Angular Pass data to child component

10 Comments / 7 minutes of reading / May 7, 2021

← Component Communication

Passing Data from child to Parent Component →

In this tutorial, we learn how Angular Passes the data to the child component. The Angular Components communicate with each other using **@Input Decorator**. We also look at how child components detect changes to these Input properties using OnChanges life Cycle hook or with a Property Setter.

Table of Contents

Passing data to a child/nested component

In the previous tutorials, we built an Angular Application and then added a child component to it. We also looked at how Angular Component communicates with its View (templates) using the data binding.

These Components are useless if they are not able to communicate with each other. They need to communicate with each other if they want to serve any useful purpose.

How to Pass data to a child component

In Angular, the Parent Component can communicate with the child component by setting its Property. To do that the Child component must expose its properties to the parent component. The Child Component does this by using the **@Input decorator**

In the Child Component

1. Import the @Input module from `@angular/Core` Library
2. Mark those property, which you need data from the parent as input property using @Input decorator

In the Parent Component

1. Bind the Child component property in the Parent Component when instantiating the Child

@Input Decorator

The @Input Decorator is used to configure the input properties of the component. This decorator as also supports change tracking.

When you mark a property as input property, then the Angular injects values into the component property using Property Binding. The Property Binding uses the [] brackets. The Binding Target (Property of the child component) is placed inside the square brackets. The Binding source is enclosed in quotes. Property binding is one way from Component to the Target in the template

@Input example

Now let us build a simple component to demonstrate the use of @Input.

Our application will have a counter which is incremented by the Parent Component. The Parent Component then passes this counter to the child component for display in its template

You can download the source of this tutorial from the Github. The initial code is available in `GettingStarted` Folder and Final Code in `InputDecorator` folder

The Child Component with @Input Decorator

Create the ChildComponent.ts under the src/app folder. And copy the following code

```
import { Component, Input  } from '@angular/core';

@Component({
    selector: 'child-component',
    template: `<h2>Child Component</h2>
            current count is {{ count }}
    `
})
export class ChildComponent {
    @Input() count: number;
}
```

Now, let us look at the code in detail

First, we import the @Input decorator from @angular/core

```
import { Component, Input  } from '@angular/core';
```

We have defined the inline template for the child component, where it displays the current count.

```
@Component({
    selector: 'child-component',
    template: `<h2>Child Component</h2>
            current count is {{ count }}
    `
})
```

The Child Component expects the count to come from the Parent Component. Hence in ChildComponent decorate the count property with @Input decorator

```
export class ChildComponent {
   @Input() count: number;
}
```

Bind to Child Property in Parent Component

Now, time to pass the Count values to the Child Component from the Parent

Open the app.component.ts and copy the following code

```
import { Component} from '@angular/core';

@Component({
  selector: 'app-root',
  template: `
       <h1>Welcome to {{title}}!</h1>
       <button (click)="increment()">Increment</button>
       <button (click)="decrement()">decrement</button>
       <child-component [count]=Counter></child-component>` ,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Component Interaction';
  Counter = 5;

  increment() {
    this.Counter++;
  }
  decrement() {
    this.Counter--;
  }
}
```

The inline template in the Parent Component has two buttons. The Buttons Increments/decrements the counter.

```
1
2     <button (click)="increment()">Increment</button>
3     <button (click)="decrement()">decrement</button>
4
```

In the next line, we are invoking the child component inside

```
1
2 <child-component [count]=Counter></child-component>
3
```

Here, we are using count property, which is a property of the child Component inside the square bracket. We bind it to Counter property of the Parent Component.

Remember square bracket represents the Property Binding in Angular.

Finally, we will add counter in Parent component and set it to 5 as shown below.

```
1
2 export class AppComponent {
3   title = 'Component Interaction';
```

```
  Counter = 5;

  increment() {
    this.Counter++;
  }
  decrement() {
    this.Counter--;
  }
}

```

That's it.

Now run the Code and you should see the following displayed in the browser

Click on Increment & Decrement buttons to see that the changes are propagated to the child component.

Various ways to use @Input Decorator

We used input @Input Decorator to mark the property in child component as input property. There are two ways you can do it Angular.

1. Using the @Input decorator to decorate the class property
2. Using the input array meta data of the component decorator

Using the @Input decorator to decorate the class property

We saw this in our above example.

```
export class ChildComponent {
    @Input() count: number;
}
```

Using the input array metadata of the component decorator

The same result can be achieved by using Input array of the @Component decorator as shown below

```
@Component({
    selector: 'child-component',
    inputs: ['count'],
    template: `<h2>Child Component</h2>
    current count is {{ count }}
`
})
export class ChildComponent {}
```

We have moved the count property to inputs array of the component metadata.

Aliasing input Property

You can Alias the input property and use the aliased name the parent component as shown below

```

export class ChildComponent {
    @Input('MyCount') count: number;
}

```

Here, we are aliasing count property with MyCount alias

In the parent component, we can use the MyCount as shown below

```

template: `
  <h1>Welcome to {{title}}!</h1>
  <child-component [MyCount]=ClickCounter></child-component>

```

Detecting the Input changes

We looked at how to pass the data from parent to the child using @Input decorator and property binding.

Passing the data to child component is not sufficient, the child Component needs to know when the input changes so that it can act upon it.

There are two ways of detecting when input changes in the child component in Angular

1. Using OnChanges LifeCycle Hook
2. Using Input Setter

Let us look at both the methods in detail

Using OnChanges LifeCycle Hook

ngOnChanges is a lifecycle hook, which angular fires when it detects changes to data-bound input property. This method receives a SimpeChanges object, which contains the current and previous property values. We can Intercept input property changes in the child component using this hook.

How to use ngOnChanges for Change Detection

1. Import the OnChanges interface, SimpleChanges , SimpleChange from @angule/core library.
2. Implement the ngOnChanges() method. The method receives the SimpleChanges object containing the changes in each input property.

Let us update our Child Component to use the OnChanges hook

Open the child.component.ts and make the following changes

```
1
2  import { Component, Input, OnChanges, SimpleChanges, SimpleChange  } from '@angular
3
4  @Component({
5     selector: 'child-component',
6     template: `<h2>Child Component</h2>
7              current count is {{ count }}
8     `
9  })
10 export class ChildComponent implements OnChanges {
11    @Input() count: number;
12
```

```
13    ngOnChanges(changes: SimpleChanges) {
14
15       for (let property in changes) {
16          if (property === 'count') {
17            console.log('Previous:', changes[property].previousValue);
18            console.log('Current:', changes[property].currentValue);
19            console.log('firstChange:', changes[property].firstChange);
20          }
21       }
22    }
23 }
24
```

First, we are Importing the required libraries

```
1
2 import { Component, Input, OnChanges, SimpleChanges, SimpleChange  } from '@angular/
3
```

Next, Modify the class to implement the `Onchanges` interface

```
1
2 export class ChildComponent implements OnChanges {
3
```

`ngOnChanges` method

```
1
2  ngOnChanges(changes: SimpleChanges) {
3
4      for (let property in changes) {
5          if (property === 'count') {
6            console.log('Previous:', changes[property].previousValue);
7            console.log('Current:', changes[property].currentValue);
8            console.log('firstChange:', changes[property].firstChange);
9          }
10     }
11   }
12
```

This method receives all the changes made to the input properties as SimpleChanges object. The SimpleChanges object whose keys are property names and values are instances of SimpleChange.

SimpleChange class Represents a basic change from a previous to a new value. It has three class members.

Property Name	Description
previousValue:any	Previous value of the input property.
currentValue:any	New or current value of the input property.
FirstChange():boolean	Boolean value, which tells us whether it was the first time the change has taken place

And we loop through the SimpleChanges to get our property count

Run the code and open the console log to watch the logs as you click on increment and decrement buttons in the parent component.

Using Input Setter

We can use the property getter and setter to detect the changes made to the input property as shown below

In the Child Component create a private property called _count

```

private _count = 0;

```

Create getter & setter on property count and attach `@Input` Decorator. We intercept the input changes from the setter function and log them to console

```

@Input()
set count(count: number) {
   this._count = count;
   console.log(count);
}
get count(): number { return this._count; }

```

Summary

In this tutorial, we looked at how to pass data from parent to child Component. The Child Component decorates the property using the `@Input` Decorator. In the Parent Component, we use property binding to bind it to the Property or method of Parent Component.

We can also track changes made to the Input Property either by Using hooking to `ngOnChanges` life cycle hook. Or using the Property setter

←

Component Communication