VUE-DAY02

1.侦听属性的深入

HTML | 复制代码

```html
<body>
    <div id="app">
        <button @click="count++">自增按钮</button>
        <h2 v-show="ishow">这是一个动态显示的标题</h2>
    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
    <script>
        new Vue({
            el: '#app',
            data:{
                count:1,
                ishow:false
            },
            watch:{
                count:{
                    handler:function(newValue){
                    if(newValue%2==0){  /* 注意这里使用的时候不要用count，而是用newValue，并且括号里要写上引用的newValue变量 */
                        this.ishow = true
                    }else{
                        this.ishow =false
                    }
                },
                immediate:true,/* 不加上立即执行immediate时，页面加载不会执行count函数，如果此时有其他函数加了immediate，则会执行那个函数 */
                },
            }
        })
    </script>
</body>
```

这里将侦听事件写成了完全体形式，用handle和immediate配合watch属性在页面加载的时候就执行一次函数

1-1侦听属性的深入

HTML | 复制代码

```html
<body>
    <div id='app'>
        <!-- <button @click="count++">自增按钮</button> -->
        <button @click="number.count1++">自增按钮</button>
        <h2 v-show="ishow">这是一个动态显示的标题</h2>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
    <script>
        new Vue({
            el: '#app',
            data: {
                number: {
                    count1: 1,
                    count2: 2,
                    count3: 3
                },
                ishow: false
            },
            watch: {
                number: {
                    // handler function zc()
                    handler: function (newValue) {
                        if (newValue.count1 % 2 == 0) {
                            this.ishow == true
                        } else {
                            this.ishow == false
                        }
                    },
                    immediate:true,
                    deep:true
                }
            }
        })
    </script>
</body>
```

这里在data中又套了一个对象number，并且将三个数据存放到其中，那么在引用其中的数据count1的时候就需要在前面加个number，比如这里在自增按钮的地方就加入了number.count1++,下面调用newValue的时候也是newValue.count1，要注意的是这里监听要写最外面一层的number，在调用的时候要用属性调用里面的count，而不是直接去监听count的变化，这里的newValue也指的是number的变化。如果没有要求打开页面立即调用一次的话，也可以将 `immediate:true` 省略掉。如果单单对其中的某一项进行监比如对count1进行监听，也可以改写成以下形式：

HTML 复制代码

```html
<body>
    <div id='app'>
        <!-- <button @click="count++">自增按钮</button> -->
        <button @click="number.count1++">自增按钮</button>
        <h2 v-show="ishow">这是一个动态显示的标题</h2>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
    <script>
        new Vue({
            el: '#app',
            data: {
                number: {
                    count1: 1,
                    count2: 2,
                    count3: 3
                },
                ishow: false
            },
            watch: {
                'number.count1'(newValue) {
                    if (newValue % 2 == 0){
                        this.ishow = true
                    }else{
                        this.ishow = false
                    }
                }
            }
        })
    </script>
</body>
```

这种情况下监听对象就是number.count1了，然后要用字符串将其引起来，否则会报错，里面的newValue代表的也是number.count1的最新变化值，这样只能实现对于number.count1的监听，不能够实现对于number里面其他的数值变化 的监听，只有当count1发生变化时才会调用该函数。

在这个 Vue 代码中，`deep:true` 被用于 `watch` 属性，表示监视对象 `number` 的深层次属性变化，即当 `number` 内部嵌套的属性 `count1` 变化时，也会触发 `watch` 的回调函数。如果不使用 `deep:true`，则只有当 `number` 对象本身被赋予一个新的值时才会触发 `watch` 的回调函数。

在该代码中，当 `number.count1` 变化时，`watch` 回调函数被触发，并根据 `count1` 的奇偶性来改变 `ishow` 的值。由于使用了 `deep:true`，所以即使 `number` 对象中的其他属性变化，也

会触发 `watch` 回调函数，尽管它们不影响 `ishow` 的值。

2.Vue的双向绑定

HTML | 复制代码

```html
<body>
    <div id='app'>
        <input type="text" v-model:value="text1">
        <h1>{{text1}}</h1>
        <input type="text" v-model="text2">
        <h2>{{text2}}</h2>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
        <script>
             new Vue({
                el:'#app' ,
                data: {
                      text1:'',
                      text2:'',
                    },

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

这里通过v-model实现了输入框input属性值value的双向绑定，input的输入值会改变data中的数据，而data中的相应数据可以被调用到DOM文档中进行渲染，实现了双向绑定。

v-mode指令可以实现表单元素的双向数据绑定，因为只有表单元素才存在输入，所以能够实现双向绑定，其他大部分元素都是用来展示的

这两种写法都是对的，由于v-modle一般绑定的就是value属性，所以可以将value省略掉

3.表单元素单个复选框的双向绑定

HTML | 复制代码

```html
<body>
    <div id='app'>
        <input type="checkbox" id="checkbox" v-model:value="checked" value="111">
        <label for="checkbox">{{checked}}</label>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
    <script>
        new Vue({
            el: '#app',
            data: {
                checked:false
            },


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

这里面其实有点东西的，我来分析一下：

去掉lable标签，将data中的checked:''改为checked:[]，就发现每次选中都会出现对应的value值111，这里v-model绑定的其实是checked属性而不是value属性，value的定义就隐藏在input标签中，如果不用数组将其保存下来是看不到选中复选框后的value值的；

4.表单元素多个复选框的双向绑定

HTML | 复制代码

```html
<body>
    <div id='app'>
        <input type="checkbox" value="man" id="gender" v-model="checked">
        <input type="checkbox" value="woman" id="gender" v-model="checked"
>
        <h1>{{checked}}</h1>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
        <script>
             new Vue({
                el:'#app' ,
                data: {
                      checked:[]
                    },

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

多个复选框可以将checked设置为一个数组,必须设置好每一个input的value值，这里checked在data中用的是数组的形式来将复选框里value属性下的内容保存下来

5.单选框的数据绑定

HTML | 复制代码

```html
<body>
    <div id='app'>
        <input type="radio" value="man" v-model="picked">
        <label>男</label>
        <input type="radio" value="woman" v-model="picked">
        <label>女</label>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
        <script>
             new Vue({
                el:'#app' ,
                data: {
                      picked:''
                    },

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

单选框的v-modle绑定的是value属性，在上面的代码中，**v-model** 绑定到了 **picked** 变量，而不是 **value** 属性。当用户选中一个单选按钮时，与该按钮相关联的 **value** 值将被设置为 **picked** 变量的值。

需要注意的是，对于单选按钮，只有被选中的按钮的 **value** 值才会被设置为 **v-model** 绑定的数据属性的值。如果没有任何一个单选按钮被选中，**v-model** 绑定的数据属性的值将为 **undefined**。

6.下拉框的数据绑定

HTML | 复制代码

```html
<body>
    <div id='app'>

        <select v-model="selected" multiple>
            <option value="上海">上海</option>
            <option value="北京">北京</option>
            <option value="江苏">江苏</option>
        </select>
        <h2>{{selected}}</h2>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
        <script>
             new Vue({
                el:'#app' ,
                data: {
                      selected:[]
                    },
            })
        </script>
</body>
```

这里v-model绑定的是select下拉框的value属性，在上面的代码中，**v-model** 绑定到了 **selected** 变量，而这个 **selected** 变量的值将会是选中选项的 **value** 属性值。当用户选择一个选项时，**selected**变量的值将被设置为该选项的 **value** 值。

需要注意的是，如果 **<select>** 元素具有 **multiple** 属性（允许用户选择多个选项），则 **v-model** 绑定的数据属性应该是一个数组，以便存储用户选择的所有选项的值。

从上述几种情况可以看出，对于text、radio（单选框）、select（下拉框）等，v-model绑定的就是value属性值，对于checkbox而言绑定的是checked属性值，如果复选框和下拉框是多选的话，要用数组去保存选中的数值

7.修饰符

HTML | 复制代码

```html
<body>
    <div id="app">
        <!-- lazy修饰符可以让双向数据绑定监听的事件变成change，默认是监听input事件 -->
        <input type="text" v-model.lazy="msg">
        <!-- input中收集到的输入内容默认为字符串类型，如果希望变成数字类型 可以添加.number修饰符 -->
        <input type="text" v-model.number="number">
        <!-- 希望去掉搜集到的表单元素前后的空格，可以使用.trim修饰符，去掉字符串前后空格 -->
        <input type="text" v-model.trim="info">
        <h3>{{msg}}</h3>
        <h3>{{number+10}}</h3>
        <h3>{{info}}</h3>
    </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',
                number:0,
                info:'info'
            },
        })


    </script>
</body>
```

其中lazy修饰符可以起到监听的作用，如果不加这个修饰符的话，用v-model实现双向数据绑定会即时同步变更信息，即input框中输入什么，msg立马就会变成相应的数据，再即时反馈到DOM文档中；而加了这个修饰符后，只有当input输入框输完数据并离焦后，才会改变data中的数据，然后DOM文档才会渲染相应的元素。

lazy修饰符的作用是当文本框失去焦点时才触发更新数据，相当于离焦事件。而不加lazy修饰符时，文本框的输入即时触发更新数据，相当于输入事件。

number修饰符的作用是将输入框中收集到的用户输入的值强制转换为数字类型。因为默认情况下，输入框的值是字符串类型，如果要将其作为数字类型进行计算，则需要先将其转换为数字类型，使用number修饰符可以在绑定时自动为绑定值转型。

trim修饰符的作用是去除用户在输入值时可能输入的前后空格，保证表单的输入数据干净。在对用户输入的数据进行处理或验证时，如果不去除前后空格，就有可能因为空格的存在导致出现错误。因此使用trim修饰符可以提高数据的准确性和处理效率。

8.姓名拼接(即时)

HTML | 复制代码

```HTML
<body>
    <div id='app'>
        姓:<input type="text" v-model="firstName">
        名:<input type="text" v-model="lastName">
        <h1>姓名：{{fullName}}</h1>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
        <script>
             new Vue({
                el:'#app' ,
                data: {
                    firstName:'',
                    lastName:''
                    },
                computed: {
                    fullName() {
                      return this.firstName + this.lastName
                    },
                },
            })
        </script>
</body>
```

9.姓名拼接(延时)

HTML | 复制代码

```html
<body>
    <div id='app'>
        姓:<input type="text" v-model="firstName">
        名:<input type="text" v-model="lastName">
        <h1>姓名：{{fullName}}</h1>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
        <script>
            new Vue({
                el:'#app' ,
                data: {
                    firstName:'',
                    lastName:'' ,
                    fullName:''
                    },
                watch: {
                    firstName() {
                      setTimeout(()=>{
                        this.fullName = this.firstName + this.lastName
                      },3000)
                    },
                    lastName() {
                      setTimeout(()=>{
                        this.fullName = this.firstName + this.lastName
                      },3000)
                    },
                },
            })
        </script>
</body>
```

如果需要用到setTimeout()来做延时效果，即当姓或名改变时，不是立马输入到姓名输入框中而是有一个延迟的效果，因为setTimeout()方法相当于一个异步的函数，因此这里在watch监听的时候要加入箭头函数并且在里面加上this。

setTimeout 里面的this默认的是指向window，箭头函数里面的this指向父级元素，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

10.实现姓名的反向输出

HTML 复制代码

```html
<body>
    <div id='app'>
        姓:<input type="text" v-model="firstName">
        名:<input type="text" v-model="lastName">
        姓名: <input type="text" v-model="fullName">
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
    <script>
        new Vue({
            el: '#app',
            data: {
                firstName: '',
                lastName: '',
                fullName: ''
            },
            watch: {  /* 这里是watch不是watched */
                // fullName: {
                //     function() {
                //         this.firstName = this.fullName.slice(0, 1)   /* slice()方法是个函数不是数组，后面跟小括号 */
                //         this.lastName = this.fullName.slice(1)
                //     }
                // },
                fullName() { /* 上面那种写法是需要用到handler和immediate的时候的监听写法 */
                    this.firstName = this.fullName.slice(0, 1)   /* slice()方法是个函数不是数组，后面跟小括号 */
                    this.lastName = this.fullName.slice(1)
                }


            }


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

这里是对姓名fullName进行监听，根据fullName的实时数据来返回相应的姓和名，也是watch的一种简写方法

HTML | 复制代码

```html
<body>
    <div id='app'>
        姓:<input type="text" v-model="firstName">
        名:<input type="text" v-model="lastName">
        姓名: <input type="text" v-model="fullName">
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
    <script>
        new Vue({
            el: '#app',
            data: {
                firstName: '',
                lastName: '',
                // fullName: ''  这个不需要了，因为下面会用get得到一个用firstName和lastName返回值
            },
            computed: {  /* 这里可不是watch */
                fullName: {
                    get: function () {
                        return this.firstName + this.lastName
                    },
                    set: function (value) {
                        // this.firstName = this.value.slice(0,1)  第二个this不需要加
                        // this.lastName = this.value.slice(1)  第二个this不需要加
                        this.firstName = value.slice(0, 1)
                        this.lastName = value.slice(1)
                    }
                }
            }
        })


        /* 利用computed计算属性加上get获取值和set触发函数，实现了fullName同firstName和lastName的双向监听，get可以根据firstName以及lastName的值的变化来改变fullName的值，set则可以当fullName的值改变时触发函数重新给firstName和lastName赋值*/

        /* 以上代码中，`computed` 对象中的 `fullName` 属性被定义为一个计算属性。计算属性是依赖于其他属性值的属性，当依赖的属性发生变化时，计算属性也会被重新计算。
                    在这里，`fullName` 计算属性使用了一个 `get` 方法和一个 `set` 方法。当用户在页面中输入 `fullName` 的值时，`set` 方法将被触发。在 `set` 方法中，我们将输入的 `value` 字符串进行拆分，将第一个字符作为 `firstName` 的值，将其余字符作为 `lastName` 的值。因此，当 `set` 方法被调用时，`firstName` 和 `lastName` 的值也将被相应地更新。
```

```
                在 `get` 方法中，`fullName` 计算属性返回 `firstName` 和 `lastName` 的组合值。因此，当 `firstName` 或 `lastName` 的值发生更改时，`fullName` 的值也将相应地被重新计算并更新。
                总之，这段代码利用 `computed` 的计算属性特性，通过 `get` 和 `set` 方法监听 `fullName` 的变化，实现了在页面中输入 `fullName` 后，自动更新 `firstName` 和 `lastName` 的值的效果。 */
    </script>
</body>
```