# Vue官方文档学习

#### 数据与方法

当一个vue实例被创建时，他向vue的响应式系统中添加了其data对象中所能找到的所有的属性，所以将data对象中的属性的值发生改变时，视图将会产生响应

Data是一个函数 返回的是一个对象

Object.freeze()方法阻止修改现有的属性，页面也不会响应变化

var obj = { foo: 'bar' }

Object.freeze(obj)

new Vue({ el: '#app', data: obj })

<div id="app">

<p>{{ foo }}</p>

<!-- 这里的 `foo` 不会更新！ -->

<button v-on:click="foo = 'baz'">Change it</button>

</div>

Vue实例还暴露了一下有用的实例属性和方法，它们都有前缀$

var data = { a: 1 }

var vm = new Vue({ el: '#example', data: data })

vm.$data === data // => true

vm.$el === document.getElementById('example') // => true // $watch 是一个实例方法 vm.$watch('a', function (newValue, oldValue) { // 这个回调将在 `vm.a` 改变后调用 })

#### 模板语法

如果数据为html代码，想要绑定到模板中去 需要使用v-html指令

<p>Using v-html directive: <span v-html="rawHtml"></span></p>

v-html指令存在的span会不存在 而是直接显示绑定的html代码rawHtml

模板表达式都被放在沙盒中，只能访问全局变量的一个白名单，如math和date

不应该在模板表达式中试图去访问用户定义的全局变量

#### 计算属性

<div id="example">

<p>Original message: "{{ message }}"</p>

<p>Computed reversed message: "{{ reversedMessage }}"</p>

</div>

var vm = new Vue({

el: '#example',

data: { message: 'Hello' },

computed: {

// 计算属性的 getter

reversedMessage: function () {

// `this` 指向 vm 实例

return this.message.split('').reverse().join('')

}

}

})

这里我们声明了一个计算属性reverseMessage 我们提供的函数将用作属性vm.reversedMessage的getter函数

<p>Reversed message: "{{ reversedMessage() }}"</p>

methods: {

reversedMessage: function () {

return this.message.split('').reverse().join('')

}

}

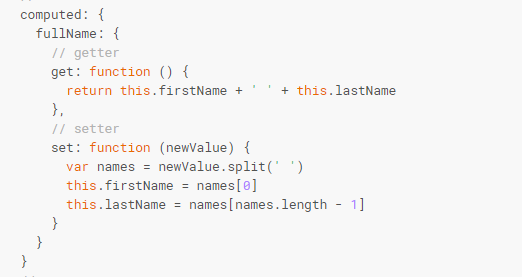
通过调用方法同样可以达到计算属性的效果

定义为一个计算属性是基于他们的依赖进行缓存的 只有当相关依赖改变时才会重新求值，如果依赖没有发生改变，计算属性会直接返回之前的计算结果

定义为一个方法是，调用方法将总会再次执行函数

定义为一个侦听属性（watch）也可以达到同样的效果，但是会很麻烦

计算属性默认只有getter 不过需要时也可以提供一个setter



在运行vm.fullName = ‘John Doe’时 setter会被调用

#### 侦听器 watch

使用watch选项允许我们执行异步操作，限制我们执行该操作的频率，并在我们得到最终结果前设置中间状态



#### 对象语法

我们可以给v-bind:class 一个对象，以动态地切换class

<div class="static" v-bind:class="{ active: isActive, 'text-danger': hasError }" ></div>

data: { isActive: true, hasError: false }

<div class="static active"></div>

class不需要一定内联在模板里

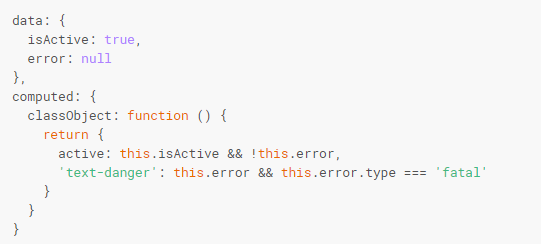
可以将class对象作为数据放在data中，

<div v-bind:class="classObject"></div>

data: { classObject: { active: true, 'text-danger': false } }

也可以作为计算属性

<div v-bind:class="classObject"></div>



#### 数组语法

<div v-bind:class="[activeClass, errorClass]"></div>

data: {

activeClass: 'active',

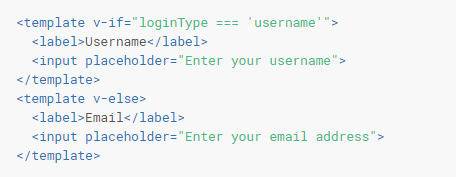
errorClass: 'text-danger'

}

<div v-bind:class="[isActive ? activeClass : '', errorClass]"></div>

<div v-bind:class="[{ active: isActive }, errorClass]"></div>

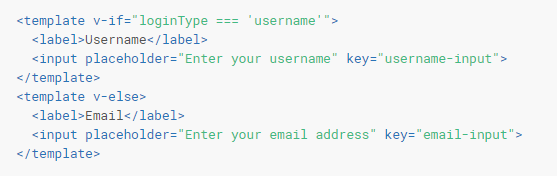
#### 用key管理可复用元素



此时input元素在切换时并没有重新渲染，只是替换掉了placeholder

所以在切换时，input中输入的内容并没有改变 所以需要key来标注两个元素是

独立的不需要复用



#### 一个对象的v-for

可以用v-for通过一个对象的属性来迭代

<div v-for="(value, key, index) in object"> {{ index }}. {{ key }}: {{ value }} </div>

new Vue({

el: '#v-for-object',

data: {

object: {

firstName: 'John',

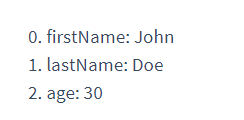
lastName: 'Doe',

age: 30

}

}

})



当使用v-for更新已经渲染过的元素列表时，它默认采用就地复用策略，

如果数据项的顺序被改变，vue将不会移动DOM元素来匹配数据项的顺序，而是简单的复用此处的每个元素，并且确保它在特定索引下显示已被渲染过的每个元素

如果想重新渲染每一个元素就需要为每一项提供key属性

#### 数组的变异方法和非变异方法

Push() pop() shift() unshift() splice() sort() reverse()

以上方法会修改原始数组 触发视图更新

filter() concat() slice()

以上方法不会修改原始数组，但是会返回一个新的数组

如果使用非变异方法也希望触发视图更新的话可以使用新数组替换旧数组



用一个含有相同元素的数组去替换原来的数组是非常高效的操作

##### 注意事项

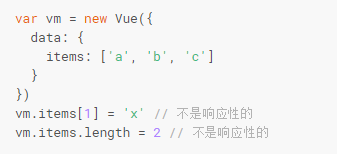
当我利用索引值直接设置一个项时

Vm.items[indexOfItem] = newValue

当我修改数组长度时

Vm.items.length = newLength

Vue是不能检测到以下数组的变动的，因此也无法触发视图更新



解决方法

Vue.set(vm.items,indexOfItem,newValue)

Vm.items.splice(indexOfItem,1,newValue)

Vm.$set是全局方法Vue.set的一个别名

Vm.$set(vm.items,indexOfItem,newVaule)

Vm.items.splice(newLength)

#### 对象更改检测注意事项

对于已经创建的实例，vue不能动态添加根级别的响应式属性，但是可以使用vue.set(object,key,value)方法向嵌套对象添加响应式属性

var vm = new Vue({

data: {

userProfile: { name: 'Anika' }

}

})

**正确方式**

Vue.set(vm.userProfile, ’age’, 27)

Vm.$set(vm.userProfile, ’age’, 27)

Vm.userProfile = Object.assign({}, vm.userProfile, {

age:27,

favoriteColor:’Vue Green’

})

**错误方式**

Object.assign(vm.userProfile, {

age:27，

favoriteColor：’vue green’

})

**Object.assign()** 方法用于将所有可枚举属性的值从一个或多个源对象复制到目标对象。它将返回目标对象 相同属性以后一个对象为准