

ChatGPT

Chào bạn, tôi đang theo lộ trình học Angular từ cơ bản tới nâng cao. Bạn đã dạy tôi các bài sau đây: Bài 1: Giới thiệu Angular và Cài đặt Môi trường Bài 2: Data Binding trong Angular Bài 3: Component Communication trong Angular Bài 4: Angular Routing (Điều hướng giữa các Component) Bài 5: Reactive Forms trong Angular Bài 6: Gọi API trong Angular với HttpClient Bài 7: Quản lý trạng thái trong Angular với NgRx (State Management) Bài 8: Angular Services và Dependency Injection Bài 9: Lifecycle Hooks trong Angular Hãy tiếp tục lộ trình trên.

Tiếp theo lộ trình học Angular, chúng ta sẽ đến với:

Bài 10: Custom Directives trong Angular

Mục tiêu: Hiểu cách tạo và sử dụng các Directives tùy chỉnh để mở rộng khả năng của Angular.

1. Directives trong Angular là gì?

Directives là các chỉ dẫn được sử dụng để thay đổi hành vi hoặc giao diện của một phần tử trong DOM. Angular cung cấp ba loại directives chính:

1. **Component Directives:** Thực chất là một dạng đặc biệt của directive có template (HTML).

2. **Attribute Directives:** Dùng để thay đổi hoặc mở rộng thuộc tính của một phần tử, ví dụ như `ngClass`, `ngStyle`.
 3. **Structural Directives:** Ảnh hưởng đến cấu trúc DOM, ví dụ `*ngIf`, `*ngFor`, `*ngSwitch`.
-

2. Tạo Custom Attribute Directive

Custom Attribute Directive giúp thay đổi thuộc tính hoặc style của phần tử HTML.

Ví dụ: Tạo directive đổi màu nền khi di chuột vào

Bước 1: Tạo một directive bằng Angular CLI:

```
sh
```

```
ng generate directive highlight
```

Lệnh này sẽ tạo file `highlight.directive.ts`.

Bước 2: Chỉnh sửa `highlight.directive.ts`:

typescript

```
import { Directive, ElementRef, HostListener, Input } from '@angular/core';

@Directive({
  selector: '[appHighlight]' // Sử dụng như một attribute
})
export class HighlightDirective {
  @Input() appHighlight = ''; // Nhận màu sắc từ component

  constructor(private el: ElementRef) {}

  @HostListener('mouseenter') onMouseEnter() {
    this.highlight(this.appHighlight || 'yellow');
  }

  @HostListener('mouseleave') onMouseLeave() {
    this.highlight('');
  }

  private highlight(color: string) {
    this.el.nativeElement.style.backgroundColor = color;
  }
}
```

Bước 3: Sử dụng directive trong component:

html

```
<p appHighlight="lightblue">Di chuột vào đây để đổi màu nền!</p>
```

3. Tạo Custom Structural Directive

Custom Structural Directive có thể thêm hoặc xóa phần tử khỏi DOM.

Ví dụ: Ẩn phần tử dựa vào điều kiện

Bước 1: Tạo một directive:

```
sh
```

```
ng generate directive unless
```

Bước 2: Chỉnh sửa `unless.directive.ts`:

```
typescript
```

```
import { Directive, Input, TemplateRef, ViewContainerRef } from
 '@angular/core';

@Directive({
  selector: '[appUnless]'
})
export class UnlessDirective {
  @Input() set appUnless(condition: boolean) {
    if (!condition) {
      this.viewContainer.createEmbeddedView(this.templateRef);
    } else {
      this.viewContainer.clear();
    }
  }

  constructor(private templateRef: TemplateRef<any>, private viewContainer:
    ViewContainerRef) {}
}
```

Bước 3: Sử dụng directive trong component:

html

```
<p *appUnless="true">Đoạn này sẽ không hiển thị vì điều kiện là true.</p>
```

4. Tổng kết

- **Attribute Directive:** Thay đổi thuộc tính hoặc style của phần tử (ví dụ: đổi màu nền khi di chuột).
- **Structural Directive:** Thay đổi cấu trúc DOM (ví dụ: hiển thị/ẩn phần tử dựa trên điều kiện).
- Sử dụng **ElementRef** để thao tác với DOM, **HostListener** để lắng nghe sự kiện, và **Input** để nhận giá trị từ component.

Bạn có muốn thực hành thêm bài tập không? 🚀

TemplateRef và ViewContainerRef có tác dụng gì

1. TemplateRef là gì?

TemplateRef đại diện cho một **template trong Angular** (được khai báo bằng **<ng-template>** hoặc một phần tử HTML chứa nội dung động).

Ví dụ sử dụng **TemplateRef**

html

```
<ng-template #myTemplate>
  <p>Đây là nội dung từ TemplateRef!</p>
</ng-template>
```

Trong TypeScript, bạn có thể tham chiếu đến `TemplateRef` bằng cách sử dụng `@ViewChild`:

typescript

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

@Component({
  selector: 'app-example',
  template: `
    <ng-template #myTemplate>
      <p>Đây là nội dung từ TemplateRef!</p>
    </ng-template>
  `
})
export class ExampleComponent {
  @ViewChild('myTemplate') template!: TemplateRef<any>;

  constructor() {}

  ngAfterViewInit() {
    console.log(this.template); // TemplateRef {elementRef: ...}
  }
}
```

👉 Công dụng chính:

Dùng để truy cập nội dung của một `<ng-template>` và chèn nó vào DOM.

2. ViewContainerRef là gì?

`ViewContainerRef` là một API cho phép thêm hoặc xóa nội dung từ DOM **động** trong Angular.

Ví dụ sử dụng `ViewContainerRef`

typescript

```
import { Component, ViewChild, TemplateRef, ViewContainerRef } from
 '@angular/core';

@Component({
  selector: 'app-example',
  template: `
    <button (click)="toggleTemplate()">Hiện thị / Ẩn Template</button>
    <ng-template #myTemplate>
      <p>Nội dung động được chèn vào ViewContainerRef!</p>
    </ng-template>
  `
})
export class ExampleComponent {
  @ViewChild('myTemplate') template!: TemplateRef<any>;
  constructor(private viewContainer: ViewContainerRef) {}

  toggleTemplate() {
    if (this.viewContainer.length > 0) {
      this.viewContainer.clear(); // Xóa tất cả nội dung bên trong
    } else {
      this.viewContainer.createEmbeddedView(this.template); // Thêm template
      vào DOM
    }
  }
}
```

👉 Công dụng chính:

- Dùng để thêm hoặc xóa một **TemplateRef** vào DOM.
- Cho phép quản lý các views động, giúp tối ưu hiệu suất.

Tóm lại:

Đối tượng	Mô tả	Công dụng chính
TemplateRef	Đại diện cho một <ng-template> trong Angular.	Lưu giữ nội dung template để có thể render lại sau.
ViewContainerRef	Đại diện cho một vùng chứa (container) trong DOM.	Chèn hoặc xóa nội dung TemplateRef vào DOM.

📌 Ứng dụng thực tế của **TemplateRef** và **ViewContainerRef**:

- Dùng trong **Structural Directives** (*ngIf, *ngFor, *ngSwitch).
- Tạo các component **động** (Dynamic Component).
- Xây dựng hệ thống **lazy loading**.

Bạn có muốn một bài tập thực hành về chủ đề này không? 🚀

@ViewChild có tác dụng gì

@ViewChild trong Angular là gì?

@ViewChild là một **decorator** trong Angular dùng để lấy tham chiếu đến một phần tử DOM, một directive, hoặc một component con trong template của component cha.

1. Cách sử dụng @ViewChild

Ví dụ 1: Truy xuất phần tử DOM

Bạn có thể dùng @ViewChild để truy cập một phần tử HTML trong template:

typescript

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

@Component({
  selector: 'app-example',
  template: `<p #myParagraph>Đoạn văn này sẽ bị thay đổi.</p>
    <button (click)="changeText()">Thay đổi nội dung</button>`
})
export class ExampleComponent implements AfterViewInit {
  @ViewChild('myParagraph') paragraph!: ElementRef;

  ngAfterViewInit() {
    console.log(this.paragraph.nativeElement.textContent); // In nội dung
    ban đầu
  }

  changeText() {
    this.paragraph.nativeElement.textContent = 'Nội dung đã thay đổi!';
  }
}
```


Giải thích:

- `@ViewChild('myParagraph') paragraph!: ElementRef;` → Lấy phần tử `<p>` có `#myParagraph`.
- `ElementRef` giúp truy cập trực tiếp vào `nativeElement` của DOM.
- `ngAfterViewInit()` đảm bảo rằng phần tử đã sẵn sàng trước khi truy cập.

Ví dụ 2: Truy xuất Component con

Bạn có thể dùng `@ViewChild` để lấy một component con trong template.

Component con (`ChildComponent`)

typescript

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

@Component({
  selector: 'app-child',
  template: `<p>Component con!</p>`
})
export class ChildComponent {
  sayHello() {
    return 'Xin chào từ component con!';
  }
}
```

Component cha (`ParentComponent`)

typescript

```
import { Component, ViewChild, AfterViewInit } from '@angular/core';
import { ChildComponent } from './child.component';

@Component({
  selector: 'app-parent',
  template: `<app-child></app-child>
    <button (click)="callChildMethod()">Gọi phương thức
con</button>`
})
export class ParentComponent implements AfterViewInit {
  @ViewChild(ChildComponent) childComponent!: ChildComponent;

  ngAfterViewInit() {
    console.log(this.childComponent.sayHello()); // "Xin chào từ component
con!"
  }

  callChildMethod() {
    alert(this.childComponent.sayHello());
  }
}
```

Giải thích:

- `@ViewChild(ChildComponent) childComponent!: ChildComponent;` → Lấy component con.
- Có thể gọi phương thức của component con từ component cha.

2. Khi nào `@ViewChild` hoạt động?

- `@ViewChild` chỉ truy xuất được phần tử sau khi view đã khởi tạo, do đó, **bạn cần sử dụng `ngAfterViewInit()`** để truy cập nó một cách an toàn.
- Nếu bạn cần cập nhật `@ViewChild` sau mỗi thay đổi trong template, hãy sử dụng `ngAfterViewChecked()`.

3. Khi nào sử dụng `@ViewChild`?

- Khi **cần truy cập hoặc thao tác DOM** trong một component.
- Khi cần **gọi phương thức hoặc lấy dữ liệu** từ component con.

- Khi muốn tối ưu **hiệu suất**, tránh sử dụng `document.querySelector()` trong Angular.

4. Sự khác biệt giữa `@ViewChild` và `@ContentChild`

Decorator	Truy xuất cái gì?	Khi nào sử dụng?
<code>@ViewChild</code>	Truy cập phần tử DOM, directive, hoặc component con bên trong template của component cha .	Khi cần thao tác trên DOM hoặc gọi phương thức của component con.
<code>@ContentChild</code>	Truy cập nội dung <code><ng-content></code> được truyền vào component cha từ bên ngoài.	Khi cần thao tác với nội dung ng-content .

Tóm tắt

- ✓ `@ViewChild` giúp truy cập phần tử, directive, hoặc component con trong **template**.
 - ✓ Chỉ có thể truy xuất giá trị sau khi Angular render xong (`ngAfterViewInit`).
 - ✓ Hữu ích khi cần thao tác trực tiếp với DOM hoặc component con.
-

📌 Bạn có muốn thực hành bài tập nhỏ để áp dụng `@ViewChild` không? 🚀

ChatGPT can make mistakes. Check important info.