`<ng-container>` vs `<ng-template>` no Angular

No Angular, **`<ng-container>`** e **`<ng-template>`** são elementos estruturais usados para manipular a renderização de componentes e elementos na DOM. Embora sejam parecidos, eles têm propósitos diferentes. Vamos entender a diferença e ver exemplos práticos!

1️⃣ `<ng-container>` - Um Contêiner Virtual

O **`<ng-container>`** é um **contêiner invisível** que **não gera um elemento real na DOM**. Ele é útil quando queremos aplicar diretivas estruturais (`*ngIf`, `*ngFor`, etc.) sem adicionar um novo nó HTML.

📌 Exemplo: Usando `<ng-container>` para evitar elementos desnecessários

Sem `<ng-container>`:

```
<div *ngIf="mostrar">
  <p>Este parágrafo só aparece se mostrar for verdadeiro.</p>
</div>
```

Aqui, o **`<div>`** será renderizado mesmo que não seja necessário.

Com `<ng-container>`:

```
<ng-container *ngIf="mostrar">
  <p>Este parágrafo só aparece se mostrar for verdadeiro.</p>
</ng-container>
```

Agora, **nenhum elemento extra será adicionado à DOM**! Apenas o `<p>` será renderizado quando `mostrar` for `true`.

📌 Exemplo: Melhorando o `*ngFor` com `<ng-container>`

Imagine que queremos exibir uma lista de itens, mas sem um elemento extra para cada um.

Sem `<ng-container>`:

```
<ul>
  <li *ngFor="let item of lista">
    {{ item }}
  </li>
</ul>
```

Aqui, o `<li>` será repetido corretamente, mas se tivéssemos um `<div>` no lugar, ele seria renderizado junto.

Com `<ng-container>`:

```
<ul>
  <ng-container *ngFor="let item of lista">
    <li>{{ item }}</li>
  </ng-container>
</ul>
```

Agora, o `<ng-container>` **não aparece na DOM**, e apenas os `<li>` são renderizados!

2️⃣ `<ng-template>` - Um Modelo de Template

O **`<ng-template>`** cria um **modelo de conteúdo que só será renderizado quando for explicitamente chamado**. Ele é útil para: ✅ Criar templates reutilizáveis
✅ Trabalhar com `ngIf` e `ngFor` sem renderizar elementos desnecessários
✅ Renderizar conteúdo dinamicamente

📌 Exemplo: Usando `<ng-template>` com `ngIf`

Sem `<ng-template>`:

```
<p *ngIf="mostrar">Esse parágrafo aparece se mostrar for verdadeiro.</p>
```

Com `<ng-template>`:

```
<ng-template #meuTemplate>
  <p>Esse parágrafo só aparece se mostrar for verdadeiro.</p>
</ng-template>

<button (click)="mostrar = !mostrar">Alternar</button>
<div *ngIf="mostrar; else meuTemplate">
  <p>Esse é o conteúdo principal.</p>
</div>
```

Aqui, o Angular **não renderiza o `<ng-template>` na DOM até que ele seja necessário**.

📌 Exemplo: Reutilizando `<ng-template>`

Podemos criar **um template reutilizável** e chamá-lo várias vezes.

```
<ng-template #mensagem let-texto>
  <p>Mensagem: {{ texto }}</p>
</ng-template>

<!-- Renderizando o template manualmente -->
<ng-container *ngTemplateOutlet="mensagem; context: {texto: 'Primeira Mensagem'}"></ng-container>
<ng-container *ngTemplateOutlet="mensagem; context: {texto: 'Segunda Mensagem'}"></ng-container>
```

Isso renderiza:

```
Mensagem: Primeira Mensagem
Mensagem: Segunda Mensagem
```

3️⃣ Diferenças Resumidas

Característica	`<ng-container>`	`<ng-template>`
Gera um elemento na DOM?	❌ Não	❌ Não
Usado para evitar elementos desnecessários?	✅ Sim	❌ Não
Usado para armazenar um template reutilizável?	❌ Não	✅ Sim
Funciona diretamente com `*ngIf` e `*ngFor`?	✅ Sim	❌ Não
Requer um método ou referência para exibição?	❌ Não	✅ Sim (`*ngTemplateOutlet`)

4️⃣ Como combinar `<ng-container>` e `<ng-template>`?

Podemos usá-los juntos para melhorar a organização do código.

📌 Exemplo: Criando uma Lista com Mensagem Personalizada

```
<ng-container *ngIf="lista.length > 0; else semItens">
  <ul>
    <ng-container *ngFor="let item of lista">
      <li>{{ item }}</li>
    </ng-container>
  </ul>
</ng-container>

<!-- Template para quando a lista estiver vazia -->
<ng-template #semItens>
  <p>Não há itens na lista.</p>
</ng-template>
```

✔️ **Se houver itens, ele os exibe.**
✔️ **Se a lista estiver vazia, o template `semItens` será mostrado.**

Conclusão

- **`<ng-container>`** é útil para **evitar elementos extras na DOM** ao usar `*ngIf`, `*ngFor` e outras diretivas.
- **`<ng-template>`** permite **armazenar templates reutilizáveis** que podem ser renderizados dinamicamente.
- **Juntos, eles ajudam a criar interfaces mais eficientes e organizadas**.

🚀 Agora você sabe quando usar `<ng-container>` e `<ng-template>` no Angular!