How to Create Custom Pipe in Angular

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

← Angular Pipes Angular Tutorial Date Pipe →

In this tutorial, we will show you how to create an Angular Custom Pipe. The Pipes are a great way to transform the appearance of elements in the template. The Angular comes with some great built-in pipes like Date pipe, Currency pipe, and Number pipe, etc. But if these pipes do not cover your needs, then we can create our own pipe in Angular.

To create a custom pipe, first we need to create a pipe class. The pipe class must implement the `PipeTransform` interface. We also decorate it with `@pipe` decorator. Give a `name` to the pipe under `name` metadata of the `@pipe` decorator. Finally, we create the `transform` method, which transforms given value to the desired output.

Table of Contents

How to Create Custom Pipes

To create a Custom Pipe, first, You need to follow these steps

1. Create a pipe class
2. Decorate the class with `@pipe` decorator.
3. Give a name to the pipe in the `name` meta data of the `@pipe` decorator. We will use this name in the template.
4. The pipe class must implement the `PipeTransform` interface. The interfaces contain only one method `transform`.
5. The first parameter to the `transform` method is the value to be transferred. The `transform` method must transform the value and return the result. You can add any number of additional arguments to the `transform` method.
6. Declare the pipe class in the Angular Module (app.module.ts)
7. Use the custom pipe just as you use other pipes.

Now let us create a Temperature converter pipe, which converts temperature from Celsius to Fahrenheit and vice versa.

Temparature Convertor Custom Pipe Example

Create a new Angular application. If you are new to Angular you can refer to the tutorial Create Angular Application.

We are using bootstrap 4 for styling. Hence open the `index.html` and add the following

```
1
2 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstra
3
```

Create a new file `temp-convertor.pipe.ts`. Under the folder `src/app`. Copy the following code and paste it.

```
1
2 import {Pipe, PipeTransform} from '@angular/core';
```

```
3
4  @pipe({
5     name: 'tempConverter'
6  })
7  export class TempConverterPipe implements PipeTransform {
8     transform(value: number, unit: string) {
9        if(value && !isNaN(value)) {
10          if (unit === 'C') {
11             var temperature = (value - 32) /1.8 ;
12             return temperature.toFixed(2);
13          } else if (unit === 'F'){
14             var temperature = (value * 1.8 ) + 32
15             return temperature.toFixed(2);
16          }
17       }
18       return;
19    }
20 }
21
```

Let us look at the code in details

We need to import the `Pipe` & `PipeTransform` libraries from Angular. These libraries are part of the Angular Core

```
1
2  import {Pipe, PipeTransform} from '@angular/core';
3
```

We decorate `TempConverterPipe` class with `@pipe` decorator. The @pipe decorator is what tells Angular that the class is a pipe. the decorator expects us to provide a name to the pipe. We have given it as `tempConverter` . This is the name we must use in the template to make use of this pipe.

Our class must implement the `PipeTransform` interface.

```
1
2  @pipe({
```

```
3     name: 'tempConverter'
4 })
5 export class TempConverterPipe implements PipeTransform {
6
7
8 }
9
```

The `PipeTransform` interface defines only one method `transform`. The interface definition is as follows.

```
1
2 interface PipeTransform {
3   transform(value: any, ...args: any[]): any
4 }
5
```

The first argument value is the value, that pipe needs to transform. We can also include any number of arguments. The method must return the final transformed data.

The following is Our implementation of the `transform` method. The first is `Value` and the second is the `Unit`. The `unit` expects either `C` (Convert to Celsius) or `F` (convert to Fahrenheit). It converts the value received to either to Celsius or to Fahrenheit based on the Unit.

```
1
2 export class TempConverterPipe implements PipeTransform {
3
4     transform(value: number, unit: string) {
5         if(value && !isNaN(value)) {
6             if (unit === 'C') {
7                 var temperature = (value - 32) /1.8 ;
8                 return temperature.toFixed(2);
9             } else if (unit === 'F'){
10                var temperature = (value * 1.8 ) + 32
11                return temperature.toFixed(2);
12            }
```

```
      }
      return;
    }

}

```

Declare the Pipe

Before using our pipe, we need to tell our component, where to find it. This is done by first by importing it and then including it in declarations array of the `AppModule`.

```

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

import { AppComponent } from './app.component';

import {TempConverterPipe} from './temp-convertor.pipe';

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

```

Using the Custom Pipe

The custom pipes are used in the same as the Angular built-in pipes are used. Add the following HTML code to your `app.component.html` file

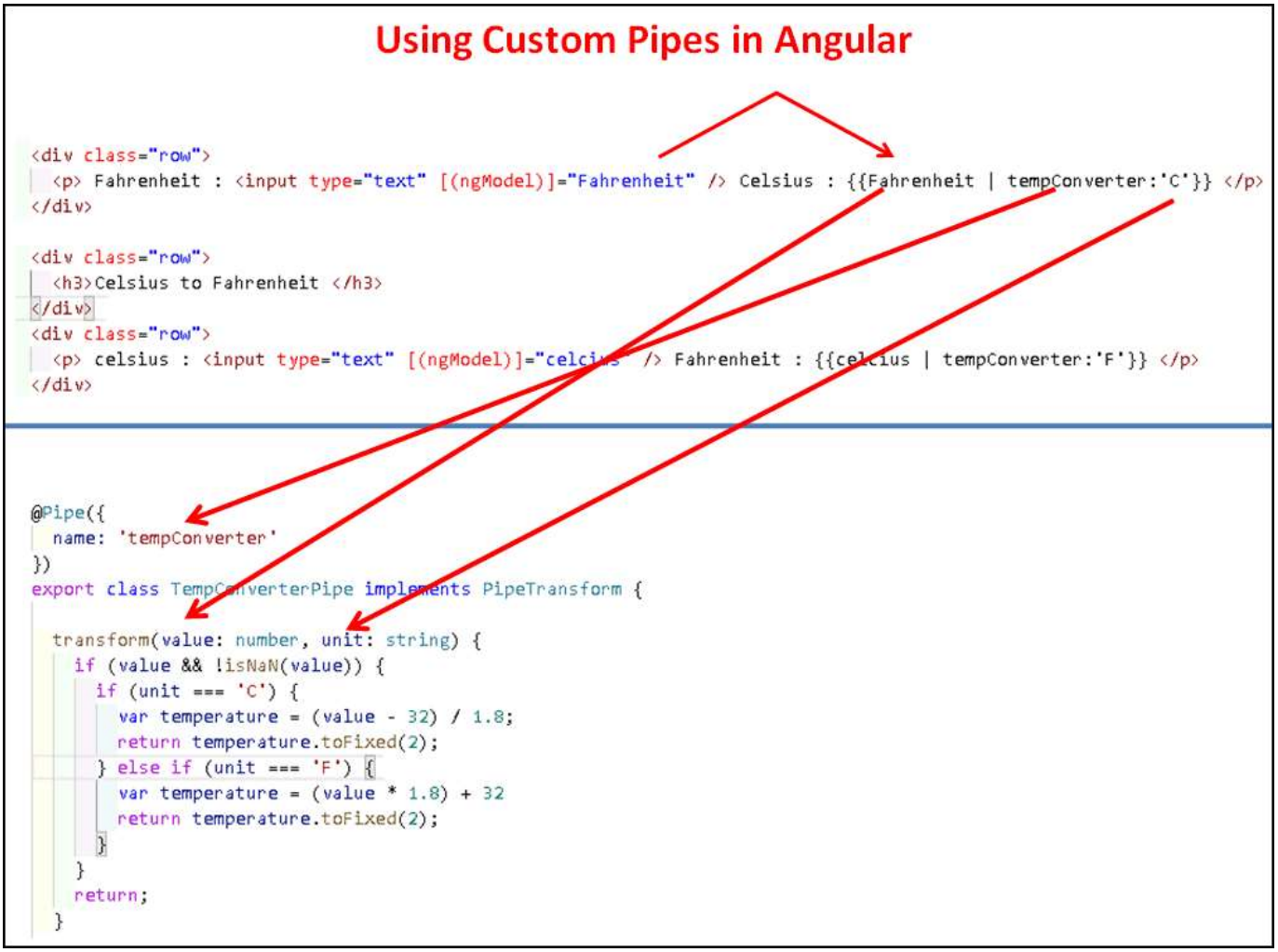
```

<div class='card'>
  <div class='card-header'>
    <p>{{title}} </p>
  </div>
  <div class="card-body">
```

```
 7
 8     <div class="row">
 9       <h3>Fahrenheit to Celsius </h3>
10     </div>
11     <div class="row">
12       <p> Fahrenheit : <input type="text" [(ngModel)]="Fahrenheit" />
13       Celsius : {{Fahrenheit | tempConverter:'C'}} </p>
14     </div>
15
16     <div class="row">
17       <h3>Celsius to Fahrenheit </h3>
18     </div>
19     <div class="row">
20       <p> celsius : <input type="text" [(ngModel)]="celcius" />
21        Fahrenheit : {{celcius | tempConverter:'F'}} </p>
22     </div>
23   </div>
24 </div>
25
```

We use our pipe as follows. `Fahrenheit` is sent to the `tempConverter` as the first argument `value`. We use the `|` to indicate that the `tempConverter` is a pipe to angular. The `C` after the colon is the first argument. You can pass more than argument to the pipe by separating each argument by a `:` colon.

```
1
2 {{Fahrenheit | tempConverter:'C'}}
3
```



Using the TempConverter Pipe in Template

app.component code

```
1
2  import { Component } from '@angular/core';
3  import { FormsModule } from '@angular/forms';
4
5
6  @Component({
7    selector: 'app-root',
8    templateUrl: './app.component.html',
9    styleUrls: ['./app.component.css']
10 })
11
12
13 export class AppComponent
14 {
15     title: string = 'Angular Custom Pipe Example' ;
16     celcius: number;
```