Vue 视图

1. 基本模板语法

1.1 插值

文本

- 数据绑定最常见的形式就是使用“Mustache”语法 (双大括号) 的文本插值
- `v-text` 指令也可以用于数据绑定，如果要更新部分的 textContent ， 需要使用 {{ Mustache }} 插值。
- 通过使用 `v-once` 指令，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

原始HTML

双大括号会将数据解释为普通文本，而非 HTML 代码。为了输出真正的 HTML，你需要使用 v-html 指令

> 你的站点上动态渲染的任意 HTML 可能会非常危险，因为它很容易导致 XSS 攻击。请只对可信内容使用 HTML 插值，绝不要对用户提供的内容使用插值。

属性

- Mustache语法不能作用在 HTML 特性上，遇到这种情况应该使用 v-bind 指令
- 在布尔属性的情况下，它们的存在即暗示为 true， 如果值是 null、undefined 或 false，则属性不会被包含在渲染出来的

使用JavaScript表达式

```
{{ number + 1 }}

{{ ok ? 'YES' : 'NO' }}

{{ message.split('').reverse().join('') }}

<div v-bind:id="'list-' + id"></div>

<!-- 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 } }}
```

防止刷新或加载闪烁

```
<style>
    [v-cloak] {
        display: none !important;
    }
</style>

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

- v-cloak并不需要添加到每个标签，只要在el挂载的标签上添加就可以

1.2 指令

指令

- 指令 (Directives) 是带有 v- 前缀的特殊属性。
- 指令特性的值预期是单个 JavaScript 表达式
- 指令的职责是，当表达式的值改变时，将其产生的连带影响，响应式地作用于 DOM。

指令列表

- v-text

- v-html
- v-show
- v-if
- v-else
- v-else-if
- v-for
- v-on
- v-bind
- v-model
- v-pre
- v-cloak
- v-once

缩写

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>
```

2. 条件渲染与列表渲染

2.1 条件渲染

相关指令

- v-if
- v-else
- v-else-if

控制多个元素

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

复用已有元素

- Vue 会尽可能高效地渲染元素，通常会复用已有元素而不是从头开始渲染
- 要想每次都重新渲染，只需添加一个具有唯一值的 key 属性

v-show

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

与 v-if 的区别

- `v-if` 是“真正”的条件渲染，因为它会确保在切换过程中条件块内的事件监听器和子组件适当地被销毁和重建。
- `v-show` 就简单得多——不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 CSS 进行切换。
- 一般来说，v-if 有更高的切换开销，而 v-show 有更高的初始渲染开销。因此，如果需要非常频繁地切换，则使用 v-show 较好；如果在运行时条件很少改变，则使用 v-if 较好。

2.2 列表渲染

遍历数组

```
<ul id="example-1">
  <li v-for="item in items">
    {{ item.message }}
  </li>
</ul>
```

v-for 还支持一个可选的第二个参数为当前项的索引

```
<ul id="example-2">
  <li v-for="(item, index) in items">
    {{ index }} - {{ item.message }}
  </li>
</ul>
```

遍历对象

```
<!-- 只取值 -->
<li v-for="value in object">
  {{ value }}
</li>

<!-- 值、属性名 -->
<div v-for="(value, key) in object">
  {{ key }}: {{ value }}
</div>

<!--值、属性名、索引-->
<div v-for="(value, key, index) in object">
  {{ index }}. {{ key }}: {{ value }}
</div>
```

key

- 当 Vue.js 用 v-for 正在更新已渲染过的元素列表时，它默认用“就地复用”策略。
- 这个默认的模式是高效的，但是只适用于不依赖子组件状态或临时 DOM 状态 的列表渲染输出。
- 为了给 Vue 一个提示，以便它能跟踪每个节点的身份，从而重用和重新排序现有元素，你需要为每项提供一个唯一 key 属性。因为在遍历，需要用 v-bind 来绑定动态值

- 建议尽可能在使用 v-for 时提供 key，除非遍历输出的 DOM 内容非常简单，或者是刻意依赖默认行为以获取性能上的提升。

更新检测

数组

由于 JavaScript 的限制，Vue 不能检测以下变动的数组：

当你利用索引直接设置一个项时，例如：vm.items[indexOfItem] = newValue 当你修改数组的长度时，例如：vm.items.length = newLength

对象

还是由于 JavaScript 的限制，Vue 不能检测对象属性的添加或删除

方法

- `Vue.set()`
- `vm.$set()`

遍历数字

```
<div>
  <span v-for="n in 10">{{ n }} </span>
</div>
```

`v-for` on `<tmplate>`

```
<ul>
  <template v-for="item in items">
    <li>{{ item.msg }}</li>
    <li class="divider" role="presentation"></li>
  </template>
</ul>
```

`v-for` 和 `v-if`

当它们处于同一节点，v-for 的优先级比 v-if 更高，这意味着 v-if 将分别重复运行于每个 v-for 循环中。

```
<li v-for="todo in todos" v-if="!todo.isComplete">
  {{ todo }}
</li>
```

3 样式

3.1 绑定HTML Class

对象语法

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

数组语法

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

3.2 绑定内联样式

对象语法

```
<div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>
<div v-bind:style="styleObject"></div>
```

数组语法

数组语法可以将多个样式对象应用到同一个元素上

```
div v-bind:style="[baseStyles, overridingStyles]"></div>
```

自动添加前缀

当 `v-bind:style` 使用需要添加浏览器引擎前缀的 CSS 属性时，如 transform，Vue.js 会自动侦测并添加相应的前缀。

多重值

从 2.3.0 起你可以为 style 绑定中的属性提供一个包含多个值的数组，常用于提供多个带前缀的值

```
<div :style="{ display: ['-webkit-box', '-ms-flexbox', 'flex'] }"></div>
```

这样写只会渲染数组中最后一个被浏览器支持的值。在本例中，如果浏览器支持不带浏览器前缀的 flexbox，那么就只会渲染 display: flex。

4. 事件

4.1 事件绑定

监听事件

```
<div id="example-1">
  <button v-on:click="counter += 1">Add 1</button>
  <p>The button above has been clicked {{ counter }} times.</p>
</div>
```

事件处理方法

```
<div id="example-2">
  <!-- `greet` 是在下面定义的方法名 -->
  <button v-on:click="greet">Greet</button>
</div>
```

内联调用方法

```
<div id="example-3">
  <button v-on:click="say('hi')">Say hi</button>
  <button v-on:click="say('what')">Say what</button>
</div>
```

有时也需要在内联语句处理器中访问原始的 DOM 事件。可以用特殊变量 $event 把它传入方法

```
<button v-on:click="warn('Form cannot be submitted yet.', $event)">
  Submit
</button>
```

4.2 事件修饰符

- `.stop`
- `.prevent`
- `.capture`
- `.self`
- `.once`
- `.passive`

```
<!-- 阻止单击事件继续传播 阻止事件冒泡-->
<a v-on:click.stop="doThis"></a>

<!-- 提交事件不再重载页面 阻止默认事件-->
<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 2.1.4新增-->
<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 2.3.0新增-->
<!-- 而不会等待 `onScroll` 完成  -->
<!-- 这其中包含 `event.preventDefault()` 的情况 -->
<!--.passive 修饰符尤其能够提升移动端的性能。-->
<div v-on:scroll.passive="onScroll">...</div>
```

4.3 按键修饰符

数字

```
<!-- 只有在 `keyCode` 是 13 时调用 `vm.submit()` -->
<input v-on:keyup.13="submit">
```

按键别名

```
<!-- 回车键 -->
<input v-on:keyup.enter="submit">
```

- .enter
- .tab
- .delete (捕获“删除”和“退格”键)
- .esc
- .space
- .up
- .down
- .left
- .right

可以通过全局 config.keyCodes 对象自定义按键修饰符别名：

```
// 可以使用 `v-on:keyup.f1`
Vue.config.keyCodes.f1 = 112
```

4.4 系统修饰键

可以用如下修饰符来实现仅在按下相应按键时才触发鼠标或键盘事件的监听器。

- .ctrl
- .alt
- .shift
- .meta

> 注意：在 Mac 系统键盘上，meta 对应 command 键 (⌘)。在 Windows 系统键盘 meta 对应 Windows 徽标键 (⊞)。在 Sun 操作系统键盘上，meta 对应实心宝石键 (◆)。在其他特定键盘上，尤其在 MIT 和 Lisp 机器的键盘、以及其后继产品，比如 Knight 键盘、space-cadet 键盘，meta 被标记为“META”。在 Symbolics 键盘上，meta 被标记为“META”或者“Meta”。

```
<!-- Alt + C -->
<input @keyup.alt.67="clear">

<!-- Ctrl + Click -->
<div @click.ctrl="doSomething">Do something</div>
```

5 表单输入绑定

你可以用 `v-model` 指令在表单 `<input>` 及 `<textarea>` 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。

`v-model` 会忽略所有表单元素的 value、checked、selected 特性的初始值而总是将 Vue 实例的数据作为数据来源。你应该通过 JavaScript 在组件的 data 选项中声明初始值。

5.1 基础用法

文本

```
<input v-model="message" placeholder="edit me">
<textarea v-model="message" placeholder="add multiple lines"></textarea>
```

复选框

单个复选框，绑定到布尔值：

```
<input type="checkbox" id="checkbox" v-model="checked">

<input
  type="checkbox"
  v-model="toggle"
  true-value="yes"
  false-value="no"
>
```

多个复选框，绑定到同一个数组

```
<div id='example'>
  <input type="checkbox" id="jack" value="Jack" v-model="checkedNames">
  <label for="jack">Jack</label>
  <input type="checkbox" id="john" value="John" v-model="checkedNames">
  <label for="john">John</label>
  <input type="checkbox" id="mike" value="Mike" v-model="checkedNames">
  <label for="mike">Mike</label>
  <br>
  <span>Checked names: {{ checkedNames }}</span>
</div>
```

单选按钮

绑定value对应的字符串

```
<div id="example">
  <input type="radio" id="one" value="One" v-model="picked">
  <label for="one">One</label>
  <br>
  <input type="radio" id="two" value="Two" v-model="picked">
  <label for="two">Two</label>
  <br>
  <span>Picked: {{ picked }}</span>
</div>
```

选择框

单选，绑定对应所选的值

```
<div id="example">
  <select v-model="selected">
    <option disabled value="">请选择</option>
    <option>A</option>
    <option>B</option>
    <option>C</option>
  </select>
  <span>Selected: {{ selected }}</span>
</div>
```

多选时，绑定到一个数组

```
<div id="example">
  <select v-model="selected" multiple style="width: 50px;">
    <option>A</option>
    <option>B</option>
    <option>C</option>
  </select>
  <br>
  <span>Selected: {{ selected }}</span>
</div>
```

5.2 修饰符

`.lazy`

在默认情况下，v-model 在每次 input 事件触发后将输入框的值与数据进行同步

添加 lazy 修饰符，从而转变为使用 change 事件进行同步

```
<!-- 在“change”时而非“input”时更新 -->
<input v-model.lazy="msg" >
```

`.number`

如果想自动将用户的输入值转为数值类型，可以给 v-model 添加 number 修饰符

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

`.trim`

自动过滤用户输入的首尾空白字符，可以给 v-model 添加 trim 修饰符

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