Angular - @Input and @Output vs. Injectable Service

Asked 1 year, 2 months ago Active 1 year, 2 months ago Viewed 3k times

4

2

I'm asking myself where are the differences between `@Input` / `@Output` in parent-/childcomponents **and** using services which is only once instatiated with dependency Injection `@Injectable()` . Or are there any differences besides Input/Output can only be used in parent-/child-comp.

Following Example for better visualization:

With @Input:

```
<parent-comp>
   <child-comp [inputFromParent]="valueFromParent"></child-comp>
</parent-comp>
```

ChildComponent:

```
@Component({
  selector: 'child-comp',
  template: ...
})
export class ChildComponent {
  @Input() public inputFromParent: string;
}
```

With dependency Injection

```
@Injectable()
export class Service {
   private value: string;

public get value(): string {
   return value;
}

public set value(input): void {
   value = input;
}

}
```

Now we can set the value in the parent comp. and get the value in the child comp with dependancy injection. ChildComponent:

```
@Component({
  selector: 'child-comp',
  template: ...
})
export class ChildComponent {
  private value: string;
  constructor(private service: Service) {
  this.value = this.service.getValue;
}

}
```

Even though the first approach looks simpler, I recognized using 3-4 properties carriyng through parent-/child comp. with `@Input` / `@Output` makes the tamplete very confusing and clunky.

angular typescript design-patterns dependency-injection

asked Jun 25 '18 at 16:07
MarcoLe
246 1 7 31

using a service is more flexible because it creates a singleton instance of the service using @Injectable() decorator and it can be injected in any component even in another service. – Niladri Jun 25 '18 at 16:13

I would tend to avoid sacrificing correctness for flexibility. – prettyfly Jun 25 '18 at 16:25

1 Answer

15

Not exactly a question with a defined answer, but....

`@Input` and `@Output` are useful if the communication between a parent and child is just that, between a parent and child. It wouldn't make sense to have a service that maintains singleton data for just 2 components (or however deeply nested grandparent -> parent ->

child components are).

They're also useful if your parent needs to react to a change in the child. For example, clicking a button in a child component that calls a function in the parent:

```
<my-child-component (myOutputEmitter)="reactToChildChange($event)"></my-child-component>
```

And in parent:

```
reactToChildChange(data: any) {
  // do something with data
}
```

If you find yourself passing many `@Input` properties to a child, and want to tidy up a template, then you can define an interface for the input, and pass it instead. e.g.

```
export interface MyChildProperties {
   property?: any;
   anotherProperty?: any;
   andAnotherProperty?: any;
}
```

Then you can pass a definition to your child, which is set from the parent:

```
childProperties: MyChildProperties = {
    property: 'foo',
    anotherProperty: 'bar',
    andAnotherProperty: 'zoob'
}
```

Then your child component may have:

```
@Input properties: MyChildProperties;
```

and your template becomes:

```
<my-child-component [properties]="childProperties"></my-child-component>
```

Your child can access those properties from `properties.property` , `properties.anotherProperty` , etc.

Clean, tidy, and your data is now contained to those components that need to communicate.

Services, however, should be used where *more than one component* needs access to read/write data across your entire application. Consider a `UserService` for example, where many different components need to be able to access the currently logged in user. In this case, a service is sensible, as its a singleton, so once you have set your logged in user, any components that inject the `UserService` can access its data and functions.

Similarly, if you were to use a service for reacting to change, then you'd find yourself writing services with observables so that your components could subscribe to changes in the data. Eventemitters already give you this pattern with `@Output` as shown above.

If it were a simple parent -> child communication, this is unnecessary overhead, and should be avoided.

That said, if you find yourself using services to manage global state, you'd be better off using some form of state management such as ngrx

answered Jun 25 '18 at 16:25

prettyfly
1,563 1 12 23

2 Excellent answer. Thank you! – Chaitanya Bangera May 16 at 16:47

Got a question that you can’t ask on public Stack Overflow? Learn more about sharing private information with Stack Overflow for Teams. ✕