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 计算属性与监视属性

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 监视属性

监视属性（watch），当被监视的属性变化时, 回调函数自动调用, 进行相关操作，监视的属性必须存在，才能进行监视。监视有2种写法：

1. new Vue时传入watch配置

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

2. 通过vm.$watch监视

```
vm.$watch('isHot', {
    immediate: true,
    handler(newValue, oldValue) {
        console.log(被修改了'isHot被修改了', 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。

监视多层级中某个属性的变化:

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

监测层级中任一属性值的改变:

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