Control DOM Outside Your Vue.js App with portal-vue

alligator.io/vuejs/portal-vue

One of Vue.js' greatest strengths is it's ability to be used to enhance or replace parts of older apps or even static pages. This progressive nature allows Vue to be incrementally adopted or just used to improve a pre-existing app. portal-vue by Linus Borg extends this flexibility by allowing Vue components to be rendered to anywhere in the DOM, outside of their parent components, or even the whole Vue app!

Installation

Install *portal-vue* via. Yarn or NPM.

```
# Yarn
$ yarn add portal-vue
# NPM
$ npm install portal-vue --save
```

Now, enable the *PortalVue* plugin.

main.js

```
import Vue from 'vue';
import PortalVue from 'portal-vue';
import App from 'App.vue';

Vue.use(PortalVue);

new Vue({
  el: '#app',
  render: h => h(App)
});
```

Targeting Components

So, let’s create the source component that creates the original *portal*. It can have it’s own content as well, only stuff inside the *portal* component gets moved.

AComponent.vue

```
<template>
  <div>
    <portal to="other-component">
      <p></p>
    </portal>
    <p>Other stuff stays here.</p>
  </div>
</template>

<script>

export default {
  data() {
    return {
      message: 'I get rendered in OtherComponent!'
    }
  }
}

</script>
```

Now, as long as both *AComponent* and *OtherComponent* are rendered, the content from *AComponent*’s *portal* will end up rendered in *OtherComponent*. You can even have multiple portals in a single component, going different places!

AComponent.vue

```
<template>
  <div>
    <portal-target name="other-component">
    </portal-target>
    <p>I have my own stuff too!</p>
  </div>
</template>
```

Targeting Anywhere in the DOM

With a teeny-tiny change, we can have the portal content output to anywhere in the DOM of the entire webpage!

AComponent.vue

```
<template>
  <div>
    <portal target-el="#place-on-the-page">
      <p></p>
    </portal>
    <p>Other stuff stays here.</p>
  </div>
</template>

<script>

export default {
  data() {
    return {
      message: 'I get rendered in the element with the id #place-on-the-page!'
    }
  }
}

</script>
```

Now, as long as both *AComponent* and *OtherComponent* are rendered, the content from *AComponent*'s *portal* will end up rendered in *OtherComponent*. You can even have multiple portals in a single component, going different places!

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Vue-Portal Example</title>
  </head>
  <body>

    <div id="app">
      ...
    </div>
    <script src="/dist/build.js"></script>


    <section class="something-else">
      <h4>What is going on here! Who let the Vue app out?</h4>


      <div id="place-on-the-page">
      </div>
    </section>
  </body>
</html>
```

Neat, huh?

Options

- You can toggle whether or not content should go “through” the portal with the *disabled* prop. If set to false, it will render the content inside the *portal* component instead of the *portal-target* or *target-el*. This prop is reactive, so you can change it at will.
- If the portal is disabled, you can also add the *slim* prop to avoid adding an extra wrapper element.
- You can also use the *tag* prop to determine which element the *portal* component renders as when *disabled*.

Potential Issues

See an in-depth explanation here.

- *Portal* and *PortalTarget* components can behave oddly, as they are abstract components, so don’t try to manipulate or access them like normal components.

- When using SSR, the *portal-target* component must appear after the *portal* component in the DOM. Otherwise Vue will get confused and re-render the whole app.
- Also when using SSR, you should probably make sure any targetted elements outside the DOM are not actual HTML Elements. Use *custom* (even fake) ones, like <*my-target*>.
- *refs* on portal’ed content are currently asynchronous and aren’t available on the first or second tick of your component. You’ll have to wait a bit using *setTimeout* or call *this.nextTick* twice. It’s probably a good idea to avoid using *refs* on portal’ed content.