**@ViewChild 설명**

@ViewChild는 Angular에서 부모 컴포넌트가 자식 컴포넌트, 디렉티브, 또는 DOM 엘리먼트를 직접 참조할 수 있도록 해주는 데코레이터입니다. 이를 통해 자식의 **속성이나 메서드**에 접근하거나 조작할 수 있습니다.

**@ViewChild 사용법**

@ViewChild는 보통 다음과 같은 방법으로 사용됩니다.

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

@Component({

selector: 'app-parent',

template: `

<app-child></app-child>

<div #myDiv>안녕하세요!</div>

`,

})

export class ParentComponent {

@ViewChild(ChildComponent) childComponent: ChildComponent;

@ViewChild('myDiv') myDiv: ElementRef;

ngAfterViewInit() {

// 자식 컴포넌트의 메서드 호출

this.childComponent.doSomething();

// DOM 엘리먼트의 속성 변경

this.myDiv.nativeElement.textContent = '반갑습니다!';

}

}

* @ViewChild(ChildComponent) childComponent: ChildComponent;: ChildComponent 타입의 자식 컴포넌트를 참조합니다.
* @ViewChild('myDiv') myDiv: ElementRef;: 템플릿에서 #myDiv로 참조된 DOM 엘리먼트를 ElementRef 타입으로 가져옵니다.

**@ViewChild의 주요 속성**

@ViewChild 데코레이터는 두 번째 인자로 옵션 객체를 받습니다. 자주 사용되는 속성들은 다음과 같습니다.

* { static: boolean }: 이 속성은 @ViewChild가 뷰 초기화 시점에 바로 해결될지 (true), 아니면 뷰의 변경 감지 주기 이후에 해결될지 (false)를 결정합니다.
  + static: true: 참조하려는 엘리먼트가 ngIf와 같은 조건부 렌더링에 영향을 받지 않고 **항상 DOM에 존재하는 경우**에 사용합니다. 이 경우 ngOnInit에서 접근 가능합니다.
  + static: false: 참조하려는 엘리먼트가 ngIf, ngFor 등 **조건부로 렌더링 되는 경우**에 사용합니다. 이 경우 ngAfterViewInit에서 접근해야 합니다.

Angular 8부터는 static 속성을 명시적으로 설정해야 합니다.

* { read: any }: 이 속성은 참조하려는 엘리먼트에서 **어떤 타입을 읽어올지** 명시합니다. 예를 들어, 자식 컴포넌트의 인스턴스가 아닌 TemplateRef나 ViewContainerRef 등을 가져오고 싶을 때 사용합니다.

TypeScript

@ViewChild(ChildComponent, { read: ViewContainerRef })

private viewContainerRef: ViewContainerRef;

**@ViewChild vs. @ContentChild**

이 둘은 혼동하기 쉬우므로 차이를 명확히 이해하는 것이 중요합니다.

* @ViewChild: 컴포넌트의 **템플릿**에 선언된 **뷰(View)** 내의 엘리먼트를 참조합니다. 즉, 부모 컴포넌트의 .html 파일 내에 있는 자식 컴포넌트, 디렉티브, 또는 DOM 엘리먼트를 가져옵니다.
* @ContentChild: 컴포넌트의 <ng-content>를 통해 **외부에서 주입된 콘텐츠**를 참조합니다. <ng-content>를 사용하는 컴포넌트 내부에 **외부로부터 전달된 콘텐츠**를 가져올 때 사용합니다.

쉽게 말해, @ViewChild는 "내 템플릿에 있는 자식", @ContentChild는 "나에게 전달된 외부 콘텐츠"를 가져오는 데 사용됩니다.