Event Handling

Watch a free video lesson on Vue School

Listening to Events

We can use the `v-on` directive, which we typically shorten to the `@` symbol, to listen to DOM events and run some JavaScript when they're triggered. The usage would be `v-on:click="handler"` or with the shortcut, `@click="handler"`.

The handler value can be one of the following:

1. **Inline handlers:** Inline JavaScript to be executed when the event is triggered (similar to the native `onclick` attribute).
2. **Method handlers:** A property name or path that points to a method defined on the component.

Inline Handlers

Inline handlers are typically used in simple cases, for example:

```js
const count = ref(0)
```

```template
<button @click="count++">Add 1</button>
<p>Count is: {{ count }}</p>
```

Try it in the Playground

Method Handlers

The logic for many event handlers will be more complex though, and likely isn't feasible with inline handlers. That's why `v-on` can also accept the name or path of a component method you'd like to call.

For example:

```js
const name = ref('Vue.js')

function greet(event) {
  alert(`Hello ${name.value}!`)
  // `event` is the native DOM event
  if (event) {
    alert(event.target.tagName)
  }
}
```

```template
<!-- `greet` is the name of the method defined above -->
<button @click="greet">Greet</button>
```

Try it in the Playground

A method handler automatically receives the native DOM Event object that triggers it - in the example above, we are able to access the element dispatching the event via `event.target`.

See also: Typing Event Handlers TS

Method vs. Inline Detection

The template compiler detects method handlers by checking whether the `v-on` value string is a valid JavaScript identifier or property access path. For example, `foo`, `foo.bar` and `foo['bar']` are treated as method handlers, while `foo()` and `count++` are treated as inline handlers.

Calling Methods in Inline Handlers

Instead of binding directly to a method name, we can also call methods in an inline handler. This allows us to pass the method custom arguments instead of the native event:

```
}
```

```template
<button @click="say('hello')">Say hello</button>
<button @click="say('bye')">Say bye</button>
```

Try it in the Playground

Accessing Event Argument in Inline Handlers

Sometimes we also need to access the original DOM event in an inline handler. You can pass it into a method using the special `$event` variable, or use an inline arrow function:

```template
<!-- using $event special variable -->
<button @click="warn('Form cannot be submitted yet.', $event)">
  Submit
</button>

<!-- using inline arrow function -->
<button @click="(event) => warn('Form cannot be submitted yet.', event)">
  Submit
</button>
```

```js
function warn(message, event) {
  // now we have access to the native event
  if (event) {
    event.preventDefault()
  }
  alert(message)
}
```

Event Modifiers

It is a very common need to call `event.preventDefault()` or `event.stopPropagation()` inside event handlers. Although we can do this easily inside methods, it would be better if the methods can be purely about data logic rather than having to deal with DOM event details.

To address this problem, Vue provides **event modifiers** for `v-on`. Recall that modifiers are directive postfixes denoted by a dot.

- `.stop`

- `.capture`
- `.once`
- `.passive`

```template
<!-- the click event's propagation will be stopped -->
<a @click.stop="doThis"></a>

<!-- the submit event will no longer reload the page -->
<form @submit.prevent="onSubmit"></form>

<!-- modifiers can be chained -->
<a @click.stop.prevent="doThat"></a>

<!-- just the modifier -->
<form @submit.prevent></form>

<!-- only trigger handler if event.target is the element itself -->
<!-- i.e. not from a child element -->
<div @click.self="doThat">...</div>
```

> ⓘ TIP
>
> Order matters when using modifiers because the relevant code is generated in the same order. Therefore using `@click.prevent.self` will prevent **click's default action on the element itself and its children**, while `@click.self.prevent` will only prevent click's default action on the element itself.

The `.capture`, `.once`, and `.passive` modifiers mirror the options of the native `addEventListener` method:

```template
<!-- use capture mode when adding the event listener     -->
<!-- i.e. an event targeting an inner element is handled -->
<!-- here before being handled by that element           -->
<div @click.capture="doThis">...</div>

<!-- the click event will be triggered at most once -->
<a @click.once="doThis"></a>

<!-- the scroll event's default behavior (scrolling) will happen -->
<!-- immediately, instead of waiting for `onScroll` to complete  -->
<!-- in case it contains `event.preventDefault()`                -->
<div @scroll.passive="onScroll">...</div>
```

The `.passive` modifier is typically used with touch event listeners for improving performance on mobile devices.

Do not use `.passive` and `.prevent` together, because `.passive` already indicates to the browser that you *do not* intend to prevent the event's default behavior, and you will likely see a warning from the browser if you do so.

Key Modifiers

When listening for keyboard events, we often need to check for specific keys. Vue allows adding key modifiers for `v-on` or `@` when listening for key events:

```template
<!-- only call `submit` when the `key` is `Enter` -->
<input @keyup.enter="submit" />
```

You can directly use any valid key names exposed via `KeyboardEvent.key` as modifiers by converting them to kebab-case.

```template
<input @keyup.page-down="onPageDown" />
```

In the above example, the handler will only be called if `$event.key` is equal to `'PageDown'`.

Key Aliases

Vue provides aliases for the most commonly used keys:

- `.enter`
- `.tab`
- `.delete` (captures both "Delete" and "Backspace" keys)
- `.esc`
- `.space`
- `.up`
- `.down`
- `.left`
- `.right`

System Modifier Keys

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

> ⓘ Note
>
> On Macintosh keyboards, meta is the command key (⌘). On Windows keyboards, meta is the Windows key (⊞). On Sun Microsystems keyboards, meta is marked as a solid diamond (◆). On certain keyboards, specifically MIT and Lisp machine keyboards and successors, such as the Knight keyboard, space-cadet keyboard, meta is labeled “META”. On Symbolics keyboards, meta is labeled “META” or “Meta”.

For example:

```template
<!-- Alt + Enter -->
<input @keyup.alt.enter="clear" />

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

> ⓘ TIP
>
> Note that modifier keys are different from regular keys and when used with `keyup` events, they have to be pressed when the event is emitted. In other words, `keyup.ctrl` will only trigger if you release a key while holding down `ctrl`. It won't trigger if you release the `ctrl` key alone.

`.exact` Modifier

The `.exact` modifier allows control of the exact combination of system modifiers needed to trigger an event.

```template
<!-- this will fire even if Alt or Shift is also pressed -->
<button @click.ctrl="onClick">A</button>

<!-- this will only fire when Ctrl and no other keys are pressed -->
<button @click.ctrl.exact="onCtrlClick">A</button>

<!-- this will only fire when no system modifiers are pressed -->
<button @click.exact="onClick">A</button>
```

Mouse Button Modifiers

- `.left`
- `.right`
- `.middle`

These modifiers restrict the handler to events triggered by a specific mouse button.

Note, however, that `.left`, `.right`, and `.middle` modifier names are based on the typical right-handed mouse layout, but in fact represent "main", "secondary", and "auxiliary" pointing device event triggers, respectively, and not the actual physical buttons. So that for a left-handed mouse layout the "main" button might physically be the right one but would trigger the `.left` modifier handler. Or a trackpad might trigger the `.left` handler with a one-finger tap, the `.right` handler with a two-finger tap, and the `.middle` handler with a three-finger tap. Similarly, other devices and event sources generating "mouse" events might have trigger modes that are not related to "left" and "right" whatsoever.

Edit this page on GitHub

Previous
List Rendering

Next
Form Input Bindings