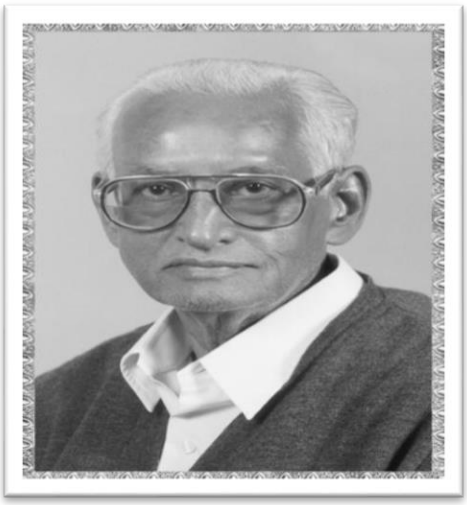
SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)



Lt. Shree Chimanbhai Shukla

MSCIT SEM-3 ANGULARJS

Shree H.N.Shukla college2
vaishali nagar
Near Amrapali Under Bridge,
Raiya road
Rajkot
Ph No:-0281 2440478

Shree H.N.Shukla college3
vaishali nagar
Near Amrapali Under Bridge,
Raiya road
Rajkot
Ph No:-0281 2440478

Unit : 2

DATA BINDING IN ANGULAR

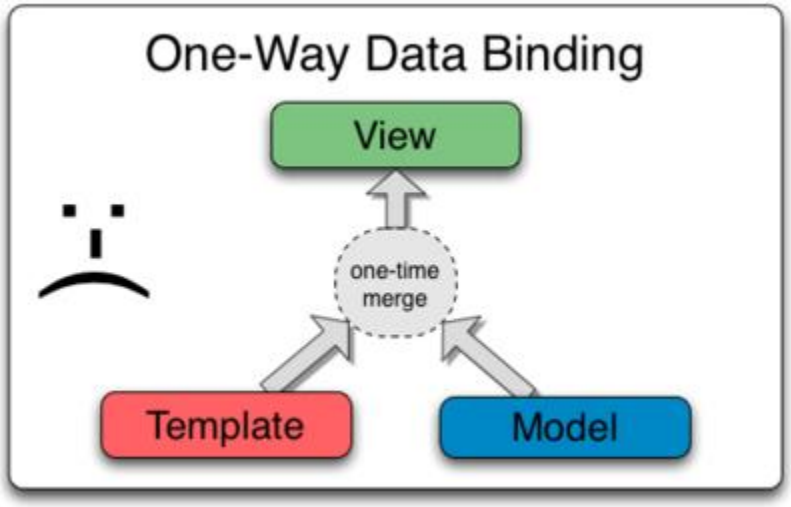
Data Binding?

- ✓ Data-binding in AngularJS apps is the automatic synchronization of data between the model and view components.
- ✓ The way that AngularJS implements data-binding lets you treat the model as the single-source-of-truth in your application.
- ✓ The view is a projection of the model at all times.
- ✓ When the model changes, the view reflects the change, and vice versa.

AngularJS provides two types of Data Binding:

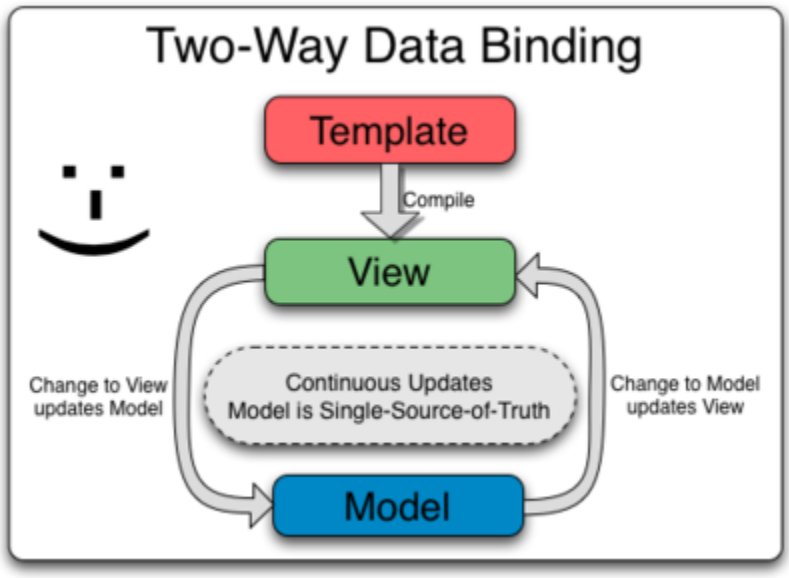
- One-way data binding.
- Two-way data binding.

Data Binding in Classical Template Systems



Most templating systems bind data in only one direction: they merge template and model components together into a view. After the merge occurs, changes to the model or related sections of the view are NOT automatically reflected in the view. Worse, any changes that the user makes to the view are not reflected in the model. This means that the developer has to write code that constantly syncs the view with the model and the model with the view.

Data Binding in AngularJS Templates



AngularJS templates work differently. First the template (which is the uncompiled HTML along with any additional markup or directives) is compiled on the browser. The compilation step produces a live view. Any changes to the view are immediately reflected in the model, and any changes in the model are propagated to the view. The model is the single-source-of-truth for the application state, greatly simplifying the programming model for the developer. You can think of the view as simply an instant projection of your model.

Because the view is just a projection of the model, the controller is completely separated from the view and unaware of it. This makes testing a snap because it is easy to test your controller in isolation without the view and the related DOM/browser dependency.

> reflected in the view. Worse, any changes that the user makes to the view are not reflected in the model. This

Property binding

Property binding in Angular helps you set values for properties of HTML elements or directives. Use property binding to do things such as toggle button features, set paths programmatically, and share values between components.

Understanding the flow of data

Property binding moves a value in one direction, from a component's property into a target element property.

For more information on listening for events, see Event binding.

To read a target element property or call one of its methods, see the API reference for ViewChild and ContentChild.

Binding to a property

To bind to an element's property, enclose it in square brackets, `[]`, which identifies the property as a target property.

A target property is the DOM property to which you want to assign a value.

To assign a value to a target property for the image element's `src` property, type the following code:

src/app/app.component.html

```
content_copy<img alt="item" [src]="itemImageUrl">
```

In most cases, the target name is the name of a property, even when it appears to be the name of an attribute.

In this example, `src` is the name of the <img> element property.

The brackets, `[]`, cause Angular to evaluate the right-hand side of the assignment as a dynamic expression.

Without the brackets, Angular treats the right-hand side as a string literal and sets the property to that static value.

To assign a string to a property, type the following code:

src/app.component.html

```
content_copy<app-item-detail childItem="parentItem"></app-item-
detail>
```

Omitting the brackets renders the string `parentItem`, not the value of `parentItem`.

Setting an element property to a component property value

To bind the `src` property of an <img> element to a component's property, place `src` in square brackets followed by an equal sign and then the property.

Using the property `itemImageUrl`, type the following code:

src/app/app.component.html

```
content_copy<img alt="item" [src]="itemImageUrl">
```

Declare the `itemImageUrl` property in the class, in this case `AppComponent`.

src/app/app.component.ts

```
content_copyitemImageUrl = '../assets/phone.svg';
```

colspan and *colSpan*

A common point of confusion is between the attribute, `colspan`, and the property, `colSpan`. Notice that these two names differ by only a single letter.

To use property binding using `colSpan`, type the following:

src/app/app.component.html

```
content_copy<!-- Notice the colSpan property is camel case -->

<tr><td [colSpan]="1 + 1">Three-Four</td></tr>
```

To disable a button while the component's `isUnchanged` property is `true`, type the following:

src/app/app.component.html

```
content_copy<!-- Bind button disabled state to `isUnchanged`
property -->

<button type="button" [disabled]="isUnchanged">Disabled
Button</button>
```

To set a property of a directive, type the following:

src/app/app.component.html

```
content_copy<p [ngClass]="classes">[ngClass] binding to the classes
property making this blue</p>
```

To set the model property of a custom component for parent and child components to communicate with each other, type the following:

src/app/app.component.html

```
content_copy<app-item-detail [childItem]="parentItem"></app-item-
detail>
```

Toggling button features

To use a Boolean value to disable a button's features, bind the `disabled` DOM attribute to a Boolean property in the class.

src/app/app.component.html

```
content_copy<!-- Bind button disabled state to `isUnchanged`
property -->

<button type="button" [disabled]="isUnchanged">Disabled
Button</button>
```

Because the value of the property `isUnchanged` is `true` in the `AppComponent`, Angular disables the button.

src/app/app.component.ts

```
content_copyisUnchanged = true;
```

What is the difference between interpolation and property binding?

Difference between String interpolation and Property Binding

	String Interpolation	Property Binding
Syntax	`${variable}`	`[]`
Usage	Used to insert variables into strings, such as in template literals or in JSX in React.	Used to bind variables or properties to HTML attributes or DOM properties.
Variable/Property Evaluation	is Evaluated at runtime, meaning any changes to the variables will be reflected in the interpolated string.	Evaluated when the property is bound and does not update automatically if the value changes later.
Applicable to	Strings and templates.	HTML elements and DOM properties.
Dynamic Updating	Automatically update the string when the variable changes.	Require manual re-binding if the variable changes.
Data Types	Work with strings, numbers, and boolean values.	Can bind to any data type, including arrays, objects, and functions.
Compatibility	Works with most modern browsers.	Works with Angular, React, and other

	String Interpolation	Property Binding
		front-end frameworks.

Binding to events

To bind to an event you use the Angular event binding syntax. This syntax consists of a target event name within parentheses to the left of an equal sign, and a quoted template statement to the right.

Create the following example; the target event name is `click` and the template statement is `onSave()`.

Event binding syntax

```
content_copy<button (click)="onSave()">Save</button>
```

The event binding listens for the button's click events and calls the component's `onSave()` method whenever a click occurs.

Determining an event target

To determine an event target, Angular checks if the name of the target event matches an event property of a known directive.

Create the following example: (Angular checks to see if `myClick` is an event on the custom `ClickDirective`)

src/app/app.component.html

```
content_copy<h4>myClick is an event on the custom ClickDirective:</h4>

<button type="button" (myClick)="clickMessage=$event" clickable>click
with myClick</button>

{{clickMessage}}
```

If the target event name, `myClick` fails to match an output property of `ClickDirective`, Angular will instead bind to the `myClick` event on the underlying DOM element.

Binding to passive events

This is an advanced technique that is not necessary for most applications. You may find this useful if you want to optimize frequently occurring events that are causing performance problems.

Angular also supports passive event listeners. For example, use the following steps to make a scroll event passive.

1. Create a file `zone-flags.ts` under `src` directory.
2. Add the following line into this file.

```
content_copy(window as any)['__zone_symbol__PASSIVE_EVENTS'] =
['scroll'];
```

3. In the `src/polyfills.ts` file, before importing zone.js, import the newly created `zone-flags`.

```
4. content_copyimport './zone-flags';

   import 'zone.js';  // Included with Angular CLI.
```

After those steps, if you add event listeners for the `scroll` event, the listeners will be `passive`.

Binding to keyboard events

You can bind to keyboard events using Angular's binding syntax. You can specify the key or code that you would like to bind to keyboard events. They `key` and `code` fields are a native part of the browser keyboard event object. By default, event binding assumes you want to use the `key` field on the keyboard event. You can also use the `code` field.

Combinations of keys can be separated by a `.` (period). For example, `keydown.enter` will allow you to bind events to the `enter` key. You can also use modifier keys, such as `shift`, `alt`, `control`, and the `command` keys from Mac. The following example shows how to bind a keyboard event to `keydown.shift.t`.

```
content_copy<input (keydown.shift.t)="onKeydown($event)" />
```

Depending on the operating system, some key combinations might create special characters instead of the key combination that you expect. MacOS, for example, creates special characters when you use the option and shift keys together. If you bind to `keydown.shift.alt.t`, on macOS, that combination produces a ˇ character instead of a `t`, which doesn't match the binding and won't trigger your event handler. To bind to `keydown.shift.alt.t` on macOS, use the `code` keyboard event field to get the correct behavior, such as `keydown.code.shiftleft.altleft.keyt` shown in this example.

```
content_copy<input
(keydown.code.shiftleft.altleft.keyt)="onKeydown($event)" />
```

The `code` field is more specific than the `key` field. The `key` field always reports `shift`, whereas the `code` field will specify `leftshift` or `rightshift`. When using the `code` field, you might need to add separate bindings to catch all the behaviors you want. Using the `code` field avoids the need to handle OS specific behaviors such as the `shift + option` behavior on macOS.

For more information, visit the full reference for key and code to help build out your event strings.

tEvent binding

Event binding lets you listen for and respond to user actions such as keystrokes, mouse movements, clicks, and touches.

> See the live example / download example for a working example containing the code snippets in this guide.

Prerequisites

- Basics of components
- Basics of templates
- Binding syntax
- Template statements

Binding to events

To bind to an event you use the Angular event binding syntax. This syntax consists of a target event name within parentheses to the left of an equal sign, and a quoted template statement to the right.

Create the following example; the target event name is `click` and the template statement is `onSave()`.

Event binding syntax

```
content_copy<button (click)="onSave()">Save</button>
```

The event binding listens for the button's click events and calls the component's `onSave()` method whenever a click occurs.

Determining an event target

To determine an event target, Angular checks if the name of the target event matches an event property of a known directive.

Create the following example: (Angular checks to see if `myClick` is an event on the custom `ClickDirective`)

src/app/app.component.html

```
content_copy<h4>myClick is an event on the custom
ClickDirective:</h4>

<button type="button" (myClick)="clickMessage=$event"
clickable>click with myClick</button>

{{clickMessage}}
```

If the target event name, `myClick` fails to match an output property of `ClickDirective`, Angular will instead bind to the `myClick` event on the underlying DOM element.

Binding to passive events

This is an advanced technique that is not necessary for most applications. You may find this useful if you want to optimize frequently occurring events that are causing performance problems.

Angular also supports passive event listeners. For example, use the following steps to make a scroll event passive.

1. Create a file `zone-flags.ts` under `src` directory.
2. Add the following line into this file.

```
content_copy(window as
any)['__zone_symbol__PASSIVE_EVENTS'] = ['scroll'];
```

3. In the `src/polyfills.ts` file, before importing zone.js, import the newly created `zone-flags`.

```
4. content_copyimport './zone-flags';

   import 'zone.js';  // Included with Angular CLI.
```

After those steps, if you add event listeners for the `scroll` event, the listeners will be `passive`.

Binding to keyboard events

You can bind to keyboard events using Angular's binding syntax. You can specify the key or code that you would like to bind to keyboard events. They `key` and `code` fields are a native part of the browser keyboard event object. By default, event binding assumes you want to use the `key` field on the keyboard event. You can also use the `code` field.

Combinations of keys can be separated by a `.` (period). For example, `keydown.enter` will allow you to bind events to the `enter` key. You can also use modifier keys, such as `shift`, `alt`, `control`, and the `command` keys from Mac. The following example shows how to bind a keyboard event to `keydown.shift.t`.

```
content_copy<input (keydown.shift.t)="onKeydown($event)" />
```

Depending on the operating system, some key combinations might create special characters instead of the key combination that you expect. MacOS, for example, creates special characters when you use the option and shift keys together. If you bind to `keydown.shift.alt.t`, on macOS, that combination produces a ˇ character instead of a `t`, which doesn't match the binding and won't trigger your event handler. To bind to `keydown.shift.alt.t` on macOS, use the `code` keyboard event field to get the correct behavior, such as `keydown.code.shiftleft.altleft.keyt` shown in this example.

```
content_copy<input
(keydown.code.shiftleft.altleft.keyt)="onKeydown($event)"
/>
```

The `code` field is more specific than the `key` field. The `key` field always reports `shift`, whereas the `code` field will specify `leftshift` or `rightshift`. When using the `code` field, you might need to add separate bindings to catch all the behaviors you want. Using the `code` field avoids the need to handle OS specific behaviors such as the `shift + option` behavior on macOS.

Angular 7 Components

Components are the key features of Angular. The whole application is built by using different components.

The core idea behind Angular is to build components. They make your complex application into reusable parts which you can reuse very easily.

How to create a new component?

Open WebStorm>> Go to your project source folder>> Expand the app directory and create a new directory named "server".

Now, create the component within server directory. Right click on the server directory and create a new file named as "server.component.ts". It is the newly created component.

Components are used to build webpages in Angular but they require modules to bundle them together. Now, you have to register our new components in module.

Creating component with CLI

Syntax

1. ng generate component component_name
2. Or
3. ng g c component_name

Let's see how to create a new component by using command line.

Open Command prompt and stop **ng serve** command if it is running on the browser.

Type **ng generate component server2** to create a new component named server2.

You can also use a shortcut **ng g c server2** to do the same task.

In the above image, you can see that a new component named "server2" is created. It contains the same other components which you have seen when we create a first app..

1. server2.component.css
2. server2.component.html
3. server2.component.spec.ts
4. server2.component.ts

Here, **server2.component.spec.ts** component is used for testing purpose. You can delete it by right click on that.

What is async in Angular?

In Angular, the async pipe is a pipe that essentially does these three tasks: It subscribes to an observable or a promise and returns the last emitted value. Whenever a new value is emitted, it marks the component to be checked. That means Angular will run Change Detector for that component in the next cycle.

Async functions with AngularJS

April 5, 2015Vijay Thirugnanam

AngularJS allows to define async functions. An example is the $http service. HTTP service calls an API and returns a response as a promise.

```
$http.get(url)

    .then(function(res){

        // handle the response

    }, function(res){

        // handle the error

    });
```

Retrieve the result from the Promise object using `then` function. The function has two arguments: `onSuccess` callback and `onFailure` callback.

The $q service in AngularJS creates a promise using the `defer` function.

```
$scope.callDb = function () {

  var deferred = $q.defer();

  promise.then(function(res){

    var someObj = {};

    deferred.resolve(someObj);

  }, function(err){

    var errObj = {};
```

```
    deferred.reject(errObj);

  });

  return deferred.promise;

};
```

Pass the result using the `resolve` function. If there is an error, pass the error using `reject` function.

I want to give a concrete example. Cordova has a SQLite plugin (ngCordova). Create a DbService that returns a list of employees.

```
app.service('DbService',

    function ($q, $cordovaSQLite) {

        this.get = function() {

            var q = $q.defer();

            var query = "SELECT idemployee, name from employee";

            var db = $cordovaSQLite.openDB("employee.sqlite",
0);

            $cordovaSQLite.execute(db, query, []).then(function
(res) {

                var employees = [];

                for (index = 0; index < res.rows.length;
index++) {

                    employees.push(res.rows.item(index));
```

```
                }

                q.resolve(employees);

            }, function (err) {

                q.reject(null);

            });

            return q.promise;

        };

    });
```

In the above example, `DbService` is an Angular service. We have a SQL that retrieves a list of employees. SQLite plugin has an execute function which queries the database using the SQL. The plugin returns a promise. Retrieve a list of employees from the promise. And return a new promise using Angular's `$q` service.

Call the `DbService` to get the employee list. The `get` method returns a promise. From the `then` function of the promise, we set the employees to the scope object.

```
function get() {

    DbService.get()

    .then(function(employees){

        $scope.employees = employees;

    });

}
```

Bind the Angular's scope object to a table using ng-repeat directive.

template interpolation in Angular

What is the template interpolation in Angular?

In AngularJS, Interpolation is a way to transfer the data from a TypeScript code to an HTML template (view), i.e. it is a method by which we can put an expression in between some text and get the value of that expression. Interpolation basically binds the text with the expression value.

Interpolation and template expressions

Interpolation allows you to incorporate calculated strings into the text between HTML element tags and within attribute assignments. Template expressions are what you use to calculate those strings.

See the live example / descargar ejemplo for all of the syntax and code snippets in this guide.

Interpolation { { . . . } }

Interpolation refers to embedding expressions into marked up text. By default, interpolation uses as its delimiter the double curly braces, `{{` and `}}`.

In the following snippet, `{{ currentCustomer }}` is an example of interpolation.
src/app/app.component.html

```
content_copy<h3>Current customer: {{ currentCustomer }}</h3>
```

The text between the braces is often the name of a component property. Angular replaces that name with the string value of the corresponding component property.
src/app/app.component.html

```
content_copy<p>{{title}}</p>

<div><img src="{{itemImageUrl}}"></div>
```

In the example above, Angular evaluates the `title` and `itemImageUrl` properties and fills in the blanks, first displaying some title text and then an image.

More generally, the text between the braces is a template expression that Angular first evaluates and then converts to a string. The following interpolation illustrates the point by adding two numbers:
src/app/app.component.html

```
content_copy<!-- "The sum of 1 + 1 is 2" -->

<p>The sum of 1 + 1 is {{1 + 1}}.</p>
```

The expression can invoke methods of the host component such as `getVal()` in the following example:
src/app/app.component.html

```
content_copy<!-- "The sum of 1 + 1 is not 4" -->
```

```
<p>The sum of 1 + 1 is not {{1 + 1 + getVal()}}.</p>
```

Angular evaluates all expressions in double curly braces, converts the expression results to strings, and links them with neighboring literal strings. Finally, it assigns this composite interpolated result to an element or directive property.

You appear to be inserting the result between element tags and assigning it to attributes. However, interpolation is a special syntax that Angular converts into a *property binding*.

If you'd like to use something other than `{{` and `}}`, you can configure the interpolation delimiter via the interpolation option in the `Component` metadata.

Template expressions

A template expression produces a value and appears within the double curly braces, `{{ }}`. Angular executes the expression and assigns it to a property of a binding target; the target could be an HTML element, a component, or a directive.

The interpolation braces in `{{1 + 1}}` surround the template expression `1 + 1`. In the property binding, a template expression appears in quotes to the right of the = symbol as in `[property]="expression"`.

In terms of syntax, template expressions are similar to JavaScript. Many JavaScript expressions are legal template expressions, with a few exceptions.

You can't use JavaScript expressions that have or promote side effects, including:

- Assignments (=, +=, -=, . . .)
- Operators such as `new`, `typeof`, `instanceof`, etc.
- Chaining expressions with `;` or `,`
- The increment and decrement operators ++ and --
- Some of the ES2015+ operators

Other notable differences from JavaScript syntax include:

- No support for the bitwise operators such as `|` and `&`
- New template expression operators, such as `|`, `?.` and `!`

Expression context

The *expression context* is typically the *component* instance. In the following snippets, the `recommended` within double curly braces and the `itemImageUrl2` in quotes refer to properties of the `AppComponent`.

src/app/app.component.html

```
content_copy<h4>{{recommended}}</h4>

<img [src]="itemImageUrl2">
```

An expression may also refer to properties of the *template's* context such as a template input variable,

`let customer`, or a template reference variable, `#customerInput`.

src/app/app.component.html (template input variable)

```
content_copy<ul>

  <li *ngFor="let customer of customers">{{customer.name}}</li>

</ul>
```

src/app/app.component.html (template reference variable)

```
content_copy<label>Type something:

  <input #customerInput>{{customerInput.value}}

</label>
```

The context for terms in an expression is a blend of the *template variables*, the directive's *context* object (if it has one), and the component's *members*. If you reference a name that belongs to more than one of these namespaces, the template variable name takes precedence, followed by a name in the directive's *context*, and, lastly, the component's member names.

The previous example presents such a name collision. The component has a `customer` property and the `*ngFor` defines a `customer` template variable.

The `customer` in `{{customer.name}}` refers to the template input variable, not the component's property.

Template expressions cannot refer to anything in the global namespace, except `undefined`. They can't refer to `window` or `document`. Additionally, they can't call `console.log()` or `Math.max()` and they are restricted to referencing members of the expression context.

Expression guidelines

When using template expressions follow these guidelines:

- Simplicity
- Quick execution
- No visible side effects

Simplicity

Although it's possible to write complex template expressions, it's a better practice to avoid them.

A property name or method call should be the norm, but an occasional Boolean negation, `!`, is OK. Otherwise, confine application and business logic to the component, where it is easier to develop and test.

Quick execution

Angular executes template expressions after every change detection cycle. Change detection cycles are triggered by many asynchronous activities such as promise resolutions, HTTP results, timer events, key presses and mouse moves.

Expressions should finish quickly or the user experience may drag, especially on slower devices. Consider caching values when their computation is expensive.
No visible side effects

A template expression should not change any application state other than the value of the target property.

This rule is essential to Angular's "unidirectional data flow" policy. You should never worry that reading a component value might change some other displayed value. The view should be stable throughout a single rendering pass.

An idempotent expression is ideal because it is free of side effects and improves Angular's change detection performance. In Angular terms, an idempotent expression always returns *exactly the same thing* until one of its dependent values changes.

Dependent values should not change during a single turn of the event loop. If an idempotent expression returns a string or a number, it returns the same string or number when called twice in a row. If the expression returns an object, including an `array`, it returns the same object *reference* when called twice in a row.

> There is one exception to this behavior that applies to `*ngFor`. `*ngFor` has `trackBy` functionality that can deal with referential inequality of objects when iterating over them. See *ngFor with `trackBy` for details.

Two-way Data Binding in AngularJS

-

In this article, we will see the Data Binding, along with understanding how the flow of code is from a Typescript file to an HTML file & vice-versa through their implementation.

In AngularJS, **Data Binding** refers to the synchronization between the model and view. In **Two-way data binding**, the flow of data is bidirectional i.e. when the data in the model changes, the changes are reflected in the view and when the data in the view changes it is reflected in the model. Two-way data binding is achieved by using the *ng-model directive*. The ng-model directive transfers data from the view to the model and from the model to the view.
Approach: The following approach will be implemented to achieve the Two-way Binding:

- Create a module

```
var app=angular.module('myApp',[])
```

- Add a controller to the module. Here you can write the logic for updating the model as the view changes.

```
app.controller('myCtrl',function($scope){})
```

- Specify the ng-model directive in the element

```
<input ng-model="name"/>
```