

Vue.js

——渐进式JavaScript 框架

# 什么是Vue

Vue (读音 /vjuː/，类似于 view) 是一套用于构建用户界面的前端框架（[https://cn.vuejs.org](https://cn.vuejs.org/)）。它易于上手功能强大，并且其**渐进式**的设计理念更增加了它使用的灵活性：可以使用各个部件构建成完整的框架，也可以只当作js库来使用其核心功能，**声明式渲染**。

**渐进式框架**

<https://www.zhihu.com/question/51907207>

各个部件之间相互独立，我们可以只是简单的使用其核心功能，而不需要为了使用框架而引入其他的部件，写无用但是框架规定必需要用的代码。

**声明式渲染**

Vue.js 的核心是一个允许采用简洁的语法来声明式地将数据渲染进 DOM 的系统，此时数据和 DOM 已经被建立了关联，所有东西都是**响应式的**，即改变数据后DOM元素会自动改变，在输入控件中修改值，数据也会相应改变。

# 创建第一个Vue实例

首先要引入Vue库，Vue只依赖于js，不依赖于jquery。

<div id="testVue">

**{{**message**}}**

**{{**withSuffix**}}**

</div>

<script src="https://cdn.jsdelivr.net/npm/vue@2.6.10/dist/vue.js"></script>

<script>

var vue = new Vue({

el: "#testVue",

data: {

message: "Vue实例",

isShow: true

},

computed: {

withSuffix: function(){

return this.message + "--firstVue";

}

},

methods: {

showClick: function () {

vue.isShow = !vue.isShow;

}

},

created: function () {

this.message = "My First Vue";

}

})

</script>

每个 Vue 应用都是通过用 Vue 函数创建一个新的 Vue 实例开始的。

el指定了该Vue实例的作用范围，只有是el指定的元素以及其子元素中才能使用这个Vue实例。

当一个 Vue 实例被创建时，它将 data 对象中的所有的属性加入到 Vue 的**响应式系统**中。当这些属性的值发生改变时，视图将会产生“响应”，即匹配更新为新的值。值得注意的是只有当实例被创建时就已经存在于 data 中的属性才是**响应式**的。

每个 Vue 实例在被创建时都要经过一系列的初始化过程——例如，需要设置数据监听、编译模板、将实例挂载到 DOM 并在数据变化时更新 DOM 等。同时在这个过程中也会运行一些叫做**生命周期钩子**的函数，这给了用户在不同阶段添加自己的代码的机会。比如 [created](https://cn.vuejs.org/v2/api/#created)。下图展示了实例的生命周期。



# Vue语法

Vue的基本语法包括以下几部分

1. 模板语法：包括插值和指令
2. 计算属性
3. 侦听器
4. Class与Style绑定
5. 条件渲染
6. 列表渲染
7. 事件处理
8. 表单输入绑定
9. 组件

其中模板语法是最基础的部分，它说明了怎么将响应式的数据绑定到html(View)中；计算属性可以看作对于数据属性的封装，并可以在数据属性改变时进行响应；侦听器用于监听某个数据属性，在其发生改变时执行方法；4~8是对于指令的具体使用；组件是可复用的 Vue 实例。下面我们具体查看每一部分的内容。

# 模板语法

Vue.js 使用了基于 HTML 的模板语法，允许开发者声明式地将 DOM 绑定至底层 Vue 实例的数据。所有 Vue.js 的模板都是合法的 HTML ，所以能被遵循规范的浏览器和 HTML 解析器解析。

在底层的实现上，Vue 将模板编译成虚拟 DOM 渲染函数。结合响应系统，Vue 能够智能地计算出最少需要重新渲染多少组件，并把 DOM 操作次数减到最少。

**插值**

插值是指使用**“Mustache”语法 (双大括号)** 直接将数据属性插入到html中，Mustache 标签中可以放入数据属性也可以放入js代码，Mustache 标签将会被替代为数据属性的值或js代码的结果。此时绑定的数据属性发生改变，插值处的内容会更新。

通过使用[**v-once 指令**](https://cn.vuejs.org/v2/api/#v-once)，你也能执行一次性地插值，当数据改变时，插值处的内容不会更新。但请留心这会影响到该节点上的其它数据绑定。

例子：**{{**message**}} {{**message + ‘我是后缀’**}} {{**1+2+3**}} <span v-once>{{**message**}}</span>**

**指令**

刚刚使用到的v-once是一个指令，指令 (Directives) 是带有 v- 前缀的特殊特性。指令特性的值预期是**单个 JavaScript 表达式** (v-for是例外情况，稍后我们再讨论)。指令的职责是，当表达式的值改变时，将其产生的连带影响，响应式地作用于 DOM。

例子：<p v-if="seen">现在你看到我了</p>

这里，v-if 指令将根据表达式 seen 的值的真假来插入/移除 <p> 元素。

一些指令能够接收一个“参数”，在指令名称之后以冒号表示。例如，v-bind 指令可以用于响应式地更新 HTML 特性：

例子：<a v-bind:href="url">...</a>

在这里 href 是参数，告知 v-bind 指令将该元素的 href 特性与表达式 url 的值绑定。

另一个例子是 v-on 指令，它用于监听 DOM 事件：

例子：<a v-on:click="doSomething">...</a>

在这里参数是监听的事件名。我们也会更详细地讨论事件处理。

# 计算属性

模板内的表达式非常便利，但是设计它们的初衷是用于简单运算的。在模板中放入太多的逻辑会让模板过重且难以维护。例如：

<div id="example">

{{ message.split('').reverse().join('') }}

</div>

在这个地方，模板不再是简单的声明式逻辑。你必须看一段时间才能意识到，这里是想要显示变量 message 的翻转字符串。当你想要在模板中多次引用此处的翻转字符串时，就会更加难以处理。所以，对于任何复杂逻辑，你都应当使用计算属性。

计算属性可以理解为对数据属性的封装，并且数据属性发生改变时，与其有关的计算属性会自动更新并重新渲染绑定该计算属性的视图。计算属性声明在computed属性中。我们可以像使用普通的数据属性一样来使用计算属性

上述例子应用计算属性来实现：

<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('')

}

}

})

# 侦听器

计算属性可以在与其相关的属性发生变化时做出响应，但是它不应该作为为了监听数据变化而执行回调函数来使用。比如如下例子：

computed: {

productList: function () {

return getProductList(this.queryText);

}

},

计算属性productList与queryText相关，我们根据queryText的变化获取新的productList。我们的目标是在queryText发生变化时获取新的productList，productList并不是由queryText直接封装而来，此时不应该使用计算属性，而应该用侦听器：

watch: {

queryText: function () {

this.productList = getProductList(this.queryText);

}

}

侦听器声明在watch属性中，上面的监听器表示，监听queryText数据属性，当其发生变化时，执行回调方法。

# Class 与 Style 绑定

操作元素的 class 列表和内联样式是数据绑定的一个常见需求。因为它们都是属性，所以我们可以用 v-bind 处理它们：只需要通过表达式计算出字符串结果即可。当表达式中用到的数据属性或计算属性改变时，会重新渲染元素。

例子：

<div v-bind:class="vmClass" v-bind:style="vmStyle">aa</div>

data: {

vmClass: "aa",

vmStyle: 'background-color:red'

},

当然我们也可以使用表达式：

<div v-bind:class="isUsed ? vmClass : ''" v-bind:style="isUsed ? vmStyle: '' ">aa</div> data: {

isUsed:false,

vmClass: "aa",

vmStyle: 'background-color:red'

}

不过，字符串拼接麻烦且易错。因此，在将 v-bind 用于 class 和 style 时，Vue.js 做了专门的增强。表达式结果的类型除了字符串之外，还可以是对象或数组。

# 条件渲染

v-if 指令用于条件性地渲染一块内容。这块内容只会在指令的表达式返回 truthy 值的时候被渲染。

<h1 v-if="awesome">Vue is awesome!</h1>

也可以用 v-else-if 添加一个“else-if 块”，用v-else if 添加一个“else块”。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。

<h1 v-if="awesome==‘A’">Vue is A!</h1>

<h1 v-else-if="awesome==‘B’"> Vue is B!</h1>

<h1 v-else>Oh no 😢</h1>

因为 v-if 是一个指令，所以必须将它添加到一个元素上。但是如果想切换多个元素呢？此时可以把一个 <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>

另一个用于根据条件展示元素的选项是 v-show 指令。用法大致一样：

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

不同的是带有 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。

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。

v-if 也是**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。

相比之下，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。

一般来说，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。

# 列表渲染

我们可以用 v-for 指令基于一个数组来渲染一个列表。v-for 指令需要使用 item in items 形式的特殊语法，其中 items 是源数据数组，而 item 则是被迭代的数组元素的**别名**。

<ul id="example-1">

<li v-for="item in items">

{{ item.message }}

</li>

</ul>

var example1 = new Vue({

el: '#example-1',

data: {

items: [

{ message: 'Foo' },

{ message: 'Bar' }

]

}

})

在 v-for 块中，我们可以访问所有父作用域的属性。v-for 还支持一个可选的第二个参数，即当前项的索引。

<ul id="example-2">

<li v-for="(item, index) in items">

{{ parentMessage }} - {{ index }} - {{ item.message }}

</li>

</ul>

var example2 = new Vue({

el: '#example-2',

data: {

parentMessage: 'Parent',

items: [

{ message: 'Foo' },

{ message: 'Bar' }

]

}

})

# 事件处理

可以用 v-on 指令监听 DOM 事件，并在触发时运行一些 JavaScript 代码。

<div id="example-1">

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

<p>The button above has been clicked {{ counter }} times.</p>

</div>

var example1 = new Vue({

el: '#example-1',

data: {

counter: 0

}

})

然而许多事件处理逻辑会更为复杂，所以直接把 JavaScript 代码写在 v-on 指令中是不可行的。因此 v-on 还可以接收一个需要调用的方法名称。可调用的方法声明在methods属性中。

<div id="example-2">

<!-- `greet` 是在下面定义的方法名 -->

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

</div>

var example2 = new Vue({

el: '#example-2',

data: {

name: 'Vue.js'

},

// 在 `methods` 对象中定义方法

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)

}

}

}

})

在v-on中调用方法时，可以直接写方法名，不写括号，表示为不显示传递任何参数，此时会默认传递一个event参数，代表原生DOM事件。此时实际上是绑定了某个方法。如果写()，则代表使用js代码调用方法，括号中可以传递参数；此时不会默认传递event参数，如果需要，可以使用$event来显示传递。

# 表单输入绑定

可以用 v-model 指令在表单 <input>、<textarea> 及 <select> 元素上创建双向数据绑定。它会根据控件类型自动选取正确的方法来更新元素。

v-model 会忽略所有表单元素的 value、checked、selected 特性的初始值而总是将 Vue 实例的数据作为数据来源。

v-model 在内部为不同的输入元素使用不同的属性：

* text 和 textarea 元素使用 value 属性；
* checkbox 和 radio 使用 checked 属性；
* select 字段使用 value 属性。

例子：

**文本、单行文本**

<input v-model="message" placeholder="edit me">

<textarea v-model="message" placeholder="add multiple lines"></textarea>

实际上是将message赋值给value

[**复选框**](https://cn.vuejs.org/v2/guide/forms.html#%E5%A4%8D%E9%80%89%E6%A1%86)

单个复选框，绑定到布尔值，将vmChecked绑定到checked 属性，vmChecked为bool值：

<input type="checkbox" id="checkbox" v-model="vmChecked">

多个复选框，绑定到同一个数组，将**checkedNames中是否存在value属性的值**绑定到checked 属性：

<input type="checkbox" id="jack" value="Jack" v-model="checkedNames">

<label for="jack">Jack</label>

<input type="checkbox" id="john" value="John" v-model="checkedNames">

<label for="john">John</label>

<input type="checkbox" id="mike" value="Mike" v-model="checkedNames">

<label for="mike">Mike</label>

data: {

checkedNames: []

}

[**单选按钮**](https://cn.vuejs.org/v2/guide/forms.html#%E5%8D%95%E9%80%89%E6%8C%89%E9%92%AE)

<input type="radio" id="one" value="One" v-model="picked">

<label for="one">One</label>

<br>

<input type="radio" id="two" value="Two" v-model="picked">

<label for="two">Two</label>

data: {

picked: ''

}

将picked的值是否等于value属性的值绑定到checked属性上。

[**选择框**](https://cn.vuejs.org/v2/guide/forms.html#%E9%80%89%E6%8B%A9%E6%A1%86)

单选时绑定到字符串，将selected赋值给select的value

<select v-model="selected">

<option disabled value="">请选择</option>

<option>A</option>

<option>B</option>

<option>C</option>

</select>

data: {

selected: ''

}

多选时 绑定到一个数组，将selected赋值给select的value：

<select v-model="selected" multiple style="width: 50px;">

<option>A</option>

<option>B</option>

<option>C</option>

</select>

data: {

selected: []

}

# 组件

组件是可复用的 Vue 实例，且带有一个名字：在这个例子中是 <button-counter>。我们可以在一个通过 new Vue 创建的 Vue 根实例中，把这个组件作为自定义元素来使用：

Vue.component('button-counter', {

data: function () {

return {

count: 0

}

},

template: '<button v-on:click="count++">You clicked me {{ count }} times.</button>'

})

<div id="components-demo">

<button-counter></button-counter>

<button-counter></button-counter>

<button-counter></button-counter>

</div>

new Vue({ el: '#components-demo' })

因为组件是可复用的 Vue 实例，所以它们与 new Vue 接收相同的选项，例如 data、computed、watch、methods 以及生命周期钩子等。

当我们定义这个 <button-counter> 组件时，你可能会发现它的 data 并不是像这样直接提供一个对象：

data: {

count: 0

}

取而代之的是，**一个组件的 data 选项必须是一个函数**，因此每个实例可以维护一份被返回对象的独立的拷贝：

data: function () {

return {

count: 0

}

}

# end

本次分享只是Vue最核心内容的最基础的语法的使用，并且每一种语法种还有很多内容需要探讨，并且它们都很重要，这些将会再下次分享种来讨论。

下次分享内容：

1. 指令的动态参数、修饰符、缩写
2. 计算属性与方法和侦听器的对比
3. 计算属性的setter
4. Vue对于class和style绑定的增强
5. 在条件渲染和列表渲染种使用key
6. 在列表渲染中使用对象
7. 数组更新的渲染
8. 事件处理的修饰符
9. 表单输入绑定中的值绑定
10. 在数据更新后进行dom操作