AngularJs

What is AngularJs?

AngularJs is a front end frameworks for building structured and dynamic applications.It interanally uses javascript.

Why Angularjs?

Lets say we want to develop a single page application.

We know that in a single page application, the DOM load should happen only once. The subsequent server side requests should be XHR requests and modifying DOM using JavaScript .To develop a big application which uses JavaScript extensively we need to have an application structure and support for basic features like sending REST based requests, managing templates, caching,security, and optimization.

To solve our design problem, we cant use jquery because it just provides us utility functions for generic tasks, But how about the following

- application structure
- managing the continuously javascript code
- sending crud REST requests,
- binding to data
- Templating!
- Testing
- Browser History

To address the above requirements we need a framework and Angularjs solves more that just the problems mentioned above

Angularjs Features

Following are the features which make AngularJs an ideal framework for developing web applications

1) Providing structure to develop front end web applications, Angularjs has inbuilt support for modules,controllers, factories, and services to modularize the code

2)Inbuilt support for Ajax and REST to communicate to the server.

3)Two way binding to data and view. This makes angular a great choice for making interactive UI components quickly

4)Dependency Injection to inject the object required within the application without again instantiating.

5)HTML compilation and inbuilt templating. Forget the days when we had to take the headache of template compilation etc.

6)Fantastic support for Unit Testing. Angularjs Comes up with fantastic support for unit testing. The structure of angularjs application makes it very easy to test.

Getting started with AngularJs

Download angularjs from https://angularjs.org/.

Note that you can also use package management tools like bower to set up angularjs from command line.

Once downloaded, just include angularjs within the script tag. Then we need to decide the boundary of our application. That is done by simply adding an attribute ng-app to any tag. That all! Angularjs will operate within the starting and ending of that tag that contains ng-app.

Now to test it, just write {{1+5}} and angularjs will evaluate the expression and show the output.

Getting started with AngularJs

Download angularjs from https://angularjs.org/.

Note that you can also use package management tools like bower to set up angularjs from command line.

Once downloaded, just include angularjs within the script tag. Then we need to decide the boundary of our application. That is done by simply adding an attribute ng-app to any tag. That all! Angularjs will operate within the starting and ending of that tag that contains ng-app.

Now to test it, just write {{1+5}} and angularjs will evaluate the expression and show the output.

What are angularjs Expressions

AmgularJs Expressions are similar to javascript expressions that get evaluated within double curly braces.

Ex: {{1+8}

we can also write variable, objects and arrays inside like {{{employee.name}} or {{product['name']}}

We can write javascript expression within these double curly braces and angularjs evaluates them.

Ex:

```
<div ng-app>
<p>The addition of 1 and 5 is {{1+5}}</p>
<p >The multiplication of 1 and 5 is {{1+5}}</p>
<p >The division of of 12 b 3  is {{12/3}}</p>
```

Displaying object values within expression

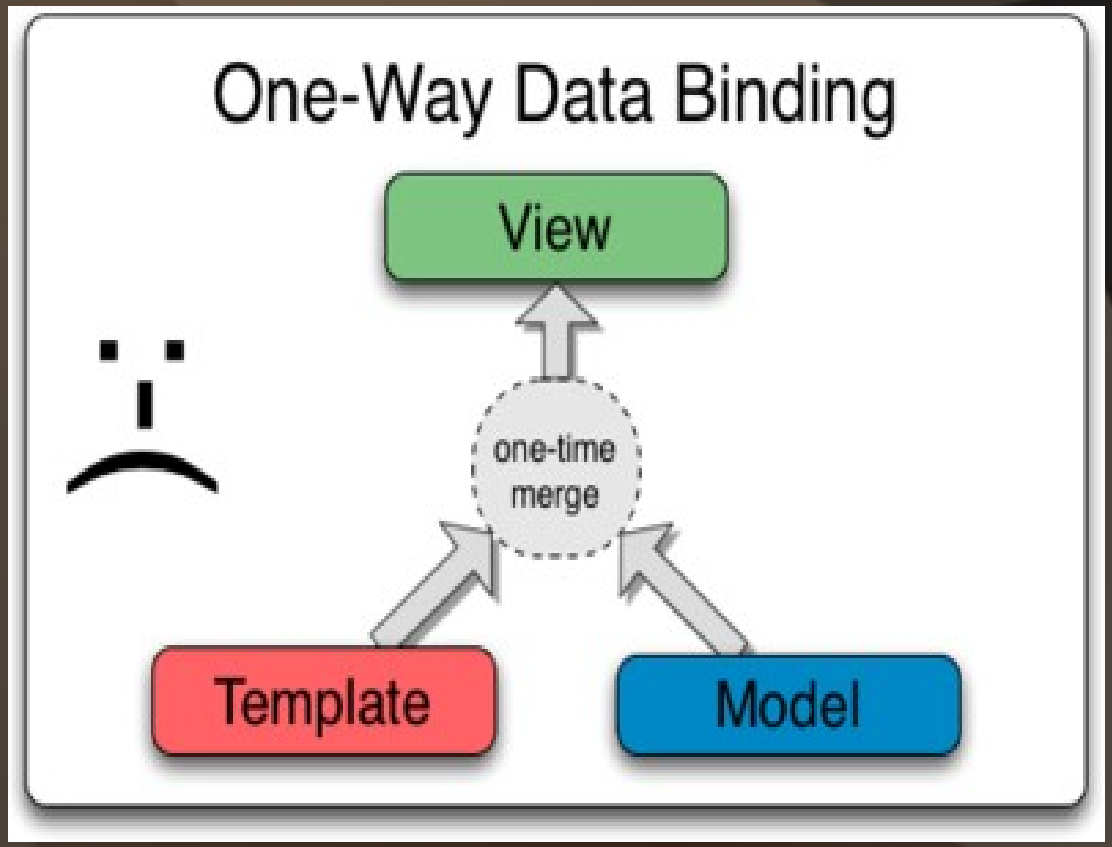
To display an object value within anuglarjs Expression we need to initialize the value .we will be using a custom angularjs attribute called ng-init for this. Have a look at the example below

Refer to ng-init.html

```
<div ng-app>

<div ng-init = "employee = {name:'smith',age:27}">

<p> The employee name and age are {{employee.name}} and
    {{employee.age}}

</div>
```

Data binding

Data binding is one of the most useful features in angularjs.
Usual templating system have one way binding.
One way binding means that data is bound to the view once during rendering. Next time if there is any change in the data, we have to again refresh the view to reflect the data change.Lets have a look a the on way binding in the following diagram.



Two way data binding in Angularjs

Angularjs has two way data binding. Means the data and the template view are in sync every time. Any changes to the data instantly changes the view and vice versa.

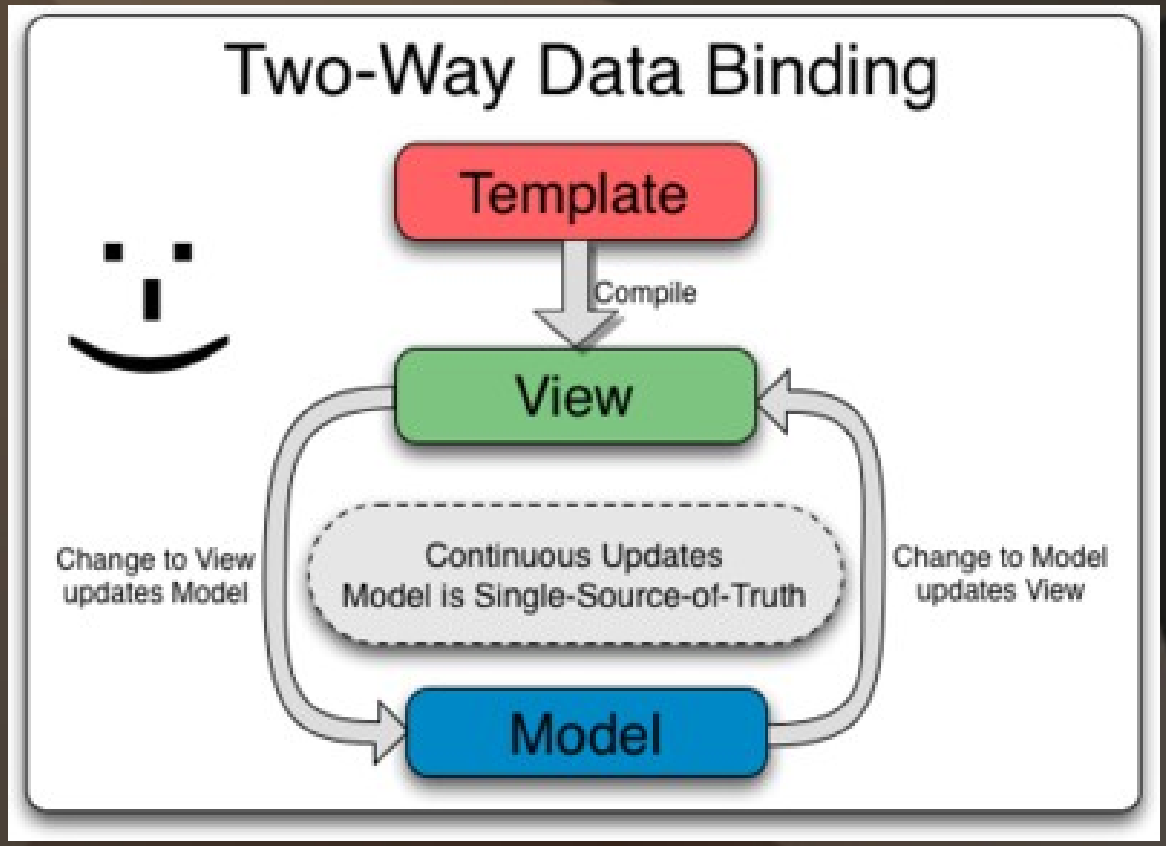
The mechanism for this feature is as follows:

First the template which is the uncompiled HTML is compiled on the browser. This compilation step produces a live view. Afterwards any changes to the view are immediately reflected in the model, and any changes in the model are propagated to the view.

AngularJs everytime keeps an eye and for any model change, it runs the cycle of updating the view. This process is called dirty checking.

Next we will see the pictorial representation of two way Binding.

The Two way data binding feature of anuglarjs keeps the data(model) in sync with the view.



Two way data binding example

We will be covering model later. For now lets understand that we have a directive called ng-model which makes an input control to represent a model. For our example our model is a simple text input. Any changes we make the value inside the input box will get reflected within the view which is angularjs expression. Below is the example

Refer to binding.html

```
<!doctype html>
<html>
<head>
</head>
<body>
<!-- initialzing the app -->
<div ng-app>
<!-- storing the value of input field into a variable name -->
<p>Name: <input type="text" ng-model="name"></p>
<p>{{name}}</p>
</div>
```

Understanding AngularJS Modules

A module is a collection of code for an independent piece of functionality .In angularjs we use module as a container for the different parts of our app like controllers,directives,services etc. Angularjs recommends us to organize our code into modules. Each module can have its own parts of code. Consider the example of an ecommerce application. We can divide the application into independent set of functionality like payment,cart,product,user and all of these can represent a module. This implies that we can create a payment module,user module and so on.

So how can we create a module?

Simple, just call the angular.module function as shown below

```
Var userModuleReference = angular.module(“userModule”,[]);
```

Note that we have got a reference returned of the module instance in the variable userModuleReference. The second parameter to the .module function is a comma separated value of modules required for the userModule. In this example the user module doesnt depend on any module so we can write the empty square bracket there.

Understanding The main Module

If our application is a collection of many modules, then we also need a main module from where we will initialize our application. Its just like the main method of Java, or c,c++.

To initialize our main module, we need to give the value of ng-app attribute with the value of the main module. Lets say our main module is myApp.

We can give a reference to myApp module in . This is what bootstraps the app using our module.

Retrieving the module

If any part of script we want to retrieve a particular module, we need to call the module method without the module dependencies array.

The code var myModule = angular.module('myModule',[]) creates the module and returns the reference. But var myModule = angular.module('myModule') returns the reference of already created module.

Note that if the module is not created before, the the above line with throw error.

Controllers

Controller is the function that adds variables and functionality to the scope of a view.We can think of controller as a function that contains the business logic of the application. Technically a controller is a JavaScript constructor function.

Lets have a look at the example of a basic controller.

Refer to basic_controller.html

```
<script type="text/javascript">
var myModule = anguslar.module('myModule', []);
myModule.controller('MyCtrl', function ($scope) {
$scope.location = "New York";
$scope.name = "smith";
});
</script>
</head>
<body>
<div ng-app="myModule"
<div ng-controller="MyCtrl">
<p>{{"Name is "+ name + " and location is " + location}}</p>
</div> </div>
```

Understanding the example

Lets now look at the important points of the previous example.

First that we have created a module and also a controller. Note that we define the controller by calling the controller function which is a property of the module object. The format is :

Modulereference.controller(<controller-name>,<controller-function>).

Next we pass a $scope object to the controller function. Note that $scope is an inbuilt angularjs object that will contain the environment of the controller.

We can add any variables or even functions as properties to this $scope object. All the properties and functions added to the $scope object are automatically available within the element that has ng-controller value as value of the controller function name.

Note that controller defines the body of the environment for the view.

To add a controller within the boundary of a DOM element, just add ng-controller property with value as the controller name defined within the script. When angularjs parses the entire html. It instantiates the controller object and defines the scope within the DOM element boundry,,,ie starting and ending of the DOM element.

Note that we have attached two variables name and location to the $scope and they become available within the div that has ng-contoller = “MyCtrl”.

Attaching functions to controller scope

As we have seen that we can attach variables to the controller. Similarly we can also attach functions to the controller. Lets now create a simple example to demonstrate how can achieve it.

We will be using a special directive called ng-click which can be added to any DOM element to have a click property.say you want to assign a click property to a button, just add <button ng-click = “displayMsg()”>click</button>. The question is where we need to define the displayMsg function. Well we can always attach it to the $scope variable within the defined controller for the view. Below is the example

Refer to controllerwithfunction.html

```
<script type="text/javascript">
var myModule = angular.module('myModule', []);
myModule.controller('MyCtrl', function ($scope) {
$scope.click = function(msg) {
alert(msg);
};
});
</script>
</head>
<body>
<div ng-app="myModule"
<div ng-controller="MyCtrl">
<button ng-click="click('hello how r u')">Show Message</button>
</div>
</div>
```

Controller with function example

Lets have a look at one more example. Here we are directly assigning an object to the scope and having an attached function to display the value of the variable within the object.

Refer to controllerwithfunction1.html

```
<script type="text/javascript">
var myModule = angular.module('myModule', []);
myModule.controller('MyCtrl', function ($scope) {
var dataOb = { "firstname": "Jack",
"lastname": "Smith" };
$scope.name = dataOb;
$scope.click = function() {
alert('The first names is '+$scope.name.firstname);
};
});
</script>
</head>
<body>
<div ng-app="myModule"
<div ng-controller="MyCtrl">
<p>{{name.firstname + " " + name.lastname}}</p>
<button ng-click="click()">Show name</button>
</div>
</div>
```

Using ng-model and data binding

The ngModel directive binds an input,select, textarea or custom form control to a property on the scope using controller.This is where user can enter data and it is automatically updated in the view.

The ng-model directive gives us the flexibility of binding a data to the user input.

ngModel will bind the property given by evaluating the expression on the current scope. If the property doesn't already exist on this scope, it will be created implicitly and added to the scope.

lets have a look at the example in the next slide.

Using ng-model and data binding

Note that the scope automatically gets created using the ng-model directive.

Also see the live binding. Try entering some value in the text boxes and you can see the input data in the view changed as you type in

Refer to ng-model_example.html

.

```
<script type="text/javascript">
var myModule = angular.module('myModule', []);
myModule.controller('MyCtrl', function ($scope) {
$scope.click = function() {
alert("first name is "+$scope.firstname+"and last name is "+$scope.lastname);
};
});
</script>
</head>
<body>
<div ng-app="myModule"
<div ng-controller="MyCtrl">
<p>Firstname: <input type="text" ng-model="firstname"></p>
<p>Lastname: <input type="text" ng-model="lastname"></p>
<p>{{firstname + " " + lastname}}</p>
<button ng-click="click()">Show Name</button>
</div>
</div>
```

Understanding directives

Directives are the most powerful feature of angularjs. Angularjs comes up with inbuilt attributes that can be directly added to the html elements to add behaviour and functionality.

Directives can be inbuilt means they come with behaviours and properties provided by angularjs or custom means we can create our own directives.

Example of inbuilt directives is the ng-app to declare the main application.

We will be learning about custom directives later. In next slide, we will see a list of common directives which we will be using often in our angularjs apps.

Inbuilt Directives

Follwing are the commonly used directives:

ng-click : To attach a click event.

ng-change: Fires a function when user changes input

ng-init: to initialize data

ng-show: shows or hides the given HTML element based on the expression provided to the ngHide attribute

ng-href : to point a url for a link

ng-include: This directive fetches, compiles and includes an external HTML snippet.

ng-model: to bind a form input .

ng-repeat:This directive is used to repeat the html for a particular collection.Useful to show an array items in a list etc.

ng-selected:If the expression value is true, then special attribute "selected" will be set on the specified element.

ng-show:It shows or hides the given HTML element based on the expression provided to the ng-show attribute.

Understanding Scopes

Now that we know how to create a controller and attach a scope lets understand more about scopes.

Every part of an AngularJS application has a parent scope which is within the ng-app boundry.

This scope is called the $rootScope. All scopes have prototypal inheritance, meaning that they have access to their parent scopes.

so what happens if a property is not find within the controller scope?

In this case, AngularJS will crawl

up to the parent scope and look for the property or method there. If AngularJS it can’t find the property there, it will walk to that scope’s parent and so on till it reaches the $rootScope. Lets look at an example to understand scopes further.

$rootscope

Every module has a single root scope. All other scopes are descendant scopes of the root scope. Scopes provide separation between the model and the view.

Note that $rootscope may be different for different modules.

The data set in $rootscope is available to all the controllers within the application.

Ok so the question is ..how do we set the value in a root scope? Yes,,we can definitely set the value within a controller, But that will only effect the rooscope value for that particular controller. The changed value will not reflect in other controllers!

Here comes the run function of a module to our rescue. Whenever the module loads, it runs the run function before loading any controllers and is the ideal place to set the values within rootscope.

See the example snippet in the next slide

$rootscope

Refer to rootscope.html

```
<script type="text/javascript">
var myModule = angular.module('myModule', []);
myModule.run(function($rootScope){
$rootScope.appName = "Testing Root Scope";
})
myModule.controller('DataCtrlOne', function ($scope) {
$scope.firstname = "John";
$scope.lastname = "Smith";
});
myModule.controller('DataCtrlTwo', function ($scope) {
$scope.firstname = "Luke";
$scope.lastname = "Johnson";
});
</script>
</head>
<body>
<div ng-app="myModule">
<p>{{"The application name is"+appName}}</p>
<div ng-controller="DataCtrlOne">
<p>Name is {{firstname + " " + lastname}}</p>
<p>The Modified application name is {{appName}}</p>
</div>
<div ng-controller="DataCtrlTwo">
<p>Name is {{firstname + " " + lastname}}</p>
<p>The application name is {{+appName}}</p>
</div> </div>
```

Scopes Hierarchy

What happens if we keep one controller within another controller?

The controller that has other controller inside becomes the parent controller. Now the child controller will inherit all the properties of the parent controller.

Any properties of the parent controller can be over written.

The example in the next slide creates a parent controller and child controller. Note that whatever controller we keep inside the parent controller element becomes the child controller.

Here we have kept child controller wtihin parent controller in view to demonstrate how inhertiance works. Note that when child controller is present inside parent controller it is able to access the property firstname.

We have also kept child controller in view outside the main controller and in this case the child doesnt inherit the firstname property.

Have a look at example controller_inhertiance.html

Scopes Hierarchy

Refer to controller_inhertiance.html

```
<script type="text/javascript">
var myModule = angular.module('myModule', []);
myModule.controller('MyCtrl', function ($scope) {
var model = { "firstname": "Jack",
"lastname": "Smith" };
$scope.model = model;
});
myModule.controller('ChildCtrl', function ($scope) {
var modelchild = { "hobby": "Tourism",
"interests": "coding" };
$scope.company = "Google";
$scope.modelchild      = modelchild;
});
</script></head>
<body>
<div id = "boundry" ng-app="myModule">
<div id = "parent" ng-controller="MyCtrl">
<p>Firstname: <input type="text" ng-model="model.firstname"></p>
<p>Lastname: <input type="text" ng-model="model.lastname"></p>
<p>{{model.firstname + " " + model.lastname}}</p>
your hobby is {{modelchild.hobby}}<div id = "child" ng-controller = "ChildCtrl">
  Your hobby is {{modelchild.hobby}}        your first name is {{model.firstname}}
</div>
</div>
<div id = "indepndent" ng-controller = "ChildCtrl">
Your hobby is {{modelchild.hobby}} your first name is {{model.firstname}}
</div></div>
```

Using $parent keyword

Angular scopes include a variable called $parent that refers to the parent scope of a controller. If a controller is at the root of the application, then parent would be the root scope..ie $rootscope

There are instances when we have overwritten the property of the parent controller within child controller and we still want to access or modify it. In this case we can use the $parent keyword.

Lets have a look at this with an example..refer to keyword_$parent.html

```
<script type="text/javascript">
var myModule = angular.module('myModule', []);
myModule.controller('MyCtrl', function ($scope) {
$scope.name = "Smith";
});
myModule.controller('ChildCtrl', function ($scope) {
$scope.name = "John";
  });
</script></head>
<body>
<div id = "boundry" ng-app="myModule">
<div id = "parent" ng-controller="MyCtrl">
<div id = "child" ng-controller = "ChildCtrl">
  This is name property of child {{name}}
  This is name property of parent {{$parent.name}}
</div></div></div>
```

Using this operator within controller

We have seen that we can $scope injectibile to give access to the data within the html. Though this works fine, we can also use the this operator of the controller instance to achieve the same.

The beneift of using this operator is that we dont need to use the $scope to attach data to the scope. Lets see how it works with an example in the next slide

Example Using this operator within controller

Refer to controller_this.html

```
<script src="angular.js" type="text/javascript">
</script>
<script type="text/javascript">
var myModule = angular.module('myModule', []);

myModule.controller('MyCtrl', function () {

var model = { "firstname": "Jack",
"lastname": "Smith" };

this.model = model;
this.click = function() {
alert($scope.model.firstname);
};
});
</script>
</head>
<body>
<div ng-app="myModule">
<div ng-controller="MyCtrl as ctrlRef">
<p>Firstname: <input type="text" ng-model="ctrlRef.model.firstname"></p>
<p>Lastname: <input type="text" ng-model="ctrlRef.model.lastname"></p>
<p>{{parent.model.firstname + " " + parent.model.lastname}}</p>
<button ng-click = "ctrlRef.click()">Click</button>
</div>
</body>
```

Using this operator for nested controllers

The benefit of using this notation shines when using nested controllers. Note that can refer to the parent and child using the alias we define within the html.

Lets see the example in the next section where we have parent controller and child controller. Note that parent and child is defined by which controller is nested in the html.

Using this for nested controllers

Refer to controller_this_inheritance.html

```
<script type="text/javascript">
var myModule = angular.module('myModule', []);
myModule.controller('MyCtrl', function () {
var model = { "firstname": "Jack",
"lastname": "Smith" };
this.model = model;
this.click = function() {
alert($scope.model.firstname);
};
});
myModule.controller('ChildCtrl', function () {
var modelchild = { "hobby": "Tourism",
"interests": "coding" };
this.company = "Google";
this.modelchild     = modelchild;
this.click = function() {
alert(this.modelchild.hobby);
};
});
</script>
</head>
<body>
<div id = "boundry" ng-app="myModule">
<div id = "indepndent" ng-controller = "ChildCtrl as indp">
Your hobby is {{indp.modelchild.hobby}}
your first name is {{indp.model.firstname}}
</div>
<div id = "parent" ng-controller="MyCtrl as parent">
<p>Firstname: <input type="text" ng-model="parent.model.firstname"></p>
<p>Lastname: <input type="text" ng-model="parent.model.lastname"></p>
<p>{{parent.model.firstname + " " + parent.model.lastname}}</p>
your hobby is {{child.modelchild.hobby}}
<div id = "child" ng-controller = "ChildCtrl as child">
Your hobby is {{child.modelchild.hobby}}
your first name is {{parent.model.firstname}}
</div>
<button ng-click="child.click()">Show Name</button>
</div>
</div>
```

Inbuilt filters in Angularjs

Below is the list of inbuilt filters

- Filter
- Currency
- Number
- Date
- Json
- Lowercase
- Uppercase
- LimitTo
- Orderby

Filters

Filters in Angularjs provide a way to format the data we display to the user.There are several built-in filters available in angularjs. we can as well as an easy way to create our own.

How to use a filter. To use a filter we simply need to put a pipe after the text or data we want to format followed by the filter name.

filter_expression is the expression. After filter we can optionally specigy

One inbuilt filter is filter with name uppercase. This filter as the name shows converts the lowercase letters to the uppercase. Belw is the usage to use it

{{“smith” |uppercase}} . Lets see a simple usage of a filter

Refe to basic_uppercase_filter.html

```
<script type="text/javascript">
var myModule = angular.module('myModule', []);
</script>
</head>
<body>
<div ng-app="myModule"
<p>Name in uppercase is {{"smith"|uppercase}}</p>
 </div>
```

Note that we can pipe in filters. Means we can apply more than one filter. Just add the pipe and call the filter

Using filter within Javascript

Filters can also be used within the javascript code. The format is
$filter(<filter-name>')(<expression>)
Below is the example for using it

```
<script type="text/javascript">
var myModule = angular.module('myModule', []);
myModule.controller('MyCtrl', function ($scope,$filter) {
$scope.name = $filter('uppercase')("smith");
});
</script>
</head>
<body>
<div ng-app="myModule"
<div ng-controller="MyCtrl">
<p>Name in uppercase is {{name}}</p>
</div>
 </div>
```

Passing Arguments to Filters using currency filter

For arguments, we can pass them with a colon after the filter name and for multiple arguments, we append a colon after each argument.
Lets understand this using a currency filter
This filter Formats a number as a currency

{{ currency_expression | currency : symbol : fractionSize}}

Below is the code(filename is currency_filter.html)

```
<script>
  angular.module('myModule', [])
    .controller('ExampleController', ['$scope','$filter', function($scope,$filter) {
      $scope.amount = 1234.565555;
    }]);
</script>
  </head>
  <body ng-app = 'myModule'>
 <div ng-controller="ExampleController">
  <input type="number" ng-model="amount"> <br>
  default currency symbol ($): <span id="currency-default">{{amount | currency }}</span><br>
  custom currency identifier (USD$): <span id="currency-custom">{{amount |
     currency:"USD$":3}}</span>
</div>
```

Using Number Filter

The number filter is used to filter numbers.

This number filter Formats a number as text.

If input is not a number an empty string is returned.

If input is an infinite then the Infinity symbol '∞' is returned

Below is the syntax

{{ number_expression | number : fractionSize}}

Below is the example(number.html).

```
<script>
  angular.module('myModule', [])
    .controller('NumberController', ['$scope', function($scope) {
      $scope.val = 2345.978777899;
    }]);
</script>
  </head>
  <body ng-app = 'myModule'>
 <div ng-controller="NumberController">
  Enter number: <input ng-model='val'><br>
  THis is Default formatting: <span>{{val | number}}</span><br>
  Displaying number with No fractions: <span>{{val | number:0}}</span><br>
  Displaying a Negative number: <span>{{-val | number:2}}</span>
</div>
```

Using Date Filter

The date filter formats date to a string based on the requested format.

Template format

{{ date_expression | date : format : timezone}}

Note that data can be either an ISO date object or date string or timestamp

Ex(refer to file date_filter.html):

```
<span>{{1288323623006 | date:'yyyy-MM-dd HH:mm:ss Z'}}</span><br>
```

Using Json Filter

The Json filter allows to convert a JavaScript object into JSON string and is mostly useful for debugging

Ex:

{{ json_expression | json : spacing}}

Note that spacing option is the number of spaces to use per indentation and defaults to 2

Refer to example json_filter.html

```
<script>
 angular.module('myModule', [])
    .controller('JsonController', ['$scope', function($scope) {
      $scope.val = {
       "name":"smith",
       "age":24
      };
    }]);
</script>
  </head>
  <body ng-app = 'myModule'>
 <div ng-controller="JsonController">
{{val|json}}
</div>
```

Limit filter

This filter creates a new array or string containing only a specified number of elements as mentioned in the limitto filter. If we use a number as input, it is converted to a string.

Syntax:

{{ limitTo_expression | limitTo : limit : begin}} or

$filter('limitTo')(input, limit, begin)

Below is the example(refer to number_limit.html)

```
<script>
  angular.module('mainModule', [])
    .controller('LimitController', ['$scope', function($scope) {
      $scope.numbers = [1,2,3,4,5,6,7,8,9,12,14];
      $scope.numLimit = 2;
    }]);
</script>
  </head>
  <body ng-app = 'mainModule'>
 <div ng-controller="LimitController">
  Limit {{numbers}} to: <input type="number" step="1" ng-model="numLimit">
  <p>Output numbers: {{ numbers | limitTo:numLimit }}</p>
</div>
```

Limit to for strings

We can use limit to filter for strings too. Below is the example showing the different usages.

Refer to file filter_limitto_usages.html

```
<script>
  angular.module('mainModule', [])
    .controller('LimitController', ['$scope', function($scope) {
      $scope.numbers = [1,2,3,4,5,6,7,8,9,12,14];
      $scope.letters = "lets see how many characters are displayed with limit";
      $scope.longNumber = 1232345432342;
      $scope.numLimit = 4;
      $scope.letterLimit = 5;
      $scope.longNumberLimit = 5;
    }]);
</script>
  </head>
  <body ng-app = 'mainModule'>
 <div ng-controller="LimitController">
  Limit {{numbers}} to: <input type="number" step="2" ng-model="numLimit">
  <p>Output numbers: {{ numbers | limitTo:numLimit }}</p>
  Limit {{letters}} to: <input type="number" step="1" ng-model="letterLimit">
  <p>Output letters: {{ letters | limitTo:letterLimit }}</p>
  Limit {{longNumber}} to: <input type="number" step="1" ng-
     model="longNumberLimit">
  <p>Output long number: {{ longNumber | limitTo:longNumberLimit }}</p>
</div>
```

orderBy filter

This filter orders a specified array by the expression predicate.

The result is is ordered alphabetically for strings and numerically for numbers.

To order numbers property ensure that numbers are actually being saved as numbers and not strings.

Below is the syntax

{{ orderBy_expression | orderBy : expression : reverse}}

See the example code below(refer to orderby_basic_filter.html)

```
<script>
  angular.module('orderByApp', [])
    .controller('OrderByController',function($scope) {
      $scope.friends =
          [{name:'Smith',phone:'555-1512', age:20},
           {name:'John', phone:'555-6676', age:29},
           {name:'Ajay', phone:'555-6621', age:31},
           ];
    });
</script>
<div ng-app = "orderByApp">
<div ng-controller="OrderByController">
  <table>
    <tr>
      <th>Name</th>
      <th>Phone Number</th>
      <th>Age</th>
    </tr>
    <tr ng-repeat="friend in friends | orderBy:'age'">
      <td>{{friend.name}}</td>   <td>{{friend.phone}}</td> <td>{{friend.age}}</td>
    </tr>
  </table>
</div>
```

Limit filter

This filter creates a new array or string containing only a specified number of elements as mentioned in the limitto filter. If we use a number as input, it is converted to a string.

Syntax:

{{ limitTo_expression | limitTo : limit : begin}} or

$filter('limitTo')(input, limit, begin)

Below is the example(refer to number_limit.html)

```
<script>
  angular.module('mainModule', [])
    .controller('LimitController', ['$scope', function($scope) {
      $scope.numbers = [1,2,3,4,5,6,7,8,9,12,14];
      $scope.numLimit = 2;
    }]);
</script>
  </head>
  <body ng-app = 'mainModule'>
 <div ng-controller="LimitController">
  Limit {{numbers}} to: <input type="number" step="1" ng-model="numLimit">
  <p>Output numbers: {{ numbers | limitTo:numLimit }}</p>
</div>
```

Using filter Filter

filter: This is a very important filter. It selects a subset of items from array and returns it as a new array. Below is the syntax

{{ filter_expression | filter : expression : comparator}}

Here filter_expression is the expression, expression parameter represents the reference data to which we have to filter and the comparator is the comparison function or value to decide if the value can be considered a match.

Lets see a simple example below

```
<script>
  var app = angular.module('searchModule',[]);
  </script>
 </head>
 <body ng-app = 'searchModule'>
 <div ng-init="friends = [{name:'John', phone:'555-1276'},
 {name:'Manish', phone:'800-1234'},
 {name:'Sumit', phone:'555-4321'},
 {name:'Aadi', phone:'555-5678'},
 {name:'Tyler', phone:'555-8765'},
 {name:'Bantu', phone:'555-5678'}]"></div>
 Search: <input ng-model="searchData">
 <table id="searchTextResults">
  <tr><th>Name</th><th>Phone</th></tr>
  <tr ng-repeat="friend in friends | filter:searchData">
    <td>{{friend.name}}</td>  <td>{{friend.phone}}</td>
  </tr>
 </table>
```

Understanding the filter more

Its important to note that in the previous example, the filtering happens also for objects and properties inside. IN the previous example, we can search for the phnone number and names both! This is a very strong feature of this filter.

What we do to only search names?just change the ng-model value as searchData.phone instead of searchData

How about making a strict search. Means that only if the full value is a match, the data is returned.

How to use? Simple use friendObj in friends | filter:searchData:true

Note that the last parameter can be an expression also!

Custom filters

We can also create custom filters to meet custom requirements.
Below is the syntax

```
ModuleRef.filter('Custom-Filter-Name',function(){
    return function(optional-parameter){
            return modifiedValue;

  }
});
```

Lets have a look at the example code. We are creating a filter named greet that adds a Good Morning to the string.(refer to basic_custom_filter.html).

```
<script type="text/javascript">
 var app = angular.module('myApp',[]);
  app.filter('greet',function(){
  return function(name){
   return 'Good Morning '+name;
 }
});
</script>
</head>
<body ng-app = "myApp">
<div ng-init=
"customers = [{name:'SMith'}, {name:'tiya'}, {name:'Luke'}, {name:'Mac'}]">
<h1>Customers</h1>
<ul>
<li data-ng-repeat="customer in customers">{{
customer.name |greet }}</li>
</ul>
</div>
```

Filtering Arrays

We can also create a custom filter to filter arrays. Lets have a look at the following example to see the implementation.

We are creating a filter named strln5 that filters elements with string length <=5
refer to filter_custom_example.html).

```
<script type="text/javascript">
var app = angular.module('myApp',[]);
app.filter('strln5',function(){
return function(items){
var filtered = [];
    for (var i = 0; i < items.length; i++) {
     var item = items[i];
      // check if the individual Array element is length <4
      if (item.name.length<=5) {
        filtered.push(item);
      }
    }
       return filtered;
  }
})
</script>
</head>
<body data-ng-app = "myApp">
<div data-ng-init=
"customers = [{name:'jack'}, {name:'tina'}, {name:'johnson'}, {name:'donald'}]">
<h1>Customers</h1>
<ul>
<li data-ng-repeat="customer in customers | strln5">{{customer.name }}</li>
</ul></div>
```

exploring config and run function for module

We have 2 functions namely config and run function for the module.

The config function is the first function that runs when the module loads. After the config function, the run function runs. See the console messages below to see the cycle of load.

Lets see this with the example below

```
<script>
angular
  .module('main', ['module1', 'module2'])
  .config(function () { console.log("came here config main module");})
  .run(function () { console.log("came here run main module"); })
  .controller('MainController',function(){
    console.log("inside MyController of main");
  });
angular
  .module('module1', ['module2'])
  .config(function () {console.log("came here config of module1"); })
  .run(function () { console.log("came here run of module1"); });
angular
  .module('module2', [])
  .config(function () { console.log("came here config of module2");})
  .run(function () { console.log("came here run of module2"); });
</script>
</head>
<body ng-app = "main">
<div ng-controller = "MainController">
</div>
```

Explaining the Example for config and run functions

Lets understand the previous example.

We have the main module which has 1 dependency namely module1.

The module1 again has dependency of module2.

Note that main module has module1 dependency. Again the module1 has module2 dependency. Therefore first the module2 config function will run followed by config function of module1 and finally the config function of main module will run.

Similarly the run function of the module2 followed by run function of module1 and then the main module's run function.

Finally the controller of the main function will run.

Note that we can only inject one dependency which is a provider.

Note that At this phase, we can not inject either services or factories because the dependency injector has not instantiated the services and factories.

We can initialize all the configuration parameters in the config function for a module.

At run phase we can inject all the dependencies because injectors have initialized and hence are available to be used.

Routing

We need routing in single page applications too. What is a route for anularjs Application?

Say we have a website made in angularjs. If it has 3 pages namely home,contactus and news, then we need to fire the appropriate controllers when the user navigates from home to contacus. We can do this by attaching a click even to the menu item also, but it is not recommended way. Why?

Because if we attach click to the menu items, then a user can not navigate to the other page using a url.

A real single page application also has routes.

Consider the home page url is mysite.com. To navigate to contactus we can make a url like mysite.com/#contactus which will naviate the page to the contact us. We call them as hashbang urls.

To implement routing we can use ngRoute module. Note that to use this module we need to download the module javascript file and add it to the page.

Next we have to include the ngRoute module in dependency array while declaring the module

Routing example

Refer to routing/routingexample.html.

```
<script>
   var module = angular.module("sampleApp", ['ngRoute']);
   module.config(['$routeProvider',
      function($routeProvider) {
         $routeProvider.
               when('/route1/:param', {
                  templateUrl: 'templates/template1.html',
                  controller: 'RouteController'
               }).
               when('/route2/:param', {
                  templateUrl: 'templates/template2.html',
                  controller: 'RouteController'
               }).
               otherwise({
                  redirectTo: '/',
                  templateUrl:'templates/index.html',
                  controller:'IndexController'
               });
      }]);

   module.controller("RouteController", function($scope, $routeParams) {
      $scope.param = $routeParams.param;
     // alert($scope.param);
   })
   .controller('IndexController',function($scope){
      $scope.message = "Welcome to home page";
   })
</script>
</head>
<body ng-app="sampleApp">
<a href="#/route1/abcd">Route 1</a><br/>
<a href="#/route2/1234">Route 2</a><br/>
<div ng-view></div>
</body>
```

Routing example explained

Few concepts to note here.

First that config is the function that runs first when the module loads. We can set up the configuration for our module within this function. Here we are injecting the $routeProvider and setting up the routes here.

Next every route has a controller associated with it which is fired when the user navigates to the route.Corresponding to the route we have the template which is loaded for the route. We use the ng-view directive here which is replaced by the template provided for routing.Below is the code taken from the example

```
when('/route2/:param', {
        templateUrl: 'templates/template2.html',
        controller: 'RouteController'
    })
```

The following code is fired when none of the routes above are found. We have loaded the index page data here.

```
otherwise({
    redirectTo: '/',
    templateUrl:'templates/index.html',
    controller:'IndexController'
});
```

Dependency Injection

Dependency injection is a design pattern where dependencies for a component are provided by the system on demand.

Consider the example of $scope variable which is made available whenever required by a controller. But the question is how is the variable automatically made available?

The answer is dependency injection where the Angular injector injects the $scope variable wherever it is required. We can inject dependencies at many places within our angular code.

We have already seen that there are many variables available to use whenever we require them. $scope is one such variable.

We can also inject the objects that we want to be made available using services which we will be covering in the next section.

The three ways to inject dependences are Services, Factories and Providers

Creating Services

We can create a singleton JavaScript object using a service. This object contains a set of functions where can put the logic and use it. Using services we can separate common logic and provide it to the controllers or other components. Refer to dependency_injection/service.html

```
<script>
var serviceExample = angular.module('serviceExample', []);
serviceExample.service('wordService', function() {
this.reverseWord = function(input) {
var result = "";
input = input || "";
for (var i=0; i<input.length; i++) {
result = input.charAt(i) + result;
}
return result;
}
this.capitalize = function(input){
return input.toUpperCase();
}
});
serviceExample.controller('MyCtrl' ,function ($scope, wordService) {
$scope.reversename = wordService.reverseWord($scope.name);
$scope.$watch('name',function(oldVal,newVal){
$scope.reversename = wordService.reverseWord($scope.name);
})
$scope.capitalizeName = function(){
$scope.reversename = wordService.capitalize($scope.reversename);
}
});
</script>
</head>
<body ng-app = "serviceExample">
<div ng-controller = "MyCtrl">
<input type = "text" ng-model = "name">
Name in reverse is: {{reversename}}
<button ng-click = "capitalizeName()">Capitalize</button>
</div>
```

Services Example explained

In previous example we created a service which has two function. One to reverse the word and other to covert into capital letters. These two functions can be made reusable by attaching them to a service as in the example below

Its important to understand that the dependency injector service of anuglarJs automatically instantiates the constructor service function and creates an object. That object can be injected within the controller. The name of the object is the same as the name of the service. Using this object we can now access all instance functions(ie functions referred using this) .

In the example we have used the $watch which watches the changes in the value of name. Upon change we call the function which reverses name by calling the function in the service.

Similarly we also call the function capitalize within our controller.

Remember that we should not keep all repeatable logic within our controllers. We should instead keep it in the services for better re usability and code separation.

Creating factories

We can also create a singleton object using a factory. Refer to dependency_injection/factory.html

```
script>
var factoryExample = angular.module('factoryExample', []);
factoryExample.factory('wordFactory', function() {
var literalObject = {
 reverseWord :function(input) {
var result = "";
input = input || "";
for (var i=0; i<input.length; i++) {
result = input.charAt(i) + result;
}
return result;
},
capitalize :function(input){
return input.toUpperCase(); }
}
return literalObject;
});
factoryExample.controller('MyCtrl' ,function ($scope, wordFactory) {
$scope.reversename = wordFactory.reverseWord($scope.name);
$scope.$watch('name',function(oldVal,newVal){
$scope.reversename = wordFactory.reverseWord($scope.name);
})
$scope.capitalizeName = function(){
$scope.reversename = wordFactory.capitalize($scope.reversename);
}
});
</script></head>
<body ng-app = "factoryExample">
<div ng-controller = "MyCtrl">
<input type = "text" ng-model = "name">
Name in reverse is: {{reversename}}
<button ng-click = "capitalizeName()">Capitalize</button></div>
```

Factory Example explained

A factory is similar to the service that it also provides a singleton JavaScript object using angular dependency injection.

The difference between a factory and service is that in a factory we can return our own custom object instead of the instantiated object of the function.

In the example we have created an object literal, added functions to it and returned it. Note that we did not attach functions to the function instance using this operator

The example logic is the same as of services example.

Creating Providers

We can also create providers. Providers are again singletons but are more configurable.

```
<script>
var providerExample = angular.module('providerExample', []);
providerExample.provider('word', function() {
    this.name = 'Default';
    this.$get = function() {
        var name = this.name;
        return {
            sayHello: function() {
                return "Hey..How r u.. " + name + "!"
            }
        }
    };
    this.setName = function(name) {
        this.name = name;
    };
});
//configuring the provider
providerExample.config(function(wordProvider){
    wordProvider.setName('Smith');
});
providerExample.controller('MyCtrl' ,function ($scope, word) {
$scope.displayMessage = function(){
alert(word.sayHello());
}
});
</script>
</head>
<body ng-app = "providerExample">
<div ng-controller = "MyCtrl">
<button ng-click = "displayMessage()">Display Message</button>
```

Provider Example explained

A provider is advanced and configurable. A provider is different from factories and services in the aspect that It can be configured at the configuration phase of the module in the config function.

The function within the this.$get is instantiated and returned via the injector. To demonstrate this we have created a function called sayHello which returns a message appended by name variable. This variable name is set in the setName function . Important point to note is that the setName function is not available to the singleton object. This is available to the injectible named wordProvider in the config function.

Note that we always have to add the Provider string to the name of the provider that we created to inject it at the config phase. In our case the name of the provider is word so the name of the object in config function is wordProvider. Notice that at the config phase we get the full instance of the function and hence we can call the setName function and set the name value as “Smith”.

If we don't call the setName function then the function sayHello will append the value “Default” instead of “Smith”.

Directives

We have already gone through directives. Lets again revise. A directive is a custom tag, attribute, class or comment that adds a behaviour to html.

Ex: Instead of creating entire HTML and source for a cart button, we can create a tag named <cart/> .

If you place <cart/> tag in your HTML source , you will get full functionalities of the cart.

Lets start with creating a simple directive to understand how we can create directives.

In the example below we create a custom directive tag called hello. When angular loads, it compiles the tag and displays “Hi There” message in a div. Refer to example directives/basic_directive.html.

```
<body>
   <hello></hello>
  <script>
   var appModule = angular.module('app', []);
   appModule.directive('hello', function() {
    return {
      restrict: 'E',
      template: '<div>Hi there</div>',
      replace: true
    };
   });
  </script>
  </body>
```

Creating a custom directive

Lets understand the previous example. To create a directive we call the function directive on the module like.Note that first parameter is the name of the directive and the second is the function that returns an object. This object contains all the settings of the directive. We call this object as directive definition object.

```
appModule.directive('hello',function(){
  return directiveDefinitionObject
}
```

The directive definition object contains many properties like restrict which had a value E means that the directive is a html element. Next is the template property that is replaced with the directive. If we give value of replace property as true, it will replace the directive witth the html in the template.If we set this value to false, then the html of template property will be put inside the directive tag.refer to example directives/directive_replace_Value_false.html

The full Directive API

Below is the full directive API with all properties.

```
var appModule = angular.module('appModule',[]);
appModule.directive('namespaceDirectiveName', function factory(injectables) {
var directiveDefinitionObject = {
restrict: string,
priority: number,
template: string,
templateUrl: string,
replace: bool,
transclude: bool,
scope: bool or object,
controller: function controllerConstructor($scope,
$element,
$attrs,
$transclude),
require: string,
link: function postLink(scope, iElement, iAttrs) { ... },
compile: function compile(tElement, tAttrs, transclude) {
return {
pre: function preLink(scope, iElement, iAttrs, controller) { ... },
post: function postLink(scope, iElement, iAttrs, controller) { ... }
}
}
};
return directiveDefinitionObject;
});
```

Understanding the restrict property

The restrict property declares if the directive can be used in a template as an element, attribute, class, comment, or any combination.

See the list below

Character	Declaration Type	Example
E	element	<hello title=Products></hello>
A	attribute	</menu>
C	class	
M	comment	<!-- directive: menu Products -->

What if you want to use your directive as combination of behaviours. Say you want to to use your directive as either an element or an attribute? Simply pass EA as the restrict string.

If you omit the restrict property, the default is A,

Creating a directive as attribute,element and class

Lets create a custom diretive with name custom-dir that can be used as comment,attribute as well as element.

Note that our direcitve name is custom-dir in html view, however if there is a hyphen, we need to remove it and the first character should be capital. Ex in our case for cusom-dir, the name in directive declaration is **customDir.** Refer to directives/directive_restrict_variations.html

```
<body>
    <custom-dir></custom-dir>
    <br>
    <p custom-dir></p>
    <br>
    <h3 class = "custom-dir"></h3>
  <script>
    var appModule = angular.module('app', []);
    appModule.directive('customDir', function() {
      return {
        restrict: 'EAC',
        template: '<div> <h4>HI</h4><p>I am created by directive template</p></div>',
        replace: true
       };
    });
  </script>
  </body>
```

Understanding directive priority

Directives have priority property. In case there are multiple directives in an element, the directive with the higher priority gets applied first.

If we have multiple directives on a single DOM element and If the

order in which they're applied matters,we can use the priority property to order their application.

Note that Higher number priority directives run first.

The default priority is 0 if we don't specify one.

Ex for For ng-repeat, we use a priority value of 1000

Understanding Template property

Directives have priority property. In case there are multiple directives in an element, the directive with the higher priority gets applied first.

If we have multiple directives on a single DOM element and If the

order in which they're applied matters,we can use the priority property to order their application.

Note that Higher number priority directives run first.

The default priority is 0 if we don't specify one.

Ex for For ng-repeat, we use a priority value of 1000

Understanding Template property

We can replace or wrap the contents of an element with a html that we provide in the template property. We have already seen it in our introductory example

we wrote a directive: a <hello> element that just replaces itself with the html we provided.

Note that the default behavior is to append content to elements. We usualyy l

set replace to true to replace the original template

Using templateUrl for external templates

The template property has a drawback of getting the html code in the javascript code. A better approach is to keep the html code in a separate html file. We achieve this using the templateUrl property. Lets see the example below(refer to directives/external_template.html)

```
<script src="angular.js" type="text/javascript"></script>
<script>
var appModule = angular.module('app', []);
appModule.directive('externalTemplate', function() {
return {
restrict: 'E',
templateUrl: 'templates/external_template.html',
replace: true
};
})
</script>
<body ng-app="app">
<external-template></external-template>

</body>
```

Transclusion

We can also move the original content within the new template through the transclude property.

if set to true, the directive will delete the original content within the directive,but make it available for reinsertion within the template through a directive called ng-transclude. See the example below(refer to directives/transclude.html)

```
 <body>
    <div hello><h3> hey i am h3 level heading</h3><p> i am a
     paragraph</p></div>
<script>
  var appModule = angular.module('app', []);
  appModule.directive('hello', function() {
    return {
      restrict:'A',
      template: '<div>Hi there <h3 ng-transclude></h3></div>',
      transclude: true,
      replace: true
    };
  });
</script>
  </body>
```

Scopes

Three types of scopes available for directives

1. The existing scope of our directive's DOM element.

2. A new scope We can create that inherits from our enclosing controller's scope.

Here, we will have the ability to read all the values in the scopes and this scope will be shared with any other directives on our DOM element that request this kind of scope and can be used to communicate with them.

3. An isolate scope that inherits no model properties from its parent. We can use this option when we need to isolate the operation of our directive from the parent scope usually while creating reusable components.

using existing scope

To create existing scope we set the value of scope as false.S ee the example(refer to directives/parentscope_directive.html)

Note that in this example the scope is inherited from parent. Also note that any change we do to the parent property in directive also changes the parent property.

```
<script>
  var app = angular.module("app",[]);
  app.controller("Ctrl1",function($scope){
    $scope.name = "Harry";
    $scope.alertName = function(){
      alert($scope.name); }; });
  app.directive("testDirective", function(){
    return {
      restrict: "A",
      scope: false,
      template: "<div>Your name is : {{name}}"+
      "Change your name witht the input: <input type='text' ng-model='name' /></div>",
      replace:true
    };
  });
  </script>
</head>
<body ng-app = "app">
 <div ng-controller="Ctrl1">
  <h2 ng-click="alertName()">Hi  {{name}}, Click to alert your name</h2>
  <div test-directive></div> </div>
```

Creating new scope with inheritance

If we set the scope value to true, it creates a new scope with inheriting all the properties from the parent. However unlike the previous example, this time the the directive scope and parent scope are not linked. Means any change of property by directive will not change the parent property. Lets see in the example below.

In the same example change the scope value from false to true and test it again.

Please refer to the example directives/newscope_directive.html

Difference between scope true vs scope false

When scope is set to “true”, AngularJS creates a new scope by inheriting parent scope. Any changes made to this new scope will not reflect back to the parent scope. However, any changes made in the controller of the parent scope will be reflected in the directive scope.

with scope set to “false”, the parent controller and directive are using the same scope object. This means any changes to the controller or directive will be in sync.

Create isolated scope

Isolate scopes doesn't inherit model properties, they are still children of their parent scope. Like all other scopes, they have a $parent property that references their parent.

When you create an isolate scope, you don’t have access to anything in the parent scope’s model by default. You can, however, specify that you want specific attributes passed into your directive. You can think of these attribute names as parameters to the function.

To create a isolate scope we just need to write scope:{} in the DDO(directive definition Object).

The new scope also known as Isolated scope because it is completely detached from its parent scope.

Communicating in isolate scope

We have 3 modes in which we can communicate with parent properties within our directive for an isolated scope. Angularjs gives us flexibility to choose what properties we want to inherit from the parent and at what level of behavior(ie one way or two way binding) .This way we can selectively choose what we need within our directive.

Following are the modes

1. "@" one way text binding to parent's scope properties
2. "=" two-way binding to parents scope properties
3. "&" One way Behavior or Method binding

Lets understand each of these in the coming slides

One way text binding

We can create text bindings which are prefixed with @. They are always strings. Whatever we write as attribute value, it will be parsed and returned as strings. Note that we need to keep the parent property value inside an expression. Ie inside double curly braces. Lets see the example below(refer to directives/oneway_binding_text.html). Refer to the next slide for explanation

```
<script>

 var app = angular.module("app",[]);
 app.controller("Ctrl1",function($scope){
    $scope.name = "Smith";
    $scope.alertName = function(){
      alert($scope.name);
    };
  });
  app.directive("testDirective", function(){
    return {
      restrict: "EA",
      scope: {personName:"@pName"},
      template: "<div>Your name is : {{personName}}",
      replace:true,
      link: function (scope, iElm, iAttrs) {
       alert(scope.personName);
  }
    };
  });
  </script>
</head>
<body ng-app = "app">
 <div ng-controller="Ctrl1">
  <h2 ng-click="alertName()">Hi  {{name}}, Click to alert your name</h2>
  <div test-directive p-name = "John {{name}}"></div>
</div>
```

One way textual binding Example explained

Lets understand the one way textual binding. In the previous example the parent controller is Ctrl1. It has a name property which we want to send to the isolated directive.To do this we will follow 2 steps.

Step1: in the scope value we set `scope: {personName:"@pName"}`. Here the property personName is the value that will be used within directive. @pName is the name that is attribute name in the directive element. However the trick is that pName will become p-name attribute in the directive element.

Step2: In the directive element write an attribute with name p-name with value that we want to be made available to the directive. In our example we want the name property of the controller to be prepended with John. We do it as follows `<div test-directive p-name = "John {{name}}"></div>`

Thats it.. now the property personName is available within the directive scope.

Important point to note is that it is 1 way binding. Changing the value of name in parent will effect its value in the directive.

Two way binding

Two-way bindings are prefixed by = and they can be of any type.
These work like actual bindings ie. any changes to a bound value will be reflected in everywhere.
Lets see an example below(refer to directives/twoway_binding.html)

```
<script>
  var app = angular.module("app",[]);
  app.controller("Ctrl1",function($scope){
    $scope.name = "Smith";
    $scope.alertName = function(){
      alert($scope.name);
    };
  });
  app.directive("testDirective", function(){
    return {
      restrict: "EA",
      scope: {personName:"=pName"},
      template: "<div>Your name is : {{personName}}Change your name witht the input: <input
      type='text' ng-model='personName' /></div>",
      replace:true,
    };
  });
  </script>
</head>
<body ng-app = "app">
 <div ng-controller="Ctrl1">
  <h2 ng-click="alertName()">Hi  {{name}}, Click to alert your name</h2>
  <div test-directive p-name = "name"></div>
</div>
```

Two way binding example explained

Two-way bindings are prefixed by = and they can be of any type. Its important to note that we can not use expressions(ie double curly braces) for two way bindings.Again we have to follow two simple steps same as we did for one way textual binding. Following are the steps

Step1: in the scope value we set scope: {personName:"=pName"}. Here the property personName is the value that will be used within directive. @pName is the name that is attribute name in the directive element. However the trick is that pName will become p-name attribute in the directive element.

Step2: In the directive element write an attribute with name p-name with value that is in the parent. We do it as follows <div test-directive p-name = "name">

Thats it.. now the property personName is available within the directive scope.

Important point to note is that it is 2 way binding. Changing the value of name in directive will effect its value in parent and vice versa.

Method binding using @

The third type of binding is using @. This binding binds a value, object or a function. This type of binding returns a function for a text property or object. We need to call the function to access the value.

Lets see the example below(refer to directives/oneway_binding_&.html)

```
<script>
  var app = angular.module("app",[]);
  app.controller("Ctrl1",function($scope){
    $scope.name = "Smith";
    $scope.alertName = function(){
      alert($scope.name);
    };
  });
  app.directive("testDirective", function(){
    return {
      restrict: "EA",
      scope: {personName:"&pName"},
      template: "<div>Your name is : {{personName()}}</div>",
      replace:true,

    };
  });
  </script>
</head>
<body ng-app = "app">
 <div ng-controller="Ctrl1">
  <h2 ng-click="alertName()">Hi  {{name}}, Click to alert your name</h2>
  <div test-directive p-name = "name"></div>
</div>
```

Understanding the Method Inheriting example

As you can see in the example we exactly follow the same steps. Just one difference. While referring a method in directive as attribute we refer like this

```
<div test-directive  bind-alert = "alertName()"></div>
```

Here bind-alert binds to alertName of the parent scope.

Understanding the example

Lets understand the example. again we have followed the same steps as before. Note that in DDO(directive definition object) we write as
scope: {personName:"&pName"}, .. Here personName is the function that returns the value.

Note that the binding is obviously one way not two way.

This way we can also inherit functions from parent scope. Lets now try to inherit a function using this technique.

Method binding for isolate scope using &

Lets understand with an example how we can bind a method using & notation.
See the code below(refer to directives/oneway_method_binding.html).

```
<script>
  var app = angular.module("app",[]);
  app.controller("Ctrl1",function($scope){
    $scope.name = "Smith";
    $scope.alertName = function(){
      alert($scope.name);
      return "yoo";
    };
  });
  app.directive("testDirective", function(){
    return {
      restrict: "EA",
      scope: {sayName:"&bindAlert"},
      template: "<button ng-click = 'sayName()'>Click Here</button>",
      replace:true,
    };
  });
  </script>
</head>
<body ng-app = "app">
 <div ng-controller="Ctrl1">
  <h2 ng-click="alertName()">Hi  {{name}}, Click to alert your name</h2>
  <div test-directive  bind-alert = "alertName()"></div>
</div>
</body>
```

Understanding the Method Inheriting example

As you can see in the example we exactly follow the same steps. Just one difference. While referring a method in directive as attribute we refer like this

```
<div test-directive  bind-alert = "alertName()"></div>
```

Here bind-alert binds to alertName of the parent scope.

Simple Binding

Now we have understood that we can bind properties of parent in three ways in our isolate directive scope.
Note that if we dont want to refer with a different names in directive scope and directive DOM, we can drop the name after the = , @ or &.
Lets see a simple example demonstrating the concept(refer to directives/bindingexample_showing_all.html)

```
<script>
  var app = angular.module("app",[]);
  app.controller("Ctrl1",function($scope){
    $scope.name = "Smith";
    $scope.age = 28;
    $scope.alertName = function(){
      alert($scope.name);
    };
  });
  app.directive("testDirective", function(){
    return {
      restrict: "EA",
      scope: {
        sayName:"&",
        personAge:"=",
        personName: "@"
      },
      template: "<div><h4>The age is {{personAge}} and name is {{personName}}</h4><button ng-click =
     'sayName()'>Click Here to view age</button> <br>MOdify Age: <input type = 'text' ng-model  =
     'personAge'><br>Modify Name:<input ng-model = 'personName'></div>",
      replace:true,
    };
  });
  </script>
</head>
<body ng-app = "app">
 <div ng-controller="Ctrl1">
  <h2>Hi  Name is :{{name}} and age is {{age}}</h2>
  <div test-directive  say-name = "alertName()" person-age ="age" person-name = "John {{name}}"></div>
</div>
```

Using controllers in directives

We can use controllers in directives.

Directives can also have controllers to attach data or functions within their scope. We just need to add the controller property to the DDO with the function that we want to be our controller. Below is the syntax

```
angular.module('app', [])
.directive('testDirective', function() {
restrict: 'E',
controller:
function($scope, $element, $attrs, $transclude) {
   // The controller logic goes here
  }
})
```

Note that we can inject the Angular injectors here like $scope,$element,$attrs and $transclude.

Here $element represents the element reference of the directive.

$attrs contains the attributes object for the current element.

$transclude is the function to modify the DOM and play with the content within the directive if we need.

In the coming secion we will be seeing all of these in our examples.

Controller example for directive

This example shows how to use the controller within the directive. This is similar to what we have used in our controllers example earlier.

Note that directive element is the element that represents the directive or contains the directive(in case of class or attribute).In our example the directive element is <div test-directive attr1 = "attr" attr2 = "attr2">

Next note that within the directive we have placed 2 attributes with name attr1 and attr2. We have placed them to show that we can also access these attributes within our controller. Within our controller we have placed the function showParametersInConsole that prints the attributes in the console.
We have kept a greet method within our controller which we can in the directive template using a button.

Next lets understand the $transclude function.

Understanding the $transclude function.

The $transclude function is normally not required as angular itself takes care of transclusion. However in scenariios if we want to append some html or modify html, we can make use of the $transclude function.

Below is the snippet showing the use of $transclude function.note that we can pass params amongst which is the first param here with the name clone. This parameter contains reference to the transcluded element that contains the html within the directive element.

```
$transclude(function(clone){
        //this is the reference to the cloned element
        console.log(clone);
        //this will print text inside the directive
        console.log(clone.text());
        var a = angular.element('<div>');
        a.attr('class','divclass');
        a.text("i am created using transclude function");
        $element.append(a);
       })
```

$transclude function example

In the example below we are adding a new div with class as divclass using the $transclude function.

```
<script>
  var app = angular.module("app",[]);
  app.directive("testDirective", function(){
    return {
      restrict: "EA",
      template: "<div><h4>I am heading inside the directive</h4></div>",
      replace:true,
      transclude:true,
      controller:function($scope, $element, $attrs, $transclude){
        $transclude(function(clone){
          //this is the reference to the cloned element
          console.log(clone);
          //this will print text inside the directive
          console.log(clone.text());
          var a = angular.element('<div>');
          a.attr('class','divclass');
          a.text("i am created using transclude function");
          $element.append(a);
        })
      }
    };
  });
  </script>
</head>
<body ng-app = "app">
    <div test-directive attr1 = "attr" attr2 = "attr2">I am to be cloned</div>
</body>
```

Using controllerAs for controllers

We have another option in DDO to create an alias of a controller reference. Though its v simple concept, yets its very powerful and very helpful. Using controllers as option, we no longer need to use the $scope within our controller. We can simply use the this current controller instance.

The important point is that when we are trying to access any property or funtion withn the html, we can use the reference as the string defined in the controllerAs option in the directive.

In the next example we have created a directive with a controller that has a conroller with 2 variables and one function within it. Note that instead of using $scope, we have used controller current instance this to attach them.

$transclude function example

In the example below we are adding a new div with class as divclass using the $transclude function.

```
<script>
  var app = angular.module("app",[]);
  app.directive("testDirective", function(){
    return {
      restrict: "EA",
      template: "<div><h4>I am heading inside the directive</h4></div>",
      replace:true,
      transclude:true,
      controller:function($scope, $element, $attrs, $transclude){
        $transclude(function(clone){
          //this is the reference to the cloned element
          console.log(clone);
          //this will print text inside the directive
          console.log(clone.text());
          var a = angular.element('<div>');
          a.attr('class','divclass');
          a.text("i am created using transclude function");
          $element.append(a);
        })
      }
    };
  });
  </script>
</head>
<body ng-app = "app">
    <div test-directive attr1 = "attr" attr2 = "attr2">I am to be cloned</div>
</body>
```

Using controllerAs for controllers

We have another option in DDO to create an alias of a controller reference. Though its v simple concept, yets its very powerful and very helpful. Using controllers as option, we no longer need to use the $scope within our controller. We can simply use the this current controller instance.

The important point is that when we are trying to access any property or funtion withn the html, we can use the reference as the string defined in the controllerAs option in the directive.

In the next example we have created a directive with a controller that has a conroller with 2 variables and one function within it. Note that instead of using $scope, we have used controller current instance this to attach them.

Using controllerAs example

Example demonstration use of controllerAs option for a directive.

```
  <script>
  var app = angular.module("app",[]);
  app.directive("testDirective", function(){
    return {
      restrict: "EA",
      template: "<div>Application Name is {{controllerRef.appName}} <h4>I am
     heading inside the directive</h4><button ng-click = 'controllerRef.greet()'>CLick
     me</button></div>",
      replace:true,
      controllerAs:"controllerRef",
      controller:function(){
        this.appName = "Ecommerce App";
        this.msg = "Hello..Good Morning";
        this.greet = function(){
          alert(this.msg);
        }
      }
    };
  });
  </script>
</head>
<body ng-app = "app">
    <div test-directive attr1 = "attr" attr2 = "attr2">I am to be cloned</div>
</body>
```

Using external controller withn a directive

We can also use external controller defined within the application. Thus controllers can be reused within the directives. We just need to specify the controller name within the directive. Note that the name of the controller is a string so we have to keep within single or double quotes.

Lets see the next example.you can see that we have defined the controller Ctrl1 outside the directive, yet we are able to use the controller functions and properties within our directive.

Lets see the example in the next slide demonstrating this concept.

Using external controller withn a directive

We can also use external controller defined within the application. Thus controllers can be reused within the directives. We just need to specify the controller name within the directive. Note that the name of the controller is a string so we have to keep within single or double quotes.

Lets see the next example.you can see that we have defined the controller Ctrl1 outside the directive, yet we are able to use the controller functions and properties within our directive.

Lets see the example in the next slide demonstrating this concept.

Using controllerAs example

Example demonstration use of external controller for a directive.

```
<script>
  var app = angular.module("app",[]);
  app.controller("Ctrl1",function($scope){
    $scope.name = "Smith";
    $scope.age = 28;
    $scope.alertName = function(){
      alert($scope.name);
    };
  });
  app.directive("testDirective", function(){
    return {
      restrict: "EA",
      template: "<div><h4>I am heading inside the directive</h4><button ng-click =
     'alertName()'>CLick me</button></div>",
      replace:true,
      controller:'Ctrl1'
    };
  });
  </script>
</head>
<body ng-app = "app">
  <h2>Hi  Name is :{{name}} and age is {{age}}</h2>
  <div test-directive></div>
</body>
```

Using require for directives

One directive can use controller of other directive. To do it we need to use the require property in the DDO and set its value to the directive whose controller we want to use.

In the next slide we have created 3 directives. One directive is the main directive with the name productCollection. The two other directives namely brush and cream are attribute based directives. The directive productCollection has a function called add item in its controller. We will be using this function to add items to the main directive.

All we need is to write place the two directives namely brush and cream as attributes to the main directive. Automatically both the items are added to the items list of main directive(productCollection). We have used require in both the directives brush and cream. Its important to note that we can get access to the controller of productCollection in these directives within the link function. If we use require, automatically the fourth parameter has the reference to the controller we mention in require property.

Using require example part 1

This is the main product collection directive. The next slide will continue this example.

```
<script>
  var app = angular.module("app",[]);
  app.directive("productCollection", function(){
    return {
      restrict: "E",
      template: "<div><h4>This is the basket</h4><button ng-click = 'pCRef.totalAmount()'>Show
     Total</button><button ng-click = 'pCRef.showItemsNo()'>Items Count</button></div>",
      replace:true,
      controllerAs:"pCRef",
      controller:function($scope, $element, $attrs, $transclude){
        this.itemsCount = 0;
        this.items = [];
        this.msg = "test";
       this.showItemsNo = function(){
        console.log('in')
        alert(this.items.length);
       }
       this.totalAmount = function(){
        var totalAmount = 0;
        for(var i = 0;i<this.items.length;i++){
          totalAmount+=this.items[i].price;
        }
        alert(totalAmount);
       }
       this.addItem = function(item){
        this.items.push(item);
       }
      }
    };
  });
```

Using require example part 2

This is continuation of example in previous slide. Here we have written the two directives that will require the main directive.

```
app.directive("brush",function(){
    return {
      require:"productCollection",
      link:function(scope, element, attrs,pCCtrl ){
        console.log(pCCtrl);
        pCCtrl.addItem({name:'brush',price:12});
      }
    }
  })

  app.directive("cream",function(){
    return {
      require:"productCollection",
      link:function(scope, element, attrs,pCCtrl ){
        pCCtrl.addItem({name:'cream',price:30});
      }
    }
  })
  </script>
</head>
<body ng-app = "app">

  <product-collection  brush ></product-collection>
</body>
```

Using the $transclude function

We have already covered what is transclusion. We can put the contents between the directive tags in our directive view using transclude property. Note that we used the ng-transclude property to get the content within the directive view. Angular gives us power to also use the $tranclude function by which we can put the contents inside directive tags in directive view. Using $tranclude function we get a lot of flexibility to decide where and how we want to modify the content. Note that we pass a function to the $transclude function

```
<script>
 var app = angular.module("app",[]);
 app.directive("testDirective", function(){
   return {
     restrict: "EA",
     template: "<div><h4>I am heading inside the directive</h4></div>",
     replace:true,
     transclude:true,
     controller:function($scope, $element, $attrs, $transclude){
       $transclude(function(clone){
         //this will print text inside the directive
         var a = angular.element('<div>');
         a.attr('class','divclass');
         a.text("i am created using transclude function");
         $element.append(a);
       })
     }
   };
 });
 </script>
</head>
<body ng-app = "app">
   <div test-directive attr1 = "attr" attr2 = "attr2">I am to be cloned</div>
```

Server Communcation

In the coming section, We will be covering about server communication. We will understand how we can send the data to and fro to server.

To start with we will be using the $http service, learn about setting headers, Modifying them, Caching server responses.

Later part we will learn even a better and advanced way using the Restful built in API in AngularJs which is a better and recommended way of developing the REST API.

$http service

The full $http API

```
$http({
method: string,
url: string,
params: object,
data: string or object,
headers: object,
transformRequest: function transform(data, headersGetter) or
an array of functions,
transformResponse: function transform(data, headersGetter)
or
an array of functions,
cache: boolean or Cache object,
timeout: number,
withCredentials: boolean
});
```

Understanding parameters in details

method: The method represents the method of communicating to the server. It can be GET,POST,DELETE etc

url: the Url parameter represents the url of the server.
params:The parameters to send with the request.
data: the data to be sent with the request. Ex post data.
Headers: any additional headers to be sent with the request.
transformRequest: the function to modify the request
transformResponse: the function to modify the server response
cache:boolean parameter for caching or custom caching object reference
timeout:the number to represent the number of milliseconds after which request will timeout
WithCredentials: setting this parameter to true will also send the session id with the request.

Simple $http request example

Refer to scm/http/http_get.html. Note that the

```
<script type="text/javascript">
var myModule = angular.module('myModule',
[]);

myModule.controller('MyCtrl', function ($scope,$http,$sce) {
$scope.todos = [];
$scope.errorMsg = null;
$scope.click = function() {
$http.get('slim/todos', {params: {id: '5'}
}).success(function(data, status, headers, config) {
$scope.todos = data;
}).error(function(data, status, headers, config) {
$scope.errorMsg = $sce.trustAsHtml(data);
});
};
});
</script>
</head>
<body>
<div ng-app="myModule"
<div ng-controller="MyCtrl">
<div ng-repeat = "todo in todos">
status is: {{todo.status}} and description is: {{todo.description}}
</div>
<div ng-bind-html = "errorMsg"></div>
<button ng-click="click()">Show Number</button>
</div>
</div>
```

Understanding $http request example

The example makes a get request to the server which returns an array of todo items. We bind it to ng-repeat. As the data comes from server, the data changes.

Note that the errorMsg contains the error if any happens . The data parameter of the error functions contains the response from the server.

Try changing the url and you will see the error message from the server shown in the view.

Note that If we try to show the error message directly It will show the html code returned from the server. This is a security feature to avoid xss attack. To show an html that we trust we have to pass it to the trustAsHtml function of $sce service like this $scope.errorMsg = $sce.trustAsHtml(data);

The returned valued can then be bound to the html using the ng-bing-html directive as shown in the html like this

$http function shortcuts

Angular provides a list of functions which are shortcuts for making different types of requests.

$http.get :for making get request
$http.head: for making head request
$http.post :post requests shortcut
$http.put: put request shortcut
$http.delete : delete request shortcut
$http.jsonp : for JSONP requests
$http.patch : For making patch requests

Following is the syntax to use the functions

```
$http.get('/serverUrl').success(successCallback);
$http.post('/serverUrl', data).success(successCallback);
```

Caching the server Response

TO implement caching we can set the caching parameter value to be true.

```
$http.get('http://server/api', {
cache: true
}).success(function() { //success handling });
```

Note that for multiple simultaneous requests for same URL, only one request is made to the server and the response is used for all the requests. Even if the response is being served from the cache,It is still Asynchronous in nature means our code will behave as it did when it first made the request.

request and response Transformation

Request and response transformation is the modification of request or response that is made by angular.

Why do we need to modify the request or server response?
There are many scenarios..Ex : to add custom headers, to add additional data to request.

We may also modify the response too for few scenarios, ex: if the server returns JSON string, we might want to convert it into object to avoid it doing ourself everytime.

Angular provides few inbuilt request and response tranformations too

Request transformations

If the data property of the config request object contains an object,angular serializes it into JSON format.

Response transformations

If an XSRF prefix is detected in the response,angular strips it.
If a JSON response is detected, angular deserializes it using a JSON parser

Custom transformations

If we don’t want some of the transformations, or to add our own, then we can pass in functions as part of the config.

We can transform a request or a response at the request/response phase using the transforRequest and transformResponse property of the object that we pass to the $http function.

Note that this method is good if we want to modify a single request/response.

If we want to modify the request/response globally, we need to do it in the config function of the module. Lets understand this in the next section.

Custom response transformation example

Refer to scm/http/responsetransformation.html

```
<script type="text/javascript">
var myModule = angular.module('myModule',
[]);
myModule.config(function($httpProvider){
$httpProvider.defaults.headers.common.token = 'LXYM564';
$httpProvider.defaults.transformResponse = function(data){
return {responseData:angular.fromJson(data),currentTime:Date.now()};
}
})
myModule.controller('MyCtrl', function ($scope,$http,$sce) {
$scope.todos = [],$scope.errorMsg = null;
$scope.click = function() {
$http.get('slim/todos').success(function(data, status, headers, config) {
console.log(data);
$scope.todos = data.responseData;
}).error(function(data, status, headers, config) {
$scope.errorMsg = $sce.trustAsHtml(data);
});
};
});
</script></head>
<body><div ng-app="myModule"
<div ng-controller="MyCtrl">
<div ng-repeat = "todo in todos">
status is: {{todo.status}} and description is: {{todo.description}}</div>
<div ng-bind-html = "errorMsg"></div><button ng-click="click()">Show Number</button>
    </div></div>
```

Custom transformation example explained

IN the previous example we have transformed the response for all requests in the config function of the module. We have added a token value to the request header. If you see the console you can see the token value as header.

Also we have modified the response that is received from the server. Note that the response received was a json string, so we converted to object, added a new todos object to the array and returned it.

Handling Restful Resources

$

http provides a great API for communcating with server but to make REST requests with $http API means we have to write code for GET,POST, UPDATE and DELETE. However we have a better API, the restful API.

To have the restful API, we need to use the ngResource Module.
The ngResource module provides a well built API to handle rest based calls.
To use it we have to download the angular-resource.js file.
Next while declaring the app we have to include ngResource as dependency.

Ex: Var app = angular.module('myApp',[ngResource]);

Using the ngResource API

To create a resource object we need use the $resource service
Below is the sytax

```
var resourceOb = $resource('api/comments/:id');
```

Here the :id represents the dynamic parameter id for the REST request
This resource object consists of a set of functions to communcate with the REST API. Following are the functions availble on the resource object

```
'get':    {method:'GET'},
'save':   {method:'POST'},
'query':  {method:'GET', isArray:true},
'remove': {method:'DELETE'},
'delete': {method:'DELETE'}
```

To calll a get method we just need to write

```
Var resultOb= Entry.get({id:12},function(){
        –  //have success handler here
});
```

Lets see the usage with a complete example in the next section

Simple ngResource Example

Refer to scm/rest/simplerest_example.html

```
<script>
var app = angular.module('myApp',['ngResource']);
app.factory('Entry', function($resource) {
  return $resource('api/comments/:id');
});
app.controller('test',function($scope, Entry) {
  $scope.entries = null, $scope.singleComment = null,$scope.id = 12;
  $scope.fetch = function(id){
  $scope.singleComment = Entry.get({id:$scope.id});
}
 $scope.deleteComment = function(){
  Entry.delete({id:$scope.id},function(){
  alert("Comment delete request sent to server");
}) }
$scope.postBack = function(){
  Entry.save($scope.singleComment,function(){
  alert("Modified comment successfully posted back to server");
}) }
$scope.fetchAll = function(){
$scope.entries = Entry.query();
} });
</script></head><body>
<div ng-controller = "test">
<button ng-click = "fetchAll()">Fetch All</button>
<div id = "all-comments">
<div ng-repeat = "entry in entries">comment fetched id comment with id {{entry.id}} and fulll description as
    {{entry.description}}
</div></div>
<input type = "number" ng-model = "id"><button ng-click = "fetch()">Get Comment</button>
<button ng-click = "deleteComment()">Delete COmment</button>
<div id = "single-comment">The comment with id {{id}} has status {{singleComment.status}} and full descriptions as
    {{singleComment.fulldescription}}</div>
MOdify the comment below
<input type  = "text" ng-model = "singleComment.status"><input type  = "text" ng-model = "singleComment.fulldescription">
<button ng-click = "postBack()">Post back </button></div>
```

Using the returned restful object

Note that When data is returned from the server, the returned object is also an instance of the resource class.

The actions save, remove and delete are available on this object as methods with $ prefix.
This technique allows us to to easily perform CRUD operations on the returned object itself.
The previous example can also be rewritten like this. (refer to scm/rest/restexample.html)

```
<script>
var app = angular.module('myApp',['ngResource']);
app.factory('Entry', function($resource) {
return $resource('api/comments/:id');
});
app.controller('test',function($scope, Entry) {
$scope.entries = null, $scope.singleComment = null,$scope.id = 12;
$scope.fetch = function(id){
$scope.singleComment = Entry.get({id:$scope.id});
}
$scope.deleteComment = function(){
$scope.singleComment.$delete();
}
$scope.postBack = function(){
$scope.singleComment.$save();
}
$scope.fetchAll = function(){
$scope.entries = Entry.query();
}
});
</script>
```

Restful example explained

IN the previous example we have tried to use the REST API to make get,post and delete requests.

Note that we have used the factory singleton object to retrieve the resource object.

To fetch all the comments from the API, we have use the query function which returns all the comments. Its important to understand that we dont need to bind the result in the success callback. The data is automatically retrieved and reflected in the template as it is returned.See the snippet
`$scope.entries = Entry.query();`

Here the return value is directly bound to the $scope.entries. We havents bound the value in the success callback. This is taken care by angularjs.

Caching in Angularjs

The $cacheFactory service generates cache objects for all Angular services.
We can create a cache object by var appCache = $cacheFactory('AppCacheID');
Here AppCacheID is the id of appCache.

Note that we can pass the optional object to the $cacheFactory where we can speciify the number of cache objects it can store.

With this returned object(appCache in this case) we can store and retrieve the cache data

Using cache object

The cache object has many methods to store,retrieve and get information about the caches.

Following are the methods

info() :This method returns ID, size and options of the cache object.

put():This method stores the data in the cache, The first parameter is the key and second is the data to store.appCache.put('name','sumit');
appCache.put('hobby','coding');

get(): get methods retrieves the data from the cache for the given cache id.
appCache.get('name');

remove(): This function removes a key-value pair from the cache, if it’s found. If not found, it returns undefined .appCache.remove('name');

removeAll(): This function resets the cache and removes all cached values.
appcache.removeAll()

destroy() This method removes all references of the current cache from the $cacheFactory cache registry. Ie appCache.destroy()

$http caching

The $http service creates a cache with the ID $http. To enable the $http request to use this default cache object we have to pass a cache parameter for $http service.

Ex

```
$http({
method: 'GET',
url: '/api/comments.json',
cache: true
});
```

Note FOr every request,the key for the $http cache is the full-path URL.

Using $http cache within code

Usually we dont need to use the $http cache object. But if needed we can retrieve it using

```
var cache = $cacheFactory.get('$http');
```

To retrieve the cache object for a particular url

```
var commentsCache =
cache.get('http://acadgild.com/api/comments.json');

// To Delete the cache entry for the previous request
cache.remove('http://acadgild.com/api/comments.json');

// TO remove the entire cache
cache.removeAll();
```

Using custom cache for $http

NOte that we can use our own custom cache for $http too.
Instead of passing a boolean true for $http, we can pass the instance of the cache.
Below is the example

```
var appCache = $cacheFactory.get('AppCacheID');

$http({
method: 'GET',
url: '/api/comments.json',
cache: appCache
});
```

Configuring cache settings using for $http

We can also configure the setting for cache for $http .
Below is the snippet

```
angular.module('CommentApp')
.config(function($httpProvider) {
$httpProvider.defaults.cache =
$cacheFactory('appCache', {capacity: 30});
});
```

Automated Testing

For AngularJs Applications we need 2 types of testing

1)Unit Testing: Unit testing is for testing the code. Its similar to the way we test server side application..ie Junit.

2)End to End testing :Unit testing is not enough. We also need to know how the application behaves to the user interactions in the browser. For this we need to simulate user interactions at the browser.

For unit testing javascript applications we can use jasmine. For End to End testing we can use Protractor.

we will be using protractor testing framework built over selenium and jasmine.

End to End Testing vs Unit testing

We now know that there are 2 types of testing.

Which testing should we use? Unit or End To End ?

The answer is that we should use both to make a robust application.

Unit testing is good for testing controllers whereas end to end is good to check the changes in templates and view. A good application needs both unit and end to end testing.

List of Frameworks needed to complete application

Lets again have a look at the list of frameworks we need for testing

Jasmine: Jasmine is the framework for unit testing

Karma:Karma is a task runner for unit testing. It assist in running the test and provided a lot more features to make testing easier

Protractor:Protractor is the End to end testing framework. It is used to write and execute e2e tests for angular.

jasmine

Jasmine is a behavior-driven development unit testing framework for testing JavaScript code.

Jasmine from be downloaded from http://jasmine.github.io/

Jasmine can also be installed using npm(node package manager).Use the command in

npm install -g jasmine

Alternatively you can download the standalone version of jasmine..Coz its just JavaScript!

We wont be going deep into testing with jasmine because we will be using protractor for testing.

Jasmine spec

An expectation in Jasmine is an assertion that is either true or false.

A spec is a collection of exceptions

A spec with all expectations as true is a passing spec.

If a spec has one or more false expectations, it is a failing spec.

Ex of a spec

```
  it("and has a positive case", function() {
    var a = 5,b=5;
    expect(a).toBe(b);
  });
```

What is a Suite

A suite is a collection of specs.
Below is an example to create a suite. A suite is a collection of specs.

Ex of a suite

```
describe("A suite", function() {
  it("and has a positive case", function() {
    var a = 5,b=5;
    expect(a).toBe(b);
  });
});
```

Setup and tear down for tests

Following are global functions namely beforeEach, afterEach, beforeAll, and afterAll.

The beforeEach function is called once before each spec in the describe is run.
The afterEach function is called once after each spec.
They can be used to reset variables before a spec.

The beforeAll function is called only once before all the specs in a describe are run.
The afterAll function is called after all specs finish.

Settting up and creating tests

We will be downloading the standalone version of jasmine.

After downloading we have the following folders
1)lib: The lib folder contains the libraries for jasmine
2)spec: spec is the folder where will be keeping our tests
3)src: the folder where our code to be tested will be kept

We also have a file called specrunner.html which is the file that we run from browser to run our tests.

The full example is set up at folder named jasminetesting.

Note that in specrunner.html we include the code to be tested, the unit test and the jasmine testing libraries.

A simple example of unit testing with jasmine

We have 2 functions which adds numbers and other multiplies the numbers. W will be writing unit tests in jasmine for them

Below is the code to be tested

```
function AddTwoNumbers(a,b) {
  return a + b; //Function to return addition of two numbers a and b.
}

function MultiplyTwoNumbers(a,b) {
  return a * b; //Function to return product of two numbers a and b.
}
```

Code for unit test

The code is kept as jasminetesting/spec/unittest.js
Below is the code
To test the code just run the specrunner.html file.

```
describe("Testing Two Numbers", function() {
    var a = 4,b = 3;
    beforeEach(function() {
        console.log("Setting up ");
        a = 4,b =3;
    });
    afterEach(function() {
        console.log("Tearing down ");
        a=0,b=0;
    });
    it("Add Numbers", function() {
        expect(AddTwoNumbers(a,b)).toEqual(7);
    });
    it("Multiply Numbers", function() {
        expect(MultiplyTwoNumbers(a,b)).toEqual(12);
    });
    it("Compare Numbers to be Greater Than", function() {
     expect(a).toBeGreaterThan(b);
 });
});
```

Setting up unit testing for AngularJs

To start with we need to install karma and karma cli on the system.

This can be done either using npm(node package manager).

You just need to run npm install -g karma and npm install -g karma-cli.

The above two commands install karma on the machine.

Once done we can now start setting up our environment.

Lets create a directory called project_one. inside this directory we will be creating three directories namely scripts, test,. Inside test directory we will be creating 2 directories namely e2e and unit. e2e will contain end to end tests and unit will contain unit tests.

Create a file called codetotest.js and keep it in the scripts folder. Create a codetotestSpec.js file and keep inside test/unit folder

Use the command line and go to project_one directory.

Inside the directory fire the command karma init for windows and sudo karma init for linux

We use this command to create a karma configuration file. This command automatically asks for some options and finally creates a karma configuration file. The karma will read this file and run the tests.

As soon as we run this command karma init karma will ask you a few options.

Lets look at the options in the next slide.

karma init options

Karma will ask for Setting the unit testing framework. The default is jasmine. We can also use other frameworks like mocha or chai. Note that use tab key to select other choices

---

Which testing framework do you want to use ?
Press tab to list possible options. Enter to move to the next question.
> jasmine

If you are using requireJs then say yes to the next option. Default is no

---

Do you want to use Require.js ?
This will add Require.js plugin.
Press tab to list possible options. Enter to move to the next question.
> no

We can next select the browsers to choose. We will use chrome the default value

---

Do you want to capture any browsers automatically ?
Press tab to list possible options. Enter empty string to move to the next question.
> Chrome

Next option asks for the location of source files to be tested and tests to be run. see the answer highlighted in blue. Note that you can enter muliple options using an enter.ie after entering one option press enter. If you press enter without entering any option it moves to the next question

---

What is the location of your source and test files ?
You can use glob patterns, eg. "js/*.js" or "test/**/*Spec.js".
Enter empty string to move to the next question.
> scripts/*.js test/unit/*Spec.jss
WARN [init]: There is no file matching this pattern.
> scripts/*.js
> test/unit/*Spec.js
>

---

karma init options continued

Next option asks for any pattern of files to be excluded. For now press enter without entering any option.

Should any of the files included by the previous patterns be excluded ?
*You can use glob patterns, eg. "**/*.swp".*
Enter empty string to move to the next question.

Next karma asks if we want to detect any changes in code and retest it. We will use the default yes and press enter.

Do you want Karma to watch all the files and run the tests on change ?
Press tab to list possible options.
> yes

After this we can see the message which says that the config files have been generated.

Testing the code using karma

Remember in the last exercise we tested using stand alone jasmine. This time we will do it using karma.

To test, lets get the code from folder jasminetesting/src/codetotest.js and keep it replace it with the file project_one/scripts/codetotest.js

Next we will open the file inside project_one/test/unit/codetotestSpec.js and copy the contents of jasminetesting/spec/unittest.js into it.

Thats all. What we are doing is running the tests we wrote earlier using karma.

Now go inside the project_one directory and fire the command

```
karma start karma.conf.js
```

Now karma will automatically open the browser and run the tests for us.

After the tests run change some value in the file project_one/test/unit/codetotestSpec.js .

ie. change

```
expect(AddTwoNumbers(a,b)).toEqual(7);
```

To

```
expect(AddTwoNumbers(a,b)).toEqual(17);
```

Karma will automatically detect the change and run the test for you again.

Setting up protractor for End to End testing

Lets now install protractor.

We can install protractor using the command
npm install -g protractor

After that update the webdriver by
npm install webdriver-manager update

This will install protractor in your system

In the next section we will set up a simple testing project with protractor.

Setting up protractor for testing

Lets set up a small project and test with protractor.

Create a folder named project_e2e. Create a folder inside the project_e2e with name src where we will keep our source files.

Lets now create a simple code to run with protractor.

The code just adds 2 values provided in the input box and displays the output on clicking the add button. The code is shown below.(refer to testing/project_e2e/src/ng-model_example.html)

```
<script src="angular.js" type="text/javascript">
</script>
<script type="text/javascript">
var myModule = angular.module('myModule', []);
myModule.controller('MyCtrl', function ($scope) {
$scope.result = null;
$scope.click = function() {
$scope.result = parseInt($scope.a)+parseInt($scope.b);
};
});
</script>
</head>
<body ng-app="myModule">
<div>
<div ng-controller="MyCtrl">
<p>Firstname: <input type="text" ng-model="a"></p>
<p>Lastname: <input type="text" ng-model="b"></p>
<p>{{result}}</p>
<button ng-click="click()">Show Name</button>
</div>
</div>
```

Setting up protractor for testing

To run the protractor we need to create a configuration file that will store config values of protractor. Create a file with the name config.js and keep the following code inside that

Refer to testing/project_e2e/config.js

```
exports.config = {
  seleniumAddress:'http://localhost:4444/wd/hub',
  capabilities:{
    browserName: 'chrome'
  },
  specs: ['ng-model_exampleSpec.js'],
  jasmineNodeOpts:{
    showColor:true
  }
}
```

seleniumAddress is the address where selenium runs.

capabilities property is for which browsers we want to run the test.

specs property is an array that consists of the file to be run by the protractor for testing

jasmineNodeOpts represents options that we can pass to the jasmine node.

Running the test

To run thethe test first we need to start the selenium server. We will use the command webdriver-manager start to start the server.

After the server is started, we will open a new terminal. Go to testing/project_e2e/ directory. Then we fire the command

protractor config.js

This command will load the config.js in the protractor and start the testing

.

You will the chrome browser opened, values entered automatically and tested by the framework.