Custom Validator in Template Driven Forms in Angular

3 Comments / 6 minutes of reading / March 9, 2023

← Validation in Template Driven Forms

Angular Tutorial

Angular Async Validator →

Learn how to build custom validator in template-driven forms. In Angular, we have two API to build Angular Forms. They are Reactive Forms and template-driven forms. In Template-driven forms, we define validation rule as an HTML attribute in the HTML markup. The Angular provides a few built-in Validation attributes out of the box. In this tutorial, we will talk about how to build a custom validation attribute using Angular Directive. We also show you how to pass a parameter & how to inject service into the validation attribute.

Table of Contents

Custom Validator in Template Driven Forms

Create a new Angular Project. Copy the following code to app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {

  constructor() {
  }

}
```

Copy the following code app.component.html

```
<h1>Custom Validator in Template driven forms</h1>

<h2>Template Form</h2>

<form #myForm="ngForm" (ngSubmit)="onSubmit(myForm)" novalidate>

  <label for="numVal">Number :</label>
  <input type="text" name="numVal" ngModel #numVal="ngModel">

  <p>Is Form Valid : {{myForm.valid}} </p>

  <p>Form  : {{ myForm.value | json}} </p>
  <p>
    <button type="submit" [disabled]="!myForm.valid">Submit</button>
  </p>

</form>
```

```
19
```

It has only one input field `numVal`. Let us create a validator to ensure that the value of the `numVal` is greater than 10.

Built-in Validators

The Angular Forms Module already has a few built-in validators. They are listed below. But we do not have a greater than validator.

1. Required validator
2. Min length Validator
3. Max length Validator
4. Pattern Validator
5. Email Validator

How to Build Custom Validator in template-driven form

Building a Validator in template-driven forms is similar to building an Angular directive. The directive must implement the Validator interface.

Validator Interface

```
1
2  interface Validator {
3    validate(control: AbstractControl): ValidationErrors | null
4    registerOnValidatorChange(fn: () => void)?: void
5  }
6
```

The directive must implement the `validate` function. Notice that the `validate` function has the same signature as the ValidatorFn Interface. Whenever the `Validator` directive

is invoked angular looks for the `validate` method and invokes it.

Validate Function

A `Validator` is just a function, which must implement ValidatorFn Interface.

```
interface ValidatorFn {
  (control: AbstractControl): ValidationErrors | null
}
```

The function takes the AbstractControl. This is the base class for `FormControl`, `FormGroup`, and `FormArray`. The validator function must return a list of errors i.e ValidationErrors or `null` if the validation has passed

Custom Validator Example

Create the `gte.validator.ts` and copy the following code.

```
import { Validator, NG_VALIDATORS, FormControl } from '@angular/forms'
import { Directive, OnInit, forwardRef } from '@angular/core';


@Directive({
```

```
  selector: '[gteValidator]',
  providers: [
    { provide: NG_VALIDATORS, useExisting: gteValidatorDirective, multi: true }
  ]
})
export class gteValidatorDirective implements Validator, OnInit {

  ngOnInit() {
  }

  validate(c: FormControl) {

    let v: number = +c.value;

    if (isNaN(v)) {
      return { 'gte': true, 'requiredValue': 10 }
    }

    if (v <= +10) {
      return { 'gte': true, 'requiredValue': 10 }
    }

    return null;
  }
}

```

We decorate the directive using `@Directive` decorator.

We use the directive as an attribute in the HTML template. The attribute needs a name or selector. We assign the name as `gteValidator` in the selector metadata section of the directive decorator.

```

selector: '[gteValidator]',

```

The Angular knows nothing about the Validation capabilities of our directive. Hence we need to register it in Angular Providers metadata using the special injection token

NG_VALIDATORS. We also set `multi:true` because there can be more validation directives.

```
{ provide: NG_VALIDATORS, useExisting: gteValidatorDirective, multi: true }
```

The directive class must implement the validate method. The validate method must honor the ValidatorFn Interface.

```
validate(c: FormControl) {

  let v: number = +c.value;

  if (isNaN(v)) {
    return { 'gte': true, 'requiredValue': 10 }
  }

  if (v <= +10) {
    return { 'gte': true, 'requiredValue': 10 }
  }

  return null;
}
```

It is a simple function, which checks if the value is a number and is less than 10. It returns null if it passes all checks.

If Validation fails it returns the `ValidationErrors`. It is a key-value pair object of type `[key: string]: any` and it defines the broken rule. The `key` is the string and should contain the name of the broken rule. The value can be anything, but usually set to `true` .

We return the following key-value pair when the validation fails

```
return { 'gte': true, 'requiredValue': 10 }
```

The `'gte': true:` indicates that the validation has failed. `'requiredValue': 10` is used by the template to display that the expected value is greater than 10.

Using the Custom Validator

Since this is a template-driven form., we do not have to do anything in the component class. In the HTML template just add the attribute `gteValidator` as shown below

```
<label for="numVal">Number :</label>

<input type="text" name="numVal" ngModel #numVal="ngModel" gteValidator>

<div *ngIf="!numVal.valid && (numVal.dirty ||numVal.touched)">
  <div *ngIf="numVal.errors.gte">
    The number should be greater than {{numVal.errors.requiredValue}}
  </div>
</div>
```

Validators return `ValidationErrors`. They are added to the control's `errors` collection of the control. The `valid` property of the control is set to `false`.

Hence we check if the `valid` property. We also check the `dirty` and `touched` property. Because we do not want to display the error message when the form is displayed for the first time.

We check if the `gte` is true and display the error message. Note that `gte` is the name of the `key` we used while creating the validator.

We also make use of `requiredValue` to show a meaningful message to the user.

```
<div *ngIf="numVal.errors.gte">
  The number should be greater than {{numVal.errors.requiredValue}}
</div>
```

Passing Parameter to Validator

We have hardcoded the value of 10 in the above example. This will make our validator difficult to reuse. If we want to resue it, we need to pass the number to be checked as the parameter.

Since they are directives, we can use Input decorator to pass the parameter to the Validator.

Open the template add the special attribute `gteNum="20"`

```
<input type="text" name="numVal" ngModel #numVal="ngModel" gteValidator gteNum="20
```

You can read the `gteNum` from the template using the Input decorator as shown below

```

@Input("gteNum") gteNum:number

```

Now, you can remove the hardcoded value 10 and use the `gteNum` instead.

The complete validator code is as shown below.

```
import { Validator, NG_VALIDATORS, FormControl } from '@angular/forms'
import { Directive,  Input } from '@angular/core';

@Directive({
  selector: '[gteValidator]',
  providers: [
    { provide: NG_VALIDATORS, useExisting: gteValidatorDirective, multi: true }
  ]
})
export class gteValidatorDirective implements Validator {

  @Input("gteNum") gteNum:number

  validate(c: FormControl) {

    let v: number = +c.value;

    if (isNaN(v)) {
      return { 'gte': true, 'requiredValue': this.gteNum }
    }

    if (v <= +this.gteNum) {
      return { 'gte': true, 'requiredValue': this.gteNum }
    }

    return null;
  }

}

```

Injecting Service into Validator

The validator may depend on some external service to validate the value. For Example, it may need to fetch data from the back end server.

Let us move the validation logic in the above validator to a separate service. Create a service `gte.service.ts` and copy the following code

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

@Injectable({
  providedIn: 'root',
})
export class gteService {

  gte(num:any, requiredValue:Number) : Boolean {

    if (isNaN(num)) {
      return false;
    }

    if (num <= +requiredValue) {
      return false;
    }

    return true;
  }
}


```

In the validation directive, create a constructor method and inject the service. The complete code is as shown below

```
import { Validator, NG_VALIDATORS, FormControl } from '@angular/forms'
import { Directive,  Input } from '@angular/core';
import { gteService } from 'projects/injectService1/src/app/gte.service';

@Directive({
  selector: '[gteValidator]',
  providers: [
```