ngModel & Two way Data binding in Angular

13 Comments / 5 minutes of reading / March 9, 2023

⟵ Event Binding Angular Tutorial ngmodelchange ⟶

In this article let us explore the two way data binding in Angular and how NgModel implements the two-way binding in Angular Forms. The `ngModel` is a built-in directive and is part of the FormsModule. The Two-way binding uses the syntax `[()]`

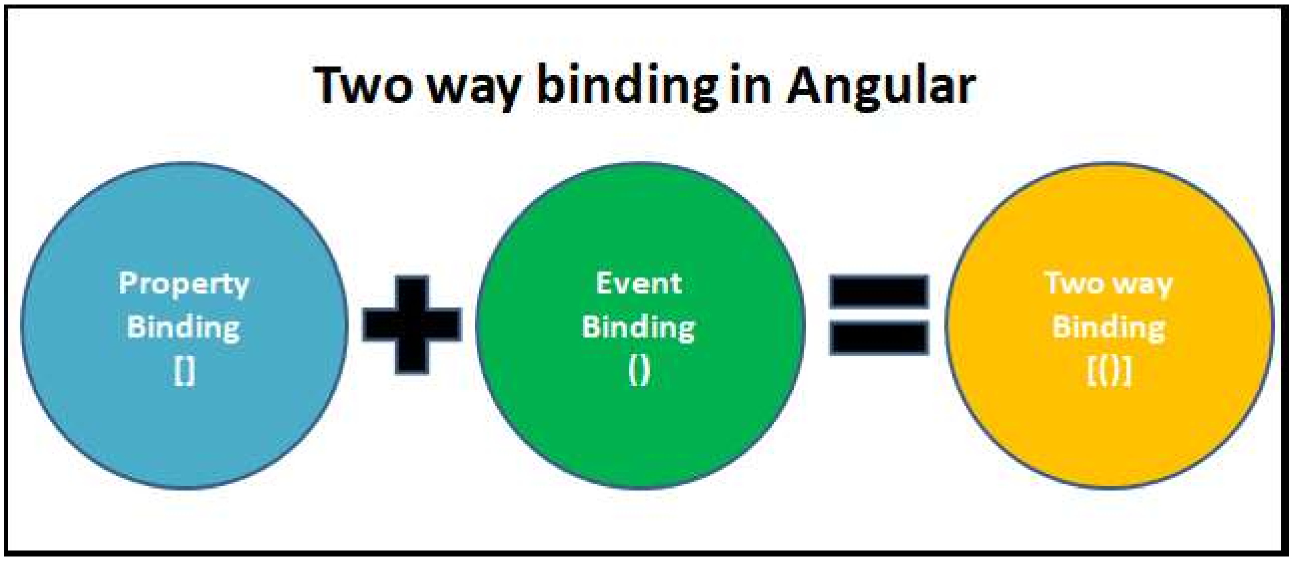
Table of Contents

What is Two way data binding

Two way data binding means that changes made to our model in the component are propagated to the view and that any changes made in the view are immediately updated in the underlying component data.

Two way data binding is useful in data entry forms. Whenever a user makes changes to a form field, we would like to update our model. Similarly, when we update the model with new data, we would like to update the view as well

The two way data binding is nothing but both property binding & event binding applied together. Property Binding is one way from component to view. The event binding is one way from view to component. If we combine both we will get the Two-way binding.



Two way using property & Event Binding

The following example shows how we can achieve two-way binding using the combination of property binding & event binding

Create a new Angular application

copy the following code to app.component.html

```

h2>Example 1</h2>
<input type="text" [value]="name" (input)="name=$event.target.value">
<p> You entered {{name}}</p>
<button (click)="clearName()">Clear</button>

```

Update the app.component.ts with the following code.

```

 name=""
 clearName() {
   this.name="";
 }


```

We bind the name property to the input element ([value]="name"). We also use the event binding (input)="name=$event.target.value" . It updates the name property whenever the input changes. The Angular interpolation updates the {{name}} , so we know the value of name property.

$event.target.value raises the error Property 'value' does not exist on type 'EventTarget' if fullTemplateTypeCheck is set to true under angularCompilerOptions in the tsconfig.json .

The error is due to the fact that the value property is not guaranteed to exist in the $event.target .

To solve this problem either you can use the `$any` typecast function (`$any($event.target).value`) to stop the type checking in the template or set `fullTemplateTypeCheck` to `false` in `tsconfig.json`.

Two-way binding syntax

The above example uses the event & property binding combination to achieve the two-way binding. But Angular does provide a way to achieve the two-way binding using the syntax `[()]`. Note that both square & parentheses are used here. This is now known as **Banana in a box** syntax. The square indicates the Property binding & parentheses indicates the event binding.

For Example

```

<someElement [(someProperty)]="value"></someElement>

```

The above syntax sets up both property & event binding. But to make use of it, the property must follow the following naming convention.

If we are binding to a settable property called `someProperty` of an element, then the element must have the corresponding change event named `somePropertyChange`

But most HTML elements have a `value` property. But do not have a `valueChange` event, instead, they usually have an `input` event. Hence they cannot be used in the above syntax

For Example, the following will not work as there is no `valueChange` event supported by the `input` element.

Hence we have a `ngModel` directive.

What is ngModel

The Angular uses the `ngModel` directive to achieve the two-way binding on HTML Form elements. It binds to a form element like `input`, `select`, `selectarea`. etc.

Internally It uses the `ngModel` in property, binding to bind to the `value` property and `ngModelChange` which binds to the input event.

How to use ngModel

The `ngModel` directive is not part of the Angular Core library. It is part of the `FormsModule` library. You need to import the `FormsModule` package into your Angular module.

In the template use the following syntax

```

<input type="text" name="value" [(ngModel)]="value">

```

The ngModel directive placed inside the square & parentheses as shown above. This is assigned to the Template Expression. Template Expression is the property in the component class

ngModel Example

Import FormsModule

Open the app.module.ts and make the following changes

```

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

```

Template

```

<h2>Example 2</h2>
<input type="text" name="value" [(ngModel)]="value">
<p> You entered {{value}}</p>
<button (click)="clearValue()">Clear</button>

```

Component

```

value="";
clearValue() {
  this.value="";
}


```

The ngModel data property sets the element’s value property and the ngModelChange event property listens for changes to the element’s value.

Run the project and see that as you modify the name, the component class model is automatically updated.

Custom Two-way binding

As we mentioned earlier the [()] to work, we need to have a property with the change event as <nameofProperty>Change.

We do not have any HTML Elements which follows the above naming conventions, but we can create a custom component

create new component and name it as counter.component.ts . copy the following code.

```

import { Component, Input, Output, EventEmitter } from '@angular/core';
@Component({
 selector: 'counter',
 template: `
     <div>
       <p>
         Count: {{ count }}
         <button (click)="increment()">Increment</button>
       </p>
     </div>
   `
})
export class CounterComponent {

  @Input() count: number = 0;
  @Output() countChange: EventEmitter<number> = new EventEmitter<number>();

  increment() {
   this.count++;
   this.countChange.emit(this.count);
 }
}


```

The component has two properties one is input property count decorated with `@Input()`. The other in is an event (or output property), which we decorate with `@Output()`. We name the input property as `count`. Hence the output property becomes `countChange`

Now we can use this component and create two-way binding to the `count` property using the syntax `[(count)]`.

```

<h2>Example 3</h2>
<counter [(count)]="count"></counter>
<p> Current Count {{count}}</p>
<button (click)="clearCount()">Clear</button>

```