AngularJS

MVC Based JavaScript Framework

Extends HTML with new attributes.
Perfect for Single Page Applications (SPAs).
Client side MVC framework...

From : Anil Kumar
Kooud Software Pvt Lmt.

AngularJS

Agenda

What is AngularJS
Environment Setup
MVC Architecture in AngularJS
Directives and Expressions
How AngularJS simplifies & reduce the coding.
Modules and Controllers
Scope in AngularJS
Round trip of **ng-*** Directives

AngularJS is a Client-Side MVC JavaScript framework. official definition.

It was originally developed in 2009 by Misko Hevery and Adam Abrons.
It is now maintained by Google.
Its latest version is 1.4.3 and new framework is AngularJS 2.0.

Features :

MVC architecture in development.

Rich Internet Application and Single Page Application.
Cross browser compatibility of JS code.
Freely available resource. [angularjs.org]

Cont...

Core Features :

- Scope
- Data-Binding
- Controller
- Services
- Factories
- Directive
- Filters
- Template
- Routing
- Constants

Cont...

Advantages :

Single Page Application development in clean and maintainable way .
Two way data binding within model and view.
Provides dependency injection and allow independence of each model.
More functionality with less coding.
AngularJS runs on all major browser and smartphones including Android & ISO

Disadvantages :

Being JavaScript based framework, it is not secure. Requires server side authentication and authorization.
If JavaScript is disabled means only view can be displayed nor any functionality is supported.

Content Delivery Network :

```
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js">
</script>
```

Download AngularJS hosting files locally : download

There are two different options legacy and latest. The names itself are self descriptive. legacy has version less than 1.2.x and latest has 1.5.x version.

Example when locally available :

```
<script src= "D://AngularProject/js/angular.min.js"></script>
```

Note :

Using CDN have recent updated library functionalities.

While using CDN, you need server connection compulsorily.

MVC Architecture in AngularJS

AngularJS is designed using MVC pattern, so it's called as Client Side MVC framework. It divides the whole process into Model, View and Controller.

Mapping of MVC to AngularJS Directives

Model	→	ng-model	i,e JS Object
View	→	ng-bind	i,e html or DOM
Controller	→	ng-controller	i,e JS classes or functions

How It Works :

Model and View are both transference to each other, whenever data updated in view is immediately effects the model and whenever model is updated with data in controller is calls the view to change the view data. So controller is glue between view and model.

Controller responds to user triggers and performs validation on received inputs and updates the model. Model data is organized as JSON object in AngularJS.

Directives

AngularJS directives are extends HTML tags with new attributes. AngularJS directives starts with ng-* prefix.

ng-app	→	This starts AngularJS Application. try it
ng-init	→	This directive initiate required application data. try it
ng-model	→	This directive binds the value of HTML controls to AngularJS application data.
ng-bind	→	This directive binds application data to the HTML view.

Expressions

AngularJS expressions are written inside double braces: {{ expression }}.
It is alternative to **ng-bind** directive**.**
Examples :

```
<div ng-app="myApp" ng-init="myName='Ram'; lname='patil';">
    Summation of 20 & 40 is {{ 20 + 30 }}
    My Name is {{myName}} <span ng-bind='lname' />
<div>
```

AngularJS

How AngularJS simplifies & reduce the coding.

Let us see, one scenario of updating content of **<p>** tag and **<input>** text box in JavaScript, jQuery and finally using AngularJS.

In JavaScript : Requires DOM manipulation in JavaScript.

```
<p id='p1'> Hello World </p>
<input type='text'  id='name' value='Ram'/>
<script>
    var pDom = document.getElementById("p1");
    pDom.innerHTML = "New text!";

    var nDom = document.getElementById("name")
    nDom.value = "Sham";
</script>
```

Even it becomes more complex when application have more dependencies of updating data on other DOM elements

Cont...

AngularJS

How AngularJS simplifies & reduce the coding.

Let us see, one scenario of updating content of **<p>** tag and **<input>** text box in JavaScript, jQuery and finally using AngularJS.

In jQuery:

```
<p id='p1'> Hello World </p>
<input type='text'  id='name' value='Ram'/>

<script>
    $(document).ready(function(){
        $("#p1").html("New Text");
        $("#name").val("Sham");
    });
</script>
```

Here we have dependency on id selector or class selector for manipulating DOM.

Cont...

AngularJS

How AngularJS simplifies & reduce the coding.

Let us see, one scenario of updating content of **<p>** tag and **<input>** text box in JavaScript, jQuery and finally using AngularJS.

In AngularJS:

```
<div ng-app="myApp"  ng-controller="myCtrl">
    <p ng-bind='p1'> Hello World </p>
    <input type='text'  ng-model='name'  value='Ram'/>
</div>
<script>
    var app = angular.module('myApp', []);
    app.controller('myCtrl', function($scope) {

        $scope.p1 = "New Text";
        $scope.name =  "Sham";
    });
</script>
```

It's like a simple variable assignment to model data i,e via $scope object.

Cont...

AngularJS

How AngularJS simplifies & reduce the coding.

Let's try simple AngularJS example : open in browser

```
<!DOCTYPE html>
<html>
	<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js">
	</script>
<body>
	<div ng-app="" ng-init="fName='Sham'; lName='Sundar';">

		<h2>First Name : <input type="text" ng-model="fName" /></h2>
		<h2>Last Name : <input type="text" ng-model="lName" /></h2>

		<h2>First Name : <span ng-bind="fName" /></h3>
		<h2>Last Name : {{lName}}</h3>
		<h3>Full Name : {{fName+ "  "+ lName}}</h3>
	</div>
</body>
</html>
```

AngularJS Modules

AngularJS application is consider as a single module, hence module defines an application. There may be more than one module for single application.

- Modules are defined using **ng-app** directive.
- Module is container for the application controller, written in JavaScript.
- Even modules can be written in separate js file for global use.

Creating a Module :

```
<div ng-app="myApp">  </div>

<script>
      var app = angular.module("myApp",  []);
</script>
```

Cont...

AngularJS Controller

AngularJS applications are controlled by controllers. AngularJS controller controllers the flow of data in the application module.

- Controllers are defined using **ng-controller** directive.
- AngularJS controllers are regular JavaScript Object.
- Each controller accepts the $scope as default parameter representing application model data.
- All controllers are belongs to one of the module in application.
- There may be more than one controller in single module for different aspects.
- Even controllers also can be written in separate js file for global use.

Cont...

AngularJS Controller : different way of defining controller.

```
<div ng-app="myApp"  ng-controller="myCtrl">

    First Name: <input type="text" ng-model="fName">
    Last Name: <input type="text" ng-model="lName">
    Full Name: {{fName + " " + lName}}
</div>

<script>
    angular.module("myApp", []).controller("myCtrl", function($scope){

    });
</script>
<script>
    var app = angular.module("myApp", []);
    app.controller("myCtrl", function($scope){

    });
</script>
```

Cont...

AngularJS Controller : different way of defining controller.

```
<script>
	var app = angular.module("myApp", []);
	app.controller("myCtrl", myControllerFun);

	function myControllerFun($scope)  { … }
</script>
```

```
<script>
	var app = angular.module("myApp", []);
	app.controller("myCtrl", ["$scope", "$http",  function(myScope, httpCon)	{

		// resolve the reference, if  this controller js file is manifested
	}]);
</script>
```

Cont...

AngularJS Controller : demo program try it

```
<body>
    <div ng-app="myApp" ng-controller="personCtrl">
        First Name: <input type="text" ng-model="firstName">
        Last Name: <input type="text" ng-model="lastName">
        Full Name: {{fullName()}}
    </div>
</body>
<script>
    var app = angular.module('myApp', []);
    app.controller('personCtrl', function($scope) {
        $scope.firstName = "John";
        $scope.lastName = "Doe";
        $scope.fullName = function() {
            return $scope.firstName + " " + $scope.lastName;
        };
    });
</script>
```

Cont...

AngularJS Controller : demo program try it

In large application it's better to place AngularJS controller in external file.

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js">
</script>
<body>
    <div ng-app="myApp" ng-controller="personCtrl">
        First Name: <input type="text" ng-model="firstName">
        Last Name: <input type="text" ng-model="lastName">
        Full Name: {{fullName()}}
    </div>
    <script src="personController.js"></script>
</body>
</html>
```

AngularJS Scope :

Scope is binding between HTML view and Javascript(controller).

Scope is an object with properties and methods.

How to use Scope :

Scope object is denoted by $scope variable name.

When defining controller, just we pass the scope object.

Only property/method name is enough in view to access $scope values.

Example :

```
angular("myApp", []).controller("myCtrl", function($scope){
    $scope.message = "Have A Nice Day !";
    $scope.getMessage = function()    {
        return $scope.message;
    }
});
<div ng-app = "myApp" ng-controller = "myCtrl">
    <span ng-bind = "message" /> {{getMessage()}}
<div>
```

Cont...

Root Scope

Root scope is parent of all current scopes related respective controllers.
It is created on the top HTML elements that contains the **ng-app** directive.
Root scope is denoted by $rootScope and it's available in entire application.

```
<body ng-app="myApp">
    <p>message from rootScope's : {{message}} </p>
    <div ng-controller="myCtrl">
        <p>message from the scope of the controller's : {{message}}</p>
    </div>
    <p>message from rootScope's is still : {{message}}</p>
    <script>
        var app = angular.module('myApp', []);
        app.run(function($rootScope) {
            $rootScope.message = "Hello ! How are you...";
        });
        app.controller('myCtrl', function($scope) {
            $scope.message = "I'm fine...";
        });
    </script>
</body>
```

Cont...

Scope vs Root Scope Result try it

message from rootScope's : **Hello ! How are you...**

message from the scope of the controller's : **I'm fine...**

message from rootScope's is still : **Hello ! How are you...**

Note : Property value of current scope can't override the value root scope.

Round trip of **ng-*** Directives :

- **ng-show**
- **ng-hide**
- **ng-disabled**
- **ng-repeate**
- **ng-class**
- **ng-style**
- **ng-switch**
- **ng-animate**
- **ng-include**
- **ng-click**
- **ng-dblclick**
- **ng-key***
- **ng-mouse***

click here for all ng-*

Cont...

Round trip of **ng-*** Directives : **ng-show**

The **ng-show** directive shows or hides HTML element depending on passed expression to attribute.

Usage of **ng-show** :

<div **ng-show** = "true/false"> …..

<div **ng-show** = "checkedFlag"> …. try it

<div **ng-show** = "selection == 'Apple' ">

<div **ng-show** = "getStatus()" >

Cont...

Round trip of **ng-*** Directives : **ng-hide**

The **ng-hide** directive hides or shows HTML element depending on passed expression to attribute.

Usage of **ng-hide** :

<div **ng-hide** = "true/false"> …..

<div **ng-hide** = "checkedFlag"> …. try it

<div **ng-hide** = "index == 3 ">

<div **ng-hide** = "getStatus()" >

Cont...

Round trip of **ng-*** Directives : **ng-disabled**

The **ng-disabled** directive sets the disabled attribute on HTML element if passed expression is evaluates to true.

Very helpful in enabling/disabling input elements such as check box, text box, selection box, radio box and buttons.

Usage of **ng-disabled** : try it

Check <input type="checkbox" ng-model="checked" />

<input type='text' ng-model="button" ng-disabled="checked" />

<button ng-model="button" ng-disabled="checked">Button</button>

<select ng-disabled='true'> … </select>

Cont...

Round trip of **ng-*** Directives : **ng-repeat**

The **ng-repeat** directive repeats an HTML element by cloning that once, for each element in the collection.

This is applied on arrays or array of objects for similar repeated operation.

AngularJS ng-repeat is perfect for database CRUD operation.

Some Special properties provided by local scope of **ng-repeat** are :

$index	: integer	→	maintains iteration offset of elements.
$first	: boolean	→	true if current element is first element.
$middle	: boolean	→	true if current element is in between first & last.
$last	: boolean	→	true if current element is last element.
$even	: boolean	→	true if current element index is even.
$odd	: boolean	→	true if current element index is odd.

Cont...

Round trip of **ng-*** Directives : **ng-repeat**

Iterating over array elements : try it

```
<div ng-app="" ng-init="names=['Jani', 'Hege', 'Kai']">
    <ul>
        <li ng-repeat="x in names">  {{ x }}  </li>
    </ul>
</div>
```

Iterating over object : try it

```
<div ng-app="" ng-init="person=[{name:'Ram', age:'21'},
{name:'sham', age:18}, {name:'sundar', age:25}]">
    <ul>
        <li ng-repeat="p in person">
            {{$index + 1}}.  {{p.name}} is {{p.age}} year old.
        </li>
    </ul>
</div>
```

Cont...

Round trip of **ng-*** Directives : **ng-class**

The **ng-class** directive allows us to add/remove the CSS classes dynamically based on some conditions and data.

There are many ways to add CSS classes on any HTML elements.

1] **ng-class** using class name directly

```
<h2 ng-class=" 'text-danger' ">using single class name</h2>

<h2 ng-class=" 'text-danger   magenta' ">using multiple class name.</h2>

<h2 ng-class="['text-danger',  'magenta']">using multiple class name as array
      of class names.</h2>
```

Cont...

Round trip of **ng-*** Directives : **ng-class**

There are many ways to add CSS classes on any HTML elements.

2] **ng-class** using class name from string name of model variables

```
<input type="text" ng-model="textType">

<div ng-class="textType">Look! I'm Words!</div>
```

Using Array syntax for more than one type of CSS properties bounding

```
<input type="text" ng-model="styleOne">
<input type="text" ng-model="styleTwo">

<div ng-class="[styleOne, styleTwo]">Look! I'm Words!</div>
```

Cont...

Round trip of **ng-*** Directives : **ng-class**

There are many ways to add CSS classes on any HTML elements.

3] **ng-class** using class name from evaluated expressions.

```
<input type="checkbox" ng-model="awesome"> Are You Awesome?
<input type="checkbox" ng-model="large"> Are You a Large Text?
```

It adds the class 'text-success', if and only if the variable 'awesome' is true.

```
<div ng-class="{ 'text-success': awesome, 'text-large': large }">
```

4] **ng-class** using the Ternary Operator.

```
<h1 ng-class=" $even ? 'even-row' : 'odd-row' ">Me Odd or Even?</h1>

<h1 ng-class="awesome ? 'text-success' : 'text-danger' "> Ohh.. MG</h1>
```

Cont...

Round trip of **ng-*** Directives : **ng-class**

There are many ways to add CSS classes on any HTML elements.

5] **ng-class** using with **ng-repeat** properties.

```
<table class="table table-hover">
      <tr>
            <th> Sl No </th> <th> City </th> <th> Contry </th>
      </tr>
      <tr  ng-repeat="Arr in Arrs"
            ng-class="{'danger':$first, 'warning':$last, 'active':$even, 'info':$odd}">
            <td>      {{$index + 1}}   </td>
            <td>      {{Arr.City}}      </td>
            <td>      {{Arr.Country}}  </td>
      </tr>
</table>
```

Cont...

Round trip of **ng-*** Directives : **ng-class**

There are many ways to add CSS classes on any HTML elements.

5] **ng-class** using condition on map of objects.

```
<tr  ng-repeat="Arr in Arrs"
      ng-class="{false:'active', true:'info'}[$even]">
      <td>     {{$index + 1}}   </td>
      <td>     {{Arr.City}}     </td>
      <td>     {{Arr.Country}} </td>
</tr>

ng-class="{false:'danger', true:'info'}[$first || $last] "
ng-class="{false:'danger', true:'info'}[$index != 0 || $last == true] "
```

Cont...

Round trip of **ng-*** Directives : **ng-style**

The **ng-style** directive allows you to set CSS style on an HTML element conditionally.

```
<input type="button" value="clear"       ng-click="myStyle={}">
<input type="button" value="set color"  ng-click="myStyle={color:'red'}">
<input type="button" value="set background"
      ng-click="myStyle={'background-color':'blue'}">

<span ng-style="myStyle">Sample Text</span>
```

```
<h1 ng-style="myObj">Welcome</h1>

  $scope.myObj = {    "color" : "red",
                      "background-color" : "blue",
                      "font-size" : "60px"
}
```

Cont...

Round trip of **ng-*** Directives : **ng-style**

The **ng-style** directive allows you to set CSS style on an HTML element conditionally.

```
<input type="button" value="clear"       ng-click="myStyle={}">
<input type="button" value="set color"  ng-click="myStyle={color:'red'}">
<input type="button" value="set background"
      ng-click="myStyle={'background-color':'blue'}">

<span ng-style="myStyle">Sample Text</span>
```

```
<h1 ng-style="myObj">Welcome</h1>

  $scope.myObj = {    "color" : "red",
                      "background-color" : "blue",
                      "font-size" : "60px"
}
```

Cont...

Round trip of **ng-*** Directives : **ng-switch**

The **ng-switch** directive allows dynamically swap DOM structure for a template element conditionally. try it

Similar to switch of other languages, having default option if none is matched.
Best suit of tab navigation and conditional content rendering.
n**g-switch** expression should evaluate to **String** literal.

```
<input type="radio" value="apple"  ng-model="fruits"> Apple
<input type="radio" value="banana" ng-model="fruits"> Banana
<input type="radio" value="mango" ng-model="fruits"> Mango

<div ng-switch="fruits">
      <div ng-switch-when="apple">A for Apple</div>
      <div ng-switch-when="banana">B for Banana</div>
      <div ng-switch-when="mango">C for Chotta Mango</div>
      <div ng-switch-default="mango">Select Fruit</div>
</div>
```

Cont...

Round trip of **ng-*** Directives : **ng-include**

The **ng-include** directive fetches, compiles and inserts an external HTML fragment into the current page. Added fragment is considered as child nodes of specific element.

```
<body ng-app="">

    <div ng-include="'myFile.htm'"></div>
</body>
```

try it

The **ng-include** attribute value can be an expression, returning a file name.
The **ng-include** can have onload and autoscroll attributes.
onload → perform some action after loading the external HTML fragment.
autoscroll → makes scrolling behavior automatic depending on content.

If onload event is going to call function means it should be within scope.

Cont...

Round trip of **ng-*** Directives : **ng-if**

The **ng-if** directive removes or recreates a portion of the DOM tree based on an the outcome of expression.

Difference between **ng-if** and **ng-hide/ng-show** is **ng-if** completely removes the element from DOM in case if expression evaluates to false and again recreates element newly once it again turns to true.

Keep HTML:

```
<input type="checkbox" ng-model="myVar" ng-init="myVar = true">

<div ng-if="myVar">
    <h1>Welcome</h1>
    <p>Welcome to my home.</p>
    <hr>
</div>
```

try it

Cont...

Round trip of **ng-*** Directives : **ng-click**

The **ng-click** directive allows to change state/value of scope variable and specify custom behavior using JS function when click event is performed on container element.

```
<button ng-click="count = count + 1" ng-init="count=0"> Increment </button>

<span>  count: {{count}}</span>
```

try it

```
<button ng-click="showInfo = !showInfo"> Hide/Show Info </button>

<div ng-show="showInfo"   ng-init="showInfo=false;">
     <h1> Some Information </h1>
</div>
```

Cont...

Round trip of **ng-*** Directives : **ng-click**

```
<tr ng-repeat="rec in contactsBook">
      <td ng-bind="rec.name"></td>      <td ng-bind="rec.number"></td>
      <td>
            <button value="Modify"   ng-click="update(rec.name)" />
            <button value="Remove"   ng-click="remove($index)" />
            <button value="Call"     ng-click="call($index)" />
      </td>
</tr>

angular.module("myapp", []).controller("ContactBook", function($scope){
      $scope.contactsBook = [  {name:"Ram",    number:"8123456789"},
                               {name:"Sham",   number:"8987653210"}];
      $scope.update   = function(name)     {....};
      $scope.remove   = function(recIndex) {....};
});
function call(callerIndex) {....}
```

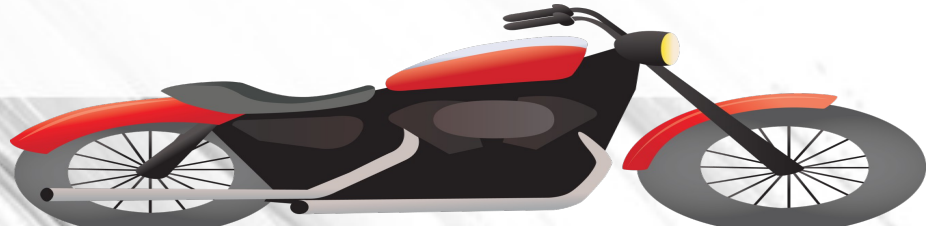
Cont...

Round trip of **ng-*** Directives : **ng-mouse* and ng-key* examples**

- **ng-click**
- **ng-dblclick**
- **ng-mouseup**
- **ng-mousedown**
- **ng-mouseenter**
- **ng-mouseleave**
- **ng-mouseover**
- **ng-mousemove**
- **ng-keyup**
- **ng-keydown**
- **ng-keypress**
- **ng-focus**
- **ng-change**

Note : The called method should be in one of the scope of an AngularJs controller.

If simply variables are used within **ng-*=' ',** then that should be belongs to AngularJs scope within application.



Trip completed click here remaining ng-* dir

Filters in AngularJS acts as a data format specifiers with additional power of searching and data presentation control ability.

AngularJS Filters :

- **currency** → Represents string/number in localized currency format.
- **date** → Format a date to a specified format [custom format].
- **filter** → Provides searching option and selects subset from array.
- **json** → Represents JSON object in JSON string format.
- **limitTo** → Provides navigation capability by limiting no of records.
- **lowercase** → Represents a string in lower case format.
- **number** → Provides numerical representation capability for numbers.
- **orderBy** → Provides sorting/ordering over data set for tabular/list data.
- **uppercase** → Represents a string in upper case format.

Note : Filters are added to expressions by using the pipe character '|' followed by a one of the filter name.

Ex :

```
<p> The first name is {{ firstName | uppercase }}</p>
```

Filter : currency → {{number | **currency** : symbol : fraction-size}}

Description

symbol	→	specifies currency symbol for different localization. Optional By default, the locale currency format is used
fraction-size	→	controls number of digits after decimal fraction. Optional

Example :

```
<div ng-app="myApp" ng-init="price:5000.4589">

    <p>Price = {{ price | currency : "INR Rs. " : 3 }}</p>
</div>
```

Output → Price = INR Rs. 5,000.459 try it

Filter : date → {{date-instance | **date** : format : timezone}}

Date input can be derived from date object, milliseconds or simple string representing data&time like “2016-01-05T09:05:05.035Z”

Description :

format	→	specifies date format to be shown. Optional By default, it uses 'MMM d,y' format uses standard format specifiers of date representation string.
timezone	→	provides UTC for universal time representation. Optional

Example :

```
<p>  Date = {{ "2016-01-05T01:00:00.000Z" |
                    date : 'dd-MMM-yyyy HH:mm:ss:sss' : '+0100' }}     </p>
```

Output → Date = 05-Jan-2016 02:00:00:000 try it

Filter : filter →{{ arrayexp | **filter** : expression : comparator }}

The filter allows us to search particular items in an array, and return subset of an array containing only the matching items form original array.

Note : Filter can only be used for arrays.

Description :

arrayexp	→	this arrayexp is collection of elements or object. filter will be applied only on array elements.
expression	→	expression is string/object that contains matching value. can be used against more than one matching as object{k→v}.
comparator	→	check for case-sensitive filter if it is true. or you can apply your own addition filtering checks/logic using functions, return value from function must be boolean.

Filter Example :

```
<div ng-app="myApp"
     ng-init="cars = ['Aston Martin', 'Audi', 'Bentley', 'BMW', 'Bugatti']" >

     <ul>
          <li ng-repeat="x in cars | filter : 'A'">{{x}}</li>
     </ul>
</div>
```

try it

```
<ul>
     <li ng-repeat="x in customers | filter : {'name' : 'O', 'city' : 'London'} : false">
          {{x.name + ", " + x.city}}
     </li>
</ul>

$scope.customers = [{"name" : "Alfreds Futterkiste", "city" : "Berlin"},
                    {"name" : "Around the Horn", "city" : "London"}….];
```

try it

Filter : json → {{ object | **json** : spacing }}

Description

object → it may be JavaScript any object or JSON object.

spacing → inserts spacing indent to left per JSON attribute. Optional default indent is 2 per JSON attribute.

Example :

```
<div ng-app="myApp" ng-init="cars=['Audi', 'BMW','Ford']">
        <pre>{{cars | json}}</pre>
</div>
```

Output →

```
[
  "Audi",
  "BMW",
  "Ford"
]
```

try it

Filter : limitTo → {{ object | **limitTo** : limit : begin }}

The limitTo filter can effectively used for navigation purpose over tabular record, and also used for sub-string functions in string and numbers representation.

Description

limit → restricts number of records/elements to be displayed.

begin → it tells staring index in records/elements to displayed. Optional default index is 0.

Usage

Array → returns subset of array elements that are in mention limit.
String → returns sub-string containing only specified number character.
Number → returns the string of digits containing only specified number of digits.

When negative value is used for limit/begin, then it return elements starting from end of the array/string instead of beginning.

Filter : number → {{ string | **number** : fractionsize}}

The **number** filter formats a number to a string as numerical representation.

Description

fractionsize → defines number of digits to be used after decimal point.

Example :

```
<div ng-app="myApp" ng-init="weight = 9999;">

        <h1> weight  is {{weight | number : 3}} kg</h1>
</div>
```

Output → **weight is 9,999.000 kg**

try it

Filter : orderBy → {{ array | **orderBy** : expression : reverse }}

The **orderBy** filter allows us to sort an array. By default, strings are sorted alphabetically and numbers are sorted numerically.

Description

try it

expression → determines the property name and order type in object list.
String → if it is array of objects, you can sort by values of one of object properties.
If you prepend '-' sign, it sorts in descending order.

Function → you can use function to create organized sorting.

reverse → manually we can specify here sorting order.
[true/false] → [ASC/DESC].

Filter :

lowercase → {{ string | **lowercase** }}
uppercase → {{ string | **uppercase**}}

These two are string manipulation filters.

Description

lowercase	→	filter converts the string to lowercase letters.
uppercase	→	filter converts the string to uppercase letters.

Custom Filters :

The custom filters allows to define our own filtering logic over a data set and formats the data representation style as well.
Custom filters are defined using **.filter()** component and registering that to application module, it will be available to application context.

Syntax

```
angular.module('myApp', []).filter('customFilter',  function(){

    return function(filterDefaultParam,  param1,  param2,  param3, ….){
        return string/object;
    }
});
```

Note : If custom filter is applied on simple data or single object type means it should return simple data type as result of filtering.

If custom filter is applied on array means, it should return new array consisting of old array's resulting elements, not simple data type.

Custom Filters :

Example

try it

```
<h1> {{ "Hello World"  |  equalsTo : "Dead World"  : "OK" : "OMG"  }} </h1>

<script>
      var app = angular.module('myApp', []);
      app.filter('equalsTo',  function(){

            return function(defaultParam,  s1,  s2,  s3){
                  if(angular.equals(defaultParam, s1))
                        return s2;
                  else
                        return s3;
            }
      });
</script>
```

HTML Form Elements :

Forms in AngularJS provides data-binding and validation for input controls and enriches form filling and validation process.

Input controls provides data-binding using ng-model directive.

Input Controls are in HTML

Input elements	→	text, password, email, date, number etc
Select elements	→	select, checkbox, radio
Button elements	→	button, submit, reset
Textarea elements	→	textarea only

NOTE : All input controls have same behavior when using with AngularJs **ng-module** directive, but checkbox have different behavior when dealing with multiple checkbox with same property name.

HTML Form Elements Validation :

try it

Offers client-side form validation.
Monitors state of the forms and input fields (input, textarea, select),
Informs the user about current status of input elements.
Tracks the information like valid, invalid, dirty, touched or not yet and submitted.
All these are properties of form/input element, either true/false.

Input Fields States :

$untouched	→	not touched.
$touched	→	touched.
$pristine	→	not modified.
$dirty	→	modified.
$invalid	→	not valid.
$valid	→	valid.

Form States :

$pristine	→	no fields are modified.
$dirty	→	one or more fields are modified.
$invalid	→	form content is not valid.
$valid	→	form content is valid.
$submitted	→	form is submitted.

Using these status flag we can effectively validate form properties at client side.
We can also make use of CSS on validation failure depending on status flags.

HTML Form Elements Validation : handling checkbox & radio

Check Box : As compare to other input controls, checkbox is different in nature of input elements operation and when controlled using JavaScript.

Problem : Check-box represents multiple value when used in HTML, while accessing it is treated as Arrray of elements of type String in JavaScript. But AngularJs bounds each HTML elements to single property in controller, where as checkbox is Array of String.

Solution : To solve this make use of two objects. **try it**

1 → To hold the all active and inactive list of checkbox values.

2 → To hold only active list of checkbox values.

Finally refer the active object to get selected values from list of checkbox.

Radio : Handling of radio buttons is more simpler in AngularJs than core JavaScript. Even also it represents multiple selection options, finally it returns single selected value.

Good News : No need of any manual operation like in JavaScript for radio buttons.

Services and Factories :

Services and factories are just a function for business layer of application.
Services and factories are both same in their functionality, only differs in the way of defining them.
Both are designed using singleton pattern, only one instance in available through-out the application.

Services

```
angular.module('myApp', [])
.service('myServices', function(){

      this.someFunction = function(){

      };
});
```

Factories

```
angular.module('myApp', [])
.factory('myFactory', funciton(){
      return {
            getMethod : function()      {

            };
      };
});
```

Services and Factories Examples:

Services

```
angular.module('myApp', [])
.service('myServices', function(){

      this.getAllUsers = function(){
            …..
            return usersList;
      };
      this.addUser(name, phno){
            var user = {name:name,
                  PhNo:phno};
            userList.push(user);
      };
      this.getUser(index){
            userList[index];
      };
});
```

Factories

```
angular.module('myApp', [])
.factory('myFactory', funciton(){
      var factory = {};
      factory.getAllUsers = function(){
            …..
            return usersList;
      };
      factory.addUser(name, phno){
      var user = {name:name, PhNo:phno};
            userList.push(user);
      };
      factory.getUser(index){
            userList[index];
      };
      return factory;
});
```

Services and Factories :

Differences between Services & Factories

Services

- Returns the instance of function(Object)
- Bounds the properties to the object.
- Invoked using new keyword as constructor.
- Services uses class object concept.
- Only object can be returned.
- No return statement required.

Factories

- Returns the function itself.
- Bounds the properties to function as key → value pair.
- Invoked like a simple function call.
- Factories are build on standards of factory methods concept.
- Any thing can be returned from factory method.
- Return statement is required.