No Angular, o decorador `@ViewChild` permite acessar um componente filho, elementos HTML ou diretivas presentes no template do componente pai. Ele é amplamente usado para manipular diretamente propriedades ou métodos de componentes filhos ou para interagir com elementos do DOM.

A seguir, apresento exemplos e casos de uso detalhados do `@ViewChild`:

---

1. Acessar Componentes Filhos

Exemplo Básico: Chamar Métodos do Filho

Template do Pai (`app.component.html`)

```
<app-filho #filhoComp></app-filho>
<!-- Referência ao componente filho com a variável de template #filhoComp
-->

<button (click)="saudarFilho()">Saudar Filho</button>
<!-- Botão para chamar o método no filho -->
```

Código do Filho (`filho.component.ts`)

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

@Component({
  selector: 'app-filho',
  template: `<p>{{message}}</p>`,
})
export class FilhoComponent {
  message = 'Olá do Componente Filho!';

  saudar() {
    alert('Olá do Componente Filho!');
  }
}
```

Código do Pai (`app.component.ts`)

```
import { Component, ViewChild } from '@angular/core';
import { FilhoComponent } from './filho/filho.component';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {
  @ViewChild('filhoComp') filhoComp!: FilhoComponent;

  saudarFilho() {
    this.filhoComp.saudar();
    this.filhoComp.message = 'Mensagem atualizada pelo Pai!';
  }
}
```

O que acontece:

- O botão no pai chama o método `saudarFilho`, que:
  - Executa o método `saudar()` no componente filho.
  - Atualiza a propriedade `message` no filho.

---

2. Acessar Elementos HTML

Exemplo: Manipular um Elemento HTML Diretamente

Template do Pai (`app.component.html`)

```
<div #minhaDiv>
  Conteúdo Inicial
</div>

<button (click)="alterarTexto()">Alterar Texto</button>
```

Código do Pai (`app.component.ts`)

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

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {
  @ViewChild('minhaDiv') minhaDiv!: ElementRef<HTMLDivElement>;

  alterarTexto() {
    this.minhaDiv.nativeElement.textContent = 'Texto Alterado!';
    this.minhaDiv.nativeElement.style.color = 'blue';
  }
}
```

O que acontece:

- A função `alterarTexto` altera o conteúdo e o estilo do elemento `div` referenciado.

---

3. Acessar Diretivas Personalizadas

Exemplo: Interagir com Diretivas Usando `@ViewChild`

Template (`app.component.html`)

```
<p highlight #highlightDir="highlightDirective">Texto com diretiva</p>
<button (click)="mudarCor()">Mudar Cor</button>
```

Diretiva Personalizada (`highlight.directive.ts`)

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

@Directive({
  selector: '[highlight]',
  exportAs: 'highlightDirective',
})
export class HighlightDirective {
  constructor(private el: ElementRef) {}
```

```
  mudarCor(cor: string) {
    this.el.nativeElement.style.backgroundColor = cor;
  }
}
```

Código do Pai (`app.component.ts`)

```
import { Component, ViewChild } from '@angular/core';
import { HighlightDirective } from './highlight.directive';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {
  @ViewChild('highlightDir') highlightDirective!: HighlightDirective;

  mudarCor() {
    this.highlightDirective.mudarCor('yellow');
  }
}
```

O que acontece:

- A diretiva `highlight` é acessada e manipulada diretamente pelo componente pai.

---

4. Acessar Vários Elementos ou Componentes

Exemplo: Usar `@ViewChildren` para Acessar uma Lista de Elementos

Template (`app.component.html`)

```
<p #paragrafo>Texto 1</p>
<p #paragrafo>Texto 2</p>
```

```
<p #paragrafo>Texto 3</p>

<button (click)="alterarParagrafos()">Alterar Textos</button>
```

Código do Pai (`app.component.ts`)

```
import { Component, QueryList, ViewChildren, ElementRef } from
'@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {
  @ViewChildren('paragrafo') paragrafos!: QueryList<ElementRef>;

  alterarParagrafos() {
    this.paragrafos.forEach((paragrafo, index) => {
      paragrafo.nativeElement.textContent = `Texto Alterado ${index + 1}`;
    });
  }
}
```

O que acontece:

- O método `alterarParagrafos` percorre todos os elementos referenciados com `#paragrafo` e altera o texto de cada um.

---

5. Acessar Componentes Aninhados

Exemplo: Interagir com Componentes em Níveis Diferentes

Template do Pai (`app.component.html`)

```
<app-filho #filhoComp>
  <app-neto #netoComp></app-neto>
</app-filho>
```

```
<button (click)="interagirComNeto()">Interagir com Neto</button>
```

Código do Neto (`neto.component.ts`)

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

@Component({
  selector: 'app-neto',
  template: `<p>Sou o Neto!</p>`,
})
export class NetoComponent {
  mensagem = 'Sou o componente Neto';

  mostrarMensagem() {
    alert(this.mensagem);
  }
}
```

Código do Filho (`filho.component.ts`)

```
import { Component, ViewChild } from '@angular/core';
import { NetoComponent } from './neto/neto.component';

@Component({
  selector: 'app-filho',
  template: `<ng-content></ng-content>`,
})
export class FilhoComponent {
  @ViewChild('netoComp') netoCompRef!: NetoComponent;
}
```

Código do Pai (`app.component.ts`)

```
import { Component, ViewChild } from '@angular/core';
import { FilhoComponent } from './filho/filho.component';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
```

```
})
export class AppComponent {
  @ViewChild('filhoComp') filhoCompRef!: FilhoComponent;

  interagirComNeto() {
    this.filhoCompRef.netoCompRef.mostrarMensagem();
  }
}
```

O que acontece:

- O componente pai acessa o neto através da referência no filho.

Resumo do que Pode ser Feito com `@ViewChild`

1. **Manipular Propriedades e Métodos de Componentes Filhos**:
    - Chamando métodos e alterando propriedades diretamente.
2. **Interagir com Elementos HTML**:
    - Modificar o DOM diretamente.
3. **Acessar Diretivas Personalizadas**:
    - Trabalhar com diretivas no template.
4. **Trabalhar com Listas de Elementos ou Componentes**:
    - Usar `@ViewChildren` para manipular múltiplos elementos.
5. **Acessar Componentes Aninhados**:
    - Atravessar níveis de hierarquia para interagir com componentes netos ou bisnetos.

Essas funcionalidades tornam o `@ViewChild` uma ferramenta essencial para comunicação e manipulação entre componentes no Angular.