Learning Everything

Stephen CUI[1]

January 14, 2022

[1]cuixuanStephen@gmail.com

Contents

Part I

从Vue2到Vue3

Chapter 1

Vue基础

1.1 初识Vue

1.2 模板语法

Vue模板语法有2大类：

1. 插值语法：用于解析标签体内容，主要写法是：`{{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：用于解析标签（包括：标签属性、标签体内容、绑定事件.....）。例如：`v-bind:href="xxx"`，xxx是JS表达式，且可以直接读取到data中的所有属性。

1.3 数据绑定

Vue中有两种数据绑定方式：

1. 单向数据绑定（v-bind）：数据只能从data流向页面。
2. 双向数据绑定（v-model）：数据不仅能从data流向页面，还可以从页面流向data。双向绑定一般都应用在表达类元素上，如input、select等，`v-model: value`可以简写为：`v-model`，因为`v-model`默认收集的就是value的值。

```
<div id="root">
    <input type="text" v-bind:value="name"><br />
    <input type="text" v-model:value="name">
    <input type="text" v-model="name">
</div>
```

1.4 el与data的两种写法

- el有2种写法：
  1. new Vue配置el属性
  2. 先创建Vue实例，然后通过`vm.$mount('#root')`指定el的值。
- data有2种写法：
  1. 对象式
  2. 函数式

一个重要的原则：由Vue管理的实例，一定不要写箭头函数，一旦写了箭头函数，this指针就不再是Vue实例了。

1.5 理解MVVM模型

1.6 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（变成了window）；
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；

1.6.1 事件修饰符

主要由以下6种事件修饰符，用于操作对事件的处理。

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；

事件修饰符可以连续写，比如`@click.stop.prevent`，先阻止冒泡，然后阻止默认事件。

1.6.2 键盘事件

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，必须配合keydown去使用)、上(up)、下(down)、左(left)、右(right)。

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）。

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。

此外，也可以使用keyCode去指定具体的按键（不推荐），`Vue.config.keyCodes.customName = keyCode`，可以去定制按键别名。

```
<div id="root">
    <h1>Hello, {{ name }}</h1>
    <input type="text" placeholder="enter for hint information" @keyup.enter="showInfo">
</div>
```

```
<script>
    new Vue({
        el: "#root",
        data: {
            name: 'Vue2-3'
        },
        methods: {
            showInfo(event) {
                // if (event.keyCode !== 13) return
                console.log(event.target.value)
            }
        }
    })
</script>
```

1.7 计算属性与侦听属性

1.7.1 计算属性

需要使用的属性不存在，要通过vm实例已有的属性（Property）计算得来，底层借助了`Object.defineproperty`方法提供的`getter`与`setter`。

get函数执行的时刻为以下2种：

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；

尽管使用methods方式以及插值方式都可以实现，但是计算属性由内部缓存机制（复用），调试方便。

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。

```
computed: {
    fullName: {
        get() {
            return this.firstName + '-' + this.lastName
        },

        set(value) {
            console.log('set', value);
            const arr = value.split('-');
            this.firstName = arr[0];
            this.lastName = arr[1];
        }
    }
}
```

如果不考虑修改计算属性，那么get的计算属性可以简写为：

```
computed: {
    fullName() {
        return this.firstName + ' ' + this.lastName
    }
}
```

1.7.2 侦听属性

侦听属性（watch），当被侦听的属性变化时, 回调函数自动调用, 进行相关操作，侦听的属性必须存在，才能进行侦听。侦听有2种写法：

1. new Vue时传入watch配置

```
watch: {
    isHot: {
        immediate: true,
        handler(newValue, oldValue) {
            console.log('isHot was modified', newValue, oldValue)
        }
    }
}
```

2. 通过vm.$watch侦听

```
vm.$watch('isHot', {
    immediate: true,
    handler(newValue, oldValue) {
        console.log('isHot was modified', newValue, oldValue)
    }
})
```

Vue中的watch默认不监测对象内部值的改变（一层、最外层），配置deep: true可以监测对象内部值的改变（内层）。

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，使用watch时根据数据具体结构，决定是否使用深度监测。

侦听多层级中某个属性的变化:

```
"numbers.a": {
    handler() {
        console.log('a was modified')
    }
}
```

侦听层级中任一属性值的改变：

```
numbers: {
    deep: true,
    handler() {
        console.log('numbers were modified')
    }
}
```

如果没有深层次的参数，那么侦听函数可以简写：

```
isHot(newValue, oldValue) {
    console.log('isHot was modified', newValue, oldValue)
},
```

computed与watch的对比：

1. computed能完成的功能，watch都能完成
2. watch能完成的功能，computed不一定能够完成，比如watch可以执行异步操作，而computed不能执行。

两个重要的原则：

Notes

1. 所有被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管理的函数（定时器的回调函数、Ajax的回调函数等），最好写成箭头函数，这样this的指向才是vm或者组件实例对象。

1.8 绑定样式

1.8.1 绑定class样式

字符串写法：样式的类名不确定，需要动态指定，但是其个数是确定的，仅追加一个样式值。

```
<div class="basic" :class="mood" @click="changeStyle">{{ mood  }}</div>
```

数组写法：要绑定的样式个数与名称均不确定。

```
<div class="basic" :class="moodArr">{{ mood }}</div>
```

对象写法：要绑定的样式个数确定、名字也确定，但是动态决定用不用。

```
<div class="basic" :class="moodObj">{{ mood }}</div>
```

1.8.2 绑定style样式

可以使用对象或者数组进行样式的绑定，但是实际中使用的不多。

```
<div class="basic" :style="styleObj">{{ mood }}</div>
<br>
<div class="basic" :style="[styleObj, styleObj2]">{{ mood }}</div>
<br>
<div class="basic" :style="styleArr">{{ mood }}</div>
```

```
fontSize: 40,
    styleObj: {
        fontSize: '40px',
        color: 'red',
    },
    styleObj2: {
        backgroundColor: 'orange'
    },
    styleArr: [
```

```
    {
        fontSize: '40px',
        color: 'red',
    },
    {
        backgroundColor: 'orange'
    }
]
```

1.9 条件渲染

v-show与v-if都可以实现条件渲染，但是v-if控制的组件节点直接不存在，隐藏的较为彻底。两种都可以使用表达式、布尔值以及Vue管理的属性值作为控制显示的条件。

```
<h1 v-show='showAttribute'>Hello, {{ name }}</h1>
<h1 v-show='true'>Hello, {{ name }}</h1>
<h1 v-show='1 === 3'>Hello, {{ name }}</h1>
<hr>
<h1 v-if='1 === 3'>Hello, {{ name }}</h1>
<h1 v-if='true'>Hello, {{ name }}</h1>
<h1 v-if='showAttribute'>Hello, {{ name }}</h1>
```

v-if适用于切换频率较低的场景、不展示的DOM元素直接被移除，其余v-else-if使用时不能被打断，需要联合使用，可以配合template使用。

v-show适用于切换频率较高的场景，不展示的DOM元素不会被移除，仅仅是使用样式隐藏掉。

1.10 列表渲染

列表的渲染可以使用v-for指令，其语法格式为：

```
v-for="(item, index)" in Object/Array/String/Numbers :key="index"
```

1.10.1 key的作用与原理

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：
   (a) 如果旧虚拟DOM中找到了与新虚拟DOM相同的key，则若虚拟DOM中内容没变, 直接使用之前的真实DOM!，否则生成新的真实DOM，随后替换掉页面中之前的真实DOM。

(b) 如果旧虚拟DOM中未找到与新虚拟DOM相同的key，则创建新的真实DOM，随后渲染到到页面。

2. 用index作为key可能会引发的问题：
   (a) 若对数据进行逆序添加、逆序删除等破坏顺序操作，则会产生没有必要的真实DOM更新（界面效果没问题, 但效率低）。
   (b) 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）。
3. 开发中如何选择key?
   (a) 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。
   (b) 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。

1.10.2 列表过滤

对于一串字符串，使用`indexOf`搜索，直接搜索空字符串也是可以返回结果的。

```
watch: {
    keyWord: {
        immediate: true,
        handler(val) {
            this.filterPersonArr = this.personArr.filter((value) => {
                return value.name.indexOf(val) !== -1
            })
        }
    }
}
```

对于这个功能，计算属性也可以实现，相比之下更加简单：

```
computed: {
    filterPersons() {
        return this.personArr.filter((value) => {
            return value.name.indexOf(this.keyWord) !== -1
        })
    }
}
```

1.10.3 列表排序

```
computed: {
    filterPersons() {
        const arr = this.personArr.filter((value) => {
            return value.name.indexOf(this.keyWord) !== -1
        })
        if (this.sortType) {
            arr.sort((p1, p2) => {
                return this.sortType === 1 ? p1.age - p2.age : p2.age - p1.age
            })
        }
        return arr
    }
}
```

1.10.4 数据监测

vue会监视data中所有层次的数据。对对象和数组的数据监视有所不同。

监视对象中的数据，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：
   (a) `Vue.set(target, propertyName/index, value)`
   (b) `vm.$set(target, propertyName/index, value)`

监视数组中的数据，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
3. 在Vue修改数组中的某个元素一定要用如下方法：
   (a) push()、pop()、shift()、unshift()、splice()、sort()、reverse()
   (b) `Vue.set()` 或 `vm.$set()`

Warnings

Vue.set() 和 vm.$set() 不能给vm 或 vm的根数据对象添加属性