@Host Decorator in Angular

2 Comments / 5 minutes of reading / March 9, 2023

← Self, SkipSelf, Optional | Angular Tutorial | View Providers →

@Host decorator is one of the Resolution Modifiers in Angular. The Others are @Self, SkipSelf & Optional. These Decorators configures how the DI Framework resolves the dependencies. We already covered @Self, @SkipSelf & @Optional Decorators. In this tutorial, we will learn about `@Host` in detail.

Table of Contents

@Host
@Host Example
Parent & Child Components with @Host Decorator
Directives & @Host Decorator
Multiple Directives
Content Projection & @Host Decorator
Reference

@Host

The definition of @Host from the Angular Docs

> *Parameter decorator on a view-provider parameter of a class constructor that tells the DI framework to resolve the view by*

> *checking injectors of child elements, and stop when reaching the host element of the current component.*

The definition is a little confusing. So let us try to simplify it a bit.

1. The `@Host` property searches for the dependency inside the **component's template only**.
2. It starts with the current Injector and continues to search in the Injector hierarchy **until it reaches the host element** of the **current component**.
3. It **does not search** for the dependency in the `Providers` of the **host element**.
4. But **it does search** in the `ViewProviders` of the **host element**.
5. Module injector is never searched in the case of `@Host` flag

It looks somewhat similar to **@Self** . But `@Self` only checks in the Injector of the **current component**.

But the `@Host` checks for the dependency in the **current template**.

@Host Example

Let us look at the Angular Project from StackBlitz. It is the same Project from in the tutorial @Self, @SkipSelf & @Optional Decorators.

It has three Components. `AppComponent` , `ChildComponent` & `GrandChildComponent` . One `testDirective` directive. And `RandomService` , which generates the Random number, when instantiated.

All the components & directives display the random number from the `RandomService` .

Make the Providers empty in all components & directives so that `RandomService` is always provided by the `AppModule`. Run the App and you should see the same number displayed by all the components & directives. Also, you will `RandomService Constructed` message only once in the console window indicating that a single instance of the service is created.

Parent & Child Components with @Host Decorator

In the template of ChildComponent, we have added `GrandChildComponent`.

child.component.ts

```
1
2  @Component({
3    selector: "my-child",
4    providers: [],
5    viewProviders: [],
6    template: `
7      <div class="box">
8        <p>ChildComponent => {{ randomNo }}</p>
9
10       <my-grandChild></my-grandChild>
11     </div>
12   `
13 })
14 export class ChildComponent {
15
```

Now, go to the `GrandChildComponent` and add the `@Host` decorator on the `randomService`

```
1
2  export class GrandChildComponent {
3    randomNo;
4    constructor(@Host() private randomService: RandomService) {
5      this.randomNo = randomService?.RandomNo;
6    }
7  }
8
```

Immidetaly you will see the message.

No provider for RandomService found in NodeInjector

The GrandChildComponent is part of the following template of ChildComponent. This Template is hosted inside the ChildComponent. Hence ChildComponent is the host component.

```
<div class="box">
  <p>ChildComponent => {{ randomNo }}</p>

  <my-grandChild></my-grandChild>
</div>
```

@Host first looks for the Dependency in the template. It starts with the Providers of the GrandChildComponent.

Next, it will move to ChildComponent, which is the Host Component. Here it will only look in the ViewProviders and not in Providers array.

Hence there are two ways in which you can remove this error

Add the RandomService to the Providers of the GrandChildComponent. Because that is where DI looks first for dependency.

The second option is to add RandomService to viewProviders array of the ChildComponent.

Directives & @Host Decorator

The Components are nothing but Directives with a View. Hence Directives works in the same way

Goto the `testDirective` and add the `@host` decorator on `randomService`. The error immediately shows up in the console window. *No provider for* `RandomService` *found in* `NodeInjector`.

```
1
2 export class testDirective implements OnInit {
3   constructor(
4     private el: ElementRef,
5     @Host() private randomService: RandomService
6   ) {}
7
```

The `testDirective` is part of this template from `GrandChildComponent`. The Host of the template is `GrandChildComponent`.

```
1
2   template: `
3     <div class="box">
4       GrandChildComponent => {{ randomNo }}
5
6       <div class="dirbox" testDirective>fdf</div>
7
8     </div>
9   `,
10
```

@Host will look for the Dependency in this order

1. Providers array of `testDirective`
2. `ViewProviders` of the `GrandChildComponent`

Hence adding the RandomService in any of those places will resolve the error.

Multiple Directives

Consider the following StackBlitz example. Here we have three directives. All of them has a dependency on randomService .

Now go to cDirective and add @Host on RandomService .

To understand how @Host resolves the Dependency, take a look at the template, where we have used cDirective

The template from the ChildComponent (Hence our Host Component). The aDir , bDir & cDir all part of this template. As mentioned earlier @Host only looks at this template while resolving the dependency.

```
1
2   <div class="box">
3     <p>ChildComponent => {{ randomNo }}</p>
4
5     <div aDir>
6       <div bDir>
7         <div cDir></div>
8       </div>
9     </div>
10  </div>
11
```

@Host will look for the Dependency in this order

1. Providers of cDirective
2. Providers of bDirective
3. Providers of aDirective

4. ViewProviders of ChildComponent

Put RandomService in any of those place and error will go away

What if we add @Host in bDirective instead of cDirective. The checking will be done in the following order. cDirective is not checked as it is the child of bDirective

1. Providers of bDirective
2. Providers of aDirective
3. ViewProviders of ChildComponen

Content Projection & @Host Decorator

Now, look at the StackBlitz example. The App has three components (AppComponent , ChildComponent & GrandChildComponent) and a service (RandomService). Here we are making use of Content Projection to project the GrandChildComponent into the ChildComponent .

Open the GrandChildComponent and add the @Host decorator on randomService .

Like in our previous examples, let us go and look at the template where you will find the GrandChildComponent . You will find it in template of the AppComponent .

app.component.ts

```
1
2    <div class="box">
3      <p>AppComponent => {{ randomNo }}</p>
4      <my-child>
5        <my-grandChild></my-grandChild>
6      </my-child>
7    </div>
8
```

Hence the above is our template and the host is `AppComponent`.

`@Host` will look for the Dependency in this order

1. Providers of `GrandChildComponent`
2. Providers of `ChildComponent`
3. ViewProviders of `AppComponent`

Reference

1. A curious case of the @Host decorator and Element Injectors in Angular
2. @Host API

Read More

1. Angular Tutorial
2. Services
3. Dependency injection
4. Injector, @Injectable & @Inject
5. Providers
6. Injection Token
7. Hierarchical Dependency Injection
8. Angular Singleton Service
9. ProvidedIn root, any & platform
10. @Self, @SkipSelf & @Optional Decorators
11. @Host Decorator in Angular

← Self, SkipSelf, Optional | Angular Tutorial | View Providers →