Teleport

Watch a free video lesson on Vue School

`<Teleport>` is a built-in component that allows us to "teleport" a part of a component's template into a DOM node that exists outside the DOM hierarchy of that component.

Basic Usage

Sometimes a part of a component's template belongs to it logically, but from a visual standpoint, it should be displayed somewhere else in the DOM, perhaps even outside of the Vue application.

The most common example of this is when building a full-screen modal. Ideally, we want the code for the modal's button and the modal itself to be written within the same single-file component, since they are both related to the open / close state of the modal. But that means the modal will be rendered alongside the button, deeply nested in the application's DOM hierarchy. This can create some tricky issues when positioning the modal via CSS.

Consider the following HTML structure.

```template
<div class="outer">
  <h3>Vue Teleport Example</h3>
  <div>
    <MyModal />
  </div>
</div>
```

And here is the implementation of `<MyModal>`:

```vue
<script setup>
import { ref } from 'vue'

const open = ref(false)
```

```
<template>
  <button @click="open = true">Open Modal</button>

  <div v-if="open" class="modal">
    <p>Hello from the modal!</p>
    <button @click="open = false">Close</button>
  </div>
</template>

<style scoped>
.modal {
  position: fixed;
  z-index: 999;
  top: 20%;
  left: 50%;
  width: 300px;
  margin-left: -150px;
}
</style>
```

The component contains a `<button>` to trigger the opening of the modal, and a `<div>` with a class of `.modal`, which will contain the modal's content and a button to self-close.

When using this component inside the initial HTML structure, there are a number of potential issues:

- `position: fixed` only places the element relative to the viewport when no ancestor element has `transform`, `perspective` or `filter` property set. If, for example, we intend to animate the ancestor `<div class="outer">` with a CSS transform, it would break the modal layout!

- The modal's `z-index` is constrained by its containing elements. If there is another element that overlaps with `<div class="outer">` and has a higher `z-index`, it would cover our modal.

`<Teleport>` provides a clean way to work around these, by allowing us to break out of the nested DOM structure. Let's modify `<MyModal>` to use `<Teleport>`:

```template
<button @click="open = true">Open Modal</button>

<Teleport to="body">
  <div v-if="open" class="modal">
    <p>Hello from the modal!</p>
    <button @click="open = false">Close</button>
  </div>
</Teleport>
```

The `to` target of `<Teleport>` expects a CSS selector string or an actual DOM node. Here, we are essentially telling Vue to "**teleport** this template fragment **to** the **`body`** tag".

You can click the button below and inspect the `<body>` tag via your browser's devtools:

You can combine `<Teleport>` with `<Transition>` to create animated modals - see Example here.

TIP

The teleport `to` target must be already in the DOM when the `<Teleport>` component is mounted. Ideally, this should be an element outside the entire Vue application. If targeting another element rendered by Vue, you need to make sure that element is mounted before the `<Teleport>`.

Using with Components

`<Teleport>` only alters the rendered DOM structure - it does not affect the logical hierarchy of the components. That is to say, if `<Teleport>` contains a component, that component will remain a logical child of the parent component containing the `<Teleport>`. Props passing and event emitting will continue to work the same way.

This also means that injections from a parent component work as expected, and that the child component will be nested below the parent component in the Vue Devtools, instead of being placed where the actual content moved to.

Disabling Teleport

In some cases, we may want to conditionally disable `<Teleport>`. For example, we may want to render a component as an overlay for desktop, but inline on mobile. `<Teleport>` supports the `disabled` prop which can be dynamically toggled:

```template
<Teleport :disabled="isMobile">
  ...
</Teleport>
```

We could then dynamically update `isMobile`.

Multiple Teleports on the Same Target

A common use case would be a reusable `<Modal>` component, with the potential for multiple instances to be active at the same time. For this kind of scenario, multiple `<Teleport>` components can mount their content to the same target element. The order will be a simple append, with later mounts located after earlier ones, but all within the target element.

Given the following usage:

```template
<Teleport to="#modals">
  <div>A</div>
</Teleport>
<Teleport to="#modals">
  <div>B</div>
</Teleport>
```

The rendered result would be:

```html
<div id="modals">
  <div>A</div>
  <div>B</div>
</div>
```

Deferred Teleport 3.5+

In Vue 3.5 and above, we can use the `defer` prop to defer the target resolving of a Teleport until other parts of the application have mounted. This allows the Teleport to target a container element that is rendered by Vue, but in a later part of the component tree:

```template
<Teleport defer to="#late-div">...</Teleport>

<!-- somewhere later in the template -->
<div id="late-div"></div>
```

Note that the target element must be rendered in the same mount / update tick with the Teleport - i.e. if the `<div>` is only mounted a second later, the Teleport will still report an error. The defer works similarly to the `mounted` lifecycle hook.

Related

Edit this page on GitHub

Previous
KeepAlive

Next
Suspense