8. Directives

Introduction

Dans l'architecture Angular, les directives sont partout. Les Components sont des directives à la seule différence qu'ils possèdent une fonctionnalité de templating. En Angular, une directive est une classe à laquelle on applique le décorateur `@Directive` . Il existe deux sortes de directives :

- Les attribute directives : Elles ont pour but de modifier l'apparence ou le comportement d'un élément.
- Les directives structurelles : Elles ont pour but de modifier le DOM en ajoutant, enlevant ou replaçant un élément du DOM.

Documentation officielles

Attributes directives

Présentation

Une attribute directive s'applique, comme son nom l'indique, sur les attributs d'un élement HTML. Elle peuvent s'appliquer sur les attributs existants, ou sur des attributs customisés pour enrichir un comportement.

```
<p [style.color]="success ? 'green' : 'red'">Etat du chargement</p>

<button [disabled]="formIsValid">Valider</button>
```

Créer sa propre attribut directive

Il est possible de créer sa propre attribute directive. Pour cela Angular met à disposition une API Simple pour la mettre en oeuvre via le décorateur `@Directive` . Comme les composants Angular, il suffit de renseigner l'attribut `selector` pour pouvoir l'exploiter dans une template HTML par la suite.

La directive prend en entrée du constructeur l'élement HTML auquelle elle est associée, il est donc possible de travailler sur l'élement à partir de cette référence.

```
@Directive({
  selector: "[ma-directive-atr]",
})
```

```
export class MaDirective {
  constructor(private renderer: Renderer, private el: ElementRef) {}

  @HostListener("click")
  onMouseClick() {
    this.renderer.setElementStyle(this.el.nativeElement, "font-size", "2em");
  }
}
```

Ici au click sur l'élement, la taile de la police est augmentée.

```
<div ma-directive-atr></div>
```

Comme pour un composant, une directive doit ensuite être référencée dans le module dans l'attribut `declarations`, et dans l'attribut `exports` pour être rendu disponible depuis l'extérieur du module.

Documentation officielle

Directives structurelles

Présentation

Les directives structurelles conditionnent l'affichage d'élement dans le DOM. Angular met nativement à disposition des directives :

- `*ngIf` : une condition doit être validée pour afficher un élement dans le DOM,

```
<div *ngIf="loaded">Hello</div>
```

- `*ngSwitchCase` : la valeur d'une variable définie ce qui sera affichée dans le DOM,

```
<ul [ngSwitch]="utilisateur">
  <li *ngSwitchCase="'Anto'">Hello Anto</li>
  <li *ngSwitchCase="'Kevin'">Hello Kevin</li>
  <li *ngSwitchCase="'Thibaut'">Hello Thibaut</li>
  <li *ngSwitchDefault>Hello inconnu</li>
</ul>
```

- `*ngFor` : parcours un tableau, et génère autant d'élement dans le DOM qu'il y a d'item.

```
<ul>
  <li *ngFor="let utilisateur of utilisateurs">
    {{ utilisateur.nom }}
  </li>
</ul>
```

Vous noterez que les directives structurelles sont déclarées avec `*` dans la template HTML, c'est ce qui indique à Angular qu'elles sont structurelles, et non pas attributaires.

Créer sa propre directive structurelle

Comme pour un attribute directive, on utilisera le décorateur `@Directive`.

La directive ne prend pas les mêmes paramètres en entrée du constructeur, cette fois elle prend en paramètre le conteneur de l'élement HTML, ainsi qu'une référence vers la template de l'élement lui même. On peut donc

comprendre que cette directive ne créer l'élement qu'à partir du moment ou la condition de la directive en entrée est satisfaite. De plus si la condition est insatisfaite, alors que le composant a été créé sur DOM, la directive doit supprimer le composant du DOM.

```
@Directive({ selector: "[estOk]" })
export class IfDirective {
  constructor(
    private templateRef: TemplateRef<any>,
    private viewContainer: ViewContainerRef
  ) {}

  @Input()
  set estOk(ok: boolean) {
    if (ok) {
      this.viewContainer.createEmbeddedView(this.templateRef);
    } else {
      this.viewContainer.clear();
    }
  }
}
```

Documentation officielle

Excercice

Présentation

Application Météo.

L'objectif de cet excercice est d'afficher la liste des items de météo avec la directive ngFor. Nous nous étions arrété au parsage des élements du Flux JSON, et nous avons instancié un composant de présentation d'un item. Nous allons brancher tout ce petit monde.

Etapes d'utilisation de ngFor

1. Créer un boucle d'affichage avec ngFor dans le composant `meteo-view.component.ts` afin de parcourir la liste des résultats,
2. Exploiter le selecteur de `meteo-itemrenderer.component.ts` dans la boucle pour afficher le contenu de chaque item,
3. Le composant `meteo-itemrenderer` prendra en entrée un VO `meteo-item.vo` et l'affichera au travers de sa template. Utiliser pour le cela le décorateur `@Input()` sur la propriété setté.

Documentation ici

4. Animer l'affichage de l'item par une animation CSS native (cf TP JS Vanilla où cela a déjà été fait.).

L'item est affiché avec un Fade de 0 à 1. Cette fois, au lieu de créer la CSS dynamiquement, nous allons la modifier par binding.

Pour récupérer l'index d'un element (afin de mettre un délais sur l'affichage), utiliser l'index diponible dans la boucle ngFor dans `meteo-view.component.ts` , et utiliser un @Input pour passer cette information au composant `meteo-itemrenderer.component.ts` (par exemple propriété `indexOfElement`).

Vous binderez ensuite cette propriété à la propriété `animation-delay` dans la template `meteo-itemrenderer.component.html` , afin d'appliquer l'effet. Comme ceci :

```
  <div
    class="item-weather"
    *ngIf="meteoItem"
    [ngStyle]="{ 'animation-delay': indexOfElement / 10 + 's' }"
  >
```

Bravo, vous affichez la météo d'une ville, avec un effet très sexy !

Nous allons à présent parler formatage de la données avec les Pipes pour présenter correctement les libellés.

Correction disponible ici