Angular Async Validator Example

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

← Custom Validator in Template Driven Forms

Angular Tutorial

Cross field validation →

In this guide let us learn how to create a custom async validator in Angular. The creating an async validator is very similar to the Sync validators. The only difference is that the async Validators must return the result of the validation as an observable (or as Promise).

Angular does not provide any built-in async validators as in the case of sync validators. But building one is very simple.

Table of Contents

How to Create Async Validator

Creating a Async Validator is simple as creating a function, which must obey the following rules

1. The function must implement the AsyncValidatorFn Interface, which defines the signature of the validator function.
2. The function must return either an observable or a promise
3. Return null for valid, or an ValidationErrors if the input is invalid

AsyncValidatorFn

The AsyncValidatorFn is an Interface, which defines the signature of the validator function.

```
1
2 interface AsyncValidatorFn {
3   (control: AbstractControl): Promise<ValidationErrors | null> | Observable<ValidationErrors
4 }
5
```

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. The only difference it has with the Sync Validator is the return type. It must return either a promise or an observable .

Async Validator Example

We build gte validator in how to create a custom validator in Angular tutorial. In this Async Validator Example, let us convert that validator to Async Validator.

Create a new Angular Application. Add the gte.validator.ts and copy the following code.

```
1
```

```
import { AbstractControl, ValidationErrors } from '@angular/forms'

import { Observable, of } from 'rxjs';

export function gte(control: AbstractControl):
        Observable<ValidationErrors> | null {

    const v:number=+control.value;

    console.log(v)

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

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

    return of(null)

}

```

First, import the AbstractControl and ValidationErrors from the @angular/forms. Since we need to return an observable, we also need to import Observable from the rxjs library.

The validator function must follow the AsyncValidatorFn Interface. It should receive the AbstractControl as its parameter. It can be FormControl, FormGroup or FormArray.

The function must validate the control value and return `ValidationErrors` if any errors are found otherwise `null`. It must return them as observable.

The `ValidationErrors` 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`.

The validation logic is very simple. Check if the value of the control is a number using the isNaN method. Also, check if the value is less than or equal to 10. If both the rules are valid and then return `null`

If the validation fails then return the `ValidationErrors`. You can use anything for the `key`, but it is advisable to use the name of the validator i.e `gte` as the key. Also, assign `true` as value. You can as well assign a string value.

You can return more than one `key-value` pair as shown in the above example. The second key `requiredValue` returns the value 10. We use this in the template to show the error message.

We use the `of` operator convert the result into an observable and return it

Using the Async Validator

To use this validator first, import it in the component class.

```
1
2 import { gte } from './gte.validator';
3
```

Add the validator to the Async Validator collection of the FormControl as shown below. The async validator is the third argument to the FormControl.

```
 myForm = new FormGroup({
   numVal: new FormControl('', [gte]),
 })

 // Without FormGroup
 // this.myForm = new FormGroup({
 //   numVal: new FormControl('', null, [gte]),
 // })
```

That's it. The complete app.component.ts code as show below

```
import { Component } from '@angular/core';
import { FormGroup, FormControl } from '@angular/forms'
import { gte } from './gte.validator';


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

  constructor() {
  }
```

```
16
17
18  myForm = new FormGroup({
19    numVal: new FormControl('',null, [gte]),
20  })
21
22  get numVal() {
23    return this.myForm.get('numVal');
24  }
25
26
27  onSubmit() {
28    console.log(this.myForm.value);
29  }
30
```

The complete app.component.html

```
<h1>Async Validator in Angular</h1>

<h2>Reactive Form</h2>

<form autocomplete="off" [formGroup]="myForm" (ngSubmit)="onSubmit()" novalidate>

  <div>
    <label for="numVal">Number :</label>
    <input type="text" id="numVal" name="numVal" formControlName="numVal">
    <div *ngIf="!numVal.valid && (numVal.dirty ||numVal.touched)">
      <div *ngIf="numVal.errors.gte">
        The number should be greater than {{numVal.errors.requiredValue}}
      </div>
    </div>

  </div>


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

  <p>
    <button type="submit" [disabled]="!myForm.valid">Submit</button>
  </p>


</form>

```

app.module.ts

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


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

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
```

```
    ReactiveFormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }


```

The use case for Async Validators

We use the async validator when we need to send an HTTP call to the server to check if the data is valid.

The following code shows how you can send a HTTP Request to verify the data.

If the data is Valid we will return nothing, else we return the ValidationErrors i.e. ({ 'InValid': true }) .

```

import { AbstractControl, ValidationErrors } from '@angular/forms'
import { Observable, pipe } from 'rxjs';
import { map, debounceTime } from 'rxjs/operators';

export function validate(control: AbstractControl): Observable<ValidationErrors> | null {

  const value: string = control.value;

  return this.http.get(this.baseURL + 'checkIfValid/?value=' + value)
    .pipe(
      debounceTime(500),
      map( (data:any) =>  {
          if (!data.isValid) return ({ 'InValid': true })
      })
    )

}

```