Vue.js的指令是指v-开头，作用于html标签，提供一些特殊的特性，当指令被绑定到html元素的时候，指令会为被绑定的元素添加一些特殊的行为，可以将指令看成html的一种属性

1. v-text

v-text主要用来更新textContent，可以等同于JS的text属性。

```
<span v-text="msg"></span>
```

这两者等价：

```
<span>{{msg}}</span>
```

---

2. v-html

双大括号的方式会将数据解释为纯文本，而非HTML。为了输出真正的HTML，可以用v-html指令。它等同于JS的innerHtml属性。

```
<div v-html="rawHtml"></div>
```

这个div的内容将会替换成属性值rawHtml，直接作为HTML进行渲染。

---

3. v-pre

v-pre主要用来跳过这个元素和它的子元素编译过程。可以用来显示原始的Mustache标签。跳过大量没有指令的节点加快编译。

```
<div id="app"> <span v-pre>{{message}}</span> //这条语句不进行编译 <span>{{message}}</span> </div>
```

最终仅显示第二个span的内容

---

4. v-cloak

这个指令是用来保持在元素上直到关联实例结束时进行编译。

```
<div id="app" v-cloak> <div> {{message}} </div> </div> <script type="text/javascript"> new Vue({ el:'#app', data:{ message:'hello world' } }) </script>
```

在页面加载时会闪烁，先显示:

```
<div> {{message}} </div>
```

然后才会编译为：

```
<div> hello world! </div>
```

---

5. v-once

v-once关联的实例，只会渲染一次。之后的重新渲染，实例极其所有的子节点将被视为静态内容跳过，这可以用于优化更新性能。

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

上面的例子中，msg,list即使产生改变，也不会重新渲染。

---

6. v-if

v-if可以实现条件渲染，Vue会根据表达式的值的真假条件来渲染元素。

```
<a v-if="ok">yes</a>
```

如果属性值ok为true，则显示。否则，不会渲染这个元素。

---

7. v-else

v-else是搭配v-if使用的，它必须紧跟在v-if或者v-else-if后面，否则不起作用。

```
<a v-if="ok">yes</a> <a v-else>No</a>
```

---

8. v-else-if

v-else-if充当v-if的else-if块，可以链式的使用多次。可以更加方便的实现switch语句。

```
<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>
```

---

9. v-show

```
<h1 v-show="ok">hello world</h1>
```

也是用于根据条件展示元素。和v-if不同的是，如果v-if的值是false，则这个元素被销毁，不在dom中。但是v-show的元素会始终被渲染并保存在dom中，它只是简单的切换css的dispaly属性。

注意：v-if有更高的切换开销

v-show有更高的初始渲染开销。

因此，如果要非常频繁的切换，则使用v-show较好；如果在运行时条件不太可能改变，则v-if较好

---

10. v-for

用v-for指令根据遍历数组来进行渲染

有下面两种遍历形式

```
<div v-for="(item,index) in items"></div> //使用in，index是一个可选参数，表示当前项的索引 <div v-for="item of items"></div> //使用of
```

下面是一个例子，并且在v-for中，拥有对父作用域属性的完全访问权限。

```
<ul id="app"> <li v-for="item in items"> {{parent}}-{{item.text}} </li> </ul> <script type="text/javascript"> var example = new Vue({ el:’#app’, data:{ parent:’父作用域’ items:[ {text:’文本1’}, {text:’文本2’} ] } }) </script>
```

会被渲染为：

```
<ul id="app"> <li>父作用域-文本1</li> <li>父作用域-文本2</li> </ul>
```

注意：当v-for和v-if同处于一个节点时，v-for的优先级比v-if更高。这意味着v-if将运行在每个v-for循环中

---

11. v-bind

v-bind用来**动态的绑定一个或者多个特性**。没有参数时，可以绑定到一个包含键值对的对象。常用于动态绑定class和style。以及href等。

简写为一个冒号【：】

<1>对象语法：

```
//进行类切换的例子 <div id="app"> <!--当data里面定义的isActive等于true时，is-active这个类才会被添加起作用--> <!--当data里面定义的hasError等于true时，text-danger这个类才会被添加起作用--> <div :class="{'is-active':isActive, 'text-danger':hasError}"></div> </div> <script> var app = new Vue({ el: '#app', data: { isActive: true, hasError: false } }) </script>
```

渲染结果：

```
<!--因为hasError: false，所以text-danger不被渲染--> <div class = "is-active"></div>
```

<2>数组语法

```
<div id="app"> <!--数组语法：errorClass在data对应的类一定会添加--> <!--is-active是对象语法，根据activeClass对应的取值决定是否添加--> <p :class="[{'is-active':activeClass},errorClass]">12345</p> </div> <script> var app = new Vue({ el: '#app', data: { activeClass: false, errorClass: 'text-danger' } }) </script>
```

渲染结果：

```
<!--因为activeClass: false，所以is-active不被渲染--> <p class = "text-danger"></p>
```

<3>直接绑定数据对象

```
<div id="app"> <!--在vue实例的data中定义了classObject对象，这个对象里面是所有类名及其真值--> <!--当里面的类的值是true时会被渲染--> <div :class="classObject">12345</div> </div> <script> var app = new Vue({ el: '#app', data: { classObject:{ 'is-active': false, 'text-danger':true } } }) </script>
```

渲染结果：

```
<!--因为'is-active': false，所以is-active不被渲染--> <div class = "text-danger"></div>
```

---

12. v-model

这个指令用于在表单上创建**双向数据绑定。**

v-model会忽略所有表单元素的value、checked、selected特性的初始值。因为它选择Vue实例数据做为具体的值。

```
<div id="app"> <input v-model="somebody"> <p>hello {{somebody}}</p> </div> <script> var app = new Vue({ el: '#app', data: { somebody:'小明' } }) </script>
```

这个例子中直接在浏览器input中输入别的名字，下面的p的内容会直接跟着变。这就是双向数据绑定。

v-model修饰符

<1> .lazy

默认情况下，v-model同步输入框的值和数据。可以通过这个修饰符，转变为在change事件再同步。

```
<input v-model.lazy="msg">
```

<2> .number

自动将用户的输入值转化为数值类型

```
<input v-model.number="msg">
```

<3> .trim

自动过滤用户输入的首尾空格

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

---

13. v-on

v-on主要用来监听dom事件，以便执行一些代码块。表达式可以是一个方法名。

简写为：【 @ 】

```
<div id="app"> <button @click="consoleLog"></button> </div> <script> var app = new Vue({ el: '#app', methods:{ consoleLog:function (event) { console.log(1) } } }) </script>
```

事件修饰符

- .stop 阻止事件继续传播
- .prevent 事件不再重载页面
- .capture 使用事件捕获模式,即元素自身触发的事件先在此处处理，然后才交由内部元素进行处理
- .self 只当在 event.target 是当前元素自身时触发处理函数
- .once 事件将只会触发一次
- .passive 告诉浏览器你不想阻止事件的默认行为

```
<!-- 阻止单击事件继续传播 --> <a v-on:click.stop="doThis"></a> <!-- 提交事件不再重载页面 --> <form v-on:submit.prevent="onSubmit"></form> <!-- 修饰符可以串联 --> <a v-on:click.stop.prevent="doThat"></a> <!-- 只有修饰符 --> <form v-on:submit.prevent></form> <!-- 添加事件监听器时使用事件捕获模式 --> <!-- 即元素自身触发的事件先在此处处理，然后才交由内部元素进行处理 --> <div v-on:click.capture="doThis">...</div> <!-- 只当在 event.target 是当前元素自身时触发处理函数 --> <!-- 即事件不是从内部元素触发的 --> <div v-on:click.self="doThat">...</div> <!-- 点击事件将只会触发一次 --> <a v-on:click.once="doThis"></a> <!-- 滚动事件的默认行为 (即滚动行为) 将会立即触发 --> <!-- 而不会等待 `onScroll` 完成 --> <!-- 这其中包含 `event.preventDefault()` 的情况 --> <div v-on:scroll.passive="onScroll">...</div>
```

使用修饰符时，顺序很重要；相应的代码会以同样的顺序产生。因此，用v-on:click.prevent.self会**阻止所有的点击**，而 v-on:click.self.prevent 只会阻止对元素自身的点击。