Form Input Bindings

Watch a free video lesson on Vue School

When dealing with forms on the frontend, we often need to sync the state of form input elements with corresponding state in JavaScript. It can be cumbersome to manually wire up value bindings and change event listeners:

```template
<input
  :value="text"
  @input="event => text = event.target.value">
```

The `v-model` directive helps us simplify the above to:

```template
<input v-model="text">
```

In addition, `v-model` can be used on inputs of different types, `<textarea>`, and `<select>` elements. It automatically expands to different DOM property and event pairs based on the element it is used on:

- `<input>` with text types and `<textarea>` elements use `value` property and `input` event;
- `<input type="checkbox">` and `<input type="radio">` use `checked` property and `change` event;
- `<select>` uses `value` as a prop and `change` as an event.

Note

`v-model` will ignore the initial `value`, `checked` or `selected` attributes found on any form elements. It will always treat the current bound JavaScript state as the source of truth. You should declare the initial value on the JavaScript side, using reactivity APIs.

Basic Usage

Text

```
<p>Message is: {{ message }}</p>
<input v-model="message" placeholder="edit me" />
```

Message is:

edit me

Try it in the Playground

Note

For languages that require an IME (Chinese, Japanese, Korean, etc.), you'll notice that `v-model` doesn't get updated during IME composition. If you want to respond to these updates as well, use your own `input` event listener and `value` binding instead of using `v-model`.

Multiline Text

```
<span>Multiline message is:</span>
<p style="white-space: pre-line;">{{ message }}</p>
<textarea v-model="message" placeholder="add multiple lines"></textarea>
```

Multiline message is:

add multiple lines

Try it in the Playground

Note that interpolation inside `<textarea>` won't work. Use `v-model` instead.

```
<!-- bad -->
<textarea>{{ text }}</textarea>
```

Checkbox

Single checkbox, boolean value:

```template
<input type="checkbox" id="checkbox" v-model="checked" />
<label for="checkbox">{{ checked }}</label>
```

☐ false

Try it in the Playground

We can also bind multiple checkboxes to the same array or Set value:

```js
const checkedNames = ref([])
```

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

<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>
```

Checked names: []

☐ Jack ☐ John ☐ Mike

In this case, the `checkedNames` array will always contain the values from the currently checked boxes.

Try it in the Playground

Radio

```template
<div>Picked: {{ picked }}</div>

<input type="radio" id="one" value="One" v-model="picked" />
```

```
<input type="radio" id="two" value="Two" v-model="picked" />
<label for="two">Two</label>
```

Picked:

○ One ○ Two

Try it in the Playground

Select

Single select:

```template
<div>Selected: {{ selected }}</div>

<select v-model="selected">
  <option disabled value="">Please select one</option>
  <option>A</option>
  <option>B</option>
  <option>C</option>
</select>
```

Selected:

Please select one

Try it in the Playground

> **Note**
>
> If the initial value of your `v-model` expression does not match any of the options, the `<select>` element will render in an "unselected" state. On iOS this will cause the user not being able to select the first item because iOS does not fire a change event in this case. It is therefore recommended to provide a disabled option with an empty value, as demonstrated in the example above.

Multiple select (bound to array):

```template
<div>Selected: {{ selected }}</div>

<select v-model="selected" multiple>
  <option>A</option>
  <option>B</option>
```

Selected: []

A
B
C

Try it in the Playground

Select options can be dynamically rendered with `v-for`:

```js
const selected = ref('A')

const options = ref([
  { text: 'One', value: 'A' },
  { text: 'Two', value: 'B' },
  { text: 'Three', value: 'C' }
])
```

```template
<div>Selected: {{ selected }}</div>

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

Selected: A

One

Try it in the Playground

Value Bindings

For radio, checkbox and select options, the `v-model` binding values are usually static strings (or booleans for checkbox):

```template
<!-- `picked` is a string "a" when checked -->
<input type="radio" v-model="picked" value="a" />
```

```
<input type="checkbox" v-model="toggle" />

<!-- `selected` is a string "abc" when the first option is selected -->
<select v-model="selected">
  <option value="abc">ABC</option>
</select>
```

But sometimes we may want to bind the value to a dynamic property on the current active instance. We can use `v-bind` to achieve that. In addition, using `v-bind` allows us to bind the input value to non-string values.

Checkbox

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

`true-value` and `false-value` are Vue-specific attributes that only work with `v-model`. Here the `toggle` property's value will be set to `'yes'` when the box is checked, and set to `'no'` when unchecked. You can also bind them to dynamic values using `v-bind`:

```template
<input
  type="checkbox"
  v-model="toggle"
  :true-value="dynamicTrueValue"
  :false-value="dynamicFalseValue" />
```

Tip

The `true-value` and `false-value` attributes don't affect the input's `value` attribute, because browsers don't include unchecked boxes in form submissions. To guarantee that one of two values is submitted in a form (e.g. "yes" or "no"), use radio inputs instead.

Radio

```template
<input type="radio" v-model="pick" :value="first" />
<input type="radio" v-model="pick" :value="second" />
```

Select Options

```template
<select v-model="selected">
  <!-- inline object literal -->
  <option :value="{ number: 123 }">123</option>
</select>
```

`v-model` supports value bindings of non-string values as well! In the above example, when the option is selected, `selected` will be set to the object literal value of `{ number: 123 }`.

Modifiers

`.lazy`

By default, `v-model` syncs the input with the data after each `input` event (with the exception of IME composition as stated above). You can add the `lazy` modifier to instead sync after `change` events:

```template
<!-- synced after "change" instead of "input" -->
<input v-model.lazy="msg" />
```

`.number`

If you want user input to be automatically typecast as a number, you can add the `number` modifier to your `v-model` managed inputs:

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

If the value cannot be parsed with `parseFloat()`, then the original (string) value is used instead. In particular, if the input is empty (for instance after the user clearing the input field), an empty string is returned. This behavior differs from the DOM property `valueAsNumber`.

The `number` modifier is applied automatically if the input has `type="number"`.

`.trim`

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

`v-model` with Components

> If you're not yet familiar with Vue's components, you can skip this for now.

HTML's built-in input types won't always meet your needs. Fortunately, Vue components allow you to build reusable inputs with completely customized behavior. These inputs even work with `v-model`! To learn more, read about Usage with `v-model` in the Components guide.

Edit this page on GitHub

Previous
Event Handling

Next
Watchers