How to Use ngTemplateOutlet in Angular

10 Comments / 7 minutes of reading / March 9, 2023

⟵ ngTemplate Angular Tutorial HostBinding & HostListner ⟶

ngTemplateOutlet is a directive. It instantiates a template dynamically using a template reference and context object as parameters. In this guide, we will learn how to use it in Angular. We will show you several ngTemplateOutlet examples to learn from.

Table of Contents

What is ngTemplateOutlet?

ngTemplateOutlet is a structural directive. We use it to insert a template (created by ngTemplate) in various sections of our DOM. For example, you can define a few templates to display an item and use them display at several places in the View and also swap that template as per the user's choice.

How to use ngTemplateOutlet

First let us see a very simple example of `ngTemplateOutlet`.

In the following code, we have a template defined using the `ng-template`. The Template reference variable holds the reference the template. (`TemplateRef`).

The template does not render itself. We must use a structural directive to render it. That is what `ngTemplateOutlet` does

We pass the Template Reference to the `ngTemplateOutlet` directive. It renders the template. Any inner content that `ngTemplateOutlet` encloses will not be rendered.

```

<h3>Example 1</h3>

<!--  This is our template. template1 is the template reference variable holds the referenc
  template1 is of type TemplateRef This won't be rendered here -->


<ng-template #template1>
  <p>This is our template. It will be displayed on the div *ngTemplateOutlet="myTemplate
</ng-template>

<p>The Template appears after this</p>


<!--
  We want to render the above template here.
  Hence we use the ngTemplateOutlet directive
  Assign template1 to ngTemplateOutlet
-->


<ng-container *ngTemplateOutlet="template1">
```

```
  This text is not displayed
</ng-container>


<!--
  Use can use any element.
  Here we use div instead of ng-container
  Div is not rendered
-->


<div *ngTemplateOutlet="template1">
</div>

```

The following code does not render the div.

```

<div *ngTemplateOutlet="template1">
</div>

```

i.e because the angular converts the above into the following ng-template syntax. The ngTemplateOutlet replaces everything inside the ng-template element and renders the template pointed by template1

```

<ng-template [ngTemplateOutlet]="template1">
  <div></div>
</ng-template>

```

The above use case is a simple one. You can achieve the same using a ngIf or ngSwitch directives. You can also make use of content projection using the ngContent.

Passing data to ngTemplateOutlet

We can also pass data to the using its second property ngTemplateOutletContext .

The following code creates a template. We name it as messageTemplate . The let-value creates a local variable with the name value

```

<ng-template let-value="value" #messageTemplate>
   <p>Value Received from the Parent is  {{value}}</p>
</ng-template>

```

We can pass any value to the value using the ngTemplateOutletContext property

```

<ng-container [ngTemplateOutlet]="messageTemplate"
      [ngTemplateOutletContext] ="{value:'1000'}">
</ng-container>

```

Alternatively you can also use the following syntax.

```

<ng-container *ngTemplateOutlet="messageTemplate; context:{value:100}">
</ng-container>

```

Pass more than one value.

```

<ng-template let-name="nameVar" let-message="messageVar" #template3>
  <p>Dear {{name}} , {{message}} </p>
</ng-template>


<ng-container [ngTemplateOutlet]="templates"
        [ngTemplateOutletContext] ="{nameVar:'Guest',messageVar:'Welcome to our site'}'
</ng-container>

```

Pass an object.

```

<ng-template let-person="person"  #template4>
  <p>Dear {{person.name}} , {{person.message}} </p>
</ng-template>


<ng-container [ngTemplateOutlet]="templates"
         [ngTemplateOutletContext] ="{ person:{name:'Guest',message:'Welcome to our sit
</ng-container>

```

Using $implicit

If you use the key `$implicit` in the context object will set its value as default for all the local variables.

For Example we have not assigned anything to the `let-name` so it will take the value from the `$implicit`, which is `Guest`.

```

<ng-template let-name let-message="message" #template3>
  <p>Dear {{name}} , {{message}} </p>
</ng-template>

<ng-container [ngTemplateOutlet]="templates"
           [ngTemplateOutletContext] ="{$implicit:'Guest',message:'Welcome to our site'}">
```

```
</ng-container>

```

And in the following code, both `name` & `message` gets the value from the `$implicit` i.e `Guest`

```

<ng-template let-name let-message #template3>
  <p>Dear {{name}} , {{message}} </p>
</ng-template>


<ng-container [ngTemplateOutlet]="template3"
     [ngTemplateOutletContext] ="{$implicit:'Guest',message:'Welcome to our site'}">
</ng-container>

```

Passing Template to a Child Component

We can pass the entire template to a child component from the parent component. The technique is similar to passing data from parent to child component.

Create a parent component. Add a `ng-template` and name it as `#parentTemplate`.

Pass the `parentTemplate` to the child component using the property binding. (`<child [customTemplate] = "parentTemplate" > </child>`)

```

import { Component, TemplateRef, Input } from '@angular/core';

@Component({
  selector: 'parent',
  template: `

  <h1>Parent component</h1>
  <ng-template #parentTemplate>
    <p>
      This Template is defined in Parent.
    We will send it to child component
  </p>
</ng-template>

<child [customTemplate]="parentTemplate"></child>
`
})
export class ParentComponent {

}

```

In the Child, component receive the `parentTemplate` using the @Input(). And then pass it to `ngTemplateOutlet`.

```

@Component({
  selector: 'child',
  template: `
  <h2>Child component</h2>

  <ng-container *ngTemplateOutlet="customTemplate">
  </ng-container>
  `
})
export class ChildComponent {

  @Input() customTemplate: TemplateRef<HTMLElement>;

}

```

Using ViewChild to Access the template

Use the `ViewChild` to get the access to the `parentTemplate` in the component.

```

import { Component, TemplateRef, Input, OnInit, ViewChild, AfterViewInit } from '@angula

@Component({
  selector: 'parent',
  template: `

  <h1>Parent component</h1>
  <ng-template #parentTemplate>
    <p>
      This Template is defined in Parent.
      We will send it to child component
    </p>
  </ng-template>

  <child [customTemplate]="parentTemplate"></child>

  `
})
export class ParentComponent implements OnInit, AfterViewInit {

  @ViewChild('parentTemplate',null) myTemplate:TemplateRef<HTMLElement>;

  ngAfterViewInit() {
    console.log(this.myTemplate)
  }

}

```

Content Projection and ngTemplate

The content projection and `ngTemplate` can be used together.

The following is the Parent component, which uses the content projection to pass a template to the child.

```

import { Component, TemplateRef, Input, OnInit, ViewChild, AfterViewInit } from '@angula

@Component({
  selector: 'parent1',
  template: `

  <h1>Parent Component </h1>

  <child1>
    <p>This Template is Projected to the Child</p>
  </child1>
  `
})
export class Parent1Component {
}

```

In the child, we add it into a ngTemplate .

```

import { Component, TemplateRef, Input, OnInit, ViewChild, AfterViewInit } from '@angula

@Component({
  selector: 'child1',
  template: `

  <h1>Child Component </h1>

  <ng-template #parentTemplate>
    <ng-content></ng-content>
  </ng-template>

  <ng-template [ngTemplateOutlet]="parentTemplate"></ng-template>

  `
})
export class Child1Component {
}

```

ngTemplateOutlet Example

The application we are going to build will display items either in card or list format.

Create a new application. Open the app.component.html

First, we ask the user Display Mode. He has to choose from the card & list using the select option dropdown.

```

```

```
<label for="mode">Display Mode:</label>
<select [(ngModel)]="mode">
  <option *ngFor="let item of modeOptions" [ngValue]="item.mode">{{item.mode}}</opti
</select>


```

Next, create a template for the card display. Name it as cardTemplate . The template takes items as input. Loop items collection using the ngFor to display the item header and content in the card format.

```

<ng-template let-items #cardTemplate>
  <div *ngFor="let item of items">
    <h1>{{item.header}}</h1>
    <p>{{item.content}}</p>
  </div>
</ng-template>

```

The listTemplate uses the ul to display the items in list format.

```

<ng-template let-items #listTemplate>
  <ul>
    <li *ngFor="let item of items">
      <strong>{{item.header}} </strong> ( {{item.content}} )
    </li>
  </ul>
```

```
</ng-template>

```

We finally pass the items to the item-view component. We also pass the template to it.

```

<item-view [itemTemplate]="template" [items]="items">
</item-view>

```

Now open the app.component.ts

First, get the reference to both the template using the ViewChild .

```

  @ViewChild('cardTemplate',null) cardTemplate:TemplateRef<HTMLElement>;
  @ViewChild('listTemplate',null) listTemplate:TemplateRef<HTMLElement>;

```

Define items , mode & modeOptions

```

mode ="card"

 items = [
   {
     header: 'Angular Tutorial',
     content: 'The Angular Tutorial for Beginners & Professionals'
   },
   {
     header: 'Typescript Tutorial',
     content: 'The Complete Guide to Typescript'
   },
   {
     header: 'Entity Framework Code Tutorial',
     content: 'Learn Everything about Entity Framework Core'
   },
 ];

```

```
modeOptions = [
  { mode: "card" },
  { mode: "list" },
];

```

the `template` returns either `listTemplate` or `cardTemplate` depending on the value of `mode`.

```

get template() {

  if(this.mode=="list") return this.listTemplate
  return this.cardTemplate
}

```

The ItemViewComponent recives the `items` to display and `itemTemplate` to use from the parent component.

```

@Input() items: any[] = [];
@Input() itemTemplate: TemplateRef<HTMLElement>;

```

Pass the `itemTemplate` to the `ngTemplateOutlet` to display the item. Use the `ngTemplateOutletContext` to pass the `items` collection.

```

<ng-container [ngTemplateOutlet]="itemTemplate" [ngTemplateOutletContext]="{$implicit
</ng-container>

```

Complete Source code

app.component.ts

```

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'ngTemplateOutlet Example';

  @ViewChild('cardTemplate',null) cardTemplate:TemplateRef<HTMLElement>;
  @ViewChild('listTemplate',null) listTemplate:TemplateRef<HTMLElement>;

  mode ="card"

  items = [
    {
      header: 'Angular Tutorial',
      content: 'The Angular Tutorial for Beginners & Professionals'
    },
    {
      header: 'Typescript Tutorial',
      content: 'The Complete Guide to Typescript'
    },
    {
      header: 'Entity Framework Code Tutorial',
      content: 'Learn Everything about Entity Framework Core'
    },
  ];

  modeOptions = [
    { mode: "card" },
    { mode: "list" },
  ];

  get template() {

    if(this.mode=="list") return this.listTemplate
    return this.cardTemplate
  }

}

```

app.component.html

```

<h1>ngTemplateOutlet Example</h1>

<label for="mode">Display Mode:</label>
<select [(ngModel)]="mode">
  <option *ngFor="let item of modeOptions" [ngValue]="item.mode">{{item.mode}}</op
</select>

<ng-template let-items #cardTemplate>
  <div *ngFor="let item of items">
    <h1>{{item.header}}</h1>
    <p>{{item.content}}</p>
  </div>
</ng-template>

<ng-template let-items #listTemplate>
  <ul>
    <li *ngFor="let item of items">
      <strong>{{item.header}} </strong> ( {{item.content}} )
    </li>
  </ul>
</ng-template>

<item-view [itemTemplate]="template" [items]="items">
</item-view>

```

item-view.component.ts

```

import { Component, Input, TemplateRef } from '@angular/core';


@Component({
  selector: 'item-view',
  template: `
  <h2>Item View</h2>

  <ng-container [ngTemplateOutlet]="itemTemplate" [ngTemplateOutletContext]="{$implic
  </ng-container>
  `
})
export class ItemViewComponent {

  @Input() items: any[] = [];
  @Input() itemTemplate: TemplateRef<HTMLElement>;

}

```

app.module.ts

```

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule} from '@angular/forms';

import { AppComponent } from './app.component';
import { ItemViewComponent } from './item-view.component';

@NgModule({
  declarations: [
    AppComponent,
    ItemViewComponent
  ],
  imports: [
    BrowserModule,FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

Reference

1. ngTemplateOutlet API

Read More

1. ng-Content & Content Projection
2. @input, @output & EventEmitter
3. Template Reference Variable
4. ng-container