Configuração `static` do `@ViewChild`

O decorador `@ViewChild` é utilizado no Angular para acessar elementos HTML ou componentes filhos no template. A configuração `static` é um parâmetro opcional que determina quando o Angular deve resolver a referência ao elemento durante o ciclo de vida do componente.

Valores para `static`:

1. **`static: true`**:
    - A referência ao elemento será resolvida **durante a fase `ngOnInit`**, antes de o conteúdo ser renderizado.
    - Geralmente usado quando o elemento não depende de diretivas condicionais como `*ngIf` (ou seja, o elemento sempre está presente no DOM).
2. **`static: false`**:
    - A referência será resolvida **durante a fase `ngAfterViewInit`**, ou seja, após a view ser completamente inicializada.
    - Deve ser usado quando o elemento pode ser renderizado condicionalmente (ex.: dentro de um `*ngIf`).

Exemplo 1: Elemento com `static: true`

O elemento HTML existe no template **desde o início** e não é afetado por diretivas condicionais.

```
@Component({
  selector: 'app-root',
  template: `<input #meuInput type="text">`
})
export class AppComponent implements OnInit {
  @ViewChild('meuInput', { static: true })
  meuInputEl!: ElementRef<HTMLInputElement>;

  ngOnInit() {
    console.log('ngOnInit:', this.meuInputEl); // Elemento já disponível.
    this.meuInputEl.nativeElement.focus(); // Coloca o foco no input.
  }
}
```

- **Resultado:** O foco será definido no campo de texto durante a fase `ngOnInit`.

Exemplo 2: Elemento com `static: false`

O elemento HTML é adicionado condicionalmente usando uma diretiva como `*ngIf`.

```
@Component({
  selector: 'app-root',
  template: `
    <div *ngIf="mostrar">
      <input #meuInput type="text">
    </div>
    <button (click)="mostrar = true">Exibir Input</button>
  `
})
export class AppComponent implements AfterViewInit {
  @ViewChild('meuInput', { static: false })
  meuInputEl!: ElementRef<HTMLInputElement>;

  mostrar = false;

  ngAfterViewInit() {
    if (this.meuInputEl) {
      this.meuInputEl.nativeElement.focus(); // Será chamado apenas quando o input existir.
    }
  }
}
```

- **Resultado:** O elemento será referenciado após o botão ser clicado e o input for exibido.

Exemplo 3: Acessar componente filho com `static: true`

Você pode usar `@ViewChild` para acessar métodos ou propriedades de um componente filho.

Componente filho:

```
@Component({
  selector: 'app-filho',
  template: `<p>Sou o componente filho</p>`
})
export class FilhoComponent {
```

```
  dizerOi() {
    console.log('Oi do filho!');
  }
}
```

Componente pai:

```
@Component({
  selector: 'app-root',
  template: `<app-filho #filhoComp></app-filho>`
})
export class AppComponent implements OnInit {
  @ViewChild('filhoComp', { static: true })
  filhoCompRef!: FilhoComponent;

  ngOnInit() {
    this.filhoCompRef.dizerOi(); // Acessa o método do componente filho durante o OnInit.
  }
}
```

- **Resultado:** O método `dizerOi` será chamado quando o componente pai for inicializado.

Exemplo 4: Manipulando listas de elementos com `ViewChildren`

Quando você precisa manipular múltiplos elementos ou componentes filhos, pode usar `ViewChildren`. O parâmetro `static` **não se aplica** neste caso, porque os elementos são resolvidos após a view ser renderizada.

```
@Component({
  selector: 'app-root',
  template: `
    <div *ngFor="let item of items">
      <p #paragrafo>{{ item }}</p>
    </div>
  `
})
export class AppComponent implements AfterViewInit {
  @ViewChildren('paragrafo')
  paragrafos!: QueryList<ElementRef<HTMLParagraphElement>>;

  items = ['Item 1', 'Item 2', 'Item 3'];

  ngAfterViewInit() {
```

```
    this.paragrafos.forEach((p, index) => {
      p.nativeElement.style.color = index % 2 === 0 ? 'blue' : 'green'; // Altera a cor de cada parágrafo.
    });
  }
}
```

- **Resultado:** Cada parágrafo terá uma cor alternada entre azul e verde após ser renderizado.

Quando usar `static: true` ou `false`

Situação	Configuração
Elemento sempre existe no DOM.	`static: true`
Elemento é condicionado por `*ngIf`.	`static: false`
Elemento é criado ou removido dinamicamente.	`static: false`
Componente filho está sempre presente.	`static: true`
Componente filho é condicional.	`static: false`

O que pode ser feito com `@ViewChild`

1. **Manipular elementos HTML diretamente:**
    - Alterar texto, atributos ou estilos.

```
this.meuInputEl.nativeElement.value = 'Novo Valor';
this.meuInputEl.nativeElement.style.backgroundColor = 'yellow';
```

2.
3. **Acessar e controlar componentes filhos:**
    - Invocar métodos ou modificar propriedades do componente filho.

```
this.filhoCompRef.dizerOi();
this.filhoCompRef.message = 'Mensagem Alterada';
```

4. **Usar elementos para capturar eventos:**
    - Vincular eventos ou ouvir mudanças diretamente no DOM.

```
this.meuInputEl.nativeElement.addEventListener('input', (event) => {
  console.log('Input alterado:', event.target.value);
});
```

5. **Acessar elementos dinâmicos em listas:**
    - Usar `ViewChildren` para interagir com múltiplos elementos.
