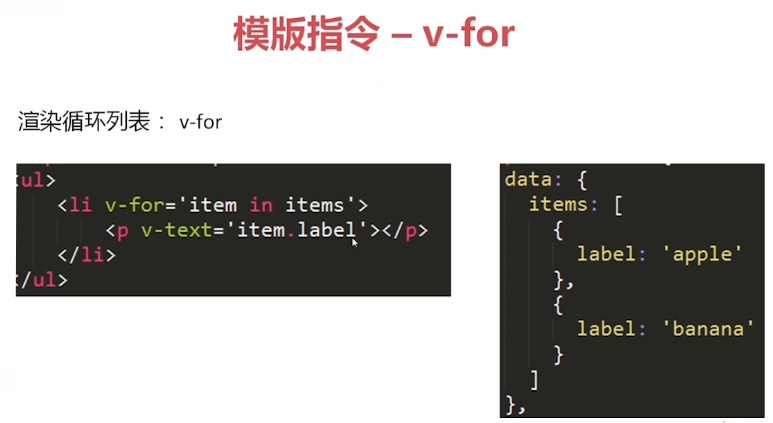


区别：

V-if:不显示dom元素

V-show 控制display ，存在dom





下方为简写

从method中取方法



对属性的操作

最常用的是class

简写：class

数组（同时有多个class）

可以混写

**[使用 JavaScript 表达式](http://cn.vuejs.org/v2/guide/syntax.html" \l "使用-JavaScript-表达式)**

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（Directives）**

是带有 v- 前缀的特殊属性。指令属性的值预期是**单一 JavaScript 表达式**（除了 v-for，之后再讨论）。指令的职责就是当其表达式的值改变时相应地将某些行为应用到 DOM 上。让我们回顾一下在介绍里的例子：

|  |
| --- |
| <p v-if="seen">Now you see me</p> |

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

一些指令能接受一个“参数”，在指令后以冒号指明

v-bind 指令被用来响应地更新 HTML 属性：

|  |
| --- |
| <a v-bind:href="url"></a>  另一个例子是 v-on 指令，它用于监听 DOM 事件： |

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

修饰符（Modifiers）是以半角句号 . 指明的特殊后缀，用于指出一个指定应该以特殊方式绑定。例如，.prevent 修饰符告诉 v-on 指令对于触发的事件调用 event.preventDefault()：

|  |
| --- |
| <form v-on:submit.prevent="onSubmit"></form> |

### [v-bind 缩写](http://cn.vuejs.org/v2/guide/syntax.html" \l "v-bind-缩写)

|  |
| --- |
| <!-- 完整语法 -->  <a v-bind:href="url"></a>  <!-- 缩写 -->  <a :href="url"></a> |

### v-on 缩写

|  |
| --- |
| <!-- 完整语法 -->  <a v-on:click="doSomething"></a>  <!-- 缩写 -->  <a @click="doSomething"></a> |

### [计算缓存 vs Methods](http://cn.vuejs.org/v2/guide/computed.html" \l "计算缓存-vs-Methods)

你可能已经注意到我们可以通过调用表达式中的method来达到同样的效果

不经过计算属性，我们可以在 method 中定义一个相同的函数来替代它。对于最终的结果，两种方式确实是相同的。然而，不同的是**计算属性是基于它的依赖缓存**。计算属性只有在它的相关依赖发生改变时才会重新取值。这就意味着只要 message 没有发生改变，多次访问 reversedMessage 计算属性会立即返回之前的计算结果，而不必再次执行函数。

这也同样意味着如下计算属性将不会更新，因为 Date.now() 不是响应式依赖：

|  |
| --- |
| computed: {  now: function () {  return Date.now()  }  } |

相比而言，每当重新渲染的时候，method 调用**总会**执行函数。

我们也可以在对象中传入更多属性用来动态切换多个 class 。此外， v-bind:class 指令可以与普通的 class 属性共存。如下模板:

|  |
| --- |
| <div class="static"  v-bind:class="{ active: isActive, 'text-danger': hasError }">  </div> |

v-bind:style 的对象语法十分直观——看着非常像 CSS ，其实它是一个 JavaScript 对象。 CSS 属性名可以用驼峰式（camelCase）或短横分隔命名（kebab-case）：

### [<template> 中 v-if 条件组](http://cn.vuejs.org/v2/guide/conditional.html" \l "lt-template-gt-中-v-if-条件组)

因为 v-if 是一个指令，需要将它添加到一个元素上。但是如果我们想切换多个元素呢？此时我们可以把一个 <template> 元素当做包装元素，并在上面使用 v-if，最终的渲染结果不会包含它。

|  |
| --- |
| <template v-if="ok">  <h1>Title</h1>  <p>Paragraph 1</p>  <p>Paragraph 2</p>  </template> |

### v-else

可以用 v-else 指令给 v-if 添加一个 “else” 块：

|  |
| --- |
| <div v-if="Math.random() > 0.5">  Sorry  </div>  <div v-else>  Not sorry  </div> |

v-else 元素必须紧跟在 v-if 元素或者 v-else-if的后面——否则它不能被识别。

使用组件时，大多数可以传入到 Vue 构造器中的选项可以在注册组件时使用，有一个例外： data 必须是函数。 实际上，如果你这么做：

|  |
| --- |
| Vue.component('my-component', {  template: '<span>{{ message }}</span>',  data: {  message: 'hello'  }  }) |

那么 Vue 会在控制台发出警告，告诉你在组件中 data 必须是一个函数。最好理解这种规则的存在意义。

|  |
| --- |
| <div id="example-2">  <simple-counter></simple-counter>  <simple-counter></simple-counter>  <simple-counter></simple-counter>  </div> |

|  |
| --- |
| var data = { counter: 0 }  Vue.component('simple-counter', {  template: '<button v-on:click="counter += 1">{{ counter }}</button>',   // data 是一个函数，因此 Vue 不会警告，  // 但是我们为每一个组件返回了同一个对象引用  data: function () {  return data  }  })  new Vue({  el: '#example-2'  }) |

0 0 0

### [使用 Prop 传递数据](http://cn.vuejs.org/v2/guide/components.html" \l "使用-Prop-传递数据)

组件实例的作用域是**孤立的**。这意味着不能并且不应该在子组件的模板内直接引用父组件的数据。可以使用 props 把数据传给子组件。

组件

当被包裹的元素为特殊元素时

is 属性：

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

使用组件时，大多数可以传入到 Vue 构造器中的选项可以在注册组件时使用，有一个例外： data 必须是函数。

### [使用 v-on 绑定自定义事件](http://cn.vuejs.org/v2/guide/components.html" \l "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/" \l "Instance-Methods-Events)，即：

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