# 第1章 概述

Vue 构造函数

Vue构造函数是 Vue.js 的核心。它是一个构造函数，允许您创建 Vue 实例。创建 Vue 实例很简单：

var vm = new Vue({ /\* 选项 \*/ })

当你实例化一个 Vue 实例时，你需要传入一个选项对象，它可以包含有关 DOM 元素、数据对象、mixin 方法、生命周期回调等的信息。请参阅组件选项的完整列表。

每个 Vue 实例本质上都是一个 ViewModel（因此vm您将在整个文档中看到变量名称）。它有一个关联的 DOM 元素$el，大致是 MVVM 中的 V。它还有一个关联的 JavaScript 对象$data，对应于 MVVM 中的 M。更改 M 会导致 V 中的更新。对于双向绑定，在 V 中触发的用户输入会导致 M 中的更改。有关 Vue 实例上可用属性的更多详细信息，请查看Instance Properties。

每个 Vue 实例也有许多实例方法，涵盖数据观察、事件通信和 DOM 操作。

构造Vue函数本身还公开了Global API，它允许您扩展Vue类、配置全局设置和注册全局自定义资产，例如组件、指令、过滤器等。

初始化

如果您在实例化时提供了el选项，Vue 实例将立即进入编译阶段。否则，它将等到vm.$mount()被调用后才开始编译。在编译阶段，Vue 遍历 DOM 并收集它运行的指令，并用这些指令“链接”数据和 DOM。一旦链接，这些 DOM 节点现在被称为由 Vue 实例管理。一个 DOM 节点只能由一个 Vue 实例管理，并且不应多次编译。

数据代理

Vue 实例代理访问它们的$data对象，所以如果你有vm.$data.msg你也可以访问它作为vm.msg. 这可能看起来有点神奇，但完全是可选的。您可以坚持vm.$data.msg更明确的数据访问。vm然而，注意和之间的区别仍然很重要vm.$data，因为其他 Vue 实例无法将前者作为数据观察到。

还值得注意的是，数据对象不一定属于单个 Vue 实例——多个 ViewModel 可以观察同一条数据，无论是直接作为$data还是嵌套在它下面。当多个组件需要对共享的全局状态对象做出反应时，这很有用。

成分

在构建复杂的 UI 时，我们经常需要将我们的应用程序构建为组件树。Vue.js 的组件系统可以让你做到这一点。可以扩展 Vue 构造函数来定义封装自己的模板和数据逻辑的组件，并且 Vue 提供了类似 Web Component 的语法供您在模板中组合组件。有关更多详细信息，请阅读指南的组件系统部分。

# 第2章 组件选项

数据

数据

类型： Object | Function

限制：仅Function在使用时接受Vue.extend()。

Vue 实例的数据对象。它可以通过以下方式访问vm.$data：

1

2

3

4

5

var data = { a: 1 }

var vm = new Vue({

data: data

})

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

Vue 实例将代理对其所有属性的访问，因此您可以操作 Vue 实例上的属性并将更改同步回实际的数据对象：

1

2

3

4

5

vm.a // -> 1

vm.a = 2

data.a // -> 2

data.a = 3

vm.a // -> 3

对象必须符合 JSON（无循环引用）。您可以像使用普通对象一样使用它，并且在使用JSON.stringify. 您还可以在多个 Vue 实例之间共享它。

data这里的一个特殊情况是使用Vue.extend(). 由于我们不希望从该扩展构造函数创建的所有实例共享嵌套对象，因此我们必须提供一个函数来返回默认数据的新副本：

1

2

3

4

5

6

7

8

9

10

var MyComponent = Vue.extend({

data: function () { return { message: '一些默认数据。' ， 对象：{ 新鲜：真实 } } } }）

在底层，Vue.js 附加了一个隐藏属性\_\_ob\_\_，并递归地将对象的可枚举属性转换为 getter 和 setter 以启用依赖项收集。带有以开头$或被\_跳过的键的属性。

方法

类型： Object

要混合到 Vue 实例中的方法。您可以直接在 VM 实例上访问这些方法，或在指令表达式中使用它们。所有方法的this上下文都会自动绑定到 Vue 实例。

例子：

1

2

3

4

5

6

7

8

9

10

var vm = new Vue({

data: { a: 1 },

methods: {

plus: function () { this .a++ } } }) vm.plus() vm.a // 2

计算的

类型： Object

要混合到 Vue 实例中的计算属性。所有 getter 和 setter 的this上下文都会自动绑定到 Vue 实例。

例子：

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

var vm = new Vue({

data: { a: 1 },

computed: { // get only, 只需要一个函数 aDouble: function (

) { return this .a \* 2 }, // get 和 set aPlus: { get: function (

) { return this .a + 1 }, set: function ( v ) { this .a = v - 1 } } } }) vm.aPlus // -> 2 vm.aPlus = 3 vm.a // -> 2 vm.aDouble // -> 4

参数属性

类型： Array

要在 Vue 实例上设置为初始数据的属性名称数组。在将数据传递给组件时很有用。

例子：

1

2

3

4

5

6

7

Vue.component( 'param-demo' , {

paramAttributes: [ 'size' , 'message' ],

编译: function () { console .log( this .size ) // -> 100 console .log( this .message) // -> 'hello!' } })

1

<参数演示 大小= “100” 消息= “你好！” > </参数演示>

参数属性也可以包含插值标签。插值将针对父对象进行评估，并且在幕后它们将被编译为v-with，这意味着当插值表达式的值发生变化时，组件的相应属性也将更新：

1

< param-demo message = "{{parentMessage}}" > </ param-demo >

关于连字符属性的注释

HTML 属性名称忽略大小写差异，因此我们通常使用连字符属性而不是驼峰式大小写。paramAttributes与包含连字符的属性一起使用时有一些特殊情况：

如果属性是数据属性，data-前缀将被自动剥离；

如果属性仍然包含破折号，它将被骆驼化。这是因为在模板中访问包含破折号的顶级属性很不方便：除非您使用笨拙的语法，否则表达式my-param将被解析为减号表达式。this['my-param']

这意味着参数属性data-hello将在 vm 上设置为vm.hello; 并将my-param设置为vm.myParam.

DOM

埃尔

类型： String | HTMLElement | Function

限制：Function仅在使用时接受类型Vue.extend()。

为 Vue 实例提供现有的 DOM 元素。它可以是 CSS 选择器字符串、实际的 HTMLElement 或返回 HTMLElement 的函数。解析的元素将可以作为vm.$el.

在 中使用时Vue.extend，必须提供一个函数，以便每个实例都获得一个单独创建的元素。

如果该选项在实例化时可用，则实例将立即进入编译；否则，用户将不得不显式调用vm.$mount()以手动启动编译。

模板

类型： String

要插入的字符串模板vm.$el。除非模板中存在内容插入点，否则内部的任何现有标记vm.$el都将被覆盖。如果替换选项是，则模板将完全替换。truevm.$el

如果它以它开头，#它将用作 querySelector 并使用所选元素的 innerHTML 和模板字符串。这允许使用通用<script type="x-template">技巧来包含模板。

Vue.js 使用基于 DOM 的模板。编译器遍历 DOM 元素并查找指令并创建数据绑定。这意味着所有 Vue.js 模板都是可解析的 HTML，可以被浏览器转换为实际的 DOM 元素。Vue.js 将字符串模板转换为 DOM 片段，以便在创建更多 Vue 实例时可以克隆它们。如果您希望您的模板是有效的 HTML，您可以将指令前缀配置为以data-.

代替

类型： Boolean

默认： false

限制：仅当模板选项也存在时才受到尊重。

是否将原始vm.$el内容替换为模板的内容，而不是附加到模板内容。

生命周期

所有生命周期钩子的this上下文都绑定到它们所属的 Vue 实例。Vue 实例还将为每个钩子以"hook:<hookName>". 例如 for ，将触发created一个事件。"hook:created"

创建

类型： Function

创建实例后同步调用。在这个阶段，实例已经完成了对选项的处理，这意味着已经设置了以下内容：数据观察、计算属性、方法、监视/事件回调。但是，DOM 编译尚未开始，该$el属性将不可用。

编译前

类型： Function

在编译开始之前调用。

编译

类型： Function

编译完成后调用。在这个阶段，所有指令都已链接，因此数据更改将触发 DOM 更新。但是，$el不保证已经插入到文档中。

准备好

类型： Function

编译后调用，第一次插入到文档中。请注意，此插入必须通过 Vue 执行（使用类似方法或作为指令更新的结果）才能触发钩子。$elvm.$appendTo()ready

随附的

类型： Function

当vm.$el通过指令或 VM 实例方法（例如$appendTo(). 直接操作vm.$el不会触发这个钩子。

分离的

类型： Function

vm.$el当被指令或 VM 实例方法从 DOM 中移除时调用。直接操作vm.$el不会触发这个钩子。

之前销毁

类型： Function

在 Vue 实例被销毁之前调用。在这个阶段，实例仍然可以正常工作。

毁坏

类型： Function

在 Vue 实例被销毁后调用。调用此钩子时，Vue 实例的所有绑定和指令都已解除绑定，并且所有子 Vue 实例也已销毁。

请注意，如果有离开转换，则在转换完成后destroyed调用挂钩。

资产

这些是私有资产，在编译期间只能用于此 Vue 实例及其子级。

指令

类型： Object

可供 Vue 实例使用的指令哈希。有关如何编写自定义指令的详细信息，请参阅编写自定义指令。

过滤器

类型： Object

可供 Vue 实例使用的过滤器哈希。有关如何编写自定义过滤器的详细信息，请参阅编写自定义过滤器。

成分

类型： Object

可供 Vue 实例使用的组件哈希。有关如何扩展和组合 Vue 实例的详细信息，请参阅组件系统。

部分

类型： Object

可供 Vue 实例使用的部分哈希。另请参阅v-partial。

过渡

类型： Object

可用于 Vue 实例的转换哈希。有关详细信息，请参阅过渡指南。

其他

继承

类型： Boolean

默认： false

是否继承父范围数据。true如果要创建继承父范围的组件，请将其设置为。当inherit设置为 时true，您可以：

绑定到组件模板中的父作用域属性；

通过原型继承直接访问组件实例本身的父属性。

使用时要知道的一件重要事情inherit: true是子级也可以设置父级属性，因为所有 Vue 实例数据属性都是 getter/setter。

例子：

1

2

3

4

5

6

7

8

9

10

11

12

13

var parent = new Vue({

data: { a: 1 }

})

var child = parent.$addChild({

inherit: true ,

data: { b: 2 }

})

child.a // -> 1

child.b / / -> 2

// 以下行修改了 parent.a

// 而不是在 child 上创建新属性：

child.a = 2

parent.a // -> 2

事件

一个对象，其中键是要侦听的事件，值是相应的回调。请注意，这些是 Vue 事件而不是 DOM 事件。该值也可以是方法名称的字符串。Vue 实例将$on()在实例化时调用对象中的每个条目。

例子：

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

var vm = new Vue({

events: { 'hook:created' : function (

) { console .log( 'created!' ) }, greeting: function ( msg ) { console .log(msg) }, // 也可以使用字符串作为方法 bye: 'sayGoodbye' }, methods: { sayGoodbye: function (

) { console .log( 'goodbye!' ) } } }) // -> 创建！vm.$emit( 'greeting' , 'hi!' ) // -> hi! vm.$emit( 'bye' ) // -> 再见！

手表

类型：Object

一个对象，其中键是要观察的表达式，值是相应的回调。该值也可以是方法名称的字符串。Vue 实例将$watch()在实例化时调用对象中的每个条目。

例子：

1

2

3

4

5

6

7

8

9

10

11

var vm = new Vue({

data: {

a: 1

},

watch: { 'a' : function ( val, oldVal ) { console .log( 'new: %s, old: %s' , val, oldVal) } } }) vm.a = 2 // -> 新：2，旧：1

混入

类型：Array

该mixins选项接受一个 mixin 对象数组。这些 mixin 对象可以像普通实例对象一样包含实例选项，并且它们将使用相同的选项合并逻辑与最终选项合并Vue.extend()。例如，如果你的 mixin 包含一个 created 钩子并且组件本身也有一个，那么这两个函数都会被调用。

例子：

1

2

3

4

5

6

7

8

9

var mixin = {

创建：函数() {控制台.log( 2 ) }

}

var vm = new Vue({

created: function () {控制台.log( 1 ) },

mixins: [mixin]

})

// -> 1

// -> 2

姓名

类型：String

限制：仅在Vue.extend().

在控制台中检查扩展的 Vue 组件时，默认的构造函数名称是VueComponent，它提供的信息不是很多。通过将可选name选项传递给Vue.extend()，您将获得更好的检查输出，以便您知道正在查看的组件。该字符串将被骆驼化并用作组件的构造函数名称。

例子：

1

2

3

4

5

var Ctor = Vue.extend({

name: 'cool-stuff'

})

var vm = new Ctor()

console .log(vm) // -> CoolStuff {$el: null,

# 第3章 实例属性

vm.$el

类型： HTMLElement

只读

Vue 实例管理的 DOM 元素。

vm.$data

类型： Object

Vue 实例正在观察的数据对象。您可以将其与新对象交换。Vue 实例代理访问其数据对象的属性。

vm.$options

类型： Object

用于当前 Vue 实例的实例化选项。当您想在选项中包含自定义属性时，这很有用：

1

2

3

4

5

6

new Vue({

customOption: 'foo' ,

created: function () {控制台.log( this .$options.customOption) // -> 'foo' } })

vm.$parent

类型： Vue

只读

父实例，如果当前实例有一个。

vm.$root

类型： Vue

只读

当前组件树的根 Vue 实例。如果当前实例没有父母，则该值将是它自己。

vm.$

类型： Object

只读

包含已v-ref注册子组件的对象。有关更多详细信息，请参阅v-ref。

vm.$$

类型： Object

只读

包含已v-el注册的 DOM 元素的对象。有关更多详细信息，请参阅v-el。

元属性

由创建的实例v-repeat也将具有一些元属性，vm.$index例如vm.$key和vm.$value。有关更多详细信息，请参阅使用指南v-repeat。

# 第4章 实例方法

数据

您可以观察 Vue 实例上的数据变化。请注意，所有监视回调都是异步触发的。此外，值更改在事件循环中进行批处理。这意味着当一个值在单个事件循环中多次更改时，回调将只被触发一次并使用最新的值。

vm.$watch( 表达式, 回调, [deep, immediate] )

表达 String

回调（新值，旧值） Function

深度 Boolean 可选

立即 Boolean 可选

观察 Vue 实例上的表达式以进行更改。表达式可以是单个键路径或实际表达式：

1

2

3

vm.$watch( 'a + b' , function ( newVal, oldVal ) { // 做点什么})

要同时检测对象内部的嵌套值更改，您需要传入true第三个deep参数。请注意，您不需要这样做来侦听 Array 突变。

1

2

3

vm.$watch( 'someObject' , callback, true )

vm.someObject.nestedValue = 123

// 回调被触发

传入true第四个immediate参数将立即使用表达式的当前值触发回调：

1

2

vm.$watch( 'a' , callback, false , true )

// 回调被立即触发，当前值为 `a`

最后，vm.$watch返回一个停止触发回调的 unwatch 函数：

1

2

3

var unwatch = vm.$watch( 'a' , cb)

// 稍后，拆除观察者

unwatch()

vm.$get( 表达式 )

表达 String

从给定表达式的 Vue 实例中检索一个值。抛出错误的表达式将被抑制并返回undefined。

vm.$set（键路径，值）

关键路径 String

价值 \*

给定有效的键路径，在 Vue 实例上设置数据值。如果路径不存在，它将被创建。

vm.$add（键路径，值）

关键路径 String

价值 \*

将根级别属性添加到 Vue 实例（以及它的$data）。由于 ES5 的限制，Vue 无法检测直接添加到 Object 中或从 Object 中删除的属性，因此vm.$delete请在需要时使用此方法。此外，所有观察到的对象也都使用这两种方法进行了扩充。

vm.$delete(keypath)

关键路径 String

删除 Vue 实例（以及它的$data）上的根级别属性。

vm.$eval( 表达式 )

表达 String

计算一个也可以包含过滤器的表达式。

1

2

// 假设 vm.msg = 'hello'

vm.$ eval ( 'msg | uppercase' ) // -> 'HELLO'

vm.$interpolate(templateString)

模板字符串 String

评估一段包含胡须插值的模板字符串。请注意，此方法仅执行字符串插值；不编译属性指令。

1

2

// 假设 vm.msg = 'hello'

vm.$interpolate( '{{msg}} world!' ) // -> 'hello world!'

vm.$log( [keypath] )

键路径 String 可选

将当前实例数据记录为普通对象，这比一堆 getter/setter 更易于控制台检查。也接受一个可选的键。

1

2

vm.$log() // 记录整个 ViewModel 数据

vm.$log( 'item' ) // 记录 vm.item

活动

每个 vm 也是一个事件发射器。当您有多个嵌套的 ViewModel 时，您可以使用事件系统在它们之间进行通信。

vm.$dispatch(事件，[args…])

事件 String

参数... 可选

从当前 vm 调度一个事件，该事件一直传播到其$root. 如果回调返回false，它将在其所有者实例处停止传播。

vm.$broadcast(事件，[args…])

事件 String

参数... 可选

向当前 vm 的所有子 vm 发出一个事件，该事件会一直向下进一步广播给他们的孩子。如果回调返回false，则其所有者实例将不再广播该事件。

vm.$emit( 事件，[args…] )

事件 String

参数... 可选

仅在此 vm 上触发事件。

vm.$on(事件，回调)

事件 String

打回来 Function

监听当前 vm 上的事件。

vm.$once(事件，回调)

事件 String

打回来 Function

为事件附加一个一次性监听器。

vm.$off([事件，回调])

事件 String 可选

回调 Function 可选

如果没有给出参数，则停止监听所有事件；如果仅给出事件，则删除该事件的所有回调；如果同时给出事件和回调，则仅删除该特定回调。

DOM

所有 vm DOM 操作方法都像它们的 jQuery 对应物一样工作 - 除了它们还会触发 Vue.js 转换（如果在 vm 上声明了任何转换）$el。有关过渡的更多详细信息，请参阅添加过渡效果。

vm.$appendTo(元素|选择器，[回调])

元素 HTMLElement| 选择器 String

回调 Function 可选

将 vm 附加$el到目标元素。参数可以是元素或 querySelector 字符串。

vm.$before(元素|选择器，[回调])

元素 HTMLElement| 选择器 String

回调 Function 可选

$el在目标元素之前插入 vm 。

vm.$after(元素|选择器，[回调])

元素 HTMLElement| 选择器 String

回调 Function 可选

$el在目标元素之后插入 vm 。

vm.$remove([回调])

回调 Function 可选

$el从 DOM 中删除虚拟机。

生命周期

vm.$mount([元素|选择器])

元素 HTMLElement| 选择器 String 可选

如果 Vue 实例在实例化时没有得到el选项，你可以手动调用$mount()为它分配一个元素并开始编译。如果没有提供参数，<div>将自动创建一个空的。调用$mount()已挂载的实例将无效。该方法返回实例本身，因此您可以在它之后链接其他实例方法。

vm.$destroy([删除])

删除 Boolean 可选

彻底摧毁一个虚拟机。清理它与其他现有 vm 的连接，解除其所有指令的绑定并将其$el从 DOM 中删除。此外， all$on和$watchlisteners 将被自动删除。

vm.$compile( 元素 )

元素 HTMLElement

部分编译一段 DOM（元素或 DocumentFragment）。该方法返回一个decompile函数，该函数拆除在此过程中创建的指令。请注意，反编译功能不会删除 DOM。此方法主要用于编写高级自定义指令。

vm.$addChild([选项，构造函数])

选项 Object 可选

构造函数 Function 可选

将子实例添加到当前实例。options 对象与手动实例化实例相同。或者，您可以传入一个从Vue.extend().

实例之间的父子关系有三个含义：

父母和孩子可以通过事件系统进行交流。

孩子可以访问所有父资产（例如自定义指令）。

如果继承父范围，则子级可以访问父范围数据属性。

# 第5章 全局 API

Vue.config

Vue.config是一个包含 Vue 全局设置的对象。以下是所有可用设置及其默认值的列表：

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

{ // 打印堆栈跟踪以获取警告？ debug: true , // 指令的属性前缀 prefix: 'v-' , // 插值分隔符// 对于 HTML 插值，添加// 1 个额外的最外层字符。 delimiters: [ '{{' , '}}' ], // 禁止警告？ silent: false , // 插入小胡子绑定？ interpolate: true , // 使用异步更新（用于指令和观察者）？async : true , // 允许改变观察到的 Array 的原型链？ 原型：真

}

您可以直接修改它们，例如：

1

Vue.config.debug = true // 开启调试模式

调试模式

当Vue.config.debug设置为 true 时，Vue 会自动使用同步模式，并debugger在出现警告时抛出语句。这使用户能够在浏览器开发工具中检查完整的堆栈跟踪。

调试模式在缩小的生产版本中不可用。

更改分隔符

当为文本插值设置分隔符时，将通过在两侧添加一个最外层符号来生成 HTML 插值的分隔符：

1

2

3

Vue.config.delimiters = [ '(%' , '%)' ]

// 标签现在是 (% %) 用于文本

// 和 ((% %)) 用于 HTML

Vue.extend( 选项 )

选项 Object

创建基本 Vue 构造函数的“子类”。所有实例化选项都可以在这里使用。这里要注意的特殊情况是eland data，在这种情况下必须是函数。

在内部，Vue.extend()在实例化组件之前调用所有组件选项。有关组件的更多详细信息，请参阅组件系统。

例子

1

2

3

4

5

6

7

8

9

10

11

12

13

14

var Profile = Vue.extend({

el: function () { return document .createElement( 'p' ) }, template: '{{firstName}} {{lastName}} aka {{alias}}' }) var profile = new Profile({ data: { firstName : 'Walter' , lastName : 'White' , 别名 : 'Heisenberg' } }) profile.$appendTo( 'body' )

将导致：

1

< p >沃尔特怀特又名海森堡</ p >

Vue.directive（id，[定义]）

ID String

定义 Function或Object 可选

注册或检索全局自定义指令。有关更多详细信息，请参阅编写自定义指令。

Vue.filter( id, [定义] )

ID String

定义 Function 可选

注册或检索全局自定义过滤器。有关更多详细信息，请参阅编写自定义过滤器。

Vue.component( id, [定义] )

ID String

定义 Function Constructor或Object 可选

注册或检索全局组件。有关更多详细信息，请参阅组件系统。

Vue.transition（id，[定义]）

ID String

定义 Object 可选

注册或检索全局 JavaScript 过渡效果定义。有关更多详细信息，请参阅JavaScript 转换指南。

Vue.partial( id, [定义] )

ID String

定义 String | Node 可选

注册或检索全局部分。定义可以是模板字符串、以 开头的 querySelector、#DOM 元素（其 innerHTML 将用作模板字符串）或 DocumentFragment。

例子

HTML

1

2

3

< div id = "demo" >

{{>头像}}

</ div >

JavaScript

1

2

3

4

5

6

7

8

Vue.partial( '头像' , '<img v-attr="src:avatarURL">' )

新的Vue({

el: '#demo' ,

data: {

avatarURL: '/images/avatar.jpg'

}

})

将导致：

1

2

3

< div id = "demo" > < img src = "/images/avatar.jpg" > </ div >

Vue.nextTick( 回调 )

打回来 Function

Vue.js 批处理视图更新并异步执行它们。如果可用，它会使用MutationObserver并回退到setTimeout(fn, 0). 此方法在下一次视图更新后调用回调，这在您想等到视图更新时很有用。

Vue.use(插件，[args…])

插件 Object或Function

参数... 可选

挂载一个 Vue.js 插件。如果插件是一个对象，它必须有一个install方法。如果它本身是一个函数，它将被视为安装方法。install 方法将使用 Vue 作为参数调用。有关更多详细信息，请参阅插件。

Vue.util

公开内部util模块，其中包含许多实用方法。这适用于高级插件/指令创作，因此您需要查看源代码以查看可用的内容。

# 第6章 指令

反应式指令

这些指令可以将自己绑定到 Vue 实例上的属性，或者绑定到在实例上下文中计算的表达式。当基础属性或表达式的值发生变化时，update()这些指令的函数将在下一次交易时异步调用。

文本

更新元素的textContent.

在内部，{{ Mustache }} 插值也被编译为v-texttextNode 上的指令。

v-html

更新元素的innerHTML.

使用v-html用户提供的数据可能很危险。建议您仅v-html在绝对确定数据源的安全性时使用，或通过自定义过滤器过滤不受信任的 HTML。

秀

该指令可以触发转换。

根据绑定值的真实性，将元素的显示设置为none或其原始值。

v级

该指令接受一个可选参数。

如果没有提供参数，它会将绑定的值添加到元素的 classList 中，并随着值的变化更新类。

如果提供了指令参数，则该参数将是要根据绑定值的真实性进行切换的类。结合多个子句，这可能非常有用：

1

2

3

4

5

< span v-class = "

红色：hasError，

粗体：isImportant，

隐藏：isHidden

" > </ span >

v-属性

该指令需要一个参数。

更新元素的给定属性（由参数指示）。

例子：

1

< canvas v-attr = "width:w, height:h" > </ canvas >

在内部，属性内的 {{ Mustache }} 插值被编译为计算v-attr指令。

在元素上设置属性时，您应该使用v-attr而不是 mustache 绑定。您的模板在被 Vue.js 编译之前由浏览器解析，因此当浏览器尝试获取它作为图像的 URL 时，小胡子绑定将导致 404。src<img>

v型

该指令接受一个可选参数。

将内联 CSS 样式应用于元素。

当没有参数时，绑定值可以是字符串或对象。

如果它是一个字符串，它将被设置为元素的style.cssText.

如果它是一个对象，则每个键/值对都将设置在元素的style对象上。

例子：

1

< div v-style = "myStyles" > </ div >

1

2

3

4

5

6

7

8

9

// myStyles 可以是字符串：

"color:red; font-weight:bold;"

// or an Object:

{

color: 'red' , // camelCase 和 dash-case 都有效 fontWeight: 'bold' , 'font-size' : '2em' }

当有参数时，它将被用作要应用的 CSS 属性。结合多个子句，您可以一起设置多个属性：

例子：

1

2

3

4

5

< div v-style = "

top: top + 'px',

left: left + 'px',

background-color: 'rgb(0,0,' + bg + ')'

" > </ div >

v-style检测任何所需的浏览器供应商前缀也很聪明，因此您可以只使用无前缀版本：

1

2

<!-- 如果需要，将使用-webkit-transform，例如 -->

< div v-style = "transform: 'scale(' + scale + ')'" > </ div >

建议在属性中使用v-style而不是 mustache 绑定，style因为无论版本如何，Internet Explorer 在解析 HTML 时都会删除无效的内联样式。

开启

该指令需要一个参数。

该指令要求值是函数或语句。

将事件侦听器附加到元素。事件类型由参数表示。它也是唯一可以与key过滤器一起使用的指令。有关更多详细信息，请参阅监听事件。

v型

该指令只能用于<input>、<select>或<textarea>元素。

指令参数：lazy, number, options,debounce

在表单输入元素上创建双向绑定。input默认情况下，每个事件都会同步数据。有关详细示例，请参阅处理表单。

v-if

该指令可以触发转换。

根据绑定值的真实性有条件地插入/删除元素。如果元素是一个<template>元素，它的内容将被提取为条件块。

例子：

1

2

3

4

< template v-if = "test" > < p >你好</ p > < p >世界</ p > </ template >

将渲染：

1

2

3

4

<!--v-if-start-->

< p >你好</ p >

< p >世界</ p >

<!--v-if-end-->

v-重复

该指令创建子 Vue 实例。

该指令要求值是数组、对象或数字。

该指令可以触发转换。

该指令接受一个可选参数。

指令参数：track-by

为绑定数组或对象中的每个项目创建一个子 ViewModel。如果该值是一个整数，那么就会创建那么多子 ViewModel。这些子 ViewModel 将在改变方法时自动创建/销毁，例如push()，在 Array 或 Object 上调用，或者增加或减少数量。

当没有提供参数时，子 ViewModel 将直接使用 Array 中分配的元素作为其$data. 如果该值不是对象，则将创建一个包装数据对象，并使用别名 key 在该对象上设置该值$value。

例子：

1

2

3

4

5

< ul > < li v-repeat = "用户" > {{name}} {{email}} </ li > </ ul >

如果提供了参数，将始终使用参数字符串作为别名键创建包装数据对象。这允许在模板中进行更明确的属性访问：

1

2

3

4

5

< ul > < li v-repeat = "user:users" > {{user.name}} {{user.email}} </ li > </ ul >

有关详细示例，请参阅显示列表。

v-with

该指令只能与v-component.

该指令仅接受键路径，不接受表达式。

允许子 ViewModel 从父级继承数据。您可以传入将用作data选项的对象，也可以使用不同的键将单个父属性绑定到子属性。该指令必须与 结合使用v-component。

继承对象的示例：

1

2

3

4

5

6

7

// 父数据看起来像这样

{

user: {

name: 'Foo Bar' ,

email: 'foo@bar.com'

}

}

1

2

3

4

< my-component v-with = "user" > <!-- 不用`user.`也可以访问属性 --> {{name}} {{email}} </ my-component >

继承单个属性的示例（使用相同的数据）：

1

<my-component v-with="myName: user.name, myEmail: user.email">

<!-- 您可以使用新键访问属性 -->

{{myName}} {{myEmail}}

</我的组件>

v-事件

该指令只能与v-component.

该指令接受方法名称或单个表达式语句。

允许父实例监听子实例上的事件。不同之处v-on在于v-events监听 Vue 的组件系统事件是通过vm.$emit()而不是 DOM 事件创建的。该指令允许更多分离的父子通信，而无需将事件侦听器硬编码到父组件中。请注意，它只能与 一起使用v-component，即在子组件的根元素上。

例子：

1

2

<!-- 父模板内 -->

< div v-component = "child" v-events = "change: onChildChange" > </ div >

当子组件调用this.$emit('change', ...)时，将调用父组件的onChildChange方法并传递给 的附加参数$emit()。

文字指令

文字指令将其属性值视为纯字符串；他们不会试图将自己束缚在任何东西上。他们所做的只是bind()使用字符串值执行一次函数。文字指令在其值中接受 mustache 表达式，但这些表达式只会在首次编译时评估一次，并且不会对数据更改做出反应。

v-分量

指令参数：keep-alive, wait-for, transition-mode,inline-template

可以与胡须反应

将此元素编译为具有已注册组件构造函数的子 ViewModel。这可以用于v-with从父级获取数据。有关更多详细信息，请参阅组件系统。

v-偏

可以与胡须反应

将元素的 innerHTML 替换为已注册的部分。部分可以Vue.partial()在选项中注册或传递partials。

在内部使用 mustache 标签v-partial使其具有反应性：

1

2

<!-- 内容会根据vm.partialId改变 -->

< div v-partial = "{{partialId}}" > </ div >

您还可以使用此语法（不支持反应性）：

1

< div > {{> 我的部分}} </ div >

v-过渡

可以与胡须反应

通知 Vue.js 将转换应用到此元素。当某些转换触发指令修改元素或调用 Vue 实例的 DOM 操作方法时，将应用转换类。

有关详细信息，请参阅过渡指南。

参考

在其父组件上注册对子组件的引用以便于访问。仅在与v-componentor结合使用时才受到尊重v-repeat。组件实例可以在其父$对象上访问。例如，请参阅子参考。

与 一起使用时v-repeat，该值将是一个 Array，其中包含与它们绑定的 Array 对应的所有子 Vue 实例。

v-el

在其所有者 Vue 实例上注册对 DOM 元素的引用，以便于访问。eg<div v-el="hi">将作为vm.$$.hi.

空指令

空指令不需要并且将忽略它们的属性值。

v-pre

跳过此元素及其所有子元素的编译。跳过大量没有指令的节点可以加快编译速度。

v-斗篷

此属性保留在元素上，直到关联的 ViewModel 完成编译。结合诸如 之类的 CSS 规则[v-cloak] { display: none }，该指令可用于隐藏未编译的 mustache 绑定，直到 ViewModel 准备好。

# 第7章 过滤器

大写

'abc' => 'abc'

大写

'abc' => 'ABC'

小写

'ABC' => 'abc'

货币

此过滤器采用一个可选参数

12345 => 12,345.00 美元

您可以传递一个可选参数，该参数将用作货币符号（默认为 $）。

复数

这个过滤器至少需要一个参数

根据过滤后的值对参数进行复数。当只有一个 arg 时，复数形式只需在末尾添加一个“s”。当有多个参数时，参数将用作字符串数组，对应于要复数的单词的单、双、三……形式。当要复数的数字超过 args 的长度时，它将使用数组中的最后一个条目。

例子：

1

{{count}} {{count | 复数项目}}

1 => '1 项'

2 => '2 项'

1

{{日期}}{{日期 | 复数st nd rd th}}

将导致：

1 => '1st'

2 => '2nd'

3 => '3rd'

4 => '4th'

5 => '5th'

json

此过滤器采用一个可选参数

JSON.stringify() 传入值而不是输出字符串表示（即[object Object]）。它还接受一个可选参数，即缩进级别（默认为 2）：

1

<前> {{$数据| json 4}} </前>

钥匙

此过滤器仅适用于v-on

这个过滤器只接受一个参数

包装处理程序，以便仅在 keyCode 与参数匹配时调用它。您还可以对一些常用键使用字符串别名：

进入

标签

删除

Esc键

向上

下

剩下

正确的

例子：

1

<输入 v-on = "keyup:doSomething | 键输入" >

doSomething只会在按下 Enter 键时调用。

过滤

语法： filterBy searchKey [in dataKey] .

此过滤器仅适用于v-repeat

这是一个计算过滤器

v-repeat只显示源数组的过滤版本。searchKey参数是上下文 ViewModel 上的属性键。该属性的值将用作要搜索的字符串：

1

2

3

4

< input v-model = "searchText" >

< ul > < li v-repeat = "users | filterBy searchText" > {{name}} </ li > </ ul >

应用过滤器时，它将users通过递归搜索 Array 中searchText每个项目的当前值来过滤 Array。例如，如果一个项目是{ name: 'Jack', phone: '555-123-4567' }并且searchText具有 value '555'，则该项目将被视为匹配项。

或者，您可以使用可选参数缩小要搜索的特定属性in dataKey：

1

2

3

4

< input v-model = "searchText" >

< ul > < li v-repeat = "users | filterBy searchText in name" > {{name}} </ li > </ ul >

searchText现在，只有在其name属性中找到的值时，该项目才会匹配。所以searchTextwith value'555'将不再匹配这个项目，而是'Jack'会。

最后，您可以使用引号来表示文字参数：

1

2

3

< ul > < li v-repeat = "users | filterBy '555' in 'phone'" > {{name}} </ li > </ ul >

排序依据

语法： orderBy sortKey [reverseKey] .

此过滤器仅适用于v-repeat

这是一个计算过滤器

排序v-repeat的显示结果。sortKey参数是上下文 ViewModel 上的属性键。该属性的值将用作对 Array 项进行排序的键。可选reverseKey参数也是上下文 ViewModel 上的属性键，但值的真实性将决定结果是否应该反转。

1

2

3

< ul > < li v-repeat = "users | orderBy field reverse" > {{name}} </ li > </ ul >

1

2

3

4

5

6

7

new Vue({ /\* ... \*/ data: { field: 'name' , reverse: false } })

您还可以使用引号作为文字排序键。要表示文字反转，请使用-1：

1

2

3

< ul > < li v-repeat = "users|orderBy'name'-1" > {{name}} </ li > </ ul >