Custom Directives

Introduction

In addition to the default set of directives shipped in core (like `v-model` or `v-show`), Vue also allows you to register your own custom directives.

We have introduced two forms of code reuse in Vue: components and composables. Components are the main building blocks, while composables are focused on reusing stateful logic. Custom directives, on the other hand, are mainly intended for reusing logic that involves low-level DOM access on plain elements.

A custom directive is defined as an object containing lifecycle hooks similar to those of a component. The hooks receive the element the directive is bound to. Here is an example of a directive that adds a class to an element when it is inserted into the DOM by Vue:

```vue
<script setup>
// enables v-highlight in templates
const vHighlight = {
  mounted: (el) => {
    el.classList.add('is-highlight')
  }
}
</script>

<template>
  <p v-highlight>This sentence is important!</p>
</template>
```

This sentence is important!

In `<script setup>`, any camelCase variable that starts with the `v` prefix can be used as a custom directive. In the example above, `vHighlight` can be used in the template as `v-highlight`.

```js
export default {
  setup() {
    /*...*/
  },
  directives: {
    // enables v-highlight in template
    highlight: {
      /* ... */
    }
  }
}
```

It is also common to globally register custom directives at the app level:

```js
const app = createApp({})

// make v-highlight usable in all components
app.directive('highlight', {
  /* ... */
})
```

It is possible to type global custom directives by extending the `ComponentCustomProperties` interface from `vue`

More Details: Typing Custom Global Directives TS

When to use custom directives

Custom directives should only be used when the desired functionality can only be achieved via direct DOM manipulation.

A common example of this is a `v-focus` custom directive that brings an element into focus.

```vue
<script setup>
// enables v-focus in templates
const vFocus = {
  mounted: (el) => el.focus()
}
</script>

<template>
  <input v-focus />
</template>
```

Declarative templating with built-in directives such as `v-bind` is recommended when possible because they are more efficient and server-rendering friendly.

Directive Hooks

A directive definition object can provide several hook functions (all optional):

```js
const myDirective = {
  // called before bound element's attributes
  // or event listeners are applied
  created(el, binding, vnode) {
    // see below for details on arguments
  },
  // called right before the element is inserted into the DOM.
  beforeMount(el, binding, vnode) {},
  // called when the bound element's parent component
  // and all its children are mounted.
  mounted(el, binding, vnode) {},
  // called before the parent component is updated
  beforeUpdate(el, binding, vnode, prevVnode) {},
  // called after the parent component and
  // all of its children have updated
  updated(el, binding, vnode, prevVnode) {},
  // called before the parent component is unmounted
  beforeUnmount(el, binding, vnode) {},
  // called when the parent component is unmounted
  unmounted(el, binding, vnode) {}
}
```

Hook Arguments

Directive hooks are passed these arguments:

- `el`: the element the directive is bound to. This can be used to directly manipulate the DOM.
- `binding`: an object containing the following properties.
  - `value`: The value passed to the directive. For example in `v-my-directive="1 + 1"`, the value would be `2`.
  - `oldValue`: The previous value, only available in `beforeUpdate` and `updated`. It is available whether or not the value has changed.
  - `arg`: The argument passed to the directive, if any. For example in `v-my-directive:foo`, the arg would be `"foo"`.

`{ foo: true, bar: true }`.

- `instance`: The instance of the component where the directive is used.
- `dir`: the directive definition object.

- `vnode`: the underlying VNode representing the bound element.
- `prevVnode`: the VNode representing the bound element from the previous render. Only available in the `beforeUpdate` and `updated` hooks.

As an example, consider the following directive usage:

```template
<div v-example:foo.bar="baz">
```

The `binding` argument would be an object in the shape of:

```js
{
  arg: 'foo',
  modifiers: { bar: true },
  value: /* value of `baz` */,
  oldValue: /* value of `baz` from previous update */
}
```

Similar to built-in directives, custom directive arguments can be dynamic. For example:

```template
<div v-example:[arg]="value"></div>
```

Here the directive argument will be reactively updated based on `arg` property in our component state.

> ⓘ Note
>
> Apart from `el`, you should treat these arguments as read-only and never modify them. If you need to share information across hooks, it is recommended to do so through element's dataset.

Function Shorthand

It's common for a custom directive to have the same behavior for `mounted` and `updated`, with no need for the other hooks. In such cases we can define the directive as a function:

```js
app.directive('color', (el, binding) => {
  // this will be called for both `mounted` and `updated`
  el.style.color = binding.value
})
```

Object Literals

If your directive needs multiple values, you can also pass in a JavaScript object literal. Remember, directives can take any valid JavaScript expression.

```template
<div v-demo="{ color: 'white', text: 'hello!' }"></div>
```

```js
app.directive('demo', (el, binding) => {
  console.log(binding.value.color) // => "white"
  console.log(binding.value.text) // => "hello!"
})
```

Usage on Components

> ⚠ **Not recommended**
>
> Using custom directives on components is not recommended. Unexpected behaviour may occur when a component has multiple root nodes.

When used on components, custom directives will always apply to a component's root node, similar to Fallthrough Attributes.

```template
<MyComponent v-demo="test" />
```

```template
<!-- template of MyComponent -->

<div> <!-- v-demo directive will be applied here -->
  <span>My component content</span>
</div>
```

attributes, directives can't be passed to a different element with `v-bind="$attrs"`.

Edit this page on GitHub

Previous
Composables

Next
Plugins