Web Application Development

2019

Vue.js

Part III

Components

Form Input Bindings

v-model

v-model

You can use the `v-model` directive to create two-way data bindings on form input, textarea, and select elements.

It automatically picks the correct way to update the element based on the input type.

```
<input v-model="message" placeholder="edit me">

<p>Message is: {{ message }}</p>
```

v-model

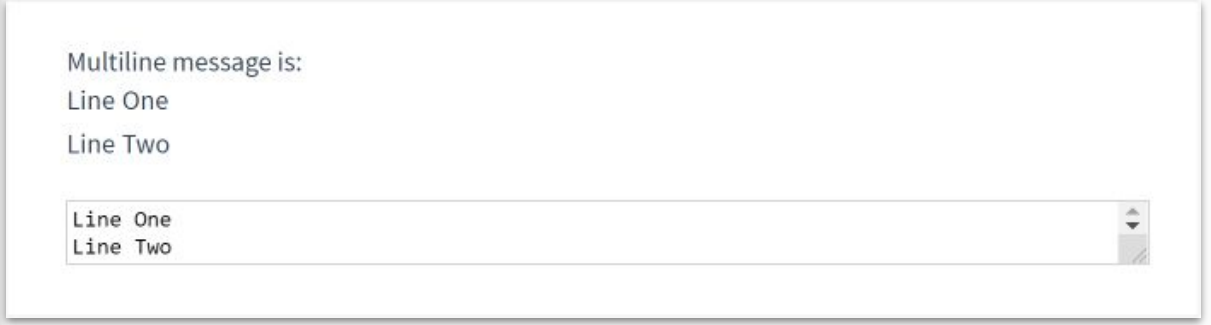
`v-model` internally uses different properties and emits different events for different input elements:

- text and textarea elements use `value` property and `input` event

```
<span>Multiline message is:</span>

<p style="white-space: pre-line;">{{ message }}</p>

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



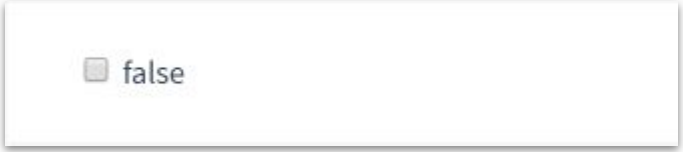
v-model

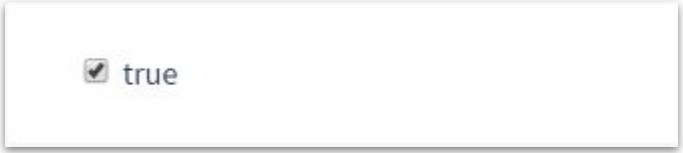
`v-model` internally uses different properties and emits different events for different input elements:

- checkboxes and radio buttons use `checked` property and `change` event

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

<label for="checkbox">{{ checked }}</label>
```





`v-model` will ignore the initial `value`, `checked` or `selected` attributes found on any form elements. It will always treat the Vue instance data as the source of truth. You should declare the initial value on the JavaScript side, inside the `data` option of your component.

v-model

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

- checkboxes and radio buttons use `checked` property and `change` event

```
<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>
      <span>Checked names: {{ checkedNames }}</span>
</div>
```

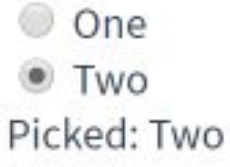
```
new Vue({
  el: '#example-3',
  data: {
    checkedNames: []
  }
})
```

v-model

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

- checkboxes and radio buttons use `checked` property and `change` event

```
<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>
```



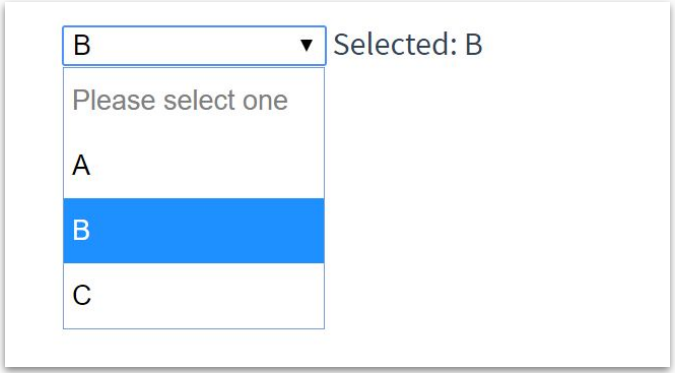
v-model

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

- select fields use `value` as a prop and `change` as an event.

```
<select v-model="selected">
      <option disabled value="">Please select one</option>
      <option>A</option>
      <option>B</option>
      <option>C</option>
</select>
<span>Selected: {{ selected }}</span>
```

```
new Vue({
  el: '...',
  data: {
    selected: ''
  }
})
```



Filters

Filters

Vue.js allows you to define filters that can be used to apply common text formatting.

Filters are usable in two places: mustache interpolations and `v-bind` expressions.

Filters should be appended to the end of the JavaScript expression, denoted by the “pipe” symbol:

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

<!-- in v-bind -->
<div v-bind:message="message | capitalize"></div>
```

Filters

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

Filters

The filter's function always receives the expression's value (the result of the former chain) as its first argument. In the above example, the `capitalize` filter function will receive the value of `message` as its argument.

Filters can be chained:

```
{{ message | filterA | filterB }}
```

Filters are JavaScript functions, therefore they can take arguments:

```
{{ message | filterA('arg1', arg2) }}
```

computed

computed

```
var vm = new Vue({
  el: '#example',
  data: {
    message: 'Hello'
  },
  computed: {
    // a computed getter
    reversedMessage: function () {
      // `this` points to the vm instance
      return this.message.split('').reverse().join('')
    }
  }
})
```

```
<div id="example">
  <p>Original message: "{{ message }}"</p>
  <p>Reversed message: "{{ reversedMessage }}"</p>
</div>
```

Original message: "Hello"

Reversed message: "olleH"

computed vs method

```
var vm = new Vue({
  el: '#example',
  data: {
    message: 'Hello'
  },
  methods: {
    reverseMessage: function () {
      return this.message.split('').reverse().join('')
    }
  }
})
```

```
<div id="example">
  <p>Original message: "{{ message }}"</p>
  <p>Reversed message: "{{ reversedMessage() }}"</p>
</div>
```

Original message: "Hello"

Reversed message: "olleH"

References

https://vuejs.org/v2/api/

Questions?

Next: Backend - Node.js I