# 一、开始

https://formly.dev/guide/getting-started

一、

tyarn add @ngx-formly/schematics @ngx-formly/material

ui-theme:

bootstrap

material

ionic

primeng

kendo

nativescript

1、

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

import { ReactiveFormsModule } from '@angular/forms';

import { FormlyModule } from '@ngx-formly/core';

import { FormlyBootstrapModule } from '@ngx-formly/bootstrap';

@NgModule({

imports: [

BrowserModule

ReactiveFormsModule,

FormlyModule.forRoot(),

FormlyBootstrapModule

],

...

})

export class AppModule { }

2 add <formly-form> inside the form tag to your AppComponent template:

<form [formGroup]="form" (ngSubmit)="onSubmit(model)">

<formly-form [form]="form" [fields]="fields" [model]="model"></formly-form>

<button type="submit" class="btn btn-default">Submit</button>

</form>

The <formly-form> component is the main container of the form, which will build and render form fields, it accept the following inputs:

fields: 用于构建表单的字段配置

form: 允许跟踪模型值和验证状态的表单实例

model: 由表单表示的模型。

export class AppComponent {

form = new FormGroup({});

model = { email: 'email@gmail.com' };

fields: FormlyFieldConfig[] = [

{

key: 'email',

type: 'input',

templateOptions: {

label: 'Email address',

placeholder: 'Enter email',

required: true,

}

}

];

onSubmit() {

console.log(this.model);

}

}

# 二 、 Properties and Options

<formly-form [form]="form" [fields]="fields" [model]="model">

</formly-form>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Default | Required | Description |
| form | FormGroup or FormArray | new FormGroup({}) | no | 允许跟踪模型值和验证状态的表单实例。 |
| fields | FormlyFieldConfig[] |  | yes | 用于构建表单的字段配置 |
| model | any |  | yes | 由表单表示的模型。 |
| options | FormlyFormOptions |  | no | 表单的选项 |

## fields

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
| key | string | 与模型相关的键。这将把字段值链接到模型。 |
| id | string | 这允许您指定字段的id。注意，如果未设置，则生成id。 |
| name | string | 如果您愿意，您可以为您的字段指定一个特定的名称。如果您使用以前的技术将表单发布到服务器，那么这是非常有用的。 |
| type | string | 要呈现的字段的类型。更多信息请访问自定义模板。 |
| component | any | 可以设置为替换类型中定义的组件。 |
| className | string | 您可以指定将应用于formly-field指令的自己的类。 |
| templateOptions | object | 这是为模板保留的。任何特定于模板的选项都可以放在这里。查看您的特定模板实现以了解对此所需的选项。 |
| template | string | 可以设置而不是使用类型来呈现自定义html内容。 |
| defaultValue | any | 如果提供了这个参数，并且编译时模型的值是未定义的，那么模型的值将被赋给defaultValue。 |
| hide | boolean | 是否隐藏字段。默认值为false。如果您希望这是有条件的使用隐藏表达式 |
| hideExpression | boolean or string or function | 根据来自其他字段的值有条件地隐藏字段。 |
| expressionProperties | boolean or string or function | 对象，其中键是要在主字段配置上设置的属性，值是用来分配该属性的表达式。 |
| focus | boolean | 是聚焦还是模糊元素字段。默认值为false。如果您希望这是有条件的，请使用expressionProperties |
| wrappers | string[] | 它可能是包装器的名称。formly字段模板将被第一个包装器包装，然后是第二个，然后是第三个，等等。还可以将它们指定为类型的一部分(这是推荐的方法)。 |
| parsers | function[] | 当模型更新时，通常通过用户输入作为管道执行的函数数组。 |
| fieldGroup | FormlyFieldConfig[] | 字段组是一种将字段分组在一起的方法，使得高级布局非常简单。它还可以用于对与同一个模型相关联的字段进行分组(如果它与其他字段的模型不同，则非常有用)。 |
| fieldArray | FormlyFieldConfig |  |
| fieldGroupClassName | string | 指定将应用于formly-group组件的类。 |
| validation | object | 具有一些有用属性的对象:消息、显示 |
| validators | any | 用于为特定字段设置验证规则。应该是键值对的对象。该值可以是要计算的表达式，也可以是要运行的函数。每个都应该返回一个布尔值，当字段有效时返回true。有关更多信息，请参见验证。 |
| asyncValidators | any | 对于需要异步验证的任何内容，都可以使用此方法。与validators api几乎完全相同，只是它必须是一个返回承诺的函数。 |
| formControl | AbstractControl | 这是字段的表单控件。它提供了更多的控件，比如运行验证器、计算状态和重置状态。 |
| modelOptions | object | 一个具有一些有用属性的对象，用于控制模型更改:debounce、updateOn |

Options

formState  
formState属性被传递给所有字段，它是一种用于字段之间通信的机制(而不必破坏您的模型)。  
resetModel和updateInitialValue  
它还接收方法resetModel和updateInitialValue，这两个方法将分别调用字段的所有resetModel和updateInitialValue。  
fieldTransform  
允许您在Formly处理或验证字段之前修改/转换字段。

# 三、验证

自定义验证

Formly提供了不同的方法来实现自定义验证。

1. 在NgModule声明中声明验证函数和消息。

自定义验证消息

要定义自定义验证消息，需要添加一个具有两个属性的对象:name和message。你将把这个对象包含到FormlyModule配置的validationMessages数组中:

message属性可以简单地定义为一个字符串，也可以定义为一个函数，该函数接收错误和验证为输入的字段(FormlyFieldConfig)。最后一个允许您进行更多的定制

export function IpValidatorMessage(err, field: FormlyFieldConfig) {

return `"${field.formControl.value}" is not a valid IP Address`;

}

...

@NgModule({

imports: [

...

FormlyModule.forRoot({

validationMessages: [

{ name: 'ip', message: IpValidatorMessage },

{ name: 'required', message: 'This field is required' },

],

}),

]

})

自定义验证函数

验证函数接收FormControl作为输入，如果没有错误，它将返回null，否则它将发送一个属性设置为true的对象。属性的名称必须与为此验证设置的错误消息的名称相同。

下面的代码示例显示了一个验证IP的函数。可以看到，出现错误时返回的对象具有一个名为“ip”的属性，该属性与自定义验证消息的名称相匹配。

export function IpValidator(control: FormControl): ValidationErrors {

return /(\d{1,3}\.){3}\d{1,3}/.test(control.value) ? null : { 'ip': true };

}

...

@NgModule({

imports: [

...

FormlyModule.forRoot({

validators: [

{ name: 'ip', validation: IpValidator },

],

validationMessages: [

{ name: 'ip', message: IpValidatorMessage },

{ name: 'required', message: 'This field is required' },

],

}),

]

})

带有自定义验证的字段

您只需要在属性validator .validation中包含在FormlyModule中声明的validate函数的名称。

{

key: 'ip',

type: 'input',

templateOptions: {

label: 'IP Address (using custom validation declared in ngModule)',

required: true,

},

validators: {

validation: ['ip'],

},

},

警报!如果函数是异步的，则需要将其包含在asyncValidators中。验证属性:

{

key: 'ip',

type: 'input',

templateOptions: {

label: 'IP Address (using custom validation declared in ngModule)',

required: true,

},

asyncValidators: {

validation: ['ipAsync'],

},

},

2. 在字段定义中声明验证函数。

自定义验证消息

[如上文所述]

自定义验证函数

export function IpValidator(control: FormControl): ValidationErrors {

return /(\d{1,3}\.){3}\d{1,3}/.test(control.value) ? null : { 'ip': true };

}

带有自定义验证的字段

您只需要在属性validator .validation中包含验证函数(在需要的地方声明)。

{

key: 'ip',

type: 'input',

templateOptions: {

label: 'IP Address (using custom validation through `validators.validation` property)',

required: true,

},

validators: {

validation: [IpValidator],

},

},

警报!如果函数是异步的，您可能需要在asyncValidators中定义它。验证属性:

{

key: 'ip',

type: 'input',

templateOptions: {

label: 'IP Address (using custom validation through `validators.validation` property)',

required: true,

},

asyncValidators: {

validation: [IpAsyncValidator],

},

3.在字段定义中声明验证函数和消息。

自定义验证消息和自定义验证功能

字段的validators属性可以接受与不同验证器匹配的不同嵌套属性。格式如下:

NAME\_OF\_VALIDATOR: {

expression: FUNCTION

message: FUNCTION | STRING

}

带有自定义验证的字段

正如在下面的代码示例中可以看到的，您只需要为validator属性(ip)提供一个名称，并在表达式属性中包含一个函数，在消息属性中包含一个字符串或自定义消息函数。

{

key: 'ip',

type: 'input',

templateOptions: {

label: 'IP Address (using custom validation through `validators.expression` property)',

description: 'custom validation message through `validators.expression` property',

required: true,

},

validators: {

ip: {

expression: (c) => /(\d{1,3}\.){3}\d{1,3}/.test(c.value),

message: (error, field: FormlyFieldConfig) => `"${field.formControl.value}" is not a valid IP Address`,

},

},

},

警报!如果函数是异步的，您可能需要在asyncValidators属性中定义它:

{

key: 'ip',

type: 'input',

templateOptions: {

label: 'IP Address (using custom validation through `validators.expression` property)',

description: 'custom validation message through `validators.expression` property',

required: true,

},

asyncValidators: {

ip: {

expression: (c) => return new Promise((resolve, reject) => {

setTimeout(() => {

resolve(/(\d{1,3}\.){3}\d{1,3}/.test(c.value));

}, 1000);

}),

message: (error, field: FormlyFieldConfig) => `"${field.formControl.value}" is not a valid IP Address`,

},

},

},

4. 在NgModule声明中以形式类型和消息声明验证函数。

自定义验证消息

创建一个自定义验证消息，如第1点所述。

export function IpValidatorMessage(err, field: FormlyFieldConfig) {

return `"${field.formControl.value}" is not a valid IP Address`;

}

...

@NgModule({

imports: [

...

FormlyModule.forRoot({

validationMessages: [

{ name: 'ip', message: IpValidatorMessage },

{ name: 'required', message: 'This field is required' },

],

}),

]

})

用于FORMLY类型的自定义验证函数

验证函数接收FormControl作为输入，它将返回一个布尔值，如果没有错误，返回false，否则返回true。

下面的代码示例显示了一个验证IP的函数。它必须返回一个布尔值，而不是ValidationErrors对象。

export function IpValidator(control: FormControl): boolean {

return /(\d{1,3}\.){3}\d{1,3}/.test(control.value);

}

...

@NgModule({

imports: [

...

FormlyModule.forRoot({

validationMessages: [

{ name: 'ip', message: IpValidatorMessage },

{ name: 'required', message: 'This field is required' },

],

types: [

{

name: 'ip',

extends: 'input',

defaultOptions: {

validators: {

ip: IpValidator // 'ip' matches the ip validation message

}

},

},

}),

]

})

使用带自定义验证的FORMLY类型的字段

formly类型必须在字段中引用

{

key: 'ip',

type: 'ip', // the formly type defined

templateOptions: {

label: 'IP Address (using custom validation declared in ngModule)',

required: true,

},

},

您可以尝试这个演示程序来测试创建自定义验证器的几个方法。

1 - 唯一值的异步验证

https://formly.dev/examples/validation/unique-value-async-validation

2 - 定制验证器

https://formly.dev/examples/validation/custom-validation

2 -

您可以在字段定义中实现验证功能。例如，如果希望使用具有相同错误消息的多个验证函数，可以使用此方法。

# 四、Formly Expressions 表达式

可以使用隐藏表达式动态隐藏字段，表达式属性允许动态更改字段的许多属性。

表达式的属性

表达式属性允许您动态更改字段的许多属性。例如，可以动态禁用表单字段。此属性的值可以是字符串或函数。您可以在这里看到一个使用字符串值的示例

{

key: 'text2',

type: 'input',

templateOptions: {

label: 'Hey!',

placeholder: 'This one is disabled if there is no text in the other input',

},

expressionProperties: {

'templateOptions.disabled': '!model.text',

},

},

在建模时字段将被隐藏。文本是空的

函数值示例:

{

key: 'country',

type: 'input',

templateOptions: {

label: 'field 2',

placeholder: ''

},

expressionProperties: {

'templateOptions.disabled': (model: any, formState: any, field: FormlyFieldConfig) => {

// access to the main model can be through `this.model` or `formState` or `model

return !formState.mainModel.text

},

}

}

该字段将在formState.mainModel时隐藏。文本是空的

警告:可以使用formstate存储信息。在这个例子中，我们保留了这个模型。formstate是选项的属性。阅读更多

options = {

formState: {

mainModel: this.model,

},

};

有条件的呈现

隐藏表达式属性用于设置字段的隐藏属性。此属性的值可以是字符串、函数或布尔值。下面是每一个例子。

带字符串值的第一个选项:

你可以在这里看到一个例子

{

key: 'iLikeTwix',

type: 'checkbox',

templateOptions: {

label: 'I like twix',

},

hideExpression: '!model.name',

}

第二个功能值选项:

你可以在这里看到一个例子

{

key: 'country',

type: 'input',

templateOptions: {

label: 'City',

placeholder: 'set to 123'

},

hideExpression: (model: any, formState: any, field: FormlyFieldConfig) => {

// access to the main model can be through `this.model` or `formState` or `model

if (formState.mainModel && formState.mainModel.city) {

return formState.mainModel.city !== "123"

}

return true;

},

},

第三个选择布尔值。你可以在这里看到一个例子

<button (click)="toggle()">Click me</button>

fields: FormlyFieldConfig[] = [

{

key: 'country',

type: 'input',

templateOptions: {

label: 'Street',

placeholder: ''

},

hideExpression: this.show

},

];

toggle(){

this.show = !this.show;

this.fields[1].hideExpression = this.show;

}

# 五、自定义模板

预先构建的模板

虽然建议您创建自己的模板，以实现最终的自定义和灵活性，但您可以使用以下预构建的模板:

Material 、Bootstrap、Ionic、Kendo、PrimeNG

创建自定义模板

创建一个自定义模板非常简单，但也非常灵活，下面的例子展示了我们如何创建一个简单的输入类型:

可以在stackblitz: https://stackblitz.com/edit/ngx-formly-custom-template中找到实际的示例

定义字段类型类和它的模板:

1 - 首先，您必须创建一个表示扩展FieldType类的字段的组件。

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

import { FieldType } from '@ngx-formly/core';

@Component({

selector: 'formly-field-input',

template: `

<input type="input" [formControl]="formControl" [formlyAttributes]="field">

`,

})

export class FormlyFieldInput extends FieldType {}

我们传递了Formly创建的formControl实例，让Formly知道这是您想要与模型关联的输入。

2 - 在NgModule声明中注册自定义类型:

import { FormlyFieldInput } from './formly-field-input';

@NgModule({

declarations: [FormlyFieldInput],

imports: [

....

FormlyModule.forRoot({

types: [

{ name: 'input', component: FormlyFieldInput },

],

}),

],

})

export class AppModule {}

类型允许您指定可在字段配置中使用的自定义类型。

一个典型的类型需要两个属性:

name:模板类型的名称。您可以在字段的类型选项中使用此选项。

组件:当设置此类型时，Formly应该创建的组件。

3 - 在表单config中使用创建的自定义类型:

export class AppComponent {

fields: FormlyFieldConfig[] = [

{

key: 'firstname',

type: 'input',

},

];

...

}

# 六、Custom Wrapper 自定义包装器

自定义包装器允许您用组件包装字段类型。

预先构建的包装

ui-bootstrap

form-field

Shows validation messages bellow field.显示验证消息。

addons

ui-ionic

form-field

Shows validation messages bellow field.显示验证消息。

ui-kendo

form-field

Shows validation messages and label for required.

ui-material

form-field

Has label, shows validation messages, shows description, and label for required.

ui-primeng

form-field

Has label, required and validation message.

创建自定义包装器

创建自定义包装器很容易，下面的示例演示如何围绕字段创建面板包装器。

参见现场演示:演示 https://stackblitz.com/angular/dleylnmrbmd?file=app%2Fapp.component.ts

1 - 定义自定义包装类和它的模板:

首先，您必须创建一个表示扩展FieldWrapper类的包装器的组件。

// panel-wrapper.component.ts

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

import { FieldWrapper } from '@ngx-formly/core';

@Component({

selector: 'formly-wrapper-panel',

template: `

<div class="card">

<h3 class="card-header">Its time to party</h3>

<h3 class="card-header">{{ to.label }}</h3>

<div class="card-body">

<ng-container #fieldComponent></ng-container>

</div>

</div>

`,

})

export class PanelWrapperComponent extends FieldWrapper {

}

fieldComponent是插入字段的地方。(< ng-container # fieldComponent > < / ng-container >)

注意:FieldWrapper组件扩展了Field，因此你可以在一个字段上附加多个wrappers。您可以同时拥有标签和验证器包装器。

2 - 在NgModule声明:Module中注册自定义包装器

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { ReactiveFormsModule } from '@angular/forms';

import { FormlyModule } from '@ngx-formly/core';

import { FormlyBootstrapModule } from '@ngx-formly/bootstrap';

import { PanelWrapperComponent } from './panel-wrapper.component';

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

@NgModule({

imports: [

CommonModule,

ReactiveFormsModule,

FormlyBootstrapModule,

FormlyModule.forRoot({

wrappers: [

{ name: 'panel', component: PanelWrapperComponent },

],

}),

],

declarations: [

AppComponent,

PanelWrapperComponent,

],

})

export class AppModule { }

Wrapper:[…]定义我们想要注入到模块中以在FormlyFieldConfig中使用的自定义包装器

3 - 创建一个使用该类型的定制FormlyFieldConfig。

fields: FormlyFieldConfig[] = [

{

key: 'address',

wrappers: ['panel'],

templateOptions: { label: 'Address' },

fieldGroup: [{

key: 'town',

type: 'input',

templateOptions: {

required: true,

type: 'text',

label: 'Town',

},

}],

},

];

wrappers: ['panel']是FormlyFieldConfig指定字段实例使用该面板的地方。

为组件创建默认包装器

有时，您总是需要带有某些包装器的组件。

@NgModule({

imports: [

CommonModule,

ReactiveFormsModule,

FormlyBootstrapModule,

FormlyModule.forRoot({

types: [

{

name: 'operator',

component: OperatorComponent,

wrappers: ['form-field']

},

],

}),

],

declarations: [

AppComponent,

OperatorComponent

],

})

export class AppModule { }

可以通过在模块中将wrappers: ['form-field']设置为该类型来实现这一点