Tuesday:Directives

Directives in Angular

A directive is a function that executes whenever the Angular compiler finds it in the DOM. Directives in Angular are used to extend the power of HTML by giving it a new syntax that actually processes data from HTML. There are two major types of directives in Angular:

1. Attribute Directives.
2. Structural Directives.

Attribute Directives

Attribute directives manipulate the DOM by changing its behaviour and appearance. We use attribute directives to apply conditional styles to elements, show or hide elements, and, dynamically change the behaviour of an element based on a changing property, say, make a form field blink red if a user tries to submit the form without filling in that specific form field.

Structural Directives

Structural directives, on the other hand, are used to create and/or destroy DOM elements. This type of directives actually add or completely remove elements from the DOM unlike some attribute directives like `hidden` which maintain the DOM as it is. We should, therefore, be careful when we decide to use structural directives.

Angular has its own predefined directives, which all have names. We have encountered some like `*ngIf` and `*ngFor` before. The good news is, we're not limited to using only those that Angular has predefined, we can create our own custom directives that can be attribute or structural directives and be in a position to use them in our app. Let's create a custom directive to understand how we can use this privilege to create our own directives in Angular.

Custom Directives

What if we wanted to strikethrough a goal after finishing it instead of deleting it? To this point, Angular has no inbuilt directive to do this for us. Let us create our own custom directive that will strikethrough a goal after we finish it.

At our terminal, let's create a directive using this command:

```
$ ng generate directive strikethrough
```

This command generates a file *strikethrough.directive.ts* which we are about to use to write the code that will perform a strikethrough and another file *strikethrough.directive.spec.ts* which is a test file. At the same time, it declares this directive in the root level modules file, *app.module.ts,* for us.

src/app/strikethrough.directive.ts

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

@Directive({
  selector: '[appStrikethrough]'
})
export class StrikethroughDirective {
  constructor(){}

}
```

A directive class has the `@directive` annotation. The annotation of a directive only has the `selector` property which is passed in as the attribute to the host element. The brackets `[]` make it an attribute directive. Angular looks in the HTML template for elements that have this selector and applies the logic that follows to the elements.

Let's go on to create this logic.

src/app/strikethrough.directive.ts

```
import { Directive,ElementRef } from '@angular/core';

@Directive({
  selector: '[appStrikethrough]'
})
export class StrikethroughDirective {
  constructor(private elem:ElementRef){}

}
```

We have imported the `ElementRef` at the top which we have used in the constructor of the directive's definition class. We use `ElementRef` to inject a reference to the host DOM element in which we will use this directive.

src/app/strikethrough.directive.ts

```
import { Directive,ElementRef} from '@angular/core';

@Directive({
  selector: '[appStrikethrough]'
})
export class StrikethroughDirective {

  constructor(private elem:ElementRef){
    this.elem.nativeElement.style.textDecoration='line-through';
  }
}
```

We have then targeted the host element's style attribute and changed the text-decoration to line-through. The `ElementRef` grants us direct access to the host DOM element through its `nativeElement` property.

Let's add this directive to our host element to see the changes it creates.

src/app/goal.component.html

```
<h1>My Goals</h1>
<ul>
    <div *ngFor='let goal of goals;let i = index'>
        <li appStrikethrough>{{goal.name}}</li>
```

```
        <button  (click)='toggleDetails(i)'>Toggle Details</button>

    <app-goal-details *ngIf="goal.showDescription" [goal]='goal' (isComplete)= 'completeGoal($even
t,i)'></app-goal-details>

    </div>
</ul>
<p *ngIf='goals.length > 5'>You have too many goals</p>
```

If our local server is still running, we see the goals have a line crossing the goal name which means that our directive actually works. That's great, right? However, using the directive this way is not helpful because we only wanted to strikethrough a goal if we have finished it.

User-initiated events

We'll put the user in control of making the strikethrough work on the app. When the user clicks a goal, it is marked as complete with the strikethrough and when the user double-clicks on the goal, it is marked an incomplete by removing the strikethrough from the goal.

Let's implement this in our directive.

src/app/strikethrough.directive.ts

```
import { Directive,ElementRef} from '@angular/core';

@Directive({
  selector: '[appStrikethrough]'
})
export class StrikethroughDirective {

  constructor(private elem:ElementRef){ }

  private textDeco(action:string){
    this.elem.nativeElement.style.textDecoration=action;

  }
}
```

We have changed our directive's logic by creating a function `textDeco()` which takes in an action and then performs a text-decoration using the action. Let's now create these actions that will feed into our `textDeco()` function.

src/app/strikethrough.directive.ts

```
import { Directive,ElementRef,HostListener} from '@angular/core';

@Directive({
  selector: '[appStrikethrough]'
})
export class StrikethroughDirective {


  constructor(private elem:ElementRef){}

  @HostListener("click") onClicks(){
    this.textDeco("line-through")
  }

  @HostListener("dblclick") onDoubleClicks(){
    this.textDeco("None")
  }

  private textDeco(action:string){
```

```
    this.elem.nativeElement.style.textDecoration=action;

  }
}
```

At the top, we have imported `HostListener` and used it to define the events that will be initiated by user actions, the first one being a click that creates a line-through and the second one being a double click which changes the text-decoration to none thus removing the line-through. We can see that for each action, we call our `textDeco()` function, `this.textDeco()` and pass in the name of the action as a string.

Let's serve our application and interact with it by clicking and double-clicking on the goals to see our directive at work.