Yallaling Goudar

Communication Between Components Using Input and Output Properties

Yallaling Goudar

Jan 29, 2019 • 7 Min read • 65,794 Views

Jan 29, 2019 • 7 Min read • 65,794 Views

Web Development Angular

Introduction

119

Introduction

Communication between the Components in Angular will help you to pass data from child components to parent components and vice-versa.

- Top ^

Passing Data from Parent to Child Component with Input Binding

When we want to pass the data from the parent component to the child component, we use input binding with @Input decorations.

Let's consider an example where PersonChildComponent has two input properties with @Input decorations. As we can see in the below example, we must import Input from '@angular/core' library.

Filename: personchild.component.ts

typescript

```typescript
import { Component, Input } from '@angular/cor

import { Person } from './Person';

@Component({
  selector: 'app-person-child',
  template: `
    <h3>{{person.name}} says:</h3>
    <p>I, {{person.name}}, welcome to Pluralsi
  `
})
export class PersonChildComponent {
  @Input() person: Person;
  @Input('master') masterName: string;
}
```

We can use aliasing with @Input binding. As we see in the above example, masterName is aliased with the master.

Intercepting Input Property Changes with a Setter and ngOnChanges()

Intercepting input property helps to act upon a value from the parent.

Changes with setter:

Let's consider an example where we are setting the personname of the input property in the child PersonChildComponent that trims the whitespace from a name and replaces an empty value with default text.

The PersonParentComponent below demonstrates name variations in the personname, including a personname with all spaces.

Filename: personchild.component.ts

typescript

```typescript
import { Component, Input } from '@angular/c

@Component({
  selector: 'app-personname-child',
  template: '<h3>"{{personname}}"</h3>'
})
export class PersonChildComponent {
  private _personname = '';

  @Input()
  set personname(personname: string) {
    this._personname = (personname && person
  }

  get personname(): string { return this._pe
}
```

File name: personparent.component.ts

typescript

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

@Component({
  selector: 'app-person-parent',
  template: `
  <h2>Master have {{personnames.length}} persc
  <app-person-child *ngFor="let personname of
  `
})
export class PersonParentComponent {
  // Displays 'Yallaling', '<no person set>',
  personnames = ['Yallaling', '   ', '  Goudar
}
```

Changes with ngOnChanges():

ngOnChanges() method of the OnChanges lifecycle hook interface detects and acts upon changes to input property values. You may prefer this approach to the property setter when watching multiple, interacting input properties.

Let's consider an example where we have MinmaxChildComponent which detects changes to the minimum and maximum input properties and composes a log message reporting these changes.

Filename: minmaxchild.component.ts

typescript

```typescript
import { Component, Input, OnChanges, Simple

@Component({
  selector: 'app-minmax-child',
  template: `
```

```
    <h3>Min value: {{minimum}} Max value: {{
    <h4>Change log:</h4>
    <ul>
      <li *ngFor="let change of changeLog">{
    </ul>
  `
})
export class MinmaxChildComponent implements
  @Input() minimum: number;
  @Input() maximum: number;
  changeLog: string[] = [];

  ngOnChanges(changes: {[propKey: string]: S
    let log: string[] = [];
    for (let propName in changes) {
      let changedProp = changes[propName];
      let to = JSON.stringify(changedProp.cu
      if (changedProp.isFirstChange()) {
        log.push(`Initial value of ${propNam
      } else {
        let from = JSON.stringify(changedPro
        log.push(`${propName} changed from $
      }
    }
    this.changeLog.push(log.join(', '));
  }
}
```

The MinmaxChildComponent supplies the minimum and maximum values and binds buttons to methods that change them.

Filename: minmaxparent.component.ts

typescript

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

@Component({
  selector: 'app-minmax-parent',
  template: `
    <h2>Source code minmax</h2>
    <button (click)="changedMin()">New minimum
    <button (click)="changedMax()">New minmax
    <app-minmax-child [major]="major" [minor]=
  `
```

```
11 })
12 export class MinmaxParentComponent {
13   minimum = 1;
14   maximum = 23;
15
16   changedMin() {
17     this.minimum++;
18   }
19
20   changedMax() {
21     this.maximum++;
22     this.minimum = 0;
23   }
24 }
```

When we click on the button 'New minimum value', the minimum value will get increased and when we click on the button 'New maximum value', the maximum value will get increased. And we can see the changed values getting logged in the changelog.

Passing Data from Child to Parent with Output Binding

An Output is an observable property annotated with an @Output decorator, the property always returns an Angular EventEmitter. Values flow out of the component as events bound with an event binding.

In Angular, a component can emit an event using @Output an EventEmitter. Both are parts of the @angular/core.

Let's consider an example where we are emitting the sum value from the component ExampleChildComponent.

Filename: examplechild.component.ts

typescript

```typescript
import { Component, EventEmitter, Output } frc
@Component({
    selector: 'app-example-child',
    template: `<button class='btn btn-primary'
})
export class ExampleChildComponent {
    @Output() valueChange = new EventEmitter()
    sum = 0;
    changeValue() {
        // You can give any function name
        this.sum = this.sum + 10;
        this.valueChange.emit(this.sum);
    }
}
```

Let's consider an example where we are going to emit an event and pass a parameter to the event. In the below example, we are emitting a value from ExampleChildComponent to ExampleComponent. Displaying the sum value from ExampleChildComponent.

Filename: example.component.ts

typescript

```typescript
import { Component, OnInit } from '@angular/cc
@Component({
    selector: 'app-example',
    template: `<app-example-child (changeValue
})
export class ExampleComponent implements OnIni
    ngOnInit() {
    }
    displaySum(sum) {
        console.log(sum);
```

```
11      }
12  }
```

Conclusion

In this guide, we have explored the Input and Output Property techniques in Angular. We have also seen different methods or ways through which we can pass the values from parent to child component and vice-versa.

You can learn more about Angular binding in my guide Attribute, Class, and Style Bindings in Angular.