Custom Directives

Sang Shin
JPassion.com
"Code with Passion!"

Topics

- Custom attribute directive
- Listening host events using @HostListerner(..)
- Parameterizing custom attribute directive using @Input()
- How structural directive gets converted internally
- Custom structural directive

Custom Attribute Directive

What we want do with a custom directive

- Create a custom directive called bg-highlight
  - When applied, it will highlight the element's background with designated color
- This directive will be used as an attribute to an element
  - <div bg-highlight>This content will be highlighted

Directive Implementation

- Since we are changing the appearance of an element, we need to have a reference to the element
  - Angular can inject it as ElementRef object
  - We can then access the native DOM element
  - this.elementRef.nativeElement.style.backgroundColor='green';
  - This is not recommended since you might want to run your app in non-browser platform in the future

- Better programming model is to Renderer instead
  - Angular can inject it as well
  - this.renderer.setElementStye(this.elementRef.nativeElement, 'background-color', 'orange')

Lab: Custom Attribute Directive

- Create a new directive called highlight
  - ng g directive bg-highlight
- Register the directive in the application module
  - directives : [BgHighlightDirective]
- In the directive, have ElementRef object injected through constructor
- Try to access the native element directly
- Try to use Renderer for cleaner code

Listening Host Events using @HostListener(..)

What we want do with a custom directive

- We want the element to change color whenever “mouseenter” and “mouseleave” events occur

- We can listen these host events using @HostListener

```
@HostListener('mouseenter') mouseenter(){
   this.renderer.setElementStye(this.elementRef.nativeElement,
                                'background-color', 'orange')
}

@HostListener('mouseleave') mouseleave(){
   this.renderer.setElementStye(this.elementRef.nativeElement,
                                'background-color', 'yellow')
}
```

Parameterizing Custom Directive

Parameterizing a custom directive

- We want let the template to pass color to the directive using property binding – don't forget the string ' inside “ 'green' ”

```
<div highlight [mouseEnterColor]=" 'green' " [mouseLeaveColor]=" 'pink' ">
```

- We can use @Input

```
@Input('mouseEnterColor') mouseEnterColor;

@HostListener('mouseenter') mouseenter(){
   this.renderer.setElementStye(this.elementRef.nativeElement,
                                'background-color',
                               this.mouseEnterColor)
}
```

How Structural Directive works internally

*ngIf and [ngIf]

- Angular internally converts *ngIf notation to [ngIf] notation
- The following code

```
<div *ngIf=”swtich”>Conditional text</div>
```

- Is converted into

```
<template [ngIf]=”switch”>
    <div>
        Conditional text
    </div>
</template>
```

Custom Structural Directive

What we want to build

- Create a custom directive called myUnless, which works exactly opposite to ngIf

Directive Implementation

- We need to have a reference to the template element- this is what we need to display or hide
  - Angular can inject it as TemplateRef object
- We also need view container
  - Angular can inject it as ViewContainerRef object
- We also needs access the condition value [ngUnless]="switch"
  - Use @Input() myUnless;
- Then add logic

```
if (!condition) {
   this.vcRef.createEmbeddedView(this.templateRef);
}
else {
   this.vcRef.clear();
}
```

Code with Passion!
JPassion.com