# Vue安装

|  |
| --- |
|  |

# Vue基础

1. 资料

书籍

* 1. 新手向：Vue 2.0 的建议学习顺序 - 知乎专栏
  2. VueJs2.0建议学习路线 - lavyun - 博客园

视频

1. 慕课网-vue.js入门基础-[**fishenal**](http://www.imooc.com/u/494742/courses?sort=publish) （基础）
2. 学习路线
   1. 基础文档
   2. ES6 module语法
   3. Webpack

## 与自定义元素的关系

你可能已经注意到 Vue.js 组件非常类似于自定义元素——它是 Web 组件规范的一部分。实际上 Vue.js 的组件语法参考了该规范。例如 Vue 组件实现了 Slot API 与 is 特性。但是，有几个关键的不同：

Web 组件规范仍然远未完成，并且没有浏览器实现。相比之下，Vue.js 组件不需要任何补丁，并且在所有支持的浏览器（IE9 及更高版本）之下表现一致。必要时，Vue.js 组件也可以放在原生自定义元素之内。

Vue.js 组件提供了原生自定义元素所不具备的一些重要功能，比如组件间的数据流，自定义事件系统，以及动态的、带特效的组件替换。

## vue实例

每个 Vue 实例都会**代理**其 data 对象里所有的属性：

注意只有这些被代理的属性是**响应的**。如果在实例创建之后添加新的属性到实例上，它不会触发视图更新。我们将在后面详细讨论响应系统。

除了 data 属性， 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 data 属性区分。例如：

### 2. 1 模板语法

Vue.js 使用了基于 HTML 的模版语法，允许开发者声明式地将 DOM 绑定至底层 Vue 实例的数据。

* 1. 插值

Mustache” 语法（双大括号）的文本插值

[**v-once 指令**](http://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

### 2.2 数组

1. 变异方法

* + push()
  + pop()
  + shift()
  + unshift()
  + splice()
  + sort()
  + reverse()

1. 重塑数组
2. 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/)

})

1. 注意事项

由于 JavaScript 的限制， Vue 不能检测以下变动的数组：

* 1. 当你直接设置一个项的索引时，例如： vm.items[indexOfItem] = newValue 解决办法

|  |
| --- |
| // Vue.set  Vue.set(example1.items, indexOfItem, newValue) |

|  |
| --- |
| // Array.prototype.splice`  example1.items.splice(indexOfItem, 1, newValue) |

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

解决办法：

example1.items.splice(newLength)

### 2.3 事件

1. 监听事件： v-on:click="counter += 1"
2. 方法事件处理器：v-on:click="greet"
3. 内联处理器方法：v-on:click="say('hi')"
   1. 内联语句处理器中访问原生 DOM 事件。可以用特殊变量 $event 把它传入方法
4. 事件修饰符 ： 通过由点(.)表示的指令后缀来调用修饰符
   1. .stop 阻止冒泡
   2. .prevent 阻止默认
   3. .capture 使用事件捕获模式
   4. .self 只当事件在该元素本身（而不是子元素）触发时触发回调
5. 按键修饰符

.13 监测常见的键值

Vue 为最常用的按键提供了别名

* + 1. enter
    2. tab
    3. delete (捕获 “删除” 和 “退格” 键)
    4. esc
    5. space
    6. up
    7. down
    8. left
    9. right

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

### 2.4 表单控件绑定

你可以用 v-model 指令在表单控件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。尽管有些神奇，但 v-model 本质上不过是语法糖，它负责监听用户的输入事件以更新数据，并特别处理一些极端的例子。

v-model 并不关心表单控件初始化所生成的值。因为它会选择 Vue 实例数据来作为具体的值。

1. 绑定 value

对于单选按钮，勾选框及选择列表选项， v-model 绑定的 value 通常是静态字符串（对于勾选框是逻辑值）：

<!-- 当选中时，`picked` 为字符串 "a" -->

<input type="radio" v-model="picked" value="a">

<!-- `toggle` 为 true 或 false -->

<input type="checkbox" v-model="toggle">

<!-- 当选中时，`selected` 为字符串 "abc" -->

<select v-model="selected">

<option value="abc">ABC</option>

</select>

但是有时我们想绑定 value 到 Vue 实例的一个动态属性上，这时可以用 v-bind 实现，并且这个属性的值可以不是字符串

2. 修饰符

**.lazy** 在默认情况下， v-model 在 input 事件中同步输入框的值与数据，但你可以添加一个修饰符 lazy ，从而转变为在 change 事件中同步：

**.number** 自动将用户的输入值转为 Number 类型（如果原值的转换结果为 NaN 则返回原值）

**.trim** 如果要自动过滤用户输入的首尾空格，可以添加 trim 修饰符到 v-model 上过滤输入：

<input v-model.trim="msg">

## 4. 组件

### 4.1 定义：

组件（Component）是 Vue.js 最强大的功能之一。组件可以扩展 HTML 元素，封装可重用的代码。在较高层面上，组件是自定义元素， Vue.js 的编译器为它添加特殊功能。在有些情况下，组件也可以是原生 HTML 元素的形式，以 is 特性扩展。

在自定义组件里，你可以像任何普通元素一样用 v-for ，然而他不能自动传递数据到组件里，因为组件有自己独立的作用域。为了传递迭代数据到组件里，我们要用 props

### 全局注册

创建一个Vue实例

1. 注册一个全局组件 Vue.component(tagName, options)
2. 使用 <my-component></my-component>

|  |
| --- |
| <div id="example">  <my-component></my-component>  </div> |

|  |
| --- |
| // 注册  Vue.component('my-component', {  template: '<div>A custom component!</div>'  })  // 创建根实例  new Vue({  el: '#example'  }) |

### 局部注册

不必在全局注册每个组件。通过使用组件实例选项注册，可以使组件仅在另一个实例/组件的作用域中可用：

var Child = {

template: '<div>A custom component!</div>'

}

new Vue({

// ...

components: {

// <my-component> 将只在父模板可用

'my-component': Child

}

})

### DOM模板解析说明

<ul> ， <ol>， <table> ， <select> 限制了能被它包裹的元素，<option> 只能出现在其它元素内部。自定义组件可能无效，因此在渲染的时候会导致错误。变通的方案是使用特殊的 is 属性：

<table>

<tr is="my-row"></tr>

</table>

应当注意，如果您使用来自以下来源之一的字符串模板，这些限制将不适用：

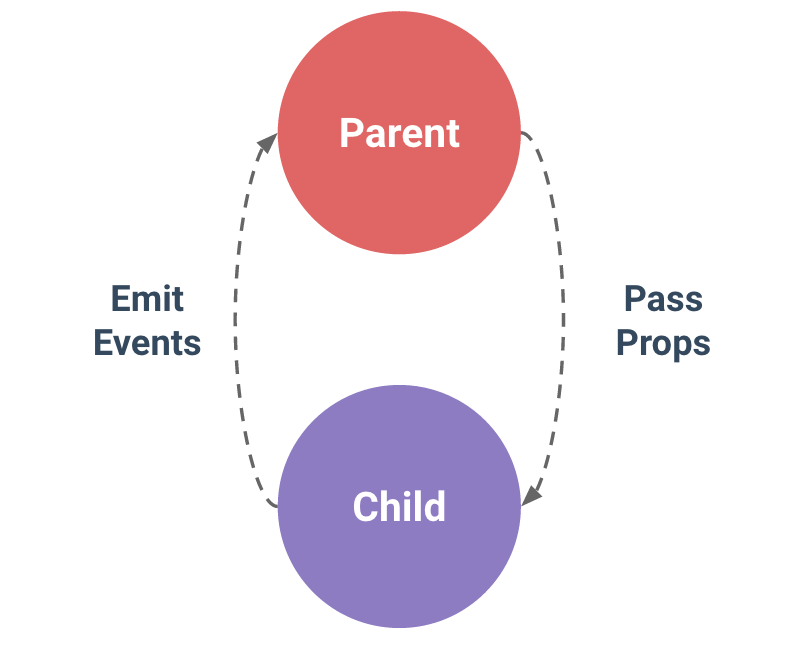
* + <script type="text/x-template">
  + JavaScript内联模版字符串
  + .vue 组件

### ****data**** 必须是函数

**为了使不同引用该组件的父组件数据可以独立保存**

### ****构成组件-**** Props

在 Vue.js 中，父子组件的关系可以总结为 **props down, events up** 。父组件通过 **props** 向下传递数据给子组件，子组件通过 **events** 给父组件发送消息。看看它们是怎么工作的。



1. Prop传递数据

prop 是父组件用来传递数据的一个自定义属性。子组件需要显式地用 [**props 选项**](http://cn.vuejs.org/v2/api/#props) 声明 “prop”：

|  |
| --- |
| Vue.component('child', {  // camelCase in JavaScript  props: ['myMessage'],  template: '<span>{{ myMessage }}</span>'  }) |

|  |
| --- |
| <!-- kebab-case in HTML | 使用的时候向myMessage传递参数-->  <child my-message="hello!"></child> |

1. 书写格式转化

HTML 特性不区分大小写。当使用非字符串模版时，prop的名字形式会从 camelCase 转为 kebab-case（短横线隔开）：、

<child **my-message**="hello!"></child>

1. 动态 Props

类似于用 v-bind 绑定 HTML 特性到一个表达式，也可以用 v-bind 动态绑定 props 的值到父组件的数据中。每当父组件的数据变化时，该变化也会传导给子组件：

<child v-bind:my-message="parentMsg"></child>

1. 字面量语法 vs 动态语法

<!-- 传递了一个字符串"1" -->

<comp some-prop="1"></comp>

<!-- 传递实际的数字 -->

<comp v-bind:some-prop="1"></comp>

### Prop 验证

组件可以为 props 指定验证要求。如果未指定验证要求，Vue 会发出警告。当组件给其他人使用时这很有用

Vue.component('example', {

props: {

// 基础类型检测 （`null` 意思是任何类型都可以）

propA: Number,

// 多种类型

propB: [String, Number],

// 必传且是字符串

propC: {

type: String,

required: true

},

// 数字，有默认值

propD: {

type: Number,

default: 100

},

// 数组／对象的默认值应当由一个工厂函数返回

propE: {

type: Object,

default: function () {

return { message: 'hello' }

}

},

// 自定义验证函数

propF: {

validator: function (value) {

return value > 10

}

}

}

})

type 可以是下面原生构造器：

* + String
  + Number
  + Boolean
  + Function
  + Object
  + Array

type 也可以是一个自定义构造器，使用 instanceof 检测。

当 prop 验证失败了， Vue 将拒绝在子组件上设置此值，如果使用的是开发版本会抛出一条警告。

### 子组件要把数据传递回去: 那就是自定义事件！

1. 使用 **v-on** 绑定自定义事件

每个 Vue 实例都实现了[**事件接口(Events interface)**](http://cn.vuejs.org/v2/api/#Instance-Methods-Events)，即：

* + 使用 $on(eventName) 监听事件
  + 使用 $emit(eventName) 触发事件

2. 另外，父组件可以在使用子组件的地方直接用 v-on 来监听子组件触发的事件

3. 给组件绑定原生事件

### 4.9 使用自定义事件的表单输入组件

<input v-model="something"> 仅仅是一个语法糖：

可以用以下方式实现

<input v-bind:value="something" v-on:input="something = $event.target.value">

简写为:

<input v-bind:value="something" v-on:input="something = arguments[0]">

### 4.10 非父子组件通信

|  |
| --- |
| var bus = new Vue() |

|  |
| --- |
| // 触发组件 A 中的事件  bus.$emit('id-selected', 1) |

|  |
| --- |
| // 在组件 B 创建的钩子中监听事件  bus.$on('id-selected', function (id) {  // ...  }) |

在更多复杂的情况下，你应该考虑使用专门的 [**状态管理模式**](http://cn.vuejs.org/v2/guide/state-management.html).

1. 编译作用域

父组件模板的内容在父组件作用域内编译；子组件模板的内容在子组件作用域内编译。

1. 单个 Slot

最初在 <slot> 标签中的任何内容都被视为**备用内容**。备用内容在子组件的作用域内编译，并且只有在宿主元素为空，且没有要插入的内容时才显示备用内容。

<slot> 如果没有分发内容则显示我。 </slot>

1. 具名Slots

<slot> 元素可以用一个特殊的属性 name 来配置如何分发内容。多个 slot 可以有不同的名字。具名 slot 将匹配内容片段中有对应 slot 特性的元素。

仍然可以有一个匿名 slot ，它是**默认 slot** ，作为找不到匹配的内容片段的备用插槽。如果没有默认的 slot ，这些找不到匹配的内容片段将被抛弃。

1. 动态组件

多个组件可以使用同一个挂载点，然后动态地在它们之间切换。使用保留的 <component>元素，动态地绑定到它的 is 特性：

1. **keep-alive**

如果把切换出去的组件保留在内存中，可以保留它的状态或避免重新渲染。为此可以添加一个 keep-alive 指令参数：

### 4.11 杂项

Vue 组件的 API 来自三部分 - props, events 和 slots ：

* + **Props** 允许外部环境传递数据给组件
  + **Events** 允许组件触发外部环境的副作用
  + **Slots** 允许外部环境将额外的内容组合在组件中。

### 4.12 子组件索引

尽管有 props 和 events ，但是有时仍然需要在 JavaScript 中直接访问子组件。为此可以使用 ref 为子组件指定一个索引 ID 。

### 4.13 异步组件

在大型应用中，我们可能需要将应用拆分为多个小模块，按需从服务器下载。为了让事情更简单， Vue.js 允许将组件定义为一个工厂函数，动态地解析组件的定义。Vue.js 只在组件需要渲染时触发工厂函数，并且把结果缓存起来，用于后面的再次渲染。例如：

Vue.component('async-example', function (resolve, reject) {

setTimeout(function () {

resolve({

template: '<div>I am async!</div>'

})

}, 1000)

})

工厂函数接收一个 resolve 回调，在收到从服务器下载的组件定义时调用。也可以调用reject(reason) 指示加载失败。这里 setTimeout 只是为了演示。怎么获取组件完全由你决定。

推荐配合使用 ：[**Webpack 的代码分割功能**](http://webpack.github.io/docs/code-splitting.html)：

|  |
| --- |
| Vue.component('async-webpack-example', function (resolve) {  // 这个特殊的 require 语法告诉 webpack  // 自动将编译后的代码分割成不同的块，  // 这些块将通过 Ajax 请求自动下载。  require(['./my-async-component'], resolve)  }) |

### 4.14 递归组件

组件在它的模板内可以递归地调用自己，不过，只有当它有 name 选项时才可以：

### 4.15 内联模版

如果子组件有 inline-template 特性，组件将把它的内容当作它的模板，而不是把它当作分发内容。这让模板更灵活

### 4.16 X-Templates

另一种定义模版的方式是在 JavaScript 标签里使用 text/x-template 类型，并且指定一个id

### 4.17使用v-once 将渲染结果缓存起来

使用 **v-once** 的低级静态组件(Cheap Static Component)

尽管在 Vue 中渲染 HTML 很快，不过当组件中包含**大量**静态内容时，可以考虑使用v-once 将渲染结果缓存起来，就像这样：

Vue.component('terms-of-service', {

template: '\

<div v-once>\

<h1>Terms of Service</h1>\

... a lot of static content ...\

</div>\

'

})

# Vue进阶

深入响应式原理

* 1. Vue执行DOM更新是异步的
  2. 如果想在Dom更新后做点什么
     1. 可以再数据变化之后立即执行Vue.nextTick(callback);这样回调在DOM更新之后就会调用

过渡效果

* 1. 在 CSS 过渡和动画中自动应用 class
  2. 可以配合使用第三方 CSS 动画库，如 Animate.css
  3. 在过渡钩子函数中使用 JavaScript 直接操作 DOM
  4. 可以配合使用第三方 JavaScript 动画库，如 Velocity.js
  5. 过渡的-CSS-类名
     1. v-enter: 定义进入过渡的开始状态。在元素被插入时生效，在下一个帧移除。
     2. v-enter-active: 定义进入过渡的结束状态。在元素被插入时生效，在transition/animation 完成之后移除。
     3. v-leave: 定义离开过渡的开始状态。在离开过渡被触发时生效，在下一个帧移除。
     4. v-leave-active: 定义离开过渡的结束状态。在离开过渡被触发时生效，在transition/animation 完成之后移除。

<name="my-transition>" 可以重置前缀

* 1. CSS 动画

CSS 动画用法同 CSS 过渡，区别是在动画中 v-enter 类名在节点插 入 DOM 后不会立即删除，而是在 animationend 事件触发时删除

* 1. 自定义过渡类名
     1. enter-class
     2. enter-active-class
     3. leave-class
     4. leave-active-class

他们的优先级高于普通的类名，这对于 Vue 的过渡系统和其他第三方 CSS 动画库，如[**Animate.css**](https://daneden.github.io/animate.css/) 结合使用十分有用

* 1. 同时使用 Transitions 和 Animations

但是，在一些场景中，你需要给同一个元素同时设置两种过渡动效，比如 animation 很快的被触发并完成了，而 transition 效果还没结束。在这种情况中，你就需要使用 type特性并设置 animation 或 transition 来明确声明你需要 Vue 监听的类型。

* 1. JavaScript 钩子

<transition

v-on:before-enter="beforeEnter"

v-on:enter="enter"

v-on:after-enter="afterEnter"

v-on:enter-cancelled="enterCancelled"

v-on:before-leave="beforeLeave"

v-on:leave="leave"

v-on:after-leave="afterLeave"

v-on:leave-cancelled="leaveCancelled"

>

<!-- ... -->

</transition>

当只用 JavaScript 过渡的时候， **在 enter 和 leave 中，回调函数 done 是必须的**。 否则，它们会被同步调用，过渡会立即完成。

推荐对于仅使用 JavaScript 过渡的元素添加 v-bind:css="false"，Vue 会跳过 CSS 的检测。这也可以避免过渡过程中 CSS 的影响。

* 1. 初始渲染的过渡

可以通过 appear 特性设置节点的在初始渲染的过渡

这里默认和进入和离开过渡一样，同样也可以自定义 CSS 类名。

自定义 JavaScript 钩子。

* 1. 过渡模式

<transition> 的默认行为 - 进入和离开同时发生

**过渡模式：**

* in-out: 新元素先进行过渡，完成之后当前元素过渡离开。
* out-in: 当前元素先进行过渡，完成之后新元素过渡进入。

<transition name="fade" mode="out-in">

* 1. 多个组件的过渡

多个组件的过渡很简单很多 - 我们不需要使用 key 特性。相反，我们只需要使用[**动态组件**](http://cn.vuejs.org/v2/guide/components.html#动态组件)

* 1. 列表过渡---<transition-group>
     1. <transition-group> 组件
* 不同于 <transition>， 它会以一个真实元素呈现：默认为一个 <span>。你也可以通过 tag 特性更换为其他元素。
* 元素 **一定需要** 指定唯一的 key 特性值

**v-move 特性**

# 其他—定义：

1. [**MVVM 模式**](https://en.wikipedia.org/wiki/Model_View_ViewModel)
2. 数据、模板、挂载元素、方法、生命周期钩子
3. Vue.js 是否有“控制器”的概念？答案是，没有
4. 模板语法
5. Computed、Data、Computed、
6. $watch: 它用于观察 Vue 实例上的数据变动。当一些数据需要根据其它数据变化时
7. \_.debounce
8. axios
9. ref ： 为子组件指定一个索引 ID
10. [**状态管理模式**](http://cn.vuejs.org/v2/guide/state-management.html).
11. [**Web组件规范草案**](https://github.com/w3c/webcomponents/blob/gh-pages/proposals/Slots-Proposal.md)
12. 编译作用域
13. currentView
14. Browserify 是不支持异步加载的
15. [**vue-devtools**](https://github.com/vuejs/vue-devtools) 来获取更加友好的检查接口
16. [**Webpack 的代码分割功能**](http://webpack.github.io/docs/code-splitting.html)

Vue.component('async-webpack-example', function (resolve) {

// 这个特殊的 require 语法告诉 webpack

// 自动将编译后的代码分割成不同的块，

// 这些块将通过 Ajax 请求自动下载。

require(['./my-async-component'], resolve)

})

1. [**生命周期图示**](http://cn.vuejs.org/v2/guide/instance.html#Lifecycle-Diagram)
2. **Vue-source： 服务器请求库**
3. **.**

# 其他—指令：

1. Mustache” 语法（双大括号）；双大括号会将数据解释为纯文本，而非 HTML 。
2. [**v-once 指令**](http://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。
3. 为了输出真正的 HTML ，你需要使用v-html 指令：
4. v-model：它使得在表单输入和应用状态中做双向数据绑定变得非常轻巧
5. v-if
6. v-else
7. v-show v-show 不支持 <template> 语法。
8. template（嵌套标签）
9. is : 在有些情况下，组件也可以是原生 HTML 元素的形式，以 is 特性扩展。
10. v-bind
11. v-on 监听 DOM 事件来触发一些 JavaScript 代码
12. Vue.set(example1.items, indexOfItem, newValue)
13. v-bind:true-value=”var”
14. v-bind:false-value=”var”
15. v-bind:value=”var”
16. v-once 将渲染结果缓存起来

# ES6-model 模板语法

1. 静态加载，不能使用表达式等加载后才赋值的东西
2. 模块整体加载：\*
   1. import \* as circle from './circle';

模块整体加载所在的那个对象（上例circle）不允许运行时改变

1. export default 命令
   1. 输出
      1. 一个模块只能有一个默认输出
      2. export default命令其实只是输出一个叫做default的变量
      3. 后面不能跟变量声明语句。

export default function () {

console.log('foo');

}

export default function foo() {

console.log('foo');

}

* 1. import接收匿名函数
     1. import customName from './export-default';

注意：customName不能加{}

* + 1. 一条import语句中，同时输入默认方法和其他变量
    2. import \_, { each } from 'lodash';

# API

1. Data
   1. vm.$data 访问原始数据对象
   2. \_ 或 $ 开头的属性 不会 被 Vue 实例代理，用例如 vm.$data.\_property 的方式访问这些属性
2. propsData
   1. 创建实例时传递 props。主要作用是方便测试。
3. watch

## BUG

|  |
| --- |
| <select v-model="selected">  <option v-for="option in options" v-bind:value="option.value">  {{ option.text }}  </option>  </select>  <span>Selected: {{ selected }}</span> |

|  |
| --- |
| new Vue({  el: '...',  data: {  selected: 'A',  options: [  { text: 'One', value: 'A' },  { text: 'Two', value: 'B' },  { text: 'Three', value: 'C' }  ]  }  }) |

报错???但是可以运行

[Vue warn]: <select multiple v-model="selected"> expects an Array value for its binding, but got String

(found in <Root>)