Angular CDK Portals

Chaz Gatian Follow
Oct 19, 2017 · 4 min read

The @angular/cdk contains a concept called portals. In this post I'll attempt to explain the concepts of a Portal, and when they should be applied. The example code in this post is referencing *@angular/cdk@2.0.0-beta.12*.

The term Portal was first coined by Ryan Florence during his talk at React.js Conf 2015. If you read Material's documentation on Portals they provide the following description.

> *A* `Portal` *is a piece of UI that can be dynamically rendered to an open slot on the page. The "piece of UI" can be either a* `Component` *or a* `TemplateRef` *and the "open slot" is a* `PortalHost`.

So wait, Angular can't render a dynamic piece of content anywhere in the page? Actually, it can. Take the following component:

```
@Component({
    selector: 'my-header',
    template: `<h1>{{ title }}</h1>`
})
export class HeaderComponent {
  title = 'Hello World';

  constructor() {
    setTimeout(_ => {
```

If we wanted to render this component in a DOM element running outside the Angular context, this is how we could accomplish this:

```
@Component({
    selector: 'app-root',
    template: ``
})
export class AppComponent implements OnInit {
    private componentRef;

    constructor(
     private componentFactoryResolver: ComponentFactor
     private injector: Injector
    ) { }

    ngOnInit() {
        // Locate an element that exists on the page
        const headerElement = document.querySelector('
        // Locate the component factory for the Header
```

Live Demo

First we locate a Node in the DOM to insert the component, then leverage the ComponentFactoryResolver to generate a new instance of the component. If you're unfamiliar with this technique Maxim Koretskyi has a great article on this topic. Also checkout the Angular documentation.

Once the component is created we need to add the component to the application's component tree.

The **downside** to this approach is we've only handled one simple case. What if we wanted to instead create an embedded template instead of a component? What if we wanted to provide a different ViewContainerRef?

. . .

Portals

Introducing Portals from the Angular cdk. They give you the ability to quickly render content into other areas of your page with little overhead, with a lot of flexibility.

There are two pieces which make up Portals concept. The first is the `PortalHost`. Think of this as a placeholder a component or template will be inserted into. Secondly, is the `Portal`. This is a component

(`ComponentPortal`), or an embedded template (`TemplatePortal`) you would like to display. For example, in the previous example the **HeaderComponent** acted as our Portal. It seems a little confusing at first, perhaps if it was named PortalContent the intent would've been much clearer. Regardless...

PortalHost: Location to insert a component or template.

Portal: Component or Embedded Template to render in a PortalHost

Lets update the previous example to make use of Portals.

```
import { DomPortalHost, Portal, ComponentPortal } from

@Component({
  selector: 'my-app',
  template: '',
})
export class AppComponent implements OnInit {
    private portalHost: DomPortalHost;
    private portal: ComponentPortal<HeaderComponent>;

    constructor(
     private componentFactoryResolver: ComponentFactor
     private injector: Injector,
     private appRef: ApplicationRef,
    ) { }

    ngOnInit() {
        // Create a portalHost from a DOM element
        this.portalHost = new DomPortalHost(
          document.querySelector('#pageHeader'),
          this.componentFactoryResolver,
```

Live Preview

Run the live preview and you'll see the **HeaderComponent** update with "Updated!" after five seconds. Lets quickly run though the changes needed.

- Create a `DomPortalHost`, this class takes the target DOM element as its location

- Create a new `ComponentPortal`
- Attach the `PortalHost` to the `Portal`

At first glance this might not seem as that much of an improvement. But behind the scenes the Angular cdk is doing a lot of work for you. In the example code we only touched on creating components and inserting them into the DOM. Lets take a quick look at what that could would look like if we were to insert an embedded template.

```
@Component({
    selector: 'my-app',
    template: `
        <ng-template #testTemplate let-name>
          <div>User {{ name }} </div>
        </ng-template>`
})
export class AppComponent implements OnInit {

private componentRef;
    @ViewChild('testTemplate') testTemplate: TemplateR

    constructor(
     private injector: Injector,
     private viewContainerRef: ViewContainerRef,
    ) { }

    ngOnInit() {
        // Locate an element that exists on the page
```

Live Demo

You can see the code is quite similar, however we need to take a very different approach to creating the embedded template. First we need to create an `viewRef` from a `ViewContainerRef` , and then since embedded templates are positioned as siblings to their ViewContainer we need to move the viewRef into the correct location. Now lets create a `TemplatePortal` and see the difference.

```
@Component({
  selector: 'my-app',
  template: `
        <ng-template #testTemplate let-name>
          <div>User {{ name }} </div>
        </ng-template>`,
})
export class AppComponent implements OnInit {
    @ViewChild('testTemplate') testTemplate: TemplateR
    private portalHost: DomPortalHost;

    constructor(
     private componentFactoryResolver: ComponentFactor
     private injector: Injector,
     private appRef: ApplicationRef,
     private viewContainerRef: ViewContainerRef,
    ) { }

    ngOnInit() {
        // Create a portalHost from a DOM element
        this.portalHost = new DomPortalHost(
          document.querySelector('#pageHeader'),
          this.componentFactoryResolver,
          this.appRef,
          this.injector
```

Live Preview

The only new code introduced here is the `TemplatePortal` instantiation. The attachment setup is identical to the `ComponentPortal`. Which also means you can use `ComponentPortals` and `TemplatePortals` interchangeably!

Portal Helper Directives

Once you begin using Portals more throughout your code base, you may want to be aware of these helper directives. These basically eliminate some of the boilerplate code within the component. In the previous example I created a `TemplatePortal` in code, but as an alternative, you can generate these through directives.

TemplatePortalDirective

Turn a embedded template into a `TemplatePortal` instance.

```
<ng-template cdkPortal #testTemplate="cdkPortal">
   <div>User: {{ name }} </div>
</ng-template
```

```
@Component({
  selector: 'my-app',
  template: `
        <ng-template cdkPortal #testTemplate="cdkPorta
          <div>User {{ name }} </div>
        </ng-template>`,
})
export class AppComponent implements OnInit {
    @ViewChild('testTemplate') testTemplatePortal: Tem
```

PortalHostDirective

Quickly create a `PortalHost` within your template for quick binding to a component or template. In my examples I used the more extravagant `DomPortalHost` which takes a DOM node. But if you have a clearly defined location in a Component template you can quickly make a `PortalHost` .

```
@Component({
    selector: 'app-test',
    template: `
        <ng-content [cdkPortalHost]="_portalInstance">
    `
})
export class TestComponent implements OnInit {
    _portalInstance: Portal<any>;

    constructor() { }
```

Portals or ngComponentOutlet?

While I was explaining Portals to a colleague of mine they mentioned, “Isn’t this the same thing as a `ngComponentOutlet` ”? And he’s partially correct. While `ngComponentOutlet` and `ngTemplate` outlets allow you to display content dynamically they can only be used within a component template.

· · ·

Additional Portal Use Cases

So now that we know what Portals are, where might one actually use this type of functionality in their projects?

Creating Tooltips

Below is an example of how one might go about creating tooltips.

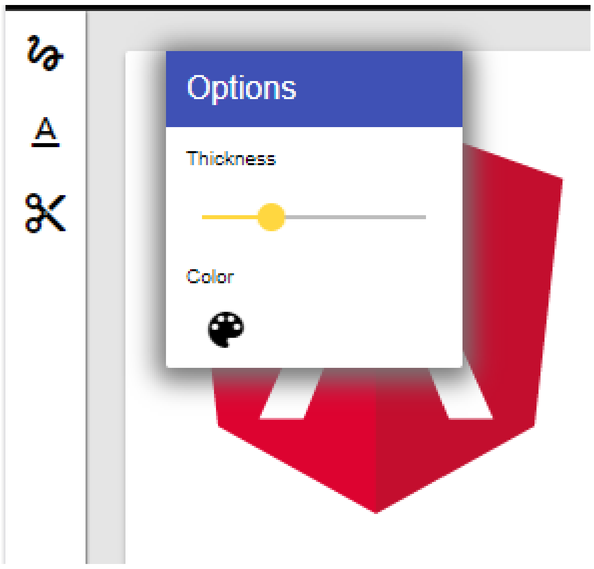
StackBlitz Live Demo

Demo creating a tooltip

stackblitz.com

Dynamically Display Content From A Child Component

Tooltips and dialogs are great use cases, but I think the most untapped usage of a Portal comes from injecting content from a component into other defined locations of the page. Take the following example.



Options content is displayed dynamically

The toolbar on the left contains a set of tools. When a tool is selected, the tool component fires an event passing a **TemplatePortal** instance. The `app-tool-options` component contains a **PortalHost** that's bound to the emitted portal.

StackBlitz Live Demo

stackblitz.com

Alex Rickabaugh recently gave a talk at Angular Mix outlining how you could dynamically render content in a left-side nav. While he doesn't show how to use Portals directly, the sample code he works through gives a more detailed view into what Portals are doing behind the scenes.

. . .

Hope you enjoyed this post. Leave a comment if you have an interesting use case for Portals. Follow me on Twitter @cgatian.

Note: In a future version of Material, PortalHosts will be renamed to PortalOutlets. Tracked by this issue.