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

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

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`
  - des décorateurs `@Input()`, `@Output()`, `@ViewChild`, `@ViewChildren`, `@ContentChild`, `@ContentChildren`

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`
  - des décorateurs `@Input()`, `@Output()`, `@ViewChild`, `@ViewChildren`, `@ContentChild`, `@ContentChildren`

Autres formes d'interaction : enfant-enfant ou parent-enfant

Le sélecteur du premier composant et celui du deuxième se trouvent au même niveau $\Rightarrow$ même parent : **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.component.html`

Angular

Le fichier `fils.component.ts`

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

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

  constructor() { }

  ngOnInit() { }
}
```

Le fichier `fils.component.html`

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

Angular

Le fichier `pere.component.ts`

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

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

  constructor() { }

  ngOnInit() { }
}
```

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

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

Angular

Et si on veut 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>
```

Angular

Et si on veut 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`

```
<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`

```
<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

Dans `pere.component.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 `pere.component.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.component.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

Dans `pere.component.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 `pere.component.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.component.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

Dans `pere.component.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 `pere.component.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.component.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.component.ts` deux attributs `ordre` et `villeNaissance` qui seront affichés dans le template.

Nouveau contenu de `fils.component.ts`

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

@Component({
  selector: 'app-fils',
  templateUrl: './fils.component.html',
  styleUrls: ['./fils.component.css']
})
export class FilsComponent implements OnInit {
  @Input() ordre = '';
  @Input() villeNaissance = '';

  constructor() { }

  ngOnInit() { }
}
```

Modifions le fichier `fils.component.html`

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

Le fichier `pere.component.ts`

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

@Component({
  selector: 'app-pere',
  templateUrl: './pere.component.html',
  styleUrls: ['./pere.component.css']
})
export class PereComponent implements OnInit {
  tab: Array<string> = ['premier', 'deuxième', 'troisième'];
  nord = 'Lille';
  sud = 'Marseille';
  capitale = 'Paris';
  constructor() { }

  ngOnInit() { }
}
```

Le fichier `pere.component.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

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

Nouveau contenu de `fils.component.ts`

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

@Component({
  selector: 'app-fils',
  templateUrl: './fils.component.html',
  styleUrls: ['./fils.component.css']
})
export class FilsComponent implements OnInit, OnChanges {
  @Input() ordre: string;
  @Input() villeNaissance: string;
  constructor() { }
  ngOnInit() { }

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

Nouveau contenu de `fils.component.ts`

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

@Component({
  selector: 'app-fils',
  templateUrl: './fils.component.html',
  styleUrls: ['./fils.component.css']
})
export class FilsComponent implements OnInit, OnChanges {
  @Input() ordre: string;
  @Input() villeNaissance: string;
  constructor() { }
  ngOnInit() { }

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

Remarque

- `SimpleChanges` est un objet JavaScript dont les clés sont les attributs changés et les valeurs sont des objets `SimpleChange`
- `SimpleChange` est un objet JavaScript contenant trois clés : `previousValue`, `currentValue` et `firstChange()` ;

Pour afficher tous les changements

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

@Component({
  selector: 'app-fils',
  templateUrl: './fils.component.html',
  styleUrls: ['./fils.component.css']
})
export class FilsComponent implements OnInit, OnChanges {
  @Input() ordre: string;
  @Input() villeNaissance: string;
  constructor() { }
  ngOnInit() { }

  ngOnChanges(changes: SimpleChanges): void {
    for (const key of Object.keys(changes)) {
      console.log(key);
      const obj = changes[key];
      for (const cle of Object.keys(obj)) {
        console.log(cle, obj[cle]);
      }
    }
  }
}
```

Angular

Exercice

- Dans `pere.component.html`, ajoutez deux `input` : un premier pour `nom` et un deuxième pour `ville`
- Dans `pere.component.ts`, définissez deux variables `nom` et un deuxième pour la `ville` et deux tableaux `noms` et `villes`
- Créez autant de `app-fils` que d'éléments dans `noms` (ou `villes`)
- Chaque fils récupère et affiche les valeurs envoyées par le composant `pere`
- L'utilisateur pourra ajouter un nombre indéterminé de composant `fils` dans `pere`.

Angular

Exercice **list-item**

- Créons deux composants `list` et `item`.
- Chaque composant `item` reçoit le texte et sa couleur d'affichage du composant `list`.
- Dans le composant `list`, on a deux zones de saisie : la première pour le texte, la deuxième pour la couleur du texte à afficher (à saisir en anglais).
- En cliquant sur le bouton `ajouter` du composant `list`, un nouveau composant `item` s'ajoute (s'affiche) au composant (dans la page) avec la couleur indiquée par l'utilisateur.
- L'utilisateur pourra ajouter un nombre indéterminé de texte.

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.component.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.component.html',
  styleUrls: ['./pere.component.css']
})
export class PereComponent implements OnInit {

  @ViewChild(FilsComponent) fils: FilsComponent;

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

  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.component.html',
  styleUrls: ['./pere.component.css']
})
export class PereComponent implements OnInit {

  @ViewChild(FilsComponent, { static: false }) fils: FilsComponent;

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

  ngOnInit() { }
}
```

`{ static: false }` car les attributs du composant enfant sont alimentés par le père et ne sont donc pas initialisés dans le composant.

on ne peut accéder aux attributs d'un composant enfant non-statique qu'après initiation de la vue

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

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

  @ViewChild(FilsComponent, { static: false }) fils: FilsComponent;
  tab: Array<string> = ['premier', 'deuxième', 'troisième'];
  nord = 'Lille';
  sud = 'Marseille';
  capitale = 'Paris';
  constructor() { }

  ngOnInit() {
    console.log(this.fils.ordre);
  }
}
```

ce code génère une erreur car le composant fils est encore indéfini.

En mettant `{ static: true }`, on indique qu'il ne faut plus attendre l'initiation de la vue (fils) car ses données sont statiques

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

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

  @ViewChild(FilsComponent, { static: true }) fils: FilsComponent;
  tab: Array<string> = ['premier', 'deuxième', 'troisième'];
  nord = 'Lille';
  sud = 'Marseille';
  capitale = 'Paris';
  constructor() { }
  ngOnInit() {
    console.log(this.fils.ordre); // affiche undefined
  }
}
```

En mettant `{ static: true }`, on indique qu'il ne faut plus attendre l'initiation de la vue (fils) car ses données sont statiques

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

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

  @ViewChild(FilsComponent, { static: true }) fils: FilsComponent;
  tab: Array<string> = ['premier', 'deuxième', 'troisième'];
  nord = 'Lille';
  sud = 'Marseille';
  capitale = 'Paris';
  constructor() { }
  ngOnInit() {
    console.log(this.fils.ordre); // affiche undefined
  }
}
```

Affecter une valeur à `ordre` dans `fils.component.ts` et vérifier que cette valeur est affichée à la place de `undefined`.

Remettons `{ static: false }` et indiquons qu'il faut attendre que la vue soit initiée pour accéder aux attributs

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

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

  @ViewChild(FilsComponent, { static: false }) fils: FilsComponent;
  tab: Array<string> = ['premier', 'deuxième', 'troisième'];
  nord = 'Lille';
  sud = 'Marseille';
  capitale = 'Paris';
  constructor() { }
  ngOnInit() { }

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

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

Utilisons @ViewChildren pour récupérer un tableau de composant fils

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

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

  @ViewChildren(FilsComponent) fils: QueryList<FilsComponent> |
     undefined;
  tab: Array<string> = ['premier', 'deuxième', 'troisième'];
  nord = 'Lille';
  sud = 'Marseille';
  capitale = 'Paris';
  constructor() { }
  ngOnInit() {  }

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

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

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

Angular

Objectif

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

Comment ?

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

Angular

Le fichier `titre.component.ts`

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

@Component({
  selector: 'app-titre',
  templateUrl: './titre.component.html',
  styleUrls: ['./titre.component.css']
})
export class TitreComponent implements OnInit {
  @Input() valeur: string;
  @Input() couleur: string;
  constructor() { }

  ngOnInit() {
  }

}
```

Le fichier `titre.component.html`

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

Angular

Dans `app.component.html`, on ajoute la balise `app-pere` contenant une balise `app-titre`

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

<router-outlet></router-outlet>
```

Angular

Dans `app.component.html`, on ajoute la balise `app-pere` contenant une balise `app-titre`

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

<router-outlet></router-outlet>
```

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

Angular

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

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

@Component({
  selector: 'app-pere',
  templateUrl: './pere.component.html',
  styleUrls: ['./pere.component.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';
  constructor() { }
  ngOnInit() { }

  ngAfterContentInit(): void {
    console.log(this.titre.valeur);
  }
}
```

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

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.component.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.component.html`

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

Angular

Le fichier `child.component.ts`

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

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

  constructor() { }

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

Angular

Le fichier `parent.component.ts`

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

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

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

Angular

Le fichier `parent.component.html`

```
<h3> Moyenne de mes enfants {{ moyenne }} </h3>
<app-child *ngFor="let enfant of enfants" [nom]="
   enfant" (message)="computeAvg($event)">
</app-child>
```

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 **clavier-touche**

- Considérons un composant `clavier` dont la classe associée a un attribut `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'] .`
- Créez un composant fils nommé `touche`.
- Le composant `touche` a un attribut `value` recevant sa valeur du tableau `lettres` défini dans le composant parent `clavier`.
- Le nombre de composants `touche` = taille 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 `div` définie dans le composant `clavier`.

Angular

Exercice **panier-article**

- Créez deux composant `panier` et `article`
- Définissez une route `/panier` pour le composant `panier`
- Créez une interface avec les attributs `id`, `désignation` et `prixUnitaire`
- Dans `panier.component.ts`, on considère le tableau `articles` donné dans la slide suivante
- Dans `panier.component.html`, on affiche chaque élément de `articles` dans un composant `article`
- Chaque composant `article` contient un champ texte pour saisir une quantité et un bouton qui permet d'envoyer l'article et la quantité au composant `panier`
- Chaque fois qu'on clique sur le bouton `ajouter` d'un article, le composant `panier` affiche `le total`

Angular

Contenu du tableau `articles`

```
articles: Array < Article > =[
  { id: 1, description: 'clavier', prixUnitaire: 20 },
  { id: 2, description: 'souris', prixUnitaire: 10 },
  { id: 3, description: 'écran', prixUnitaire: 200 },
  { id: 4, description: 'modem', prixUnitaire: 30 },
];
```

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

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

Dans `app.component.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.component.html',
  styleUrls: ['./app.component.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.component.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.component.html`
- Définissons une route `/container` pour le composant `container`

Contenu de `app.component.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>();
  constructor() { }

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

Angular

Le fichier `first.component.html`

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

Angular

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

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

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

  msg: string;

  constructor(private messageService: MessageService) { }
  ngOnInit() { }

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

Angular

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

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

Dans `second.component.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 { MessageService } from '../services/message.service';
import { Subscription } from 'rxjs';

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

  messages = [];
  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