## Angular JS Interview Questions:

**1. What is root directive? Can we use angularjs with out ng –app?**

\* The root directive in angularjs is ng-app. We declare this directive in the root element of the main html document(index.html)

\*ng-app is called root directive because the angular framework starts execution of the application from ng-app. And this process is called **Autobootstrap**`

\*We can run the angular application with out ng-app through manual bootstrap

**Manual bootstrap:** Execution of the Angular application forcefully with out ng-app is called Manual bootstrap....

\*Sample code to run the application by using manual bootstrap i.e without using ng-app

<html>

<h1 style=”color:red” ng-app=”app” ng-bind=”var\_one”></h1>

<h2 id=”myh2” ng-bind=”var\_two”></h2>

</html>

Manualbootstrap.js

angular.element(document).ready(function(response){

angular.bootstrap(document.getElementById(“myh2”),[“app”]);

});

In the abovee code where there is ng-app scope that will be executed using auto bootstrap that is h1 element

And the h2 element executes using manual bootstrap

Note: we have to include(load) the manualbootstrap.js file in the index.html file

Note: we can use any number of **ng-app** in the application.. as the application is divided in to number of modules so we can integrate every modules by creating one more new project each having a ng-app shown below

Example:

**In app.js file**

Var app = angular.module(“app”,[])

Here we can the add different ng-apps of different modules

**2. Features of Angularjs**

Angular js is a Javascript framework to create Rich Internet Applications and the application written in angularjs are cross browser compatible i.e., executes on any major web browser.

The most important key features of the Angularjs are

**Data-Binding:** It is automatic data synchronization between model and view..

**Scope:** These are objects that refer to the model. They act as a glue between controller and view.

**Controller:** A controller is a **JavaScript** file which contains attributes/properties and functions. Each controller has $scope as a parameter which refers to the application/module that controller is to control.

**Services** − AngularJS has several built-in services for example $https which is used to make requests using some url and fetch the required data.. These services creates the objects only once and this object is shared to all the controllers.. so they are called singleton objects which are instantiated only once in app.

**Filters** − AngularJS provides filters to transform data

Example: currency Format a number to a currency format. date Format a date to a specified format. **filter** Select a subset of items from an array. json Format an object to a JSON string...

These select a subset of items from an array and returns a new array

**Directives** − directives are markers on a DOM element (such as an attribute, element nameor CSS class) that tell AngularJS's **HTML compiler**  to attach a specified behavior to that DOM element

Ther are two types of directives they are

**pre -defined directives:** These are built in directives provided by the angular frame work ex: ng-model, ng-bind, ng-repeat etc

**custom directives:** creating our own directives based on application requirement called as custom directives

ex: my\_directive................. custom directive

<my\_directive> </my\_directive>......................elementt level usage

<div my\_directive></div> .................................attribute level usage

**Templates** − These are the rendered view with information from the controller and model. These can be a single file (like index.html) or multiple views in one page using "partials"

**Routing** − It is concept of switching views. This concept is in question no :4

**Deep Linking** − Deep linking allows you to encode the state of application in the URL so that it can be bookmarked. The application can then be restored from the URL to the same state.

**Dependency Injection** − AngularJS has a built-in dependency injection subsystem that helps the developer by making the application easier to develop, understand, and test.

# 3. Directives and uses:

Below are some pre-defined directives of AngularJS

|  |  |  |
| --- | --- | --- |
| Directive Name | Functionality | Example |
| 1,ng-app | It is a root directory in angularJS ,Framework will start the execution from  ng-app.  This directive can be mention in “HTML”(view). | Ex: <html ng-app=”app”> |
| 2,ng-controller | This directive used to declare controllers.  We can have one or more controllers per application.  In general we will declare controllers in view | Ex:  app. controller("ctrl",ctrl);  ctrl.$inject=["$scope"];  function ctrl($scope){  $scope.varone="angularJS";  } |
| 3,ng-model | Binding the view into the model, which other directives such as input, text area or select require.  Providing validation behavior (i.e. required, number, email, url). | Ex:  <form name="testForm" ng-controller="ExampleController">  <input ng-model="val" name=”angular”>  </form> |
| 4,ng-bind | The ng-bind attribute tells AngularJS to replace the text content of the specified HTML element with the value of a given expression  It is preferable to use  ng-bind instead of {{ expression }} | Ex:  <span ng-bind=”hi”></span> |
| 5,ng-repeat | Used to iterate the list of elements from collection | Ex:  <ng-repeat=”x in [1,2,3,4]”>  {{x}} |
| 6,ng-click | The ng-click directive allows you to specify custom behavior when an element is clicked. | Ex: <ng-click  ng-click="expression">  ...  </ng-click> |
| 7, ng-options | The ng-options attribute can be used to dynamically generate a list of <option> elements for the <select>element | Ex:  <select ng-model=”my\_model” ng-options=”x.name as x.id for x in data”></select> |
| 8,ng-init | Used to initialize the application data syntactically.  We can initialize the data in the form of key and value | Ex:  <ng-init “key1=Hi;key2=Hello”> |
| 9, ng-show | The ng-show directive shows or hides the given HTML element based on the expression provided to the ng-show attribute. | Ex:  <div ng-show="myValue"></div> |
| 10,ng-switch | Used to write the “switch cases ” and “default cases” in angular application | Ex:  <span ng-switch="expression">  <span ng-switch-when="matchValue1">...</span>  <span ng-switch-when="matchValue2">...</span>  <div ng-switch-default>...</div>  </span> |
| 11,ng-if | Used to write the conditions in view | Ex:  <div ng-repeat=’x in data’>  <div ng-if =”x!=’hello1’&& ‘hello5’”>  {{x}}  </div>  </div> |
| 12,ng-cloak | Used to avoid unparsed data in two way data binding | Ex:  <div ng-cloak style=”color:blue” ng-controller=”c1”>  {{var1}}  </div> |
| 13,ng-class-even  ng-class-odd | Used to apply css classes to the even rows of table  Used to apply css classes to the odd rows of table | Ex:  <ng-repeat =”x in data” ng-class-even=’even’”>  <ng-repeat =”x in data” ng-class-odd=’odd’”> |
| 14,ng-include | Used to split the view into number of sub views | <div ng-include=”view.html”>  </div> |
| 15,ng-dblclick | The ng-dblclick directive allows you to specify custom behavior on a double click event. | Ex:  <button ng-dblclick="count = count + 1" ng-init="count=0">  Increment (on double click)  </button>  count: {{count}} |
| 16,ng-submit | Used to submit the form to the controller | Ex:  <form ng-submit=”user defined functions”>  //form elements  <input type=””submit”>  </form> |
| 17,ng-mousedown  ng-mouseover  ng-mouseleave | Whenever we are using down operation automatically framework will execute user defined functions  Specify custom behavior on mouse over event.  Specify custom behavior on mouse leave event. | <div ng-mousedown=”down()”>  <div ng-mouseover=”over()”>  <div ng-mouseleave=”leave()”> |

**4. Dependencies for routing**

Loading the Target Templates to the Source Templates without refreshing is called as Routing in Single Page Application

In AngularJs Routing can be done in two ways**:**

* + - * + **ngRoute Module**
        + **ui.Router Module**

**ngRoute Module:** The ngRoute Module routes  your application to different pages without reloading the entire application.

**Ui.Router Module:** The **UI**-**Router** is a **routing** framework for **AngularJS** built by the AngularUI team. It provides a different approach than ngRoute in that it changes your application views based on state of the application and not just the route URL ( i.e. ngroute just route the url and ui router changes the view based on state of application)

**Development of Single Page Application By Using ngRoute Module.**

* + - * + ngRoute is the Predefined Module in AngularJS
        + ngRoute is the Native Module in AngularJS

TO Download the ngRoute module by using bower.

**bower.json-**

dependencies:{

"angular":"~1.5.0",

"angular-route":"~1.5.0"

}

TO Download the ui-router module by using bower

**bower.json -**

dependencies:{

"angular":"~1.6.0",

"angular-ui-router":"~0.2.18"

}

**Differences Between ngRoute and ui.router module.**

* + - * + ngRoute module is "native module" in angularJS

**var app = angular.module("app",["ngRoute"]);**

* + - * + ui.router module is the 3rd party module.

**var app=angular.module("app",["ui.router"]);**

* + - * + ngRoute won't supports the Nested Views in Single Page Applications.
        + ui.router supports the Nested Views in Single page Application.
        + ngRoute won't supports the Named Views in Single Page Application.
        + ui.router supports the Named Views in Single page Application.
        + ngRoute won't Supports the "Objects passing" as URL Parameters
        + ui.router supports the "Objects" as the URL Parameters
        + ngRoute in bower

angular-route:'~1.5.0'

* + - * + ui.router in bower

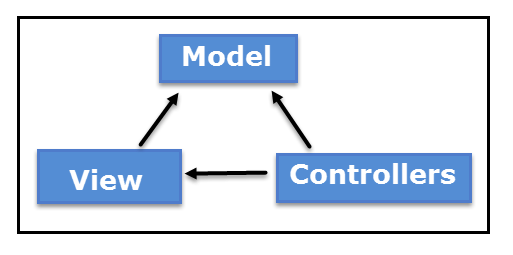
angular-ui-router:"~0.2.18"

**5. What is MVC?**

* + - **MVC is a design pattern used to isolate business logic from presentation.**
    - The MVC pattern has been given importance by many developers as a useful pattern for the reuse of object code and a pattern that allows them to reduce the time it takes to develop applications with user interfaces.
    - The Model-View-Controller (MVC) is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller. Each of these components are built to handle specific development aspects of an application. MVC is one of the most frequently used industry-standard web development framework to create scalable and extensible projects.

MVC Components

* + - **Model:** The Model component corresponds to all the data related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic related data. For example, a Customer object will retrieve the customer information from the database, manipulate it and update it data back to the database or use it to render data.
    - **View:** The View component is used for all the UI logic of the application. For example, the Customer view would include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with.
    - **Controller:** Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output. For example, the Customer controller would handle all the interactions and inputs from the Customer View and update the database using the Customer Model. The same controller would be used to view the Customer data.

****

**6. Ng-init  directive**

-> ng-init directive is used to intialize the application data Statically.

-> ng-init directive is used to define a local variable with value.

-> It is supported by all html elements.

Syntax:  <element ng-init="expression">                        </element>

Example:

          <div ng-app="app "

                 ng-init="name='naresh it' ">

               <h1>{ { name } } </h1>

          </div>

Note:In this example we are giving  'name' as local variable.

**7. Difference between $scope and   $rootscope.**

-> $scope is a glue between controller.   and view.

    $rootscope is a parent object of all $scope angular abjects created in a     web page.

->$scope is created with ng-controller.   $rootscope is created with ng-app.

->Each angular application has exactly one rootscope, but may have several    child scopes.

->The s$cope will be available only inside the controller. But we can access $rootscope in all  the controllers.

**8. What is Internationalization**

**Internationalization**  is the process of developing products in such a way that they can be localized for languages and cultures easily. Localization (l10n), is the process of adapting applications and text to enable their usability in a particular cultural or linguistic market.

We implement internationalization in two ways

**i18n** and **l10n**

**9. How to implement Internationalization in Angular Js?**

---> To implement Internationalization in angularjs, the angular-translate is an AngularJS module.

--->that brings i18n (internationalization) and l10n (localization) into your Angular app.

--->It allows you to create a JSON file that represents translation data as per language.

--->The angular-translate library (angular-translate.js) also provides built-in directives and filters that make the process of internationalizing simple.

i18n and L10n

First we need to install:

bower install angular-i18n

example:

<html>

<head>

<title>Angular JS Forms</title>

</head>

<body>

<h2>AngularJS Sample Application</h2>

<div ng-app = "mainApp" ng-controller = "StudentController">

{{fees | currency }} <br/><br/>

{{admissiondate | date }} <br/><br/>

{{rollno | number }}

</div>

<script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>

<!-- <script src = "https://code.angularjs.org/1.3.14/i18n/angular-locale\_da-dk.js"></script> -->

<script>

var mainApp = angular.module("mainApp", []);

mainApp.controller('StudentController', function($scope) {

$scope.fees = 100;

$scope.admissiondate = new Date();

$scope.rollno = 123.45;

});

</script>

</body>

</html>

**output:**

AngularJS Sample Application

$100.00

Feb 27, 2017

123.45

**10. Types of data binding in angularjs?**

The **data binding** is the data synchronization processes between the model and view components

In angularjs when model data got changed that time the view data will change automatically and vice versa.

We have two types of data bindings available in angularjs those are

                1. One-Way data binding

                2. Two-Way data binding

**One-Way data binding**

* In **One-Way** data binding, view will not update automatically when data model changed.
* we need to write custom code to make it updated every time.
* Its not a synchronization processes and it will process data in one.

**Two-way Data Binding**

In **Two-way** data binding, view updates automatically when data model changed.

* Its synchronization processes and two way data binding.
* This two-way data binding using [ng-model](http://tutlane.com/tutorial/angularjs/angularjs-ng-model-directive-with-example" \t "_blank" \o "Angularjs ng-model directive to bind data to html elements with example) directive.
* If we use ng-model directive in html control it will update value automatically whenever data got changed in input control.

11 -2 Same Questions.

12. 12 - 5 Same Questions.

13. 13-43 same question **How many types of data binding in AngularJs?**

## AngularJS Data Binding

The **data binding** is the data synchronization processes that work between the model and view components. In Angular, model treat as source of application and view is the projection of angular model.

In angularjs when model data got changed that time the view data will change automatically and vice versa.

We have two types of data bindings available in angularjs those are

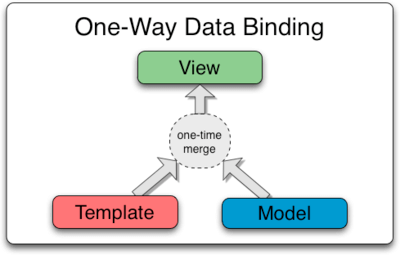
                1. One-Way data binding

                2. Two-Way data binding

We will learn each binding in detail with examples in angularjs.

AngularJS One-Way Data Binding

In **One-Way** data binding, view (UI part) not updates automatically when data model changed and we need to write custom code to make it updated every time. Its not a synchronization processes and it will process data in one way that will be like as shown following image.



We will see simple example of using one way data binding in angularjs.

AngularJS One-way Data Binding Example

Following is the example of using one way data binding in angularjs application.

[Live Preview](http://tutlane.com/example/angularjs/angularjs-one-way-data-binding-example" \t "_blank" \o "Angularjs one way data binding example)

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>

AngularJs One way Binding Example

</title>

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

<script type="text/javascript">

var app = angular.module('angularonebindapp', []);

app.controller('angularonebindingCtrl', function ($scope) {

$scope.name = 'Welcome to one way binding';

});

</script>

</head>

<body ng-app="angularonebindapp">

<div ng-controller="angularonebindingCtrl">

<p>

Message:   {{ name }}

</p>

</div>

</body>

</html>

Here if you observe above code we are binding [model](http://tutlane.com/tutorial/angularjs/angularjs-ng-model-directive-with-example" \t "_blank" \o "Angularjs ng-model directive with example) values to html elements using data bindings but html elements it won't change the values in [model](http://tutlane.com/tutorial/angularjs/angularjs-ng-model-directive-with-example" \t "_blank" \o "Angularjs ng-model directive with example) its one way data binding. Now run application and see the output.

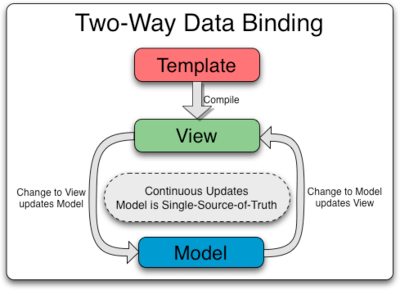
Output of AngularJS One-way Data Binding

Following is the output of angularjs one way data binding

Message: Welcome to one way binding

AngularJS Two-way Data Binding

In **Two-way** data binding, view (UI part) updates automatically when data model changed. Its synchronization processes and two way data binding that will be like as shown following image.



We can achieve this two-way data binding using [ng-model](http://tutlane.com/tutorial/angularjs/angularjs-ng-model-directive-with-example" \o "Angularjs ng-model directive to bind data to html elements with example" \t "_blank) directive. If we use ng-model directive in html control it will update value automatically whenever data got changed in input control.

**How does data binding work in the AngularJs?**

In AngularJs, it will remember present values and compare it with previous value. If any changes found in previous value that time change event will fire automatically and it will update data every time when data got changed.

The HTML tells the AngularJs compiler to create the **$watch** for controller methods and its run inside the **$apply** method. We will see simple example for two way data binding in angularjs.

AngularJS Two-way Data Binding Example

Following is the example of binding data in two way in angularjs applications.

[Live Preview](http://tutlane.com/example/angularjs/angularjs-two-way-data-binding-example" \t "_blank" \o "Angularjs two way data binding example)

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>

AngularJs Two Binding Example

</title>

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

<script type="text/javascript">

var app = angular.module('angulartwobindapp', []);

app.controller('angulartwobindingCtrl', function ($scope) {

$scope.name = 'Welcome to two way binding';

});

</script>

</head>

<body ng-app="angulartwobindapp">

<div ng-controller="angulartwobindingCtrl">

Enter Name : <input type="text" ng-model="name" style="width:250px" />

<p>

Entered Name:   {{ name }}

</p>

</div>

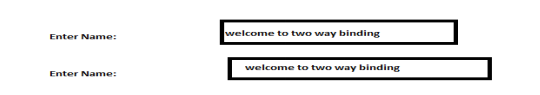
</body>

</html>

If you observe above code we defined [ng-model](http://tutlane.com/tutorial/angularjs/angularjs-ng-model-directive-with-example" \o "Angularjs ng-model directive to bind data to html elements with example" \t "_blank) objective to html control and used same [ng-model](http://tutlane.com/tutorial/angularjs/angularjs-ng-model-directive-with-example" \o "Angularjs ng-model directive to bind data to html elements with example" \t "_blank) value to show input control value. Here whenever we change input control value automatically the appearance value also will get changed. Now we will run and see the output.

Output of AngularJS Two-way Data Binding

Following is the result of two way data binding in angularjs application



**14. what is the need of ng-controller and ng-model in Angular?**

## The ng-model Directive

With the ng-model directive you can bind the value of an input field to a variable created in AngularJS.

### Example

<div ng-app="myApp" ng-controller="myCtrl">  
    Name: <input ng-model="name">  
</div>  
  
<script>  
var app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope) {  
    $scope.name = "John Doe";  
});  
</script>

AngularJS controllers **control the data** of AngularJS applications.

AngularJS controllers are regular **JavaScript Objects**.

## AngularJS Controllers

AngularJS applications are controlled by controllers.

The **ng-controller** directive defines the application controller.

A controller is a **JavaScript Object**, created by a standard JavaScript **object constructor**.

### AngularJS Example

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

|  |  |
| --- | --- |
|  | **15. $watch will watch changes which are done within the scope**  The AngularJS $watch is used to observe properties on a single object and notify you if one of them changes.  In other word watch is shallow watches the properties of an object and fires whenever any of the properties change.  This method is for watching changes of scope variables.  This method has callback function which gets called when the watching properties are changed.  example:  <!doctype html>  <html ng-app="app" ng-controller="ctrl">  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.0.1/angular.min.js"></script>  <input type="text" ng-model="expression">  <script>  var app=angular.module("app",[]);  app.controller("ctrl",ctrl);  ctrl.$inject=["$scope"]  function ctrl($scope){  $scope.$watch("expression",function(  newValue,oldValue,scope ){  console.log("newValue="+ newValue);  console.log("oldValue="+ oldValue);  console.log(scope.expression);  });  }  </script>  </html>  **16.What is the difference between $parse and $eval?**  $eval:  The AngularJS $eval method is used to executes the AngularJS expression on the current scope and returns the result.  In AngularJS, expressions are similar to JavaScript code snippets that are usually placed in bindings such as {{ expression }}.  AngularJS used internally $eval to resolve AngularJS expressions, such as {{variable}}.  The important difference between $eval and $parse is that $eval is a scope method that executes an expression on the current scope,  while $parse is a globally available service. The $eval method takes AngularJS expression and local variable object as parameter.  $eval takes an angular expression, evaluates it and returns the actual result of that expression.  for example1:  $scope.a = 2;  var result = $scope.$eval('1+1+a');  // result is 4  example2:  <!DOCTYPE html>  <html lang="en" ng-app="app" ng-controller="ctrl">  <script src="bower\_components/angular/angular.min.js"></script>  <div>  <h1>a={{a}}</h1><br>  <h1>b={{b}}</h1><br>  <h1>Result={{a \* b}}</h1>  </div>  <button ng-click="clickMe()">DEMO </button>  <script>  var app=angular.module("app",[]);  app.controller("ctrl",ctrl);  ctrl.$inject=["$scope"];  function ctrl($scope) {  $scope.a = 10;  $scope.b = 20;  $scope.emp={  ename:"rk",age:24  };  $scope.clickMe = function () {  var r = $scope.$eval("a\*b")  alert("result:" + r);  var r2=$scope.$eval("emp.ename");  alert(r2);  var r3=$scope.$eval("a\*b\*3");  alert(r3);  var r4=$scope.$eval("a\*b\*3\*c");  alert(r4);  var r5=$scope.$eval("a\*b\*3\*c",{c:5});  alert(r5);  var r6=$scope.$eval("a\*b\*3\*c",{a:2,c:5});  alert(r6);  var r=$scope.$eval(function($scope,locals){  return $scope.a\*$scope.b  })  alert(r);  var r=$scope.$eval(function($scope,locals){  return $scope.a\*$scope.b\*locals.c},  {a:2,b:3,c:6});  alert(r);  };  }  </script>  </html>  $parse:  $parse does not require scope. It takes an expression as a parameter and returns a function.  The function can be invoked with an object that can resolve the locals:  $parse takes an angular expression and returns a function that represents that expression.  For example1:  var fn = $parse('1+1+a');  var result = fn({ a: 2 });  // result is 4  example2:  <!DOCTYPE html>  <html lang="en" ng-app="app" ng-controller="ctrl">  <script src="bower\_components/angular/angular.min.js"></script>  <div>  <h1>a={{a}}</h1><br>  <h1>b={{b}}</h1><br>  <h1>Result={{a \* b}}</h1>  {{emp.ename}}  </div>  <button ng-click="clickMe()">parseDemo </button>  <script>  var app=angular.module("app",[]);  app.controller("ctrl",ctrl);  ctrl.$inject=["$scope","$parse"];  function ctrl($scope,$parse) {  $scope.a = 10;  $scope.b = 20;  $scope.emp = {  ename: "rk", age: 24  };  $scope.clickMe = function () {  // var my\_fun = $parse("a\*b"); //return a function  // var r = my\_fun($scope);  // alert("result=" + r);  //alert("result="+ $parse("a\*b")($scope)  // alert("result="+ $parse("a\*b")({a:2,b:3}));  // var my\_fun = $parse("a\*b"); //return a function  // var r1 = my\_fun($scope);  // alert("result=" + r1);  // var r2=my\_fun({a:2,b:3});  // alert("result=" + r2);  // alert($parse("emp.ename")($scope));  ($parse("emp.ename").assign($scope,"satti"));  // alert($parse("emp.ename")($scope));  }  }  </script>  **17.Explain $scope in angular?**  The scope is the binding part between the HTML (view) and the JavaScript (controller).  The scope is an object with the available properties and methods.  The scope is available for both the view and the controller.  How to Use the Scope:  When you make a controller in AngularJS, you pass the $scope object as an argument:  Example:  Properties made in the controller, can be referred to in the view:  <div ng-app="myApp" ng-controller="myCtrl">  <h1>{{carname}}</h1>  </div>  <script>  var app = angular.module('myApp', []);  app.controller('myCtrl', function($scope) {  $scope.carname = "Volvo";  });  </script>  When adding properties to the $scope object in the controller, the view (HTML) gets access to these properties.  In the view, you do not use the prefix $scope, you just refer to a propertyname, like {{carname}}.  Understanding the Scope:  If we consider an AngularJS application to consist of:  View, which is the HTML.  Model, which is the data available for the current view.  Controller, which is the JavaScript function that makes/changes/removes/controls the data.  Then the scope is the Model.  The scope is a JavaScript object with properties and methods, which are available for both the view and the controller.  Example:  If you make changes in the view, the model and the controller will be updated:  <div ng-app="myApp" ng-controller="myCtrl">  <input ng-model="name">  <h1>My name is {{name}}</h1>  </div>  <script>  var app = angular.module('myApp', []);  app.controller('myCtrl', function($scope) {  $scope.name = "Ramakrishna";  });  </script>  **18. is angularjs code is reusable**  I'm using Angular to develop commenting functionality for a web app. Currently there are two sections in the application were a user can comment:  Category  Product  About 90% of the commenting functionality is the same for both sections and as such I would like to make this reusable - i.e write some service or controller that I can reference/use as a base.  So far, my research seems to point to using a factory service but unfortunately this doesn't seem to work (I've spent the whole day running through various tutorials).  It is quite possible that I am over thinking this and making it far too complicated but I honestly don't know which way to turn anymore.  Herewith a quick and dirty overview of what I have so far:  HTML view for the category  Controller for the category (receives data from service and posts data to service in order to bind data to model)  Service for the category (retrieve and stores all the necessary data)  The product uses the same logic and a lot of the code in the service and controller will be duplicated. I've merged the two services into one service successfully but I'm having trouble doing the same for the controller.  Do I:  Write a base controller that will communicate with the above mentioned service and that will hookup with the two existing controllers  OR  Write a factory/provider service that hooks up to the two existing controllers as well as the above mentioned service. |

**19. in our programming we are writing service as function that is that is to achieve reusability**

**20. what is the difference between $watch and $observe**

$observe() is a method on the Attributes object, and as such, it can only be used to observe/watch the value change of a DOM attribute.

It is only used/called inside directives. Use $observe when you need to observe/watch a DOM attribute that contains interpolation (i.e., {{}}'s).

E.g., attr1="Name: {{name}}", then in a directive: attrs.$observe('attr1', ...).

(If you try scope.$watch(attrs.attr1, ...) it won't work because of the {{}}s -- you'll get undefined.) Use $watch for everything else.

$watch() is more complicated. It can observe/watch an "expression", where the expression can be either a function or a string.

If the expression is a string, it is $parse'd (i.e., evaluated as an Angular expression) into a function.

(It is this function that is called every digest cycle.) The string expression can not contain {{}}'s.

$watch is a method on the Scope object, so it can be used/called wherever you have access to a scope object, hence in

# a controller -- any controller -- one created via ng-view, ng-controller, or a directive controller

# a linking function in a directive, since this has access to a scope as well

Because strings are evaluated as Angular expressions, $watch is often used when you want to observe/watch a model/scope property.

E.g., attr1="myModel.some\_prop", then in a controller or link function: scope.$watch('myModel.some\_prop', ...) or

scope.$watch(attrs.attr1, ...) (or scope.$watch(attrs['attr1'], ...)).

(If you try attrs.$observe('attr1') you'll get the string myModel.some\_prop, which is probably not what you want.)

As discussed in comments on @PrimosK's answer, all $observes and $watches are checked every digest cycle.

Directives with isolate scopes are more complicated. If the '@' syntax is used, you can $observe or $watch a DOM attribute that contains interpolation (i.e., {{}}'s).

(The reason it works with $watch is because the '@' syntax does the interpolation for us, hence $watch sees a string without {{}}'s.)

To make it easier to remember which to use when, I suggest using $observe for this case also.

To help test all of this, I wrote a Plunker that defines two directives. One (d1) does not create a new scope, the other (d2) creates an isolate scope.

Each directive has the same six attributes. Each attribute is both $observe'd and $watch'ed.

<div d1 attr1="{{prop1}}-test" attr2="prop2" attr3="33" attr4="'a\_string'"

attr5="a\_string" attr6="{{1+aNumber}}"></div>

Look at the console log to see the differences between $observe and $watch in the linking function.

Then click the link and see which $observes and $watches are triggered by the property changes made by the click handler.

Notice that when the link function runs, any attributes that contain {{}}'s are not evaluated yet (so if you try to examine the attributes, you'll get undefined).

The only way to see the interpolated values is to use $observe (or $watch if using an isolate scope with '@').

#$observe is used in linking function of directives.

#$watch is used on scope to watch any changing in its values.

Keep in mind : both the function has two arguments,

$observe/$watch(value : string, callback : function);

value : is always a string reference to the watched element (the name of a scope's variable or the name of the directive's attribute to be watched)

callback : the function to be executed of the form function (oldValue, newValue)

Therefore, getting the values of these attributes is an asynchronous operation. (And this is why we need the $observe and $watch functions.)

**21. Differentiate between named routing and custom directives?**

|  |  |
| --- | --- |
| Custom directives | routing |
| 1. Creating your own directive Based on application requirement is called custom directive.  2. We can achieve custom directives in three levels.  Element level.  Attribute level.  CSS class level.  3. Here we don’t need any objects or services to use above levels.  4. Most advantage is to reuse the custom directive in the place where ever we need.  5. ng-controller directive is the connection between template(vie w) and its controller(model). | 1. Loading the target templates without refreshing web page is called as routing in single page application.  2. We can achieve named routing in two ways.  Ng-route.  Ui-router.  3. Here we need to pass objects to use the services like$route Provider, $state provider.  4. There is no necessity of reusing routing because at the time of loading web pages in the browser it also loads its resources like html, css, js files etc without refreshing the source content.  5.Mapping with key and value pairs is the connection between target(view) and its controllers(model), This is done in sconfiguration file, Mapping will done before launching web pages. |

Custom directive scenario:Instead of coding common part in every view. Let’s create a separate html file for common part and declare it as custom directive, then use it as custom directive in every html file which Leads to less time consuming, better understanding, increases reusability.22. List at least three ways to communicate between modules of your application using angular js functionality?

A module can be injected into another module, in which case the container module has access to all elements of the injected module. If you look at angular seed project, modules are created for directive, controllers, filters etc, something like this

angular.module("myApp", ["myApp.filters", "myApp.services", "myApp.directives", "myApp.controllers"]) This is more of a re usability mechanism rather than communication mechanism.

The second option is would be to use services. Since services are singleton and can be injected into any controller, they can act as a communication mechanism.

Third option is to use parent controller using $scope object as it is available to all child controllers.

You can also inject $rootScope into controllers that need to interact and use the $broadcast and $on methods to create a service bus pattern where controllers interact using pub\sub mechanism.

**23. Different types of errors in Angular js? Can we do or create custom errors?**

A. Errors:

Some Namespace errors are given below.

ng- this is the error in ng-namespace.

|  |  |
| --- | --- |
| Name | Description |
| [badident](https://docs.angularjs.org/error/ngRepeat/badident) | Invalid identifier expression |
| [dupes](https://docs.angularjs.org/error/ngRepeat/dupes) | Duplicate Key in Repeater |
| [iexp](https://docs.angularjs.org/error/ngRepeat/iexp) | Invalid Expression |
| [iidexp](https://docs.angularjs.org/error/ngRepeat/iidexp) | Invalid Identifier |

Ng-repeat- list of errors in the ngRepeat namespace.

|  |  |
| --- | --- |
| [areq](https://docs.angularjs.org/error/ng/areq) | Bad Argument |
| [badname](https://docs.angularjs.org/error/ng/badname) | Bad `hasOwnProperty` Name |
| [btstrpd](https://docs.angularjs.org/error/ng/btstrpd) | App Already Bootstrapped with this Element |
| [cpi](https://docs.angularjs.org/error/ng/cpi) | Bad Copy |
| [cpta](https://docs.angularjs.org/error/ng/cpta) | Copying TypedArray |
| [cpws](https://docs.angularjs.org/error/ng/cpws) | Copying Window or Scope |
| [test](https://docs.angularjs.org/error/ng/test) | Testability Not Found |

Ng-model: list of errors in the ngModel namespace.

|  |  |
| --- | --- |
| Name | Description |
| [constexpr](https://docs.angularjs.org/error/ngModel/constexpr) | Non-Constant Expression |
| [datefmt](https://docs.angularjs.org/error/ngModel/datefmt) | Model is not a date object |
| [nonassign](https://docs.angularjs.org/error/ngModel/nonassign) | Non-Assignable Expression |
| [nopromise](https://docs.angularjs.org/error/ngModel/nopromise) | No promise |
| [numfmt](https://docs.angularjs.org/error/ngModel/numfmt) | Model is not of type `number` |

Ng-options: list of errors in the ngOptions namespace.

|  |  |
| --- | --- |
| Name | Description |
| [iexp](https://docs.angularjs.org/error/ngOptions/iexp) | Invalid Expression |

$q: list of errors in the $q namespace.

|  |  |
| --- | --- |
| Name | Description |
| [norslvr](https://docs.angularjs.org/error/$q/norslvr) | No resolver function passed to $Q |
| [qcycle](https://docs.angularjs.org/error/$q/qcycle) | Cannot resolve a promise with itself |

The TypeError object represents an error when a value is not of the expected type.

Uncaught reference error: //didn’t got the answer.

Custom errors:

$error: Is an object hash, containing references to all invalid controls or forms, where:

keys are validation tokens (error names)

values are arrays of controls or forms that are invalid with given error.

**24. what is the difference between ngshow and ngif and Ng hide?**

ng-Show/nghide

--------------

===>ngShow directive is a part of module ng.

===>It is used to show or hide the elements based on boolean values.

===>The ng-show directive shows the specified HTML element if the expression

evaluates to true, otherwise the HTML element is hidden.

==>ngShow/ngHide work with the show and hide events that are triggered when the directive expression is true and false in Animations

==>The great part about these directives is that "we don’t have to do any of the showing or hiding ourselves with CSS or JavaScript".

===>Supported by all HTML elements.

==>Both ng-show and ng-if receive a condition and hide from view the directive’s element

in case the condition evaluates to false.

==>ng-show (and its sibling ng-hide) toggle the appearance of the element by

adding the CSS display: none style.

==>The ng-hide directive hides the HTML element if the expression evaluates to true.

==>ng-hide is also a predefined CSS class in AngularJS, and sets the element's display to none.

Syntax:

------

<element ng-show="expression"></element>

Parameter:

---------

Value Description

----- -----------

expression An expression that will show the element only if the expression returns true.

Example:

-------

<!DOCTYPE html>

<html>

<body ng-app="">

Show HTML: <input type="checkbox" ng-model="myVar">

<div ng-show="myVar">

<h1>Welcome</h1>

<p>Welcome to my Show Directive.</p>

</div>

</body>

</html>

**25. How to implement securities in angular js? And what do u mean by IAutho?**

**Security**

**-**-------

==>Security is one of the most important parts of writing a web application—perhaps

the most important part!

==>Angular is not made to enhance security of our web application but to help

the web application run smooth and user-friendly.

==>Do not mix client and server templates

==>Do not use user input to generate templates dynamically

==>Do not run user input through $scope.$eval (or any of the other expression parsing functions)

==>You can not access a angular variable/scope from console of your browser .

==>It Prevents cross-side-scripting attacks.

==>Prevents HTML injection attacks.

==>NG-CSP directive is given in angular to stop injecting inline codes to the application.

==>you have to send every http using ajax and you need to have an api for the back-end.

==>When developers build client apps with server back ends they approach the application

as though they control the entire ecosystem.

==>Assumptions are often made that the client they built will only ever talk to the server side APIs

they built in the way they designed them.

==>HTTP Requests, Whenever your application makes requests to a server there are potential security

issues that need to be blocked.

==>Using Local Caches, There are various places that the browser can store (or cache) data.

==>Within AngularJS there are objects created by the $cacheFactory.

These objects, such as $templateCache are used to store and retrieve data,

==>primarily used by $http and the script directive to cache templates and other data.

==>Similarly the browser itself offers localStorage and sessionStorage objects for caching data.

==>Attackers with local access can retrieve sensitive data from this cache even when users are not authenticated.

For instance in a long running Single Page Application (SPA), one user may "log out",

but then another user may access the application without refreshing,

in which case all the cached data is still available.

26.Differentiate named routing n custom directives...?

**21st & 26th same**

**27. what is AngularJs?**

* AngularJs is a client side UI frame work and also Client side JavaScript Framework for building Dynamic web application,it fallows MVC(Model View Controller)pattern.
* It has set of ready to use modules to simplify building of Single Page Applications.

**28. Advantages of AngularJS?**

* AngularJS supports MVC Pattern
* It supports two way data-binding
* It supports both client and server communication
* It supports Animation
* Event handles
* Supports Rest full services
* Supports Static and Angular templates

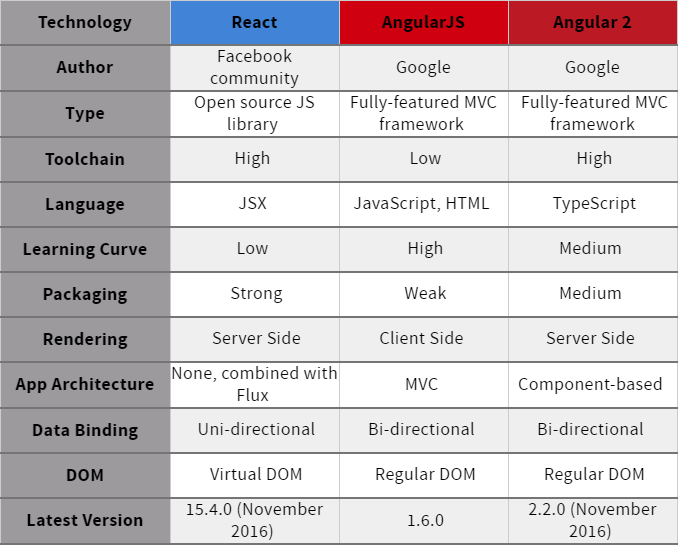
AngularJS provides reusable components.

**29. Disadvantages of AngularJs?**

* Not security
* Difficult to understanding the Angular Application because of no coding standards given by venders
* Client should enable the JavaScript then only you can run the Angular Applications
* More than 2000 watchers can severely lag the UI. That limits the complexity of your Angular forms, especially big data grids and lists.

**30. 30- 2 Same Questions**

**31.Difference AngularJS , Angular 2 and ReactJS**



**React vs Angular comparison: library against framework**

Angular is one of the most popular javascript frameworks and like other similar software suits it offers multiple out-of-the-box solutions and designs. At a time when, React ecosystem comprises any number of composable, targeted online tools and ReactJS acts as one of the building blocks.

**Advantages of angularjs**

* **Global community support**is one of the factors, that can easily make Angular the best javascript framework. Developers and designers constantly collaborate and contribute to the community, increasing credibility and reliability of the framework.
* It is a **full-fledged framework** that can run in any browser or platform. Moreover, it is consistent, overwhelmed with ready-made tools, ng components are robust and quite mature, as contrasted with React.
* **Two-way data bind** is probably the top feature, as it diffuses the impact after every minor data change and does way with the need for further effort with data sync in view and model.

Given the fact that our company makes active use of ng2, it is essential to include react vs angular 2 comparison as well.

* **TypeScript**is an enhanced JS super-set that supplies optional static type checking, object-based programming patterns, and high-performance typing features.
* Owing to **component-based architecture** components have deep binding and each of them comprises elements with only relevant functionality. What is more, they are loosely coupled and properly encapsulated. Such approach makes components easily reusable, enhance their testability and further maintainability.

**Advantages of reactjs**

* **JSX**is a JS syntax that enables HTML quotes and usage of HTML tag syntax for subcomponents rendering. It promotes building of machine-readable code and provides ability to compound components in one compile-time verified file.
* **Prompt rendering** is among the best features of React that gives a significant edge over Angular. The technology comprises smart methods to mitigate the amount of DOM operations, optimize and accelerate the updates process. Virtual DOM (Document Object Model) is of great use while handling vast databases.
* The core difference between reactjs and angularjs is that React is **JS-centric**, while ng2 remains HTML-centric. JavaScript is far more robust, than HTML, that makes React far more simple, focused and consistent.

**Disadvantages of angularjs**

* Despite a comprehensive and clear manual, **steep learning curve** and **complexity**are named among the main weak points of Angular.js. Like any other client-side rendering technology in javascript framework comparison list, programmers should place special emphasis on security to make apps reliable and safe. Though, with introduction of **Angular Universal** and pre-rendering option in ng2 this issue was defused.

**Disadvantages of react.js**

* Comparing react vs angular performance, first of all it’s worth mentioning that reactJS is **not a full-scale framework** and for this very reason integration of the UI library into a common MVC framework requires deeper programming knowledge. It is still young and not mature, considering tutorial volumes, limited ecosystem, etc.
* Apart from pros and cons of reactjs, we should also mention **Flux**that is frequently applied for adding a structure and architecture to react app. Usage of both technologies can become a challenge for non-experienced programmer, as it lacks structured and comprehensive documentation or guide.

**Conclusion**

React and Angular offer completely diverse approaches to [web application development](http://da-14.com/services/web-app-development" \t "_blank) for startup, small and midmarket businesses. Both technologies are powerful and flexible, while none of them is worse or better, than the other. Depending upon custom app goals and particular system constraints, developers can run from ng2 to React, and back.

Opting for Angular, it usually assumes the creation of core skeleton for a front-end app, whereas React.js can be applied to improve its specific parts. Moreover, it can be integrated with other frameworks, like Backbone or even well-known Angular.

**32.Difference between $scope and $rootscope?**

* The scope is the binding part between the HTML (view) and the JavaScript (controller).
* The scope is an object with the available properties and methods.
* The scope is available for both the view and the controller.
* When you make a controller in AngularJS, you pass the $scope object as an argument
* When adding properties to the $scope object in the controller, the view (HTML) gets access to these properties.

If we consider an AngularJS application to consist of:

* View, which is the HTML.
* Model, which is the data available for the current view.
* Controller, which is the JavaScript function that makes/changes/removes/controls the data.

**$Scope**

Then the scope is the Model.

The scope is a JavaScript object with properties and methods, which are available for both the view and the controller.

<div ng-app="myApp" ng-controller="myCtrl">  
  
<h1>{{carname}}</h1>  
  
</div>  
  
<scriptdr>

var app = angular.module('myApp', []);  
  
app.controller('myCtrl', function($scope) {  
    $scope.carname = "Volvo";  
});

</script>

## Root Scope

All applications have a $rootScope which is the scope created on the HTML element that contains the ng-app directive.

The rootScope is available in the entire application.

If a variable has the same name in both the current scope and in the rootScope, the application use the one in the current scope.

<body ng-app="myApp">  
  
<p>The rootScope's favorite color:</p>  
<h1>{{color}}</h1>  
  
<div ng-controller="myCtrl">  
    <p>The scope of the controller's favorite color:</p>  
    <h1>{{color}}</h1>  
</div>  
  
<p>The rootScope's favorite color is still:</p>  
<h1>{{color}}</h1>  
  
<script>

var app = angular.module('myApp', []);  
app.run(function($rootScope) {  
    $rootScope.color = 'blue';  
});  
app.controller('myCtrl', function($scope) {  
    $scope.color = "red";  
});  
</script>  
</body>

**Output:**

The rootScope's favorite color:

# blue

The scope of the controller's favorite color:

# red

The rootScope's favorite color is still:

# blue

**33.Difference between {{ }} and ng-bind?**

To bind our data to the HTML page, [Angular](http://www.ufthelp.com/p/angular-js.html" \t "_blank)gives the ability to use **{{}}** known as double curly brackets or **ng-bind**.Both ways are performing the same functionality.

**{{}}**

**Example:-**[Double Curly brackets in Angular JS](http://2.bp.blogspot.com/-jN5BxG1maJo/VkJfssf2ggI/AAAAAAAAB9E/kDHJQ5IlSDc/s1600/curly-brackets.png)

*name*is JS object or some variable we are referring to.

It makes your template very readable i.e Anyone going through a code such as **{{ name }}**can easily understand that **"name"** is a variable bounded to the DOM.

**Drawbacks of {{}}**

Sometimes when we load our application in the browser , we can notice flashing content for some milliseconds before **{{ name }}**is resolved and data is loaded.

|  |
| --- |
| [flashing-content](http://1.bp.blogspot.com/-dNJ9lDsc0Ao/VkJU_g6YmwI/AAAAAAAAB8o/C_t1zhef7JE/s1600/flashing-content.png) |

This happens because the template is loaded before AngularJS had a chance to go in and compile the elements. To resolve this issue, you can use **[ng-cloak](https://docs.angularjs.org/api/ng/directive/ngCloak" \t "_blank)**directive.

**ng-bind**  is used inside an HTML DOM element:

**Example:-**

[ng-bind](http://2.bp.blogspot.com/-EDWq9v3WfU8/VkJfZUxhOyI/AAAAAAAAB84/wVq0LV_pDHA/s1600/ng-bind.png)

We are using the same expression which we used for double curly brackets.

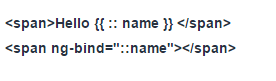
The **major difference** between **ng-bind and {{}}** is that ng-bind creates a watcher for variable passed to it(i.e. name as in above example), while curly brackets({{}}) will (store the entire expression in memory i.e. }perform dirty-checking and refreshing the expression in every digest cycle even if it is not required.

**ng-bind**will only apply when the value passed is changing actually.

In the first approach(i.e. {{}}), AngularJS evaluates the expression then replaces it with some value which sometime be left with the flashing double curly brackets but **ng-bind** saves this time by informing AngularJS to put the contents of the expression within the element itself.

**Note:-**

* {{}} sometimes cause performance issue if we have thousand of bindings in our page.In these scenario's, we should go with **ng-bind.**
* For some modules, like [translate module](https://github.com/angular-translate/angular-translate" \t "_blank) is implemented  or having binding which are not going to change , make the practice to use **:: (double colon)**before our binding. Example

[](http://2.bp.blogspot.com/--0roRw3hAHY/VkJgLeT25tI/AAAAAAAAB9I/gMl7YdTQNBY/s1600/bindonce.png)

**34)Difference between $scope and $root scope:**

$scope is an object that is accessible from current component e.g Controller, Service only.

$rootScope refers to an object which is accessible from everywhere of the application we can say it is a global scope of variable.

we can think $rootScope as global variable and $scope as local variables.

difference of $rootScope and $scope in Angularjs. $rootScope is available globally, no matter what controller you are in, whereas $scope is only available to the current controller and it's children.

$rootScope is a parent object of all whereas $scope angular objects created in a web page.

Example for $scope:

app.controlller("ctrl",ctrl);

ctrl.$inject=["$scope"];

function ctrl($scope){

$scope.data="angularJS";

}

Example for $rootscope:

app.controlller("ctrl",ctrl);

ctrl.$inject=["$rootscope","$scope"];

function ctrl($rootscope,$scope){

$scope.data="angularJS";

$rootscope.my\_un=function(){

return $scope.data;

}

}

}

**35.About ngRoute :**

1)it is predefined module in angularJs and it is native module

using this module we can achive single page application.

2)ngRoute wont support nested views in single page applications.

development of SPA using ngRoute:

step1, downlod ngRoute

step2, creating the module

app.js:

var app=angular.module("app",[ngRoute]);

step3, creating templates

page\_one.html:

{{var\_one}}

page\_two.html:

{{var\_two}}

step4, creating controllers

page\_one.js:

$scope.var\_one="i am from page one";

page\_two.js

$scope.var\_two="i am from page two";

Step5,

mapping in config phase.

mapping should be done using key and values.

it should be done in constructor using user define function called app.config.

**Above** two can be provided by default object i,e $routeProvider.

config.js

app.config(config);

config.$inject=["$routeProvider"];

function config($routeProvider){

$routeProvider.when("/page\_one",{

templateUrl:'templates/page\_one.html'

controller: /'page\_one'})

.when("/page\_two",{

templateUrl:'templates/page\_two.html'

controller: /'page\_two''})

.otherwise("/page\_one",{

templateUrl:'templates/page\_one.html'

controller: /'page\_one'});

}

step6: creating soure page

index.html

<a href="#/page\_one">page\_one</a><br/>

<a href="#/page\_two">page\_two</a><br/>

<div ng-view>

</div>

**36) Difference between ngRoute and ui.router**

|  |  |
| --- | --- |
| ***ngRoute*** | ***Ui.router*** |
| ***1*, ngRoute is native module in angularJS** | ***1,* ui.router is thirdparty module** |
| ***2, declaration:***  **Var app=angular.module(“app”,[“ngRoute”]);** | ***2, declaration:***  **var app=angular.module(“app”,[“ui.router”]);** |
| ***3)* it wont support nested views and named views in SPA** | ***3)* it will support nested views and named in SPA** |
| **4)won’t support object passing as url parameters** | **4)will support object passing as url parameters** |
| **5)syntax:**  **$stateprovider.state("page\_one",{**  **url:"/page\_one",**  **templateUrl: 'templates'/page\_one.html',**  **controller:'page\_one}')** | **5)syntax:**  **$routeProvider.when("/page\_one",{**  **templateUrl:'templates/page\_one.html'**  **controller: /'page\_one'})** |
| **6) parameters:**  **$stateparams** | **6) parameters:**  **$routeparams** |
| **7)**  **Uses href for linking** | **7) ui-router allows for us to have strong-type linking between states based on state names. Change the url in one place will update every link to that state when you build your links with [ui-sref](http://angular-ui.github.io/ui-router/site/" \l "/api/ui.router.state.directive:ui-sref)** |
| **8) keys in source page should start with “#”** | **8) not required** |

**37.What is Routing?**

Loading the Target Templates to the Source Templates without refreshing is called as Routing in Single Page Application

If you want to navigate to different pages in your application, but you also want the application to be a SPA (Single Page Application), with no page reloading, you can use the ngRoute module.

**38.In how many ways we can achieve the Routings in Angular Js**

In AngularJs Routing can be done in two ways**:**

* **ngRoute Module**
* **ui.Router Module**

**ngRoute Module:** The ngRoute Module routes  your application to different pages without reloading the entire application.

**Ui.Router Module:** The **UI**-**Router** is a **routing** framework for **AngularJS** built by the AngularUI team. It provides a different approach than ngRoute in that it changes your application views based on state of the application and not just the route URL ( i.e. ngroute just route the url and ui router changes the view based on state of application)

**39.Explain the ui router module in the in AngularJS**

The UI-Router is a routing framework for AngularJS built by the [AngularUI team](https://angular-ui.github.io/" \t "_blank). It provides a different approach than ngRoute in that it changes your application views based on **state of the application** and not just the route URL.

STATES VS URL ROUTE

With this approach, your views and routes aren’t tied down to the site URL. This way, you can change the parts of your site using your routing even if the URL does not change.

When using ngRoute, you’d have to use ngInclude or other methods and this could get confusing. Now that all of your states, routing, and views are handled in your one .config(), this would help when using a top-down view of your application.

Sample application to achieve ui router module:

**Step 1 :** Download the ui.router module by using bower.

**bower.json –**

dependencies:{

"angular":"~1.6.0",

"angular-ui-router":"~0.2.18"

}

**.bowerrc.json -**

{

"directory":"bower\_components"

}

**Step 2 :** Add the ui.router module dependency

**app.js -**

var app=angular.module("app",["ui.router"]);

**Step 3 :**

**Create the Templates**

**-----------------------**

**Basic\_uiRouter**

**templates**

**page\_one.html**

**page\_two.html**

**-------------------------**

**page\_one.html-**

{{var\_one}}

**page\_two.html-**

{{var\_two}}

**Step 4 : Create the Controllers**

**-----------------------------**

**Basic\_uiRouter**

**controllers**

**page\_one.js**

**page\_two.js**

**-----------------------------**

**page\_one.js-**

$scope.var\_one="i am from page one controller";

**page\_two.js-**

$scope.var\_two="i am from page two controller";

**Step 5 : Done the mappings**

**config.js -**

app.config(config);

config.$inject=["$stateProvider","$urlRouterProvider"];

function config($stateProvider,$urlRouterProvider){

$stateProvider.state("page\_one",{

url:"/page\_one",

templateUrl:'templates/page\_one.html',

controller:'page\_one'

})

.state("page\_two",{

url:"/page\_two",

templateUrl:'templates/page\_two.html',

controller:'page\_two'

});

$urlRouterProvider.otherwise("/page\_one");

}

**Step 6 :** Create the keys in source template (index.html)

**index.html-**

<html ng-app="app">

<a ui-sref="page\_one"><b>Page\_One</b></a>

<a ui-sref="page\_two"><b>Page\_Two</b></a>

<div ui-view>

</div>

<script src="bower\_components/angular/angular.min.js"></script>

<script src="bower\_components/angular-ui-router/release/angular-ui-router.min.js"></script>

<script src="app.js"></script>

<script src="config.js"></script>

<script src="controllers/page\_one.js"></script>

<script src="controllers/page\_two.js"></script>

</html>

**40.How to create services in the angularjs?**

Services:

* It provides us method to keep data across the lifetime of the angular app
* It provides us method to communicate data across the controllers in a consistent way
* This is a singleton object and it gets instantiated only once per application
* It is used to organize and share data and functions across the application

An AngularJS service can be created in four different ways,

* Using the service() method
* Using the factory() method
* Using the provider() method
* Using the value() method
* Using the constant() method

**Service() method:**

Services provided to the server or reusable components.

Example:

app.service(“my\_service”,my\_service);

function my\_service(){

this.my\_fun=function(){

return(“Iam from service”);

}

Var myClass=function(data) {

return data;

}

}

**Factory() method:**

**factory method** is to generate a single object or **function** that represents the service to rest of the application.

Example:

app.factory(“my\_service”,my\_service);

function my\_service(){

return(“Iam from Factory”);

}

**Provider() method:**

 Provider functions are constructor functions, whose instances are responsible for "providing" a factory for a service.

Provider() used to create the custom services, it will depend on $get.

Example:

app.provider(“my\_service”,my\_service);

function my\_service(){

this.$get=function(){

return=”iam from provider”;

}

}

**Value() method**:

Old value will over ridden with the new value.

Example:

app.value(“my\_service”,”first\_value”);

app.value(“myservice”,”second value”);

output:

second value.

**Constant() method:**

Here in this over riding may not be possib**le**

Example:

App.constant(“my\_service”,”first\_value”);

app.constant(“myservice”,”second value”);

output:

first value.

**41.what are the advantages of services in angularjs?**

* Gives us the instance of a function (object)- You just instantiated with the ‘new’ keyword and you’ll add properties to ‘this’ and the service will return ‘this’.When you pass the service into your controller, those properties on ‘this’ will now be available on that controller through your service. (Hypothetical Scenario)
* Singleton and will only be created once
* Reusable components
* Dependencies are injected as constructor arguments
* Used for simple creation logic
* If you're using a class you could use the service provider

Syntax: module.service(‘serviceName’, function);

**42.In how many can we create the services?**

There are 5 types of services we can create

* Using the service() method
* Using the factory() method
* Using the provider() method
* Using the value() method
* Using the constant() method

**43. How many types of data binding in AngularJs?**

## AngularJS Data Binding

The **data binding** is the data synchronization processes that work between the model and view components. In Angular, model treat as source of application and view is the projection of angular model.

In angularjs when model data got changed that time the view data will change automatically and vice versa.

We have two types of data bindings available in angularjs those are

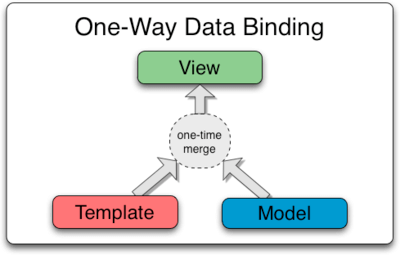
                1. One-Way data binding

                2. Two-Way data binding

We will learn each binding in detail with examples in angularjs.

AngularJS One-Way Data Binding

In **One-Way** data binding, view (UI part) not updates automatically when data model changed and we need to write custom code to make it updated every time. Its not a synchronization processes and it will process data in one way that will be like as shown following image.



We will see simple example of using one way data binding in angularjs.

AngularJS One-way Data Binding Example

Following is the example of using one way data binding in angularjs application.

[Live Preview](http://tutlane.com/example/angularjs/angularjs-one-way-data-binding-example" \t "_blank" \o "Angularjs one way data binding example)

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>

AngularJs One way Binding Example

</title>

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

<script type="text/javascript">

var app = angular.module('angularonebindapp', []);

app.controller('angularonebindingCtrl', function ($scope) {

$scope.name = 'Welcome to one way binding';

});

</script>

</head>

<body ng-app="angularonebindapp">

<div ng-controller="angularonebindingCtrl">

<p>

Message:   {{ name }}

</p>

</div>

</body>

</html>

Here if you observe above code we are binding [model](http://tutlane.com/tutorial/angularjs/angularjs-ng-model-directive-with-example" \t "_blank" \o "Angularjs ng-model directive with example) values to html elements using data bindings but html elements it won't change the values in [model](http://tutlane.com/tutorial/angularjs/angularjs-ng-model-directive-with-example" \t "_blank" \o "Angularjs ng-model directive with example) its one way data binding. Now run application and see the output.

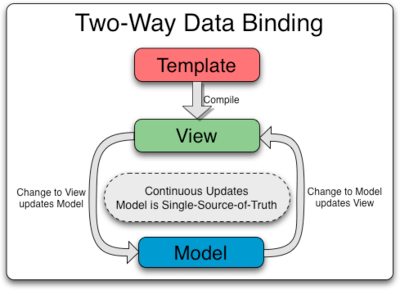
Output of AngularJS One-way Data Binding

Following is the output of angularjs one way data binding

Message: Welcome to one way binding

AngularJS Two-way Data Binding

In **Two-way** data binding, view (UI part) updates automatically when data model changed. Its synchronization processes and two way data binding that will be like as shown following image.



We can achieve this two-way data binding using [ng-model](http://tutlane.com/tutorial/angularjs/angularjs-ng-model-directive-with-example" \o "Angularjs ng-model directive to bind data to html elements with example" \t "_blank) directive. If we use ng-model directive in html control it will update value automatically whenever data got changed in input control.

**How does data binding work in the AngularJs?**

In AngularJs, it will remember present values and compare it with previous value. If any changes found in previous value that time change event will fire automatically and it will update data every time when data got changed.

The HTML tells the AngularJs compiler to create the **$watch** for controller methods and its run inside the **$apply** method. We will see simple example for two way data binding in angularjs.

AngularJS Two-way Data Binding Example

Following is the example of binding data in two way in angularjs applications.

[Live Preview](http://tutlane.com/example/angularjs/angularjs-two-way-data-binding-example" \t "_blank" \o "Angularjs two way data binding example)

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>

AngularJs Two Binding Example

</title>

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

<script type="text/javascript">

var app = angular.module('angulartwobindapp', []);

app.controller('angulartwobindingCtrl', function ($scope) {

$scope.name = 'Welcome to two way binding';

});

</script>

</head>

<body ng-app="angulartwobindapp">

<div ng-controller="angulartwobindingCtrl">

Enter Name : <input type="text" ng-model="name" style="width:250px" />

<p>

Entered Name:   {{ name }}

</p>

</div>

</body>

</html>

If you observe above code we defined [ng-model](http://tutlane.com/tutorial/angularjs/angularjs-ng-model-directive-with-example" \o "Angularjs ng-model directive to bind data to html elements with example" \t "_blank) objective to html control and used same [ng-model](http://tutlane.com/tutorial/angularjs/angularjs-ng-model-directive-with-example" \o "Angularjs ng-model directive to bind data to html elements with example" \t "_blank) value to show input control value. Here whenever we change input control value automatically the appearance value also will get changed. Now we will run and see the output.

Output of AngularJS Two-way Data Binding

Following is the result of two way data binding in angularjs application



**44.Difference between service & factory in angularjs**

AngularJS supports the concepts of "Separation of Concerns" using services architecture. Services are javascript functions and are responsible to do a specific tasks only. This makes them an individual entity which is maintainable and testable. Controllers, filters can call them as on requirement basis. Services are normally injected using dependency injection mechanism of AngularJS.

In **AngularJS**, services are reusable singleton objects that are used to organize and share code across your app. They can be injected into controllers, filters, directives. **AngularJS** provides you three ways :

* Service
* **factory**
* provider to create a service.

#### AngularJS .factory

module.factory('MyService', function() {

var factory = {};

factory.method1 = function() {

//..

}

factory.method2 = function() {

//..

}

return factory;

});

#### AngularJS .service

module.service('MyService', function() {

this.method1 = function() {

//..

}

this.method2 = function() {

//..

}

});

Example

Following example will showcase all the above mentioned directives.

*testAngularJS.htm*

<html>

<head>

<title>Angular JS Services</title>

<scriptsrc="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>

</head>

<body>

<h2>AngularJS Sample Application</h2>

<div ng-app = "mainApp"

ng-controller="CalcController">

<p>Enter a number: <input type = "number" ng-model = "number" /></p>

<button ng-click = "square()">X<sup>2</sup></button>

<p>Result: {{result}}</p>

</div>

<script>

var mainApp = angular.module("mainApp", []);

mainApp.factory('MathService', function() {

var factory = {};

factory.multiply = function(a, b) {

return a \* b

}

return factory;

});

mainApp.service('CalcService', function(MathService){

this.square = function(a) {

return MathService.multiply(a,a);

}

});

mainApp.controller('CalcController', function($scope, CalcService) {

$scope.square = function() {

$scope.result = CalcService.square($scope.number);

}

});

</script>

</body>

</html>

**45.what are the services in angularjs?**

In AngularJS you can make your own service, or use one of the many built-in services.

## What is a Service?

In AngularJS, a service is a function, or object, that is available for, and limited to, your AngularJS application.

AngularJS has about 30 built-in services. One of them is the $location service.

The $location service has methods which return information about the location of the current web page:

### Example

Use the $location service in a controller:

<!DOCTYPE html>

<html>

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

<body>

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

<p>The url of this page is:</p>

<h3>{{myUrl}}</h3>

</div>

<p>This example uses the built-in $location service to get the absolute url of the page.</p>

<script>

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

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

$scope.myUrl = $location.absUrl();

});

</script>

</body>

</html>

Output:

he url of this page is:

### https://www.w3schools.com/angular/tryit.asp?filename=try\_ng\_services

This example uses the built-in $location service to get the absolute url of the page.

Note that the $location service is passed in to the controller as an argument. In order to use the service in the controller, it must be defined as a dependency.

## Why use Services?

For many services, like the $location service, it seems like you could use objects that are already in the DOM, like the window.location object, and you could, but it would have some limitations, at least for your AngularJS application.

AngularJS constantly supervises your application, and for it to handle changes and events properly, AngularJS prefers that you use the $location service instead of the window.location object.

## The $http Service

The $http service is one of the most common used services in AngularJS applications. The service makes a request to the server, and lets your application handle the response.

### Example

Use the $http service to request data from the server:

<!DOCTYPE html>

<html>

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

<body>

​

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

​

<p>Today's welcome message is:</p>

​

<h1>{{myWelcome}}</h1>

​

</div>

​

<p>The $http service requests a page on the server, and the response is set as the value of the "myWelcome" variable.</p>

​

<script>

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

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

 $http.get("welcome.htm").then(function (response) {

     $scope.myWelcome = response.data;

});

});

</script>

​

</body>

</html>

​

OUTPUT:

Today's welcome message is:

# Hello AngularJS Students

The $http service requests a page on the server, and the response is set as the value of the "myWelcome" variable.

This example demonstrates a very simple use of the $http service.

## The $timeout Service

The $timeout service is AngularJS' version of the window.setTimeout function.

### Example

Display a new message after two seconds:

<!DOCTYPE html>

<html>

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

<body>

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

<p>This header will change after two seconds:</p>

<h1>{{myHeader}}</h1>

</div>

<p>The $timeout service runs a function after a specified number of milliseconds.</p>

<script>

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

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

$scope.myHeader = "Hello World!";

$timeout(function () {

$scope.myHeader = "How are you today?";

}, 2000);

});

</script>

</body>

</html>

**The $interval Service**

The $interval service is AngularJS' version of the window.setInterval function.

### Example

Display the time every second:

<!DOCTYPE html>

<html>

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

<body>

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

<p>The time is:</p>

<h1>{{theTime}}</h1>

</div>

<p>The $interval service runs a function every specified millisecond.</p>

<script>

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

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

$scope.theTime = new Date().toLocaleTimeString();

$interval(function () {

$scope.theTime = new Date().toLocaleTimeString();

}, 1000);

});

</script>

</body>

</html>

The time is:

# 9:19:05 PM

The $interval service runs a function every specified millisecond.

**46. why is scopeless(routescope) controller used in angularjs?**

$scope is used for communicate between controller and view. $scope binds a view (DOM element) to the viewmodel

But rootscope, There is only one rootscope in the app and it is shared among all the components of an app. $rootscope a global variable. all others $scopes are children of that $rootScope.

**For Example**

there are two controllers both have scope

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

app.controller('Ctrl1', function ($scope, $rootScope) {

$scope.msg = 'World';

$rootScope.name = 'AngularJS';

});

app.controller('Ctrl2', function ($scope, $rootScope) {

$scope.msg = 'Dot Net Tricks';

$scope.myName = $rootScope.name;

});

rootscope only availble for all controllers but scope didn't get from another controller

**47.How to create custom directives?**

Creating our own directives based on the apllication requirement called as custom directive..

We can use these directives in 3 ways

1. Element
2. Attribute
3. Css class

To create the custom directive we have to know the properties of the custom directives

**restrict:** Used to describe the usage of the custom directive.

**template:**Used to add the html to the custom directive.

**templateUrl:** Used to add the external templates(html files) to the custom directives.

**Controller:** Used to declare the controllers for the templates of custom directives.

**Sample code with all properties together.**.

**Element level :**

app.directive(“my\_directive”,my\_directive);

function my\_directive(){

return{

restrict: ‘E’, // we have A and C also the values of the property restrict

template : ‘<b> this is custom directive</b>

}

}

In index.html

<my\_directive> </ my\_directive >…………….. element level

Note : if the template is externl html file we use templateUrl: property..

Attribute level :

app.directive(“attrType”, attrType);

fuction attrType (){

return{

retrict: ‘A’, // attribute level usage

templateUrl: ‘ templates/sample.html’ ,

controller:’sample’

} }

In index.html

<div class=”container” attr\_type></div>……………………………Attribute level usage

Class level:

app.directive(“classType”, classType);

fuction classType (){

return{

retrict: ‘C’, // class level usage

templateUrl: ‘ templates/sample.html’ ,

controller:’sample’

}

}

In index.html

<div class = “ container class\_type”> </div>

All level usage :

app.directive(“allType”,allType);

function allType(){

return {

restrict: ’CAE’,

templateUrl: ‘ templates/sample.html’ ,

controller:’sample’

}

}

In index.html

<div all\_type class = “container”></div> ………attribute level usage

<div class = “container”>

<all\_type></all\_type>

</div>……………………………. Element level usage

<div class = “ container all\_type”> </div>………………… class level usage

**48.Difference between $http and $q.promise?**

$http:

This is predefined service that facilitates the communication with the remote HTTP servers. $ http makes a request to the server and returns a response.

Example:

app.service(“my\_service”,my\_service);

my\_service.$inject = [“$http”];

function my\_service($hhtp){

this.fun\_one = function(){

return $http.get(“<https://www.w3schools.com/angular/customers.php>”).then(function(response){

return response.data;

})

}

}

$q:

this predefine service is used to make asynchronous calls to the server and also used to reduce the duplicate code

for example when there are number calls hitting the server using $hhtp service and all the calls will done using $q

sample code given below:

var defferd1 = $q.defer();

this.fun\_one = function(){

return $http.get(“<https://www.w3schools.com/angular/customers.php>”).then(function(response){

deffered1.resolve(response.data);

return deffered.promise;

},function(response){

deffered1.reject(response.data);

return deffered1.promise;

})

}

}

Note: promise is used to close the asynchronous call whether the response is positive or negative..

In ctrl.js file

app.controller("ctrl",ctrl);

ctrl.$inject=["$scope","my\_service","$q"];

function ctrl($scope,my\_service,$q) {

$q.all([my\_service.fun\_one(),my\_service.fun\_two(),my\_service.fun\_three()]).then(function (response) {

$scope.var\_one=response[0];

$scope.var\_two=response[1];

$scope.var\_three=response[2];

});

**49.how many types of directives?Explain**

Directives are used to enhance the html capabilities and There are two types of directives

1. Predefined directives
2. Custom directives

**Predefined directives:**  The directives given by the framework are called pre defined directives..

Ex: ng-app, ng-controller, ng-model, ng-bind etc…

**Custom directives:** creating our own directives based on the application requirement are called as custom directives

These can be created in three ways ..

1.element

2. attribute

3.css class

Ex: my\_directive….. custom directive

<div my\_directive></div>------------------ Attribute level usage

<my\_directive></my\_directive>-----------Element level usage

<div class= “class my\_directive”></div>-----class level usage

**50.Explain the concepts/how will use $http.head()?**

The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response. The metainformation contained in the HTTP headers in response to a HEAD request SHOULD be identical to the information sent in response to a GET request. This method can be used for obtaining metainformation about the entity implied by the request without transferring the entity-body itself. This method is often used for testing hypertext links for validity, accessibility, and recent modification.

The response to a HEAD request MAY be cacheable in the sense that the information contained in the response MAY be used to update a previously cached entity from that resource. If the new field values indicate that the cached entity differs from the current entity (as would be indicated by a change in Content-Length, Content-MD5, ETag or Last-Modified), then the cache MUST treat the cache entry as stale.

The HEAD method is functionally similar to GET, except that the server replies with a response line and headers, but no entity-body. The following example makes use of HEAD method to fetch header information about hello.htm:

HEAD /hello.htm HTTP/1.1

User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)

Host: [www.tutorialspoint.com](http://www.tutorialspoint.com/" \t "_blank)

Accept-Language: en-us

Accept-Encoding: gzip, deflate

Connection: Keep-Alive  
  
The server response against the above GET request will be as follows:   
HTTP/1.1 200 OK

Date: Mon, 27 Jul 2009 12:28:53 GMT

Server: Apache/2.2.14 (Win32)

Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT

ETag: "34aa387-d-1568eb00"

Vary: Authorization,Accept

Accept-Ranges: bytes

Content-Length: 88

Content-Type: text/html

Connection: Closed

You can notice that here server the does not send any data after header.

**51. What is $q?**

* It is used to make the server calls asynchronously.
* A service that helps you run functions asynchronously.
* Asynchronous means one server call won’t effect to another server calls.

Html:

<div ng-app>

<h2>$q test</h2>

<div ng-controller="TodoCtrl">

<div ng-bind="'Promise1: ' + Promise1"></div>

<div ng-bind="'Promise2: ' + Promise2"></div>

<div ng-bind="'Promise3: ' + Promise3"></div>

<div ng-bind="'Promise4: ' + Promise4"></div>

<div ng-bind="'Promise5: ' + Promise5"></div><br />

<div ng-bind="'Status1: ' + Status1"></div>

<div ng-bind="'Status2: ' + Status2"></div>

</div>

</div>

JavaScript:

function TodoCtrl($scope, $q, $timeout) {

function createPromise(name, timeout, willSucceed) {

$scope[name] = 'Running';

var deferred = $q.defer();

$timeout(function() {

if (willSucceed) {

$scope[name] = 'Completed';

deferred.resolve(name);

} else {

$scope[name] = 'Failed';

deferred.reject(name);

}

}, timeout \* 1000);

return deferred.promise;

}

// Create 5 promises

var promises = [];

var names = [];

for (var i = 1; i <= 5; i++) {

var willSucceed = true;

if (i == 2) willSucceed = false;

promises.push(createPromise('Promise' + i, i, willSucceed));

}

// Wait for all promises

$scope.Status1 = 'Waiting';

$scope.Status2 = 'Waiting';

$q.all(promises).then(

function() {

$scope.Status1 = 'Done';

},

function() {

$scope.Status1 = 'Failed';

}

).finally(function() {

$scope.Status2 = 'Done waiting';

});

}

Output:

## $q test

Promise1: Completed

Promise2: Failed

Promise3: Completed

Promise4: Completed

Promise5: Completed

Status1: Failed

Status2: Done waiting

**53.what are the types of scopes available in custom directives**

Scope is an object that refers to the application model. It is an execution context for expressions. Scopes are arranged in hierarchical structure which mimic the DOM structure of the application. Scopes can watch expressions and propagate events.

This is to ease the understanding of the $scope in the custom directives. While creating custom directives we have to apply different types of functionalities, so for that we must know about the $scope behavior in the directives.

$scope has a very important role in AngularJS, it works as a mediator (or like a glue) between the logic & view in an Angular application. Now if we talk about the custom directives, then first question which arises is-

“In any application if we want some specific functionality and we want to reuse that in whole application module, then for this we need to develop a set of code. Angular calls it directives.”

I am not going to discuss so much about custom directives basics here. In this blog I am just focusing on use of $scope into directives.

So, when we create a custom directive it has a default scope, which is the parent scope (the controller’s scope from where the directive is called). This behavior is by default, until and unless we do not set the scope.

As we know that whenever we define a directive, there is a “directive definition object” (DDO), in which we set some parameters like- restrict, template, require, scope etc.

In this blog I will talk about the scope properties, they are  false, true, {}.

Let’s discuss them one by one.

Scope : false (Shared Scope)

In layman language false is assumed as no, so if the scope is set to false, then it means use parent scope in the directive and do not create a new scope for the directive itself. It just uses the scope of respective controller. So let us suppose if we have a controller named “homeController” and a directive “printName”, then in printName directive we will get parent scope by default and any change in scope values, either child or parent, will reflect in both.

Example:

Code Snippet

var app = angular.module(“blogDemo”, []);

app.controller(“sharedController”, function ($scope) {

    $scope.name = “rock”;

});

app.directive(“sharedDirective”, function () {

    return {

        restrict: “EA”,

        scope: false,

        template: “<div>directive scope value : {{name}}</div>” +

        “Change directive scope value : <input type=’text’ ng-model=’name’ />”

    };

});

//view (html)

<div ng-app=”blogDemo”>

      <div ng-controller=”sharedController“>

           <h2>parent scope value {{name}} </h2>

           <div shared-directive ></div>

       </div>

</div>

In this example, we can see whenever the value of parent scope changes, it will reflect in the directive scope also, because they both are sharing the same scope object.

Scope : true (Inherited Scope)

Using this property, the directive will create a new scope for itself. And inherit it from parent scope. If we do any changes to the controller scope it will reflect on directive scope, but it won’t work the other way around. This is because both of them use their own copies of scope object.

Example:

Code Snippet

//module, controller, directive

var app = angular.module(“blogDemo”,[]);

app.controller(“inheritedController”,function($scope){

    $scope.orgName = “Quovantis Parent”;

});

app.directive(“inheritedDirective”, function(){

    return {

        restrict: “EA”,

        scope: true,

        template: “<div>my organisation name is : {{orgName}}</div> type for change name : <input type=’text’ ng-model=’orgName’ />”

    };

});

//view (html)

<div ng-app=”blogDemo”>

       <div ng-controller=”inheritedController“>

            <h2>parent scope value {{orgName}} </h2>

            <div inherited-directive ></div>

        </div>

</div>

In this example when the first time directive loads, the screen will show the value of parent scope. But when we will change the value from text box. Then this will only change into child scope only. Means no change in parent scope.

Scope : {} (Isolated Scope)

One of the important features, its called isolated scope. Here too the directive will create a new scope object but it is not inherited by the parent scope, so now this scope doesn’t know anything about the parent scope.

But the question arises, if we do not have the link from parent scope then how can we get the values from it, and how can we modify it ?

The answer is- set the objects property into DDO, but for this it is necessary to set on attributes into the directive.

In isolated scope we use three prefixes which helps to bind the property or methods from the controller (parent scope) to directive (isolated scope). Lets understand how this works.

Whenever a directive finds any prefixes in its scope property in DDO, it checks it in directive declaration (in html page where the directive is called) with attribute declared on this element. We can also change the name by giving a separate attribute name after any of the prefixes.

These are @, =, &

‘@’ : One way binding

One way binding means a parent sending anything to the directive scope through the attribute, gets reflected in the directive. But if any change in the directive happens it will not reflect in parent. The @ is used to pass string values.

Example:

Code Snippet

//module

var app = angular.module(‘quovantisBlog’, []);

//controller

app.controller(‘OneWayController’, [‘$scope’, function ($scope) {

    $scope.student = {

        name: ‘Rohit’,

        class: ‘MCA’,

        Address: ‘New Delhi’

    };

}]);

// directive

app.directive(‘oneWayDirective’, function () {

    return {

        scope: {

            name: ‘@’

        },

        template: ‘Student  Name: {{ name }}’

    };

});

//view (html)  
<one-way-directive name=”{{ student.name }}”></one-way-directive>  
or  
<one-way-directive studName=”{{ student.name }}”></one-way-directive>  
then directive would be with a change in scope property.

Code Snippet

app.directive(‘oneWayDirective’, function () {

    return {

        scope: {

            name: ‘@studName’

        },

        template: ‘Student Name: {{ name }}’

    };

});

‘=’ : Two way binding

This is called two way binding, because the parent scope will also reflect to directive scope vice-versa.

It is used for passing object to the directive instead of string. This object could be changed from both sides, from parent or from directive. That is why it is called two-way.

Example:

Code Snippet

//module

var blogDemo = angular.module(‘myApp’,[]);

//directive

blogDemo.directive(‘twoWayDirective’, function() {

    return {

        restrict: ‘EA’,

        scope: { obj: “=”},

        template: ‘<div>Welcome, {{obj.fname + obj.lname}}!</div>’

};

});

//controller

blogDemo.controller(‘blogController’, function ($scope) {

$scope.obj = { fname: “shubh”, lname: “raj” };

});

//view (html)

<div ng-controller=”blogController”>

     <two-way-directive obj=”obj”></two-way-directive>

</div>

‘&’ : Method binding

Used to bind any parent’s method to directive scope. Whenever we want to call the parent methods from the directive we can use this. It is used to call external (outside of current scope) functions. Overall “&” is used to pass data as a function or method.

Example:

Code Snippet

//module

var blogDemo = angular.module(‘myApp’,[]);

//directive

blogDemo.directive(‘methodDirective’, function() {

    return {

        scope: {

            studData: ‘=’,

            swap: ‘&’

        },

        template: ‘<div>the changed names are, {{obj.fname + obj.lname}}!</div>’+

            ‘<button id=”btn1” ng-click=”swap()”>Click here to Swap student Data</button>’

    };

});

//controller

blogDemo.controller(‘blogController’, function ($scope) {

    $scope.studData = { fname: “shubh”, lname: “raj” };

    $scope.swapData = function () {

        $scope.customer = {

            fname: ‘Raj’,

            lname: ‘kumar’

        };

    };

});

//view (html)

<div ng-controller=”blogController”>

        <method-directive studData=”studData” swap=”swapData()”></method-directive>

</div>

In this example the directive creates a property inside its local scope, that is swapData. We can also understand swap as an alias for swapData. So we pass a method to ‘&’ which is then invoked by the directive whenever required.

Summarizing, Shared scope (sharing the same scope and data, can not pass the data explicitly), Inherited scope (parent scope values can be fetched into child but child means directive scope will not effect parent), Isolated scope (both controller & directive do not share the scope & data, we can explicitly pass the data using some parameters that is @, & ,= ).

**54. What is component? How do we create it and Advntages?**

* In AngularJS, a Component is a special kind of [directive](https://docs.angularjs.org/guide/directive) that uses a simpler configuration which is suitable for a component-based application structure.
* Components can be registered using the .component() method of an AngularJS module (returned by [angular.module()](https://docs.angularjs.org/guide/module)). The method takes two arguments:
* The name of the Component (as string).
* The Component config object.
* **Advantages of Components**:
* simpler configuration than plain directives
* promote sane defaults and best practices
* optimized for component-based architecture
* writing component directives will make it easier to upgrade to Angular

Creating and configuring a component

Components can be registered using the .component() method of an AngularJS module (returned by angular.module()). The method takes two arguments:

The name of the Component (as string).The Component config object. (Note that, unlike the .directive() method, this method does not take a factory function.)

**Index.js**

(function(angular) {

'use strict';

angular.module('heroApp', []).controller('MainCtrl', function MainCtrl() {

this.hero = {

name: 'Spawn'

};

});

})(window.angular);

**heroDetail.js**

(function(angular) {

'use strict';

function HeroDetailController() {

}

angular.module('heroApp').component('heroDetail', {

templateUrl: 'heroDetail.html',

controller: HeroDetailController,

bindings: {

hero: '='

}

});

})(window.angular);

**Index.html**

<!doctype html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Example - example-heroComponentSimple-production</title>

<script src=

” https://ajax.googleapis.com/ajax/libs/angularjs/1.6.1/angular.min.js"></script>

<script src="index.js"></script>

<script src="heroDetail.js"></script>

</head>

<body ng-app="heroApp">

<!-- components match only elements -->

<div ng-controller="MainCtrl as ctrl">

<b>Hero</b><br>

<hero-detail hero="ctrl.hero"></hero-detail>

</div>

</body>

</html>

**heroDetail.html**

<span>Name: {{$ctrl.hero.name}}</span>

**55.Use of compile and link function ?**

* compilation phase a directive also has a chance to modify the DOM node before a scope is attached to it;
* compilation phase we do not have access to the $scope data
* if the directive needs to access the scope, its link function allows that

 In the link phase the data i.e. ($scope) is attached to the template function and executed to get the final HTML output.

**56. what is Isolate scope?**

* An isolate scope is a scope that exists separately with no prototypal inheritance at all; a clean slate. To create an isolate scope it's as simple as setting the scope property to an empty object hash {}.

**57. 57 -55 same questions**

1. **How compilation process will happen in custom directives?**

## The AngularJS way

AngularJS processes the template in entirely different way. It works directly on HTML DOM fragments(in memory) rather than strings, and manipulates it as required. It uses two way data-binding between model and view to sync our data.

Note that ****Angular processes on DOM nodes instead of strings contents****.

There are basically two phases in which [Angular](http://www.tothenew.com/front-end-angularjs-development" \o "Angular.js Development Services) does this, compile and link.

The concept of compile and link comes from C language, where you first compile the code and then link it to actually execute it. The process is very much similar in AngularJS as well.

## The Process

1. ****Compile**** – $compile function traverse the DOM and collect all directives from DOM. For each directive it finds, it adds it to a list of directives. Once the entire DOM has been traversed, it will sort that list of directives by their priority. It takes our HTML markup or template one by one and returns a link function.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **var** $compile = ...;     // injecte $comple into our code to get link function  **var** scope = ...;        // scope for new directive  **var** parent = ...;       // DOM element where we want to apply our directive  **var** html = '<div ng-show="data"></div>';  // Our HTML code    **var** template = angular.element(html);   // Step 1A: parse HTML into DOM element to pass in $compile to get link function    **var** linkFn = $compile(template);        // Step 1B: compile the template and return link function |

2. ****Link**** – The above link function (Returned by $compile function) combines the directives with a scope to give as two way data binding. Any data-related changes affecting the model are immediately propagated to the matching view(s), and any changes made in the view(s) are reflected in the underlying model.

|  |  |
| --- | --- |
| 1  2  3 | **var** element = linkFn(scope);   // Step 2A: link 'scope' with the compiled template (Two way binding).    parent.appendChild(element);   // Step 2B: Inject our element to parent DOM element |

This can help in scenarios where you get some Markup as a string, and want it to use it with AngularJS, you’ll need to compile the HTML to make it work.

Hope this will help you to understand the compilation process in angular.js.

1. Use of $scope.apply and $scope.copy

* $scope.apply () is a method you call to get your bindings to update.

$scope.copy: Creates a deep copy of source, which should be an object or an array.

* If no destination is supplied, a copy of the object or array is created.
* If a destination is provided, all of its elements (for arrays) or properties (for objects) are deleted and then all elements/properties from the source are copied to it.
* If source is not an object or array (inc. null and undefined), source is returned.
* If source is identical to destination an exception will be thrown.

**60. How to get custom currency symbol using filters**

Html:

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

<div>Currency: <input ng-model="currencySymbol"/></div>

<div>Currency Amount: <input ng-model="currencyAmount"/></div>

<div>formatted: {{currencyAmount | currency:currencySymbol}}</div>

</div>

Javascript:

function myCtrl($scope) {

$scope.currencySymbol = 'USD$';

$scope.currencyAmount = '100.1232131';

}

**61.how to create dynamic buttons in angular js**

Requirements is to add html content dynamically and that content might have a click event on it.

So the code Angular code I have below displays a button, and when clicked, it dynamically adds another button. Clicking on the dynamically added buttons, should add another button, but I cannot get the ng-click to work on the dynamically added buttons

Index.html

<!DOCTYPE html>  
<html lang="en" >  
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.0/angular.min.js">

</script>  
<script src="app.js"></script>  
<link rel="stylesheet" href="my\_style.css">  
<script src="controllers/ctrl.js"></script>  
<div ng-app="app" ng-controller="Ctrl">  
 <button class="addfields" ng-click="addNewChoice()">Add choice</button>  
 <fieldset data-ng-repeat="choice in choicesA">  
 <input type="text" ng-model="choice.name" name="" placeholder="Enter name">  
 <button class="remove" ng-click="removeChoice($index)">-</button>  
 <button class="addfields" ng-click="addNewChoiceB(choice)">Add fields</button>  
 <div data-ng-repeat="choiceb in choice.choicesB">  
 <input type="text" ng-model="choiceb.name" name="" placeholder="Enter field">  
 <button class="remove" ng-click="removeChoiceB(choice,$index)">-</button>  
 </div>  
 </fieldset>  
  
  
 <div id="choicesDisplay">  
 <pre>choicesA = {{ choicesA }}</pre>  
 <pre data-ng-repeat="choiceb in choicesA">choicesB = {{ choiceb.choicesB }}</pre>  
 </div>  
</div>  
</html>

app.js

var app = angular.module("app", []);

my\_style.css

fieldset {  
 background: #FCFCFC;  
 padding: 16px;  
 border: 1px solid #D5D5D5;  
}  
  
.addfields {  
 margin: 10px 0;  
}  
  
#choicesDisplay {  
 padding: 10px;  
 background: rgb(227, 250, 227);  
 border: 1px solid rgb(171, 239, 171);  
 color: rgb(9, 56, 9);  
}  
.remove {  
 background: #C76868;  
 color: #FFF;  
 font-weight: bold;  
 font-size: 21px;  
 border: 0;  
 cursor: pointer;  
 display: inline-block;  
 padding: 4px 9px;  
 vertical-align: top;  
 line-height: 100%;  
}  
input[type="text"],  
select {

padding: 5px;  
}

Ctrl.js

app.controller('Ctrl',Ctrl);  
Ctrl.$inject("$scope")  
function Ctrl($scope) {  
  
 $scope.choicesA = [{  
 id: 'choice1',  
 choicesB:[]  
 }, {  
 id: 'choice2',  
 choicesB:[]  
 }];  
  
  
 $scope.addNewChoice = function() {  
 var newItemNo = $scope.choicesA.length + 1;  
 $scope.choicesA.push({  
 'id': 'choice' + newItemNo,  
 choicesB:[]  
 });  
 };  
  
 $scope.removeChoice = function(ind) {  
 $scope.choicesA.splice(ind,1);  
 };  
  
 $scope.addNewChoiceB = function(choice) {  
 var newItemNo = choice.choicesB.length + 1;  
 choice.choicesB.push({  
 'id': 'choice' + newItemNo  
 });  
 };  
  
 $scope.removeChoiceB = function(choice,ind) {  
 choice.choicesB.splice(ind,1);  
 };  
  
};

**62. 53 62 same questions**

**63. 63- 54 same questions.**

**64.What are core features of Angular JS**

AngularJS is a great JavaScript framework that has some very compelling features for not only developers, but designers as well! In this tutorial, we will cover what I consider to be the most essential features, and how they can help make your next web application awesome.

To get an idea of what you can do with AngularJS, check out the range of AngularJS items on Envato Market. You can find an image cropper, an eCommerce web application, a JSON editor, and much more.

The Elevator Pitch: AngularJS in a Nutshell

AngularJS is a new, powerful, client-side technology that provides a way of accomplishing really powerful things in a way that embraces and extends HTML, CSS and JavaScript, while shoring up some of its glaring deficiencies. It is what HTML would have been, had it been built for dynamic content.

In this article, we will cover a few of the most important AngularJS concepts to get the "big picture." It is my goal that, after seeing some of these features, you will be excited enough to go and build something fun with AngularJS.

Feature 1: Two Way Data-Binding

Think of your model as the single-source-of-truth for your application. Your model is where you go to to read or update anything in your application.

Data-binding is probably the coolest and most useful feature in AngularJS. It will save you from writing a considerable amount of boilerplate code. A typical web application may contain up to 80% of its code base, dedicated to traversing, manipulating, and listening to the DOM. Data-binding makes this code disappear, so you can focus on your application.

Think of your model as the single-source-of-truth for your application. Your model is where you go to to read or update anything in your application. The data-binding directives provide a projection of your model to the application view. This projection is seamless, and occurs without any effort from you.

Traditionally, when the model changes, the developer is responsible for manually manipulating the DOM elements and attributes to reflect these changes. This is a two-way street. In one direction, the model changes drive change in DOM elements. In the other, DOM element changes necessitate changes in the model. This is further complicated by user interaction, since the developer is then responsible for interpreting the interactions, merging them into a model, and updating the view. This is a very manual and cumbersome process, which becomes difficult to control, as an application grows in size and complexity.

There must be a better way! AngularJS' two-way data-binding handles the synchronization between the DOM and the model, and vice versa.

Here is a simple example, which demonstrates how to bind an input value to an <h1> element.

|  |  |
| --- | --- |
|  | <!doctype html>  <html ng-app>    <head>      <script src="<http://code.angularjs.org/angular-1.0.0rc10.min.js>"></script>    </head>    <body>      <div>        <label>Name:</label>        <input type="text" ng-model="yourName" placeholder="Enter a name here">        <hr>        <h1>Hello, {{yourName}}!</h1>      </div>    </body>  </html> |

This is extremely simple to set up, and almost magical…

## Feature 2: Templates

It's important to realize that at no point does AngularJS manipulate the template as strings. It's all the browser DOM.

In AngularJS, a template is just plain-old-HTML. The HTML vocabulary is extended, to contain instructions on how the model should be projected into the view.

The HTML templates are parsed by the browser into the DOM. The DOM then becomes the input to the AngularJS compiler. AngularJS traverses the DOM template for rendering instructions, which are called directives. Collectively, the directives are responsible for setting up the data-binding for your application view.

It is important to realize that at no point does AngularJS manipulate the template as strings. The input to AngularJS is browser DOM and not an HTML string. The data-bindings are DOM transformations, not string concatenations or innerHTML changes. Using the DOM as the input, rather than strings, is the biggest differentiation AngularJS has from its sibling frameworks. Using the DOM is what allows you to extend the directive vocabulary and build your own directives, or even abstract them into reusable components!

One of the greatest advantages to this approach is that it creates a tight workflow between designers and developers. Designers can mark up their HTML as they normally would, and then developers take the baton and hook in functionality, via bindings with very little effort.

Here is an example where I am using the ng-repeat directive to loop over the images array and populate what is essentially an img template.

function AlbumCtrl($scope) {

scope.images = [

{"thumbnail":"img/image\_01.png", "description":"Image 01 description"},

{"thumbnail":"img/image\_02.png", "description":"Image 02 description"},

{"thumbnail":"img/image\_03.png", "description":"Image 03 description"},

{"thumbnail":"img/image\_04.png", "description":"Image 04 description"},

{"thumbnail":"img/image\_05.png", "description":"Image 05 description"}

];

}

|  |  |
| --- | --- |
|  | <div ng-controller="AlbumCtrl">    <ul>      <li ng-repeat="image in images">        <img ng-src="{{image.thumbnail}}" alt="{{image.description}}">      </li>    </ul>  </div> |

It is also worth mentioning, as a side note, that AngularJS does not force you to learn a new syntax or extract your templates from your application.

## Feature 3: MVC

AngularJS incorporates the basic principles behind the original MVC software design pattern into how it builds client-side web applications.

The MVC or Model-View-Controller pattern means a lot of different things to different people. AngularJS does not implement MVC in the traditional sense, but rather something closer to MVVM (Model-View-ViewModel).

The Model

The model is simply the data in the application. The model is just plain old JavaScript objects. There is no need to inherit from framework classes, wrap it in proxy objects, or use special getter/setter methods to access it. The fact that we are dealing with vanilla JavaScript is a really nice feature, which cuts down on the application boilerplate.

The ViewModel

A viewmodel is an object that provides specific data and methods to maintain specific views.

The viewmodel is the $scope object that lives within the AngularJS application. $scope is just a simple JavaScript object with a small API designed to detect and broadcast changes to its state.

The Controller

The controller is responsible for setting initial state and augmenting $scope with methods to control behavior. It is worth noting that the controller does not store state and does not interact with remote services.

The View

The view is the HTML that exists after AngularJS has parsed and compiled the HTML to include rendered markup and bindings.

This division creates a solid foundation to architect your application. The $scope has a reference to the data, the controller defines behavior, and the view handles the layout and handing off interaction to the controller to respond accordingly.

Feature 4: Dependency Injection

AngularJS has a built-in dependency injection subsystem that helps the developer by making the application easier to develop, understand, and test.

Dependency Injection (DI) allows you to ask for your dependencies, rather than having to go look for them or make them yourself. Think of it as a way of saying "Hey I need X', and the DI is responsible for creating and providing it for you.

To gain access to core AngularJS services, it is simply a matter of adding that service as a parameter; AngularJS will detect that you need that service and provide an instance for you.

function EditCtrl($scope, $location, $routeParams) {

// Something clever here...

}

You are also able to define your own custom services and make those available for injection as well.

angular.

module('MyServiceModule', []).

factory('notify', ['$window', function (win) {

return function (msg) {

win.alert(msg);

};

}]);

function myController(scope, notifyService) {

scope.callNotify = function (msg) {

notifyService(msg);

};

}

myController.$inject = ['$scope', 'notify'];

Feature 5: Directives

Directives are my personal favorite feature of AngularJS. Have you ever wished that your browser would do new tricks for you? Well, now it can! This is one of my favorite parts of AngularJS. It is also probably the most challenging aspect of AngularJS.

Directives can be used to create custom HTML tags that serve as new, custom widgets. They can also be used to "decorate" elements with behavior and manipulate DOM attributes in interesting ways.

Here is a simple example of a directive that listens for an event and updates its $scope, accordingly.

myModule.directive('myComponent', function(mySharedService) {

return {

restrict: 'E',

controller: function($scope, $attrs, mySharedService) {

$scope.$on('handleBroadcast', function() {

$scope.message = 'Directive: ' + mySharedService.message;

});

},

replace: true,

template: '<input>'

};

});

Then, you can use this custom directive, like so.

|  |  |
| --- | --- |
| 1 | <my-component ng-model="message"></my-component> |

Creating your application as a composition of discrete components makes it incredibly easy to add, update or delete functionality as needed.

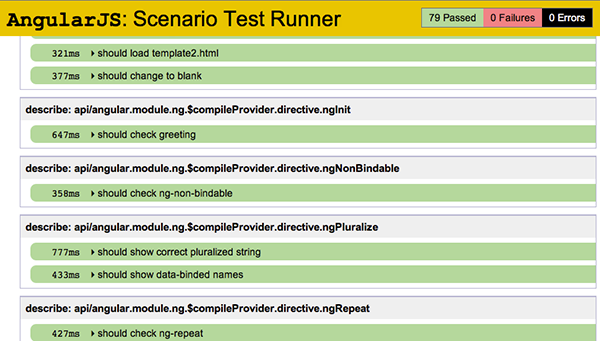
Bonus Feature: Testing

The AngularJS team feels very strongly that any code written in JavaScript needs to come with a strong set of tests. They have designed AngularJS with testability in mind, so that it makes testing your AngularJS applications as easy as possible. So there's no excuse for not doing it.

Given the fact that JavaScript is dynamic and interpreted, rather than compiled, it is extremely important for developers to adopt a disciplined mindset for writing tests.

AngularJS is written entirely from the ground up to be testable. It even comes with an end-to-end and unit test runner setup. If you would like to see this in action, go check out the angular-seed project at <https://github.com/angular/angular-seed>.

Once you have the seed project, it's a cinch to run the tests against it. Here is what the output looks like:



The API documentation is full of end-to-end tests that do an incredible job of illustrating how a certain part of the framework should work. After a while, I found myself going straight to the tests to see how something worked, and then maybe reading the rest of the documentation to figure something out.

**65.What is the exact use of $inject?**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | That is one approach to support Dependency Injection after your code is minified (if you choose to minify).  When you declare a controller, the function takes parameters:  function ($scope,notify)  When you minify the code, your function will look like this:  function (a,b)  Since AngularJS uses the function parameter names to infer DI, your code will break because AngularJS doesn't know about a or b.  To solve this problem, they provided additional ways to declare controllers (or other services/factories/etc) for that matter:  1) For controllers, use the $inject method - here you pass an array of literals that map to the parameters of your controller function. So, if you provide  [“$scope”,”notify”]  then the value of the first parameter to your function will be the a scope object associated with this controller and the second parameter will be the notify service.  2) When declaring new controllers, services, etc, you can use the array literal syntax. Here, you do something like this:  angular.module(“myModule”).controller(“MyController”,[“$scope”,”notify”,function($scope,notify{  ……...  }]);  The array as a parameter to the controller function maps the DI objects to your function parameters   1. **how databinding works internally in angularjs?**   AngularJS handle data-binding mechanism with the help of three powerful functions: $watch(), $digest() and $apply(). Most of the time AngularJS will call the $scope.$watch() and $scope.$digest() functions for you, but in some cases you may have to call these functions yourself to update new values.  ****$watch()**** - This function is used to observe changes in a variable on the $scope. It accepts three parameters: expression, listener and equality object, where listener and equality object are optional parameters.  ****$digest()**** - This function iterates through all the watches in the $scope object, and its child $scope objects (if it has any). When $digest() iterates over the watches, it checks if the value of the expression has changed. If the value has changed, AngularJS calls the listener with the new value and the old value. The $digest() function is called whenever AngularJS thinks it is necessary. For example, after a button click, or after an AJAX call. You may have some cases where AngularJS does not call the $digest() function for you. In that case you have to call it yourself.  ****$apply()**** - Angular do auto-magically updates only those model changes which are inside AngularJS context. When you do change in any model outside of the Angular context (like browser DOM events, setTimeout, XHR or third party libraries), then you need to inform Angular of the changes by calling $apply() manually. When the $apply() function call finishes AngularJS calls $digest() internally, so all data bindings are updated.   1. **What’s diff between factory ,service and providers??**  |  |  |  | | --- | --- | --- | | ****value**** | You are providing a simple literal value. | mod.value("myValue", 10); | | ****constant**** | You need to be able access that value during the configuration phase. (using .config()) | mod.constant("myValue", 10);  mod.config(function(myValue) {  console.log(myValue);}); | | ****factory**** | The value you are providing needs to be calculated based on other data. | mod.factory("myFactory", function() {  return 10;}); | | ****service**** | You are returning an object with methods. | mod.service("myService", function() {  var name = "Bob";  this.setName = function(newName) {  this.name = newName;  };  this.getName = function() {  return this.name;  }}); | | ****provider**** | You want to be able to configure, during the config phase, the object that is going to be created before it’s created. | mod.provider("greeter", function() {  var name;  this.setName = function(newName) {  name = newName;  };  this.$get = function() {  return new function() {  this.sayHi = function() {  console.log("Hi " + name;  };  };  };});  mod.config(function(greeterProvider) {  greeterProvider.setName(“John");  }); |   To drive the point home one last time here is a image of a provider with the factory, value, and service portions highlighted:  IMG_257   1. .dff bwn emit and broadcast   AngularJS provides $on, $emit, and $broadcast services for event-based communication between controllers. $emit It dispatches an event name upwards through the scope hierarchy and notify to the registered $rootScope.Scope listeners. The event life cycle starts at the scope on which $emit was called. The event traverses upwards toward the root scope and calls all registered listeners along the way. The event will stop propagating if one of the listeners cancels it.  IMG_256   1. <!DOCTYPE html> 2. <html> 3. <head> 4. <title>Broadcasting</title> 5. <script src="lib/angular.js"></script> 6. <script> 7. var app = angular.module('app', []); 8. 聽 9. app.controller("firstCtrl", function ($scope) { 10. $scope.$on('eventName', function (event, args) { 11. $scope.message = args.message; 12. console.log($scope.message); 13. }); 14. }); 15. 聽 16. app.controller("secondCtrl", function ($scope) { 17. $scope.handleClick = function (msg) { 18. $scope.$emit('eventName', { message: msg }); 19. }; 20. }); 21. 聽 22. </script> 23. </head> 24. <body ng-app="app"> 25. <div ng-controller="firstCtrl" style="border:2px solid *#E75D5C; padding:5px;*"> 26. <h1>Parent Controller</h1> 27. <p>Emit Message : </p> 28. <br /> 29. <div ng-controller="secondCtrl" style="border:2px solid *#428bca;padding:5px;*"> 30. <h1>Child Controller</h1> 31. <input ng-model="msg"> 32. <button ng-click="handleClick(msg);">Emit</button> 33. </div> 34. </div> 35. </body> 36. </html>  How it works.. IMG_257 $broadcast It dispatches an event name downwards to all child scopes (and their children) and notify to the registered $rootScope.Scope listeners. The event life cycle starts at the scope on which $broadcast was called. All listeners for the event on this scope get notified. Afterwards, the event traverses downwards toward the child scopes and calls all registered listeners along the way. The event cannot be canceled.   1. <!DOCTYPE html> 2. <html> 3. <head> 4. <title>Broadcasting</title> 5. <script src="lib/angular.js"></script> 6. <script> 7. var app = angular.module('app', []); 8. 聽 9. app.controller("firstCtrl", function ($scope) { 10. $scope.handleClick = function (msg) { 11. $scope.$broadcast('eventName', { message: msg }); 12. }; 14. }); 15. 聽 16. app.controller("secondCtrl", function ($scope) { 17. $scope.$on('eventName', function (event, args) { 18. $scope.message = args.message; 19. console.log($scope.message); 20. }); 21. }); 22. 聽 23. </script> 24. </head> 25. <body ng-app="app"> 26. <div ng-controller="firstCtrl" style="border:2px solid *#E75D5C; padding:5px;*"> 27. <h1>Parent Controller</h1> 28. <input ng-model="msg"> 29. <button ng-click="handleClick(msg);">Broadcast</button> 30. <br /><br /> 31. <div ng-controller="secondCtrl" style="border:2px solid *#428bca;padding:5px;*"> 32. <h1>Child Controller</h1> 33. <p>Broadcast Message : </p> 34. </div> 35. </div> 36. </body> 37. </html>  How it works.. IMG_258 $on It listen on events of a given type. It can catch the event dispatched by $broadcast and $emit. Note If there is no parent-child relation between your scopes you can inject $rootScope into the controller and broadcast the event to all child scopes but you cannot emit your event.  You can emit your event only when you have parent-child relation and event propagation is initiated by child. However, $emit can fire an event only for all $rootScope.$on listeners.  **69- 58- 69 sme question regarding custom directive**   1. **naming standards to define custom directives?**   Directives have camel cased names such as ngBind. The directive can be invoked by translating the camel case name into snake case with these special characters :, -, or \_. Optionally the directive can be prefixed with x-, or data- to make it HTML validator compliant. **AngularJS: How to name directives** Quick tips to avoid issues with custom directive names  AngularJS directives allow us to use our own vocabulary to create semantic HTML components. To take advantage of that we should be aware of how it works to avoid some annoying and hard to find issues. **Differences between defining and using** Before start digging into the best practices, let's understand the fundamentals. Let's see how to define and then, how to use an directive. **Defining (the JS)** To define a new custom directives all we need to do is use the **directive**method which expects two parameters: a name and a constructor function.  angular.module('myDirectives', []).  directive('myAwesomeDirective', function() {  return {  restrict: 'E',  template: '<h1>This is awesome!</h1>'  };  });  ****In this case, the name must be written using camelCase and each capital letter will represent a dash when we use this directive.**** **Using (the HTML)** <my-awesome-directive></my-awesome-directive>  ****In this case, the name must be written using dashes and each dash represents a capital letter from the directive definition.****  The [$compile](https://docs.angularjs.org/api/ng/service/$compile" \t "https://medium.com/@cironunesdev/_blank) converts from camelCase to dash-separated behind the scenes. So, make sure you use the names correctly otherwise it won't work. **Use an unique preffix** I see a lot of times people starting with Angular, which by instinct name their directives with one word like: ****<dialog>****or****<tabs>****. The reason why this is not a good ideia is because we can have new HTML elements in the future (HTML6?) named exactly with the same name of our custom directive. BTW this is the case of the ****<dialog>****element.  ****To avoid naming collision with future HTML elements, use a preffix as a kind of namespace****  A recommended and wide spread convention from the Angular community is to have****two letters preffix names**** that identifies your application or library.  Examples:   * ng- (from the core) * ui- (from angular-ui) * my- (common in examples) * wm- (Wealth Management project)   ****Make sure to also avoid the "ng-" which is used for AngularJS core directives.**** **Conclusion** TLDR;  Use camelCase in the JS and dash-separated in the HTML as expected by the AngularJS compiler. Choose an unique two letters preffix for your directives and avoid the "ng-" preffix.  **71. how we can communicate angular with nodejs or any server side tech??**  I’m assuming you have already set your enviroment, so we are going to start by creating our NodeJS project, to do that I’m going to use [express](http://expressjs.com/" \t "http://4dev.tech/2016/01/tutorial-integrating-angularjs-with-nodejs/_blank), if you don’t have it yet just install via npm with the following command line:   |  | | --- | | npm install express-generator -g |   With express installed simply navigate via terminal to the folder you want your project to be located and execute the command ****express**** passing your project name as a parameter, like this:   |  | | --- | | express myNodeProject |   When you do it you’ll see that express has already created the basic project structure for you:  [IMG_256](http://4dev.tech/wp-content/uploads/2016/01/angularnodefolder.png)Folder Structure  By default, express uses Jade as the view engine, as you can see in the ****views**** folder we already have some jade files, which we are not going use, you can get rid of them later if you want. We’ll need to make some changes here to be able to use angularjs. Let’s begin by opening the file ****routes/index.js****, it contains the following code:   |  | | --- | | **var** express = require('express');**var** router = express.Router();  */\* GET home page. \*/*  router.**get**('/', **function**(req, res, next) {  res.render('index', { title: 'Express' });});    module.exports = router; |   That’s the code responsible for rendering and redirecting to the ****index.jade**** file, what we need to do here is to change it a little bit and make it redirect to our angularjs ****index.html**** file (which we are going to create in a moment). After the changes the code should look like this:   |  | | --- | | **var** express = require('express');**var** router = express.Router();**var** path = require('path');  */\* GET home page. \*/*  router.**get**('/', **function**(req, res, next) {  res.sendFile(path.join(\_\_dirname, '../', 'views', 'index.html'));});    module.exports = router; |   Now we need to create our ****index.html**** file, I’m going to put it in the ****views**** folder with the jade files. This is how my HTML code looks like:   |  | | --- | | <!DOCTYPE html>**<html** ng-app="angularjsNodejsTutorial"**>**  **<head>**  **<title>**Integrating AngularJS with NodeJS**</title>**  **<script** src="http://cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.2/angular.js"**></script>**  **<script** src="javascripts/app.js" type="text/javascript"**></script>** *<!-- not created yet -->*  **</head>**  **<body** **>**  **<div** ng-controller="myController"**>**  **</div>**  **</body></html>** |   In this file you can use not only AngularJS, but any javascript library you want, if you run your project you’ll see that nodejs is already redirecting to this file, you can now create you angularjs module and start writing some angularjs code as usual.  At this point you are already using angular and node in your application, but they are kind of independent from each other, there are no communication between them. To finish up this tutorial I’m going to show how AngularJS can make a request to NodeJS. But before we do that, we need to come back to the ****index.js**** file (the one we just modified) and create a function to return some data, which will be called when we make the request from our angularjs controller, just copy the following code and paste it right above the ****module.exports**** on your ****index.js****.   |  | | --- | | router.**get**('/data', **function**(req,res){  res.json([{"id": 1, "name": "Mymm", "city": "Pantano do Sul"},  {"id": 2, "name": "Skyble", "city": "Guilmaro"},  {"id": 3, "name": "Tagfeed", "city": "Gnosjö"},  {"id": 4, "name": "Realcube", "city": "Jrashen"},  {"id": 5, "name": "Bluejam", "city": "Zhangjiawo"},  {"id": 6, "name": "Jayo", "city": "Obonoma"},  {"id": 7, "name": "Cogidoo", "city": "Sungsang"},  {"id": 8, "name": "Avavee", "city": "Diawara"},  {"id": 9, "name": "Tagtune", "city": "Monywa"},  {"id": 10, "name": "Centimia", "city": "Retkovci"}]);}); |   What is happening here is that the ****router.get**** function is assigning a function to the url ****‘/data’****, so when the user types ****‘/data’**** in the browser, node will call this function, which is doing nothing more than returning a json, it could be getting and handling data from the database, but as I want to keep it simple, a static json will do the job.  Now let’s create our ****app.js**** file, as you can see in the HTML code, it’s already referenced there. I’m going to put it on the ****public/javascript**** folder.   |  | | --- | | **var** app = angular.module('angularjsNodejsTutorial',[]);  app.controller('myController', **function**($scope, $http) {  $scope.data = [];  **var** request = $http.**get**('/data');  request.success(**function**(data) {  $scope.data = data;  });  request.error(**function**(data){  console.log('Error: ' + data);  });}); |   This is also a very straightforward code, I’m using the function ****$http.get**** with the argumentet ****‘/data’**** to make the request, then I’m assigning the result to ****$scope.data****.  Now we just need to modify a little bit our HTML to make it iterate over our data and show it on the screen, just add this to the div with the ****ng-controller****:   |  | | --- | | **<ul** ng-repeat="item in data"**>**  **<li>**Name: {{item.name}}, City: {{item.city}}**</li></ul>** |   That’s it!! Now just run your project by executing ****npm start**** on the terminal and you’ll be able to access the application on your browser by typing ****localhost:3000****, if you did everything right you should see a page with the data from our json.   1. **what’s the dff named routing and nested routing** 2. **write validations using ngmessages?**  **[#Validation Messages without ngMessages](https://scotch.io/tutorials/angularjs-form-validation-with-ngmessages" \l "validation-messages-without-ngmessages)** Let’s take a quick look at how forms look without using this module.  <form name="userForm">  <input  type="text"  name="username"  ng-model="user.username"  ng-minlength="3"  ng-maxlength="8"  required>  <!-- show an error if username is too short -->  <p ng-show="userForm.username.$error.minlength">Username is too short.</p>  <!-- show an error if username is too long -->  <p ng-show="userForm.username.$error.maxlength">Username is too long.</p>  <!-- show an error if this isn't filled in -->  <p ng-show="userForm.username.$error.required">Your username is required.</p>  </form>  We are explicitly showing each error message only if that error exists. This can get tedious when we have multiple errors that we want to show.  This is where ngMessages comes in. This module brings some sanity to validation messages. **[#A Quick Look at ngMessages](https://scotch.io/tutorials/angularjs-form-validation-with-ngmessages" \l "a-quick-look-at-ngmessages)** Let’s take the above example and see how that would look in ngMessages.  <form name="userForm">  <input  type="text"  name="username"  ng-model="user.username"  ng-minlength="3"  ng-maxlength="8"  required>    <div ng-messages="userForm.name.$error">  <p ng-message="minlength">Your name is too short.</p>  <p ng-message="maxlength">Your name is too long.</p>  <p ng-message="required">Your name is required.</p>  </div>  </form>  Much simpler! ngMessages will handle showing and hiding specific messages based on the errors. ngMessages is basically looping through the userForm.name.$errors object and displaying messages based on that. **[#Using ngMessages](https://scotch.io/tutorials/angularjs-form-validation-with-ngmessages" \l "using-ngmessages)** The setup for ngMessages is very simple. We just need to load the module after Angular and then inject it into our application. **LOAD DEPENDENCIES AND INJECT** <!-- load angular --><script src="//code.angularjs.org/1.4.0/angular.js"></script>  <!-- load ngmessages --><script src="//ajax.googleapis.com/ajax/libs/angularjs/1.4.0/angular-messages.js"></script>  <!-- load our custom app --><script src="app.js"></script>  Now we can inject into our application in the app.js file.  angular.module('app', ['ngMessages']); **SHOW MESSAGES** Just use the ng-messages directive and pass in the field you want and its $error object.  This is the format:  <div ng-messages="<formName>.<inputName>.$error">  <p ng-message="<validationName>">Your message here.</p></div> **[#A Sample App](https://scotch.io/tutorials/angularjs-form-validation-with-ngmessages" \l "a-sample-app)** Let’s create a simple sample application to demonstrate ngMessages in practice. We’re working from a [Plunkr](http://plnkr.co/edit/ZAaObu?p=preview). You can create a new one for yourself or just follow along in the code provided. **THE HTML** We’re going to be using some very simple HTML here. We just need a form after all.  Here’s our index.html file:  <!DOCTYPE html><html><head>  <meta charset="utf-8">  <title>ngMessages Demo</title>    <!-- CSS -->  <!-- load bootstrap and add some custom css -->  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootswatch/3.3.4/cerulean/bootstrap.min.css">  <style>body { padding-top:50px; }</style>    <!-- JS -->  <script src="//code.angularjs.org/1.4.0/angular.js"></script>  <script src="//ajax.googleapis.com/ajax/libs/angularjs/1.4.0/angular-messages.js"></script>  <script src="app.js"></script></head><body class="container" ng-app="app" ng-controller="MainCtrl as main">    <!-- create our form -->  <form name="userForm" novalidate>    <!-- name field -->  <div class="form-group">  <label>Name</label>  <input type="text" name="name" class="form-control"  ng-model="main.name"  ng-minlength="5"  ng-maxlength="10"  required>    <!-- ngMessages goes here -->  </div>    <!-- email field -->  <div class="form-group">  <label>Email</label>  <input type="email" name="email" class="form-control"  ng-model="main.email"  ng-minlength="5"  ng-maxlength="20"  required>    <!-- ngMessages goes here -->  </div>    <div class="form-group">  <button type="submit" class="btn btn-danger">Submit</button>  </div>    </form>  </body></html>  This will be the starting template for our app. We are using novalidate on our form so that we disable the HTML5 validations. We have our own validations and they look much better. **OUR ANGULAR APP** We’ve started our HTML. Now we just need to create the Angular application that we already referenced in app.js, ng-app and ng-controller.  Create an app.js file and use the following:  angular  .module('app', ['ngMessages'])  .controller('MainCtrl', MainCtrl);  function MainCtrl() {}  We don’t need to have anything in our controller right now, but this is where you would process your form. **NGMESSAGES FOR**NAME The name field only has three validations (minlength, maxlength, required).  Here are the ng-messages for this input:  <div class="help-block" ng-messages="userForm.name.$error" ng-if="userForm.name.$touched">  <p ng-message="minlength">Your name is too short.</p>  <p ng-message="maxlength">Your name is too long.</p>  <p ng-message="required">Your name is required.</p></div> **NGMESSAGES FOR**EMAIL For our email input, let’s take a different approach. If our form has multiple fields, then it can be tedious to create multiple ng-messages blocks. ngMessages gives us the ability to pull messages from an external file.  This means we can reuse the same messages for multiple fields! **[#Reusable ngMessages with File](https://scotch.io/tutorials/angularjs-form-validation-with-ngmessages" \l "reusable-ngmessages-with-file)** Let’s create a new file called messages.html. We can place all of our messages in this file and just call it with ng-messages-include.  Here’s the messages.html file:  <p ng-message="required">This field is required</p><p ng-message="minlength">This field is too short</p><p ng-message="maxlength">This field is too long</p><p ng-message="required">This field is required</p><p ng-message="email">This needs to be a valid email</p>  Now we can use it for our email input:  <div class="help-block" ng-messages="userForm.email.$error">  <div ng-messages-include="messages.html"></div></div>  This makes ngMessages very powerful and reusable. **[#Only Showing Messages After Blur](https://scotch.io/tutorials/angularjs-form-validation-with-ngmessages" \l "only-showing-messages-after-blur)** Let’s say we wanted to only show error messages after a user has clicked out of the input that they are typing into. It isn’t very intuitive to show errors even before a user has used an input.  Angular provides a simple way to do this. We just have to use [ngShow](https://scotch.io/tutorials/how-to-use-ngshow-and-nghide) and the $touched validation feature Angular provides.  For example, we can only show errors for the name input using the following:  <div class="help-block" ng-messages="userForm.name.$error" ng-show="userForm.name.$touched">  ...</div>  Now these messages will only show after an input blur. **[#Adding Bootstrap Error Classes](https://scotch.io/tutorials/angularjs-form-validation-with-ngmessages" \l "adding-bootstrap-error-classes)** We also want to use the Bootstrap provided classes (.has-error) to highlight the field as red if there is an error. We can use [ngClass](https://scotch.io/tutorials/the-many-ways-to-use-ngclass) to add the error class if the field is $invalid.  <div class="form-group" ng-class="{ 'has-error': userForm.name.$touched && userForm.name.$invalid }">  We are adding the .has-error class if this field has been $touched and $invalid. **[#Conclusion](https://scotch.io/tutorials/angularjs-form-validation-with-ngmessages" \l "conclusion)** With this simple module, Angular form validation has gotten that much easier. Try it out in your own applications and let us know how you like it. Do you prefer ngMessages, a third-party software like [angular-formly](https://scotch.io/tutorials/easy-angularjs-forms-with-angular-formly) or doing validation from scratch?   1. **what u mean by dependency injector and how to inject ??**  * It is a process where we inject the dependent objects rather then consumer creating the objects. DI is every where in Angular, Angular won’t work with out DI. * (or) * It is a software design pattern that deals with how components get hold of their dependencies. The **AngularJS** injector subsystem is in charge of creating components, resolving their dependencies, and providing them to other components as requested.   [Dependency Injection](https://en.wikipedia.org/wiki/Dependency_injection) has a nice and complex explanation on Wikipedia and elsewhere. In AngularJS however it is somewhat different and I think I need a simple explanation.  Normally you define a function that expects certain parameters and the responsibility of the user of that function to pass in the correct parameters in the correct order.  The user will know what parameters and in what order to pass by reading the documentation of your code, or if there is not enough documentation by reading the source code.  E.g. if you define a function  function store\_user (name, password, email) {  }  Then the user will have to call the function passing values that will be assigned to the respective variables:  store\_user('Foo Bar', 'secret', 'foo@bar.com'); Tell Angular what are your parameters When you create a controller, a service, or some other part of the AngularJS ecosystem, you need to declare an anonymous function that actually implements that AngularJS element.  You can write something like this:  angular.module('DemoApp', [])  .controller('DemoController', ['$scope', '$log', function($scope, $log) {  $scope.message = "Hello World";  $log.debug('logging hello');  }]);  Here the controller method receives two parameters. The first is the name of the controller ('DemoController'), the second is an array. In the array the last element is the anonymous function while all the elements before are the names of the objects Angular needs to pass to the function in the order it needs to pass them. Inside the function declaration we have the same names in the same order. Note however that outside the function declaration those are strings holding the names while inside they are the real variable name.  **examples/angular/dependency\_injection\_full.html**   1. <!DOCTYPE html> 2. <html lang="en"> 3. <head> 4. <meta charset="utf-8"> 5. <meta name="viewport" 6. content="width=device-width, initial-scale=1, user-scalable=yes"> 7. <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.5/angular.min.js"></script> 9. <script> 10. angular.module('DemoApp', []) 11. .controller('DemoController', ['$scope', '$log', function($scope, $log) { 12. $scope.message = "Hello World"; 13. $log.debug('logging hello'); 14. }]); 16. </script> 18. </head> 19. <body ng-app="DemoApp" ng-controller="DemoController"> 21. <h1>Main Title</h1> 23. </body> 24. </html>   [Try!](https://code-maven.com/try/examples/angular/dependency_injection_full.html" \t "https://code-maven.com/_new)  [$scope](https://docs.angularjs.org/guide/scope) and [$log](https://docs.angularjs.org/api/ng/service/$log) are two objects provided by AngularJS.  This is called **Dependency Injection**. That based on the values in the array Angular will know what object to "inject" into the function.  **75. How can we make Ajax calls using angular js????**  **- AngularJs provides $http service for making ajax requests to remote servers.**  **- AngularJS provides $https: control which works as a service to read data from the server.**  **- The server makes a database call to get the desired records.**    **- AngularJS needs data in JSON format.**  **- Once the data is ready, $https: can be used to get the data from server in the following manner -**    **function studentController($scope,$https:) {**  **var url = "data.txt";**  **$https:.get(url).success( function(response) {**  **$scope.students = response;**  **});**  **}**  **- Here, the file data.txt contains student records.**  **$https: service makes an ajax call and sets response to its property students.**    **students model can be used to draw tables in HTML.**  **----------------------------------------------------------------------------------------------------------------------------------------------------------------------**  **76. features and advantages and life cycle of angular js??**    Features Of AngularJS:  1) MVC Architecture:    - The first important feature, which comes to my mind, is the MVC or Model-View-Controller architecture.  \_ The MVC architecture comprises of three important elements, the model, view and controller  \_ Model View Controller is a software design pattern for developing web application.  2) MODULE:  \_ AngularJS modules divides your application into modules.  so that you can manage things easily, reusable and functional components which can be integrated with other web applications.  Each module has a unique name and it can be dependent on other modules.  EX.<script type="text/javascript">  angular.module('testApp',[]);  angular.module('testApp',['dependentModule1','dependentModule2']);  </script>  3)Two Way Data Binding:  - AngularJS provides two-way data-binding to handle the Automatic data synchronization between model and view.  - The data-binding directives helps you to bind your model data to your app view.  EX.  <div ng-controller='testController'>  <input ng-model='name'/><!-- two way -->  <p>Hello {{name}}</p> <!-- one way -->  </div>  <script>function testController($scope){  $scope.name = 'online tutorials from ustutorials.com'    }  </script>  4)DIRECTIVES:  - AngularJS directives are used to extend the HTML vocabulary.  i.e they decorate html elements with new behaviors and help to manipulate html elements attributes in interesting way.  \_ There are some built-in directives provided by angularjs like as ng-app, ng-controller, ng-repeat, ng-model etc.  You can also create your own custom directive.  5)TEMPLATES:  \_ AngularJS, templates are written with HTML that contains AngularJS-specific elements and attributes.  \_ AngularJS used these templates to show information from the model and controller.  \_ These can be a single file (like index.html) or multiple views in one page using "partials".  6)SERVICES:  \_ Services are reusable singleton objects that are used to organize and share code across your app.  \_ They can be injected into controllers, filters, directives.  \_ AngularJS provides many predifined services for example, $https:, $route, $window, $location etc.  \_ Inbuilt services are always prefixed with $ symbol.  \_ factory(),provider(),value(),constant(),service() these are custome services.  7)ROUTING:    there are two types of routings ngRoute and ui.router.  ngRoute:-    - AngularJS Routing helps you to divide your app into multiple views and bind different views to Controllers.  \_ in ngRoute module With the $routeProvider you can also define a controller for each "view".  \_$routeProvider service provides method .when() and .otherwise() to define the routes for your app.  ui.router:-  \_ Angular UI-Router is a client-side Single Page Application routing framework for AngularJS.  \_  Advantages Of AngularJS:  \_ Built by Google: AngularJS has been developed as well as maintained by dedicated Google engineers.  \_ AngularJS provides capability to create Single Page Application in a very clean and maintainable way.  \_ AngularJS provides data binding capability to HTML thus giving user a rich and responsive experience.  \_ AngularJS code is unit testable.  \_ AngularJS provides reusable components.  \_ With AngularJS, developer write less code and get more functionality.  \_ AngularJS applications can run on all major browsers and smart phones including Android and iOS based phones/tablets.  \_ In AngularJS, views are pure html pages, and controllers written in JavaScript do the business processing.  Life Cycle Of AngularJS:  The three phases of the life cycle of an AngularJS application happen each time a web page is loaded in the browser.  The following sections describe these phases of an AngularJS application.  1) The Bootstrap Phase:- The first phase of the AngularJS life cycle is the bootstrap phase,  which occurs when the AngularJS JavaScript library is downloaded to the browser.  AngularJS initializes its own necessary components and then initializes your module,  which the ng-app directive points to. The module is loaded,  and any dependencies are injected into your module and made available to code within the module.  2) The Compilation Phase:-The second phase of the AngularJS life cycle is the HTML compilation stage.  Initially when a web page is loaded, a static form of the DOM is loaded in the browser.  During the compilation phase, the static DOM is replaced with a dynamic DOM that represents the AngularJS view.  This phase involves two parts: traversing the static DOM and collecting all the directives  and then linking the directives to the appropriate JavaScript functionality in the AngularJS  built-in library or custom directive code. The directives are combined with a scope to produce the dynamic or live view.  3) The Runtime Data Binding Phase:- The final phase of the AngularJS application is the runtime phase,  which exists until the user reloads or navigates away from a web page. At that point,  any changes in the scope are reflected in the view, and any changes in the view are directly updated in the scope,  making the scope the single source of data for the view.  AngularJS behaves differently from traditional methods of binding data.    Traditional methods combine a template with data received from the engine and then manipulate the  DOM each time the data changes. AngularJS compiles the DOM only once and then links the compiled template as necessary,  making it much more efficient than traditional methods.  ---------------------------------------------------------------------------------------------------------------------------------------------------------------------    **77. how do we hide html element by button click in AngularJS? With ng-hide directive we can hide eliment by button click.**    EX.  <!DOCTYPE html>  <html ng-app="plunker">  <head>  <meta charset="utf-8" />      <title>AngularJS Plunker</title>    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css" rel="stylesheet">    <link rel="stylesheet" href="style.css" />    <script data-require="angular.js@1.3.x" src="https://code.angularjs.org/1.3.17/angular.js" data-semver="1.3.17"></script>    <script src="script.js"></script>    </head>    <body ng-controller="MainCtrl">    <p>Hello {{name}}!</p>    <button class="btn btn-primary" ng-click="showDiv=true; hideMe()">Show Div</button>    <div ng-hide="showDiv">I was hidden now you see me, but how do I hide the button?</div>      </body>  </html>      **78. difference between one way data binding and two data dat binding.**  AngularJS One-Way Data Binding:-  - In One-Way data binding, view (UI part) not updates automatically when data model  changed and we need to write custom code to make it updated every time.  - Its not a synchronization processes and it will process data in one way.    Ex.  <!DOCTYPE html>  <html>  <head>  <title> AngularJs Two Binding Example </title>  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.6.0/angular.min.js"></script>  <script type="text/javascript">  var app = angular.module('angulartwobindapp', []);  app.controller('angulartwobindingCtrl', function ($scope) {  $scope.name = 'Welcome to Tutlane.com';  });  </script>  </head>  <body ng-app="angulartwobindapp">  <div ng-controller="angulartwobindingCtrl">  <p> Message: {{ name }} </p>  </div>  </body>  </html>  AngularJS Two-Way Data Binding:-  - In Two-way data binding, view (UI part) updates automatically when data model changed.  Its synchronization processes and two way data binding.  - We can achieve this two-way data binding using ng-model directive.  -If we use ng-model directive in html control it will update value automatically whenever data got changed in input control.  Ex.  <!DOCTYPE html>  <html>  <head>  <title> AngularJs Two Binding Example </title>  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.6.0/angular.min.js"></script>  <script type="text/javascript">  var app = angular.module('angulartwobindapp', []);  app.controller('angulartwobindingCtrl', function ($scope) {  $scope.name = 'Welcome to Tutlane.com';  });  </script>  </head>  <body ng-app="angulartwobindapp">  <div ng-controller="angulartwobindingCtrl">  Enter Name : <input type="text" ng-model="name" style="width:250px" />  <p> Entered Name: {{ name }} </p>  </div>  </body>  </html>   1. **Give the differences between AngularJS and Backbone and Knockout?**   **Comparison with Backbone.js and Knockout.js**   |  |  |  |  | | --- | --- | --- | --- | | **Comparison** | **AngularJs** | **Backbone.js** | **Knockout.js** | | File Size | ~142 KB total (compressed and minified) | ~ 7.3 KB total (gzip / minified) | ~21 KB total (gzip / minified) | | Version & Licence | V1.4.2 & MIT (Open-source) | V1.2.1 & MIT (Open-source) | V3.3.0 & MIT (Open-source) | | Dependencies | No Dependencies | Dependends on underscore.js and jQuery | No Dependencies | | Data Binding | It supports full data binding and provides options for creating custom data bindings | Does not support data binding by default but does using plugins for data bindings | It fully supports data binding and can bind many attributes. It provides options for creating custom data bindings | | Routing | It supports routing feature and it's very simple | It supports routing features and it's very simple | Does not support routing by defualt but it is available with some thrid-party libraries | | Views | Uses HTML as the templating language | Does not have templates by default but we can add them easily by a thrid-party template like underscore.js and handlebars | It uses HTML as the templating language | | Testing | Can support Test Driven Development (TDD) | Does not support testing by defualt but we can use some thrid-party tester like Jasmine and Sinon.JS | Does not support testing by defualt but we can use some thrid-party tester like Jasmine and Sinon.JS | | Data | Does not support jQuery but we can use Angular's $http | Can support jQuery's $.ajax and is very easy to understand | It can support jQuery's $.ajax and knockout mapping | | Design Pattern | Can support the MVC and MVVM design patterns | It can support MVP design pattern | It can support the MVVM design pattern | | Browser | Can support IE 9, IE 10 and IE 11 | It dependends on jQuery supporting browsers like IE 6+, Chrome, Firefox, Safari 5.1+ and Opera | It can support all major browsers like IE 6+, Firefox 3.5+, Chrome, Opera and Safari | | Third-party Integration | Does not support third-party integration | Does not support third-party integration | It supports third-party integration | | Documentation | It is available documentation and community | To my knowledge there is no documentation | It has available documentation and community |   **Conclusion**  This article helps you to understand the AngularJs comparison with Backbone.js, Knokout.js and its features.  **80.Explain $watch(), $watchgroup() and $watchCollection() functions of scope?**  Angular uses $watch APIs to observe model changes on the scope.  Angular registered watchers for each variable on scope to observe the change in its value. If the value, of variable on scope is changed then the view gets updated automatically. $watch APIs has following methods to observe model changes on the scope. $watch This function is used to observe changes in a variable on the $scope. It accepts three parameters: expression, listener and equality object, where listener and equality object are optional parameters.   1. **$watch(watchExpression, listener, [objectEquality])**   Here, watchExpression is the expression in the scope to watch. This expression is called on every $digest() and returns the value that is being watched.  The listener defines a function that is called when the value of the watchExpression changes to a new value. If the watchExpression is not changed then listener will not be called.  The objectEquality is a boolean type which is used for comparing the objects for equality using angular.equals instead of comparing for reference equality.   1. **<script>** 2. **scope.name = 'shailendra';** 3. **scope.counter = 0;** 5. **scope.$watch('name', function (newVal, oldVal) {** 6. **scope.counter = scope.counter + 1;** 7. **});** 8. **</script>**  $watchgroup This function is introduced in Angular1.3. This works the same as $watch() function except that the first parameter is an array of expressions to watch.   1. **$watchGroup(watchExpression, listener)**   The listener is passed as an array with the new and old values for the watched variables. The listener is called whenever any expression in the watchExpressions array changes.   1. **<script>** 2. **$scope.teamScore = 0;** 3. **$scope.time = 0;** 4. **$scope.$watchGroup(['teamScore', 'time'], function(newVal, oldVal) {** 5. **if(newVal[0] > 20){** 6. **$scope.matchStatus = 'win';** 7. **}** 8. **else if (newVal[1] > 60){** 9. **$scope.matchStatus = 'times up';** 10. **});** 11. **</script>**  $watchCollection This function is used to watch the properties of an object and fires whenever any of the properties change. It takes an object as the first parameter and watches the properties of the object.   1. **$watchCollection(obj, listener)**   The listener is called whenever anything within the obj has been changed.   1. **<script>** 2. **$scope.names = ['shailendra', 'deepak', 'mohit', 'kapil'];** 3. **$scope.dataCount = 4;** 5. **$scope.$watchCollection('names', function (newVal, oldVal) {** 6. **$scope.dataCount = newVal.length;** 7. **});** 8. **</script>**  What do you think? I hope you will enjoy the watchers in AngularJS while developing your app with AngularJS. I would like to have feedback from my blog readers. Your valuable feedback, question, or comments about this article are always welcome   1. **What is the difference between $watch, $digest and $apply?**   AngularJS has a powerful data binding mechanism. When you do changes in a variable on the $scope object within your view, the $scope object auto-magically updates itself. Similarly, whenever the $scope object changes, the view updates itself with the new value. AngularJS handle data-binding mechanism with the help of three powerful functions: $watch(), $digest() and $apply(). Most of the time AngularJS will call the $scope.$watch() and $scope.$digest() functions for you, but in some cases you may have to call these functions yourself to update new values. Therefore it is really good to know how they work. $watch() The $scope.watch() function is used to observe changes in a variable on the $scope. It accepts three parameters : expression, listener and equality object where listener and equality object are optional parameters.   1. **<!DOCTYPE html>** 2. **<html>** 3. **<head>** 4. **<title>AngularJS Watch</title>** 5. **<script src="lib/angular.js"></script>** 6. **<script>** 7. **var myapp = angular.module("myapp", []);** 8. **var myController = myapp.controller("myController", function ($scope) {** 9. **$scope.name = 'dotnet-tricks.com';** 10. **$scope.counter = 0;** 11. **$scope.$watch('name', function (newValue, oldValue) {** 12. **$scope.counter = $scope.counter + 1;** 13. **});** 14. **});** 15. **</script>** 16. **聽** 17. **</head>** 18. **<body ng-app="myapp" ng-controller="myController">** 19. **<input type="text" ng-model="name" />** 20. **<br />** 21. **Counter:** 22. **</body>** 23. **</html>**  $digest() The $scope.$digest() function iterates through all the watches in the $scope object, and its child $scope objects (if it has any). When $digest() iterates over the watches, it checks if the value of the expression has changed. If the value has changed, AngularJS calls the change callback(listener) with the new value and the old value.  The $digest() function is called whenever AngularJS thinks it is necessary. For example, after a button click, or after an AJAX call. You may have some corner cases where AngularJS does not call the $digest() function for you. In that case you may have to call this function yourself.   1. **<!DOCTYPE html>** 2. **<html>** 3. **<head>** 4. **<title>AngularJS Digest</title>** 5. **<script src="lib/angular.js"></script>** 6. **<script>** 7. **var myapp = angular.module("myapp", []);** 8. **var myController = myapp.controller("myController", function ($scope) {** 9. **聽** 10. **$scope.datetime = new Date();** 11. **聽** 12. **$scope.updateTime = function () {** 13. **$scope.datetime = new Date();** 14. **}** 15. **聽** 16. **document.getElementById("updateTimeButton").addEventListener('click', function () {** 17. **console.log("update time clicked");** 18. **$scope.datetime = new Date();** 19. **聽** 20. **console.log($scope.datetime);** 21. **});** 22. **});** 23. **</script>** 24. **聽** 25. **</head>** 26. **<body ng-app="myapp" ng-controller="myController">** 27. **<button ng-click="updateTime()">Update time - ng-click</button>** 28. **<button id="updateTimeButton">Update time</button>** 29. **<br />** 31. **</body>** 32. **</html>**   When you will click on second button, the data binding is not updated. Since $scope.$digest() is not called after the second button's event listener is executed. In this way on clicking the second button the time will be updated in the $scope.data.time variable, but the new time will never displayed.  To fix this issue you need to add a $scope.$digest() call to the second button event listener, like this:   1. **<script type="text/javascript">** 2. **document.getElementById("updateTimeButton").addEventListener('click', function () {** 3. **console.log("update time clicked");** 4. **$scope.datetime = new Date();** 5. **聽** 6. ***//to update $scope*** 7. **$scope.$digest();** 8. **console.log($scope.datetime);** 9. **});</script>**  $apply() Angular do auto-magically updates only those model changes which are inside AngularJS context. When you do change in any model outside of the Angular context (like browser DOM events, setTimeout, XHR or third party libraries), then you need to inform Angular of the changes by calling $apply() manually. When the $apply() function call finishes AngularJS calls $digest() internally, so all data bindings are updated.  In above example, instead of calling $digest() function inside the button listener function you can used the $apply() function like this:   1. **<script>** 2. **document.getElementById("updateTimeButton").addEventListener('click', function () {** 3. **$scope.$apply(function () {** 4. **console.log("update time clicked");** 5. **$scope.datetime = new Date();** 6. **聽** 7. **console.log($scope.datetime);** 8. **});** 9. **});** 10. **</script>**  Note $digest() is faster than $apply(), since $apply() triggers watchers on the entire scope chain while $digest() triggers watchers on the current scope and its children(if it has).  When error occurs in one of the watchers, $digest() can not handled errors via $exceptionHandler service, In this case you have to handle exception yourself. While $apply() uses try catch block internally to handle errors and if error occurs in one of the watchers then it passes errors to $exceptionHandler service.  **82.What are ng-repeat special variables?**  -The ng-repeat directive has a set of special variables which you are useful while iterating the collection. These variables are as follows:  1) $index  2) $first  3) $middle  4) $last  - The $index contains the index of the element being iterated. The $first,  $middle and $last returns a boolean value depending on whether the current item is the first, middle or last element in the collection being iterated.   1. **What is scope hierarchy? OR What is scope inheritance?**   The $scope object used by views in AngularJS are organized into a hierarchy. There is a root scope, and the root scope has one or more child scopes. Each view has its own $scope (which is a child of the root scope), so whatever variables one view controller sets on its $scope variable, those variables are invisible to other controllers.  Look at this AngularJS code example:  <!DOCTYPE html>  <html lang="en">  <head>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.5/angular.min.js"></script>  </head>  <body ng-app="myapp">  <div ng-controller="myController1">  {{data.theVar}}  </div>  <div ng-controller="myController2">  {{data.theVar}}  </div>  <script>  var module = angular.module("myapp", []);  var myController1 = module.controller("myController1", function($scope) {  $scope.data = { theVar : "Value One"};  });  var myController2 = module.controller("myController2", function($scope) {  $scope.data = { theVar : "Value Two"};  });  </script>  </body>  </html>  This example contains two views, each with their own controller function. Each controller sets the variable data.theVar to different values.  When this example is executed, the $scope hierarchy will look like this:   * Root $scope   + $scope for myController 1   + $scope for myController 2   As you can see, the $scope object used by the two controllers are not the same $scope object. That is also why the example above would write out two different values for the data bindings {{data.theVar}} inside the two views. The two controller functions for the views set different values for the data.theVar variable in each their own $scope object.  **84. What is $scope and $rootScope?**  - $scope is an object that is accessible from current component e.g Controller,  Service only. $rootScope refers to an object which is accessible from everywhere of the application.    You can think $rootScope as global variable and $scope as local variables  **85. What is event handling in AngularJS?**  - When we want to create advanced AngularJS applications such as User Interaction Forms,  then we need to handle DOM events like mouse clicks, moves, keyboard presses, change events and so on.  AngularJS has a simple model for how to add event listeners.  We can attach an event listener to an HTML element using one of the following AngularJS event listener directives:  ng-click  `` ng-dbl-click  ng-mousedown  ng-mouseup  ng-mouseenter  ng-mouseleave  ng-mousemove  ng-mouseover  ng-keydown  ng-keyup  ng-keypress  ng-change  **86-79 same questions.**   1. **What is core module in AngularJS?**   The angular.module is a global place for creating, registering and retrieving AngularJS modules. All modules (AngularJS core or 3rd party) that should be available to an application must be registered using this mechanism.  Passing one argument retrieves an existing [angular.Module](https://docs.angularjs.org/api/ng/type/angular.Module), whereas passing more than one argument creates a new [angular.Module](https://docs.angularjs.org/api/ng/type/angular.Module) Module A module is a collection of services, directives, controllers, filters, and configuration information.angular.module is used to configure the [$injector](https://docs.angularjs.org/api/auto/service/$injector).  // Create a new modulevar myModule = angular.module('myModule', []);  // register a new service  myModule.value('appName', 'MyCoolApp');  // configure existing services inside initialization blocks.  myModule.config(['$locationProvider', function($locationProvider) {  // Configure existing providers  $locationProvider.hashPrefix('!');}]);  Then you can create an injector and load your modules like this:  var injector = angular.injector(['ng', 'myModule'])  However it's more likely that you'll just use [ngApp](https://docs.angularjs.org/api/ng/directive/ngApp) or [angular.bootstrap](https://docs.angularjs.org/api/ng/function/angular.bootstrap) to simplify this process for you. Usage angular.module(name, [requires], [configFn]);   | **Param** | **Type** | **Details** | | --- | --- | --- | | name | **[string](https://docs.angularjs.org/)** | The name of the module to create or retrieve. | | requires  (optional) | **[!Array.<string>=](https://docs.angularjs.org/)** | If specified then new module is being created. If unspecified then the module is being retrieved for further configuration. | | configFn  (optional) | **[Function=](https://docs.angularjs.org/)** | Optional configuration function for the module. Same as [Module#config()](https://docs.angularjs.org/api/ng/type/angular.Module" \l "config). |  ArgumentsReturns  |  |  | | --- | --- | | **[angular.Module](https://docs.angularjs.org/)** | new module with the [angular.Module](https://docs.angularjs.org/api/ng/type/angular.Module) api. |  1. **Can angular applications (ng-app) be nested within each other?**  * **AngularJS applications** cannot be **nested within each other**. Only one**AngularJS application can** be auto-bootstrapped per HTML document. The first**ngApp** found in the document **will** be used to define the root element to auto-bootstrap as an **application**   **89.How to handel event in Angularjs ?**   * To handle an event that is emitted or broadcasted, you use the $on() method. * Syntax : scope.$on(*name,* listener)  1. **Explain ngClick And ngDblclick Directives In AngularJS?**  |  | | --- | | **AngularJS ng-click and ng-dblclick event example** | | * Syntax for AngularJS click event.   <input ng-click="toggledisplay()" />   * Syntax for AngularJS double click event.   <input ng-dblclick="toggledbldisplay()" />  **Example:**   * **ng-click example:**  Click the below button to Show/Hide the div and change the button text.   **AngularJS Tutorial - Click event example.**   * **ng-dblclick example:**  Double click the below button to Show/Hide the div and change the button text.   **AngularJS Tutorial- Double click event example.**  **Source code:**  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"> <html xmlns="http://www.w3.org/1999/xhtml"> <head>     <title></title>     <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script> </head> <body>     <div ng-app="SampleApp" ng-controller="SampleCntrl">         <ul class="exmpul">             <li><strong>ng-click example:</strong>                 <br />                 Click the below button to Show/Hide the div and change the button text.                 <br />                 <br />                 <input type="button" ng-click="toggledisplay();" value="{{btntext}}" />                 <br />                 <br />                 <div style="background: green; color: White; font-size: 20px; font-weight: bold;                     padding: 10px;" ng-show="divdisp">                     AngularJS Tutorial - Click event example.                 </div>                 <br />             </li>             <li><strong>ng-dblclick example:</strong>                 <br />                 Double click the below button to Show/Hide the div and change the button text.                 <br />                 <br />                 <input type="button" ng-dblclick="toggledbldisplay();" value="{{btndbltext}}" />                 <br />                 <br />                 <div style="background: green; color: White; font-size: 20px; font-weight: bold;                     padding: 10px;" ng-show="divdbldisp">                     AngularJS Tutorial- Double click event example.                 </div>             </li>         </ul>     </div>     <script>         var myapp = angular.module("SampleApp", []);         myapp.controller("SampleCntrl", function ($scope) {             $scope.btntext = "Hide";             $scope.btndbltext = "Hide";             $scope.divdisp = true;             $scope.divdbldisp = true;             $scope.toggledisplay = function () {                 if ($scope.btntext == "Hide") {                     $scope.btntext = "Show";                     $scope.divdisp = false;                 }                 else {                     $scope.btntext = "Hide";                     $scope.divdisp = true;                 }             };              $scope.toggledbldisplay = function () {                 if ($scope.btndbltext == "Hide") {                     $scope.btndbltext = "Show";                     $scope.divdbldisp = false;                 }                 else {                     $scope.btndbltext = "Hide";                     $scope.divdbldisp = true;                 }             };         });     </script> </body> </html> |  1. **What is Constants in AngularJS?**  Constants Create an AngularJS Constant for vendor libraries’ global variables.  Why?: Provides a way to inject vendor libraries that otherwise are globals. This improves code testability by allowing you to more easily know what the dependencies of your components are (avoids leaky abstractions). It also allows you to mock these dependencies, where it makes sense.  // constants.js  /\* global toastr:false, moment:false \*/  (function() { 'use strict';  angular  .module('app.core')  .constant('toastr', toastr)  .constant('moment', moment);  })();  Use constants for values that do not change and do not come from another service. When constants are used only for a module that may be reused in multiple applications, place constants in a file per module named after the module. Until this is required, keep constants in the main module in a constants.js file.  Why?: A value that may change, even infrequently, should be retrieved from a service so you do not have to change the source code. For example, a url for a data service could be placed in a constants but a better place would be to load it from a web service.  Why?: Constants can be injected into any angular component, including providers.  Why?: When an application is separated into modules that may be reused in other applications, each stand-alone module should be able to operate on its own including any dependent constants.  // Constants used by the entire app  angular  .module('app.core')  .constant('moment', moment);  // Constants used only by the sales module  angular  .module('app.sales')  .constant('events', {  ORDER\_CREATED: 'event\_order\_created',  INVENTORY\_DEPLETED: 'event\_inventory\_depleted'  });   1. **How AngularJS handle the security?**  Security This document explains some of AngularJS's security features and best practices that you should keep in mind as you build your application. Reporting a security issue Email us at [security@angularjs.org](mailto:security@angularjs.org) to report any potential security issues in AngularJS.  Please keep in mind the points below about AngularJS's expression language. Use the latest AngularJS possible Like any software library, it is critical to keep AngularJS up to date. Please track the [CHANGELOG](https://github.com/angular/angular.js/blob/master/CHANGELOG.md) and make sure you are aware of upcoming security patches and other updates.  Be ready to update rapidly when new security-centric patches are available.  Those that stray from AngularJS standards (such as modifying AngularJS's core) may have difficulty updating, so keeping to AngularJS standards is not just a functionality issue, it's also critical in order to facilitate rapid security updates. AngularJS Templates and Expressions ****If an attacker has access to control AngularJS templates or expressions, they can exploit an AngularJS application via an XSS attack, regardless of the version.****  There are a number of ways that templates and expressions can be controlled:   * ****Generating AngularJS templates on the server containing user-provided content****. This is the most common pitfall where you are generating HTML via some server-side engine such as PHP, Java or ASP.NET. * ****Passing an expression generated from user-provided content in calls to the following methods on a** [scope](https://docs.angularjs.org/guide/scope)**:   + $watch(userContent, ...)   + $watchGroup(userContent, ...)   + $watchCollection(userContent, ...)   + $eval(userContent)   + $evalAsync(userContent)   + $apply(userContent)   + $applyAsync(userContent) * ****Passing an expression generated from user-provided content in calls to services that parse expressions****:   + $compile(userContent)   + $parse(userContent)   + $interpolate(userContent) * ****Passing an expression generated from user provided content as a predicate to** orderBy **pipe****:{{ value | orderBy : userContent }}  Sandbox removal Each version of AngularJS 1 up to, but not including 1.6, contained an expression sandbox, which reduced the surface area of the vulnerability but never removed it. ****In AngularJS 1.6 we removed this sandbox as developers kept relying upon it as a security feature even though it was always possible to access arbitrary JavaScript code if one could control the AngularJS templates or expressions of applications.****  Control of the AngularJS templates makes applications vulnerable even if there was a completely secure sandbox:   * <https://ryhanson.com/stealing-session-tokens-on-plunker-with-an-angular-expression-injection/> in this blog post the author shows a (now closed) vulnerability in the Plunker application due to server-side rendering inside an AngularJS template. * <https://ryhanson.com/angular-expression-injection-walkthrough/> in this blog post the author describes an attack, which does not rely upon an expression sandbox bypass, that can be made because the sample application is rendering a template on the server that contains user entered content.   ****It's best to design your application in such a way that users cannot change client-side templates.****   * Do not mix client and server templates * Do not use user input to generate templates dynamically * Do not run user input through $scope.$eval (or any of the other expression parsing functions listed above) * Consider using [CSP](https://docs.angularjs.org/api/ng/directive/ngCsp) (but don't rely only on CSP)   ****You can use suitably sanitized server-side templating to dynamically generate CSS, URLs, etc, but not for generating templates that are bootstrapped/compiled by AngularJS.****  ****If you must continue to allow user-provided content in an AngularJS template then the safest option is to ensure that it is only present in the part of the template that is made inert via the** [ngNonBindable](https://docs.angularjs.org/api/ng/directive/ngNonBindable)**directive.**** HTTP Requests Whenever your application makes requests to a server there are potential security issues that need to be blocked. Both server and the client must cooperate in order to eliminate these threats. AngularJS comes pre-configured with strategies that address these issues, but for this to work backend server cooperation is required. Cross Site Request Forgery (XSRF/CSRF) Protection from XSRF is provided by using the double-submit cookie defense pattern. For more information please visit [XSRF protection](https://docs.angularjs.org/api/ng/service/$http" \l "cross-site-request-forgery-xsrf-protection). JSON Hijacking Protection Protection from JSON Hijacking is provided if the server prefixes all JSON requests with following string ")]}',\n". AngularJS will automatically strip the prefix before processing it as JSON. For more information please visit [JSON Hijacking Protection](https://docs.angularjs.org/api/ng/service/$http" \l "json-vulnerability-protection).  Bear in mind that calling $http.jsonp, like in [our Yahoo! finance example](https://docs.angularjs.org/guide/concepts" \l "accessing-the-backend), gives the remote server (and, if the request is not secured, any Man-in-the-Middle attackers) instant remote code execution in your application: the result of these requests is handed off to the browser as regular <script> tag. Strict Contextual Escaping Strict Contextual Escaping (SCE) is a mode in which AngularJS requires bindings in certain contexts to require a value that is marked as safe to use for that context.  This mode is implemented by the [$sce](https://docs.angularjs.org/api/ng/service/$sce) service and various core directives.  One example of such a context is rendering arbitrary content via the [ngBindHtml](https://docs.angularjs.org/api/ng/directive/ngBindHtml) directive. If the content is provided by a user there is a chance of Cross Site Scripting (XSS) attacks. The [ngBindHtml](https://docs.angularjs.org/api/ng/directive/ngBindHtml) directive will not render content that is not marked as safe by [$sce](https://docs.angularjs.org/api/ng/service/$sce). The [ngSanitize](https://docs.angularjs.org/api/ngSanitize) module can be used to clean such user provided content and mark the content as safe.  ****Be aware that marking untrusted data as safe via calls to** [$sce.trustAsHtml](https://docs.angularjs.org/api/ng/service/$sce" \l "trustAsHtml)**, etc is dangerous and will lead to Cross Site Scripting exploits.****  For more information please visit [$sce](https://docs.angularjs.org/api/ng/service/$sce) and [$sanitize](https://docs.angularjs.org/api/ngSanitize/service/$sanitize). Using Local Caches There are various places that the browser can store (or cache) data. Within AngularJS there are objects created by the [$cacheFactory](https://docs.angularjs.org/api/ng/service/$cacheFactory). These objects, such as [$templateCache](https://docs.angularjs.org/api/ng/service/$templateCache) are used to store and retrieve data, primarily used by [$http](https://docs.angularjs.org/api/ng/service/$http) and the [script](https://docs.angularjs.org/api/ng/directive/script) directive to cache templates and other data.  Similarly the browser itself offers localStorage and sessionStorage objects for caching data.  ****Attackers with local access can retrieve sensitive data from this cache even when users are not authenticated.****  For instance in a long running Single Page Application (SPA), one user may "log out", but then another user may access the application without refreshing, in which case all the cached data is still available.  For more information please visit [Web Storage Security](https://www.whitehatsec.com/blog/web-storage-security/).   1. What is the difference between link and compile functions in Angular.js?   In compile phase the angular parser starts parsing the DOM and whenever the parser encounters a directive it creates a function. These functions are termed as template or compiled functions. In this phase we do not have access to the $scope data. In the link phase the data i.e. ($scope) is attached to the template function and executed to get the final HTML output.  Compile – It works on template. It’s like adding a class element in to the DOM (i.e., manipulation of tElement = template element), hence manipulations that apply to all DOM clones of the template associated with the directive.  Link – It works on instances. Usually used for registering DOM listeners (i.e., $watch expressions on the instance scope) as well as instance DOM manipulation. (i.e., manipulation of iElement = individual instance element).  94. What is ng-cloak ?   * The ng-cloak directive prevents the document from showing unfinished AngularJS code while AngularJS is being loaded. * AngularJS applications can make HTML documents flicker when the application is being loaded, showing the AngularJS code for a second, before all code are executed. Use the ng-cloak directive to prevent this. * The ng-cloak directive has no parameters  Example Prevent the application from flicker at page load:  <div ng-app="">  <p ng-cloak>{{ 5 + 5 }}</p>  </div>  [Try it Yourself »](https://www.w3schools.com/angular/tryit.asp?filename=try_ng_ng-cloak" \t "https://www.w3schools.com/angular/_blank) Definition and Usage The ng-cloak directive prevents the document from showing unfinished AngularJS code while AngularJS is being loaded.  AngularJS applications can make HTML documents flicker when the application is being loaded, showing the AngularJS code for a second, before all code are executed. Use the ng-cloak directive to prevent this. Syntax <element ng-cloak></element>  Supported by all HTML elements. Parameter Values The ng-cloak directive has no parameters.   1. **Mention what are the styling form that ngModel adds to CSS classes ?**  |  |  | | --- | --- | | |  | | --- | | **ngModel adds these CSS classes to allow styling of form as well as control**   •ng- valid   •ng- invalid   •ng-pristine   •ng-dirty | |   **96.Explain directives ng-if, ng-switch and ng-show.**  ngif  ----  ===>The ng-if directive removes the HTML element if the expression evaluates to false.  ===>If the if statement evaluates to true, a copy of the Element is added in the DOM.  ===>The ng-if directive is different from the ng-hide, which hides the display of the element,  where the ng-if directive completely removes the element from the DOM.  ===>The ng-if directive will destroy (completely removes) the element or recreate the element.  ==>Supported by all HTML elements.  ==>This directive can add / remove HTML elements from the DOM based on an expression.  ==>If the expression is true, it add HTML elements to DOM, otherwise HTML elements are removed from the DOM.  Syntax:  ==>ng-if, on the other hand, actually removes the element from the DOM  when the condition is false and only adds the element back once the condition turns true.  ------  <element ng-if="expression"></element>  Parameter Values:  ----------------  Value Description  ----- -----------  expression An expression that will completely remove the element if it returns false.  If it returns true, a copy of the element will be inserted instead.  EXAMPLE:  -------  <!DOCTYPE html>  <html>  <body ng-app="">  Keep HTML: <input type="checkbox" ng-model="myVar" ng-init="myVar = true">  <div ng-if="myVar">  <h1>Welcome</h1>  <p>Welcome to my if directive</p>  <hr>  </div>  <p>The DIV element will be removed when the checkbox is not checked.</p>  <p>The DIV element will return, if you check the checkbox.</p>  </body>  </html>  97.What is ng-repeat directive?  ng-repeat:  ---------  ==>The ng-repeat directive repeats a set of HTML, a given number of times.  ==>The set of HTML will be repeated once per item in a collection.  ==>The collection must be an array or an object.  Note: Each instance of the repetition is given its own scope, which consist of the current item.  ----  ==>If you have an collection of objects, the ng-repeat directive is perfect for making a HTML table,  displaying one table row for each object, and one table data for each object property.  Syntax:  ------  <element ng-repeat="expression"></element>  Supported by all HTML elements.  Parameter Values  -----------------  Value Description  ----- ----------  expression An expression that specifies how to loop the collection.  Legal Expression examples:  x in records  (key, value) in myObj  x in records track by $id(x)  Example:  -------  <!DOCTYPE html>  <html>  <body ng-app="myApp" ng-controller="myCtrl">  <h1 ng-repeat="x in records">{{x}}</h1>  <script>  var app = angular.module("myApp", []);  app.controller("myCtrl", function($scope) {  $scope.records = [  "Sachin Tendulkar",  "A.P.J Abdul Kalam",  "Abraham Lincoln",  "Nelson Mandela",  ]  });  </script>  </body>  </html>  **98. Explain ng-include directive?**  AngularJS ng-include Directive  ------------------------------  ==>The ng-include directive is useful if we want to include an external resource in an HTML template.  Definition and Usage:  --------------------  ==>The ng-include directive includes HTML from an external file.  ==>The included content will be included as childnodes of the specified element.  ==>The value of the ng-include attribute can also be an expression, returning a filename.  ==>By default, the included file must be located on the same domain as the document.  ==>Supported by all HTML elements.  Syntax  ------  <element ng-include="filename" onload="expression" autoscroll="expression" ></element>  (or)  ==>The ng-include directive can also be used as an element:  <ng-include src="filename" onload="expression" autoscroll="expression" ></ng-include>  Parameter Values  ----------------  Value Description  ----- -----------  filename A filename, written with apostrophes, or an expression which returns a filename.  onload Optional. An expression to evaluate when the included file is loaded.  autoscroll Optional. Whether or not the included section should be able to scroll  into a specific view.  AngularJS - Includes  ---------------------  ==>HTML does not support embedding html pages within html page.  ==> To achieve this functionality following ways are used −  Using Ajax − Make a server call to get the corresponding html page and  set it in innerHTML of html control.  Using Server Side Includes − JSP, PHP and other web side server technologies can  include html pages within a dynamic page.  Using AngularJS, we can embed HTML pages within a HTML page using ng-include directive.  <div ng-app = "" ng-controller = "studentController">  <div ng-include = "'main.html'"></div>  <div ng-include = "'subjects.html'"></div>  </div>  ==>By default, HTML does not provide the facility to include client side code from other files.  ==>  Client Side includes  Server Side Includes  AngularJS Includes  Client Side includes  --------------------  One of the most common ways is to include HTML code is via Javascript.  JavaScript is a programming language which can be used to manipulate the content  in an HTML page on the fly. Hence, Javascript can also be used to include code from other files.  Server Side Includes  --------------------  Server Side Includes are also available for including a common piece  of code throughout a site. This is normally done for including content  in the below parts of an HTML document.  Page header  Page footer  Navigation menu.  Note:  -----  The virtual parameter is used to specify the file  (HTML page, text file, script, etc.) that needs to be included.  If the web server process does not have access to read the file or execute the script,  the include command will fail. The 'virtual' word is a keyword that is required  to be placed in the include directive.  AngularJS Includes  ------------------  ==>Angular provides the function to include the functionality from other  AngularJS files by using the ng-include directive.  ==>The primary purpose of the "ng-include directive" is used to fetch,  compile and include an external HTML fragment in the main AngularJS application.  Summary:  --------  ==>By default, we know that HTML does not provide a direct way to include HTML content  from other files like other programming languages.  ==>Javascript along with JQuery is the best-preferred option  for embedding HTML content from other files.  ==>Another way of including HTML content from other files is to use the  <include> directive and the virtual parameter keyword.  ==>The virtual parameter keyword is used to denote the file which needs to be embedded.  This is known as server-side includes.  ==>Angular also provides the facility to include files by using the ng-include directive.  This directive can be used to inject code from external files directly into the main HTML file.  Example:  -------  <!DOCTYPE html>  <html>  <body ng-app="">  <div ng-include="'include.html'"></div>  </body>  </html>  include.html  -----------  <!DOCTYPE html>  <html>  <body>  <h1>Include Header..!!</h1><br><br>  <h5>This text has been included into the HTML page, using ng-include!</h5>  </body>  </html> |  |
|  |  |  |

1. What is template in angularjs?

# Templates

In AngularJS, templates are written with HTML that contains AngularJS-specific elements and attributes. AngularJS combines the template with information from the model and controller to render the dynamic view that a user sees in the browser.

These are the types of AngularJS elements and attributes you can use:

* [Directive](https://docs.angularjs.org/guide/directive) — An attribute or element that augments an existing DOM element or represents a reusable DOM component.
* [Markup](https://docs.angularjs.org/api/ng/service/$interpolate) — The double curly brace notation {{ }} to bind expressions to elements is built-in AngularJS markup.
* [Filter](https://docs.angularjs.org/guide/filter) — Formats data for display.
* [Form controls](https://docs.angularjs.org/guide/forms) — Validates user input.

The following code snippet shows a template with [directives](https://docs.angularjs.org/guide/directive) and curly-brace [expression](https://docs.angularjs.org/guide/expression) bindings:

<html ng-app>

<!-- Body tag augmented with ngController directive -->

<body ng-controller="MyController">

<input ng-model="foo" value="bar">

<!-- Button tag with ngClick directive, and

string expression 'buttonText'

wrapped in "{{ }}" markup -->

<button ng-click="changeFoo()">{{buttonText}}</button>

<script src="angular.js">

</body></html>

In a simple app, the template consists of HTML, CSS, and AngularJS directives contained in just one HTML file (usually index.html).

In a more complex app, you can display multiple views within one main page using "partials" – segments of template located in separate HTML files. You can use the [ngView](https://docs.angularjs.org/api/ngRoute/directive/ngView) directive to load partials based on configuration passed to the [$route](https://docs.angularjs.org/api/ngRoute/service/$route) service. The [AngularJS tutorial](https://docs.angularjs.org/tutorial/) shows this technique in steps seven and eight.

**100. Who created Angular JS ?**

AngularJS was created, as a side project, in 2009 by two developers, Misko Hevery and Adam Abrons. The two originally envisioned their project, GetAngular, to be an end-to-end tool that allowed web designers to interact with both the frontend and the backend.

Google.

One of the original creators, Adam Abrons stopped working on AngularJS but Misko Hevery and his manager, Brad Green, spun the original GetAngular project into a new project, named it AngularJS, and built a team to create an maintain it within Google.

One of AngularJS' first big wins internally at Google occurred when the company DoubleClick was acquired by Google and they started rewriting part of their application using AngularJS. Because of DoubleClick's initial success, Google seems to have invested even more resources into Angular and has blessed AngularJS for internal and external product usage.

Because of this, the Angular team inside Google has grown rapidly.

**101. What is manual bootstrap?**

If you need to have more control over the initialization process, you can use a manual bootstrapping method instead. Examples of when you'd need to do this include using script loaders or the need to perform an operation before AngularJS compiles a page.

Here is an example of manually initializing AngularJS:

<!doctype html>

<html>

<body>

<div ng-controller="MyController">

Hello {{greetMe}}!

</div>

<script src="http://code.angularjs.org/snapshot/angular.js"></script>

<script>

angular.module('myApp', [])

.controller('MyController', ['$scope', function ($scope) {

$scope.greetMe = 'World';

}]);

angular.element(function() {

angular.bootstrap(document, ['myApp']);

});

</script>

</body>

</html>

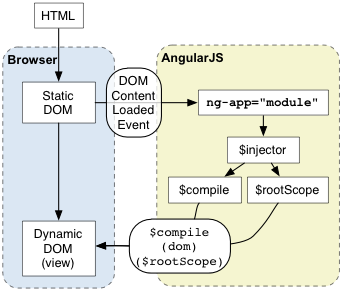
Note that we provided the name of our application module to be loaded into the injector as the second parameter of the [angular.bootstrap](https://docs.angularjs.org/api/ng/function/angular.bootstrap) function. Notice that angular.bootstrap will not create modules on the fly. You must create any custom [modules](https://docs.angularjs.org/guide/module) before you pass them as a parameter.

You should call angular.bootstrap() after you've loaded or defined your modules. You cannot add controllers, services, directives, etc after an application bootstraps.

This is the sequence that your code should follow:

1. After the page and all of the code is loaded, find the root element of your AngularJS application, which is typically the root of the document.
2. Call [angular.bootstrap](https://docs.angularjs.org/api/ng/function/angular.bootstrap) to [compile](https://docs.angularjs.org/guide/compiler) the element into an executable, bi-directionally bound application.

**102. What is Auto bootstrap?**



AngularJS initializes automatically upon DOMContentLoaded event or when the angular.js script is evaluated if at that time document.readyState is set to 'complete'. At this point AngularJS looks for the [ngApp](https://docs.angularjs.org/api/ng/directive/ngApp)directive which designates your application root. If the [ngApp](https://docs.angularjs.org/api/ng/directive/ngApp) directive is found then AngularJS will:

* load the [module](https://docs.angularjs.org/guide/module) associated with the directive.
* create the application [injector](https://docs.angularjs.org/api/auto/service/$injector)
* compile the DOM treating the [ngApp](https://docs.angularjs.org/api/ng/directive/ngApp)directive as the root of the compilation. This allows you to tell it to treat only a portion of the DOM as an AngularJS application.

<!doctype html>

<html ng-app="optionalModuleName">

<body>

I can add: {{ 1+2 }}.

<script src="angular.js"></script>

</body>

</html>

There a few things to keep in mind regardless of automatic or manual bootstrapping:

* While it's possible to bootstrap more than one AngularJS application per page, we don't actively test against this scenario. It's possible that you'll run into problems, especially with complex apps, so caution is advised.
* Do not bootstrap your app on an element with a directive that uses [transclusion](https://docs.angularjs.org/api/ng/service/$compile" \l "transclusion), such as [ngIf](https://docs.angularjs.org/api/ng/directive/ngIf), [ngInclude](https://docs.angularjs.org/api/ng/directive/ngInclude) and [ngView](https://docs.angularjs.org/api/ngRoute/directive/ngView). Doing this misplaces the app [$rootElement](https://docs.angularjs.org/api/ng/service/$rootElement) and the app's [injector](https://docs.angularjs.org/api/auto/service/$injector), causing animations to stop working and making the injector inaccessible from outside the app.

**103. Is AngularJS a library, framework, plugin or a browser extension?**

AngularJS is an open source web application framework. It was originally developed in 2009 by Misko Hevery and Adam Abrons. It is now maintained by Google. Its latest version is 1.4.3.

Definition of AngularJS as put by its [official documentation](https://docs.angularjs.org/guide/introduction" \t "_blank) is as follows −

AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. Angular's data binding and dependency injection eliminate much of the code you currently have to write. And it all happens within the browser, making it an ideal partner with any server technology.

Features

* AngularJS is a powerful JavaScript based development framework to create RICH Internet Application(RIA).
* AngularJS provides developers options to write client side application (using JavaScript) in a clean MVC(Model View Controller) way.
* Application written in AngularJS is cross-browser compliant. AngularJS automatically handles JavaScript code suitable for each browser.
* AngularJS is open source, completely free, and used by thousands of developers around the world. It is licensed under the Apache License version 2.0.

Overall, AngularJS is a framework to build large scale and high performance web application while keeping them as easy-to-maintain.

Core Features

Following are most important core features of AngularJS −

* **Data-binding** − It is the automatic synchronization of data between model and view components.
* **Scope** − These are objects that refer to the model. They act as a glue between controller and view.
* **Controller** − These are JavaScript functions that are bound to a particular scope.
* **Services** − AngularJS come with several built-in services for example $https: to make a XMLHttpRequests. These are singleton objects which are instantiated only once in app.
* **Filters** − These select a subset of items from an array and returns a new array.
* **Directives** − Directives are markers on DOM elements (such as elements, attributes, css, and more). These can be used to create custom HTML tags that serve as new, custom widgets. AngularJS has built-in directives (ngBind, ngModel...)
* **Templates** − These are the rendered view with information from the controller and model. These can be a single file (like index.html) or multiple views in one page using "partials".
* **Routing** − It is concept of switching views.
* **Model View Whatever** − MVC is a design pattern for dividing an application into different parts (called Model, View and Controller), each with distinct responsibilities. AngularJS does not implement MVC in the traditional sense, but rather something closer to MVVM (Model-View-ViewModel). The Angular JS team refers it humorously as Model View Whatever.
* **Deep Linking** − Deep linking allows you to encode the state of application in the URL so that it can be bookmarked. The application can then be restored from the URL to the same state.

**104. Does AngularJS has dependency on jQuery?**

AngularJS has no dependency on jQuery library. But it can be used with jQuery library.

If jQuery is available, angular.element is an alias for the [jQuery](http://api.jquery.com/jQuery/) function. If jQuery is not available, angular.element delegates to AngularJS's built-in subset of jQuery, called "jQuery lite" or **jqLite**.

jqLite is a tiny, API-compatible subset of jQuery that allows AngularJS to manipulate the DOM in a cross-browser compatible way. jqLite implements only the most commonly needed functionality with the goal of having a very small footprint.

To use jQuery, simply ensure it is loaded before the angular.js file. You can also use the [ngJq](https://docs.angularjs.org/api/ng/directive/ngJq) directive to specify that jqlite should be used over jQuery, or to use a specific version of jQuery if multiple versions exist on the page.

**105.What is injector?**

# $injector

$injector is used to retrieve object instances as defined by [provider](https://docs.angularjs.org/api/auto/service/$provide), instantiate types, invoke methods, and load modules.

The following always holds true:

var $injector = angular.injector();

expect($injector.get('$injector')).toBe($injector);

expect($injector.invoke(function($injector) {

return $injector;

})).toBe($injector);

## $inject Annotation

By adding an $inject property onto a function the injection parameters can be specified.

# The $inject property

If a function has an $inject property and its value is an array of strings, then the strings represent names of services to be injected into the function.

// Given

var MyController = function(obfuscatedScope, obfuscatedRoute) {

// ...

}

// Define function dependencies

MyController['$inject'] = ['$scope', '$route'];

// Then

expect(injector.annotate(MyController)).toEqual(['$scope', '$route']);

**106. What is the role of ng-app, ng-init and ng-model directives?**

1. **ng-app** – This is used to initialize an Angular.JS application. When this directive in placed in an HTML page, it basically tells Angular that this HTML page is an angular.js application.
2. The "ng-app" directive is added to our div tag to indicate that this application is an angular.js application. Note that the ng-app directive can be applied to any tag, so it can also be put in the body tag as well.
3. Because we have defined this application as an angular.js application, we can now make use of angular.js functionality. In our case, we are using expressions to simply concatenate 2 strings.

Example:

<html>

<body>

<div ng-app=””>

Name: {{ “angular”+”js”}}

</div>

</body>

</html>

1. **ng-init** – This is used to initialize application data. Sometimes you may require some local data for your application, this can be done with the ng-init directive.

The example below shows how to use the ng-init directive.

Example:

<html>

<body>

<div ng-init=”Name=’Angular js’”>

Name: {{ “Angular js}}

</div>

</body>

</html>

1. **ng-model** – And finally we have the ng-model directive, which is used to bind the value of an HTML control to application data. The example below shows how to use the ng-model directive.

In this example,

* We are going to create 2 variables called "quantity" and "price". These variables are going to be bound to 2 text input controls.
* We are then going to display the total amount based on the multiplication of both price and quantity values.

.

 <html>

<body>

<div ng-app=” ” ng-init=”quantity=1;price=5”>

People: <input type=”number” ng-model=”quantity”>

Registration price: <input type=”number” ng-model=”price”>

Total: {{ quantity + price }}

</div>

</body>

</html>

**107.What are Filters in AngularJS?**

Filters are used to change modify the data and can be clubbed in expression or directives using pipe character. Following is the list of commonly used filters.

|  |  |  |
| --- | --- | --- |
| **Sr.No.** | **Name** | **Description** |
| 1 | uppercase | converts a text to upper case text. |
| 2 | lowercase | converts a text to lower case text. |
| 3 | currency | formats text in a currency format. |
| 4 | filter | filter the array to a subset of it based on provided criteria. |
| 5 | orderby | orders the array based on provided criteria. |

**uppercase filter:**

Add uppercase filter to an expression using pipe character. Here we've added uppercase filter to print student name in all capital letters.

Example:

Enter first name:<input type = "text" ng-model = "student.firstName">

Enter last name: <input type = "text" ng-model = "student.lastName">

Name in Upper Case: {{student.fullName() | uppercase}}

## lowercase filter

Add lowercase filter to an expression using pipe character. Here we've added lowercase filter to print student name in all lowercase letters.

Example:

Enter first name:<input type = "text" ng-model = "student.firstName">

Enter last name: <input type = "text" ng-model = "student.lastName">

Name in Lower Case: {{student.fullName() | lowercase}}

## currency filter

Add currency filter to an expression returning number using pipe character. Here we've added currency filter to print fees using currency format.

Example:

Enter fees: <input type = "text" ng-model = "student.fees">

fees: {{student.fees | currency}}

## filter filter

To display only required subjects, we've used subjectName as filter.

Example:

Enter subject: <input type = "text" ng-model = "subjectName">

Subject:

<ul>

<li ng-repeat = "subject in student.subjects | filter: subjectName">

{{ subject.name + ', marks:' + subject.marks }}

</li>

</ul>

## orderby filter

To order subjects by marks, we've used orderBy marks.

Example:

Subject:

<ul>

<li ng-repeat = "subject in student.subjects | orderBy:'marks'">

{{ subject.name + ', marks:' + subject.marks }}

</li>

</ul>

**108.What is the size of angular.js file?**

The size of the compressed and minified file is < 36KB.

**109.What browsers AngularJS support?**

1. AngularJS will work with the latest versions of Chrome, Firefox, Safari, and Opera, as well as Internet Explorer version 8, 9, and 10. You will need to do some extra work for IE8  
2. To make AngularJS Work with IE8 use the following  
<html ng-app="application\_name" id="application\_name">  
Though ng-app="application\_name" is sufficient for the other browsers as mentioned in point 1; for IE 8, id attribute is also required  
3. We cannot make IE to recognize and include templates in the following manner  
<ng-include="'myInclude.tpl.html'"></ng-include>

Though, we can take a different strategy to make IE8 recognize and include templates by using  
<div ng-include="'myInclude.tpl.html'"></div>

we have to make IE8 recognize <ng-include=""> to be able to use this custom tag to include templates. We can do that by using

<head>  
<!--[if lte IE 8]>  
<script>  
document.createElement('ng-include');  
document.createElement('ng-view');  
. . .  
<![endif]-->  
</head>

4. Supporting IE7 for AngularJS  
You need to do everything that you did for IE8.  
AngularJS was not tested with IE7; so you need to adjust stuff as they come along  
IE7 does not have many APIs commonly present in web-browsers such as JSON API  
You will need libraries such as http://bestiejs.github.io/json3/ to be included in your application to support JSON

5. IE6 is not supported

**110. What is the use of filter in angularjs?**

Filters can be added in AngularJS to format data

AngularJS provides filters to transform data:

* currency Format a number to a currency format.
* date Format a date to a specified format.
* filter Select a subset of items from an array.
* json Format an object to a JSON string.
* limitTo Limits an array/string, into a specified number of elements/characters.
* lowercase Format a string to lower case.
* number Format a number to a string.
* orderBy Orders an array by an expression.
* uppercase Format a string to upper case.

**111. what are services in angularjs?**

A Service ia a reusable singleton ovject which is used to organize and share code across your app. A Service can be injected into controllers,filters,directives.

AngularJs offers several built-in services(like $http, $provide, $resource, $window, $parse) which always start with $ sign.

There are five ways to create a service as given below:

1. Service
2. Factory
3. Provider
4. Value
5. Constant
6. **What is scope in angularjs?**

Scope is a JavaScript object refers to the application model. It acts as a context for evaluating angular expressions. Basically, it acts as glue between controller and view.



Scope are hierarchical in nature and follow the DOM structure of your AngularJs app. AngularJs has two scope objects: $rootScope and $scope

**113. what is dynamic angular application?**

--> if view interacting with the controller,such type of angular application called as "DYNAMIC ANGULAR APPLICATION".

-->the following are the steps to create dynamic angular applications.

step-1: load the angular framework

eg: dymamic angular application

index.html

---

<html>

-------

-------

-------

<script

src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.1/angular.min.js"/>

</html>

step-2: declare the logical name for web application by using 'ng-app' directive.

eg:

<html ng-app="app">

--------

--------

</html>

step-3: create the controller by using 'ng-controller' directive.

eg:

<html ng-app="app">

<div ng-controller="ctrl">

--------

--------

</div>

</html>

step-4: instantiate the module

var obj=angular.mpdule("app",[]);

step-5: implement the controller.

obj.controller("ctrl",function($scope){

$scope.var\_one="angularjs";

$scope.var\_two="angular 2";

});

**114. what is static angular application?**

view doesnot interacting with the controller such type of angular applications are called "STATIC ANGULAR APPLICATIONS".

**116.What is ng-app?**

Use this directive to auto-bootstrap an AngularJS application. The ngApp directive designates the root element of the application and is typically placed near the root element of the page

Example: on the <body> or <html> tags.

There are a few things to keep in mind when using ngApp:

* only one AngularJS application can be auto-bootstrapped per HTML document. The first ngApp found in the document will be used to define the root element to auto-bootstrap as an application. To run multiple applications in an HTML document you must manually bootstrap them using [angular.bootstrap](https://docs.angularjs.org/api/ng/function/angular.bootstrap) instead.
* AngularJS applications cannot be nested within each other.
* Do not use a directive that uses [transclusion](https://docs.angularjs.org/api/ng/service/$compile" \l "transclusion) on the same element as ngApp. This includes directives such as [ngIf](https://docs.angularjs.org/api/ng/directive/ngIf), [ngInclude](https://docs.angularjs.org/api/ng/directive/ngInclude)and [ngView](https://docs.angularjs.org/api/ngRoute/directive/ngView). Doing this misplaces the app [$rootElement](https://docs.angularjs.org/api/ng/service/$rootElement) and the app's [injector](https://docs.angularjs.org/api/auto/service/$injector), causing animations to stop working and making the injector inaccessible from outside the app.

You can specify an **AngularJS module** to be used as the root module for the application. This module will be loaded into the [$injector](https://docs.angularjs.org/api/auto/service/$injector)when the application is bootstrapped. It should contain the application code needed or have dependencies on other modules that will contain the code.

**117 . Which applications are not good fit for Angularjs?**

AngularJS is known for creating dynamic web apps because of its feature of extending HTML syntax according to your application components. Other than that, it also provides a wide range of other prominent features like two way data binding, templates, MVC, dependency injection, directives, testing, and a lot more to build almost ***any kind of web app***.

AngularJS alone can’t create a web app. You need a lot of other technologies also, compatible with AngularJS to create a complete web app – say – backend, frontend, deployment technologies. So, in order to create a ***complete***web app, you need to atleast have a bit of knowledge of such other technologies as well.

**118. what is mvc?**

what is the use of mvc design pattern?

when we use mvc design pattern?

---> The Model-View-Controller (MVC) is an architectural pattern that separates an application into three main logical components:

1.model

2.view

3.controller

This pattern is used to separate application's concerns.

1) Model - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.

2)View - View represents the visualization of the data that model contains.

3)Controller - Controller acts on both model and view.

uses of mvc design pattern:-

---------------------------

(1).asp.net mvc provides clean seperation of concerns,so easy code maintainance will be possible.

(2).asp.net mvc provides better performance than asp.net webforms.asp.net mvc requested page lifecycle is very lightweighted,this requires less memory and processing time on the server to provide response to client

(3).asp.net mvc provides complete control over html,javascript,and css.this allows proper integration of latest web technologies like html5,css3,jquery,angular js.

(4).asp.net mvc supports built in support for url-routing.,this supports method based execuition .

(5).asp.net mvc supports unit testing

**119. When we use MVC design pattern.?**

In AngularJS the MVC pattern is implemented in JavaScript and HTML. The view is defined in HTML, while the model and controller are implemented in JavaScript. There are several ways that these components can be put together in AngularJS but the simplest form starts with the view.

Views

The view in an AngularJS application is created with HTML. By using a simple example, we can see that HTML can all exist inside a single page. However, almost all applications will quickly move past the single view model; in this case the fragments of HTML that make up the view will be stored in individual files. Take this HTML page as a starting point:

|  |  |
| --- | --- |
|  | <!DOCTYPE html> <html> <head> <title>Demo</title> <meta charset="utf-8" /> </head> <body> <div id="messageTitle"></div> <div id="message">Hello World</div> </body> </html> |

*A Simple HTML page without AngularJS*

At this point, it’s easy to consider this entire HTML document to be the view since there’s only one. However, because this is a Single Page Application, only a portion of the page represents the current view. In this case, the contents of the body represent the view while the HTML and HEAD elements make up the container for the individual views. In AngularJS it’s also popular to mark a specific element in the page as the view giving you more control over which portions of the page make up the container and which make up the changing view.

|  |  |
| --- | --- |
|  | <!DOCTYPE html> <html> <head> <title>Hack Hands Angular - Demos</title> <meta charset="utf-8" /> </head> <body> <h1>Hack Hands Angular Demos</h1> <div ng-view> <div id="messageTitle"></div> <div id="message">Hello World</div> </div> </body> </html> |

This HTML is not yet a functioning AngularJS view. There are a couple of steps required to include and bootstrap the AngularJS framework

1. Reference the AngularJS framework
2. Define the AngularJS application.

Referencing the AngularJS framework is a simple matter of adding a script tag referencing the framework script on a Content Delivery Network (CDN).

|  |  |
| --- | --- |
|  | <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.5/angular.min.js"></script> |

*Bootstrapping the Angular Application*

When AngularJS loads it will see this directive which will cause it to look for a module named “hackApp.” Modules are a key component of AngularJS that we’ll cover in more detail in a follow-up post. For now, it’s enough to know this view needs to be able to reference a module representing the AngularJS

**120.what is mvc?**

what is the use of mvc design pattern?

when we use mvc design pattern?

Model–View–Controller (**MVC**) is a software design pattern for implementing user interfaces on computers. It divides a given application into three interconnected parts in order to separate internal representations of information from the ways that information is presented to and accepted from the user.

---> The Model-View-Controller (MVC) is an architectural pattern that separates an application into three main logical components:

1.model

2.view

3.controller

This pattern is used to separate application's concerns.

1) Model :An architecture for building applications that separate the data (model) from the user interface (view) and the processing (controller). In practice, **MVC** views and controllers are often combined into a single object because they are closely related..

2)View - The **view** displays the model data, and sends user actions (e.g. button clicks) to the controller. The view can:

* be independent of both the model and the controller; or
* actually **be** the controller, and therefore depend on the model.

.

3)Controller The **controller** provides model data to the view, and interprets user actions such as button clicks. The controller depends on the view and the model. In some cases, the controller and the view are the same object.

uses of mvc design pattern:-

---------------------------

(1).asp.net mvc provides clean seperation of concerns,so easy code maintainance will be possible.

(2).asp.net mvc provides better performance than asp.net webforms.asp.net mvc requested page lifecycle is very lightweighted,this requires less memory and processing time on the server to provide response to client

(3).asp.net mvc provides complete control over html,javascript,and css.this allows proper integration of latest web technologies like html5,css3,jquery,angular js.

(4).asp.net mvc supports built in support for url-routing.,this supports method based execuition .

(5).asp.net mvc supports unit testing

**121. What are expressions in angularjs?**

AngularJS expressions are much like JavaScript expressions, placed inside HTML templates by using double braces such as: {{expression}}. AngularJS evaluates expesssions and then dynamically adds the result to a webpage. Like JavaScript expressions, they can contain literals, operators, and variables.

There are some valid AngularJs expressions:

Example:

* {{1+2}}
* {{x+y}}
* {{x==y}}
* {{x=2}}
* {{user.id}}

**122. What is restrict option in directive?**

The restrict option in angular directive, is used to specify how a directive will be invoked in your angular app i.e as a attribute, class, element or comment.

There are four valid options for restrict:

**Example:**

‘ A’ (Attribute) **-** <span my-directive></span>

‘ C’ (Class) **-** <span class=” my-directive:expression;”></span>

‘ E’ (Element) **-** < my-directive></ my-directive>

‘M’ (Comment) - <!—directive: my-directive expression -- >

**123. what are directives in angularjs? What are different ways to invoke a directive?**

AngularJS directives are a combinations of angularJS templates mark-ups(HTML attribute or elements, or CSS classes ) and supporting JavaScript code. The JavaScript directive code defines the template data and behaviours of the HTML elements.

AngularJS directives are used to extend the HTML vocabulary i.e they decoreate html elements with new behaviours and help to manipulate html elements attributes in interesting way.

There are some built –in directives provided by AngularJS like as ng-app, ng-controller, ng-repeat, ng-model etc.

**Ways to invoke a directive:**

There are four methods to invoke a directive in your angular app which are equivalent.

|  |  |
| --- | --- |
| **Method** | **Syntax** |
| **As an attribute** | **<span my-directive></span>** |
| **As a Class** | **<span class=”my-directive: expression;”></span>** |
| **As an element** | **<my-directive></my-directive>** |
| **As a comment** | **<!—directive: my-directive expression-->** |

**124. In Angularjs dependent on jquery?**

AngularJS has no dependency on JQuery library .But it can be used with jQuery library.

**125. What IDE'S are currently used for the development of angularjs?**

* Eclipse
* Visual Studio
* WebStorm
* TextMate
* SublimeText
* Atom
* Visual Studio Code