Angular : interaction entre composant

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

Plan

Angular

Premières formes d'interaction

- Une application **Angular** est composée de plusieurs composants
- En utilisant des formulaires et des liens, on peut envoyer des données d'un composant à un autre

Angular

Autres formes d'interaction : parent-enfant

- Ajouter le sélecteur d'un premier composant dans le template d'un deuxième composant
  - on appelle le premier composant : composant fils
  - on appelle le deuxième composant : composant parent
- Définir le sens de transmission de données par
  - un composant web **Angular** : `ng-content` (transclusion)
  - des décorateurs `@Input()`, `@Output()`, `@ViewChild`, `@ViewChildren`, `@ContentChild`, `@ContentChildren`
  - des fonctions `input()`, `output()`, `viewChild()`, `viewChildren()`, `contentChild()`, `contentChildren()` et `model()`.

Angular

Autres formes d'interaction : parent-enfant

- Ajouter le sélecteur d'un premier composant dans le template d'un deuxième composant
  - on appelle le premier composant : composant fils
  - on appelle le deuxième composant : composant parent
- Définir le sens de transmission de données par
  - un composant web **Angular** : `ng-content` (transclusion)
  - des décorateurs `@Input()`, `@Output()`, `@ViewChild`, `@ViewChildren`, `@ContentChild`, `@ContentChildren`
  - des fonctions `input()`, `output()`, `viewChild()`, `viewChildren()`, `contentChild()`, `contentChildren()` et `model()`.

Autres formes d'interaction

Si aucun lien parent-enfant entre les composants : **solution** (`Subject` et `Service`)

Angular

Avant de commencer

- Créons deux composants `pere` et `fils`
- Définissions une route `/pere` pour le composant `pere`
- Ajoutons le sélecteur du composant fils `app-fils` dans `pere.html`

Angular

Le fichier `fils.ts`

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

@Component({
  selector: 'app-fils',
  imports: [],
  templateUrl: './fils.html',
  styleUrl: './fils.css'
})
export class FilsComponent implements OnInit {

  constructor() { }

  ngOnInit() { }
}
```

Angular

Le fichier `fils.ts`

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

@Component({
  selector: 'app-fils',
  imports: [],
  templateUrl: './fils.html',
  styleUrl: './fils.css'
})
export class FilsComponent implements OnInit {

  constructor() { }

  ngOnInit() { }
}
```

Le fichier `fils.html`

```
<li>
    Je suis un fils
</li>
```

Angular

Le fichier `pere.ts`

```
import { Component, OnInit } from '@angular/core';
import { FilsComponent } from "../fils/fils";

@Component({
  selector: 'app-pere',
  imports: [FilsComponent],
  templateUrl: './pere.html',
  styleUrl: './pere.css'
})
export class PereComponent implements OnInit {

  constructor() { }

  ngOnInit() { }
}
```

Angular

Le fichier `pere.ts`

```
import { Component, OnInit } from '@angular/core';
import { FilsComponent } from "../fils/fils";

@Component({
  selector: 'app-pere',
  imports: [FilsComponent],
  templateUrl: './pere.html',
  styleUrl: './pere.css'
})
export class PereComponent implements OnInit {

  constructor() { }

  ngOnInit() { }
}
```


Dans `pere.html`**, on définit trois fils**

```
<ul>
  <app-fils></app-fils>
  <app-fils></app-fils>
  <app-fils></app-fils>
</ul>
```

Pour passer du contenu texte à notre composant enfant, on fait :

```
<ul>
    <app-fils>John Wick</app-fils>
    <app-fils>Jack Shephard</app-fils>
    <app-fils>James Ford</app-fils>
</ul>
```

Pour passer du contenu texte à notre composant enfant, on fait :

```
<ul>
    <app-fils>John Wick</app-fils>
    <app-fils>Jack Shephard</app-fils>
    <app-fils>James Ford</app-fils>
</ul>
```


En testant le résultat est :

- Je suis un fils
- Je suis un fils
- Je suis un fils

Pour passer du contenu texte à notre composant enfant, on fait :

```
<ul>
    <app-fils>John Wick</app-fils>
    <app-fils>Jack Shephard</app-fils>
    <app-fils>James Ford</app-fils>
</ul>
```


En testant le résultat est :

- Je suis un fils
- Je suis un fils
- Je suis un fils

Le contenu ajouté par le père n'apparaît pas.

Angular

Pour que le composant fils puisse récupérer le contenu défini par le parent, on utilise la balise `ng-content` (transclusion)

```
<li>
    Je suis un fils : <ng-content></ng-content>
</li>
```

Angular

Pour que le composant fils puisse récupérer le contenu défini par le parent, on utilise la balise `ng-content` (transclusion)

```
<li>
    Je suis un fils : <ng-content></ng-content>
</li>
```

En testant le résultat est

- Je suis un fils : John Wick
- Je suis un fils : Jack Shephard
- Je suis un fils : James Ford

Angular

Remarque

La balise `ng-content` a un attribut `select` qui prend comme valeur

- le nom d'un attribut de n'importe quelle balise définie dans le composant parent
- le nom d'une classe **CSS**
- le nom d'une balise

Angular

Dans `pere.html`**, on définit le nom et le prénom dans deux balises ayant deux attributs différents :** `prenomContent` **et** `nomContent`

```
<ul>
  <app-fils>
    <span prenomContent> John </span>
    <span nomContent> Wick </span>
  </app-fils>
  <app-fils>
    <span prenomContent> Jack </span>
    <span nomContent> Shephard </span>
  </app-fils>
  <app-fils>
    <span prenomContent> James </span>
    <span nomContent> Ford </span>
  </app-fils>
</ul>
```

Angular

Dans `pere.html`**, on définit le nom et le prénom dans deux balises ayant deux attributs différents :** `prenomContent` **et** `nomContent`

```
<ul>
  <app-fils>
    <span prenomContent> John </span>
    <span nomContent> Wick </span>
  </app-fils>
  <app-fils>
    <span prenomContent> Jack </span>
    <span nomContent> Shephard </span>
  </app-fils>
  <app-fils>
    <span prenomContent> James </span>
    <span nomContent> Ford </span>
  </app-fils>
</ul>
```


Dans `fils.html`**, on sélectionne les données selon les deux attributs** `prenomContent` **et** `nomContent`

```
<li>
  Je suis un fils : mon nom est <ng-content select="[nomContent]"></ng-content>
  et mon prénom est <ng-content select="[prenomContent]"></ng-content>
</li>
```

Angular

En testant le résultat est

- Je suis un fils : mon nom est Wick et mon prénom est John
- Je suis un fils : mon nom est Shephard et mon prénom est Jack
- Je suis un fils : mon nom est Ford et mon prénom est James

Angular

Dans `pere.html`**, on peut aussi définir des classes dont la valeur est soit** `prenomContent` **ou** `nomContent`

```
<ul>
  <app-fils>
    <span class=prenomContent> John </span>
    <span class=nomContent> Wick </span>
  </app-fils>
  <app-fils>
    <span class=prenomContent> Jack </span>
    <span class=nomContent> Shephard </span>
  </app-fils>
  <app-fils>
    <span class=prenomContent> James </span>
    <span class=nomContent> Ford </span>
  </app-fils>
</ul>
```

Angular

Dans `pere.html`**, on peut aussi définir des classes dont la valeur est soit** `prenomContent` **ou** `nomContent`

```
<ul>
  <app-fils>
    <span class=prenomContent> John </span>
    <span class=nomContent> Wick </span>
  </app-fils>
  <app-fils>
    <span class=prenomContent> Jack </span>
    <span class=nomContent> Shephard </span>
  </app-fils>
  <app-fils>
    <span class=prenomContent> James </span>
    <span class=nomContent> Ford </span>
  </app-fils>
</ul>
```


Dans `fils.html`**, on sélectionne les données selon leurs classes**

```
<li>
  Je suis un fils : mon nom est <ng-content select=".nomContent"></ng-content>
  et mon prénom est <ng-content select=".prenomContent"></ng-content>
</li>
```

Angular

En testant le résultat est le même

- Je suis un fils : mon nom est Wick et mon prénom est John
- Je suis un fils : mon nom est Shephard et mon prénom est Jack
- Je suis un fils : mon nom est Ford et mon prénom est James

Angular

Dans `pere.html`, on peut aussi utiliser plusieurs balises différentes

```
<ul>
    <app-fils>
        <span> John Wick </span>
        <a href="#"> link </a>
    </app-fils>
    <app-fils>
        <span> Jack Shephard </span>
        <a href="#"> link </a>
    </app-fils>
    <app-fils>
        <span> James Ford </span>
        <a href="#"> link </a>
    </app-fils>
</ul>
```

Angular

Dans `pere.html`, on peut aussi utiliser plusieurs balises différentes

```
<ul>
    <app-fils>
        <span> John Wick </span>
        <a href="#"> link </a>
    </app-fils>
    <app-fils>
        <span> Jack Shephard </span>
        <a href="#"> link </a>
    </app-fils>
    <app-fils>
        <span> James Ford </span>
        <a href="#"> link </a>
    </app-fils>
</ul>
```


Dans `fils.html`, on sélectionne les données selon les balises

```
<li>
    Nom et prénom : <ng-content select="span"></ng-content>,
    pour apprendre plus => <ng-content select="a"></ng-content>
</li>
```

Angular

En testant le résultat est

- Nom et prénom : John Wick , pour apprendre plus => link
- Nom et prénom : Jack Shephard , pour apprendre plus => link
- Nom et prénom : James Ford , pour apprendre plus => link

Angular

Décorateurs disponibles pour l'interaction entre composants

- `@Input()` : permet à un composant fils de récupérer des données de son composant parent
- `@ViewChild()` : permet à un composant parent de récupérer les données de son composant enfant
- `@ViewChildren()` : permet à un composant parent de récupérer les données de ses composants enfants
- `@Output()` : permet à un composant parent de récupérer des données de son composant enfant

Angular

Dans cet exemple

On définit dans `fils.ts` deux attributs `ordre` et `villeNaissance` qui seront affichés dans le template.

Angular

Nouveau contenu de `fils.ts`

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

@Component({
  selector: 'app-fils',
  templateUrl: './fils.html',
  styleUrl: './fils.css'
})
export class FilsComponent {
  @Input() ordre = '';
  @Input() villeNaissance = '';
}
```


Modifions le fichier `fils.html`

```
<li>
  Je suis le {{ ordre }} fils et suis de {{ villeNaissance }}
</li>
```

Angular

Le fichier `pere.ts`

```
import { Component } from '@angular/core';
import { FilsComponent } from "../fils/fils";

@Component({
  selector: 'app-pere',
  imports: [FilsComponent],
  templateUrl: './pere.html',
  styleUrls: ['./pere.css']
})
export class PereComponent {

  tab: Array<string> = ['premier', 'deuxième', 'troisième'];
  nord = 'Lille';
  sud = 'Marseille';
  capitale = 'Paris';

}
```

Angular

Le fichier `pere.ts`

```
import { Component } from '@angular/core';
import { FilsComponent } from "../fils/fils";

@Component({
  selector: 'app-pere',
  imports: [FilsComponent],
  templateUrl: './pere.html',
  styleUrls: ['./pere.css']
})
export class PereComponent {

  tab: Array<string> = ['premier', 'deuxième', 'troisième'];
  nord = 'Lille';
  sud = 'Marseille';
  capitale = 'Paris';

}
```

Le fichier `pere.html`

```
<ul>
  <app-fils [ordre]="tab[0]" [villeNaissance]="sud"></app-fils>
  <app-fils [ordre]="tab[1]" [villeNaissance]="nord"></app-fils>
  <app-fils [ordre]="tab[2]" [villeNaissance]="capitale"></app-fils>
</ul>
```

Angular

Le résultat est

- `Je suis le premier fils et suis de Marseille`
- `Je suis le deuxième fils et suis de Lille`
- `Je suis le troisième fils et suis de Paris`

Angular

Et si le parent ne passe pas valeurs au fils comme attribut, aucune erreur ne sera signalée

```
<ul>
  <app-fils></app-fils>
  <app-fils></app-fils>
  <app-fils></app-fils>
</ul>
```

Angular

Et si le parent ne passe pas valeurs au fils comme attribut, aucune erreur ne sera signalée

```
<ul>
  <app-fils></app-fils>
  <app-fils></app-fils>
  <app-fils></app-fils>
</ul>
```


Remarque

Pour interdire l'utilisation du décorateur `@Input` si le parent ne passe pas la valeur, on peut l'indiquer avec l'attribut `required`.

Angular

Rendons les inputs obligatoires dans `fils.ts`

```
export class FilsComponent {

  @Input({ required: true }) ordre = '';
  @Input({ required: true }) villeNaissance = '';

}
```

Angular

Rendons les inputs obligatoires dans `fils.ts`

```
export class FilsComponent {

  @Input({ required: true }) ordre = '';
  @Input({ required: true }) villeNaissance = '';

}
```


Ainsi ce code (de `pere.html`) déclenche une erreur

```
<ul>
  <app-fils></app-fils>
  <app-fils></app-fils>
  <app-fils></app-fils>
</ul>
```

Angular

Depuis Angular 19, la fonction `input()` est la nouvelle manière recommandée pour définir les entrées de composant

```
import { Component, input } from '@angular/core';

@Component({
  selector: 'app-fils',
  imports: [],
  templateUrl: './fils.html',
  styleUrl: './fils.css',
})
export class FilsComponent {
  ordre = input()
  villeNaissance = input()
}
```

Angular

Depuis Angular 19, la fonction `input()` est la nouvelle manière recommandée pour définir les entrées de composant

```
import { Component, input } from '@angular/core';

@Component({
  selector: 'app-fils',
  imports: [],
  templateUrl: './fils.html',
  styleUrl: './fils.css',
})
export class FilsComponent {
  ordre = input()
  villeNaissance = input()
}
```

Dans `fils.html`), on utilise le getter

```
<li>
  Je suis le {{ ordre() }} fils et suis de {{ villeNaissance() }}
</li>
```

Angular

Rendons les inputs obligatoires dans `fils.ts`

```
import { Component, input } from '@angular/core';

@Component({
  selector: 'app-fils',
  imports: [],
  templateUrl: './fils.html',
  styleUrl: './fils.css',
})
export class FilsComponent {
  ordre = input.required()
  villeNaissance = input.required()
}
```

Angular

Rendons les inputs obligatoires dans `fils.ts`

```
import { Component, input } from '@angular/core';

@Component({
  selector: 'app-fils',
  imports: [],
  templateUrl: './fils.html',
  styleUrl: './fils.css',
})
export class FilsComponent {
  ordre = input.required()
  villeNaissance = input.required()
}
```


Ainsi ce code (de `pere.html`) déclenche une erreur

```
<ul>
  <app-fils></app-fils>
  <app-fils></app-fils>
  <app-fils></app-fils>
</ul>
```

Angular

Remarques

- Les **Signal Inputs** sont disponibles en pré-version développeur depuis **Angular 17.1** et sont considérés comme prêts pour la production à partir de la version 19 d'**Angular**.
- Ils sont maintenant la méthode recommandée pour créer de nouvelles entrées de composant.
- Ils intègrent les propriétés directement dans le système de réactivité d'Angular : ce qui conduit à un code plus réactif, plus propre, et plus performant.

Angular

Les `input` peuvent être typés

```
import { Component, input } from '@angular/core';

@Component({
  selector: 'app-fils',
  imports: [],
  templateUrl: './fils.html',
  styleUrl: './fils.css',
})
export class FilsComponent {
  ordre = input<string>()
  villeNaissance = input.required<string>()
}
```

Angular

Quelques interfaces prédéfinis dans `Angular`

- `OnInit` avec une méthode `ngOnInit()` qu'on utilise pour initialiser le composant.
- `OnChange` avec une méthode `ngOnChanges()` qu'on utilise pour détecter les changement de valeurs d'un fils.

Angular

Nouveau contenu de `fils.ts`

```
import { Component, Input, OnChanges, SimpleChanges } from '@angular/core';

@Component({
  selector: 'app-fils',
  templateUrl: './fils.html',
  styleUrls: ['./fils.css']
})
export class FilsComponent implements OnChanges {

  @Input() ordre = '';
  @Input() villeNaissance = '';

  ngOnChanges(changes: SimpleChanges): void {
    console.log(changes);
  }
}
```

Angular

Nouveau contenu de `fils.ts`

```
import { Component, Input, OnChanges, SimpleChanges } from '@angular/core';

@Component({
  selector: 'app-fils',
  templateUrl: './fils.html',
  styleUrls: ['./fils.css']
})
export class FilsComponent implements OnChanges {

  @Input() ordre = '';
  @Input() villeNaissance = '';

  ngOnChanges(changes: SimpleChanges): void {
    console.log(changes);
  }
}
```


Explication

- `SimpleChange` : classe **TypeScript** avec trois attributs : `previousValue`, `currentValue` et `firstChange`.
- `SimpleChanges` : interface **TypeScript** dont les clés sont les attributs changés et les valeurs sont des objets `SimpleChange`.

Angular

Pour afficher tous les changements

```
import { Component, Input, OnChanges, SimpleChanges } from '@angular/core';

@Component({
  selector: 'app-fils',
  templateUrl: './fils.html',
  styleUrls: ['./fils.css']
})
export class FilsComponent implements OnChanges {

  @Input() ordre = '';
  @Input() villeNaissance = '';

  ngOnChanges(changes: SimpleChanges): void {
    for (const key in changes) {
      const change = changes[key];
      const prevVal = change.previousValue;
      const curVal = change.currentValue;
      const firstChange = change.firstChange;
      console.log(prevVal, curVal, firstChange)
    }
  }
}
```

Exercice : **primeur-produit**

- Première partie
  - Créez deux composants `PrimeurComponent` et `ProduitComponent` : `PrimeurComponent` est le composant parent des composants `ProduitComponent`
  - Le composant `PrimeurComponent` a un attribut `produits` (voir ci-dessous).
  - Utilisez `@for` pour créer autant de composants `ProduitComponent` que d'éléments dans le tableau `produits` : chaque `ProduitComponent` reçoit le produit qu'il doit afficher.
- Deuxième partie
  - Dans `PrimeurComponent`, ajoutez un bouton `ajouter` et trois zones de saisie : une pour le nom, une pour le prix et une pour la quantité.
  - En cliquant sur le bouton `ajouter`, un nouveau composant `produit` s'ajoute (s'affiche) au composant (dans la page) avec le nom saisi par l'utilisateur.

Attribut de la classe `PrimeurComponent`

```
produits = [
    { nom: "banane", prix: 3, quantite: 10 },
    { nom: "fraise", prix: 10, quantite: 20 },
    { nom: "poivron", prix: 5, quantite: 10 }
]
```

Exercice : **list-item**

- Première partie
  - Créez deux composants `list` et `item` et définissez une route pour `list`.
  - Le composant `list` a deux attributs `couleurs` et `textes` (voir ci-dessous).
  - Utilisez `*ngFor` pour créer autant de `app-item` que d'éléments dans les tableaux : chaque composant `item` reçoit le texte et la couleur du composant `list`.
- Deuxième partie
  - Dans le composant `list`, ajoutez un bouton `ajouter` et deux zones de saisie :une pour le texte et une pour la couleur du texte à afficher (à saisir en anglais).
  - En cliquant sur le bouton `ajouter`, un nouveau composant `item` s'ajoute (s'affiche) au composant (dans la page) avec la couleur indiquée par l'utilisateur.

Les attributs de la classe `list.ts`

```
couleurs = ['red', 'blue', 'green'];
textes = ['hi', 'salut', 'ciao'];
```

Angular

Avant de commencer

- Créons deux composants `parent` et `child`
- Définissions une route `/parent` pour le composant `parent`

Angular

Avant de commencer

- Créons deux composants `parent` et `child`
- Définissions une route `/parent` pour le composant `parent`

Dans cet exemple

- Chaque élément `child` aura un champ texte pour saisir une note et un bouton pour envoyer la valeur au composant `parent`
- Le bouton sera désactivé après envoi
- Le sélecteur du composant fils `app-child` sera ajouté dans `parent.html`
- Chaque fois que le composant `parent` reçoit une note d'un de ses fils, il recalcule la moyenne et il l'affiche

Angular

Le fichier `child.html`

```
<h6> {{ nom }} </h6>
<input type=number name=note [(ngModel)]=note>
<button (click)="send()" [disabled]="buttonStatus">
    Send
</button>
```

Angular

Le fichier `child.ts`

```
import { Component, Input, Output, EventEmitter } from '@angular/core';
import { FormsModule } from '@angular/forms';

@Component({
  selector: 'app-child',
  imports: [FormsModule],
  templateUrl: './child.html',
  styleUrl: './child.css'
})
export class ChildComponent {
  @Input() nom = '';
  @Output() message = new EventEmitter<number>();
  note = 0;
  buttonStatus = false;

  send(): void {
    this.message.emit(this.note);
    this.buttonStatus = true;
  }
}
```

Angular

Le fichier `parent.ts`

```
import { CommonModule } from '@angular/common';
import { Component } from '@angular/core';
import { ChildComponent } from "../child/child";

@Component({
  selector: 'app-parent',
  imports: [ChildComponent],
  templateUrl: './parent.html',
  styleUrl: './parent.css'
})
export class ParentComponent {
  moyenne = 0;
  somme = 0;
  nbr = 0;
  enfants = ['Wick', 'Hoffman', 'Abruzzi'];

  computeAvg(note: number): void {
    this.somme += note;
    this.nbr++;
    this.moyenne = this.somme / this.nbr;
  }
}
```

Angular

Le fichier `parent.html`

```
<h3> Moyenne de mes enfants {{ moyenne }} </h3>
@for (enfant of enfants; track $index) {
    <app-child [nom]="enfant" (message)="computeAvg($event)" />
}
```

Angular

Depuis Angular 19, on peut aussi simplifier l'écriture et améliorer les performances en utilisant les signaux (`child.ts`)

```
import { Component, input, output } from '@angular/core';
import { FormsModule } from '@angular/forms';

@Component({
  selector: 'app-child',
  imports: [FormsModule],
  templateUrl: './child.html',
  styleUrl: './child.css',
})
export class ChildComponent {
  nom = input<string>();
  message = output<number>();
  note = 0;
  buttonStatus = false;

  send(): void {
    this.message.emit(this.note);
    this.buttonStatus = true;
  }
}
```

Angular

Ajoutons le getter du signal dans `child.html`

```
<h6> {{ nom() }} </h6>
<input type=number name=note [(ngModel)]=note>
<button (click)="send()" [disabled]="buttonStatus">
    Send
</button>
```

Angular

Exercice

- Le composant `parent` doit permettre à l'utilisateur de saisir le nom affecté à un composant `child`.
- Modifiez les composants `parent` et `child` pour qu'on puisse calculer la moyenne d'un nombre variable de notes.

Angular

Exercice 1 : **primeur-produit**

- Dans `PrimeurComponent`, déclarez un attribut `total`.
- Dans `ProduitComponent`, ajoutez une zone de saisie et un bouton.
- En choisissant une quantité et appuyant sur le bouton,
  - le total sera recalculé et affiché,
  - la quantité en stock sera mise à jour
  - et le bouton sera désactivé.

Angular

Exercice **clavier-touche** (Simulation d'un clavier virtuel)

- Créez deux composants `clavier` et `touche`.
- Définissez une route pour le composant `clavier`.
- Le composant `clavier` a un attribut `lettres` (voir ci-dessous)
- Le composant `touche` a un attribut `value` recevant une valeur du tableau `lettres` (un composant fils `touche` pour chaque valeur du tableau `lettres`).
- Chaque composant `touche` affiche la lettre qu'il a reçue sur un bouton.
- En cliquant sur ce bouton, la lettre s'affiche (à la suite des autres) dans une balise `textarea` définie dans le composant `clavier`.

Contenu du tableau `lettres`

```
lettres = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
   'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z
   ']
```

Angular

Objectif

Récupérer les données du premier composant fils à partir d'un composant parent.

Angular

Objectif

Récupérer les données du premier composant fils à partir d'un composant parent.

Comment ?

- Déclarer un composant fils comme attribut d'un composant parent et le décorer avec `@ViewChild()`
- Utiliser cet attribut pour récupérer les données souhaitées

Angular

Commençons par déclarer le composant fils comme attribut dans `pere.ts` et le décorer avec `@ViewChild()`

```
import { Component, OnInit, ViewChild } from '@angular/core';
import { FilsComponent } from '../fils/fils.component';

@Component({
  selector: 'app-pere',
  templateUrl: './pere.html',
  styleUrls: ['./pere.css']
})
export class PereComponent implements OnInit {

  @ViewChild(FilsComponent) fils: FilsComponent;

  tab: Array<string> = ['premier', 'deuxième', 'troisième'];
  nord = 'Lille';
  sud = 'Marseille';
  capitale = 'Paris';

  ngOnInit() { }
}
```

Angular

Si `fils` est souligné en rouge, alors ajoutons le suffix `!`

```
import { Component, OnInit, ViewChild } from '@angular/core';
import { FilsComponent } from '../fils/fils.component';

@Component({
  selector: 'app-pere',
  templateUrl: './pere.html',
  styleUrls: ['./pere.css']
})
export class PereComponent implements OnInit {

  @ViewChild(FilsComponent) fils!: FilsComponent;

  tab: Array<string> = ['premier', 'deuxième', 'troisième'];
  nord = 'Lille';
  sud = 'Marseille';
  capitale = 'Paris';

  ngOnInit() { }
}
```

Le décorateur `@ViewChild()` a un attribut `static` qui a par défaut la valeur `false`

```
import { Component, OnInit, ViewChild } from '@angular/core';
import { FilsComponent } from '../fils/fils.component';

@Component({
  selector: 'app-pere',
  templateUrl: './pere.html',
  styleUrls: ['./pere.css']
})
export class PereComponent implements OnInit {

  @ViewChild(FilsComponent, { static: true }) elementStatic!: FilsComponent;
  @ViewChild(FilsComponent, { static: false }) elementDynamic!: FilsComponent;

  tab: Array<string> = ['premier', 'deuxième', 'troisième'];
  nord = 'Lille';
  sud = 'Marseille';
  capitale = 'Paris';

  ngOnInit() { }
}
```

Le décorateur `@ViewChild()` a un attribut `static` qui a par défaut la valeur `false`

```
import { Component, OnInit, ViewChild } from '@angular/core';
import { FilsComponent } from '../fils/fils.component';

@Component({
  selector: 'app-pere',
  templateUrl: './pere.html',
  styleUrls: ['./pere.css']
})
export class PereComponent implements OnInit {

  @ViewChild(FilsComponent, { static: true }) elementStatic!: FilsComponent;
  @ViewChild(FilsComponent, { static: false }) elementDynamic!: FilsComponent;

  tab: Array<string> = ['premier', 'deuxième', 'troisième'];
  nord = 'Lille';
  sud = 'Marseille';
  capitale = 'Paris';

  ngOnInit() { }
}
```

Explication

- `{ static: false }` : `ViewChild` n'est accessible que dans `ngAfterViewInit`.
- `{ static: true }` : à utiliser si on souhaite accéder à `ViewChild` dans `ngOnInit`.

Pour vérifier

```
import { Component, OnInit, ViewChild } from '@angular/core';
import { FilsComponent } from '../fils/fils.component';

@Component({
  selector: 'app-pere',
  imports: [FilsComponent],
  templateUrl: './pere.html',
  styleUrl: './pere.css'
})
export class PereComponent implements OnInit {

  @ViewChild(FilsComponent, { static: true }) elementStatic!: FilsComponent;
  @ViewChild(FilsComponent, { static: false }) elementDynamic!: FilsComponent;

  tab: Array<string> = ['premier', 'deuxième', 'troisième'];
  nord = 'Lille';
  sud = 'Marseille';
  capitale = 'Paris';

  ngOnInit(): void {
    console.log(this.elementStatic.ordre); // ''
    console.log(this.elementDynamic.ordre); // ERROR TypeError: Cannot read property of
      undefined (reading 'ordre')
  }

  ngAfterViewInit(): void {
    console.log(this.elementStatic.ordre); // premier
    console.log(this.elementDynamic.ordre); // premier
  }
}
```

Angular

Depuis Angular 19, on peut aussi utiliser les signaux via la fonction `viewChild()`

```
import { Component, viewChild } from '@angular/core';
import { FilsComponent } from "../fils/fils";

@Component({
  selector: 'app-pere',
  imports: [FilsComponent],
  templateUrl: './pere.html',
  styleUrl: './pere.css',
})
export class PereComponent {

  fils = viewChild(FilsComponent);
  tab: Array<string> = ['premier', 'deuxième', 'troisième'];
  nord = 'Lille';
  sud = 'Marseille';
  capitale = 'Paris';
  ngAfterViewInit(): void {
    console.log(this.fils()?.ordre());
  }
}
```

Angular

On peut aussi utiliser `required()` et éviter le chaînage optionnel

```
import { Component, viewChild } from '@angular/core';
import { FilsComponent } from "../fils/fils";

@Component({
  selector: 'app-pere',
  imports: [FilsComponent],
  templateUrl: './pere.html',
  styleUrl: './pere.css',
})
export class PereComponent {

  fils = viewChild.required(FilsComponent);
  tab: Array<string> = ['premier', 'deuxième', 'troisième'];
  nord = 'Lille';
  sud = 'Marseille';
  capitale = 'Paris';
  ngAfterViewInit(): void {
    console.log(this.fils().ordre());
  }
}
```

Angular

Objectif

Récupérer les données de tous les composants fils à partir d'un composant parent.

Angular

Objectif

Récupérer les données de tous les composants fils à partir d'un composant parent.

Comment ?

- Déclarer un `QueryList` (tableau) de composant fils comme attribut d'un composant parent et le décorer avec `@ViewChildren()`
- Utiliser cet attribut pour récupérer les données souhaitées

Angular

Utilisons `@ViewChildren` pour récupérer un tableau de composant fils

```
import { AfterViewInit, Component, OnInit, QueryList, ViewChildren } from '@angular/core';
import { FilsComponent } from '../fils/fils.component';

@Component({
  selector: 'app-pere',
  templateUrl: './pere.html',
  styleUrls: ['./pere.css']
})
export class PereComponent implements OnInit, AfterViewInit {

  @ViewChildren(FilsComponent) fils!: QueryList<FilsComponent>;

  tab: Array<string> = ['premier', 'deuxième', 'troisième'];
  nord = 'Lille';
  sud = 'Marseille';
  capitale = 'Paris';

  ngOnInit() {  }

  ngAfterViewInit(): void {
    this.fils?.forEach(elt => console.log(elt));
    // affiche les trois FilsComposant dans la console
  }

}
```

Angular

Depuis Angular 19, on peut aussi utiliser les signaux via la fonction `viewChildren()`

```
import { AfterViewInit, Component, OnInit, viewChildren } from '@angular/core';
import { FilsComponent } from "../fils/fils";

@Component({
  selector: 'app-pere',
  imports: [FilsComponent],
  templateUrl: './pere.html',
  styleUrl: './pere.css',
})
export class PereComponent implements OnInit, AfterViewInit {

  fils = viewChildren(FilsComponent)
  tab: Array<string> = ['premier', 'deuxième', 'troisième'];
  nord = 'Lille';
  sud = 'Marseille';
  capitale = 'Paris';

  ngOnInit() { }

  ngAfterViewInit(): void {
    this.fils()?.forEach(elt => console.log(elt));
  }
}
```

Angular

Exercice 1 (suite de **primeur-produit**)

- Faites les changements nécessaires pour que `ProduitComponent` n'envoie plus la quantité commandée au composant `PrimeurComponent`.
- Le composant `PrimeurComponent` doit récupérer la quantité commandée en utilisant `@ViewChildren` ou `viewChildren`.

Angular

Exercice 2 (suite de l'exercice **list-item**)

- Depuis le composant `list`, on veut permettre à l'utilisateur de modifier la couleur d'un ou tous les composant(s) `item`.
- Pour cela, on ajoute deux nouvelles zones de saisie : une pour l'indice du composant `item` (qu'on souhaite modifier sa couleur) et la deuxième est la couleur qu'on veut lui attribuer.
- Si l'indice n'existe pas (négatif ou supérieur ou égal au nombre des composants `item`), on modifie la couleur de tous les `item`. Sinon, on modifie seulement la couleur du composant `item` ayant cet indice.

Angular

Objectif

Définir un composant titre pour l'utiliser chaque fois qu'on a un composant père avec ses composants fils.

Angular

Objectif

Définir un composant titre pour l'utiliser chaque fois qu'on a un composant père avec ses composants fils.

Comment ?

- Créer un composant `titre`
- Utiliser la balise `@ContentChild` pour récupérer le contenu d'un nœud enfant défini par la balise `ng-content`

Le fichier `titre.ts`

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

@Component({
  selector: 'app-titre',
  templateUrl: './titre.html',
  styleUrls: ['./titre.css']
})
export class TitreComponent implements OnInit {

  @Input() valeur!: string;
  @Input() couleur!: string;

  constructor() { }
  ngOnInit() { }

}
```

Le fichier `titre.html`

```
<h1 [ngStyle]="{color: couleur}">{{ valeur }}</h1>
```

Angular

Dans un composant du module `cours`, on ajoute la balise `app-pere` contenant une balise `app-titre`

```
<app-pere>
  <app-titre [couleur]="'red'" [valeur]="'Mes contacts'"></app-titre>
</app-pere>
```

Angular

Dans un composant du module `cours`, on ajoute la balise `app-pere` contenant une balise `app-titre`

```
<app-pere>
  <app-titre [couleur]="'red'" [valeur]="'Mes contacts'"></app-titre>
</app-pere>
```


N'oublions pas les apostrophes autour de `red` et `Mes contacts`.

Angular

Dans `pere.html`, on récupère sélectionne le titre dans la balise `ng-content`

```
<ng-content select="app-titre"></ng-content>
<ul>
    <app-fils [ordre]="tab[0]" [villeNaissance]="sud"></app-fils>
    <app-fils [ordre]="tab[1]" [villeNaissance]="nord"></app-fils>
    <app-fils [ordre]="tab[2]" [villeNaissance]="capitale"></app-fils>
</ul>
```

Dans `pere.ts`, on peut récupérer les attributs du composant `titre`

```
import { Component, OnInit, AfterContentInit, ContentChild } from '@angular/core';
import { TitreComponent } from '../titre/titre.component';

@Component({
  selector: 'app-pere',
  templateUrl: './pere.html',
  styleUrls: ['./pere.css']
})
export class PereComponent implements OnInit, AfterContentInit {

  @ContentChild(TitreComponent, { static: false }) titre!: TitreComponent;

  tab: Array<string> = ['premier', 'deuxième', 'troisième'];
  nord = 'Lille';
  sud = 'Marseille';
  capitale = 'Paris';

  ngOnInit() { }

  ngAfterContentInit(): void {
    console.log(this.titre?.valeur ?? "Aller au composant contenant <app-pere>");
  }
}
```

Dans `pere.ts`, on peut récupérer les attributs du composant `titre`

```
import { Component, OnInit, AfterContentInit, ContentChild } from '@angular/core';
import { TitreComponent } from '../titre/titre.component';

@Component({
  selector: 'app-pere',
  templateUrl: './pere.html',
  styleUrls: ['./pere.css']
})
export class PereComponent implements OnInit, AfterContentInit {

  @ContentChild(TitreComponent, { static: false }) titre!: TitreComponent;

  tab: Array<string> = ['premier', 'deuxième', 'troisième'];
  nord = 'Lille';
  sud = 'Marseille';
  capitale = 'Paris';

  ngOnInit() { }

  ngAfterContentInit(): void {
    console.log(this.titre?.valeur ?? "Aller au composant contenant <app-pere>");
  }
}
```

Pour tester, Aller au composant contenant `<app-pere>`.

Angular

Explication

- `@ContentChild` permet de récupérer le premier composant sélectionné par `ng-content`
- `@ContentChildren` permet de récupérer tous les composants sélectionnés par `ng-content`

Angular

Explication

- `@ContentChild` permet de récupérer le premier composant sélectionné par `ng-content`
- `@ContentChildren` permet de récupérer tous les composants sélectionnés par `ng-content`

Comment ?

- Comme pour `@ViewChildren()`, déclarer un `QueryList` (tableau) de composant et le décorer avec `@ContentChildren()`
- Utiliser cet attribut pour récupérer les données souhaitées

Angular

Exercice **pays-ville** (Première partie) : à faire avec les décorateurs `@Input` et `@Output`

- Créez deux composants `PaysComponent` et `VilleComponent` : `PaysComponent` est le composant parent des composants `VilleComponent`.
- `PaysComponent` a un attribut `villes`.

```
villes = ['Marseille', 'Lyon', 'Paris']
```

- Le contenu de `villes` est affiché dans une balise `div` dans le `template` de `PaysComponent`.

```
<div>
    @for(elt of villes; track elt; let i = $index) {
        {{ elt }}
    }
</div>
```

- Utilisez `@for` pour créer autant de composants `VilleComponent` que d'éléments dans le tableau `villes` : chaque composant `VilleComponent` reçoit le nom d'une ville qu'il doit afficher dans une balise `input`.
- En modifiant la valeur de l'`input`, la valeur sera également modifiée dans le `template` de `PaysComponent`.

Angular

Première solution : code de `pays.ts`

```
import { Component } from '@angular/core';
import { VilleComponent } from '../ville/ville';

@Component({
  selector: 'app-pays',
  imports: [VilleComponent],
  templateUrl: './pays.html',
  styleUrl: './pays.css'
})
export class PaysComponent {

  villes = ['Marseille', 'Lyon', 'Paris']

  updateVille(ind: number, value: any) {
    this.villes[ind] = value
  }
}
```

Angular

Première solution : code de `pays.html`

```
<h1>Pays</h1>
<div>
    @for(elt of villes; track elt; let i = $index) {
        {{ elt }}
    }
</div>
<div>
    @for(elt of villes; track elt; let i = $index) {
    <app-ville [ville]="elt" (sendData)="updateVille(i, $event)" />
    }
</div>
```

Angular

Première solution : code de `ville.ts`

```
import { Component, EventEmitter, Input, Output } from '@angular/core';
import { FormsModule } from '@angular/forms';

@Component({
  selector: 'app-ville',
  imports: [FormsModule],
  templateUrl: './ville.html',
  styleUrl: './ville.css'
})
export class VilleComponent {
  @Input() ville = ''
  @Output() sendData = new EventEmitter<string>()

  envoyer() {
    this.sendData.emit(this.ville)
  }
}
```

Angular

Première solution : code de `ville.html`

```
<h2>Ville</h2>
<div>
    <input [(ngModel)]="ville" (input)="envoyer()">
</div>
```

Angular

Deuxième solution

- Utiliser **Two-way bainding** pour communiquer des données aux composants enfants
- Utiliser pour `@Output()` un attribut dont le nom contient le nom de l'attribut de l'`@Input()` + le mot-clé `Change`

Angular

Deuxième solution : code de `pays.ts`

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

@Component({
  selector: 'app-pays',
  templateUrl: './pays.html',
  styleUrl: './pays.css'
})
export class PaysComponent {
  villes = ['Marseille', 'Lyon', 'Paris']
}
```

Angular

Deuxième solution : code de `pays.html`

```
<h1>Pays</h1>
<div>
    @for(elt of villes; track elt) {
        {{ elt }}
    }
</div>
<div>
    @for(elt of villes; track elt; let i = $index) {
    <app-ville [(ville)]="villes[i]"/>
    }
</div>
```

Angular

Deuxième solution : code de `pays.html`

```
<h1>Pays</h1>
<div>
    @for(elt of villes; track elt) {
        {{ elt }}
    }
</div>
<div>
    @for(elt of villes; track elt; let i = $index) {
    <app-ville [(ville)]="villes[i]"/>
    }
</div>
```

Plus besoin d'écouter l'évènement déclenché par l'enfant ni de la fonction associée.

Angular

Rappel : l'écriture suivante, en JavaScript, ne permet pas de modifier le tableau mais ne génère pas d'erreur

```
villes = ['Marseille', 'Lyon', 'Paris']

for (let item in villes) {
    item = ''
}

console.log(villes);
// affiche [ 'Marseille', 'Lyon', 'Paris' ]
```

Angular

Rappel : l'écriture suivante, en JavaScript, ne permet pas de modifier le tableau mais ne génère pas d'erreur

```
villes = ['Marseille', 'Lyon', 'Paris']

for (let item in villes) {
    item = ''
}

console.log(villes);
// affiche [ 'Marseille', 'Lyon', 'Paris' ]
```


En revanche, les boucles indexées permettent de modifier le tableau

```
for (let i = 0; i < villes.length; i++) {
    villes[i] = ''
}

console.log(villes);
// affiche [ '', '', '' ]
```

Angular

Par conséquent, l'écriture suivante en Angular déclenche une erreur car Two-way binding ne permet pas de modifier la valeur d'une boucle non indexée

```
<div>
    @for(elt of villes; track elt; let i = $index) {
    <app-ville [(ville)]="elt"/>
    }
</div>
```

Angular

Par conséquent, l'écriture suivante en Angular déclenche une erreur car Two-way binding ne permet pas de modifier la valeur d'une boucle non indexée

```
<div>
    @for(elt of villes; track elt; let i = $index) {
    <app-ville [(ville)]="elt"/>
    }
</div>
```


Il suffit donc de la remplacer par

```
<div>
    @for(elt of villes; track elt; let i = $index) {
    <app-ville [(ville)]="villes[i]"/>
    }
</div>
```

Angular

Deuxième solution : code de `ville.ts`

```
import { Component, EventEmitter, Input, Output } from '
   @angular/core';

@Component({
  selector: 'app-ville',
  templateUrl: './ville.html',
  styleUrl: './ville.css'
})
export class VilleComponent {

  @Input() ville = ''
  @Output() villeChange = new EventEmitter<string>()

  envoyer() {
    this.villeChange.emit(this.ville)
  }
}
```

Angular

Deuxième solution : code de `ville.html`

```
<h2>Ville</h2>
<div>
    <input [(ngModel)]="ville" (input)="envoyer()" >
</div>
```

Angular

Troisième solution

- Pas besoin de `output` ni `@Output`.
- Utiliser `model` à la place de `@Input` et `input`.
- Aucun changement par rapport à la deuxième solution chez le parent.

Angular

Troisième solution : code de `ville.ts`

```
import { Component, model } from '@angular/core';
import { FormsModule } from '@angular/forms';

@Component({
  selector: 'app-ville',
  imports: [FormsModule],
  templateUrl: './ville.html',
  styleUrl: './ville.css'
})
export class VilleComponent {
  ville = model<string>('')
}
```

Angular

Deuxième solution : code de `ville.html`

```
<h2>Ville</h2>
<div>
    <input [(ngModel)]="ville">
</div>
```

Angular

Exercice : **pays-ville** (Deuxième partie)

- Dans le composant `PaysComponent`, ajoutons un attribut `codesPostaux` (voir ci-dessous). Chaque élément d'indice `i` du tableau `codesPostaux` sera affiché avec l'élément d'indice `i` de `villes`.
- Le composant `PaysComponent` transmet à chaque enfant (`VilleComponent`) une ville et un code postal. Les deux valeurs seront affichées dans deux `<input>` différents.
- En modifiant la valeur de l'`input`, la valeur doit être mise à jour dans le `template` de `PaysComponent`.

Attributs à déclarer dans `pays.ts`

```
villes = ['Marseille', 'Lyon', 'Paris'],
codesPostaux = ['13000', '69000', '75000']
```

Angular

Dans `app.html`, définissons un paragraphe ayant une variable locale `ref`

```
<p #ref>
  contenu initial
</p>
```

Dans `app.html`, on peut récupérer ce paragraphe grâce à la variable locale `ref` et appliquer des méthodes natives du JavaScript

```
import { Component,  ViewChild, ElementRef, AfterViewInit } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.html',
  styleUrls: ['./app.css']
})
export class AppComponent implements AfterViewInit {

  @ViewChild('ref', {static: false}) p!: ElementRef;

  ngAfterViewInit(): void {
    this.p.nativeElement.innerHTML += 'texte ajouté depuis le composant .ts';
  }
}
```

Angular

Exercice **moyenne-note**

- Créons deux composants `moyenne` et `note`
- Dans le composant `moyenne`, on peut ajouter plusieurs composants `note` (en cliquant sur un bouton `ajouter`)
- Le composant `note` permet à l'utilisateur de saisir une valeur et un coefficient. Pendant la saisie, la valeur de la moyenne (définie dans le composant `moyenne`) se met à jour.
- Vous pouvez utiliser une interface `Note` avec deux attributs `valeur` et `coefficient`

Angular

Exercice **moyenne-note**

- Créons deux composants `moyenne` et `note`
- Dans le composant `moyenne`, on peut ajouter plusieurs composants `note` (en cliquant sur un bouton `ajouter`)
- Le composant `note` permet à l'utilisateur de saisir une valeur et un coefficient. Pendant la saisie, la valeur de la moyenne (définie dans le composant `moyenne`) se met à jour.
- Vous pouvez utiliser une interface `Note` avec deux attributs `valeur` et `coefficient`

Exercice (suite de l'exercice précédent)

Le composant `note` contenant la valeur min sera affiché en rouge et le composant contenant la valeur max sera affiché en vert.

Angular

Avant de commencer

- Créons trois composants `container`, `first` et `second`
- Créons un service `message`
- Ajoutons les deux sélecteurs `app-first` et `app-second` dans `container.html`
- Définissons une route `/container` pour le composant `container`

Angular

Avant de commencer

- Créons trois composants `container`, `first` et `second`
- Créons un service `message`
- Ajoutons les deux sélecteurs `app-first` et `app-second` dans `container.html`
- Définissons une route `/container` pour le composant `container`

Contenu de `container.html`

```
<app-first></app-first>
<app-second></app-second>
```

Angular

Idée

- Définir un `subject` dans `message.service.ts`
- Injecter le service dans le constructeur de deux composants `first` et `second`
- Utiliser le service pour émettre les données d'un composant vers un autre

Angular

Contenu de notre service

```
import { Injectable } from '@angular/core';
import { Subject } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class MessageService {

  private subject = new Subject<string>();

  envoyerMessage(msg: string) {
    this.subject.next(msg);
  }
  accederMessage() {
    return this.subject;
  }
}
```

Angular

Le fichier `first.html`

```
<div>
  Message : <input type=text [(ngModel)]="msg" >
  <button (click)="ajouterMessage()">
    Ajouter message
  </button>
</div>
```

Angular

Définissons `ajouterMessage()` dans `first.ts` et utilisons `MessageService` pour envoyer le message aux observateurs

```
import { Component } from '@angular/core';
import { MessageService } from 'src/app/services/message.service';

@Component({
  selector: 'app-first',
  templateUrl: './first.html',
  styleUrls: ['./first.css']
})
export class FirstComponent {

  msg: string = '';

  constructor(private messageService: MessageService) { }

  ajouterMessage() {
    this.messageService.envoyerMessage(this.msg);
    this.msg = '';
  }
}
```

Angular

Dans `second.html`**, on affiche la liste de messages envoyés par** `FirstComponent`

```
<ul>
    <li *ngFor="let message of messages">
        {{ message }}
    </li>
</ul>
```

Angular

Dans `second.ts` , il faut s'abonner au `subject` et utiliser la méthode `accederMessage()` pour récupérer les messages envoyés

```
import { Component, OnInit, OnDestroy } from '@angular/core';
import { Subscription } from 'rxjs';
import { MessageService } from 'src/app/services/message.service';

@Component({
  selector: 'app-second',
  templateUrl: './second.html',
  styleUrls: ['./second.css']
})
export class SecondComponent implements OnInit, OnDestroy {

  messages: Array<string> = [];
  subscription!: Subscription;

  constructor(private messageService: MessageService) { }

  ngOnInit() {
    this.subscription = this.messageService.accederMessage().subscribe(
      msg => this.messages.push(msg)
    );
  }

  ngOnDestroy() {
    this.subscription.unsubscribe();
  }
}
```

Angular

Exercice

- On veut que l'échange entre deux composants soit bidirectionnel.
- Quand le premier envoie, le deuxième affiche et inversement.

Angular

Exercice

- Créez deux composant `tchat` et `participant`
- Définissez une route `/tchat` pour le composant `tchat`
- Le composant `tchat` contient un champ texte `nom` et un bouton `ajouter` qui permet d'ajouter un nouveau composant `participant` dans `tchat`
- Chaque composant `participant` peut envoyer des messages à tous les autres participants
- Les participants reçoivent immédiatement les messages envoyés et les affichent
- Un participant n'affiche pas ses propres messages