Template Reference Variable in Angular

2 Comments / 5 minutes of reading / March 9, 2023

← Input, Output & EventEmitter

Angular Tutorial

ng-container →

This guide explains the Template Reference Variable in Angular. We find out what template reference variable is. How to define and use it in Angular.

Table of Contents

Template Reference Variable

The Template reference variable is a reference to any DOM element, component or a directive in the Template. We can use it elsewhere in the template. We can also pass it to a method in the component. It can contain a reference to elements like `h1`, `div`, etc

Defining Template Reference variable

We declare Template reference variables using `#` followed by the name of the variable (`#variable`). We can also declare them using `#variable="customer"` when the component/directive defines a customer as the `exportAs` Property.

For Example

HTML Element

```
<input type="text" #firstName>
```

Component/Directive

```
<app-customer #customerList="customer"></app-customer>
```

Component/Directive with `exportAs`

```
<app-customer #customerList="customer"></app-customer>
```

Template Reference variable Example

Now let us create a simple sample application to learn how to use a template reference variable

Create a new application

```
ng new templateVariable
```

HTML Element

The following Example code defines the `#firstName` & `#lastName` template reference variables. They both contain the reference to the input HTML Element.

The input elements contain the `value` property. Hence we can use it to display the welcome message as shown below.

app.component.html

```
<h1>{{title}}</h1>

<p>
 <label for="firstName">First Name</label>
 <input (keyup)="0" type="text" #firstName id="firstName">
</p>

<p>
 <label for="lastName">Last Name</label>
 <input (keyup)="0" type="text" #lastName id="lastName">
</p>

<b>Welcome {{firstName.value}} {{lastName.value}} </b>
```

We have used `(keyup)="0"` on the `input` element. It does nothing but it forces the angular to run the change detection. change detection, in turn, updates the view.

The Angular updates the view, when it runs the change detection. The change detection runs only in response to asynchronous events, such as the arrival of HTTP responses, raising of events, etc. In the example above whenever you type on the input box, it raises the `keyup` event. It forces the angular to run the change detection, hence the view gets the latest values.

Pass Variable to Component class

You can also pass the variables to the component as shown below. Add this code app.component.html

```
<p>
<button (click)="sayHello(firstName, lastName)"> Say Hello</button>
</p>
```

Add this to app.component.ts

```
sayHello(firstName, lastName) {
  alert('Hello '+firstName.value+' '+ lastName.value)
}
```

Component/Directive

You can get a reference to the component or directive. Refer to the tutorial on How to create child/nested component in Angular

Create a new component customer-list.component.ts

```
1
2  import { Component } from '@angular/core';
3  import { Customer } from './customer';
4
5  @Component({
6    selector: 'app-customer-list',
7    templateUrl: './customer-list.component.html',
8    exportAs:'customerList'
9  })
10 export class CustomerListComponent {
11
12   selectedCustomer
13
14   customers: Customer[] = [
15
16     {customerNo: 1, name: 'Rahuld Dravid', address: '', city: 'Banglaore', state: 'Karnataka',
17     {customerNo: 2, name: 'Sachin Tendulkar', address: '', city: 'Mumbai', state: 'Maharastr
18     {customerNo: 3, name: 'Saurrav Ganguly', address: '', city: 'Kolkata', state: 'West Beng
19     {customerNo: 4, name: 'Mahendra Singh Dhoni', address: '', city: 'Ranchi', state: 'Bihar'
20     {customerNo: 5, name: 'Virat Kohli', address: '', city: 'Delhi', state: 'Delhi', country: 'Ind
21
22   ]
23 }
24
```

customer-list.component.html

```
1
2  <h2>List of Customers</h2>
3
4  <table class='table'>
5    <thead>
6      <tr>
7        <th>No</th>
8        <th>Name</th>
9        <th>Address</th>
10       <th>City</th>
11       <th>State</th>
12       <th>Select</th>
13     </tr>
14   </thead>
15   <tbody>
```

```
    <tr *ngFor="let customer of customers;">
      <td>{{customer.customerNo}}</td>
      <td>{{customer.name}}</td>
      <td>{{customer.address}}</td>
      <td>{{customer.city}}</td>
      <td>{{customer.state}}</td>
      <button (click)="selectedCustomer=customer">Select</button>
    </tr>
  </tbody>
</table>

```

customer.ts

```

export class Customer {

  customerNo: number;
  name:string ;
  address:string;
  city:string;
  state:string;
  country:string;

}

```

app.component.html

```

You have selected {{customerListComponent.selectedCustomer?.name}}
<app-customer-list #customerListComponent></app-customer-list>

```

ExportAs

Sometimes there could be more than one directive on a DOM Element.

For Example in the following code has two directives. In such a case, we need to specify which directive to use for `#variable`.

```
<a-directive b-directive #variable />
```

The components or directives can use `exportAs` to export the component/directive in a different name.

For Example, open the `customer-list.component` and add the `exportAs:'customerList'` under the `@Component` metadata

```
@Component({
 selector: 'app-customer-list',
 templateUrl: './customer-list.component.html',
 exportAs:'customerList'
})
```

Now you can use it a

```
You have selected {{customerList.selectedCustomer?.name}}
<app-customer-list #customerList="customerList"></app-customer-list>
```

The safe navigation operator (?)

The `selectedCustomer` is null when the app starts for the first time. It gets its value only when the user clicks on the select button. Hence we use `?` or a safe navigation operator to guard against null or undefined value.

Without ? the app will throw error and stops

Template Driven Forms

The ngForm directive uses the exportAs to export an instance of itself using the name ngForm . We use this in the template-driven forms in Angular.

```
<form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)">
```

Now, you can check value and validate the status of the form within the template

```
<pre>Value : {{contactForm.value | json }} </pre>
<pre>Valid : {{contactForm.valid}} </pre>
<pre>Touched : {{contactForm.touched  }} </pre>
<pre>Submitted : {{contactForm.submitted  }} </pre>
```

Template Input Variable

The Template reference variable must not be confused with the Template input variable, which we define using the let keyword in the template.

For Example, check the ngFor loop in the customer-list.component.html . The variable customer is a template input variable

```
<tr *ngFor="let customer of customers;">
   <td>{{customer.customerNo}}</td>
   <td>{{customer.name}}</td>
   <td>{{customer.address}}</td>
   <td>{{customer.city}}</td>
   <td>{{customer.state}}</td>
```

```
    <button (click)="selectedCustomer=customer">Select</button>
  </tr>
```

Variable Scope

The scope of the template reference variable is the template within which it is declared. You cannot access it outside the template.

Hence any variable declared in `CustomerListComponent` cannot be accessed from the app component although we render the customer list within the app component.

Child scope

Also, note that we can create a new child template scope (nested scope) by using the directive `ng-template` . Also the structural directive like ngIf, ngFor, etc also creates their own child scope.

The following is an example of the `<ng-template>` directive. The address variable defined inside the `ng-template` is not accessible outside it.

```
<h2>Variable Scope</h2>

<div *ngIf="false else addressTemplate"></div>

<ng-template #addressTemplate>
 <p>
  <label for="address">Address</label>
  <input (keyup)="0" type="text" #address id="address">
 </p>
 <p>The address of {{firstName.value}} {{lastName.value}} Entered {{address.value}}

</ng-template>

<!-- address is not accessible here -->
```

```
<p>
You Entered {{address?.value}}
</p>
```

ngIf Example

```
<div *ngIf="true">
  <app-customer-list #variable></app-customer-list
  //variable is accessible from here
</div>

//variable is not accessible from here
```

References

- Angular Docs
- Template Driven Forms

Summary

The Template reference variable is a reference to any DOM element, component or a directive in the Template. Use `#variable` to create a reference to it. We can also use `#variable="exportAsName"` when the component/directive defines an `exportAs` metadata. The template variable can be used anywhere in the template. The variable can be passed the component as an argument to a method. The Template reference variable is different from the template input variable which we define using the `let` keyword within the template. They are Template scoped. Also, make use of the safe navigation operator `?`, if you expect a null value.