1 v-html 指令用于输出 html 代码：

<div id="app"> <div v-html="message"></div> </div>

<script>

new Vue({ el: '#app', data: { message: '<h1>菜鸟教程</h1>' } })

</script>

2 HTML 属性中的值应使用 v-bind 指令。

以下实例判断 class1 的值，如果为 true 使用 class1 类的样式，否则不使用该类

<div id="app">

<label for="r1">修改颜色</label>

<input type="checkbox" v-model="class1" id="r1"> <br><br>

<div v-bind:class="{'class1': class1}"> directiva v-bind:class </div> </div>

<script> new Vue({ el: '#app', data:{ class1: false } }); </script>

v-bind:class="{'class1': class1}"

前面为值，后面为data中的key

3. 指令是带有 v- 前缀的特殊属性。

v-if v-else v-else-if

v-bind:href="url" 更新属性

v-bind:class="{'class1': class1}"

v-on:click="doSomething"

 v-on 指令，它用于监听 DOM 事件

<div id="app">

<p>{{ message }}</p>

<input v-model="message">

</div>

<script> new Vue({ el: '#app', data: { message: 'Runoob!' }})</script>

在 input 输入框中我们可以使用 v-model 指令来实现双向数据绑定

methods: { reverseMessage: function () { this.message = this.message.split('').reverse().join('') } }

事件放置于methods：{}中

**过滤器**

Vue.js 允许你自定义过滤器，被用作一些常见的文本格式化。由"管道符"指示, 格式如下：

<!-- 在两个大括号中 -->

{{ message | capitalize }}

<!-- 在 v-bind 指令中 -->

<div v-bind:id="rawId | formatId"></div>

过滤器函数接受表达式的值作为第一个参数。

以下实例对输入的字符串第一个字母转为大写：

**实例**

<div id="app"> {{ message | capitalize }} </div>

<script>

new Vue({

el: '#app',

data: { message: 'runoob' },

filters: {

capitalize: function (value) {

if (!value) return ''

value = value.toString()

return value.charAt(0).toUpperCase() + value.slice(1) } } })

</script>

过滤器可以串联：

{{ message | filterA | filterB }}

### v-bind 缩写

Vue.js 为两个最为常用的指令提供了特别的缩写：

<!-- 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>

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

vm.$el === document.getElementById('example') // -> true

### v-show

我们也可以使用 v-show 指令来根据条件展示元素：

## v-show 指令

<h1 v-show="ok">Hello!</h1>

注意: v-show 不支持 <template> 语法。

**v-for 指令**

<div id="app">

<ol> <li v-for="site in sites"> {{ site.name }} </li> </ol>

</div>

<script> new Vue({

el: '#app',

data: { sites: [ { name: 'Runoob' }, { name: 'Google' }, { name: 'Taobao' } ] } }) </script>

上面两个site可以修改为任意相同的两个值

<div id="app"> <ul> <li v-for="(value, key) in object"> {{ key }} : {{ value }} </li> </ul> </div>

<div id="app"> <ul> <li v-for="(value, key, index) in object"> {{ index }}. {{ key }} : {{ value }} </li> </ul> </div>

可以有3个参数，依次为值,key,index

## computed vs methods

我们可以使用 methods 来替代 computed，效果上两个都是一样的，但是 computed 是基于它的依赖缓存，只有相关依赖发生改变时才会重新取值。而使用 methods ，在重新渲染的时候，函数总会重新调用执行。

样式绑定

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

与小程序相反，判断条件在后，且当样式名中含有—时需用引号包含

<div id="app"> <div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }">菜鸟教程</div> </div>

<div v-bind:class="[errorClass ,isActive ? activeClass : '']"></div>

### 事件修饰符

Vue.js 为 v-on 提供了事件修饰符来处理 DOM 事件细节，如：event.preventDefault() 或 event.stopPropagation()。

Vue.js通过由点(.)表示的指令后缀来调用修饰符。

* .stop
* .prevent
* .capture
* .self
* .once

<!-- 阻止单击事件冒泡 -->

<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>

<!-- 添加事件侦听器时使用事件捕获模式 -->

<div v-on:click.capture="doThis">...</div>

<!-- 只当事件在该元素本身（而不是子元素）触发时触发回调 -->

<div v-on:click.self="doThat">...</div>

<!-- click 事件至少触发一次，2.1.4版本新增 -->

<a v-on:click.once="doThis"></a>

### 按键修饰符

Vue 允许为 v-on 在监听键盘事件时添加按键修饰符：

<!-- 只有在 keyCode 是 13 时调用 vm.submit() -->

<input v-on:keyup.13="submit">

记住所有的 keyCode 比较困难，所以 Vue 为最常用的按键提供了别名：

<!-- 同上 -->

<input v-on:keyup.enter="submit">

<!-- 缩写语法 -->

<input @keyup.enter="submit">

全部的按键别名：

* .enter
* .tab
* .delete (捕获 "删除" 和 "退格" 键)
* .esc
* .space
* .up
* .down
* .left
* .right
* .ctrl
* .alt
* .shift
* .meta

实例

<p><!-- Alt + C -->

<input @keyup.alt.67="clear">

<!-- Ctrl + Click -->

<div @click.ctrl="doSomething">Do something</div>

**用 v-model 指令在表单控件元素上创建双向数据绑定**

## 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：

<!-- 在 "change" 而不是 "input" 事件中更新 -->

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

### .number

如果想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），可以添加一个修饰符 number 给 v-model 来处理输入值：

<input v-model.number="age" type="number">

这通常很有用，因为在 type="number" 时 HTML 中输入的值也总是会返回字符串类型。

### .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">

注册一个全局组件语法格式如下：

Vue.component(tagName, options)

tagName 为组件名，options 为配置选项。注册后，我们可以使用以下方式来调用组件：

<tagName></tagName>

### 全局组件

所有实例都能用全局组件。

## 全局组件实例

注册一个简单的全局组件 runoob，并使用它：

<div id="app"> <runoob></runoob> </div> <script> // 注册 Vue.component('runoob', { template: '<h1>自定义组件!</h1>' }) // 创建根实例 new Vue({ el: '#app' }) </script>

### 局部组件

我们也可以在实例选项中注册局部组件，这样组件只能在这个实例中使用：

## 局部组件实例

注册一个简单的局部组件 runoob，并使用它：

<div id="app"> <runoob></runoob> </div> <script> var Child = { template: '<h1>自定义组件!</h1>' } // 创建根实例 new Vue({ el: '#app', components: { // <runoob> 将只在父模板可用 'runoob': Child } }) </script>

## Prop

prop 是父组件用来传递数据的一个自定义属性。

父组件的数据需要通过 props 把数据传给子组件，子组件需要显式地用 props 选项声明 "prop"：

<div id="app"> <child message="hello!"></child> </div>

<script>

// 注册

Vue.component('child', {

// 声明

props props: ['message'], // 同样也可以在 vm 实例中像 "this.message" 这样使用

template: '<span>{{ message }}</span>' })

// 创建根实例

new Vue({ el: '#app' }) </script>

**v-for=”(item,index) in super”**

**v-bind:class=”active”这个类不会覆盖，只会添加，所以前面可以存在类。**

**在JavaScript ES6中，export与export default均可用于导出常量、函数、文件、模块等，你可以在其它文件或模块中通过import+(常量 | 函数 | 文件 | 模块)名的方式，将其导入，以便能够对其进行使用，但在一个文件或模块中，export、import可以有多个，export default仅有一个。**

**具体使用：**

**1、//demo1.js**

**export const str = 'hello world'**

**export function f(a){**

**return a+1**

**}**

**对应的导入方式：**

**//demo2.js**

**import { str, f } from 'demo1' //也可以分开写两次，导入的时候带花括号**

**//demo1.js**

**export default const str = 'hello world'**

**对应的导入方式：**

**//demo2.js**

**import str from 'demo1' //导入的时候没有花括号**