VUE官网笔记整理

# 基础

## 安装

兼容：VUE不兼容IE8以下的浏览器

插件：Vue Devtools

CND方式: <script src="https://cdn.jsdelivr.net/npm/vue@2.5.17/dist/vue.js"></script> //通过script直接引入

NPM方式：npm install vue //最新稳定版

CLI（命令行工具）：npm i vue-cli

## 介绍

Script引入：<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script> //此为开发版本

<script src="https://cdn.jsdelivr.net/npm/vue"></script> //此为生产版本

## 声明式渲染

<div id="app">

<p>{{ message }}</p> //直接用模板渲染数据

<input type="text" name="" id="" value="" v-model="message"/> //用model双向数据绑定

<p v-bind:title="message2">悬停查看提示信息</p> //bind数据绑定属性

<p v-if="seen">you can see me</p> //判断是否显示

<ul>

<li v-for="item in list">{{item.text}}</li> //for循环数据

<mycomponent v-for="item in list" v-bind:data='item' v-bind:key="item.id"></mycomponet> //vue组件

</ul>

<button v-on:click="reset">reset message</button> //按钮事件

</div>

Vue.component('mycomponent',{

props: ['data'],

template: '<li>{{data.text}}</li>'

}) //设置自定义组件

var app = new Vue({

el: '#app',

data: {

message: 'Hello Vue!', //定义数据

message2: '页面加载于 ' + new Date().toLocaleString(),

seen: true,

list: [

{text: 'asd',id: '01'},

{text: 'bdf',id: '02'},

{text: 'dagad',id: '03'}

]

},

methods: { //挂载

reset: function(){

var \_self =this;

\_self.message = \_self.message.split('').reverse().join('')

}

}

})

## 模板语法

### 缩写

v-bind:attr='' === :attr=''

v-on:click='' === @click=''

## 计算属性和侦听器

### 计算属性

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

computed: {  
 reversedMessage: function () {  
 return this.message.split('').reverse().join('')  
 }  
 }

当 vm.message 发生改变时，所有依赖 vm.reversedMessage 的绑定也会更新

#### 计算属性缓存

计算属性是基于它们的依赖进行缓存的。只在相关依赖发生改变时它们才会重新求值。例如

computed: {  
 now: function () {  
 return Date.now()  
 }  
}

Date.now() 不是响应式依赖，所以重复使用now获取到的都是同一个时间，相比之下，每当触发重新渲染时，调用方法将**总会**再次执行函数。

#### 计算属性与侦听器

watch: {  
 firstName: function (val) {  
 this.fullName = val + ' ' + this.lastName  
 },  
 lastName: function (val) {  
 this.fullName = this.firstName + ' ' + val  
 }  
 }

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

当你有一些数据需要随着其它数据变动而变动时，更好的做法是使用计算属性而不是命令式的 watch 回调。

### 侦听器

需要在数据变化时执行异步或开销较大的操作时，这个方式是最有用的

<input v-model="question">  
watch: {  
   // 如果 `question` 发生改变，这个函数就会运行  
 question: function (newQuestion, oldQuestion) {  
 this.answer = 'Waiting for you to stop typing...'  
 this.debouncedGetAnswer()  
 }  
 }

## Class与style绑定

### 绑定HTML Class

#### 对象语法

<div v-bind:class="{ active: isActive }"></div>

active 这个 class 存在与否将取决于数据属性 isActive 的 [**truthiness**](https://developer.mozilla.org/zh-CN/docs/Glossary/Truthy)

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

Class与:class**共存**，切:class可以存在多个变量

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

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

也可不必写在**内联**里

computed: {  
 classObject: function () {  
 return {  
 active: this.isActive && !this.error,  
 'text-danger': this.error && this.error.type === 'fatal'  
 }  
 }  
}

也可以是**计算属性**

#### 数组语法

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

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

可以是三元

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

数组中写三元语法

#### 组件上使用

Vue.component('my-component', {  
 template: '<p class="foo bar">Hi</p>'  
})

<my-component class="baz boo"></my-component>

直接添加，也可以使用对象或数组语法

### 绑定内联样式

#### 对象语法

CSS 属性名可以用驼峰式 (camelCase) 或短横线分隔 (kebab-case，记得用单引号括起来) 来命名：

<div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>

直接绑定对象

|  |
| --- |
| <div v-bind:style="styleObject"></div> |

|  |
| --- |
| data: {  styleObject: {  color: 'red',  fontSize: '13px'  } } |

#### 数组语法

可以将多个样式绑定到元素上

<div v-bind:style="[baseStyles, overridingStyles]"></div>

#### 自动添加前缀

当 v-bind:style 使用需要添加[**浏览器引擎前缀**](https://developer.mozilla.org/zh-CN/docs/Glossary/Vendor_Prefix)的 CSS 属性时，如 transform，Vue.js 会自动侦测并添加相应的前缀。

#### 多重值

从 2.3.0 起你可以为 style 绑定中的属性提供一个包含多个值的数组，常用于提供多个带前缀的值，例如：

<div :style="{ display: ['-webkit-box', '-ms-flexbox', 'flex'] }"></div>

只会渲染数组中最后一个被浏览器支持的值

## 条件渲染

### v-if

<h1 v-if="ok">Yes</h1>

<h1 v-else>No</h1>

#### Template元素上使用

切换多个元素时，可以把一个 <template> 元素当做不可见的包裹元素，并在上面使用 v-if。最终的渲染结果将不包含 <template> 元素。

<template v-if="ok">  
 <h1>Title</h1>  
 <p>Paragraph 1</p>  
 <p>Paragraph 2</p>  
</template>

#### v-else

v-else 元素必须紧跟在带 v-if 或者 v-else-if 的元素的后面，否则它将不会被识别。

#### v-else-if

版本：**2.1.0 新增**

类似于 v-else，v-else-if 也必须紧跟在带 v-if 或者 v-else-if 的元素之后。

#### 用key管理可复用的元素

Vue 会尽可能高效地渲染元素，通常会复用已有元素而不是从头开始渲染。这么做除了使 Vue 变得非常快之外，还有其它一些好处。例如，如果你允许用户在不同的登录方式之间切换：

<template v-if="loginType === 'username'">  
 <label>Username</label>  
 <input placeholder="Enter your username">  
</template>  
<template v-else>  
 <label>Email</label>  
 <input placeholder="Enter your email address">  
</template>

再重复的input上增加key，切换时输入框都将被重新渲染

<template v-if="loginType === 'username'">  
 <label>Username</label>  
 <input placeholder="Enter your username" key="username-input">  
</template>  
<template v-else>  
 <label>Email</label>  
 <input placeholder="Enter your email address" key="email-input">  
</template>

### v-show

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 CSS 属性 display

<h1 v-show="ok">Hello!</h1>

### v-if与v-show

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。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

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。

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

### v-if与v-for

**不推荐**同时使用 v-if 和 v-for

## 列表渲染

### 遍历数组

items 是源数据数组并且 item 是数组元素迭代的别名

<li v-for="item in items">  
 {{ item.message }}  
</li>

data: {  
 items: [  
 { message: 'Foo' },  
 { message: 'Bar' }  
 ]  
 }

v-for 还支持一个可选的第二个参数为当前项的索引

<li v-for="(item, index) in items">  
 {{ parentMessage }} - {{ index }} - {{ item.message }}  
</li>

可以用 of 替代 in 作为分隔符，因为它是最接近 JavaScript 迭代器的语法

<div v-for="item of items"></div>

### 遍历对象

<li v-for="value in object">  
 {{ value }}  
</li>

object: {  
 firstName: 'John',  
 lastName: 'Doe',  
 age: 30  
}

第二个的参数为键名

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

第三个参数为索引

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

Ps: 在遍历对象时，是按 Object.keys() 的结果遍历，但是不能保证它的结果在不同的 JavaScript 引擎下是一致的

### key

当 Vue.js 用 v-for 正在更新已渲染过的元素列表时，它默认用“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， 而是简单复用此处每个元素，并且确保它在特定索引下显示已被渲染过的每个元素.

这个默认的模式是高效的，但是只适用于**不依赖子组件状态或临时 DOM 状态 (例如：表单输入值) 的列表渲染输出**。

它能跟踪每个节点的身份，从而重用和重新排序现有元素，你需要为每项提供一个唯一 key 属性。理想的 key 值是每项都有的唯一 id。

<div v-for="item in items" :key="item.id">  
 <!-- 内容 -->  
</div>

### 数组更新检测

#### 变异方法

以下方法会触发视图更新

push、pop、shift、unshift、splice、sort、reverse

#### 替换数组

变异方法 (mutation method)，顾名思义，会改变被这些方法调用的原始数组。相比之下，也有非变异 (non-mutating method) 方法，例如：filter(), concat() 和 slice() 。这些不会改变原始数组，但**总是返回一个新数组**。当使用非变异方法时，可以用新数组替换旧数组：

example1.items = example1.items.filter(function (item) {  
 return item.message.match(/Foo/)  
})

#### 注意事项

由于 JavaScript 的限制，Vue 不能检测以下变动的数组：

1，当你利用索引直接设置一个项时，例如：vm.items[indexOfItem] = newValue

2，,当你修改数组的长度时，例如：vm.items.length = newLength

var vm = new Vue({  
 data: {  
 items: ['a', 'b', 'c']  
 }  
})  
vm.items[1] = 'x' // 不是响应性的  
vm.items.length = 2 // 不是响应性的

可以使用以下方法

|  |
| --- |
| // Vue.set Vue.set(vm.items, indexOfItem, newValue) |

|  |
| --- |
| // Array.prototype.splice vm.items.splice(indexOfItem, 1, newValue) |

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

### 对象更改检测注意事项

还是由于 JavaScript 的限制，**Vue 不能检测对象属性的添加或删除**：

可以使用Vue.set(object, key, value)

使用 Object.assign() 或 \_.extend()时应该使用：

vm.userProfile = Object.assign({}, vm.userProfile, {  
 age: 27,  
 favoriteColor: 'Vue Green'  
})

### 显示过滤/排序结果

可以创建返回过滤或排序数组的计算属性

<li v-for="n in evenNumbers">{{ n }}</li>

data: {  
 numbers: [ 1, 2, 3, 4, 5 ]  
},  
computed: {  
 evenNumbers: function () {  
 return this.numbers.filter(function (number) {  
 return number % 2 === 0  
 })  
 }  
}

在计算属性不适用的情况下可以使用method方法

methods: {  
 even: function (numbers) {  
 return numbers.filter(function (number) {  
 return number % 2 === 0  
 })  
 }  
}

### 有范围的v-for

<span v-for="n in 10">{{ n }} </span>

### Template上使用v-for

<ul>  
 <template v-for="item in items">  
 <li>{{ item.msg }}</li>  
 <li class="divider" role="presentation"></li>  
 </template>  
</ul>

### v-for与v-if

当它们处于同一节点，v-for 的优先级比 v-if 更高，这意味着 v-if 将分别重复运行于每个 v-for 循环中。当你想为仅有的一些项渲染节点时，这种优先级的机制会十分有用

<li v-for="todo in todos" v-if="!todo.isComplete">  
 {{ todo }}  
</li>

### 一个组件的v-for

可以正常使用， 只不过需要通过组件的porp传入值

<my-component v-for="item in items" :key="item.id"></my-component>

<my-component  
 v-for="(item, index) in items"  
 v-bind:item="item"  
 v-bind:index="index"  
 v-bind:key="item.id"  
></my-component>

is="todo-item" 属性可以避开一些潜在的浏览器解析错误

<ul>  
 <li  
 is="todo-item"  
 v-for="(todo, index) in todos"  
 v-bind:key="todo.id"  
 v-bind:title="todo.title"  
 v-on:remove="todos.splice(index, 1)"  
 ></li>  
 </ul>

## 事件处理

### 监听事件

<button v-on:click="counter += 1">Add 1</button>

### 事件处理方法

<button v-on:click="greet">Greet</button>

methods: {  
 greet: function (event) {  
 // `this` 在方法里指向当前 Vue 实例  
 alert('Hello ' + this.name + '!')  
 // `event` 是原生 DOM 事件  
 if (event) {  
 alert(event.target.tagName)  
 }  
 }  
 }

### 内联处理器中的方法

<button v-on:click="say('hi')">Say hi</button>

methods: {  
 say: function (message) {  
 alert(message)  
 }  
 }

可以用特殊变量 $event 把它传入方法

<button v-on:click="warn('Form cannot be submitted yet.', $event)">  
 Submit  
</button>

methods: {  
 warn: function (message, event) {  
 // 现在我们可以访问原生事件对象  
 if (event) event.preventDefault()  
 alert(message)  
 }  
}

### 事件修饰符

Vue提供了以下修饰符

.stop 阻止单击事件继续传播

.prevent 阻止默认事件，例如提交是不刷新页面

.captrue 事件捕获模式，触发的事件现在此处处理，然后才由内部元素处理

.self 仅在当前元素触发

.once 只会触发一次 ps:2.1.3新增

.passive 触发事件的默认行为，主要用在移动端滚动事件

修饰符顺序很重要v-on:click.prevent.self 会阻止**所有的点击**，而 v-on:click.self.prevent 只会阻止对元素自身的点击。

不要把 .passive 和 .prevent 一起使用，因为 .prevent 将会被忽略，同时浏览器可能会向你展示一个警告。请记住，.passive 会告诉浏览器你不想阻止事件的默认行为。

### 按键修饰符

<!-- 只有在 `keyCode` 是 13 时调用 `vm.submit()` -->  
<input v-on:keyup.13="submit">

别名

<!-- 同上 -->  
<input v-on:keyup.enter="submit">

全部默认按键别名有

.enter

.tab

.delete 捕获删除与退格键

.esc

.space

.up

.down

.left

.right

可以通过全局 config.keyCodes 对象**[自定义按键修饰符别名](https://cn.vuejs.org/v2/api/" \l "keyCodes)**：

|  |
| --- |
| // 可以使用 `v-on:keyup.f1` Vue.config.keyCodes.f1 = 112 |

#### 自动匹配按键修饰符

可直接将 **[KeyboardEvent.key](https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/key/Key_Values" \t "_blank)** 暴露的任意有效按键名转换为 kebab-case 来作为修饰符：

<input @keyup.page-down="onPageDown">

有一些按键 (.esc 以及所有的方向键) 在 IE9 中有不同的 key 值, 如果你想支持 IE9，它们的内置别名应该是首选

### 系统修饰键

2.1.0新增

.ctrl

.alt

.shift

.meta

**Ps：在 Mac 系统键盘上，meta 对应 command 键 (⌘)。在 Windows 系统键盘 meta 对应 Windows 徽标键 (⊞)。在 Sun 操作系统键盘上，meta 对应实心宝石键 (◆)。在其他特定键盘上，尤其在 MIT 和 Lisp 机器的键盘、以及其后继产品，比如 Knight 键盘、space-cadet 键盘，meta 被标记为“META”。在 Symbolics 键盘上，meta 被标记为“META”或者“Meta”**

实例

<!-- Alt + C -->  
<input @keyup.alt.67="clear">  
<!-- Ctrl + Click -->  
<div @click.ctrl="doSomething">Do something</div>

Ps：请注意修饰键与常规按键不同，在和 keyup 事件一起用时，事件触发时修饰键必须处于按下状态。换句话说，只有在按住 ctrl 的情况下释放其它按键，才能触发 keyup.ctrl。而单单释放 ctrl 也不会触发事件。如果你想要这样的行为，请为 ctrl 换用 keyCode：keyup.17

#### .exact修饰符

2.5.0新增

.exact 修饰符允许你控制由精确的系统修饰符组合触发的事件。

<!-- 即使 Alt 或 Shift 被一同按下时也会触发 -->  
<button @click.ctrl="onClick">A</button>  
  
<!-- 有且只有 Ctrl 被按下的时候才触发 -->  
<button @click.ctrl.exact="onCtrlClick">A</button>  
  
<!-- 没有任何系统修饰符被按下的时候才触发 -->  
<button @click.exact="onClick">A</button>

#### 鼠标按钮修饰符

2.2.0新增

这些修饰符会限制处理函数仅响应特定的鼠标按钮。

.left

.right

.middle

### 为什么在HTML中监听事件

你可能注意到这种事件监听的方式违背了关注点分离 (separation of concern) 这个长期以来的优良传统。但不必担心，因为所有的 Vue.js 事件处理方法和表达式都严格绑定在当前视图的 ViewModel 上，它不会导致任何维护上的困难。实际上，使用 v-on 有几个好处：

1，扫一眼 HTML 模板便能轻松定位在 JavaScript 代码里对应的方法。

2，因为你无须在 JavaScript 里手动绑定事件，你的 ViewModel 代码可以是非常纯粹的逻辑，和 DOM 完全解耦，更易于测试。

3，当一个 ViewModel 被销毁时，所有的事件处理器都会自动被删除。你无须担心如何清理它们。

## 表单输入绑定

### 基础用法