ViewProviders in Angular

1 Comment / 4 minutes of reading / March 9, 2023

← Host Decorator | Angular Tutorial | Validation in Reactive Forms →

ViewProviders are similar to Providers except that the dependencies that you define are visible only to its view children. They are not visible to the Content children.

Table of Contents

ViewProviders

ViewProviders defines the set of injectable services that are visible only to its view DOM children. These services are not visible to views projected via content projection.

We can insert a component (child component) inside another Angular component (Parent Component). The View from that child component is called a View Child (or View Children's) of the Parent Component.

Angular Components also allow us to insert (or Project) HTML Content provided by another component. We use the element `<ng-content></ng-content>` to create a placeholder for the external content. The view created from such a component is known as Content Child (or Content Children).

Providers Vs ViewProviders

The Services declared in Components Providers are injected to both view children and content children.

But the Services declared in ViewProviders are injected only to view Children.

ViewProviders metadata is available only to Angular Components. Providers are available in NgModule, Component, Pipes, Directives, etc

ViewProviders Example

Look at the example app at stackblitz

The app has `RandomService`, which generates a random number when initialized.

random-service.ts

```
1
2  import { Injectable } from "@angular/core";
3
4  @Injectable({
5    providedIn: "root"
6  })
7  export class RandomService {
8    private _randomNo = 0;
9
10   constructor() {
11     console.log("RandomService Constructed");
12     this._randomNo = Math.floor(Math.random() * 1000);
```

```
  }

  get RandomNo() {
    return this._randomNo;
  }
}


```

ChildComponent displays the random no from the RandomService .

It also has ng-content, where the parent can inject content. The Parent component is going to inject the GrandChildComponent here.

ChildComponent also displays the GrandChildComponent as View

child.component.ts

```

import { Component, SkipSelf, Self, Optional, Host } from '@angular/core';
import { RandomService } from './random-service';

@Component({
  selector: 'my-child',
  providers: [],
  viewProviders: [],
  template: `
    <div class="box">
      <p>ChildComponent => {{ randomNo }}</p>
      <ng-content> </ng-content>

      <strong>View Child</strong>
      <my-grandChild></my-grandChild>
    </div>
  `
})
export class ChildComponent {
  randomNo;
  constructor(private randomService: RandomService) {
    this.randomNo = randomService.RandomNo;
  }
}

```

```
26
```

GrandChildComponent just displays the random no from the RandomService . We use @Optional() decorator ensures no error is thrown if the dependency is not found.

grand-child.component.ts

```
1
2  import { Component, SkipSelf, Self, Optional, Host } from "@angular/core";
3  import { RandomService } from "./random-service";
4
5  @Component({
6    selector: "my-grandChild",
7    template: `
8      <div class="box">
9        GrandChildComponent => {{ randomNo }}
10     </div>
11   `,
12   providers: [],
13   viewProviders: [],
14 })
15 export class GrandChildComponent {
16   randomNo;
17   constructor(@Optional() private randomService: RandomService) {
18     this.randomNo = randomService?.RandomNo;
19   }
20 }
21
22
```

In the AppComponent , we display the ChildComponent . We also project the GrandChildComponent inside the ChildComponent using projected content.

app.component.ts

```
1
2  import { Component, VERSION } from '@angular/core';
3  import { RandomService } from './random-service';
4
5  @Component({
6    selector: 'my-app',
7    providers: [],
8    viewProviders: [],
```

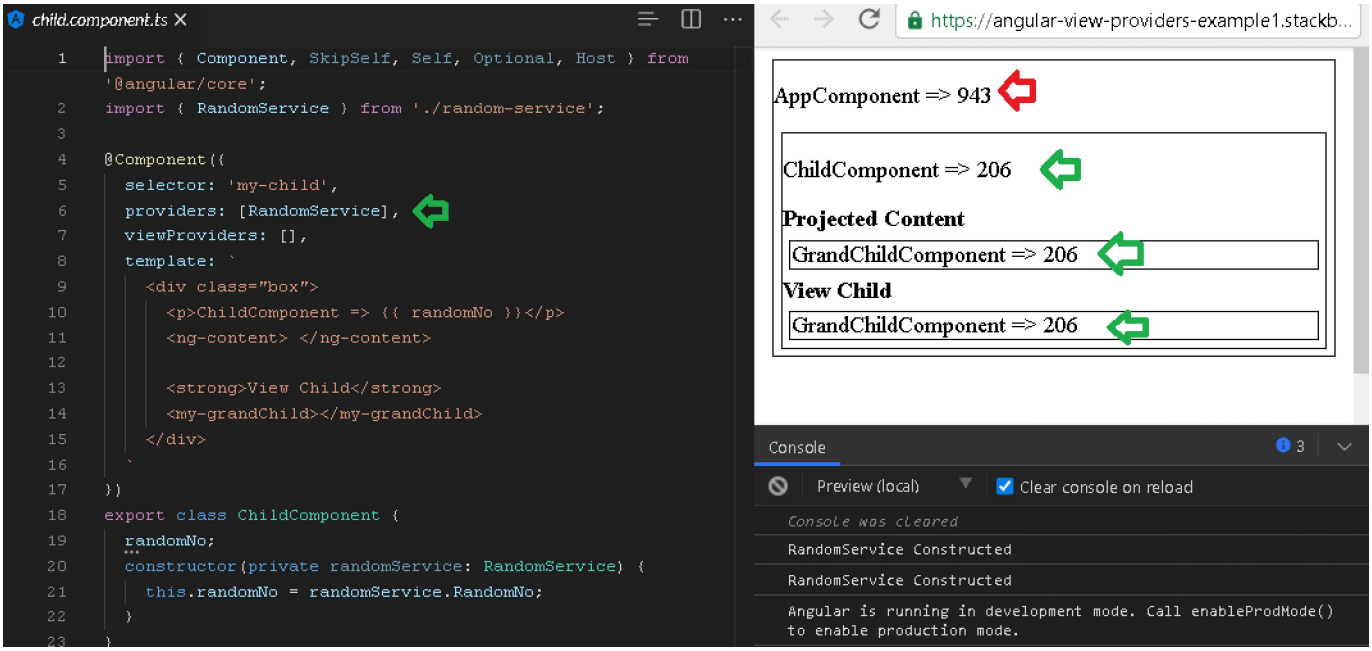
```
  template: `
    <div class="box">
    <div class="box">
      <p>AppComponent => {{ randomNo }}</p>
      <my-child>
        <strong>Projected Content</strong>
        <my-grandChild></my-grandChild>
      </my-child>
    </div>
  `
})
export class AppComponent {
  randomNo;
  constructor(private randomService: RandomService) {
    this.randomNo = randomService.RandomNo;
  }
}

```

Run the app and you will see all the components receive the same value for the random no. Because the RandomService is provided from the root module injector.

Also, note that ChildComponent displays the `GrandChildComponent` twice. Once as a Content Child (Projected Content) and also a View Child.

Add the `RandomService` to the Providers array of the `ChildComponent`. As you can see from the image below, `ChildComponent` and all its children (content children and view children) get the instance of `RandomService` provided by the `ChildComponent`.



Now, move the RandomService from Providers to ViewProviders. As you can see in the image below View Child still get the RandomService from ChildComponent, but the Projected Content does not. Projected Content gets the service from the Root Module.

The `providers` allow all children to use the services. While the `viewProviders` provides services to all its children other than projected content.

The use case for ViewProviders

This is useful when you develop libraries.

For Example, you have made `some-great-comp` , which users will use in their `user-component` into it.

```
<some-great-comp>
  <user-component></user-component>
</some-great-comp>
```

Here you do not want services that you used in your `some-great-comp` interfere with the `user-component` . Hence you provide your services in the `ViewProviders` .

Reference

1. Manual

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
12. ViewProviders