Vue JS
The Progressive JavaScript Framework

21.03.2019

Hermes

Introduction

Vue (pronounced /vjuː/, like **view**) is a **progressive framework** for building user interfaces.

Following are the features available with VueJS:

1. Virtual DOM
2. Event Handling
3. Data Binding
4. Components
5. Computed Properties
6. Watchers

Note:

1. Vue **does not support IE8** and below
2. Latest stable version: **2.6.10**
3. <script src="https://cdn.jsdelivr.net/npm/vue@2.6.10/dist/vue.js">

   </script>

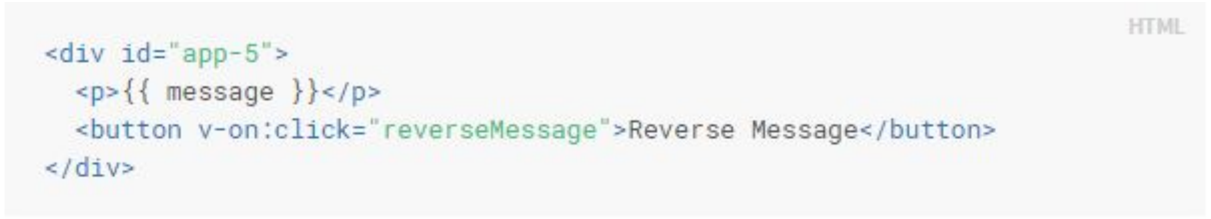
1. $ npm install vue
2. $ bower install vue

Vue Instance

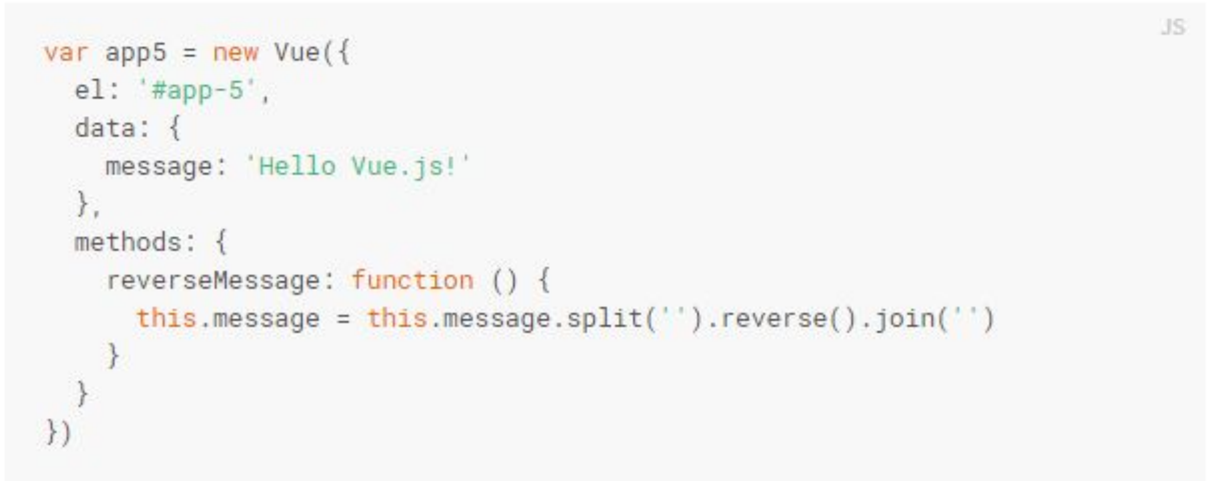
To start with VueJS, we need to create the instance of Vue, which is called the **root Vue Instance**.
The Vue instance is a **JavaScript object**.
The visible part of a Vue.js application is typically rendered via Vue.js **directives**.

Contains:

1. Element
2. Data
3. Methods
4. Filters
5. Watchers
6. Lifecycle hooks
7. Computed properties

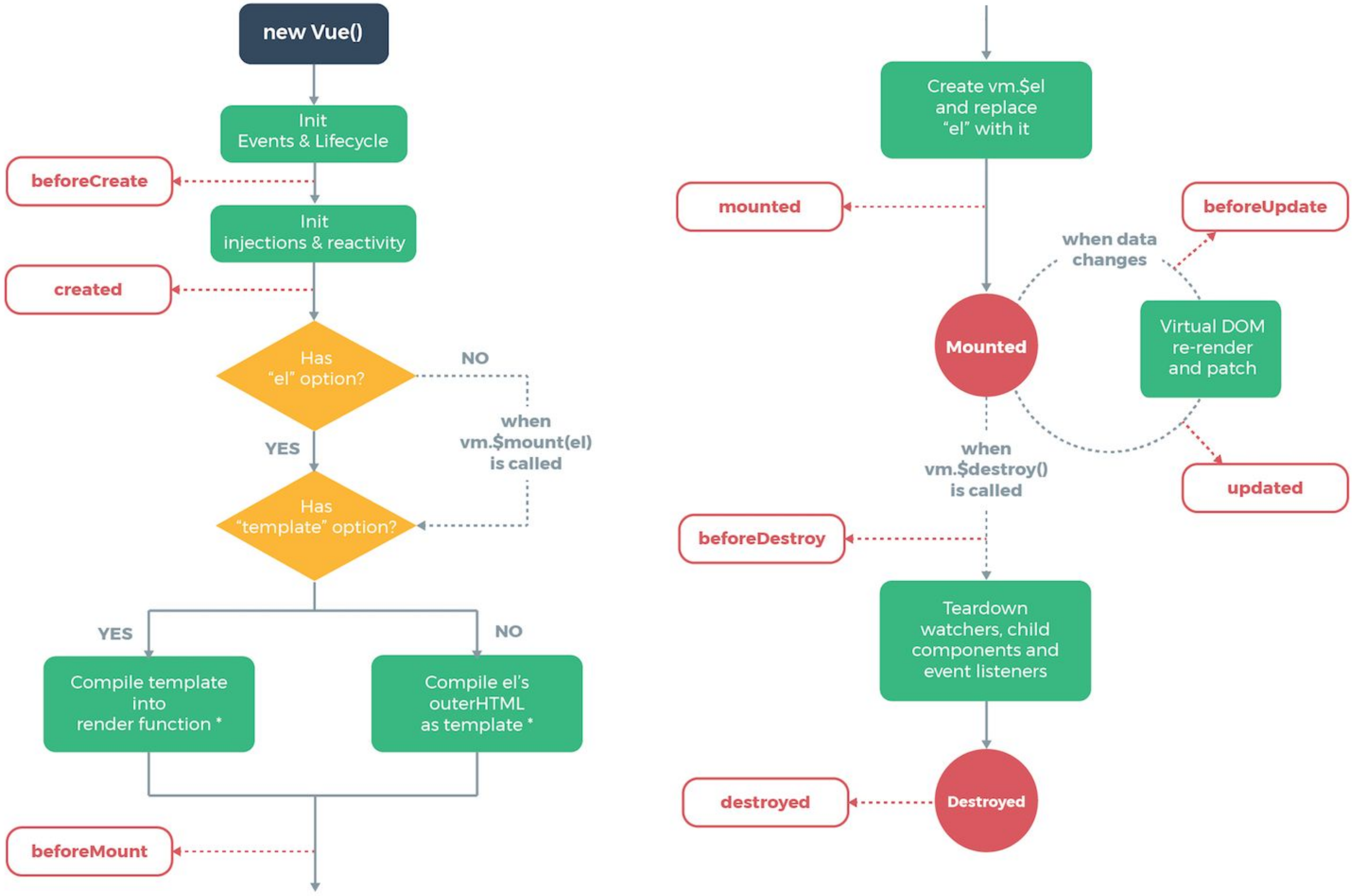


```html
<div id="app-5">
  <p>{{ message }}</p>
  <button v-on:click="reverseMessage">Reverse Message</button>
</div>
```



```js
var app5 = new Vue({
  el: '#app-5',
  data: {
    message: 'Hello Vue.js!'
  },
  methods: {
    reverseMessage: function () {
      this.message = this.message.split('').reverse().join('')
    }
  }
})
```

Lifecycle

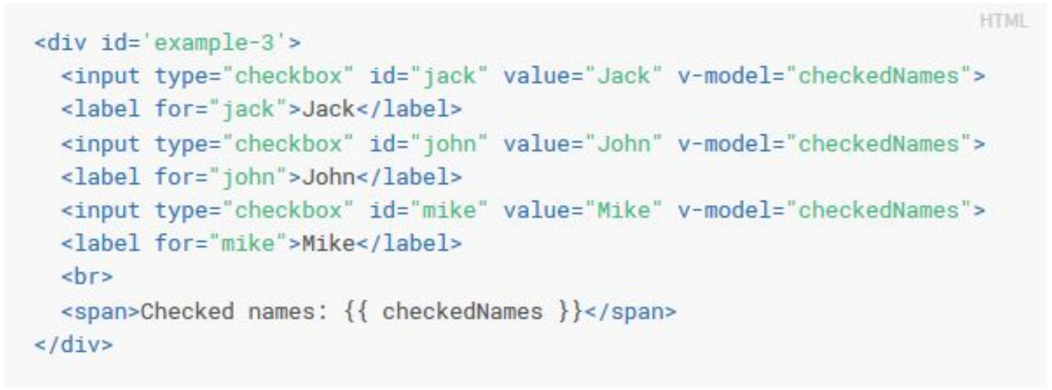

new Vue()
Init Events & Lifecycle
beforeCreate
Init injections & reactivity
created
Has "el" option?
NO
YES
when vm.$mount(el) is called
Has "template" option?
YES
NO
Compile template into render function *
Compile el's outerHTML as template *
beforeMount
Create vm.$el and replace "el" with it
mounted
beforeUpdate
when data changes
Mounted
Virtual DOM re-render and patch
updated
when vm.$destroy() is called
beforeDestroy
Teardown watchers, child components and event listeners
destroyed
Destroyed

Form binding

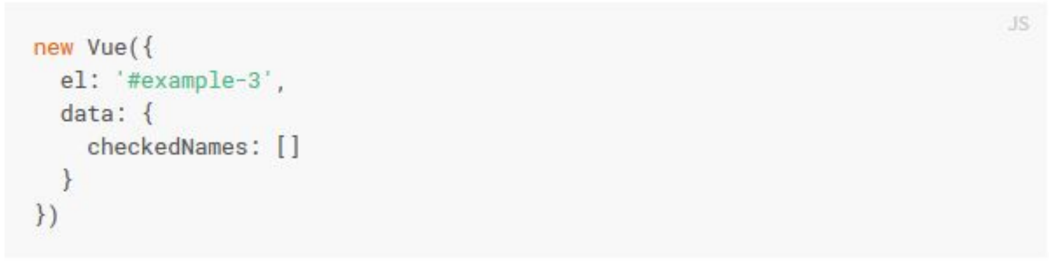
Directives: v-bind, v-model

v-model internally uses different properties and emits different events for different input elements:

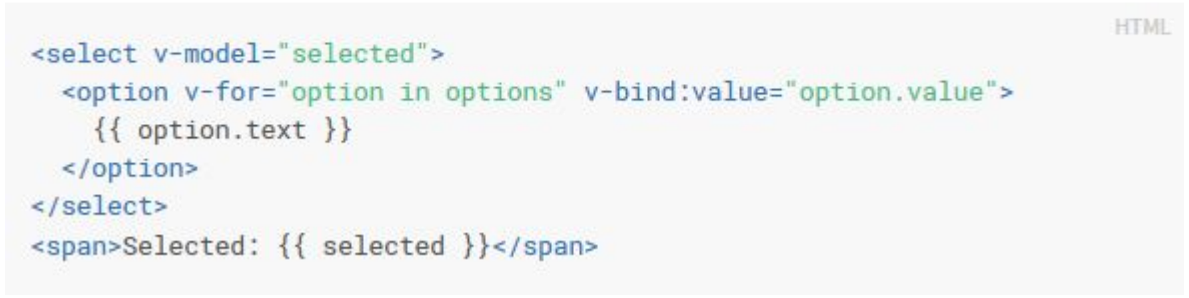
- text and textarea elements use value property and input event;
- checkboxes and radiobuttons use checked property and change event;
- select fields use value as a prop and change as an event.



```html
<div id='example-3'>
  <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>
```



```js
new Vue({
  el: '#example-3',
  data: {
    checkedNames: []
  }
})
```

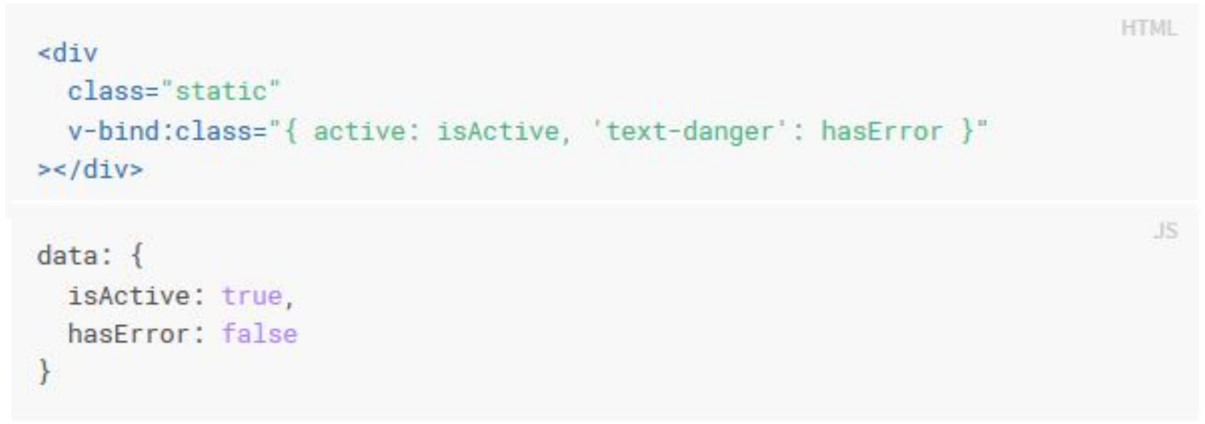


```html
<select v-model="selected">
  <option v-for="option in options" v-bind:value="option.value">
    {{ option.text }}
  </option>
</select>
<span>Selected: {{ selected }}</span>
```

Class and Style Bindings

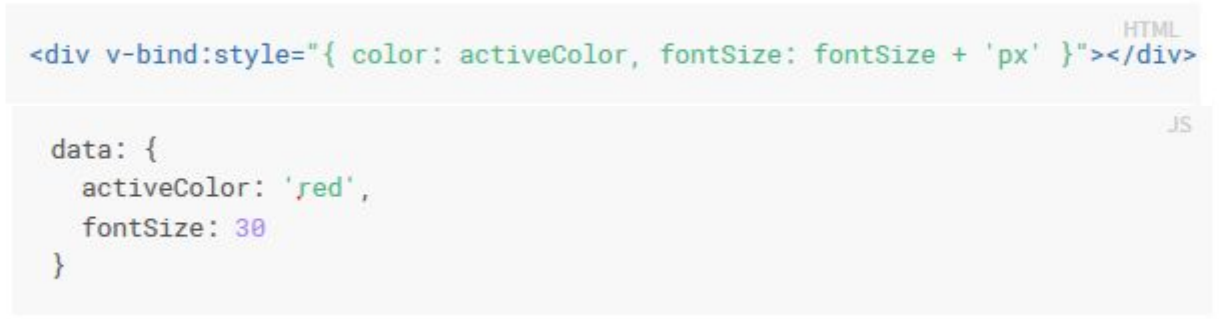
Directives:

1. **v-bind:class**
2. **v-bind:style**



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

```js
data: {
  isActive: true,
  hasError: false
}
```



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

```js
data: {
  activeColor: 'red',
  fontSize: 30
}
```

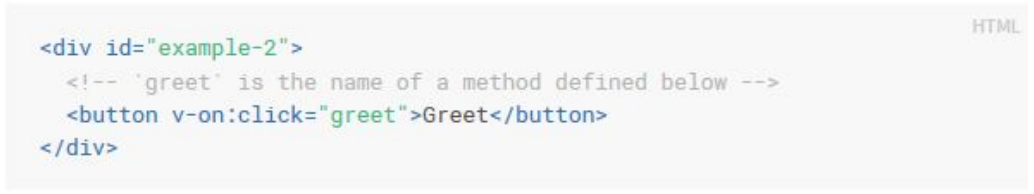
Events

Directive: v-on

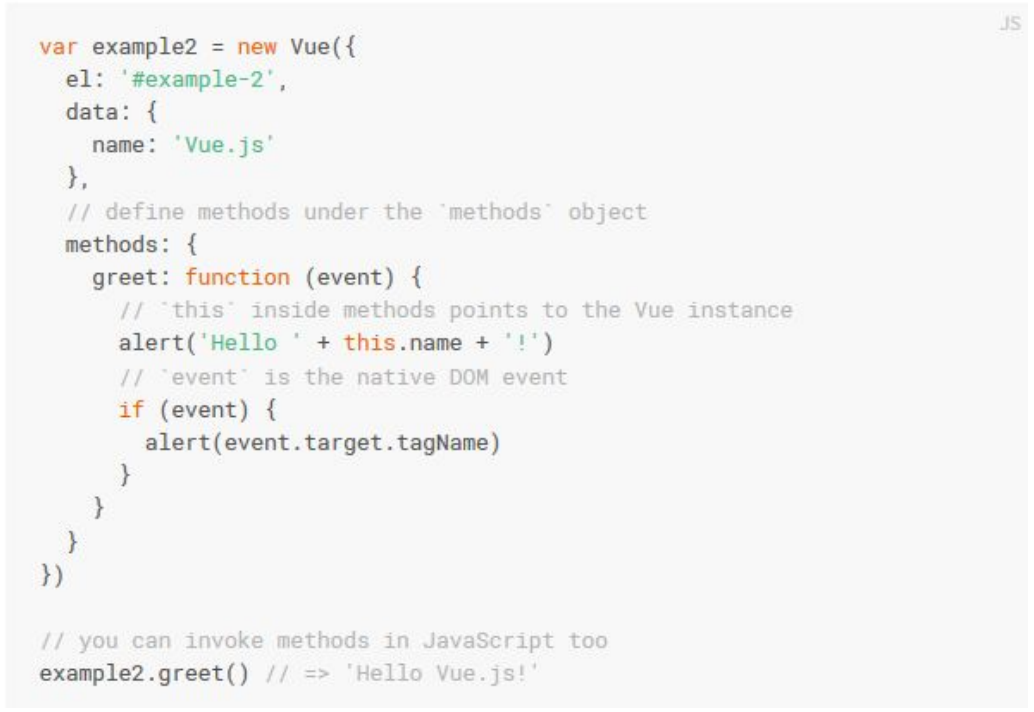
Pass event as a parameter with $event

Event listeners:

1. click
2. dblclick
3. keyup
4. keydown
5. change
6. input



```html
<div id="example-2">
  <!-- `greet` is the name of a method defined below -->
  <button v-on:click="greet">Greet</button>
</div>
```



```js
var example2 = new Vue({
  el: '#example-2',
  data: {
    name: 'Vue.js'
  },
  // define methods under the `methods` object
  methods: {
    greet: function (event) {
      // `this` inside methods points to the Vue instance
      alert('Hello ' + this.name + '!')
      // `event` is the native DOM event
      if (event) {
        alert(event.target.tagName)
      }
    }
  }
})

// you can invoke methods in JavaScript too
example2.greet() // => 'Hello Vue.js!'
```

Modifiers:

1. .stop
2. .prevent
3. .capture
4. .self
5. .once
6. .passive

List Rendering

Directive:v-for

Mapping an Array

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

```
var example1 = new Vue({
  el: '#example-1',
  data: {
    items: [
      { message: 'Foo' },
      { message: 'Bar' }
    ]
  }
})
```

Mapping an Object

```
<ul id="v-for-object" class="demo">
  <li v-for="value in object">
    {{ value }}
  </li>
</ul>
```

```
new Vue({
  el: '#v-for-object',
  data: {
    object: {
      firstName: 'John',
      lastName: 'Doe',
      age: 30
    }
  }
})
```

Computed and Watched Properties

Watchers:

1. Way to observe and react to data changes

```
<div id="demo">{{ fullName }}</div>
```

```
var vm = new Vue({
  el: '#demo',
  data: {
    firstName: 'Foo',
    lastName: 'Bar',
    fullName: 'Foo Bar'
  },
  watch: {
    firstName: function (val) {
      this.fullName = val + ' ' + this.lastName
    },
    lastName: function (val) {
      this.fullName = this.firstName + ' ' + val
    }
  }
})
```

Computed:

1. Cached based on their reactive dependencies
2. Re-evaluates only if some of its reactive dependencies changes
3. By default getter-only
4. Setter can be provided

```
var vm = new Vue({
  el: '#demo',
  data: {
    firstName: 'Foo',
    lastName: 'Bar'
  },
  computed: {
    fullName: function () {
      return this.firstName + ' ' + this.lastName
    }
  }
})
```

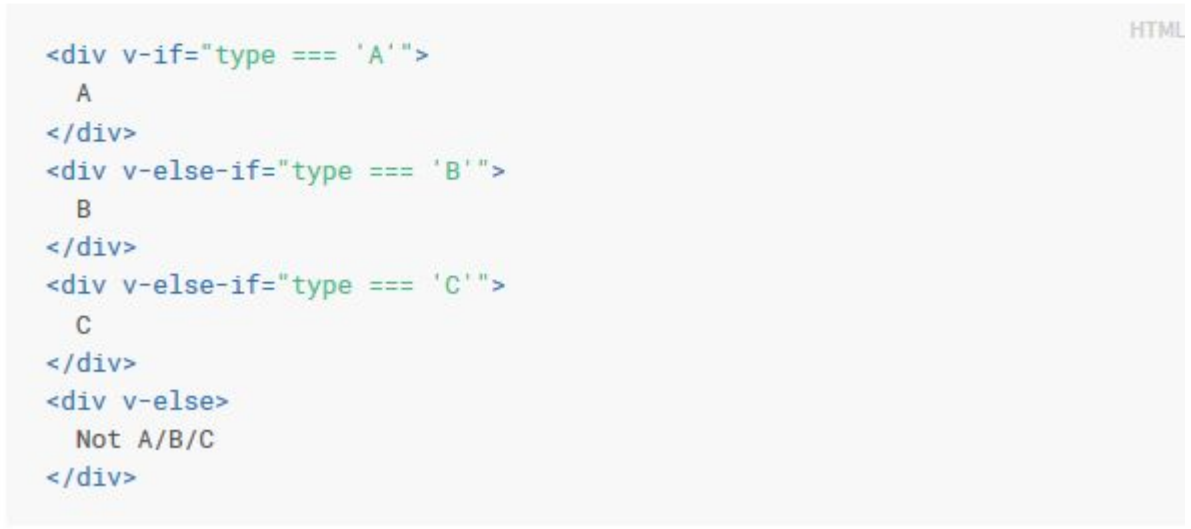
Conditional rendering

Directives:

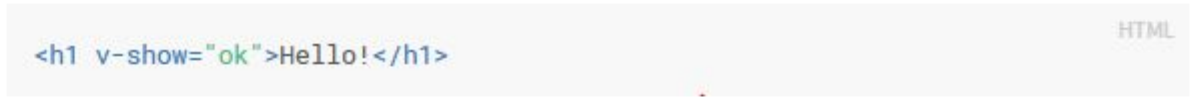
1. **v-if**
2. **v-else**
3. **v-else-if**
4. **v-show**

Usage:

1. Used to **conditionally render** a block.
2. v-if ensures that event **listeners** and **child components** inside the conditional block are properly **destroyed and re-created** during toggles.
3. v-show **toggles the display CSS property** of the element



```html
<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>
```

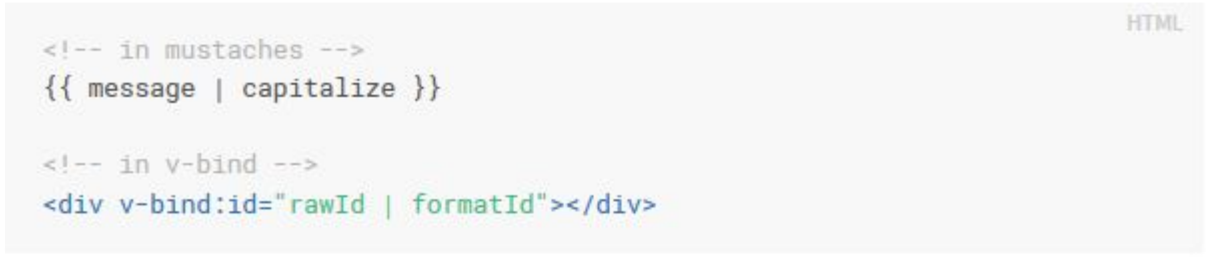


```html
<h1 v-show="ok">Hello!</h1>
```

Filters

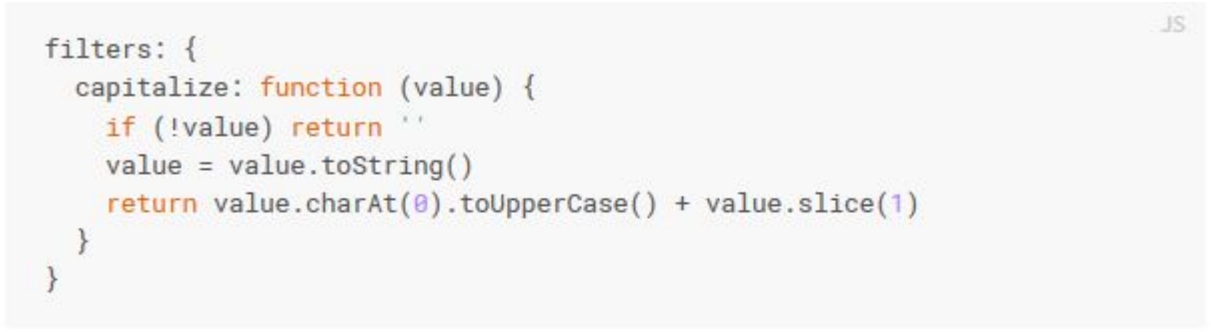
Usage:

1. Used to apply common text formatting
2. Mustache interpolations and v-bind expressions
3. Can be chained
4. Can take arguments



```html
<!-- in mustaches -->
{{ message | capitalize }}

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



```js
filters: {
  capitalize: function (value) {
    if (!value) return ''
    value = value.toString()
    return value.charAt(0).toUpperCase() + value.slice(1)
  }
}
```