Communications

SD 555 – Web Application Development III

Maharishi International University

Department of Computer Science

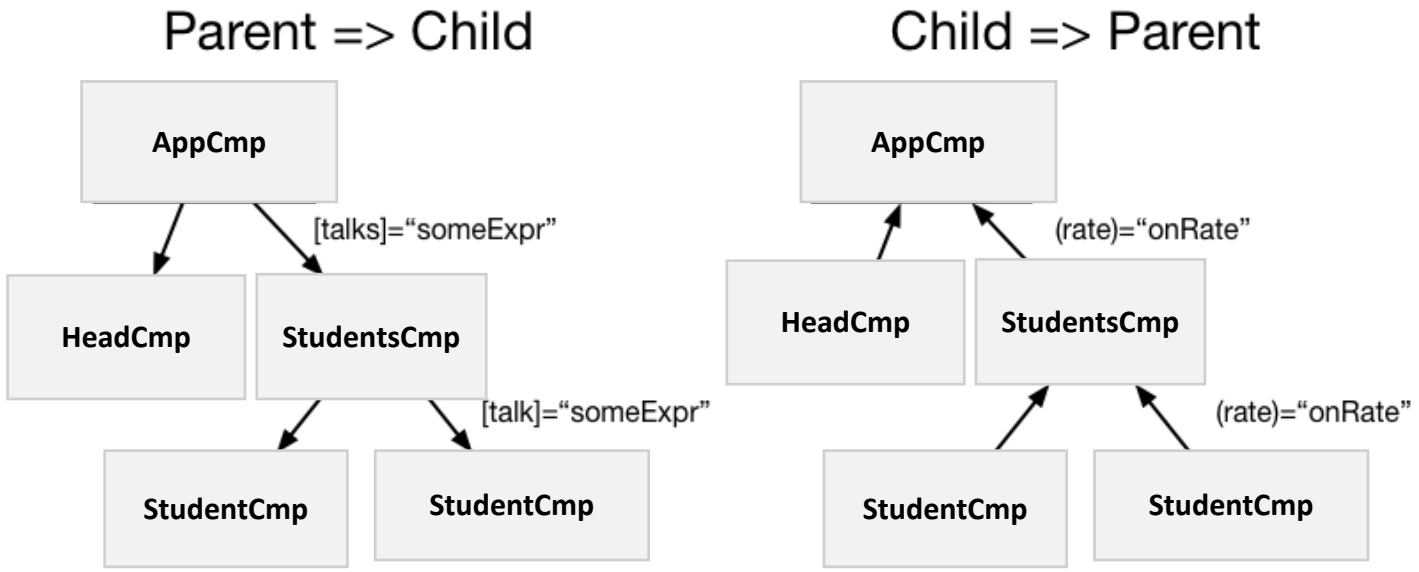
Associate Professor Asaad Saad

Maharishi International University - Fairfield, Iowa

Input and Output Properties

A component has ***input*** and ***output*** properties, which can be defined in the component decorator metadata or using class property decorators.

Data flows into a component via ***input*** properties. Data flows out of a component via ***output*** properties.



Inputs and Outputs

Input and output properties are the public API of a component. You use them when you instantiate a component in your application.

```
<myComponent
    [data]="users" <!-- input -->
    (someEvent)="handleEvent($event)"> <!-- output -->
</myComponent>
```

$event is a special variable that represents what's being emitted.

[squareBrackets] pass inputs: You can set input properties using property bindings
(parens) handle outputs: You can subscribe to output properties using event bindings

HTML markups

All HTML attributes are inputs.

All HTML events are outputs.

For example: <input />

Native input properties: **value**, **class**, **id**, **type**.. etc

Native output properties: **click**, **mouseover**, **input**, **keyup**.. etc

```
<input [value]="user.username" (keyup)="doSomething($event) "/>
```

Passing data to `inputs`

```
@Component({
    template: `
        <input [placeholder]="holderMsg" />
        <input placeholder ="Enter your email" />
        <input [placeholder]="'Enter your email'" />
        <input placeholder="{{holderMsg}}" />
    `,
})
export class ParentComponent {
    holderMsg = "Enter your email";
}
```

General Rule: If you are passing a string, you don't need to use the square brackets [], for any other data type, use the square brackets.

What is the difference between these two lines of code?

```
<component [page]="1" />
<component page="1" />
```

Input Signal

Input Signals are used to pass values from a parent to a child component. They can be optional or required.

```
@Component({
   selector: 'child'
})
export class MyComponent {

     $name = input<string>(''); // InputSignal<string>

}
```

Pass a value from Parent-To-Child

```
@Component({
  selector: 'parent',
  standalone: true,
  imports: [ChildComponent],
  template: `
    <child [$msgFromParent]="msgToChild" />
  `
})
export class AppComponent {

  msgToChild = 'hello son';

}
```

```
@Component({
  selector: 'child',
  standalone: true,
  template: ` {{ $msgFromParent() }} `
})
export class ChildComponent {

  $msgFromParent = input.required();

}
```

ngOnInit Lifecycle Hook

You can read the **`input()`** value starting with **`ngOnInit`** lifecycle hook.

```
@Component({
        selector: 'app-cmp'
})
class AppCmp implements OnInit {
        $message = input();

        ngOnInit() { console.log(this.$message())}
}
```

effect as a Lifecycle Hook

`effect` triggers when a component **`input()`** property is set.

```
@Component({
        selector: 'app-cmp'
})
class AppCmp {
        $id = input();

        constructor(){
           effect(()=>{
                // make http call once the $id signal is set and ready
               console.log($id());
           })
        }
}
```

Component outputs

When we want to send data out from a component to its **direct parent**, we use **output**, and they are usually Events so the parent component could listen to them and add a handler when they emit any value.

```
@Component({
   selector: 'child'
})
export class MyComponent {

    msgFromChild = output<string>(); // OutputEmitterRef<string>

}
```

Pass a value from Child-To-Parent

```
@Component({
  selector: 'parent',
  standalone: true,
  imports: [ChildComponent],
  template: `
    <child (msgFromChild)="receive($event)" />
  `
})
export class AppComponent {

  receive(data: string) {
    console.log('received', data);
  }

}
```

```
@Component({
  selector: 'child',
  standalone: true,
   template: `
    <button (click)="send()">Send to parent</button>`
})
export class ChildComponent {

  msgFromChild = output<string>();

  send() {
    this.msgFromChild.emit('hello father');
  }
}
```

Model Signals

Model signals are input signals and output at the same time, that allow the component author to write values into the property. You can bind a writable signal or a plain JavaScript property to a model input.

```
export class ChildComponent {
  // this automatically creates an input called "name" and an output named "nameChange"
  // set a value from parent using [name] and listen to any changes using (nameChange)="handler()"
  // you can also use the "banana-in-a-box" syntax [(name)] to set and listen to any value changes
  name = model();
}
```

Model Example

```
@Component({
  selector: 'parent',
  standalone: true,
  imports: [ChildComponent],
  template: `
    <p>{{ $data() }}</p>
    <button (click)="sendToChild()">send to child</button>
    <child [($msg)]="data" />
  `
})
export class AppComponent {

  $data = signal('initial value');

  sendToChild() {
    this.$data.set('hello son');
  }

}
```

```
@Component({
  selector: 'child',
  standalone: true,
  template: `
    <button (click)="send()">send to parent</button>
    {{ $msg() }}
  `
})
export class ChildComponent {

  $msg = model<string>();

  send() {
    this.$msg.set('hello father');
  }

}
```