性必须通过返回值的形式表达
            //         return this.msg.split('').reverse().join('')
            //     }
            // }
            computed: {
                // 3.如果计算属性依赖的数据没有变化 它不会重新计算 同样能提高性能
                reverse() {
                    console.log('计算属性被调用了');
                    //计算属性必须通过返回值的形式表达
                    return this.msg.split('').reverse().join('')
                }
            }
        })
    </script>
</body>
```

14.侦听属性

HTML | 复制代码

```html
<body>
    <div id="app">
        <button @click="count++">自增按钮</button>
        <h2 v-show="ishow">动态显示的标题</h2>
    </div>
    <!-- 开发环境版本，包含了有帮助的命令行警告 -->
    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
    <script>
        new Vue({
            el: '#app',
            data: {
                //如果count值是偶数 h2显示 count是奇数 h2隐藏
                count:0,
                ishow:true
            },
            watch:{
                //监听count的变化 如果count变化了 执行回调函数
                count(newValue,oldValue){  //首先这里不要将watch中的变量名看成函数，这里指的就是data里面的count值，监听的就是这个count对象，不要加this变成this.count
                    //这里的a代表newValue，是最新的值；b代表oldValue，是上一次的值；也可以将其修改为newValue和oldValue
                    if(newValue%2===0){
                        this.ishow = true
                    }else{
                        this.ishow = false
                    }
                }
            }
        })
    </script>
</body>
```