# Vue

Vue.js是一个构建数据驱动的 web 界面的库。Vue.js 的目标是通过尽可能简单的 API 实现**响应的数据绑定**和**组合的视图组件**。

Vue.js 的核心是一个响应的数据绑定系统，它让数据与 DOM 保持同步非常简单。

## Vue 实例

## 构造器

每个 Vue.js 应用的起步都是通过构造函数 Vue 创建一个 **Vue 的根实例**：

var vm = new Vue({

// 选项

})

在实例化 Vue 时，需要传入一个**选项对象**，它可以包含数据、模板、挂载元素、方法、生命周期钩子等选项。全部的选项可以在 API 文档中查看。

一个 Vue 应用由一个通过 new Vue 创建的**根 Vue 实例**，以及可选的嵌套的、可复用的组件树组成。

可以扩展 Vue 构造器，从而用预定义选项创建可复用的**组件构造器**：

var MyComponent = Vue.extend({

// 扩展选项

})

// 所有的 `MyComponent` 实例都将以预定义的扩展选项被创建

var myComponentInstance = new MyComponent()

## 属性与方法

每个 Vue 实例都会**代理**其 data 对象里所有的属性：

var data = { a: 1 }

var vm = new Vue({

data: data

})

vm.a === data.a // -> true

// 设置属性也会影响到原始数据

vm.a = 2

data.a // -> 2

// ... 反之亦然

data.a = 3

vm.a // -> 3

Vue 实例暴露了一些有用的实例属性与方法。这些属性与方法都有前缀 $，以便与代理的数据属性区分。

## 实例生命周期

Vue 实例在创建时有一系列初始化步骤——例如，它需要建立数据观察，编译模板，创建必要的数据绑定。在此过程中，它也将调用一些**生命周期钩子**，给自定义逻辑提供运行机会。例如 created 钩子在实例创建后调用：

var vm = new Vue({

data: {

a: 1

},

created: function () {

// `this` 指向 vm 实例

console.log('a is: ' + this.a)

}

})

// -> "a is: 1"

也有一些其它的钩子，在实例生命周期的不同阶段调用，如 compiled、 ready 、destroyed。钩子的 this 指向调用它的 Vue 实例。一些用户可能会问 Vue.js 是否有“控制器”的概念？答案是，没有。组件的自定义逻辑可以分割在这些钩子中。

## 数据绑定语法

Vue.js 的模板是基于 DOM 实现的。这意味着所有的 Vue.js 模板都是可解析的有效的 [HTML](https://www.w3cschool.cn/html/html-tutorial.html)，且通过一些特殊的特性做了增强。Vue 模板因而从根本上不同于基于字符串的模板，请记住这点。

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。

## 插值

### 文本

数据绑定最基础的形式是文本插值，使用 "Mustache" 语法（双大括号）：

<span>Message: {{ msg }}</span>

Mustache 标签会被相应数据对象的 msg 属性的值替换。每当这个属性变化时它也会更新。

你也可以只处理单次插值，今后的数据变化就不会再引起插值更新了：

<span>This will never change: {{\* msg }}</span>

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。

<span v-once>这个将不会改变: {{ msg }}</span>

### [原始 HTML](https://cn.vuejs.org/v2/guide/syntax.html#%E5%8E%9F%E5%A7%8B-HTML)

双大括号会将数据解释为普通文本，而非 HTML 代码。为了输出真正的 HTML，你需要使用 v-html 指令：

<p>Using mustaches: {{ rawHtml }}</p>

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

### [特性](https://cn.vuejs.org/v2/guide/syntax.html#%E7%89%B9%E6%80%A7)

Mustache 语法不能作用在 HTML 特性上，遇到这种情况应该使用 [**v-bind 指令**](https://cn.vuejs.org/v2/api/#v-bind)：

<div v-bind:id="dynamicId"></div>

在布尔特性的情况下，它们的存在即暗示为 true，v-bind 工作起来略有不同，在这个例子中：

<button v-bind:disabled="isButtonDisabled">Button</button>

如果 isButtonDisabled 的值是 null、undefined 或 false，则 disabled 特性甚至不会被包含在渲染出来的 <button> 元素中。

### [使用 JavaScript 表达式](https://cn.vuejs.org/v2/guide/syntax.html#%E4%BD%BF%E7%94%A8-JavaScript-%E8%A1%A8%E8%BE%BE%E5%BC%8F)

迄今为止，在我们的模板中，我们一直都只绑定简单的属性键值。但实际上，对于所有的数据绑定，Vue.js 都提供了完全的 JavaScript 表达式支持。

{{ number + 1 }}

{{ ok ? 'YES' : 'NO' }}

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

<div v-bind:id="'list-' + id"></div>

这些表达式会在所属 Vue 实例的数据作用域下作为 JavaScript 被解析。有个限制就是，每个绑定都只能包含**单个表达式**，所以下面的例子都**不会**生效。

<!-- 这是语句，不是表达式 -->

{{ var a = 1 }}

<!-- 流控制也不会生效，请使用三元表达式 -->

{{ if (ok) { return message } }}

## [指令](https://cn.vuejs.org/v2/guide/syntax.html#%E6%8C%87%E4%BB%A4)

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。回顾我们在介绍中看到的例子：

<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。

### [参数](https://cn.vuejs.org/v2/guide/syntax.html#%E5%8F%82%E6%95%B0)

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：

<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：

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

### [动态参数](https://cn.vuejs.org/v2/guide/syntax.html#%E5%8A%A8%E6%80%81%E5%8F%82%E6%95%B0)

**2.6.0 新增**

从 2.6.0 开始，可以用方括号括起来的 JavaScript 表达式作为一个指令的参数：

<a v-bind:[attributeName]="url"> ... </a>

这里的 attributeName 会被作为一个 JavaScript 表达式进行动态求值，求得的值将会作为最终的参数来使用。例如，如果你的 Vue 实例有一个 data 属性 attributeName，其值为 "href"，那么这个绑定将等价于 v-bind:href。

同样地，你可以使用动态参数为一个动态的事件名绑定处理函数：

<a v-on:[eventName]="doSomething"> ... </a>

同样地，当 eventName 的值为 "focus" 时，v-on:[eventName] 将等价于 v-on:focus。

#### 对动态参数的值的约束

动态参数预期会求出一个字符串，异常情况下值为 null。这个特殊的 null 值可以被显性地用于移除绑定。任何其它非字符串类型的值都将会触发一个警告。

#### 对动态参数表达式的约束

动态参数表达式有一些语法约束，因为某些字符，例如空格和引号，放在 HTML 特性名里是无效的。同样，在 DOM 中使用模板时你需要回避大写键名。

1. 引入Vue:

**<!-- 开发环境版本，包含了有帮助的命令行警告 -->**

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

1. Vue.js 的核心是一个允许采用简洁的模板语法来声明式地将数据渲染进 DOM 的系统：

<div id="app">

{{ message }}

</div>

var app = new Vue({

el: '#app',

data: {

message: 'Hello Vue!'

}

})

1. v-bind 特性被称为**指令**。指令带有前缀 v-，以表示它们是 Vue 提供的特殊特性。它们会在渲染的 DOM 上应用特殊的响应式行为。在这里，该指令的意思是：“将这个元素节点的 title 特性和 Vue 实例的 message 属性保持一致”。

**<div id="app-2">**

**<span v-bind:title="message">**

**鼠标悬停几秒钟查看此处动态绑定的提示信息！**

**</span>**

**</div>**

**var app2 = new Vue({**

**el: '#app-2',**

**data: {**

**message: '页面加载于 ' + new Date().toLocaleString()**

**}**

**})**

1. 条件与循环

控制切换一个元素是否显示也相当简单：

v-if 条件指令

<div id="app-3">

<p v-if="seen">现在你看到我了</p>

</div>

var app3 = new Vue({

el: '#app-3',

data: {

seen: true

}

})

继续在控制台输入 app3.seen = false，你会发现之前显示的消息消失了。

v-for 指令可以绑定数组的数据来渲染一个项目列表：

<div id="app-4">

<ol>

<li v-for="todo in todos">

{{ todo.text }}

</li>

</ol>

</div>

var app4 = new Vue({

el: '#app-4',

data: {

todos: [

{ text: '学习 JavaScript' },

{ text: '学习 Vue' },

{ text: '整个牛项目' }

]

}

})

在控制台里，输入 app4.todos.push({ text: '新项目' })，你会发现列表最后添加了一个新项目。

1. v-on 指令添加一个事件监听器，通过它调用在 Vue 实例中定义的方法：

<div id="app-5">

<p>{{ message }}</p>

<button v-on:click="reverseMessage">逆转消息</button>

</div>

var app5 = new Vue({

el: '#app-5',

data: {

message: 'Hello Vue.js!'

},

methods: {

reverseMessage: function () {

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

}

}

})

1. v-model 指令，它能轻松实现表单输入和应用状态之间的双向绑定。

<div id="app-6">

<p>{{ message }}</p>

<input v-model="message">

</div>

var app6 = new Vue({

el: '#app-6',

data: {

message: 'Hello Vue!'

}

})

1. **创建一个Vue实例**

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：

var vm = new Vue({

// 选项

})

一个 Vue 应用由一个通过 new Vue 创建的**根 Vue 实例**，以及可选的嵌套的、可复用的组件树组成。

1. **数据与方法**

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。

// 我们的数据对象

var data = { a: 1 }

// 该对象被加入到一个 Vue 实例中

var vm = new Vue({

data: data

})

// 获得这个实例上的属性

// 返回源数据中对应的字段

vm.a == data.a // => true

// 设置属性也会影响到原始数据

vm.a = 2

data.a // => 2

// ……反之亦然

data.a = 3

vm.a // => 3

当这些数据改变时，视图会进行重渲染。

Vue 实例还暴露了一些有用的实例属性与方法。它们都有前缀 $，以便与用户定义的属性区分开来。

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

})

1. **实例生命周期钩子**

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。

[**created**](https://cn.vuejs.org/v2/api/#created) 钩子可以用来在一个实例被创建之后执行代码：

new Vue({

data: {

a: 1

},

created: function () {

// `this` 指向 vm 实例

console.log('a is: ' + this.a)

}

})

// => "a is: 1"

在实例生命周期的不同阶段被调用，如 [**mounted**](https://cn.vuejs.org/v2/api/#mounted)、**[updated](https://cn.vuejs.org/v2/api/" \l "updated)**和 [**destroyed**](https://cn.vuejs.org/v2/api/#destroyed)。生命周期钩子的 this 上下文指向调用它的 Vue 实例。

不要在选项属性或回调上使用[**箭头函数**](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Functions/Arrow_functions)，比如 created: () => console.log(this.a) 或 vm.$watch('a', newValue => this.myMethod())。因为箭头函数并没有 this，this 会作为变量一直向上级词法作用域查找，直至找到位置，经常导致 Uncaught TypeError: Cannot read property of undefined 或 Uncaught TypeError: this.myMethod is not a function 之类的错误。