NgModel

DIRECTIVE

Creates a FormControl instance from a domain model and binds it to a form control element.

See more...

See also

- RadioControlValueAccessor
- SelectControlValueAccessor

Exported from

- FormsModule

Selectors

- **[ngModel]**:*not([formControlName]):not([formControl])*

Properties

Property	Description
control: FormControl	***Read-Only***
@Input() name: string	Tracks the name bound to the directive. If a parent form exists, it use this name as a key to retrieve this control's value.
@Input('disabled') isDisabled: boolean	Tracks whether the control is disabled.
@Input('ngModel') model: any	Tracks the value bound to this directive.
@Input('ngModelOptions') options: { name?: string; standalone?: boolean; updateOn?: FormHooks; }	Tracks the configuration options for this ngModel instance. name: An alternative to setting the name attribute on the form contr element. See the example for using NgModel as a standalone contr standalone: When set to true, the ngModel will not register itself wit parent form, and acts as if it's not in the form. Defaults to false. If no

Property	Description
	parent form exists, this option has no effect. updateOn: Defines the event upon which the form control value and validity update. Defaults to 'change'. Possible values: 'change' | 'blur' | 'submit'.
@Output('ngModelChange') update: EventEmitter	Event emitter for producing the ngModelChange event after the view model updates.
path: string[]	***Read-Only*** Returns an array that represents the path from the top-level form to control. Each index is the string name of the control on that level.
formDirective: any	***Read-Only*** The top-level directive for this control if present, otherwise null.

Inherited from NgControl

- name: string | number | null
- valueAccessor: ControlValueAccessor | null

Inherited from AbstractControlDirective

- abstract control: AbstractControl | null
- value: any
- valid: boolean | null
- invalid: boolean | null
- pending: boolean | null
- disabled: boolean | null
- enabled: boolean | null
- errors: ValidationErrors | null
- pristine: boolean | null

- dirty: boolean | null
- touched: boolean | null
- status: string | null
- untouched: boolean | null
- statusChanges: Observable<any> | null
- valueChanges: Observable<any> | null
- path: string[] | null
- validator: ValidatorFn | null
- asyncValidator: AsyncValidatorFn | null

Template variable references

Identifier	Usage
ngModel	#myTemplateVar="ngModel"

Description

The FormControl instance tracks the value, user interaction, and validation status of the control and keeps the view synced with the model. If used within a parent form, the directive also registers itself with the form as a child control.

This directive is used by itself or as part of a larger form. Use the ngModel selector to activate it.

It accepts a domain model as an optional Input. If you have a one-way binding to ngModel with [] syntax, changing the domain model's value in the component class sets the value in the view. If you have a two-way binding with [()] syntax (also known as 'banana-in-a-box syntax'), the value in the UI always syncs back to the domain model in your class.

To inspect the properties of the associated FormControl (like the validity state), export the directive into a local template variable using ngModel as the key (ex: #myVar="ngModel"). You can then access the control using the directive's control property. However, the most commonly used properties (like valid and dirty) also exist on the control for direct access. See a full list of properties directly available in AbstractControlDirective.

Using ngModel on a standalone control

The following examples show a simple standalone control using ngModel:

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

```
@Component({
 selector: 'example-app',
 template: `
  <input [(ngModel)]="name" #ctrl="ngModel" required />

  <p>Value: {{ name }}</p>
  <p>Valid: {{ ctrl.valid }}</p>

  <button (click)="setValue()">Set value</button>
 `,
})
export class SimpleNgModelComp {
 name: string = '';

 setValue() {
  this.name = 'Nancy';
 }
}
```

When using the ngModel within <form> tags, you'll also need to supply a name attribute so that the control can be registered with the parent form under that name.

In the context of a parent form, it's often unnecessary to include one-way or two-way binding, as the parent form syncs the value for you. You access its properties by exporting it into a local template variable using ngForm such as (#f="ngForm"). Use the variable where needed on form submission.

If you do need to populate initial values into your form, using a one-way binding for ngModel tends to be sufficient as long as you use the exported form's value rather than the domain model's value on submit.

Using ngModel within a form

The following example shows controls using ngModel within a form:

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

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

```
@Component({
 selector: 'example-app',
 template: `
  <form #f="ngForm" (ngSubmit)="onSubmit(f)" novalidate>
   <input name="first" ngModel required #first="ngModel" />
   <input name="last" ngModel />
   <button>Submit</button>
  </form>

  <p>First name value: {{ first.value }}</p>
  <p>First name valid: {{ first.valid }}</p>
  <p>Form value: {{ f.value | json }}</p>
  <p>Form valid: {{ f.valid }}</p>
 `,
})
export class SimpleFormComp {
 onSubmit(f: NgForm) {
  console.log(f.value); // { first: '', last: '' }
  console.log(f.valid); // false
 }
}
```

Using a standalone ngModel within a group

The following example shows you how to use a standalone ngModel control within a form. This controls the display of the form, but doesn't contain form data.

```
content_copy<form>
 <input name="login" ngModel placeholder="Login">
 <input type="checkbox" ngModel [ngModelOptions]="{standalone: true}"> Show more options?
</form>
```

<!-- form value: {login: ''} -->

Setting the ngModel name attribute through options

The following example shows you an alternate way to set the name attribute. Here, an attribute identified as name is used within a custom form control component. To still be able to specify the NgModel's name, you must specify it using the ngModelOptions input instead.

content_copy<form>

<my-custom-form-control name="Nancy" ngModel [ngModelOptions]="{name: 'user'}">

</my-custom-form-control>

</form>

<!-- form value: {user: ''} -->

Methods

viewToModelUpdate()

code

Sets the new value for the view model and emits an ngModelChange event.

viewToModelUpdate(newValue: any): void

Parameters

newValue	any	The new value emitted by ngModelChange.

Returns

void

Inherited from NgControl

- abstract viewToModelUpdate(newValue: any): void

Inherited from AbstractControlDirective

- reset(value: any = undefined): void
- hasError(errorCode: string, path?: string | (string | number)[]): boolean
- getError(errorCode: string, path?: string | (string | number)[]): any

(This article is sourced from: https://angular.io/api/forms/NgModel and the content of the article is here in case of link breakage.)