Propriété input

Distinction entre source et cible d'une liaison de données

Dans les leçons précédentes nous nous étions concentrés sur des liaisons dans les templates qui se situaient **du côté droit des expressions** déclarant les liaisons.

Il s'agissait de déclarer la **source de liaison de données**.

Ainsi par exemple :

```
<div [hidden]="!valide">


<div *ngFor="let item of items">{{item.nom}}</div>
```

Nous allons maintenant nous concentrer sur **les liaisons à des cibles** : c'est-à-dire à des propriétés de directives **situées du côté gauche** des déclarations de liaison.

Ces propriétés de directives doivent être déclarées comme des `input` ou des `output`.

Pour rappel : les composants sont des directives particulières.

Note : pour bien distinguer la **cible** et la **source** dans une liaison dans une liaison de données :

- **La cible** d'une liaison est toujours entourée par `[]`, `()` ou `[()]`.
- **La source** d'une liaison est toujours entourée par des guillemets `" "` ou `{{}}`.li>

Ainsi par exemple :

```
<img [src]="urlImage"/>
```

Dans cet exemple, la source est `urlImage` et la cible est la propriété `src` native de l'élément HTML `image`.

Alors que dans cet exemple :

```
<app-enfant [prenom]="prenom"></app-enfant>
```

`"prenom"` est la **source** de la liaison de données (propriété située sur la classe de l'élément parent).

`[prenom]` est la **cible** de la liaison de données (propriété située sur la classe de l'élément enfant).

Déclaration d'une propriété `Input`

Les propriétés cibles dans une liaison de données doivent être explicitement marquées comme étant des `input` ou des `output` en utilisant des décorateurs.

Pour reprendre notre exemple précédent, il est donc nécessaire de déclarer sur le composant enfant que la propriété `prenom` est la cible d'une liaison de données :

```
@Input() prenom: string;
```

A la place de déclarer un `input` avec un décorateur `@Input()` précédent la propriété cible, nous aurions également pu utiliser la propriété `inputs` sur le décorateur du composant de la classe enfant.

Attention !, vous devez choisir entre déclarer la cible d'une liaison de données sur les métadonnées du composant enfant (avec la propriété `inputs`) ou alors dans la classe de ce composant avec le décorateur `@Input()` . Ne faites pas les deux !

Nous recommandons l'utilisation du décorateur pour débuter car il est plus expressif.

Approfondissement

Le décorateur `@Input()` permet de déclarer qu'une propriété est publique et qu'elle est disponible pour une liaison par un composant parent. Sans ce décorateur, Angular refuse la liaison de propriété.

Mais alors pourquoi, pour lier une propriété d'un élément du template d'un composant, à une propriété de sa classe, il n'y a pas besoin de mettre `@Input()` ?

La raison est simple : Angular traite le template du composant comme appartenant au composant. Le template du composant peut donc lier n'importe quelle propriété à la classe du composant.

Mais en revanche, par défaut les propriétés des composants ou des directives sont cachées pour les autres composants/directives. C'est pour cette raison qu'il est nécessaire d'utiliser `@Input()` afin de les rendre publiques.

Exemple

Maintenant que vous avez ces bases théoriques en tête, nous allons reprendre l'exemple vu dans la vidéo en repartant de la base de code de la leçon précédente.

Vérifiez que vous avez bien dans `angular.json` , sous la clé `"@schematics/angular:component"` , la propriété `"skipTests": true` .

Nous commençons par générer un composant :

```
ng g c fruit
```

Modification de `app.component.html`

Nous déclarons **une liaison de propriété** sur le composant `fruit` dans notre composant `app.component.html` :

```
<div
  style="height:100vh; padding-top:150px;"
  class="container d-flex flex-column"
>
  <div class="d-flex flex-row">
    <input class="m-10 flex-fill" [(ngModel)]="fruit" type="text" />
```

```
    <button class="btn btn-primary m-10"
(click)="addFruit()">Ajouter</button>
  </div>
  <ul class="list-primary">
    <app-fruit [fruit]="f" *ngFor="let f of fruits"></app-fruit>
  </ul>
</div>
```

Ici nous créons une instance du composant `fruit` pour chaque élément contenu dans le tableau `fruits`.

A chaque itération, nous effectuons **une liaison de propriété** entre la valeur de l'itération en cours de `f` et une propriété que nous allons déclarer dans le composant `fruit`. Nous appelons cette propriété `fruit`.

Modification de `fruit.component.ts`

Dans la classe du composant `fruit`, nous allons déclarer une propriété `input` en utilisant le décorateur `@Input()`.

De cette manière, `Angular` sait qu'il existe une liaison de propriété avec le composant parent :

```
import { Component, Input, OnInit } from "@angular/core";


@Component({
  selector: "app-fruit",
  templateUrl: "./fruit.component.html",
  styleUrls: ["./fruit.component.scss"]
})
export class FruitComponent implements OnInit {
  @Input() fruit: string;
  constructor() {}


  ngOnInit() {}
}
```

Attention ! Le nom de la propriété décorée avec `@Input()` doit exactement correspondre à la propriété utilisée dans le template du composant parent.

Modification de `fruit.component.html`

Il ne nous reste plus qu'à effectuer une interpolation de la propriété `fruit` dans le template :

```
<li>{{fruit}}</li>
```

Nous avons vu comment réaliser une communication dans le sens composant parent, (ici le composant racine `app`), vers le composant enfant (ici, le composant `fruit`).

Nous communiquons bien des données depuis la classe du composant parent vers la classe du composant enfant.

Code exécutable

Voici où nous en sommes

https://stackblitz.com/edit/angular-chap4-l2