Aqui estão exemplos detalhados de como acessar elementos HTML em um componente Angular usando o decorador `@ViewChild`, incluindo casos comuns e o que pode ser feito:

O que é `@ViewChild`?

O `@ViewChild` é um decorador Angular usado para obter uma referência direta a um elemento HTML ou um componente filho no template. Ele permite manipular diretamente elementos DOM ou acessar métodos/atributos de componentes aninhados.

Exemplo 1: Acessar e Alterar um Campo de Entrada

Template:

```
<input type="text" #meuInput placeholder="Digite algo">
<button (click)="focusInput()">Colocar Foco</button>
<button (click)="updateInputValue()">Alterar Valor</button>
```

TypeScript:

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  @ViewChild('meuInput') meuInputEl!: ElementRef<HTMLInputElement>;

  // Define o foco no campo de entrada
  focusInput() {
    this.meuInputEl.nativeElement.focus();
  }

  // Altera o valor do campo de entrada
  updateInputValue() {
```

```
    this.meuInputEl.nativeElement.value = 'Valor Atualizado!';
  }
}
```

Exemplo 2: Alterar o Conteúdo de uma Div

Template:

```
<div #minhaDiv>Texto Original</div>
<button (click)="changeDivContent()">Alterar Conteúdo</button>
```

TypeScript:

```
@ViewChild('minhaDiv') minhaDivEl!: ElementRef<HTMLDivElement>;

changeDivContent() {
  this.minhaDivEl.nativeElement.textContent = 'Novo Conteúdo!';
}
```

Exemplo 3: Alterar o Estilo de um Elemento

Template:

```
<p #paragrafo>Este é um parágrafo estilizado.</p>
<button (click)="applyStyles()">Aplicar Estilo</button>
```

TypeScript:

```
@ViewChild('paragrafo') paragrafoEl!: ElementRef<HTMLParagraphElement>;

applyStyles() {
  const elemento = this.paragrafoEl.nativeElement;
  elemento.style.color = 'blue';
  elemento.style.fontSize = '20px';
  elemento.style.fontWeight = 'bold';
}
```

Exemplo 4: Acessar um Componente Filho

Template:

```
<app-child #child></app-child>
<button (click)="callChildMethod()">Chamar Método do Filho</button>
```

Componente Filho:

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

@Component({
  selector: 'app-child',
  template: `<p>Componente Filho</p>`,
})
export class ChildComponent {
  showAlert() {
    alert('Método do componente filho chamado!');
  }
}
```

TypeScript do Componente Pai:

```
@ViewChild('child') childComponent!: ChildComponent;

callChildMethod() {
  this.childComponent.showAlert();
}
```

Exemplo 5: Manipular Atributos Dinamicamente

Template:

```
<input type="text" #dynamicInput placeholder="Digite algo...">
<button (click)="disableInput()">Desativar Campo</button>
```

TypeScript:

```
@ViewChild('dynamicInput') dynamicInputEl!: ElementRef<HTMLInputElement>;

disableInput() {
```

```
  this.dynamicInputEl.nativeElement.disabled = true;
}
```

Exemplo 6: Detectar Alterações com `AfterViewInit`

Quando o elemento pode não estar disponível no momento da inicialização do componente, use o ciclo de vida `AfterViewInit`.

Template:

```
<div #delayedDiv>Texto inicial...</div>
```

TypeScript:

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent implements AfterViewInit {
  @ViewChild('delayedDiv') delayedDivEl!: ElementRef<HTMLDivElement>;

  ngAfterViewInit() {
    this.delayedDivEl.nativeElement.textContent = 'Conteúdo atualizado após inicialização!';
  }
}
```

Exemplo 7: Controlar a Visibilidade de um Elemento

Template:

```
<div #toggleDiv>Texto visível</div>
<button (click)="toggleVisibility()">Mostrar/Esconder</button>
```

TypeScript:

```
@ViewChild('toggleDiv') toggleDivEl!: ElementRef<HTMLDivElement>;
visible: boolean = true;

toggleVisibility() {
  this.visible = !this.visible;
  this.toggleDivEl.nativeElement.style.display = this.visible ? 'block' : 'none';
}
```

Exemplo 8: Passar Elementos como Parâmetro

Template:

```
<input #inputA placeholder="Campo A">
<input #inputB placeholder="Campo B">
<button (click)="swapValues(inputA, inputB)">Trocar Valores</button>
```

TypeScript:

```
swapValues(elA: ElementRef<HTMLInputElement>, elB:
ElementRef<HTMLInputElement>) {
  const temp = elA.nativeElement.value;
  elA.nativeElement.value = elB.nativeElement.value;
  elB.nativeElement.value = temp;
}
```

O que se Pode Fazer com `@ViewChild`

1. **Manipular diretamente elementos do DOM**:
    - Alterar valores, texto ou estilos.
    - Controlar visibilidade.
    - Adicionar/remover classes.
2. **Acessar métodos e propriedades de componentes filhos**:
    - Chamar métodos.
    - Acessar variáveis e estados.

3. **Interagir dinamicamente com elementos**:
    - Atualizar campos dinamicamente.
    - Passar elementos entre funções como parâmetros.
4. **Detectar e manipular elementos no ciclo de vida**:
    - Usar `ngAfterViewInit` para executar ações após a inicialização dos elementos.
5. **Controlar atributos do HTML**:
    - Desativar, habilitar ou adicionar atributos (ex.: `disabled`, `readonly`).

Com esses exemplos, é possível ver a versatilidade do `@ViewChild` para interagir com elementos do DOM e componentes filhos. Isso oferece muito controle e flexibilidade no desenvolvimento de aplicações Angular!