官网：<https://cn.vuejs.org/v2/api/>

**[全局配置](https://cn.vuejs.org/v2/api/" \l "全局配置)**

Vue.config 是一个对象，包含 Vue 的全局配置。可以在启动应用之前修改下列属性：

**silent**

* **类型**：boolean
* **默认值**：false
* **用法**：

|  |
| --- |
| Vue.config.silent = true |

* 取消 Vue 所有的日志与警告。
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22silent%22&type=Code)

**optionMergeStrategies**

* **类型**：{ [key: string]: Function }
* **默认值**：{}
* **用法**：

|  |
| --- |
| Vue.config.optionMergeStrategies.\_my\_option = function (parent, child, vm) {  return child + 1  }  const Profile = Vue.extend({  \_my\_option: 1  })  // Profile.options.\_my\_option = 2 |

* 自定义合并策略的选项。
* 合并策略选项分别接受第一个参数作为父实例，第二个参数为子实例，Vue 实例上下文被作为第三个参数传入。
* **参考** [**自定义选项的混合策略**](https://cn.vuejs.org/v2/guide/mixins.html#自定义选项混合策略)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22optionMergeStrategies%22&type=Code)

**devtools**

* **类型**：boolean
* **默认值**：true (生产版为 false)
* **用法**：

|  |
| --- |
| // 务必在加载 Vue 之后，立即同步设置以下内容  Vue.config.devtools = true |

* 配置是否允许 [**vue-devtools**](https://github.com/vuejs/vue-devtools) 检查代码。开发版本默认为 true，生产版本默认为 false。生产版本设为 true 可以启用检查。
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22devtools%22&type=Code)

**errorHandler**

* **类型**：Function
* **默认值**：undefined
* **用法**：

|  |
| --- |
| Vue.config.errorHandler = function (err, vm, info) {  // handle error  // `info` 是 Vue 特定的错误信息，比如错误所在的生命周期钩子  // 只在 2.2.0+ 可用  } |

* 指定组件的渲染和观察期间未捕获错误的处理函数。这个处理函数被调用时，可获取错误信息和 Vue 实例。
* **从 2.2.0 起，这个钩子也会捕获组件生命周期钩子里的错误。同样的，当这个钩子是 undefined 时，被捕获的错误会通过 console.error 输出而避免应用崩溃。**
* **从 2.4.0 起这个钩子也会捕获 Vue 自定义事件句柄内部的错误了。**
* [**Sentry**](https://sentry.io/)**，一个错误追踪服务，通过此选项提供**[**官方集成**](https://sentry.io/for/vue/)**。**
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22errorHandler%22&type=Code)

**warnHandler**

**2.4.0 新增**

* **类型**：Function
* **默认值**：undefined
* **用法**：

|  |
| --- |
| Vue.config.warnHandler = function (msg, vm, trace) {  // `trace` 是组件的继承关系追踪  } |

* 为 Vue 的运行时警告赋于一个自定义句柄。注意这只会在开发者环境下生效，在生产环境下它会被忽略。

**ignoredElements**

* **类型**：Array<string>
* **默认值**：[]
* **用法**：

|  |
| --- |
| Vue.config.ignoredElements = [  'my-custom-web-component', 'another-web-component'  ] |

* 须使 Vue 忽略在 Vue 之外的自定义元素 (e.g. 使用了 Web Components APIs)。否则，它会假设你忘记注册全局组件或者拼错了组件名称，从而抛出一个关于 Unknown custom element 的警告。
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22ignoredElements%22&type=Code)

**keyCodes**

* **类型**：{ [key: string]: number | Array<number> }
* **默认值**：{}
* **用法**：

|  |
| --- |
| Vue.config.keyCodes = {  v: 86,  f1: 112,  // camelCase 不可用  mediaPlayPause: 179,  // 取而代之的是 kebab-case 且用双引号括起来  "media-play-pause": 179,  up: [38, 87]  } |

|  |
| --- |
| <input type="text" @keyup.media-play-pause="method"> |

* 给 v-on 自定义键位别名。
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22keyCodes%22&type=Code)

**performance**

**2.2.0 新增**

* **类型**：boolean
* **默认值**：false (自 2.2.3 起)
* **用法**：

设置为 true 以在浏览器开发工具中启用对组件初始化、编译、渲染和打补丁的性能追踪。只适用于开发模式和支持 [**performance.mark**](https://developer.mozilla.org/en-US/docs/Web/API/Performance/mark) API 的浏览器上。

**productionTip**

**2.2.0 新增**

* **类型**：boolean
* **默认值**：true
* **用法**：

设置为 false 以阻止 vue 在启动时生成生产提示。

**全局 API**

**[Vue.extend( options )](https://cn.vuejs.org/v2/api/" \l "Vue-extend)**

* **参数**：
  + {Object} options
* **用法**：

使用基础 Vue 构造器，创建一个“子类”。参数是一个包含组件选项的对象。

data 选项是特例，需要注意 - 在 Vue.extend() 中它必须是函数

|  |
| --- |
| <div id="mount-point"></div> |

|  |
| --- |
| // 创建构造器  var Profile = Vue.extend({  template: '<p>{{firstName}} {{lastName}} aka {{alias}}</p>',  data: function () {  return {  firstName: 'Walter',  lastName: 'White',  alias: 'Heisenberg'  }  }  })  // 创建 Profile 实例，并挂载到一个元素上。  new Profile().$mount('#mount-point') |

结果如下：

|  |
| --- |
| <p>Walter White aka Heisenberg</p> |

* **参考**：[**组件**](https://cn.vuejs.org/v2/guide/components.html)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.extend%22%20OR%20%22extend%22&type=Code)

**[Vue.nextTick( [callback, context] )](https://cn.vuejs.org/v2/api/" \l "Vue-nextTick)**

* **参数**：
  + {Function} [callback]
  + {Object} [context]
* **用法**：

在下次 DOM 更新循环结束之后执行延迟回调。在修改数据之后立即使用这个方法，获取更新后的 DOM。

|  |
| --- |
| // 修改数据  vm.msg = 'Hello'  // DOM 还没有更新  Vue.nextTick(function () {  // DOM 更新了  }) |

**2.1.0 起新增：如果没有提供回调且在支持 Promise 的环境中，则返回一个 Promise。请注意 Vue 不自带 Promise 的 polyfill，所以如果你的目标浏览器不原生支持 Promise (IE：你们都看我干嘛)，你得自行 polyfill。**

* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.nextTick%22%20OR%20%22nextTick%22&type=Code)

**[Vue.set( target, key, value )](https://cn.vuejs.org/v2/api/" \l "Vue-set)**

* **参数**：
  + {Object | Array} target
  + {string | number} key
  + {any} value
* **返回值**：设置的值。
* **用法**：

设置对象的属性。如果对象是响应式的，确保属性被创建后也是响应式的，同时触发视图更新。这个方法主要用于避开 Vue 不能检测属性被添加的限制。

注意对象不能是 Vue 实例，或者 Vue 实例的根数据对象。

* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.set%22%20OR%20%22set%22&type=Code)

**[Vue.delete( target, key )](https://cn.vuejs.org/v2/api/" \l "Vue-delete)**

* **参数**：
  + {Object | Array} target
  + {string | number} key/index

**仅在 2.2.0+ 版本中支持 Array + index 用法。**

* **用法**：

删除对象的属性。如果对象是响应式的，确保删除能触发更新视图。这个方法主要用于避开 Vue 不能检测到属性被删除的限制，但是你应该很少会使用它。

**在 2.2.0+ 中同样支持在数组上工作。**

目标对象不能是一个 Vue 示例或 Vue 示例的根数据对象。

* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.delete%22%20OR%20%22delete%22&type=Code)

**[Vue.directive( id, [definition] )](https://cn.vuejs.org/v2/api/" \l "Vue-directive)**

* **参数**：
  + {string} id
  + {Function | Object} [definition]
* **用法**：

注册或获取全局指令。

|  |
| --- |
| // 注册  Vue.directive('my-directive', {  bind: function () {},  inserted: function () {},  update: function () {},  componentUpdated: function () {},  unbind: function () {}  })  // 注册 (指令函数)  Vue.directive('my-directive', function () {  // 这里将会被 `bind` 和 `update` 调用  })  // getter，返回已注册的指令  var myDirective = Vue.directive('my-directive') |

* **参考**：[**自定义指令**](https://cn.vuejs.org/v2/guide/custom-directive.html)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.directive%22%20OR%20%22directive%22&type=Code)

**[Vue.filter( id, [definition] )](https://cn.vuejs.org/v2/api/" \l "Vue-filter)**

* **参数**：
  + {string} id
  + {Function} [definition]
* **用法**：

注册或获取全局过滤器。

|  |
| --- |
| // 注册  Vue.filter('my-filter', function (value) {  // 返回处理后的值  })  // getter，返回已注册的过滤器  var myFilter = Vue.filter('my-filter') |

* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.filter%22%20OR%20%22filter%22&type=Code)

**[Vue.component( id, [definition] )](https://cn.vuejs.org/v2/api/" \l "Vue-component)**

* **参数**：
  + {string} id
  + {Function | Object} [definition]
* **用法**：

注册或获取全局组件。注册还会自动使用给定的id设置组件的名称

|  |
| --- |
| // 注册组件，传入一个扩展过的构造器  Vue.component('my-component', Vue.extend({ /\* ... \*/ }))  // 注册组件，传入一个选项对象 (自动调用 Vue.extend)  Vue.component('my-component', { /\* ... \*/ })  // 获取注册的组件 (始终返回构造器)  var MyComponent = Vue.component('my-component') |

* **参考**：[**组件**](https://cn.vuejs.org/v2/guide/components.html)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.component%22%20OR%20%22component%22&type=Code)

**[Vue.use( plugin )](https://cn.vuejs.org/v2/api/" \l "Vue-use)**

* **参数**：
  + {Object | Function} plugin
* **用法**：

安装 Vue.js 插件。如果插件是一个对象，必须提供 install 方法。如果插件是一个函数，它会被作为 install 方法。install 方法将被作为 Vue 的参数调用。

当 install 方法被同一个插件多次调用，插件将只会被安装一次。

* **参考**：[**插件**](https://cn.vuejs.org/v2/guide/plugins.html)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.use%22%20OR%20%22use%22&type=Code)

**[Vue.mixin( mixin )](https://cn.vuejs.org/v2/api/" \l "Vue-mixin)**

* **参数**：
  + {Object} mixin
* **用法**：

全局注册一个混合，影响注册之后所有创建的每个 Vue 实例。插件作者可以使用混合，向组件注入自定义的行为。**不推荐在应用代码中使用**。

* **参考**：[**全局混合**](https://cn.vuejs.org/v2/guide/mixins.html#全局混合)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.mixin%22%20OR%20%22mixin%22&type=Code)

**[Vue.compile( template )](https://cn.vuejs.org/v2/api/" \l "Vue-compile)**

* **参数**：
  + {string} template
* **用法**：

在 render 函数中编译模板字符串。**只在独立构建时有效**

|  |
| --- |
| var res = Vue.compile('<div><span>{{ msg }}</span></div>')  new Vue({  data: {  msg: 'hello'  },  render: res.render,  staticRenderFns: res.staticRenderFns  }) |

* **参考**：[**渲染函数**](https://cn.vuejs.org/v2/guide/render-function.html)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.compile%22%20OR%20%22compile%22&type=Code)

**[Vue.version](https://cn.vuejs.org/v2/api/" \l "Vue-version)**

* **细节**：提供字符串形式的 Vue 安装版本号。这对社区的插件和组件来说非常有用，你可以根据不同的版本号采取不同的策略。
* **用法**：

|  |
| --- |
| var version = Number(Vue.version.split('.')[0])  if (version === 2) {  // Vue v2.x.x  } else if (version === 1) {  // Vue v1.x.x  } else {  // Unsupported versions of Vue  } |

* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.version%22%20OR%20%22version%22&type=Code)

**选项 / 数据**

**data**

* **类型**：Object | Function
* **限制**：组件的定义只接受 function。
* **详细**：

Vue 实例的数据对象。Vue 将会递归将 data 的属性转换为 getter/setter，从而让 data 的属性能够响应数据变化。**对象必须是纯粹的对象 (含有零个或多个的 key/value 对)**：浏览器 API 创建的原生对象，原型上的属性会被忽略。大概来说，data 应该只能是数据 - 不推荐观察拥有状态行为的对象。

一旦观察过，不需要再次在数据对象上添加响应式属性。因此推荐在创建实例之前，就声明所有的根级响应式属性。

实例创建之后，可以通过 vm.$data 访问原始数据对象。Vue 实例也代理了 data 对象上所有的属性，因此访问 vm.a 等价于访问 vm.$data.a。

以 \_ 或 $ 开头的属性 **不会** 被 Vue 实例代理，因为它们可能和 Vue 内置的属性、API 方法冲突。你可以使用例如 vm.$data.\_property 的方式访问这些属性。

当一个**组件**被定义，data 必须声明为返回一个初始数据对象的函数，因为组件可能被用来创建多个实例。如果 data 仍然是一个纯粹的对象，则所有的实例将**共享引用**同一个数据对象！通过提供 data 函数，每次创建一个新实例后，我们能够调用 data 函数，从而返回初始数据的一个全新副本数据对象。

如果需要，可以通过将 vm.$data 传入 JSON.parse(JSON.stringify(...)) 得到深拷贝的原始数据对象。

* **示例**：

|  |
| --- |
| var data = { a: 1 }  // 直接创建一个实例  var vm = new Vue({  data: data  })  vm.a // => 1  vm.$data === data // => true  // Vue.extend() 中 data 必须是函数  var Component = Vue.extend({  data: function () {  return { a: 1 }  }  }) |

* 注意，**不应该对 data 属性使用箭头函数** (例如data: () => { return { a: this.myProp }})。理由是箭头函数绑定了父级作用域的上下文，所以 this 将不会按照期望指向 Vue 实例，this.myProp 将是 undefined。
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22data%22&type=Code)

**props**

* **类型**：Array<string> | Object
* **详细**：

props 可以是数组或对象，用于接收来自父组件的数据。props 可以是简单的数组，或者使用对象作为替代，对象允许配置高级选项，如类型检测、自定义校验和设置默认值。

* **示例**：

|  |
| --- |
| // 简单语法  Vue.component('props-demo-simple', {  props: ['size', 'myMessage']  })  // 对象语法，提供校验  Vue.component('props-demo-advanced', {  props: {  // 检测类型  height: Number,  // 检测类型 + 其他验证  age: {  type: Number,  default: 0,  required: true,  validator: function (value) {  return value >= 0  }  }  }  }) |

* **参考**：[**Props**](https://cn.vuejs.org/v2/guide/components.html#Props)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22props%22&type=Code)

**propsData**

* **类型**：{ [key: string]: any }
* **限制**：只用于 new 创建的实例中。
* **详细**：

创建实例时传递 props。主要作用是方便测试。

* **示例**：

|  |
| --- |
| var Comp = Vue.extend({  props: ['msg'],  template: '<div>{{ msg }}</div>'  })  var vm = new Comp({  propsData: {  msg: 'hello'  }  }) |

* **参考**：[**Props**](https://cn.vuejs.org/v2/guide/components.html#Props)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22propsData%22&type=Code)

**computed**

* **类型**：{ [key: string]: Function | { get: Function, set: Function } }
* **详细**：

计算属性将被混入到 Vue 实例中。所有 getter 和 setter 的 this 上下文自动地绑定为 Vue 实例。

注意，**不应该使用箭头函数来定义计算属性函数** (例如 aDouble: () => this.a \* 2)。理由是箭头函数绑定了父级作用域的上下文，所以 this 将不会按照期望指向 Vue 实例，this.a 将是 undefined。

计算属性的结果会被缓存，除非依赖的响应式属性变化才会重新计算。注意，如果实例范畴之外的依赖 (比如非响应式的 not reactive) 是**不会**触发计算属性更新的。

* **示例**：

|  |
| --- |
| var vm = new Vue({  data: { a: 1 },  computed: {  // 仅读取  aDouble: function () {  return this.a \* 2  },  // 读取和设置  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 |

* **参考**：[**计算属性**](https://cn.vuejs.org/v2/guide/computed.html)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22computed%22&type=Code)

**methods**

* **类型**：{ [key: string]: Function }
* **详细**：

methods 将被混入到 Vue 实例中。可以直接通过 VM 实例访问这些方法，或者在指令表达式中使用。方法中的 this 自动绑定为 Vue 实例。

注意，**不应该使用箭头函数来定义 method 函数** (例如 plus: () => this.a++)。理由是箭头函数绑定了父级作用域的上下文，所以 this 将不会按照期望指向 Vue 实例，this.a 将是 undefined。

* **示例**：

|  |
| --- |
| var vm = new Vue({  data: { a: 1 },  methods: {  plus: function () {  this.a++  }  }  })  vm.plus()  vm.a // 2 |

* **参考**：[**事件处理器**](https://cn.vuejs.org/v2/guide/events.html)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22methods%22&type=Code)

**watch**

* **类型**：{ [key: string]: string | Function | Object }
* **详细**：

一个对象，键是需要观察的表达式，值是对应回调函数。值也可以是方法名，或者包含选项的对象。Vue 实例将会在实例化时调用 $watch()，遍历 watch 对象的每一个属性。

* **示例**：

|  |
| --- |
| var vm = new Vue({  data: {  a: 1,  b: 2,  c: 3  },  watch: {  a: function (val, oldVal) {  console.log('new: %s, old: %s', val, oldVal)  },  // 方法名  b: 'someMethod',  // 深度 watcher  c: {  handler: function (val, oldVal) { /\* ... \*/ },  deep: true  }  }  })  vm.a = 2 // => new: 2, old: 1 |

* 注意，**不应该使用箭头函数来定义 watcher 函数** (例如 searchQuery: newValue => this.updateAutocomplete(newValue))。理由是箭头函数绑定了父级作用域的上下文，所以 this 将不会按照期望指向 Vue 实例，this.updateAutocomplete 将是 undefined。
* **参考**：[**实例方法 / 数据 - vm.$watch**](https://cn.vuejs.org/v2/api/#vm-watch)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22watch%22&type=Code)

**选项 / DOM**

**el**

* **类型**：string | HTMLElement
* **限制**：只在由 new 创建的实例中遵守。
* **详细**：

提供一个在页面上已存在的 DOM 元素作为 Vue 实例的挂载目标。可以是 CSS 选择器，也可以是一个 HTMLElement 实例。

在实例挂载之后，元素可以用 vm.$el 访问。

如果这个选项在实例化时有作用，实例将立即进入编译过程，否则，需要显式调用 vm.$mount() 手动开启编译。

提供的元素只能作为挂载点。不同于 Vue 1.x，所有的挂载元素会被 Vue 生成的 DOM 替换。因此不推荐挂载 root 实例到 <html> 或者 <body> 上。

如果 render 函数和 template 属性都不存在，挂载 DOM 元素的 HTML 会被提取出来用作模板，此时，必须使用 Runtime + Compiler 构建的 Vue 库。

* **参考**：
  + [**生命周期图示**](https://cn.vuejs.org/v2/guide/instance.html#生命周期图示)
  + [**运行时 + 编译器 vs. 只包含运行时**](https://cn.vuejs.org/v2/guide/installation.html#运行时-编译器-vs-只包含运行时)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22el%22&type=Code)

**template**

* **类型**：string
* **详细**：

一个字符串模板作为 Vue 实例的标识使用。模板将会 **替换** 挂载的元素。挂载元素的内容都将被忽略，除非模板的内容有分发插槽。

如果值以 # 开始，则它用作选项符，将使用匹配元素的 innerHTML 作为模板。常用的技巧是用 <script type="x-template"> 包含模板。

出于安全考虑，您应该只使用您信任的 Vue 模板。避免使用其他人生成的内容作为您的模板。

如果 Vue 选项中包含渲染函数，该模板将被忽略。

* **参考**：
  + [**生命周期图示**](https://cn.vuejs.org/v2/guide/instance.html#生命周期图示)
  + [**用插槽分发内容**](https://cn.vuejs.org/v2/guide/components.html#使用插槽分发内容)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22template%22&type=Code)

**render**

* **类型**：(createElement: () => VNode) => VNode
* **详细**：

字符串模板的代替方案，允许你发挥 JavaScript 最大的编程能力。该渲染函数接收一个 createElement 方法作为第一个参数用来创建 VNode。

如果组件是一个函数组件，渲染函数还会接收一个额外的 context 参数，为没有实例的函数组件提供上下文信息。

Vue 选项中的 render 函数若存在，则 Vue 构造函数不会从 template 选项或通过 el 选项指定的挂载元素中提取出的 HTML 模板编译渲染函数。

* **参考**：[**渲染函数**](https://cn.vuejs.org/v2/guide/render-function.html)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22render%22&type=Code)

**renderError**

**2.2.0 新增**

* **类型**：(createElement: () => VNode, error: Error) => VNode
* **详细**：

**只在开发者环境下工作。**

当 render 函数遭遇错误时，提供另外一种渲染输出。其错误将会作为第二个参数传递到 renderError。这个功能配合 hot-reload 非常实用。

* **示例**：

|  |
| --- |
| new Vue({  render (h) {  throw new Error('oops')  },  renderError (h, err) {  return h('pre', { style: { color: 'red' }}, err.stack)  }  }).$mount('#app') |

* **参考**：[**渲染函数**](https://cn.vuejs.org/v2/guide/render-function.html)

**选项 / 生命周期钩子**

所有的生命周期钩子自动绑定 this 上下文到实例中，因此你可以访问数据，对属性和方法进行运算。这意味着 **你不能使用箭头函数来定义一个生命周期方法** (例如 created: () => this.fetchTodos())。这是因为箭头函数绑定了父上下文，因此 this 与你期待的 Vue 实例不同，this.fetchTodos 的行为未定义。

**beforeCreate**

* **类型**：Function
* **详细**：

在实例初始化之后，数据观测 (data observer) 和 event/watcher 事件配置之前被调用。

* **参考**：[**生命周期图示**](https://cn.vuejs.org/v2/guide/instance.html#生命周期图示)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22beforeCreate%22&type=Code)

**created**

* **类型**：Function
* **详细**：

在实例创建完成后被立即调用。在这一步，实例已完成以下的配置：数据观测 (data observer)，属性和方法的运算，watch/event 事件回调。然而，挂载阶段还没开始，$el 属性目前不可见。

* **参考**：[**生命周期图示**](https://cn.vuejs.org/v2/guide/instance.html#生命周期图示)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22created%22&type=Code)

**beforeMount**

* **类型**：Function
* **详细**：

在挂载开始之前被调用：相关的 render 函数首次被调用。

**该钩子在服务器端渲染期间不被调用。**

* **参考**：[**生命周期图示**](https://cn.vuejs.org/v2/guide/instance.html#生命周期图示)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22beforeMount%22&type=Code)

**mounted**

* **类型**：Function
* **详细**：

el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用该钩子。如果 root 实例挂载了一个文档内元素，当 mounted 被调用时 vm.$el 也在文档内。

注意 mounted **不会**承诺所有的子组件也都一起被挂载。如果你希望等到整个视图都渲染完毕，可以用 [**vm.$nextTick**](https://cn.vuejs.org/v2/api/#vm-nextTick) 替换掉 mounted：

|  |
| --- |
| mounted: function () {  this.$nextTick(function () {  // Code that will run only after the  // entire view has been rendered  })  } |

**该钩子在服务器端渲染期间不被调用。**

* **参考**：[**生命周期图示**](https://cn.vuejs.org/v2/guide/instance.html#生命周期图示)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22mounted%22&type=Code)

**beforeUpdate**

* **类型**：Function
* **详细**：

数据更新时调用，发生在虚拟 DOM 重新渲染和打补丁之前。

你可以在这个钩子中进一步地更改状态，这不会触发附加的重渲染过程。

**该钩子在服务器端渲染期间不被调用。**

* **参考**：[**生命周期图示**](https://cn.vuejs.org/v2/guide/instance.html#生命周期图示)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22beforeUpdate%22&type=Code)

**updated**

* **类型**：Function
* **详细**：

由于数据更改导致的虚拟 DOM 重新渲染和打补丁，在这之后会调用该钩子。

当这个钩子被调用时，组件 DOM 已经更新，所以你现在可以执行依赖于 DOM 的操作。然而在大多数情况下，你应该避免在此期间更改状态。如果要相应状态改变，通常最好使用[**计算属性**](https://cn.vuejs.org/v2/api/#computed)或 [**watcher**](https://cn.vuejs.org/v2/api/#watch) 取而代之。

注意 updated **不会**承诺所有的子组件也都一起被重绘。如果你希望等到整个视图都重绘完毕，可以用 [**vm.$nextTick**](https://cn.vuejs.org/v2/api/#vm-nextTick) 替换掉 updated：

|  |
| --- |
| updated: function () {  this.$nextTick(function () {  // Code that will run only after the  // entire view has been re-rendered  })  } |

**该钩子在服务器端渲染期间不被调用。**

* **参考**：[**生命周期图示**](https://cn.vuejs.org/v2/guide/instance.html#生命周期图示)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22updated%22&type=Code)

**activated**

* **类型**：Function
* **详细**：

keep-alive 组件激活时调用。

**该钩子在服务器端渲染期间不被调用。**

* **参考**：
  + [**构建组件 - keep-alive**](https://cn.vuejs.org/v2/api/#keep-alive)
  + [**动态组件 - keep-alive**](https://cn.vuejs.org/v2/guide/components.html#keep-alive)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22activated%22&type=Code)

**deactivated**

* **类型**：Function
* **详细**：

keep-alive 组件停用时调用。

**该钩子在服务器端渲染期间不被调用。**

* **参考**：
  + [**构建组件 - keep-alive**](https://cn.vuejs.org/v2/api/#keep-alive)
  + [**动态组件 - keep-alive**](https://cn.vuejs.org/v2/guide/components.html#keep-alive)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22deactivated%22&type=Code)

**beforeDestroy**

* **类型**：Function
* **详细**：

实例销毁之前调用。在这一步，实例仍然完全可用。

**该钩子在服务器端渲染期间不被调用。**

* **参考**：[**生命周期图示**](https://cn.vuejs.org/v2/guide/instance.html#生命周期图示)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22beforeDestroy%22&type=Code)

**destroyed**

* **类型**：Function
* **详细**：

Vue 实例销毁后调用。调用后，Vue 实例指示的所有东西都会解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁。

**该钩子在服务器端渲染期间不被调用。**

* **参考**：[**生命周期图示**](https://cn.vuejs.org/v2/guide/instance.html#生命周期图示)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22destroyed%22&type=Code)

**选项 / 资源**

**directives**

* **类型**：Object
* **详细**：
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22directives%22&type=Code)

包含 Vue 实例可用指令的哈希表。

* **参考**：[**自定义指令**](https://cn.vuejs.org/v2/guide/custom-directive.html)

**filters**

* **类型**：Object
* **详细**：
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22filters%22&type=Code)

包含 Vue 实例可用过滤器的哈希表。

* **参考**：[**Vue.filter**](https://cn.vuejs.org/v2/api/#Vue-filter)

**components**

* **类型**：Object
* **详细**：
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22components%22&type=Code)

包含 Vue 实例可用组件的哈希表。

* **参考**：[**组件**](https://cn.vuejs.org/v2/guide/components.html)

**选项 / 组合**

**parent**

* **类型**：Vue instance
* **详细**：

指定已创建的实例之父实例，在两者之间建立父子关系。子实例可以用 this.$parent访问父实例，子实例被推入父实例的 $children 数组中。

节制地使用 $parent 和 $children - 它们的主要目的是作为访问组件的应急方法。更推荐用 props 和 events 实现父子组件通信

* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22parent%22&type=Code)

**mixins**

* **类型**：Array<Object>
* **详细**：

mixins 选项接受一个混合对象的数组。这些混合实例对象可以像正常的实例对象一样包含选项，他们将在 Vue.extend() 里最终选择使用相同的选项合并逻辑合并。举例：如果你混合包含一个钩子而创建组件本身也有一个，两个函数将被调用。  
Mixin 钩子按照传入顺序依次调用，并在调用组件自身的钩子之前被调用。

* **示例**：

|  |
| --- |
| var mixin = {  created: function () { console.log(1) }  }  var vm = new Vue({  created: function () { console.log(2) },  mixins: [mixin]  })  // => 1  // => 2 |

* **参考**：[**混合**](https://cn.vuejs.org/v2/guide/mixins.html)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22mixins%22&type=Code)

**extends**

* **类型**：Object | Function
* **详细**：

允许声明扩展另一个组件(可以是一个简单的选项对象或构造函数)，而无需使用 Vue.extend。这主要是为了便于扩展单文件组件。

这和 mixins 类似，区别在于，组件自身的选项会比要扩展的源组件具有更高的优先级。

* **示例**：

|  |
| --- |
| var CompA = { ... }  // 在没有调用 `Vue.extend` 时候继承 CompA  var CompB = {  extends: CompA,  ...  } |

* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22extends%22&type=Code)

**provide / inject**

**2.2.0 新增**

* **类型**：
  + **provide**：Object | () => Object
  + **inject**：Array<string> | { [key: string]: string | Symbol }
* **详细**：

provide 和 inject 主要为高阶插件/组件库提供用例。并不推荐直接用于应用程序代码中。

这对选项需要一起使用，以允许一个祖先组件向其所有子孙后代注入一个依赖，不论组件层次有多深，并在起上下游关系成立的时间里始终生效。如果你熟悉 React，这与 React 的上下文特性很相似。

provide 选项应该是一个对象或返回一个对象的函数。该对象包含可注入其子孙的属性。在该对象中你可以使用 ES2015 Symbols 作为 key，但是只在原生支持 Symbol 和 Reflect.ownKeys 的环境下可工作。

inject 选项应该是一个字符串数组或一个对象，该对象的 key 代表了本地绑定的名称，value 为其 key (字符串或 Symbol) 以在可用的注入中搜索。

**提示：provide 和 inject 绑定并不是可响应的。这是刻意为之的。然而，如果你传入了一个可监听的对象，那么其对象的属性还是可响应的。**

* **示例**：

|  |
| --- |
| var Provider = {  provide: {  foo: 'bar'  },  // ...  }  var Child = {  inject: ['foo'],  created () {  console.log(this.foo) // => "bar"  }  // ...  } |

* 利用 ES2015 Symbols、函数 provide 和对象 inject：

|  |
| --- |
| const s = Symbol()  const Provider = {  provide () {  return {  [s]: 'foo'  }  }  }  const Child = {  inject: { s },  // ...  } |

* **接下来 2 个例子只工作在 Vue 2.2.1 或更高版本。低于这个版本时，注入的值会在 props 和 data 初始化之后得到。**
* 使用一个注入的值作为一个属性的默认值：

|  |
| --- |
| const Child = {  inject: ['foo'],  props: {  bar: {  default () {  return this.foo  }  }  }  } |

* 使用一个注入的值作为数据入口：

|  |
| --- |
| const Child = {  inject: ['foo'],  data () {  return {  bar: this.foo  }  }  } |

**选项 / 其它**

**name**

* **类型**：string
* **限制**：只有作为组件选项时起作用。
* **详细**：

允许组件模板递归地调用自身。注意，组件在全局用 Vue.component() 注册时，全局 ID 自动作为组件的 name。

指定 name 选项的另一个好处是便于调试。有名字的组件有更友好的警告信息。另外，当在有 [**vue-devtools**](https://github.com/vuejs/vue-devtools)，未命名组件将显示成 <AnonymousComponent>，这很没有语义。通过提供 name 选项，可以获得更有语义信息的组件树。

* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22name%22&type=Code)

**delimiters**

* **类型**：Array<string>
* **默认值**：["{{", "}}"]
* **限制**：这个选项只在完整构建版本中的浏览器内编译时可用。
* **详细**：

改变纯文本插入分隔符。

* **示例**：

|  |
| --- |
| new Vue({  delimiters: ['${', '}']  })  // 分隔符变成了 ES6 模板字符串的风格 |

* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22delimiters%22&type=Code)

**functional**

* **类型**：boolean
* **详细**：

使组件无状态 (没有 data ) 和无实例 (没有 this 上下文)。他们用一个简单的 render函数返回虚拟节点使他们更容易渲染。

* **参考**：[**函数式组件**](https://cn.vuejs.org/v2/guide/render-function.html#函数式组件)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22functional%22&type=Code)

**model**

**2.2.0 新增**

* **类型**：{ prop?: string, event?: string }
* **详细**：

允许一个自定义组件在使用 v-model 时定制 prop 和 event。默认情况下，一个组件上的 v-model 会把 value 用作 prop 且把 input 用作 event，但是一些输入类型比如单选框和复选框按钮可能像使用 value prop 来达到不同的目的。使用 model 选项可以回避这些情况产生的冲突。

* **Example**：

|  |
| --- |
| Vue.component('my-checkbox', {  model: {  prop: 'checked',  event: 'change'  },  props: {  // this allows using the `value` prop for a different purpose  value: String,  // use `checked` as the prop which take the place of `value`  checked: {  type: Number,  default: 0  }  },  // ...  }) |

|  |
| --- |
| <my-checkbox v-model="foo" value="some value"></my-checkbox> |

* 上述代码相当于：

|  |
| --- |
| <my-checkbox  :checked="foo"  @change="val => { foo = val }"  value="some value">  </my-checkbox> |

**inheritAttrs**

**2.4.0 新增**

* **类型**：boolean
* **默认值**：true
* **详细**：

默认情况下父作用域的不被认作 props 的特性绑定 (attribute bindings) 将会“回退”且作为普通的 HTML 特性应用在子组件的根元素上。当撰写包裹一个目标元素或另一个组件的组件时，这可能不会总是符合预期行为。通过设置 inheritAttrs 到 false，这些默认行为将会被去掉。而通过 (同样是 2.4 新增的) 实例属性 $attrs 可以让这些特性生效，且可以通过 v-bind 显性的绑定到非根元素上。

注意：这个选项**不影响** class 和 style 绑定。

**comments**

**2.4.0 新增**

* **类型**：boolean
* **默认值**：false
* **限制**：这个选项只在完整构建版本中的浏览器内编译时可用。
* **详细**：

当设为 true 时，将会保留且渲染模板中的 HTML 注释。默认行为是舍弃它们。

**实例属性**

**vm.$data**

* **类型**：Object
* **详细**：

Vue 实例观察的数据对象。Vue 实例代理了对其 data 对象属性的访问。

* **参考**：[**选项 / 数据 - data**](https://cn.vuejs.org/v2/api/#data)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.prototype.%24data%22&type=Code)

**vm.$props**

**2.2.0 新增**

* **类型**：Object
* **详细**：

当前组件接收到的 props 对象。Vue 实例代理了对其 props 对象属性的访问。

**vm.$el**

* **类型**：HTMLElement
* **只读**
* **详细**：

Vue 实例使用的根 DOM 元素。

* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.prototype.%24el%22&type=Code)

**vm.$options**

* **类型**：Object
* **只读**
* **详细**：

用于当前 Vue 实例的初始化选项。需要在选项中包含自定义属性时会有用处：

|  |
| --- |
| new Vue({  customOption: 'foo',  created: function () {  console.log(this.$options.customOption) // => 'foo'  }  }) |

* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.prototype.%24options%22&type=Code)

**vm.$parent**

* **类型**：Vue instance
* **只读**
* **详细**：

父实例，如果当前实例有的话。

* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.prototype.%24parent%22&type=Code)

**vm.$root**

* **类型**：Vue instance
* **只读**
* **详细**：

当前组件树的根 Vue 实例。如果当前实例没有父实例，此实例将会是其自己。

* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.prototype.%24root%22&type=Code)

**vm.$children**

* **类型**：Array<Vue instance>
* **只读**
* **详细**：

当前实例的直接子组件。**需要注意 $children 并不保证顺序，也不是响应式的。**如果你发现自己正在尝试使用 $children 来进行数据绑定，考虑使用一个数组配合 v-for来生成子组件，并且使用 Array 作为真正的来源。

* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.prototype.%24children%22&type=Code)

**vm.$slots**

* **类型**：{ [name: string]: ?Array<VNode> }
* **只读**
* **详细**：

用来访问被[**插槽分发**](https://cn.vuejs.org/v2/guide/components.html#使用插槽分发内容)的内容。每个[**具名插槽**](https://cn.vuejs.org/v2/guide/components.html#具名插槽) 有其相应的属性 (例如：slot="foo" 中的内容将会在 vm.$slots.foo 中被找到)。default 属性包括了所有没有被包含在具名插槽中的节点。

在使用[**渲染函数**](https://cn.vuejs.org/v2/guide/render-function.html)书写一个组件时，访问 vm.$slots 最有帮助。

* **示例**：

|  |
| --- |
| <blog-post>  <h1 slot="header">  About Me  </h1>  <p>Here's some page content, which will be included in vm.$slots.default, because it's not inside a named slot.</p>  <p slot="footer">  Copyright 2016 Evan You  </p>  <p>If I have some content down here, it will also be included in vm.$slots.default.</p>.  </blog-post> |

|  |
| --- |
| Vue.component('blog-post', {  render: function (createElement) {  var header = this.$slots.header  var body = this.$slots.default  var footer = this.$slots.footer  return createElement('div', [  createElement('header', header),  createElement('main', body),  createElement('footer', footer)  ])  }  }) |

* **参考**：
  + [**<slot> 组件**](https://cn.vuejs.org/v2/api/#slot-1)
  + [**使用插槽分发内容**](https://cn.vuejs.org/v2/guide/components.html#使用插槽分发内容)
  + [**渲染函数 - 插槽**](https://cn.vuejs.org/v2/guide/render-function.html#插槽)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.prototype.%24slots%22&type=Code)

**vm.$scopedSlots**

**2.1.0 新增**

* **类型**：{ [name: string]: props => VNode | Array<VNode> }
* **只读**
* **详细**：

用来访问[**作用域插槽**](https://cn.vuejs.org/v2/guide/components.html#作用域插槽)。对于包括 默认 slot 在内的每一个插槽，该对象都包含一个返回相应 VNode 的函数。

vm.$scopedSlots 在使用[**渲染函数**](https://cn.vuejs.org/v2/guide/render-function.html)开发一个组件时特别有用。

* **参考**：
  + [**<slot> 组件**](https://cn.vuejs.org/v2/api/#slot-1)
  + [**作用域插槽**](https://cn.vuejs.org/v2/guide/components.html#作用域插槽)
  + [**渲染函数 - 插槽**](https://cn.vuejs.org/v2/guide/render-function.html#插槽)

**vm.$refs**

* **类型**：Object
* **只读**
* **详细**：

一个对象，持有已注册过 ref 的所有子组件。

* **参考**：
  + [**子组件引用**](https://cn.vuejs.org/v2/guide/components.html#子组件索引)
  + [**特殊特性 - ref**](https://cn.vuejs.org/v2/api/#ref)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.prototype.%24refs%22&type=Code)

**vm.$isServer**

* **类型**：boolean
* **只读**
* **详细**：

当前 Vue 实例是否运行于服务器。

* **参考**：[**服务端渲染**](https://cn.vuejs.org/v2/guide/ssr.html)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.prototype.%24isServer%22&type=Code)

**vm.$attrs**

* **类型**：{ [key: string]: string }
* **只读**
* **详细**：

包含了父作用域中不被认为 (且不预期为) props 的特性绑定 (class 和 style 除外)。当一个组件没有声明任何 props 时，这里会包含所有父作用域的绑定 (class 和 style除外)，并且可以通过 v-bind="$attrs" 传入内部组件——在创建更高层次的组件时非常有用。

* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.prototype.%24attrs%22&type=Code)

**vm.$listeners**

* **类型**：{ [key: string]: Function | Array<Function> }
* **只读**
* **详细**：

包含了父作用域中的 (不含 .native 修饰器的) v-on 事件监听器。它可以通过 v-on="$listeners" 传入内部组件——在创建更高层次的组件时非常有用。

* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.prototype.%24listeners%22&type=Code)

**实例方法 / 数据**

**[vm.$watch( expOrFn, callback, [options] )](https://cn.vuejs.org/v2/api/" \l "vm-watch)**

* **参数**：
  + {string | Function} expOrFn
  + {Function | Object} callback
  + {Object} [options]
    - {boolean} deep
    - {boolean} immediate
* **返回值**：{Function} unwatch
* **用法**：

观察 Vue 实例变化的一个表达式或计算属性函数。回调函数得到的参数为新值和旧值。表达式只接受监督的键路径。对于更复杂的表达式，用一个函数取代。

注意：在变异 (不是替换) 对象或数组时，旧值将与新值相同，因为它们的引用指向同一个对象/数组。Vue 不会保留变异之前值的副本。

* **示例**：

|  |
| --- |
| // 键路径  vm.$watch('a.b.c', function (newVal, oldVal) {  // 做点什么  })  // 函数  vm.$watch(  function () {  return this.a + this.b  },  function (newVal, oldVal) {  // 做点什么  }  ) |

* vm.$watch 返回一个取消观察函数，用来停止触发回调：

|  |
| --- |
| var unwatch = vm.$watch('a', cb)  // 之后取消观察  unwatch() |

* **选项：deep**

为了发现对象内部值的变化，可以在选项参数中指定 deep: true 。注意监听数组的变动不需要这么做。

|  |
| --- |
| vm.$watch('someObject', callback, {  deep: true  })  vm.someObject.nestedValue = 123  // callback is fired |

* **选项：immediate**

在选项参数中指定 immediate: true 将立即以表达式的当前值触发回调：

|  |
| --- |
| vm.$watch('a', callback, {  immediate: true  })  // 立即以 `a` 的当前值触发回调 |

* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.prototype.%24watch%22&type=Code)

**[vm.$set( target, key, value )](https://cn.vuejs.org/v2/api/" \l "vm-set)**

* **参数**：
  + {Object | Array} target
  + {string | number} key
  + {any} value
* **返回值**：设置的值。
* **用法**：

这是全局 Vue.set 的**别名**。

* **参考**：[**Vue.set**](https://cn.vuejs.org/v2/api/#Vue-set)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.prototype.%24set%22&type=Code)

**[vm.$delete( target, key )](https://cn.vuejs.org/v2/api/" \l "vm-delete)**

* **参数**：
  + {Object | Array} target
  + {string | number} key
* **用法**：

这是全局 Vue.delete 的**别名**。

* **参考**：[**Vue.delete**](https://cn.vuejs.org/v2/api/#Vue-delete)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.prototype.%24delete%22&type=Code)

**实例方法 / 事件**

**[vm.$on( event, callback )](https://cn.vuejs.org/v2/api/" \l "vm-on)**

* **参数**：
  + {string | Array<string>} event (数组只在 2.2.0+ 中支持)
  + {Function} callback
* **用法**：

监听当前实例上的自定义事件。事件可以由vm.$emit触发。回调函数会接收所有传入事件触发函数的额外参数。

* **示例**：

|  |
| --- |
| vm.$on('test', function (msg) {  console.log(msg)  })  vm.$emit('test', 'hi')  // => "hi" |

* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.prototype.%24on%22&type=Code)

**[vm.$once( event, callback )](https://cn.vuejs.org/v2/api/" \l "vm-once)**

* **参数**：
  + {string} event
  + {Function} callback
* **用法**：

监听一个自定义事件，但是只触发一次，在第一次触发之后移除监听器。

* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.prototype.%24once%22&type=Code)

**[vm.$off( [event, callback] )](https://cn.vuejs.org/v2/api/" \l "vm-off)**

* **参数**：
  + {string | Array<string>} event (只在 2.2.2+ 支持数组)
  + {Function} [callback]
* **用法**：

移除自定义事件监听器。

* + 如果没有提供参数，则移除所有的事件监听器；
  + 如果只提供了事件，则移除该事件所有的监听器；
  + 如果同时提供了事件与回调，则只移除这个回调的监听器。
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.prototype.%24off%22&type=Code)

**[vm.$emit( event, […args] )](https://cn.vuejs.org/v2/api/" \l "vm-emit)**

* **参数**：
  + {string} event
  + [...args]

触发当前实例上的事件。附加参数都会传给监听器回调。

* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.prototype.%24emit%22&type=Code)

**实例方法 / 生命周期**

**[vm.$mount( [elementOrSelector] )](https://cn.vuejs.org/v2/api/" \l "vm-mount)**

* **参数**：
  + {Element | string} [elementOrSelector]
  + {boolean} [hydrating]
* **返回值**：vm - 实例自身
* **用法**：

如果 Vue 实例在实例化时没有收到 el 选项，则它处于“未挂载”状态，没有关联的 DOM 元素。可以使用 vm.$mount() 手动地挂载一个未挂载的实例。

如果没有提供 elementOrSelector 参数，模板将被渲染为文档之外的的元素，并且你必须使用原生 DOM API 把它插入文档中。

这个方法返回实例自身，因而可以链式调用其它实例方法。

* **示例**：

|  |
| --- |
| var MyComponent = Vue.extend({  template: '<div>Hello!</div>'  })  // 创建并挂载到 #app (会替换 #app)  new MyComponent().$mount('#app')  // 同上  new MyComponent({ el: '#app' })  // 或者，在文档之外渲染并且随后挂载  var component = new MyComponent().$mount()  document.getElementById('app').appendChild(component.$el) |

* **参考**：
  + [**生命周期图示**](https://cn.vuejs.org/v2/guide/instance.html#生命周期图示)
  + [**服务端渲染**](https://cn.vuejs.org/v2/guide/ssr.html)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.prototype.%24mount%22&type=Code)

**[vm.$forceUpdate()](https://cn.vuejs.org/v2/api/" \l "vm-forceUpdate)**

* **示例**：

迫使 Vue 实例重新渲染。注意它仅仅影响实例本身和插入插槽内容的子组件，而不是所有子组件。

* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.prototype.%24forceUpdate%22&type=Code)

**[vm.$nextTick( [callback] )](https://cn.vuejs.org/v2/api/" \l "vm-nextTick)**

* **参数**：
  + {Function} [callback]
* **用法**：

将回调延迟到下次 DOM 更新循环之后执行。在修改数据之后立即使用它，然后等待 DOM 更新。它跟全局方法 Vue.nextTick 一样，不同的是回调的 this 自动绑定到调用它的实例上。

**2.1.0 起新增：如果没有提供回调且在支持 Promise 的环境中，则返回一个 Promise。请注意 Vue 不自带 Promise 的 polyfill，所以如果你的目标浏览器不是原生支持 Promise (IE：你们都看我干嘛)，你得自行 polyfill。**

* **示例**：

|  |
| --- |
| new Vue({  // ...  methods: {  // ...  example: function () {  // 修改数据  this.message = 'changed'  // DOM 还没有更新  this.$nextTick(function () {  // DOM 现在更新了  // `this` 绑定到当前实例  this.doSomethingElse()  })  }  }  }) |

* **参考**：[**Vue.nextTick**](https://cn.vuejs.org/v2/api/#Vue-nextTick)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.prototype.%24nextTick%22&type=Code)

**[vm.$destroy()](https://cn.vuejs.org/v2/api/" \l "vm-destroy)**

* **用法**：

完全销毁一个实例。清理它与其它实例的连接，解绑它的全部指令及事件监听器。

触发 beforeDestroy 和 destroyed 的钩子。

在大多数场景中你不应该调用这个方法。最好使用 v-if 和 v-for 指令以数据驱动的方式控制子组件的生命周期。

* **参考**：[**生命周期图示**](https://cn.vuejs.org/v2/guide/instance.html#生命周期图示)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22Vue.prototype.%24destroy%22&type=Code)

**指令**

**v-text**

* **预期**：string
* **详细**：

更新元素的 textContent。如果要更新部分的 textContent ，需要使用 {{ Mustache }} 插值。

* **示例**：

|  |
| --- |
| <span v-text="msg"></span>  <!-- 和下面的一样 -->  <span>{{msg}}</span> |

* **参考**：[**数据绑定语法 - 插值**](https://cn.vuejs.org/v2/guide/syntax.html#插值)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22v-text%22&type=Code)

**v-html**

* **预期**：string
* **详细**：

更新元素的 innerHTML 。**注意：内容按普通 HTML 插入 - 不会作为 Vue 模板进行编译**。如果试图使用 v-html 组合模板，可以重新考虑是否通过使用组件来替代。

在网站上动态渲染任意 HTML 是非常危险的，因为容易导致 [**XSS 攻击**](https://en.wikipedia.org/wiki/Cross-site_scripting)。只在可信内容上使用 v-html，**永不**用在用户提交的内容上。

* **示例**：

|  |
| --- |
| <div v-html="html"></div> |

* **参考**：[**数据绑定语法 - 插值**](https://cn.vuejs.org/v2/guide/syntax.html#纯-HTML)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22v-html%22&type=Code)

**v-show**

* **预期**：any
* **用法**：

根据表达式之真假值，切换元素的 display CSS 属性。

当条件变化时该指令触发过渡效果。

当和 v-if 一起使用时，v-for 的优先级比 v-if 更高。详见[**列表渲染教程**](https://cn.vuejs.org/v2/guide/list.html#v-for-with-v-if)

* **参考**：[**条件渲染 - v-show**](https://cn.vuejs.org/v2/guide/conditional.html#v-show)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22v-show%22&type=Code)

**v-if**

* **预期**：any
* **用法**：

根据表达式的值的真假条件渲染元素。在切换时元素及它的数据绑定 / 组件被销毁并重建。如果元素是 <template> ，将提出它的内容作为条件块。

当条件变化时该指令触发过渡效果。

* **参考**：[**条件渲染 - v-if**](https://cn.vuejs.org/v2/guide/conditional.html)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22v-if%22&type=Code)

**v-else**

* **不需要表达式**
* **限制**：前一兄弟元素必须有 v-if 或 v-else-if。
* **用法**：

为 v-if 或者 v-else-if 添加“else 块”。

|  |
| --- |
| <div v-if="Math.random() > 0.5">  Now you see me  </div>  <div v-else>  Now you don't  </div> |

* **参考**：[**条件渲染 - v-else**](https://cn.vuejs.org/v2/guide/conditional.html#v-else)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22v-else%22&type=Code)

**v-else-if**

**2.1.0 新增**

* **类型**：any
* **限制**：前一兄弟元素必须有 v-if 或 v-else-if。
* **用法**：

表示 v-if 的 “else if 块”。可以链式调用。

|  |
| --- |
| <div v-if="type === 'A'">  A  </div>  <div v-else-if="type === 'B'">  B  </div>  <div v-else-if="type === 'C'">  C  </div>  <div v-else>  Not A/B/C  </div> |

* **参考**：[**条件渲染 - v-else-if**](https://cn.vuejs.org/v2/guide/conditional.html#v-else-if)

**v-for**

* **预期**：Array | Object | number | string
* **用法**：

基于源数据多次渲染元素或模板块。此指令之值，必须使用特定语法 alias in expression ，为当前遍历的元素提供别名：

|  |
| --- |
| <div v-for="item in items">  {{ item.text }}  </div> |

另外也可以为数组索引指定别名 (或者用于对象的键)：

|  |
| --- |
| <div v-for="(item, index) in items"></div>  <div v-for="(val, key) in object"></div>  <div v-for="(val, key, index) in object"></div> |

v-for 默认行为试着不改变整体，而是替换元素。迫使其重新排序的元素，您需要提供一个 key 的特殊属性：

|  |
| --- |
| <div v-for="item in items" :key="item.id">  {{ item.text }}  </div> |

v-for 的详细用法可以通过以下链接查看教程详细说明。

* **参考**：
  + [**列表渲染**](https://cn.vuejs.org/v2/guide/list.html)
  + [**key**](https://cn.vuejs.org/v2/guide/list.html#key)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22v-for%22&type=Code)

**v-on**

* **缩写**：@
* **预期**：Function | Inline Statement | Object
* **参数**：event
* **修饰符**：
  + .stop - 调用 event.stopPropagation()。
  + .prevent - 调用 event.preventDefault()。
  + .capture - 添加事件侦听器时使用 capture 模式。
  + .self - 只当事件是从侦听器绑定的元素本身触发时才触发回调。
  + .{keyCode | keyAlias} - 只当事件是从特定键触发时才触发回调。
  + .native - 监听组件根元素的原生事件。
  + .once - 只触发一次回调。
  + .left - (2.2.0) 只当点击鼠标左键时触发。
  + .right - (2.2.0) 只当点击鼠标右键时触发。
  + .middle - (2.2.0) 只当点击鼠标中键时触发。
  + .passive - (2.3.0) 以 { passive: true } 模式添加侦听器
* **用法**：

绑定事件监听器。事件类型由参数指定。表达式可以是一个方法的名字或一个内联语句，如果没有修饰符也可以省略。

从 2.4.0 开始，v-on 同样支持不带参数绑定一个事件/监听器键值对的对象。注意当使用对象语法时，是不支持任何修饰器的。

用在普通元素上时，只能监听 **原生 DOM 事件**。用在自定义元素组件上时，也可以监听子组件触发的**自定义事件**。

在监听原生 DOM 事件时，方法以事件为唯一的参数。如果使用内联语句，语句可以访问一个 $event 属性：v-on:click="handle('ok', $event)"。

* **示例**：

|  |
| --- |
| <!-- 方法处理器 -->  <button v-on:click="doThis"></button>  <!-- 对象语法 (2.4.0+) -->  <button v-on="{ mousedown: doThis, mouseup: doThat }"></button>  <!-- 内联语句 -->  <button v-on:click="doThat('hello', $event)"></button>  <!-- 缩写 -->  <button @click="doThis"></button>  <!-- 停止冒泡 -->  <button @click.stop="doThis"></button>  <!-- 阻止默认行为 -->  <button @click.prevent="doThis"></button>  <!-- 阻止默认行为，没有表达式 -->  <form @submit.prevent></form>  <!-- 串联修饰符 -->  <button @click.stop.prevent="doThis"></button>  <!-- 键修饰符，键别名 -->  <input @keyup.enter="onEnter">  <!-- 键修饰符，键代码 -->  <input @keyup.13="onEnter">  <!-- 点击回调只会触发一次 -->  <button v-on:click.once="doThis"></button> |

* 在子组件上监听自定义事件 (当子组件触发“my-event”时将调用事件处理器)：

|  |
| --- |
| <my-component @my-event="handleThis"></my-component>  <!-- 内联语句 -->  <my-component @my-event="handleThis(123, $event)"></my-component>  <!-- 组件中的原生事件 -->  <my-component @click.native="onClick"></my-component> |

* **参考**：
  + [**事件处理器**](https://cn.vuejs.org/v2/guide/events.html)
  + [**组件 - 自定义事件**](https://cn.vuejs.org/v2/guide/components.html#自定义事件)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22v-on%22&type=Code)

**v-bind**

* **缩写**：:
* **预期**：any (with argument) | Object (without argument)
* **参数**：attrOrProp (optional)
* **修饰符**：
  + .prop - 被用于绑定 DOM 属性 (property)。([**差别在哪里？**](https://stackoverflow.com/questions/6003819/properties-and-attributes-in-html#answer-6004028))
  + .camel - (2.1.0+) 将 kebab-case 特性名转换为 camelCase. (从 2.1.0 开始支持)
  + .sync (2.3.0+) 语法糖，会扩展成一个更新父组件绑定值的 v-on 侦听器。
* **用法**：

动态地绑定一个或多个特性，或一个组件 prop 到表达式。

在绑定 class 或 style 特性时，支持其它类型的值，如数组或对象。可以通过下面的教程链接查看详情。

在绑定 prop 时，prop 必须在子组件中声明。可以用修饰符指定不同的绑定类型。

没有参数时，可以绑定到一个包含键值对的对象。注意此时 class 和 style 绑定不支持数组和对象。

* **示例**：

|  |
| --- |
| <!-- 绑定一个属性 -->  <img v-bind:src="imageSrc">  <!-- 缩写 -->  <img :src="imageSrc">  <!-- 内联字符串拼接 -->  <img :src="'/path/to/images/' + fileName">  <!-- class 绑定 -->  <div :class="{ red: isRed }"></div>  <div :class="[classA, classB]"></div>  <div :class="[classA, { classB: isB, classC: isC }]">  <!-- style 绑定 -->  <div :style="{ fontSize: size + 'px' }"></div>  <div :style="[styleObjectA, styleObjectB]"></div>  <!-- 绑定一个有属性的对象 -->  <div v-bind="{ id: someProp, 'other-attr': otherProp }"></div>  <!-- 通过 prop 修饰符绑定 DOM 属性 -->  <div v-bind:text-content.prop="text"></div>  <!-- prop 绑定。“prop”必须在 my-component 中声明。-->  <my-component :prop="someThing"></my-component>  <!-- 通过 $props 将父组件的 props 一起传给子组件 -->  <child-component v-bind="$props"></child-component>  <!-- XLink -->  <svg><a :xlink:special="foo"></a></svg> |

* .camel 修饰符允许在使用 DOM 模板时将 v-bind 属性名称驼峰化，例如 SVG 的 viewBox 属性：

|  |
| --- |
| <svg :view-box.camel="viewBox"></svg> |

* 在使用字符串模板或通过 vue-loader/vueify 编译时，无需使用 .camel。
* **参考**：
  + [**Class 与 Style 绑定**](https://cn.vuejs.org/v2/guide/class-and-style.html)
  + [**组件 - Props**](https://cn.vuejs.org/v2/guide/components.html#Props)
  + [**组件 - .sync 修饰符**](https://cn.vuejs.org/v2/guide/components.html#sync-修饰符)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22v-bind%22&type=Code)

**v-model**

* **预期**：随表单控件类型不同而不同。
* **限制**：
  + <input>
  + <select>
  + <textarea>
  + components
* **修饰符**：
  + [**.lazy**](https://cn.vuejs.org/v2/guide/forms.html#lazy) - 取代 input 监听 change 事件
  + [**.number**](https://cn.vuejs.org/v2/guide/forms.html#number) - 输入字符串转为数字
  + [**.trim**](https://cn.vuejs.org/v2/guide/forms.html#trim) - 输入首尾空格过滤
* **用法**：

在表单控件或者组件上创建双向绑定。细节请看下面的教程链接。

* **参考**：
  + [**表单控件绑定**](https://cn.vuejs.org/v2/guide/forms.html)
  + [**组件 - 在输入组件上使用自定义事件**](https://cn.vuejs.org/v2/guide/components.html#使用自定义事件的表单输入组件)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22v-model%22&type=Code)

**v-pre**

* **不需要表达式**
* **用法**：

跳过这个元素和它的子元素的编译过程。可以用来显示原始 Mustache 标签。跳过大量没有指令的节点会加快编译。

* **示例**：

|  |
| --- |
| <span v-pre>{{ this will not be compiled }}</span> |

* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22v-pre%22&type=Code)

**v-cloak**

* **不需要表达式**
* **用法**：

这个指令保持在元素上直到关联实例结束编译。和 CSS 规则如 [v-cloak] { display: none } 一起用时，这个指令可以隐藏未编译的 Mustache 标签直到实例准备完毕。

* **示例**：

|  |
| --- |
| [v-cloak] {  display: none;  } |

|  |
| --- |
| <div v-cloak>  {{ message }}  </div> |

* 不会显示，直到编译结束。
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22v-cloak%22&type=Code)

**v-once**

* **不需要表达式**
* **详细**：

只渲染元素和组件**一次**。随后的重新渲染，元素/组件及其所有的子节点将被视为静态内容并跳过。这可以用于优化更新性能。

|  |
| --- |
| <!-- 单个元素 -->  <span v-once>This will never change: {{msg}}</span>  <!-- 有子元素 -->  <div v-once>  <h1>comment</h1>  <p>{{msg}}</p>  </div>  <!-- 组件 -->  <my-component v-once :comment="msg"></my-component>  <!-- `v-for` 指令-->  <ul>  <li v-for="i in list" v-once>{{i}}</li>  </ul> |

* **参考**：
  + [**数据绑定语法- 插值**](https://cn.vuejs.org/v2/guide/syntax.html#插值)
  + [**组件 - 对低开销的静态组件使用 v-once**](https://cn.vuejs.org/v2/guide/components.html#对低开销的静态组件使用-v-once)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22v-once%22&type=Code)

**特殊特性**

**key**

* **预期**：number | string

key 的特殊属性主要用在 Vue 的虚拟 DOM 算法，在新旧 nodes 对比时辨识 VNodes。如果不使用 key，Vue 会使用一种最大限度减少动态元素并且尽可能的尝试修复/再利用相同类型元素的算法。使用 key，它会基于 key 的变化重新排列元素顺序，并且会移除 key 不存在的元素。

有相同父元素的子元素必须有**独特的 key**。重复的 key 会造成渲染错误。

最常见的用例是结合 v-for：

|  |
| --- |
| <ul>  <li v-for="item in items" :key="item.id">...</li>  </ul> |

它也可以用于强制替换元素/组件而不是重复使用它。当你遇到如下场景时它可能会很有用：

* + 完整地触发组件的生命周期钩子
  + 触发过渡

例如：

|  |
| --- |
| <transition>  <span :key="text">{{ text }}</span>  </transition> |

当 text 发生改变时，<span> 会随时被更新，因此会触发过渡。

* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22key%22&type=Code)

**ref**

* **预期**：string

ref 被用来给元素或子组件注册引用信息。引用信息将会注册在父组件的 $refs 对象上。如果在普通的 DOM 元素上使用，引用指向的就是 DOM 元素；如果用在子组件上，引用就指向组件实例：

|  |
| --- |
| <!-- vm.$refs.p will be the DOM node -->  <p ref="p">hello</p>  <!-- vm.$refs.child will be the child comp instance -->  <child-comp ref="child"></child-comp> |

当 v-for 用于元素或组件的时候，引用信息将是包含 DOM 节点或组件实例的数组。

关于 ref 注册时间的重要说明：因为 ref 本身是作为渲染结果被创建的，在初始渲染的时候你不能访问它们 - 它们还不存在！$refs 也不是响应式的，因此你不应该试图用它在模板中做数据绑定。

* **参考**：[**子组件 Refs**](https://cn.vuejs.org/v2/guide/components.html#子组件索引)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22ref%22&type=Code)

**slot**

* **预期**：string

用于标记往哪个具名插槽中插入子组件内容。

详细用法，请参考下面指南部分的链接。

* **参考**：[**具名插槽**](https://cn.vuejs.org/v2/guide/components.html#具名插槽)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22slot%22&type=Code)

**is**

* **预期**：string

用于[**动态组件**](https://cn.vuejs.org/v2/guide/components.html#动态组件)且基于 [**DOM 内模板的限制**](https://cn.vuejs.org/v2/guide/components.html#DOM-模板解析说明)来工作。

示例：

|  |
| --- |
| <!-- component changes when currentView changes -->  <component v-bind:is="currentView"></component>  <!-- necessary because `<my-row>` would be invalid inside -->  <!-- a `<table>` element and so would be hoisted out -->  <table>  <tr is="my-row"></tr>  </table> |

更多的使用细节，请移步至下面的链接。

* **See also**：
  + [**动态组件**](https://cn.vuejs.org/v2/guide/components.html#动态组件)
  + [**DOM 模板解析说明**](https://cn.vuejs.org/v2/guide/components.html#DOM-模板解析说明)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22is%22&type=Code)

**内置的组件**

**component**

* **Props**：
  + is - string | ComponentDefinition | ComponentConstructor
  + inline-template - boolean
* **用法**：

渲染一个“元组件”为动态组件。依 is 的值，来决定哪个组件被渲染。

|  |
| --- |
| <!-- 动态组件由 vm 实例的属性值 `componentId` 控制 -->  <component :is="componentId"></component>  <!-- 也能够渲染注册过的组件或 prop 传入的组件 -->  <component :is="$options.components.child"></component> |

* **参考**：[**动态组件**](https://cn.vuejs.org/v2/guide/components.html#动态组件)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22component%22&type=Code)

**transition**

* **Props**：
  + name - string，用于自动生成 CSS 过渡类名。例如：name: 'fade' 将自动拓展为.fade-enter，.fade-enter-active等。默认类名为 "v"
  + appear - boolean，是否在初始渲染时使用过渡。默认为 false。
  + css - boolean，是否使用 CSS 过渡类。默认为 true。如果设置为 false，将只通过组件事件触发注册的 JavaScript 钩子。
  + type - string，指定过渡事件类型，侦听过渡何时结束。有效值为 "transition" 和 "animation"。默认 Vue.js 将自动检测出持续时间长的为过渡事件类型。
  + mode - string，控制离开/进入的过渡时间序列。有效的模式有 "out-in" 和 "in-out"；默认同时生效。
  + enter-class - string
  + leave-class - string
  + appear-class - string
  + enter-to-class - string
  + leave-to-class - string
  + appear-to-class - string
  + enter-active-class - string
  + leave-active-class - string
  + appear-active-class - string
* **事件**：
  + before-enter
  + before-leave
  + before-appear
  + enter
  + leave
  + appear
  + after-enter
  + after-leave
  + after-appear
  + enter-cancelled
  + leave-cancelled (v-show only)
  + appear-cancelled
* **用法**：

<transition> 元素作为单个元素/组件的过渡效果。<transition> 只会把过渡效果应用到其包裹的内容上，而不会额外渲染 DOM 元素，也不会出现在检测过的组件层级中。

|  |
| --- |
| <!-- 简单元素 -->  <transition>  <div v-if="ok">toggled content</div>  </transition>  <!-- 动态组件 -->  <transition name="fade" mode="out-in" appear>  <component :is="view"></component>  </transition>  <!-- 事件钩子 -->  <div id="transition-demo">  <transition @after-enter="transitionComplete">  <div v-show="ok">toggled content</div>  </transition>  </div> |

|  |
| --- |
| new Vue({  ...  methods: {  transitionComplete: function (el) {  // 传入 'el' 这个 DOM 元素作为参数。  }  }  ...  }).$mount('#transition-demo') |

* **参考**：[**过渡：进入，离开和列表**](https://cn.vuejs.org/v2/guide/transitions.html)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22transition%22&type=Code)

**transition-group**

* **Props**：
  + tag - string，默认为 span
  + move-class - 覆盖移动过渡期间应用的 CSS 类。
  + 除了 mode，其他特性和 <transition> 相同。
* **事件**：
  + 事件和 <transition> 相同。
* **用法**：

<transition-group> 元素作为多个元素/组件的过渡效果。<transition-group> 渲染一个真实的 DOM 元素。默认渲染 <span>，可以通过 tag 属性配置哪个元素应该被渲染。

注意，每个 <transition-group> 的子节点必须有 **独立的 key** ，动画才能正常工作

<transition-group> 支持通过 CSS transform 过渡移动。当一个子节点被更新，从屏幕上的位置发生变化，它将会获取应用 CSS 移动类 (通过 name 属性或配置 move-class属性自动生成)。如果 CSS transform 属性是“可过渡”属性，当应用移动类时，将会使用 [**FLIP 技术**](https://aerotwist.com/blog/flip-your-animations/) 使元素流畅地到达动画终点。

|  |
| --- |
| <transition-group tag="ul" name="slide">  <li v-for="item in items" :key="item.id">  {{ item.text }}  </li>  </transition-group> |

* **参考**：[**过渡：进入，离开和列表**](https://cn.vuejs.org/v2/guide/transitions.html)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22transition-group%22&type=Code)

**keep-alive**

* **Props**：
  + include - 字符串或正则表达式。只有匹配的组件会被缓存。
  + exclude - 字符串或正则表达式。任何匹配的组件都不会被缓存。
* **用法**：

<keep-alive> 包裹动态组件时，会缓存不活动的组件实例，而不是销毁它们。和 <transition> 相似，<keep-alive> 是一个抽象组件：它自身不会渲染一个 DOM 元素，也不会出现在父组件链中。

当组件在 <keep-alive> 内被切换，它的 activated 和 deactivated 这两个生命周期钩子函数将会被对应执行。

**在 2.2.0 及其更高版本中，activated 和 deactivated 将会在 <keep-alive> 树内的所有嵌套组件中触发。**

主要用于保留组件状态或避免重新渲染。

|  |
| --- |
| <!-- 基本 -->  <keep-alive>  <component :is="view"></component>  </keep-alive>  <!-- 多个条件判断的子组件 -->  <keep-alive>  <comp-a v-if="a > 1"></comp-a>  <comp-b v-else></comp-b>  </keep-alive>  <!-- 和 `<transition>` 一起使用 -->  <transition>  <keep-alive>  <component :is="view"></component>  </keep-alive>  </transition> |

注意，<keep-alive> 是用在其一个直属的子组件被开关的情形。如果你在其中有 v-if 则不会工作。如果有上述的多个条件性的子元素，<keep-alive> 要求同时只有一个子元素被渲染。

* **include and exclude**

**2.1.0 新增**

include 和 exclude 属性允许组件有条件地缓存。二者都可以用逗号分隔字符串、正则表达式或一个数组来表示：

|  |
| --- |
| <!-- 逗号分隔字符串 -->  <keep-alive include="a,b">  <component :is="view"></component>  </keep-alive>  <!-- 正则表达式 (使用 `v-bind`) -->  <keep-alive :include="/a|b/">  <component :is="view"></component>  </keep-alive>  <!-- 数组 (使用 `v-bind`) -->  <keep-alive :include="['a', 'b']">  <component :is="view"></component>  </keep-alive> |

匹配首先检查组件自身的 name 选项，如果 name 选项不可用，则匹配它的局部注册名称 (父组件 components 选项的键值)。匿名组件不能被匹配。

<keep-alive> 不会在函数式组件中正常工作，因为它们没有缓存实例。

* **参考**：[**动态组件 - keep-alive**](https://cn.vuejs.org/v2/guide/components.html#keep-alive)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22keep-alive%22&type=Code)

**slot**

* **Props**：
  + name - string，用于命名插槽。
* **Usage**：

<slot> 元素作为组件模板之中的内容分发插槽。<slot> 元素自身将被替换。

详细用法，请参考下面教程的链接。

* **参考**：[**使用插槽分发内容**](https://cn.vuejs.org/v2/guide/components.html#使用插槽分发内容)
* [**源代码**](https://github.com/search?utf8=%E2%9C%93&q=repo%3Avuejs%2Fvue+extension%3Ajs+%22slot%22&type=Code)

**VNode 接口**

* 请参考 [**VNode class declaration**](https://github.com/vuejs/vue/blob/dev/src/core/vdom/vnode.js)。

**服务端渲染**

* 请参考 [**vue-server-renderer package documentation**](https://github.com/vuejs/vue/tree/dev/packages/vue-server-renderer)。

发现错误？想参与编辑？ [**在 Github 上编辑此页！**](https://github.com/vuejs/cn.vuejs.org/blob/master/src/v2/api/index.md)