1. **入门指南 （https://github.com/udos86/ng-dynamic-forms）**

tyarn add @ng-dynamic-forms/core

tyarn add @ng-dynamic-forms/ui-material

1. **基本用法**

1 、导入UI模块

import { DynamicFormsMaterialUIModule } from "@ng-dynamic-forms/ui-material";

@NgModule({

imports: [ReactiveFormsModule, DynamicFormsMaterialUIModule]

})

export class AppModule {}

2. 定义表单模型:

import {

DynamicFormModel,

DynamicCheckboxModel,

DynamicInputModel,

DynamicRadioGroupModel

} from "@ng-dynamic-forms/core";

export const MY\_FORM\_MODEL: DynamicFormModel = [

new DynamicInputModel({

id: "sampleInput",

label: "Sample Input",

maxLength: 42,

placeholder: "Sample input"

}),

new DynamicRadioGroupModel<string>({

id: "sampleRadioGroup",

label: "Sample Radio Group",

options: [

{

label: "Option 1",

value: "option-1",

},

{

label: "Option 2",

value: "option-2"

},

{

label: "Option 3",

value: "option-3"

}

],

value: "option-3"

}),

new DynamicCheckboxModel({

id: "sampleCheckbox",

label: "I do agree"

})

];

3. 通过DynamicFormService创建一个FormGroup:

import { MY\_FORM\_MODEL } from "./my-dynamic-form.model";

import { DynamicFormModel, DynamicFormService } from "@ng-dynamic-forms/core";

export class MyDynamicFormComponent implements OnInit {

formModel: DynamicFormModel = MY\_FORM\_MODEL;

formGroup: FormGroup;

constructor(private formService: DynamicFormService) {}

ngOnInit() {

this.formGroup = this.formService.createFormGroup(this.formModel);

}

}

4. 将DynamicFormComponent添加到模板并绑定它的[group]和[model]属性:

<form [formGroup]="formGroup">

<dynamic-material-form [group]="formGroup" [model]="formModel">

</dynamic-material-form>

</form>

1. **用户界面模块**

Material 、NG Bootstrap 、ngx-bootstrap、 PrimeNG

你可以立即插入你最喜欢的形式控制安装适当的包和它的对等依赖:

npm i @ng-dynamic-forms/ui-<library-name> -S

yarn add @ng-dynamic-forms/ui-material

现在只需导入UI模块:

@NgModule({

imports: [

ReactiveFormsModule,

DynamicFormsMaterialUIModule

]

})

export class AppModule {}

为了创建表单标记，所有UI模块都带有一个DynamicFormComponent，可以很容易地添加到组件模板中:

<form [formGroup]="formGroup">

<dynamic-material-form [group]="formGroup" [model]="formModel">

</dynamic-material-form>

</form>

或者，你可以直接使用一个特定的DynamicFormControlComponent来获得对渲染的更多控制:

<form [formGroup]="formGroup">

<dynamic-material-form-control \*ngFor="let controlModel of formModel"

[group]="formGroup" [model]="controlModel">

</dynamic-material-form-control>

</form>

由于技术限制或外部依赖关系仍在开发中，各UI包对主要表单控件的支持各不相同。兼容性见下表:

https://github.com/udos86/ng-dynamic-forms

1. **Form Groups**

为了提高明确性，通常认为将表单分组为几个逻辑fieldset部分是很好的实践。

因此，NG动态表单支持开箱即用的表单组嵌套!

1. 在你的表单模型中声明一个DynamicFormGroupModel，并将它的模型添加到组数组中:

export const MY\_FORM\_MODEL: DynamicFormModel = [

new DynamicFormGroupModel({

id: "fullName",

legend: "Name",

group: [

new DynamicInputModel({

id: "firstName",

label: "First Name"

}),

new DynamicInputModel({

id: "lastName",

label: "Last Name"

})

]

}),

new DynamicFormGroupModel({

id: "address",

legend: "Address",

group: [

new DynamicInputModel({

id: "street",

label: "street"

}),

new DynamicInputModel({

id: "zipCode",

label: "Zip Code"

})

]

})

];

2. 创建一个FormGroup并添加一个DynamicFormComponent:

ngOnInit() {

this.formGroup = this.formService.createFormGroup(this.formModel);

}

<form [formGroup]="formGroup">

<dynamic-material-form [group]="formGroup" [model]="formModel">

</dynamic-material-form>

</form>

3.要操作一个现有的DynamicFormGroupModel，你可以简单地使用DynamicFormService:

addFormGroupControl(...)

insertFormGroupControl(...)

moveFormGroupControl(...)

removeFormGroupControl(...)

1. **Form Arrays**

有时表单需要允许用户动态地添加多个相同类型的项，例如地址、产品等等。

特别是因为这个原因，Angular提供了所谓的表单数组。幸运的是，NG动态表单能够管理这样的嵌套表单结构!

1. 添加一个动态formarraymodel到你的表单模型:

export const MY\_FORM\_MODEL: DynamicFormModel = [

new DynamicFormArrayModel({

id: "myFormArray"

})

];

2. 将groupFactory属性添加到DynamicFormArrayModel中，并为其分配一个函数，该函数返回单个表单数组项的结构:

new DynamicFormArrayModel({

id: "myFormArray",

initialCount: 5,

groupFactory: () => {

return [

new DynamicInputModel({

id: "myInput",

label: "My Input"

})

];

}

})

3.通过DynamicFormService创建一个FormGroup，并将其绑定到组件模板:

this.formGroup = this.formService.createFormGroup(this.formModel);

<form [formGroup]="formGroup">

<dynamic-material-form [group]="formGroup" [model]="formModel">

</dynamic-material-form>

<button type="button" (click)="addItem()">Add item</button>

<button type="button" (click)="clear()">Remove all items</button>

</form>

4. 你现在可以很容易地修改你的表单数组与DynamicFormService:

ngOnInit() {

this.formArrayModel =

this.formService.findModelById<DynamicFormArrayModel>("myFormArray", this.formModel);

this.formArrayControl =

this.formService.findControlByModel<FormArray>(this.formArrayModel, this.formGroup);

}

addItem() {

this.formService.addFormArrayGroup(this.formArrayControl, this.formArrayModel);

this.formService.detectChanges();

}

clear() {

this.formService.clearFormArray(this.formArrayControl, this.formArrayModel);

this.formService.detectChanges();

}

永远不要忘记在运行时更新表单时通过detectChanges触发更改检测!

好吧，效果不错!

但是如果我们想为每个数组组附加一个额外的remove <button>呢?

特别是在这种情况下，你可以添加<ng-template>并声明一些自定义内容，这些内容对所有的表单数组组都是一样的:

<form [formGroup]="formGroup">

<dynamic-material-form [group]="formGroup" [model]="formModel">

<ng-template modelId="myFormArray">

<button type="button" (click)="onClick()">Label</button>

</ng-template>

</dynamic-material-form>

</form>

当<ng-template>应用于DynamicFormArrayModel时，NgTemplateOutletContext被内部绑定到相关的DynamicFormArrayGroupModel。

这意味着您可以通过声明本地默认模板变量或单独的本地模板变量来访问group对象及其属性。

参见自定义模板一章

<form [formGroup]="formGroup">

<dynamic-material-form [group]="formGroup" [model]="formModel">

<ng-template modelId="myFormArray" let-group let-index="index" let-context="context">

<button type="button" (click)="removeItem(context, index)">

Remove Item </button>

<button type="button" (click)="insertItem(group.context, group.index + 1)">

Add Item </button>

</ng-template>

</dynamic-material-form>

</form>

这是非常有用的，当你想实现一个删除或插入功能:

removeItem(context: DynamicFormArrayModel, index: number) {

this.formService.removeFormArrayGroup(index, this.formArrayControl, context);

this.formService.detectChanges();

}

insertItem(context: DynamicFormArrayModel, index: number) {

this.formService.insertFormArrayGroup(index, this.formArrayControl, context);

this.formService.detectChanges();

}

再次使用DynamicFormService，你甚至可以动态改变表单数组中组的顺序:

this.formService.moveFormArrayGroup(index, -1, this.formArrayControl, context);

this.formService.detectChanges();

1. **Form Layouts**

Form Layouts

当使用NG动态表单UI包时，例如UI -bootstrap，底层CSS库的所有基本表单类(如form-group或form-control)都会自动放置在对应DynamicFormControlComponent组件的模板中。

除此之外，NG动态表单并没有对可选的CSS类做任何进一步的假设，而是将高级布局留给了您。这是坚实的，但不唐突。

假设我们要实现一个漂亮的对齐Bootstrap horizonal form...

首先，我们必须在模板中的<form>元素中附加强制的引导CSS类form-horizontal:

<form class="form-horizontal" [formGroup]="formGroup">

<dynamic-bootstrap-form [group]="formGroup" [model]="formModel">

</dynamic-bootstrap-form>

</form>

现在我们需要使用引导网格系统来定位<label>和form-control。

但是，由于表单控件的所有模板逻辑都封装在组件范围内，所以我们不能直接将那些必要的CSS类附加到标记中。

别担心!

通过将DynamicFormLayout绑定到DynamicFormControlComponent，可以为任何表单控件提供任意CSS类。

DynamicFormLayout是一个简单的对象文字，它将CSS类配置对象与模型id关联起来。

通过区分元素和网格上下文，NG动态表单可以自动应用基于位置标识符的组件模板中的CSS类字符串:

export const MY\_FORM\_LAYOUT = {

"myFormControlModelId": {

element: { label: "control-label" },

grid: {

control: "col-sm-9",

label: "col-sm-3"

}

},

"myOtherFormControlModelId": {

element: { label: "control-label" },

grid: {

control: "col-sm-9",

label: "col-sm-3"

}

}

};

为了引用这个DynamicFormLayout，我们现在只需要创建另一个组件类成员:

import { MY\_FORM\_LAYOUT } from "./my-dynamic-form.layout";

export class MyDynamicFormComponent implements OnInit {

formModel: DynamicFormModel = MY\_FORM\_MODEL;

formGroup: FormGroup;

formLayout: DynamicFormLayout = MY\_FORM\_LAYOUT;

constructor(private formService: DynamicFormService) {}

ngOnInit() {

this.formGroup = this.formService.createFormGroup(this.formModel);

}

}

最后，我们通过输入绑定将表单布局传递给DynamicFormComponent:

<form [formGroup]="formGroup">

<dynamic-bootstrap-form [group]="formGroup"

[layout]="formLayout"

[model]="formModel"></dynamic-bootstrap-form>

</form>

使用这种方法，我们能够严格地将布局信息从纯表单模型中分离出来。

1. **表单控件的配置 Form Control Configuration**

无论您使用哪个UI库，通常都有一组基本属性，它们普遍适用于特定类型的表单控件。

因此，当选择例如一个滑动器组件时，你总是会找到一个@Input()来控制它的最小/最大值和它的方向。

在这种情况下，NG动态表单直接在相应的DynamicFormControlModel上提供了一个抽象的配置属性:

new DynamicSliderModel({

id: "mySlider",

min: 0,

max: 10,

vertical: true

})

此外，通常还有完全特定于库的附加组件特性。

因此，在使用例如Material时，有一个唯一的@Input()来反转一个slider组件。

NG动态表单还允许您自由地使用这样的单个参数。

你需要做的就是把它放在你的DynamicFormValueControlModel的额外配置对象:

new DynamicSliderModel({

id: "mySlider",

min: 0,

max: 10,

vertical: true

additional: {

invert: true

}

})

1. 表单控件事件 **Form Control Events**

在开发表单时，跟踪特定表单控件上发生的特定事件通常很有用。

有了NG Dynamic Forms，你可以直接听到三个最常见的事件，blur, change and focus，都在DynamicFormControlComponent和DynamicFormComponent:

<dynamic-material-form [group]="formGroup"

[model]="formModel" (blur)="onBlur($event)"

(change)="onChange($event)"(focus)="onFocus($event)">

</dynamic-material-form>

<form [formGroup]="myFormGroup">

<dynamic-material-form-control \*ngFor="let controlModel of myFormModel"

[group]="myFormGroup" [model]="controlModel"

(blur)="onBlur($event)" (change)="onChange($event)" (focus)="onFocus($event)">

</dynamic-material-form-control>

</form>

传递给处理程序函数的对象为您提供进一步处理所需的任何控制和模型信息。

$event属性甚至授予对原始事件的访问权:

interface DynamicFormControlEvent {

$event: Event | FocusEvent | DynamicFormControlEvent | any;

context: DynamicFormArrayGroupModel | null;

control: FormControl;

group: FormGroup;

model: DynamicFormControlModel;

type: string;

}

但在使用UI库时，通常会为某些表单控件组件提供一组附加事件。

当然，NG动态表单不会让您失望。

所有自定义UI事件都由单独的@Output()使用各自的库前缀汇集在一起。

<dynamic-material-form [group]="formGroup" [model]="formModel"

(matEvent)="onMatEvent($event)">

</dynamic-material-form>

1. **更新表单控件**

NG动态表单完全依赖于Angular的ReactiveFormsModule。因此，DynamicFormValueControlModel的值属性不能通过[(ngModel)]进行双向绑定。而且，从RC.6开始，Angular就不允许在响应式表单中绑定disabled属性。

然而，在运行时更新表单控件的值或状态可以很容易地实现。首先我们需要得到它的DynamicFormControlModel表示的引用:

const inputModel = this.formService

.findModelById<DynamicInputModel>("myInput", this.formModel);

在那之后，我们只是把DynamicFormValueControlModel的方便值和禁用setter发挥作用，我们很好:

inputModel.value = "New Value";

inputModel.disabled = true;

修改立即反映在用户界面中。到目前为止一切顺利。

但是其他数据呢?既然DynamicFormControlModel通过Angular核心机制直接绑定到一个DOM元素，那么更改它的一个属性也应该自动触发用户界面的更新，对吗?

现在要小心!

由于性能原因，动态表单采用了变化检测策略。按下引擎盖。因此，除了值和禁用之外，更改DynamicFormControl上的任何属性都不会导致自动DOM更新。

相反，您必须在更新模型之后调用DynamicFormService上的detectChanges()，以指示库应该手动触发更改检测。

inputModel.label = "New Label";

this.formService.detectChanges();

要优化这一点，您可以有选择地传递DynamicFormComponent来检测更改()，以减少即将到来的更改检测所影响的元素数量:

@ViewChild(DynamicMaterialFormComponent, {static: false})

formComponent: DynamicMaterialFormComponent;

//...

this.formService.detectChanges(this.formComponent);

1. **自定义模板**

如前所述，通过CSS类，NG动态表单在调整表单布局方面给了您很大的自由。

但是，在某些情况下，还需要为一些表单控件添加自定义标记。

要做到这一点，只需将<ng-template>放入动态表单控件元素中，并设置一个modelId属性将其分配给某个控件。

<form [formGroup]="formGroup">

<dynamic-material-form [group]="formGroup" [model]="formModel">

<ng-template modelId="myInput">

<p>Some custom markup</p>

</ng-template>

</dynamic-material-form>

</form>

或者，你也可以应用modelType而不是modelId来重用同一类型的几个表单控件的模板:

<form [formGroup]="formGroup">

<dynamic-material-form [group]="formGroup" [model]="formModel">

<ng-template modelType="ARRAY">

<p>Just some custom markup</p>

</ng-template>

</dynamic-material-form>

</form>

而且越来越好!

因为对于每个模板NgTemplateOutletContext内部绑定到相应的DynamicFormControlModel，你可以使用本地模板变量来引用你的模型的属性:

<form [formGroup]="formGroup">

<dynamic-material-form [group]="formGroup" [model]="formModel">

<ng-template modelId="myInput" let-id="id">

<p>Some custom markup for {{ id }}</p>

</ng-template>

</dynamic-material-form>

</form>

还不相信吗?

一些UI库，例如Kendo UI，允许通过模板指令对表单控件进行详细的定制。

你肯定不想错过这样一个功能，是吗?

这就是为什么NG动态表单可以掌握这一点!

你所要做的就是添加一个as属性到你的模板，并指定它的使用:

<form [formGroup]="formGroup">

<dynamic-kendo-form [group]="formGroup" [model]="formModel">

<ng-template modelId="myDropDownList" as="kendoDropDownListHeaderTemplate">

<p>My Kendo DropDown Header Template</p>

</ng-template>

</dynamic-kendo-form>

</form>

最后，您可以使用align属性来确定模板是在实际表单控件之前还是之后呈现:

<form [formGroup]="formGroup">

<dynamic-material-form [group]="formGroup" [model]="formModel">

<ng-template modelId="myInput" align="START">

<p>Some custom markup</p>

</ng-template>

</dynamic-material-form>

</form>

1. **定制验证器**

向任何DynamicFormControlModel中添加内置的Angular验证器都非常简单!

只需在Validators或asyncValidators配置对象中引用Validators类的函数名:

new DynamicInputModel({

id: "myInput",

label: "My Input",

validators: {

required: null,

minLength: 3

}

})

到目前为止一切顺利!

但是如果您想引入一些定制的验证器呢?

export function myCustomValidator(control: AbstractControl):

ValidationErrors | null {

let hasError =

control.value ? (control.value as string).startsWith("abc") : false;

return hasError ? {myCustomValidator: true} : null;

}

只需通过默认的NG\_VALIDATORS或NG\_ASYNC\_VALIDATORS令牌提供验证器函数:

@NgModule({

// ...

providers: [

{provide: NG\_VALIDATORS, useValue: myCustomValidator, multi: true}

]

})

Note: thoughtram.io - Custom Validators in Angular 2

https://blog.thoughtram.io/angular/2016/03/14/custom-validators-in-angular-2.html

现在你可以将你的自定义验证器应用到你的模型上了:

new DynamicInputModel({

id: "myInput",

label: "My Input",

validators: {

myCustomValidator: null

}

})

但是要小心!有一个陷阱!

在内部，NG动态表单根据提供的验证器的函数名解析它。

不过，当为生产而丑化代码时，这些信息将不可挽回地丢失。

为了避免这个问题，NG Dynamic Forms提供了一个特殊的InjectionToken<Map<string, Validator | ValidatorFactory>>命名为DYNAMIC\_VALIDATORS，你还应该为它提供定制的验证函数:

providers: [

{

provide: NG\_VALIDATORS,

useValue: myCustomValidator,

multi: true

},

{

provide: DYNAMIC\_VALIDATORS,

useValue: new Map<string, Validator | ValidatorFactory>([

["myCustomValidator", myCustomValidator]

])

}

]

你也可以在同一个输入bu上有多个验证器，在useValue中提供多个键|值对:

providers: [

{

provide: NG\_VALIDATORS,

useValue: myCustomValidator,

multi: true

},

{

provide: NG\_VALIDATORS,

useValue: myOtherCustomValidator,

multi: true

},

{

provide: DYNAMIC\_VALIDATORS,

useValue: new Map<string, Validator | ValidatorFactory>([

["myCustomValidator", myCustomValidator],

["myOtherCustomValidator", myOtherCustomValidator]

])

}

]

另一个适用于大多数情况的解决方案是使用交替验证符号:

new DynamicInputModel({

id: "myInput",

label: "My Input",

validators: {

myCustomValidator: {

name: myCustomValidator.name,

args: null

}

}

})

1. **自定义表单控件**

从版本6 NG动态表单开始，您可以轻松地在自定义表单控件中插入插件。

事先遵循标准程序，以建立您自定义的Angular表单控件:

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

import { ControlValueAccessor, NG\_VALUE\_ACCESSOR } from "@angular/forms";

@Component({

selector: 'my-custom-form-control',

templateUrl: './my-custom-form-control.component.html',

providers: [

{

provide: NG\_VALUE\_ACCESSOR,

useExisting: forwardRef(() => MyCustomFormControlComponent),

multi: true

}

]

})

export class MyCustomFormControlComponent implements ControlValueAccessor {

//...

}

现在创建一个新的DynamicFormControlComponent:

import { ChangeDetectionStrategy, Component,

EventEmitter, Input, Output, ViewChild } from "@angular/core";

import { FormGroup } from "@angular/forms";

import {

DynamicFormControlComponent,

DynamicFormControlCustomEvent,

DynamicFormLayout,

DynamicFormLayoutService,

DynamicFormValidationService,

} from "@ng-dynamic-forms/core";

import { MyCustomFormControlComponent } from "...";

@Component({

selector: "my-dynamic-custom-form-control",

templateUrl: "./my-dynamic-custom-form-control.component.html",

changeDetection: ChangeDetectionStrategy.OnPush

})

export class MyDynamicCustomFormControlComponent

extends DynamicFormControlComponent {

@Input() group: FormGroup;

@Input() layout: DynamicFormLayout;

@Input() model: /\* corresponding DynamicFormControlModel \*/;

@Output() blur: EventEmitter<any> = new EventEmitter();

@Output() change: EventEmitter<any> = new EventEmitter();

@Output() customEvent: EventEmitter<DynamicFormControlCustomEvent>

= new EventEmitter();

@Output() focus: EventEmitter<any> = new EventEmitter();

@ViewChild(MyCustomFormControlComponent)

myCustomFormControlComponent: MyCustomFormControlComponent;

constructor(protected layoutService: DynamicFormLayoutService,

protected validationService: DynamicFormValidationService) {

super(layoutService, validationService);

}

}

接下来将自定义表单控件嵌入到组件模板中:

<ng-container [formGroup]="group">

<my-custom-form-control [formControlName]="model.id"

[name]="model.name"

[ngClass]="[

getClass('element', 'control'), getClass('grid', 'control')]"

(blur)="onBlur($event)" (change)="onChange($event)"

(focus)="onFocus($event)">

</my-custom-form-control>

</ng-container>

然后添加你新实现的DynamicFormControl到你的app模块的entryComponents:

entryComponents: [MyDynamicCustomFormControlComponent]

最后，提供DYNAMIC\_FORM\_CONTROL\_MAP\_FN来覆盖具体DynamicFormControlModel到其对应的DynamicFormControlComponent的默认映射;

providers: [

{

provide: DYNAMIC\_FORM\_CONTROL\_MAP\_FN,

useValue: (model: DynamicFormControlModel): Type<DynamicFormControl> | null => {

switch (model.type) {

case /\* corresponding DynamicFormControlModel \*/:

return MyDynamicCustomFormControlComponent;

}

}

}

]

1. **验证消息**

向用户交付有意义的验证信息是良好表单设计的重要组成部分。

然而，HTML5已经提供了一些本地功能，你很可能希望使用Angular机制来获得对验证逻辑和相应消息输出的更多控制。

为了避免一开始就过于固执己见，NG动态表单最初开发时并没有考虑任何突兀的验证消息系统。

然而，由于其非常常见的用例，基于模型的错误消息传递最终成为一个可选的内置特性。

只需添加一个errorMessages对象到任何DynamicFormControlModel，并分配基于验证器名称的错误消息模板:

new DynamicInputModel({

id: "myInput",

label: "My Input",

validators: {

required: null

},

errorMessages: {

required: "{{ label }} is required."

}

})

注意:错误消息模板允许以下占位符:

其中propertyName是模型的属性，例如{{label}}。

{{ validator.propertyName }}。其中propertyName是验证函数返回的对象的属性，例如{{validator。在模式验证器的情况下requiredPattern}}。

{{ propertyName }} where propertyName is a property of the model, for example {{ label }}.

{{ validator.propertyName }} where propertyName is a property of the object returned by validation function, for example {{ validator.requiredPattern }} in case of pattern validator.

只要在DynamicFormControlModel上声明了errorMessages，就会自动启用错误消息传递。

默认情况下，当相应的表单控件无效且同时被模糊且已经被触摸时，将显示错误消息。

由于显示错误消息的条件通常取决于个人用户体验偏好，您可以通过dynamic\_error\_messagesmatcher注入令牌提供一个所谓的DynamicErrorMessagesMatcher来定制它:

export const MY\_CUSTOM\_ERROR\_MESSAGES\_MATCHER =

(control: AbstractControl, model: DynamicFormControlModel, hasFocus: boolean) => { return //... };

providers: [

{

provide: DYNAMIC\_ERROR\_MESSAGES\_MATCHER,

useValue: myCustomErrorMessagesMatcher

},

]

默认情况下，DEFAULT\_ERROR\_STATE\_MATCHER是活动的:

export const DEFAULT\_ERROR\_STATE\_MATCHER: DynamicErrorMessagesMatcher =

(control: AbstractControl, model: DynamicFormControlModel,

hasFocus: boolean) => {

return control.touched && !hasFocus;

};

这里请注意，NG动态表单总是假设控件无效，并且将模型上定义的错误消息作为固定的先决条件。

1. **相关表单控件**

在许多形式中，某个窗体控件的状态直接取决于其他窗体控件的值或状态。

手动实现这样的连接将非常耗时，并且只会产生不需要的样板代码。

NG动态表单允许您使用所谓的DynamicFormControlMatchers以声明方式添加表单控制关系。

matcher定义当相关表单控件上的值或状态发生更改时，应该对预定义匹配执行的操作。

export interface DynamicFormControlMatcher {

match: string;

opposingMatch: string | null;

onChange(hasMatch: boolean, model: DynamicFormControlModel,

control: FormControl, injector: Injector): void;

}

目前有以下默认匹配器可用

DisabledMatcher

HiddenMatcher

RequiredMatcher

注意:一定要确保在你的app.module中提供了每一个DynamicFormControlMatcher:

providers: [

// ...

DISABLED\_MATCHER,

REQUIRED\_MATCHER

]

这样你也可以完全自由地实现自己的自定义匹配器:

export const MyCustomMatcher: DynamicFormControlMatcher = {

match: MATCH\_CUSTOM,

opposingMatch: MATCH\_CUSTOM\_OPPOSITE,

onChange(hasMatch: boolean, model: DynamicFormControlModel): void {

if (hasMatch) {

console.log("It's a match");

}

}

};

export const MY\_CUSTOM\_MATCHER: ValueProvider = {

provide: DYNAMIC\_MATCHERS,

useValue: MyCustomMatcher,

multi: true

};

假设我们需要在选择菜单mySelect的第三个选项被选中时禁用textarea myTextArea。

只需将一个关系属性添加到DynamicFormControlModel中，然后通过为某个DynamicFormControlCondition设置一个匹配项来声明一个dynamicformcontrolrelationship:

new DynamicTextAreaModel(

{

id: "myTextArea",

label: "My Textarea",

relations: [

{

match: MATCH\_DISABLED,

when: [

{ id: "mySelect", value: "option-3" }

]

}

]

}

就是这样——库会自动将所有的片段添加到引擎盖下。

但是如果myTextArea也依赖于另一个控件myRadioGroup呢?

幸运的是，还有对多相关表单控件的支持。

只需添加第二个DynamicFormControlCondition条目并(可选地)定义所有条件应该如何通过操作符逻辑连接:

new DynamicTextAreaModel(

{

id: "myTextArea",

label: "My Textarea",

relations: [

{

match: MATCH\_DISABLED,

operator: AND\_OPERATOR,

when: [

{ id: "mySelect", value: "option-3" },

{ id: "myRadioGroup", value: "option-4" }

]

}

]

}

)

1. **JSON导出&导入**

您可能迟早会希望持久化动态表单模型，以便在将来某个时候恢复它。

这就是为什么所有的DynamicFormControlModels都准备好适当地导出到JSON:

storeForm() {

let json: string = JSON.stringify(this.formModel);

// ...store JSON in localStorage or transfer to server

}

为了从JSON中重新创建一个表单，只需使用DynamicFormService提供的相应函数:

restoreForm() {

let json: string;

// ...load JSON from localStorage or server

this.formModel = this.formService.fromJSON(json);

}

1. **JSON Form Models**

默认情况下，NG动态表单包含原型继承，并强制您在建模表单时使用构造函数。

不过，根据您的一般设置或个人偏好，有时更适合用纯JSON提供表单模型。

幸运的是，这是非常好的，也得到了支持。

要指定一个单独的JSON表单控件模型，只需分配mandatory type属性:

storeForm() {

let json: string = JSON.stringify(this.formModel);

// ...store JSON in localStorage or transfer to server

}

为了从JSON中重新创建一个表单，只需使用DynamicFormService提供的相应函数:

restoreForm() {

let json: string;

// ...load JSON from localStorage or server

this.formModel = this.formService.fromJSON(json);

}

1. **JSON形式的模型**

默认情况下，NG动态表单包含原型继承，并强制您在建模表单时使用构造函数。

不过，根据您的一般设置或个人偏好，有时更适合用纯JSON提供表单模型。

幸运的是，这是非常好的，也得到了支持。

要指定一个单独的JSON表单控件模型，只需分配mandatory type属性:

[

{

"type": "INPUT",

"id": "sampleInput",

"label": "Sample Input",

"maxLength": 42,

"placeholder": "Sample input"

},

{

"type": "RADIO\_GROUP",

"id": "sampleRadioGroup",

"label": "Sample Radio Group",

"options": [

{

"label": "Option 1",

"value": "option-1",

},

{

"label": "Option 2",

"value": "option-2"

},

{

"label": "Option 3",

"value": "option-3"

}

],

"value": "option-3"

},

{

"type": "CHECKBOX",

"id": "sampleCheckbox",

"label": "I do agree"

}

]

在将JSON表单模型异步加载到应用程序中之后，不要忘记在创建FormGroup之前通过fromJSON()转换它。

ngOnInit() {

this.http.get<object[]>('./app/my-dynamic-form.model.json').subscribe(formModelJson => {

this.formModel = this.formService.fromJSON(formModelJson);

this.formGroup = this.formService.createFormGroup(this.formModel);

});

}

1. **Text Masks**

每当需要以预定义的值格式填充<input>元素时，文本掩码就会对表单进行很好的增强，以指导用户。

由于Angular在默认情况下没有提供合适的特性，所以NG动态表单集成了一个外部文本掩码指令。

这就是为什么大多数UI包需要安装一个额外的对等依赖:

npm install angular2-text-mask --save

你现在可以根据文本蒙版文档向任何DynamicInputModel添加蒙版属性:

new DynamicInputModel({

id: "maskedInput",

label: "Masked Input",

mask: ['(', /[1-9]/, /\d/, /\d/, ')', ' ', /\d/, /\d/, /\d/, '-', /\d/, /\d/, /\d/, /\d/],

}),

还可以将函数作为掩码传递。该函数将在每次更改时接收用户输入。该函数将返回如上所述的掩码数组。文字蒙版插件

https://github.com/text-mask/text-mask/tree/master/addons/

new DynamicInputModel({

id: "maskedInput",

label: "Masked Input",

mask: createNumberMask({

prefix: "",

suffix: " $"

}),

}),

请注意，像Kendo UI这样的一些UI库带有自己的文本掩码实现，它们可能依赖于不同的文本掩码字符串/正则表达式。

1. **Autocompletion**

添加自动输入完成可以成为良好用户体验的关键因素(特别是在移动设备上)，在设计表单时应该始终加以考虑。

这就是为什么NG动态表单会让您在这里继续学习!

根据HTML5标准行为，在默认情况下，autocomplete属性总是绑定到任何DynamicFormTextInputControl表单元素的on。然而，你可以完全禁用这一功能，显式地设置相应的模型属性为off:

import { AUTOCOMPLETE\_OFF } from "@ng-dynamic-forms/core";

let model = new DynamicInputModel({

id: "myInput",

label: "My Input",

autoComplete: AUTOCOMPLETE\_OFF

});

此外，NG Dynamic Forms包含全新的HTML5 autofill detail token，通过提供AUTOFILL\_<TOKEN\_NAME|FIELD\_NAME>字符串常量和AutoFillUtils来帮助您组合一个有效的表达式:

注意:Jason Grigsby - 自动填充:web开发人员应该知道什么，但是不应该知道什么

https://cloudfour.com/thinks/autofill-what-web-devs-should-know-but-dont/

import {

AutoFillUtils,

AUTOFILL\_TOKEN\_BILLING,

AUTOFILL\_FIELD\_NAME,

AUTOCOMPLETE\_ON

} from "@ng-dynamic-forms/core";

export class MySample {

constructor() {

let expression = `${AUTOFILL\_TOKEN\_BILLING} ${AUTOFILL\_FIELD\_NAME}`;

let model = new DynamicInputModel({

id: "myInput",

label: "My Input",

autoComplete: AutoFillUtils.validate(expression) ? expression : AUTOCOMPLETE\_ON

});

}

}

此外，你还可以通过设置DynamicInputControlModel的list属性来提供HTML5 datalists:

new DynamicInputModel({

id: "myInput",

label: "My Input",

list: ["Alabama", "Alaska", "Arizona", "Arkansas"]

})

1. **常见问题解答**

1 、为什么要使用NG动态表单?

你的Angular表单将变得高度可维护性，因为你再也不用关心保持模板标记和程序代码的同步了。

2、何时使用NG动态表单?

每当你的Angular应用被多个复杂的表单驱动时。

3、什么时候不应该使用NG动态表单?

当你的Angular应用不得不只显示非常简单的表单或者非常独立的表单UI时。

4、使用NG动态表单有什么缺点吗?

对于非常个性化的表单布局存在一定的限制。

5、NG动态表单支持自定义表单控件吗?

是的,确实。

6、Angular还有其他动态表单库吗?

是的，也就是ng-formly, ngx-forms和angular-formio。

7、除了提供问题或代码之外，我如何支持这个项目?

星存储库。