**属性和选项**

Formly Form

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

Inputs

**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**

表单要表示的模型。.

**options FormlyFormOptions no**

表单的选项。

Outputs

Name Description

(modelChange) 触发了模型值的变化

1 fields

**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。说明如果不设置，则生成该id。

**name string**

如果愿意，可以为字段指定一个特定的名称。如果您使用过去的技术将表单发布到服务器，这是非常有用的。

**type string**

要呈现的字段类型。更多信息见自定义模板。

**component any**

可以设置为替换在类型中定义的组件。

**className string**

你可以指定你自己的类，它将被应用到formly-field指令

**templateOptions object**

这是为模板保留的。任何特定于模板的选项都在这里。查看特定的模板实现，了解所需的选项。

**template string**

可以设置来代替“类型”，以呈现自定义html内容。

**defaultValue any**

如果提供了这个，并且模型在编译时的值未定义，那么模型的值将被分配为defaultValue。

**hide boolean**

是否隐藏字段。默认值为false。如果你希望这是有条件的，使用hideExpression。

**hideExpression boolean or string or function**

根据其他字段的值有条件地隐藏该字段。

**expressionProperties boolean or string or function**

一个对象，其中键是要在主字段配置中设置的属性，值是用于分配该属性的表达式。

**focus boolean**

是否聚焦或模糊元素字段。默认值为false。如果你希望它是有条件的，请使用expressionProperties

**wrappers string[]**

预计这将是包装商的名字。formly字段模板将由第一个包装器包装，然后是第二个，然后是第三个，等等。您还可以将它们指定为类型的一部分(这是推荐的方法)。

**parsers function[]**

要在模型更新时(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，使高级布局非常简单。它还可以用于对与同一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，则很有用)。

**fieldArray FormlyFieldConfig**

**fieldGroupClassName string**

指定您自己的类，该类将应用于正式组组件。

**Validation object**

一个具有几个有用属性的对象:messages, show

**Validators any**

用于为特定字段设置验证规则。应该是键-值对的对象。该值可以是要求值的表达式，也可以是要运行的函数。每个字段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**

对于任何需要异步验证的内容，都可以使用这个方法。与validators api几乎完全相同，除了它必须是一个返回promise的函数。

**formControl AbstractControl**

这是字段的FormControl。它为您提供了更多的控制，如运行验证器、计算状态和重置状态。

**modelOptions object**

一个对象，它有一些有用的属性来控制模型的变化:debounce, updateOn

2 options

**formState**

formState属性被传递给所有字段，是字段之间通信的一种机制(而不必破坏模型)。

**resetModel and updateInitialValue**

这还接收方法resetModel和updateInitialValue，它们将分别调用字段的所有resetModel和updateInitialValue。

**fieldTransform**

允许您在正式处理或验证字段之前修改/转换字段。

# Validation

提供了实现自定义验证的不同方法。

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，您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属性可以只是一个字符串，也可以定义为接收错误和验证为输入的字段(FormlyFieldConfig)的函数。最后一个选项允许您进行更多的定制，比如包含值、允许的最小值或最大值等。

下面的代码示例演示了如何使用formControl来包含一个新消息"ip"和一个生成字符串消息的函数。值和一个新消息“required”作为一个简单的字符串。

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验证IP的函数。可以看到，当出现错误时返回的对象有一个名为“ip”的属性，它与自定义验证消息的名称匹配。

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

你只需要在validators.validation属性中包含FormlyModule中声明的验证函数名。

{

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，你需要在asyncValidators中包含它。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

(如上所述)

自定义验证函数

您可以在字段定义中实现验证函数。例如，如果您想使用几个带有相同错误消息的验证函数，则可以使用此方法。

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validators.validation属性中包含您想要声明的验证函数。

{

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，你可能需要在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。格式将是:

NAME\_OF\_VALIDATOR: {

expression: FUNCTION

message: FUNCTION | STRING

}

带有自定义验证的字段

如下面的代码示例所示，您只需要为验证器属性(ip)指定一个名称，并在表达式属性中包含一个函数，在message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，你可能需要在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. 在正式类型中声明验证函数，在NgModule声明中声明消息。

自定义验证消息

如第1点所述，创建一个自定义验证消息。

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' },

],

}),

]

})

用于正式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，如果没有错误则返回false，否则返回true。

下面的代码示例显示了验证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'匹配ip验证消息

}

},

},

}),

]

})

字段使用具有自定义验证的正式类型

正式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来测试创建自定义验证器的几种方法

# Formly Expressions

您可以使用hideExpression动态隐藏字段，Expression Properties for允许您动态更改字段的许多属性。

Expression Properties

表达式属性允许您动态地更改字段的许多属性。例如，您可以动态禁用一个表单字段。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',

},

},

model.text是空的时，该字段将隐藏，

函数值示例:

{

key: 'country',

type: 'input',

templateOptions: {

label: 'field 2',

placeholder: ''

},

expressionProperties: {

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

// 可以通过“this”访问主模型。'或' formState '或' model '

return !formState.mainModel.text

},

}

}

当formState.mainModel.text是空的，该字段将隐藏

ALERT:可以使用formstate存储信息。在本例中，我们保留了模型。formstate是一个选项属性。阅读更多

options = {

formState: {

mainModel: this.model,

},

};

Conditional Rendering（条件渲染）

HideExpression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。下面是每一种方法的一个例子。

第一个字符串值选项:

{

key: 'iLikeTwix',

type: 'checkbox',

templateOptions: {

label: 'I like twix',

},

hideExpression: '!model.name',

}

第二个函数值选项:

{

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;

},

},

第三个选项，使用布尔值。你可以在这里看到一个例子

HTML

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

TS

fields: FormlyFieldConfig[] = [

{

key: 'country',

type: 'input',

templateOptions: {

label: 'Street',

placeholder: ''

},

hideExpression: this.show

},

];

toggle(){

this.show = !this.show;

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

}

Custom Templates

预先构建的模板

虽然建议创建你自己的模板，最终定制和灵活性，有预构建的模板，你可以使用:

Material Bootstrap Ionic Kendo PrimeNG

创建自定义模板

创建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:

1 定义字段类型类及其模板:

首先，您必须创建一个组件来表示扩展FieldType类的字段。

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

@Component({

selector: 'formly-field-input',

template: `

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

`,

})

export class FormlyFieldInput extends FieldType {}

我们传递了一个formControl实例，它是Formly创建的，以让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 {}

types 允许您指定可在字段配置中使用的自定义类型。

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type选项中使用它。

component:设置此类型时应该创建的组件。

3 在配置表单中使用创建的自定义类型:

export class AppComponent {

fields: FormlyFieldConfig[] = [

{

key: 'firstname',

type: 'input',

},

];

...

}

定制的包装

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-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。

1 定义自定义包装类及其模板:

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是字段插入的地方吗. (<ng-container #fieldComponent></ng-container>)

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

2 在NgModule声明中注册自定义包装器:Module

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

@NgModule({

imports: [

FormlyModule.forRoot({

wrappers: [

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

],

}),

],

declarations: [

PanelWrapperComponent,

],

})

export class AppModule { }

wrappers: [ ... ] 在FormlyFieldConfig中定义了我们想要注入到模块中的自定义包装器吗

3 创建一个使用该类型的自定义FormlyFieldConfig。

fields: FormlyFieldConfig[] = [

{

key: 'address',

wrappers: ['panel'],

templateOptions: { label: 'Address' },

fieldGroup: [{

key: 'town',

type: 'input',

templateOptions: {

required: true,

type: 'text',

label: 'Town',

},

}],

},

];

wrappers: ['panel'], FormlyFieldConfig在哪里分配字段实例来使用那个面板

4 为组件创建默认包装器

有时，您总是需要一个具有特定包装的组件。

@NgModule({

imports: [

FormlyModule.forRoot({

types: [

{

name: 'operator',

component: OperatorComponent,

wrappers: ['form-field']

},

],

}),

],

declarations: [

OperatorComponent

],

})

export class AppModule { }

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