**Custom - async**

https://docs.rxweb.io/form-validations/async/template-driven/directives

异步验证模板驱动将允许该字段验证来自服务器的特定字段的唯一值。这种异步验证可以根据需要进行定制。

假设您想要创建一个用户表单，其中包含像userName和email这样的字段，并且您希望用户输入唯一的userName。这个用户名的唯一值将从服务器验证。您可以创建一个自定义验证器函数。在这里，根据需求可能会出现这些场景。

对userName字段应用异步验证，以检查用户输入的内容是否唯一。

export class AsyncCompleteTemplateDrivenValidationDirectivesComponent implements OnInit {

countryNameUnique(control: FormControl) {

const promise = new Promise((resolve, reject) => {

if (control.value != "India")

resolve(null)

else

resolve({ 'async':{message: '值应该是唯一的'} })

});

return promise;

}

}

<input type="text" name="countryName" [(ngModel)]="user.countryName"

class="form-control" [async]="countryNameUnique.bind(this)"/>

**Decorator**

**基于全局函数的异步示例**

当你想在多个组件中全局使用异步验证器函数时，你可以使用基于全局的异步函数

export class User {

**@async([ isUniqueUserName ])**

userName: string;

}

function isUniqueUserName(control: FormControl) {

const promise = new Promise((resolve, reject) => {

if (!control.value == false)

resolve(null)

else

resolve({ 'async':{message: 'You must enter a value'} })

});

return promise;

}

**基于组件函数的异步示例**

当你想在你的组件中使用异步验证器函数时，你可以使用基于组件的异步函数。

export class User {

@prop()

userName: string;

}

export class AsyncComponentComponent implements OnInit {

userFormGroup: FormGroup

constructor( private formBuilder: RxFormBuilder) { }

ngOnInit() {

let user = new User();

var formBuilderConfig = new FormBuilderConfiguration();

formBuilderConfig.dynamicValidation = {

userName: {

async: [this.**isUniqueUserName**()]

}

}

this.userFormGroup = this.formBuilder.formGroup(user, formBuilderConfig);

}

**isUniqueUserName**(): AsyncValidatorFn {

return (control: FormControl) => {

const promise = new Promise((resolve, reject) => {

if (!control.value == false)

resolve(null)

else

resolve({ 'async':{message: 'You must enter a value'} })

});

return promise;

}

}

}

**Custom - compose**

https://docs.rxweb.io/form-validations/compose/validators#composeconfig

组合验证验证器用于在特定的formControl上应用多个验证。

假设您想要创建UserInfo表单，其中包含诸如firstName、lastName、age、cityName、countryName等字段，并对这些字段应用多个验证。在这里，根据需求可能会出现这些场景。

1 对firstName字段应用compose验证。

2 使用消息键在lastName字段上应用compose验证。

3 根据表单中匹配的条件对年龄字段应用compose验证，比如firstName是'Bharat'，那么只有年龄必须验证(基于函数)。

4 在表单中应用基于匹配条件的合成验证，比如如果firstName是'Bharat'，那么只有cityName必须被验证(基于字符串数据类型)

5 Ap在countryName字段上编写验证。

6 基于服务器规则动态应用组合验证。

**ComposeConfig**

**conditionalExpression**

Type : Function | string

如果条件在conditionalExpression函数中匹配，则应该应用Compose验证。在检查conditionalExpression时，验证框架将传递两个参数。这两个参数是当前的FormGroup值和根FormGroup值。您可以将条件应用于相应的对象值。

如果需要动态验证意味着它在客户端代码中不是固定的，那么它将根据一些标准进行更改。在这个场景中，你可以根据来自web服务器的字符串格式的表达式值来绑定表达式。conditionalExpression将与客户端函数一样工作。

果你不返回表达式或固定布尔值true或false，这将不起作用;like: conditionalExpression: (x) =x.toggle

将conditionalExpression与Function对象绑定。

RxwebValidators.compose({validators:[RxwebValidators.digit()] ,

conditionalExpression:(x,y) => x.firstName == "Bharat" });

将conditionalExpression与字符串对象绑定。

RxwebValidators.compose({validators:[RxwebValidators.alpha()] ,

conditionalExpression:'x => x.firstName =="Bharat"' });

**message**

Type : string

覆盖全局配置消息并在各自的FormControl上设置自定义消息。

RxwebValidators.compose({validators:[RxwebValidators.alpha()] ,

message:'You can only enter alphabets.' });

**messageKey**

Type : string

编写验证的messageKey选项用于设置基于密钥的验证消息。

RxwebValidators.compose({validators:[RxwebValidators.required(),

RxwebValidators.alpha()] ,messageKey:'composeMessageKey' });

**validators**

Type : ValidatorFn[]

它是一个rxwebValidators数组。验证器是根据您想要应用的验证的相关需求设置的。这里必须指定要使用的验证器的名称。

RxwebValidators.compose({validators:[RxwebValidators.required(), RxwebValidators.alpha()] });

ngOnInit() {

this.userFormGroup = this.formBuilder.group({

firstName:['', RxwebValidators.compose({

validators:[RxwebValidators.required(), RxwebValidators.alpha()] })],

lastName:['', RxwebValidators.compose({

validators:[RxwebValidators.required(), RxwebValidators.alpha()] ,

messageKey:'composeMessageKey' })],

age:['', RxwebValidators.compose({

validators:[RxwebValidators.digit()]

,conditionalExpression:(x,y) => x.firstName == "Bharat" })],

city:['', RxwebValidators.compose({

validators:[RxwebValidators.alpha()]

,conditionalExpression:'x => x.firstName =="Bharat"' })],

countryName:['', RxwebValidators.compose(

{validators:[RxwebValidators.alpha()] ,message:'只能输入字母' })],

});

}

**export class User {**

@compose({validators:[RxwebValidators.required(), RxwebValidators.alpha()] })

firstName: string;

//如果你想应用“function”类型的条件表达式

@compose({validators:[RxwebValidators.digit()] ,

conditionalExpression:(x,y) => x.firstName == "Bharat" })

age: number;

//如果你想应用类型为string的条件表达式

@compose({validators:[RxwebValidators.alpha()] ,

conditionalExpression:'x => x.firstName =="Bharat"' })

city: string;

@compose({validators:[RxwebValidators.alpha()] ,

message:'You can only enter alphabets.' })

countryName: string;

@compose({validators:[uniqueCountryCode()]})

countryCode: string;

}

function uniqueCountryCode() {

return (control: AbstractControl) => {

if (control.value != "IND")

return null;

else

return {

uniqueCountryCode: {message:"Country Code Should be Unique"}

}

}

}

**Custom - mask**

掩码验证验证器将检查输入的值是否符合指定的模式，并允许用户根据固定的格式输入字符。

假设您想要创建一个包含诸如firstName、phoneNumber、postalCode、zipcode、contactNumber和alternateContactNumber等字段的用户表单。在这里，根据需求可能会出现这些场景。

1 检查验证的phoneNumber。

2 应用掩码验证基于匹配条件的形式，如firstName是' Bharat '，那么postalCode的值应该是正确的格式。

3 contactNumber字段上的自定义消息。

4 使用messageKey属性显示alternateContactNumber字段上的自定义消息。

5基于服务器规则动态应用掩码验证。

**BaseConfig**

下面的选项不是RxwebValidators.mask()验证器中必须使用的。如果需要，那么使用下面的选项。

**conditionalExpression**

Type : Function | string

如果条件在conditionalExpression函数中匹配，则应应用掩码验证。在检查conditionalExpression时，验证框架将传递两个参数。这两个参数是当前的FormGroup值和根FormGroup值。您可以将条件应用于相应的对象值。

如果需要动态验证意味着它在客户端代码中不是固定的，那么它将根据一些标准进行更改。在这个场景中，你可以根据来自web服务器的字符串格式的表达式值来绑定表达式。conditionalExpression将与客户端函数一样工作。

如果你不返回表达式或固定布尔值true或false，这将不起作用;like: conditionalExpression: (x) =x.toggle

将conditionalExpression与Function对象绑定。

将conditionalExpression与字符串对象绑定。

**valueWithMask**

Type : boolean

向mac验证器添加掩码

**minLength**

Type : number

在设置minLength属性时使用数字选项

**message**

Type : string

覆盖全局配置消息并在各自的FormControl上设置自定义消息。

**messageKey**

Type : string

当用户希望在其某些字段上显示不同的自定义验证消息时，可以使用BaseConfig的messageKey属性。用户可以通过在ReactiveFormConfig中定义一个全局自定义messageKey，并在验证的message属性中设置它。

**Dynamic mask Example**

这个动态掩码示例是基于传递给formBuilderConfiguration中的json执行的，formBuilderConfiguration位于响应式表单验证器的RxFormBuilder之下。带有函数的conditionalExpression不会应用于动态掩码示例。这个例子将满足我们最后一点的要求。

文档错误，没有案例

Custom - pattern

模式验证验证器将允许用户输入匹配预定义的模式值参数的输入。

假设您希望创建一个User表单，其中包含Username、Zipcode、Age等字段，并且希望用户输入包含预定义值的输入。在这里，根据需求可能会出现这些场景。

允许

1 输入只包含字母的用户名。

2 Ap基于表单中匹配条件的厚度模式验证，如Username是John，则只有模式验证必须应用于Age值(即:，年龄字段只能是数字)。

3 在Zipcode字段上添加自定义消息。

4 使用messageKey属性显示Pin Code字段上的自定义消息。

5 基于服务器规则动态应用模式验证。

userName:['', RxwebValidators.pattern({expression:{'onlyAlpha': /^[A-Za-z]+$/} })],

zipCode:['', RxwebValidators.pattern({expression:{'zipCode':/^[0-9]{5}(?:-[0-9]{4})?$/ } ,

message:'Zip code should match 12345 or 12345-6789' })],

contactNumber:['', RxwebValidators.pattern({expression:{'onlyDigit': /^[0-9]\*$/} ,

conditionalExpression:(x,y) => x.userName == "Bharat" })],

age:['', RxwebValidators.pattern({expression:{'onlyDigit': /^[0-9]\*$/} ,

conditionalExpression:'x => x.userName=="Bharat"' })],

pinCode:['', RxwebValidators.pattern({expression:{'pinCode':/^[1-9][0-9]{5}$/ } ,

messageKey:'patternMessageKey' })],

**动态模式的例子**

这个动态模式的例子是基于传递到formBuilderConfiguration中的json执行的，formBuilderConfiguration属于响应式表单验证器的RxFormBuilder。带有函数的conditionalExpression不会应用于动态模式示例。这个例子将满足我们最后一点的要求。

let formBuilderConfiguration = new FormBuilderConfiguration();

this.http.get('assets/dynamic.json?v='+environment.appVersion).subscribe(dynamic => {

formBuilderConfiguration.dynamicValidation = JSON.parse(JSON.stringify(dynamic));

var user = { userName:'', zipCode:'', contactNumber:'', age:'', pinCode:'', }

this.userFormGroup = this.formBuilder.group(user,formBuilderConfiguration);

})

export class User {

@pattern({expression:{'onlyAlpha': /^[A-Za-z]+$/} })

userName: string;

@pattern({expression:{'zipCode':/^[0-9]{5}(?:-[0-9]{4})?$/ }

,message:'Zip code should match 12345 or 12345-6789' })

zipCode: string;

//如果你想应用“function”类型的条件表达式

@pattern({expression:{'onlyDigit': /^[0-9]\*$/}

,conditionalExpression:(x,y) => x.userName == "Bharat" })

contactNumber: number;

/如果你想应用类型为string的条件表达式

@pattern({expression:{'onlyDigit': /^[0-9]\*$/}

,conditionalExpression:'x => x.userName=="Bharat"' })

age: string;

@pattern({expression:{'pinCode':/^[1-9][0-9]{5}$/ }

,messageKey:'patternMessageKey' })

pinCode: string;

}

Date and time - Date

<https://docs.rxweb.io/form-validations/date/validators>

**date**

日期验证验证器允许用户输入正确的日期格式的输入。

假设您想要创建一个UserInfo表单，其中包含诸如BirthDate、AdmissionDate、EnrollmentDate和AllocationDate等字段，并且您希望用户只输入日期输入。在这里，根据需求可能会出现这些场景。

1 允许用户只输入日期输入的生日日期。

2 根据表单中匹配的条件应用日期验证，比如如果BirthDate是16/04/1997，那么AdmissionDate值应该是日期格式的(带有函数的条件验证)。

3 根据表单中匹配的条件应用日期验证，比如如果BirthDate是16/04/1997，那么EnrollmentDate值应该是日期格式的(带有字符串的条件验证)。

4 在AllocationDate字段上应用自定义消息。

5 使用messageKey属性在确认日期字段上显示自定义消息。

6 基于服务器规则动态应用日期验证。

要在应用程序中全局配置日期格式，请参考ReactiveFormConfig

**BaseConfig**

下面的选项不是RxwebValidators.date()验证器中必须使用的。如果需要，那么使用下面的选项。

conditionalExpression : 如果条件在conditionalExpression函数中匹配，则应该应用日期验证。在检查conditionalExpression时，验证框架将传递两个参数。这两个参数是当前的FormGroup值和根FormGroup值。您可以将条件应用于相应的对象值。如果需要动态验证意味着它在客户端代码中不是固定的，那么它将根据一些标准进行更改。在这个场景中，你可以根据来自web服务器的字符串格式的表达式值来绑定表达式。conditionalExpression将与客户端函数一样工作。

allowISO: 根据国际标准化组织，允许用标准方式表示数字日历日期

message : 覆盖全局配置消息并在各自的FormControl上设置自定义错误消息

messageKey : 当用户希望在其某些字段上显示不同的自定义验证消息时，可以使用BaseConfig的messageKey属性。用户可以通过在ReactiveFormConfig中定义一个全局自定义messageKey，并在验证的message属性中设置它。

**allowISODate**

Type : boolean

根据国际标准化组织，允许用标准方式表示数字日历日期。例如:1981-04-05，即YYYY-MM-DD格式

RxwebValidators.date({allowISODate:true });

**conditionalExpression**

Type : Function | string

如果条件在conditionalExpression函数中匹配，则应该应用日期验证。在检查conditionalExpression时，验证框架将传递两个参数。这两个参数是当前的FormGroup值和根FormGroup值。您可以将条件应用于相应的对象值。

如果需要动态验证意味着它在客户端代码中不是固定的，那么它将根据一些标准进行更改。在这个场景中，你可以根据来自web服务器的字符串格式的表达式值来绑定表达式。conditionalExpression将与客户端函数一样工作。

如果你不返回表达式或固定布尔值true或false，这将不起作用;like: conditionalExpression: (x) =x.toggle

将conditionalExpression与Function对象绑定。

RxwebValidators.date({conditionalExpression:(x,y) => x.birthDate == "16/04/1997" });

将conditionalExpression与字符串对象绑定。

RxwebValidators.date({conditionalExpression:'x => x.birthDate =="16/04/1997"' });

**message**

Type : string

覆盖全局配置消息并在各自的FormControl上设置自定义消息。

RxwebValidators.date({message:'{{0}} is not a valid date' });

**messageKey**

Type : string

当用户希望在其某些字段上显示不同的自定义验证消息时，可以使用BaseConfig的messageKey属性。用户可以通过在ReactiveFormConfig中定义一个全局自定义messageKey，并在验证的message属性中设置它。

**完成日期的例子**

这个完整日期示例包括所有的BaseConfig属性，它将满足场景1、2、3、4和5的要求。

ngOnInit() {

this.userInfoFormGroup = this.formBuilder.group({

birthDate: ['', RxwebValidators.date()],

registrationDate: ['', RxwebValidators.date({ allowISODate: true })],

admissionDate: ['', RxwebValidators.date({ conditionalExpression: (x, y) => x.birthDate == '16/04/1997' })],

enrollmentDate: ['', RxwebValidators.date({ conditionalExpression: 'x => x.birthDate =="16/04/1997"' })],

allocationDate: ['', RxwebValidators.date({ message: '{{0}} is not a valid date' })],

confirmationDate: ['', RxwebValidators.date({ messageKey: 'dateMessageKey' })]

});

}

动态日期的例子

这个动态日期的例子是基于传递到formBuilderConfiguration中的json执行的，formBuilderConfiguration属于响应式表单验证器的RxFormBuilder。带有函数的conditionalExpression将不会应用于动态日期示例。这个例子将满足我们最后一点的要求。

ngOnInit() {

let formBuilderConfiguration = new FormBuilderConfiguration();

this.http.get('assets/examples/reactive-form-validators/validators/date/dynamic/dynamic.json?v='

environment.appVersion).subscribe(dynamic => {

formBuilderConfiguration.dynamicValidation = JSON.parse(JSON.stringify(dynamic));

var userInfo = {

birthDate: '',

registrationDate: '',

admissionDate: '',

enrollmentDate: '',

allocationDate: '',

confirmationDate: ''

};

this.userInfoFormGroup = this.formBuilder.group(userInfo, formBuilderConfiguration);

});

}

**Decorator**

export class UserInfo {

@date()

birthDate: string;

@date({allowISODate:true })

registrationDate: string;

//If you want to apply conditional expression of type 'function'

@date({conditionalExpression:(x,y) => x.birthDate == "16/04/1997" })

admissionDate: string;

//If you want to apply conditional expression of type 'string'

@date({conditionalExpression:'x => x.birthDate =="16/04/1997"' })

enrollmentDate: string;

@date({message:'{{0}} is not a valid date' })

allocationDate: string;

@date({messageKey:'dateMessageKey' })

confirmationDate: string;

}

{

"birthDate": {

"date": true

},

"registrationDate": {

"date": {

"allowISODate": "true"

}

},

"enrollmentDate": {

"date": {

"conditionalExpression": "x => x.birthDate ==\"16/04/1997\""

}

},

"allocationDate": {

"date": {

"message": "{{0}} is not a valid date"

}

},

"confirmationDate": {

"date": {

"messageKey": "dateMessageKey"

}

}

}

ngOnInit() {

let userInfo = new UserInfo();

let formBuilderConfiguration = new FormBuilderConfiguration();

this.http.get('assets/dynamic.json?v=' + environment.appVersion).subscribe(dynamic => {

formBuilderConfiguration.dynamicValidation = JSON.parse(JSON.stringify(dynamic));

this.userInfoFormGroup = this.formBuilder.formGroup(userInfo, formBuilderConfiguration);

});

}

**Date and time - maxDate**

maxDate验证验证器将允许用户输入小于maxDate值参数的日期。

1 基于fieldName在lastRegistrationDate中应用MaxDate验证，MaxDate必须被设置。

2 添加字段registrationDate没有任何条件表达式。

3 根据表单中匹配的条件应用MaxDate验证，比如如果userName是' John '，那么birthDate值应该是有效日期，不超过最大日期。

4 在注册日期字段上添加自定义消息。

5 添加要在属性中限制number的值。最长日期是“2018年7月30日”。

6 使用messageKey属性显示辞职日期字段上的自定义消息。

7 基于服务器规则动态应用MaxDate验证。

要在应用程序中全局配置日期格式，请参考ReactiveFormConfig

**conditionalExpression**

Type : Function | string

如果条件在conditionalExpression函数中匹配，则应应用最大日期验证。在检查conditionalExpression时，验证框架将传递两个参数。这两个参数是当前的FormGroup值和根FormGroup值。您可以将条件应用于相应的对象值。

如果需要动态验证意味着它在客户端代码中不是固定的，那么它将根据一些标准进行更改。在这个场景中，你可以根据来自web服务器的字符串格式的表达式值来绑定表达式。conditionalExpression将与客户端函数一样工作。

如果你不返回表达式或固定布尔值true或false，这将不起作用;like: conditionalExpression: (x) =x.toggle

将conditionalExpression与Function对象绑定。

RxwebValidators.maxDate({value:'07/30/2018' ,conditionalExpression:(x,y) => x.userName == "Bharat" });

将conditionalExpression与字符串对象绑定。

RxwebValidators.maxDate({value:'07/30/2018' ,conditionalExpression:'x => x.userName == "Bharat"' });

**fieldName**

Type : string

MaxDate验证应该基于fieldName应用，MaxDate必须在该fieldName上设置。

RxwebValidators.maxDate({fieldName:'enrollmentDate' });

**message**

Type : string

覆盖全局配置消息并在各自的FormControl上设置自定义错误消息

RxwebValidators.maxDate({value:'07/30/2018' ,message:'{{0}} exceeds the Maximum Date Limit' });

**messageKey**

Type : string

当用户希望在其某些字段上显示不同的自定义验证消息时，可以使用BaseConfig的messageKey属性。用户可以通过在ReactiveFormConfig中定义一个全局自定义messageKey，并在验证的message属性中设置它。

RxwebValidators.maxDate({value:'07/30/2018' ,messageKey:'maxDateMessageKey' });

**operator**

Type : string

操作符用于根据小于或小于等于特定值验证日期，默认验证基于小于等于。

RxwebValidators.maxDate({value:'07/30/2018' ,operator:'<' });

**value**

Type : Date | string

输入要在属性中限制的日期值。

RxwebValidators.maxDate({value:'07/30/2018' });

**完整的MaxDate示例**

这个Complete MaxDate示例包括所有的DateConfig属性，它将满足场景1、2、3、4、5和6的要求。

ngOnInit() {

this.userFormGroup = this.formBuilder.group({

userName: [''],

allocationDate: ['', RxwebValidators.maxDate({ value: '07/30/2018' })],

birthDate: ['', RxwebValidators.maxDate({

value: '07/30/2018',

conditionalExpression: (x, y) => x.userName == 'Bharat'

})],

admissionDate: ['', RxwebValidators.maxDate({

value: '07/30/2018',

conditionalExpression: 'x => x.userName == "Bharat"'

})],

registrationDate: ['', RxwebValidators.maxDate({

value: '07/30/2018',

message: '{{0}} exceeds the Maximum Date Limit'

})],

enrollmentDate: [''],

lastRegistrationDate: ['', RxwebValidators.maxDate({ fieldName: 'enrollmentDate' })],

confirmationDate: ['', RxwebValidators.maxDate({ value: '07/30/2018', operator: '<' })],

resignationDate: ['', RxwebValidators.maxDate({ value: '07/30/2018', messageKey: 'maxDateMessageKey' })]

});

}

**动态MaxDate例子**

这个Dynamic maxDate的例子是基于传递到formBuilderConfiguration中的json执行的，formBuilderConfiguration属于响应式表单验证器的RxFormBuilder。带有函数的conditionalExpression不会应用于动态maxDate示例。这个例子将满足我们最后一点的要求。

ngOnInit() {

let formBuilderConfiguration = new FormBuilderConfiguration();

this.http.get('assets/examples/reactive-form-validators/validators/maxDate/dynamic/dynamic.json?v=' + environment.appVersion).subscribe(dynamic => {

formBuilderConfiguration.dynamicValidation = JSON.parse(JSON.stringify(dynamic));

var user = {

userName: '',

allocationDate: '',

birthDate: '',

admissionDate: '',

registrationDate: '',

enrollmentDate: '',

lastRegistrationDate: '',

confirmationDate: '',

resignationDate: ''

};

this.userFormGroup = this.formBuilder.group(user, formBuilderConfiguration);

});

}

**Decorator**

export class User {

@prop()

userName: string;

@maxDate({ value: '07/30/2018' })

allocationDate: string;

//如果你想应用“function”类型的条件表达式

@maxDate({ value: '07/30/2018', conditionalExpression: (x, y) => x.userName == 'Bharat' })

birthDate: string;

//如果你想应用类型为string的条件表达式

@maxDate({ value: '07/30/2018', conditionalExpression: 'x => x.userName == "Bharat"' })

admissionDate: string;

@maxDate({ value: '07/30/2018', message: '{{0}} exceeds the Maximum Date Limit' })

registrationDate: string;

@prop()

enrollmentDate: string;

@maxDate({ fieldName: 'enrollmentDate' })

lastRegistrationDate: string;

@maxDate({ value: '07/30/2018', operator: '<' })

confirmationDate: string;

@maxDate({ value: '07/30/2018', messageKey: 'maxDateMessageKey' })

resignationDate: string;

}

dynamic.json

{

"userName": {},

"allocationDate": {

"maxDate": {

"value": "07/30/2018"

}

},

"admissionDate": {

"maxDate": {

"value": "07/30/2018",

"conditionalExpression": "x => x.userName == \"Bharat\""

}

},

"registrationDate": {

"maxDate": {

"value": "07/30/2018",

"message": "{{0}} exceeds the Maximum Date Limit"

}

},

"enrollmentDate": {},

"lastRegistrationDate": {

"maxDate": {

"fieldName": "enrollmentDate"

}

},

"confirmationDate": {

"maxDate": {

"value": "07/30/2018",

"operator": "<"

}

},

"resignationDate": {

"maxDate": {

"value": "07/30/2018",

"messageKey": "maxDateMessageKey"

}

}

}

ngOnInit() {

let user = new User();

let formBuilderConfiguration = new FormBuilderConfiguration();

this.http.get('assets/dynamic.json?v=' + environment.appVersion).subscribe(dynamic => {

formBuilderConfiguration.dynamicValidation = JSON.parse(JSON.stringify(dynamic));

this.userFormGroup = this.formBuilder.formGroup(user, formBuilderConfiguration);

});

}

**allOfAsync**

https://docs.rxweb.io/async-validators/allOfAsync/validators  
allOfAsync验证验证器将检查用户是否已输入给定字段的所有值或非异步。

**什么时候使用**假设您想要创建一个雇员表单，其中包含部门、projectDomains、资格、技能和爱好等字段。您希望用户输入特定字段的所有值。在这里，该字段以数组的形式获取，并根据此选择通过异步提供matchValues应用于属性。在这里，根据需求可能会出现这些场景。

在projectDomains字段上应用allOfAsync验证，您希望用户选择matchValues中提到的所有项目域。  
在兴趣爱好字段上显示自定义验证消息。  
使用messageKey属性显示语言字段上的自定义消息。  
基于服务器规则动态应用allOfAsync验证。

让我们看看allOfAsync验证器是如何满足需求的。

**基本allOfAsync验证**通过Angular的FormBuilder服务，我们在组件中创建了FormGroup。  
这里我们介绍了添加表单操作。

@Component({

selector: 'app-allOfAsync-add-validator',

templateUrl: './all-of-async-add.component.html'

})

export class AllOfAsyncAddValidatorComponent implements OnInit {

employeeInfoFormGroup: FormGroup

projectDomains: string[] = [];

projectDomainsArray: string[] = ["ECommerce", "Banking", "Educational", "Gaming"];

constructor(

private formBuilder: FormBuilder, private http: HttpClient) { }

ngOnInit() {

this.employeeInfoFormGroup = this.formBuilder.group({

department: [''],

projectDomains: ['',[], RxwebValidators.allOfAsync({validatorConfig:of({ matchValues: ["ECommerce", "Banking"] })})],

});

}

addProjectDomain(element: any, index: number) {

var indexOf = this.projectDomains.indexOf(element.value);

element.checked ? this.projectDomains.push(element.value) : this.projectDomains.splice(indexOf, 1);

this.employeeInfoFormGroup.controls.projectDomains.setValue(this.projectDomains);

}

}

**ValidatorConfig**

类型:Observable  
Validator配置用于根据服务器规则使用关键字的动态解析属性的值，如下所示  
下面的选项在RxwebValidators.allOfAsync(({validatorConfig:of({<property\_name>:<value>})}))验证器中不是强制使用的。如果需要，那么使用下面的选项。

Option Description

matchValues： matchValues是一个数组，用于验证值的匹配。根据它，数组中的所有值都应该匹配

message： 覆盖全局配置消息并在各自的FormControl上设置自定义错误消息

messageKey： 当用户希望在其某些字段上显示不同的自定义验证消息时，可以使用BaseConfig的messageKey属性。用户可以通过在ReactiveFormConfig中定义一个全局自定义messageKey，并在验证的message属性中设置它。

**matchValues**

Type : any[]

RxwebValidators.allOfAsync({validatorConfig:of({ matchValues: ["ECommerce", "Banking"] })

**message**

Type : string

RxwebValidators.allOfAsync(

{validatorConfig:

of({matchValues:["Drawing", "Singing","Dancing","Travelling"],message: "请选择所有的爱好"}

)

**messageKey**

Type : string

RxwebValidators.allOfAsync(

{validatorConfig:of({ matchValues: ["English", "Spanish", "French"], messageKey: 'allOfAsyncMessageKey' })

**完整的allOfAsync示例**这个完整的allOfAsync示例包含了所有的ArrayConfig属性，它将满足场景1、2、3和4的要求。

ngOnInit() {

this.employeeInfoFormGroup = this.formBuilder.group({

department: [''],

projectDomains: ['', [], RxwebValidators.allOfAsync({ validatorConfig: of({ matchValues: ['ECommerce', 'Banking', 'Educational', 'Gaming'] }) })],

qualifications: ['', [], RxwebValidators.allOfAsync({

validatorConfig: of({

matchValues: ['Secondary', 'Senior Secondary', 'B.Tech.', 'M.Tech.', 'B.C.A.', 'M.C.A.'],

conditionalExpression: (x, y) => x.department == 'DotNet'

})

})],

skills: ['', [], RxwebValidators.allOfAsync({

validatorConfig: of({

matchValues: ['MVC', 'AngularJS', 'Angular 5', 'C#', 'Web Api', 'SQL Server'],

conditionalExpression: 'x => x.department ==\'DotNet\''

})

})],

hobbies: ['', [], RxwebValidators.allOfAsync({

validatorConfig: of({

matchValues: ['Drawing', 'Singing', 'Dancing', 'Travelling', 'Sports'],

message: 'Please select all hobbies'

})

})],

languages: ['', [], RxwebValidators.allOfAsync({

validatorConfig: of({

matchValues: ['English', 'Spanish', 'French'],

messageKey: 'allOfAsyncMessageKey'

})

})]

});

}

**choiceAsync**

<https://docs.rxweb.io/async-validators/choiceAsync/validators>

choiceAsync验证验证器将检查用户输入的值是否与属性中定义的范围异步匹配

**什么时候使用**

假设您想要创建一个雇员表单，其中包含部门、projectDomains、资格、技能和爱好等字段。您希望用户为特定的choiceAsync选择特定范围的值。这里以数组的形式获取字段，并根据此情况，通过异步应用minlength和maxLength将choiceAsync应用于属性。在这里，根据需求可能会出现这些场景。

projectDomains字段，您希望用户在其中输入最少三个域。  
您希望用户最多输入4个条件的条件字段。  
在爱好字段中添加自定义消息。  
使用messageKey属性显示certification字段上的自定义消息。  
基于服务器规则动态应用choiceAsync验证。

languages: ['', [], RxwebValidators.choiceAsync({

validatorConfig: of({

minLength: 2,

conditionalExpression: (x, y) => x.department == 'DotNet'

})

})],

skills: ['', [], RxwebValidators.choiceAsync({

validatorConfig: of({

minLength: 2,

conditionalExpression: 'x => x.department ==\'DotNet\''

})

})],

hobbies: ['', [], RxwebValidators.choiceAsync({

validatorConfig: of({

maxLength: 4,

message: 'Please select upto 4 hobby'

})

})],

certifications: ['', [], RxwebValidators.choiceAsync({

validatorConfig: of({

minLength: 2,

messageKey: 'choiceMessageKey'

})

})],

noneOfAsync  
noneOfAsync验证验证器将检查是否没有从给定输入中异步选择任何值。

<https://docs.rxweb.io/async-validators/noneOfAsync/validators>

**什么时候使用**让我们假设您正在创建一个雇员表单，您希望雇员在其中不输入任何包含部门、爱好和技能等字段的特定值。在这里，该字段以数组的形式获取，并通过异步应用matchvalues将选择应用于属性。在这里，根据需求可能会出现这些场景。

您希望用户根据匹配值在其中输入技能的技能字段。  
应用noneOfAsync验证基于匹配条件的表单，如如果部门是' dotnet '，那么技能值应该基于匹配值。  
爱好上的自定义消息字段。  
使用messageKey属性显示语言字段上的自定义消息。  
基于服务器规则动态应用noneOfAsync验证。

this.employeeInfoFormGroup = this.formBuilder.group({

department: [''],

projectDomains: ['', [], RxwebValidators.noneOfAsync({ validatorConfig: of({ matchValues: ['ECommerce', 'Banking', 'Educational', 'Gaming'] }) })],

qualifications: ['', [], RxwebValidators.noneOfAsync({

validatorConfig: of({

matchValues: ['Secondary', 'Senior Secondary', 'B.Tech', 'M.Tech', 'B.C.A.', 'M.C.A.'],

conditionalExpression: (x, y) => x.department == 'DotNet'

})

})],

skills: ['', [], RxwebValidators.noneOfAsync({

validatorConfig: of({

matchValues: ['MVC', 'AngularJS', 'Angular 5', 'C#', 'Web Api', 'SQL Server'],

conditionalExpression: 'x => x.department ==\'DotNet\''

})

})],

hobbies: ['', [], RxwebValidators.noneOfAsync({

validatorConfig: of({

matchValues: ['Drawing', 'Singing', 'Dancing', 'Travelling', 'Sports'],

message: 'Please do not select any hobby'

})

})],

languages: ['', [], RxwebValidators.noneOfAsync({

validatorConfig: of({

matchValues: ['English', 'Spanish', 'French'],

messageKey: 'noneOfMessageKey'

})

})]

});

**oneOfAsync**  
oneOfAsync验证验证器将检查用户是否异步输入了任何一个给定的输入。

<https://docs.rxweb.io/async-validators/oneOfAsync/validators>

**什么时候使用**让我们假设您正在创建一个雇员表单，您希望雇员在其中输入某个特定值的任何一个值，该值包含部门、爱好和技能等字段。在这里，该字段以数组的形式获取，并根据此，oneOfAsync通过应用matchvalues异步应用于属性。在这里，根据需求可能会出现这些场景。

您希望用户根据匹配值在其中输入技能的技能字段。  
根据表单中的匹配条件应用oneOfAsync验证，比如如果部门是“dotnet”，那么技能值应该基于匹配值。  
爱好上的自定义消息字段。  
使用messageKey属性显示语言字段上的自定义消息。  
基于服务器规则动态应用oneOfAsync验证。

this.employeeInfoFormGroup = this.formBuilder.group({

department: [''],

projectDomains: ['',[],

RxwebValidators.oneOfAsync({ matchValues: ["ECommerce", "Banking", "Educational", "Gaming"] })})],

qualifications: ['',[],

RxwebValidators.oneOfAsync({ matchValues:

["Secondary", "Senior Secondary", "B.Tech", "M.Tech", "B.C.A.", "M.C.A."],

conditionalExpression: (x, y) => x.department == 'DotNet' })})

],

skills: ['',[],

RxwebValidators.oneOfAsync({ matchValues:

["MVC", "AngularJS", "Angular 5", "C#", "Web Api", "SQL Server"],

conditionalExpression: "x => x.department =='DotNet'" })})],

hobbies: ['',[],

RxwebValidators.oneOfAsync({ matchValues:

["Drawing", "Singing", "Dancing", "Travelling", "Sports"],

message: "Please select atleast 1 hobby" })})],

languages: ['',[],

RxwebValidators.oneOfAsync({ matchValues:

["English", "Spanish", "French"],

messageKey: 'oneOfAsyncMessageKey' })})]

});

**patternAsync**patternAsync验证验证器将允许用户异步输入匹配预定义的patternAsync值参数的输入。

<https://docs.rxweb.io/async-validators/patternAsync/validators>

允许输入只包含字母的用户名。  
根据表单中匹配的条件应用patternAsync验证，例如，如果Username是John，那么只有patternAsync验证必须应用到Age值(即:Age)。，年龄字段只能是数字)。  
在Zipcode字段上添加自定义消息。  
使用messageKey属性显示Pin Code字段上的自定义消息。  
基于服务器规则动态应用patternAsync验证。

this.userFormGroup = this.formBuilder.group({

userName: ['', [], RxwebValidators.patternAsync({ validatorConfig: of({ expression: { 'onlyAlpha': /^[A-Za-z]+$/ } }) })],

zipCode: ['', [], RxwebValidators.patternAsync({

validatorConfig: of({

expression: { 'zipCode': /^[0-9]{5}(?:-[0-9]{4})?$/ },

message: 'Zip code should match 12345 or 12345-6789'

})

})],

pinCode: ['', [], RxwebValidators.patternAsync({

validatorConfig: of({

expression: { 'pinCode': /^[1-9][0-9]{5}$/ },

messageKey: 'patternMessageKey'

})

})]

});

通过验证器、验证装饰器或模板表单在表单上应用验证的三种方法