## What is AngularJS?

AngularJS is an open-source JavaScript framework developed by Google. It helps you to create single-page applications or one-page web applications that only require HTML, CSS, and JavaScript on the client side. It is based on MV-\* pattern and allow you to build well structured, easily testable, and maintainable front-end applications. AngularJS has changed the way to web development. It is not based on jQuery to perform its operations. In spite of using ASP.NET Web form, ASP.NET MVC, PHP, JSP, Ruby on Rails for web development, you can do your complete web development by using most powerful and adaptive JavaScript Framework AngularJS. There is no doubt, JavaScript frameworks like AngularJS, Ember etc. are the future of web development.

## Why to use AngularJS?

There are following reasons to choose AngularJS as a web development framework:

1. It is based on MVC pattern which helps you to organize your web apps or web application properly.
2. It extends HTML by attaching directives to your HTML markup with new attributes or tags and expressions in order to define very powerful templates.
3. It also allows you to create your own directives, making reusable components that fill your needs and abstract your DOM manipulation logic.
4. It supports two-way data binding i.e. connects your HTML (views) to your JavaScript objects (models) seamlessly. In this way any change in model will update the view and vice versa without any DOM manipulation or event handling.
5. It encapsulates the behavior of your application in controllers which are instantiated with the help of dependency injection.
6. It supports services that can be injected into your controllers to use some utility code to fulfil your need. For example, it provides $http service to communicate with REST service.
7. It supports dependency injection which helps you to test your angular app code very easily.
8. Also, AngularJS is mature community to help you. It has widely support over the internet.

## Why this project is called "AngularJS"?

Html has angle brackets i.e. <, > and ng sound like Angular. That’s why it is called AngularJS.

## What are the advantages of AngularJS?

There are following advantages of AngularJS:

* **Data Binding** - AngularJS provides a powerful data binding mechanism to bind data to HTML elements by using scope.
* **Customize & Extensible** - AngularJS is customized and extensible as per you requirement. You can create your own custom components like directives, services etc.
* **Code Reusability** - AngularJS allows you to write code which can be reused. For example custom directive which you can reuse.
* **Support** – AngularJS is mature community to help you. It has widely support over the internet. Also, AngularJS is supported by Google which gives it an advantage.
* **Compatibility** - AngularJS is based on JavaScript which makes it easier to integrate with any other JavaScript library and runnable on browsers like IE, Opera, FF, Safari, Chrome etc.
* **Testing** - AngularJS is designed to be testable so that you can test your AngularJS app components as easy as possible. It has dependency injection at its core, which makes it easy to test.

## How AngularJS is different from other JavaScript Framework?

Today, AngularJS is the most popular and dominant JavaScript framework for professional web development. It is well suited for small, large and any sized web app and web application. AngularJS is different from other JavaScript framework in following ways:

1. AngularJS mark-up lives in the DOM.
2. AngularJS uses plain old JavaScript objects (POJO).
3. AngularJS is leverages with Dependency Injection.

## What IDEs you can use for AngularJS development?

AngularJS development can be done with the help of following IDEs:

1. Visual Studio 2012, 2013, 2015 or higher
2. Eclipse
3. WebStorm
4. Sublime Text
5. TextMate

## Does AngularJS has dependency on jQuery?

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

## How to use jQuery with AngularJS?

By default AngularJS use jQLite which is the subset of jQuery. If you want to use jQuery then simply load the jQuery library before loading the AngularJS. By doing so, Angular will skip jQLite and will started to use jQuery library.

## Compare the features of AngularJS and jQuery?

The comparison of AngularJS and jQuery features are given below:

|  |  |  |
| --- | --- | --- |
| Features | jQuery | AngularJS |
|  |  |  |
| Abstract The DOM | Y | Y |
|  |  |  |
| Animation Support | Y | Y |
|  |  |  |
| AJAX/JSONP | Y | Y |
|  |  |  |
| Cross Module Communication | Y | Y |
|  |  |  |
| Deferred Promises | Y | Y |
|  |  |  |
| Form Validation | N | Y |
|  |  |  |
| Integration Test Runner | N | Y |
|  |  |  |
| Unit Test Runner | Y | Y |
|  |  |  |
| Localization | N | Y |
|  |  |  |
| MVC Pattern | N | Y |
|  |  |  |
| Template | N | Y |
|  |  |  |
| Two-way Binding | N | Y |
|  |  |  |
| One-way Binding | N | Y |
|  |  |  |
| Dependency Injection | N | Y |
|  |  |  |
| Routing | N | Y |
|  |  |  |
| Restful API | N | Y |
|  |  |  |

## What is jQLite or jQuery lite?

jQLite is a subset of jQuery that is built directly into AngularJS. jQLite provides you all the useful features of jQuery. In fact it provides you limited features or functions of jQuery. Here is a table of supported jQuery methods by jQLite.

|  |  |
| --- | --- |
| jQuery Method | Limitation, if any |
|  |  |
|  |  |
| addClass() |  |
|  |  |
| after() |  |
|  |  |
| append() |  |
|  |  |
| attr() |  |
|  |  |
| bind() | Does not support namespace, selectors and eventData |
|  |  |
| children | Does not support selectors |
|  |  |
| clone() |  |
|  |  |
| contents() |  |
|  |  |
| css() |  |
|  |  |
| data() |  |
|  |  |
| detach() |  |
|  |  |
| empty() |  |
|  |  |
| eq() |  |
|  |  |
| find() | Limited to lookups by tag name |
|  |  |
| hasClass() |  |
|  |  |
| html() |  |
|  |  |
| text() | Does not support selectors |
|  |  |
| on() | Does not support namespace, selectors and eventData |
|  |  |
| off() | Does not support namespace, selectors |
|  |  |
| one() | Does not support namespace, selectors |
|  |  |
| parent() | Does not support selectors |
|  |  |
| prepend() |  |
|  |  |
| prop |  |
|  |  |
| ready() |  |
|  |  |
| remove |  |
|  |  |
| removeAttr() |  |
|  |  |
| removeClass() |  |
|  |  |
| removeData() |  |
|  |  |
| replaceWith() |  |
|  |  |
| toggleClass() |  |
|  |  |
| triggerHandler() | Passes a dummy event object to handlers |
|  |  |
| unbind() | Does not support namespace |
|  |  |
| val() |  |
|  |  |
| wrap() |  |
|  |  |

## How to access jQLite or jQuery with AngularJS? OR What is angular.element() in AngularJS?

jQuery lite or the full jQuery library if available, can be accessed via the AngularJS code by using the element() function in AngularJS. Basically, ***angular.element() is an alias for the jQuery function i.e.***

angular.element() === jQuery() === $()

<html>

<head>

<script src="lib/angular.js"></script> <script type="text/javascript">

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

app.controller("mainCtrl", function ($scope, $element) {

$scope.clickme = function () {

var elem = angular.element(document.querySelector('#txtName')); console.log(elem.val())// console the value of textbox

};

});

</script>

</head>

<body ng-app="app">

<div ng-controller="mainCtrl">

<input type="text" id="txtName" value="Shailendra Chauhan"/>

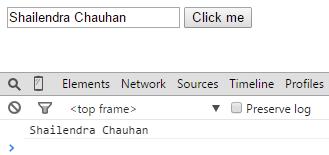
<button type="button" ng-click="clickme()">Click me</button>

</div>

</body>

</html>

When you will click on the Click me button, the value of textbox will be logged on console as given below:



## Is AngularJS a library, framework, plugin or a browser extension?

. AngularJS is a first class JavaScript framework which allows you to build well structured, easily testable, and maintainable front-end applications. It is not a library since library provides you limited functionality or has dependencies to other libraries.

It is not a plugin or browser extension since it is based on JavaScript and compatible with both desktop and mobile browsers.

## What browsers AngularJS support?

The latest version of AngularJS 1.3 support Safari, Chrome, Firefox, Opera 15+, IE9+ and mobile browsers (Android, Chrome Mobile, iOS Safari, Opera Mobile).

AngularJS 1.3 has dropped support for IE8 but AngularJS 1.2 will continue to support IE8.

## What is the size of angular.js file?

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

## What are AngularJS features?

The features of AngularJS are listed below:

* Modules
* Directives
* Templates
* Scope
* Expressions
* Data Binding
* MVC (Model, View & Controller)
* Validations
* Filters
* Services
* Routing
* Dependency Injection
* Testing

## How AngularJS handle the security?

AngularJS provide following built-in protection from basic security holes:

* Prevent HTML injection attacks.
* Prevent Cross-Site-Scripting (CSS) attacks.
* Prevent XSRF protection for server side communication.

Also, AngularJS is designed to be compatible with other security measures like Content Security Policy (CSP), HTTPS (SSL/TLS) and server-side authentication and authorization that greatly reduce the possible attacks.

## What are Modules in AngularJS?

AngularJS modules are containers just like namespace in C#. They divide an angular app into small, reusable and functional components which can be integrated with other angular app. Each module is identified by a unique name and can be dependent on other modules. In AngularJS, every web page (view) can have a single module assigned to it via ng-app directive.

<script type="text/javascript">

// defining module

angular.module('myApp', []);

//OR defining module which has dependency on other modules

angular.module('myApp', ['dependentModule1', 'dependentModule2']);

</script>

**Using an AngularJS module into your app**

You can bootstrap your app by using your AngularJS module as given below:

<html ng-app="myApp">

<head>

...

</head>

<body>

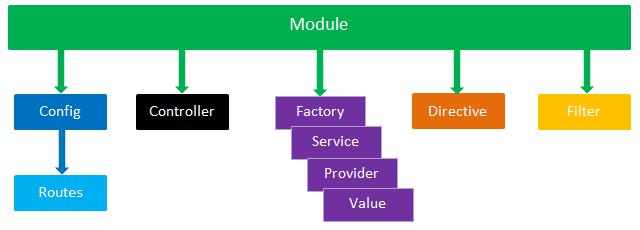
...

</body>

</html>

## What components can be defined within AngularJS modules?

You can define following components with in your angular module:



1. Directive
2. Filter
3. Controller
4. Factory
5. Service
6. Provider
7. Value
8. Config settings and Routes

## What is core module in AngularJS?

ng is the core module in angular. This module is loaded by default when an angular app is started. This module provides the essential components for your angular app like directives, services/factories, filters, global APIs and testing components.

## How angular modules load the dependencies?

An angular module use **configuration and run** blocks to inject dependencies (like ***providers, services and constants***) which get applied to the angular app during the bootstrap process.

## What is difference between config() and run() method in AngularJS?

Configuration block – This block is executed during the provider registration and configuration phase. **Only providers and constants can be injected into configuration blocks**. This block is used to inject module wise configuration settings to prevent accidental instantiation of services before they have been fully configured. This block is created using config() method.

angular.module('myModule', []).

config(function (injectables) { // provider-injector

//This is an example of config block.

//You can have as many of these as you want.

//You can only inject Providers (not instances)

//into config blocks.

}).

run(function (injectables) { // instance-injector

//This is an example of a run block.

//You can have as many of these as you want.

//You can only inject instances (not Providers)

//into run blocks

});

Run block – This block is executed after the configuration block. It is used to inject instances and constants. This block is created using run() method. This method is like as main method in C or C++.

The run block is a great place to put event handlers that need to be executed at the root level for the application. For example, authentication handlers.

## When dependent modules of a module are loaded?

A module might have dependencies on other modules. The dependent modules are loaded by angular before the requiring module is loaded.

In other words the configuration blocks of the dependent modules execute before the configuration blocks of the requiring module. The same is true for the run blocks. Each module can only be loaded once, even if multiple other modules require it.

## What is Global API?

Global API provides you global functions to perform common JavaScript tasks such as comparing objects, deep copying, iterating through objects, and converting JSON data etc. All global functions can be accessed by using the angular object. The list of global functions is given below:

|  |  |
| --- | --- |
| **Name** | **Description** |
| angular.lowercase | Converts the specified string to lowercase. |
| [angular.uppercase](https://docs.angularjs.org/api/ng/function/angular.uppercase) | Converts the specified string to uppercase. |
| angular.forEach | Invokes the iterator function once for each item in Obj. Collection, which can be either an object or an array. |
| angular.isUndefined | Determines if a reference is undefined. |
| angular.isDefined | Determines if a reference is defined. |
| angular.isObject | Determines if a reference is an Object. |
| angular.isString | Determines if a reference is a String. |
| angular.isNumber | Determines if a reference is a Number. |
| angular.isDate | Determines if a value is a date. |
| angular.isArray | Determines if a reference is an Array. |
| angular.isFunction | Determines if a reference is a Function. |
| angular.isElement | Determines if a reference is a DOM element (or wrapped jQuery element). |
| angular.copy | Creates a deep copy of source, which should be an object or an array. |
| angular.equals | Determines if two objects or two values are equivalent. Supports value types, regular expressions, arrays and objects |
| angular.bind | Returns a function which calls function fn bound to self |
| angular.toJson | Serializes input into a JSON-formatted string. Properties with leading $$ characters will be stripped since angular uses this notation internally. |
| angular.fromJson | Deserializes a JSON string. |
| angular.bootstrap | Use this function to manually start up angular application. |
| angular.injector | Creates an injector object that can be used for retrieving services as well as for dependency injection. |
| angular.reloadWithDebugInfo | Use this function to reload the current application with debug information turned on. |
| angular.element | Wraps a raw DOM element or HTML string as a [jQuery](http://jquery.com/) element. |
| angular.module | Used for creating, registering and retrieving Angular modules. |

## What is Angular Prefixes $ and $$?

To prevent accidental name collisions with your code, Angular prefixes names of public objects with $ and names of private objects with $$. So, do not use the $ or $$ prefix in your code.

## What are Filters in AngularJS?

Filters are used to format data before displaying it to the user. They can be used in view templates, controllers, services and directives. There are some built-in filters provided by AngularJS like as Currency, Date, Number, OrderBy, Lowercase, Uppercase etc. You can also create your own filters.

**Filter Syntax**

{{ expression | filter}}

**Filter Example**

<script type="text/javascript">

{{ 14 | currency }} //returns $14.00

</script>

## 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 expressions and then dynamically adds the result to a web page. Like JavaScript expressions, they can contain literals, operators, and variables.

There are some valid AngularJS expressions:

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

## How AngularJS expressions are different from the JavaScript expressions?

AngularJS expressions are much like JavaScript expressions but they are different from JavaScript expressions in the following ways:

* Angular expressions can be added inside the HTML templates.
* Angular expressions doesn't support control flow statements (conditionals, loops, or exceptions).
* Angular expressions support filters to format data before displaying it.

## What are Directives in AngularJS?

AngularJS directives are a combination of AngularJS template markups (HTML attributes or elements, orCSS classes) and supporting JavaScript code. The JavaScript directive code defines the template data and behaviors of the HTML elements.

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.

## What is the role of ng-app, ng-init and ng-model directives?

The main role of these directives is explained as:

* ng-app - Initialize the angular app.
* ng-init - Initialize the angular app data.
* ng-model - Bind the html elements like input, select, text area to angular app model.

## How to create custom directives in AngularJS?

You can create your own custom directive by using following syntax:

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

//creating custom directive syntax

app.directive("myDir", function () {

return {

restrict: "E", //define directive type like E = element,

//A = attribute, C = class, M = comment

scope: { //create a new child scope or an isolate scope

title: '@' //@ reads the attribute value,

//= provides two-way binding,

//& works with functions

},

template: "<div> {{ myName }} </div>", // define HTML markup

templateUrl: 'mytemplate.html', //path to the template, used

// by the directive

replace: true | false, //replace original markup

// with template yes/no

transclude: true | false, // copy original HTML content yes/no

controller: function (scope) {

//define controller, associated with the directive template

//TODO:

},

link: function (scope, element, attrs, controller) {

//define function, used for DOM manipulation

//TODO:

}

}

});

## What are different 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 --> |
|  |  |

## 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 an attribute, class, element or comment.

There are four valid options for restrict:

* 'A' (Attribute)- <span my-directive></span>
* 'C' (Class)- <span class="my-directive:expression;"></span> 'E' (Element)- <my-directive></my-directive>
* 'E' (Element)- <my-directive></my-directive>
* 'M' (Comment)- <!-- directive: my-directive expression -->

## Can you define multiple restrict options on a directive?

You can also specify multiple restrict options to support more than one methods of directive invocation as an element or an attribute. Make sure all are specified in the restrict keyword as:

restrict: 'EA'

## What is auto bootstrap process in AngularJS? OR How AngularJS is initialized automatically?

Angular initializes automatically upon **DOMContentLoaded** event or when the angular.js script isdownloaded to the browser and the *document.readyState* is set to **complete**. At this point AngularJS looks for the ng-app directive which is the root of angular app compilation and tells about AngularJS part within DOM. When the ng-app directive is found then Angular will:

* Load the module associated with the directive.
* Create the application injector.
* Compile the DOM starting from the ng-app root element.

This process is called auto-bootstrapping.

<html>

<body ng-app="myApp">

<div ng-controller="Ctrl">

Hello {{msg}}!

</div>

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

<script>

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

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

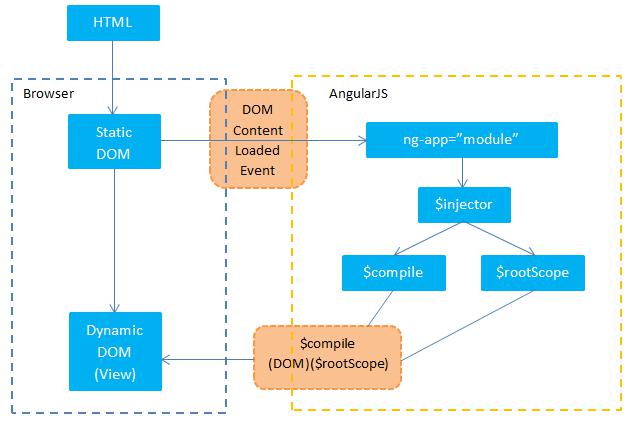
$scope.msg = 'World';

});

</script>

</body>

</html>



## What is manual bootstrap process in AngularJS? OR How AngularJS is initialized manually?

You can manually initialized your angular app by using angular.bootstrap() function. This function takes the modules as parameters and should be called within angular.element(document).ready() function. The angular.element(document).ready() function is fired when the DOM is ready for manipulation.

<html>

<body>

<div ng-controller="Ctrl">

Hello {{msg}}!

</div>

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

<script>

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

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

$scope.msg = 'World';

});

//manual bootstrap process

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

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

});

</script>

</body>

</html>

**Note**

* You should not use the ng-app directive when manually bootstrapping your app.
* You should not mix up the automatic and manual way of bootstrapping your app.
* Define modules, controller, services etc. before manually bootstrapping your app as defined in above example.

## How to bootstrap your angular app for multiple modules?

AngularJS is automatically initialized for one module. But sometimes, it is required to bootstrap for multiple modules and it can be achieved by using two methods:

1. Automatic bootstrap (by combining multiple modules into one module) - You can combine multiple modules into single modules and your angular app will be automatically initialized for newly created module and other modules will act as dependent modules for newly created ones.

For example, suppose you have two modules, module1 and module2, and you have to automatically initialize your app based on these two modules then you achieve this in following way.

<html>

<head>

<title>Multiple modules bootstrap</title>

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

<script>

//module1

var app1 = angular.module("module1", []); app1.controller("Controller1", function ($scope) {

$scope.name = "Shailendra Chauhan";

});

//module2

var app2 = angular.module("module2", []); app2.controller("Controller2", function ($scope) {

$scope.name = "Deepak Chauhan";

});

//module3 dependent on module1 & module2

angular.module("app", ["module1", "module2"]);

</script>

</head>

<body>

<!--angularjs auto bootstrap process-->

<div ng-app="app">

<h1>Multiple modules bootstrap</h1>

<div ng-controller="Controller2">

{{name}}

</div>

<div ng-controller="Controller1">

{{name}}

</div>

</div>

</body>

</html>

1. Manual bootstrap – You can manually bootstrap your app by using angular.bootstrap() function, for multiple modules. The above example can be rewritten as for manual bootstrap process as given below:

<html>

<head>

<title>Multiple modules bootstrap</title>

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

<script>

//module1

var app1 = angular.module("module1", []);

app1.controller("Controller1", function ($scope) {

$scope.name = "Shailendra Chauhan";

});

//module2

var app2 = angular.module("module2", []);

app2.controller("Controller2", function ($scope) {

$scope.name = "Deepak Chauhan";

});

//manual bootstrap process

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

var div1 = document.getElementById('div1');

var div2 = document.getElementById('div2');

//bootstrap div1 for module1 and module2

angular.bootstrap(div1, ['module1', 'module2']);

//bootstrap div2 only for module1

angular.bootstrap(div2, ['module1']);

});

</script>

</head>

<body>

<!--angularjs manual bootstrap process-->

<div id="div1">

<h1>Multiple modules bootstrap</h1>

<div ng-controller="Controller1">

{{name}}

</div>

<div ng-controller="Controller2">

{{name}}

</div>

</div>

<div id="div2">

<div ng-controller="Controller1">

{{name}}

</div>

</div>

</body>

</html>

## What is scope in AngularJS?

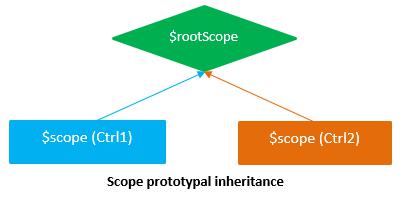
Scope is a JavaScript object that refers to the application model. It acts as a context for evaluating angularexpressions. Basically, it acts as glue between controller and view.



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

## What is $scope and $rootScope?

$scope - A $scope is a JavaScript object which is used for communication between controller and view. Basically, $scope binds a view (DOM element) to the model and functions defined in a controller.



**$rootScope** - The $rootScope is the top-most scope. An app can have only one $rootScope which will be sharedamong all the components of an app. Hence it acts like a global variable. All other $scopes are children of the $rootScope.

**For example**, suppose you have two controllers: Ctrl1 and Ctrl2 as given below:

<!doctype html>

<html>

<body ng-app="myApp">

<div ng-controller="Ctrl1" style="border:2px solid blue; padding:5px">

Hello {{msg}}!

<br />

Hello {{name}}! (rootScope)

</div>

<br />

<div ng-controller="Ctrl2" style="border:2px solid green; padding:5px">

Hello {{msg}}!

<br />

Hey {{myName}}! <br />

Hi {{name}}! (rootScope)

</div>

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

<script>

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;

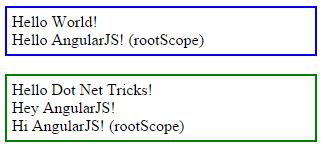
});

</script>

</body>

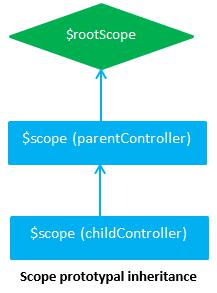
</html>

**Output**



## 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 $rootScope can has one or more child scopes. Each controller has its own $scope (which is a child of the $rootScope), so whatever variables you create on $scope within controller, these variables are accessible by the view based on this controller.



**For example**, suppose you have two controllers: ParentController and ChildController as given below:

<html>

<head>

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

<script>

var app = angular.module('ScopeChain', []); app.controller("parentController", function ($scope) {

$scope.managerName = 'Shailendra Chauhan';

$scope.$parent.companyName = 'Dot Net Tricks'; //attached to

//$rootScope

});

app.controller("childController", function ($scope, $controller) {

$scope.teamLeadName = 'Deepak Chauhan';

});

</script>

</head>

<body ng-app="ScopeChain">

<div ng-controller="parentController ">

<table style="border:2px solid #e37112">

<caption>Parent Controller</caption>

<tr>

<td>Manager Name</td>

<td>{{managerName}}</td>

</tr>

<tr>

<td>Company Name</td>

<td>{{companyName}}</td>

</tr>

<tr>

<td>

<table ng-controller="childController"

style="border:2px solid #428bca">

<caption>Child Controller</caption>

<tr>

<td>Team Lead Name</td>

<td>{{ teamLeadName }}</td>

</tr>

<tr>

<td>Reporting To</td>

<td>{{managerName}}</td>

</tr>

<tr>

<td>Company Name</td>

<td>{{companyName}}</td>

</tr>

</table>

</td>

</tr>

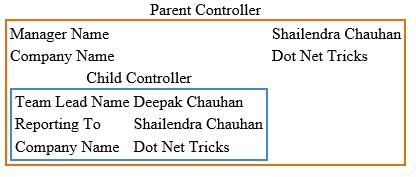
</table>

</div>

</body>

</html>

**Output**



## What is the difference between $scope and scope?

The module factory methods*like controller, directive, factory, filter, service, animation*,*config and run*receive arguments through dependency injection (DI). In case of DI, you inject the **scope object** with the dollar prefix i.e. **$scope**. The reason is the **injected arguments** must match to the names of **injectable objects** followed by dollar ($) prefix.

**For example**, you can inject the scope and element objects into a controller as given below:

module.controller('MyController', function ($scope, $element) {

// injected arguments

});

then the methods like directive linker function don’t receive arguments through dependency injection, you just pass the scope object without using dollar prefix i.e. scope. The reason is the passing arguments are received by its caller.

module.directive('myDirective', function () // injected arguments here

{

return {

// linker function does not use dependency injection link:

function (scope, el, attrs) {

// the calling function will passes the three arguments to the

//linker: scope, element and attributes, in the same order

}

};

});

In the case of non-dependency injected arguments, you can give the name of injected objects as you wish. The above code can be re-written as:

module.directive("myDirective", function () {

return {

link: function (s, e, a) {

//s == scope //e == element //a == attributes

}

};

});

In short, in case of DI the **scope object** is received as **$scope** while in case of non-DI **scope object** is received as **scope** or with any name.

## How AngularJS is compiled?

Angular's HTML compiler allows you to teach the browser new HTML syntax. The compiler allows you to attach new behaviors or attributes to any HTML element. Angular calls these behaviors as directives.

AngularJS compilation process takes place in the web browser; no server side or pre-compilation step is involved. Angular uses $compiler service to compile your angular HTML page. The angular' compilation process begins after your HTML page (static DOM) is fully loaded. It happens in two phases:

* Compile - It traverse the DOM and collect all of the directives. The result is a linking function.
* Link - It combines the directives with a scope and produces a live view. Any changes in the scope model are reflected in the view, and any user interactions with the view are reflected in the scope model.

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.

## How AngularJS compilation is different from other JavaScript frameworks?

If you have worked on templates in other java script framework/library like backbone and jQuery, they process the template as a string and result as a string. You have to dumped this result string into the DOM where you wanted it with innerHTML()

AngularJS process the template in another way. It directly works on HTML DOM rather than strings and manipulates it as required. It uses two way data-binding between model and view to sync your data.

## How Directives are compiled?

It is important to note that Angular operates on DOM nodes rather than strings. Usually, you don't notice this because when an html page loads, the web browser parses HTML into the DOM automatically.

HTML compilation happens in three phases:

The $compile traverses the DOM and looks for directives. 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. Then, each directive’s own compile function is executed, giving each directive the chance to modify the DOM itself.
* Each compile function returns a linking function, which is then composed into a combined linking function and returned.
* $compile links the template with the scope by calling the combined linking function from the previous step. This in turn will call the linking function of the individual directives, registering listeners on the elements and setting up $watch with the scope as each directive is configured to do.

The pseudo code for the above process is given below:

var $compile = ...; // injected into your code var scope = ...;

var parent = ...; // DOM element where the compiled template can be appended

var html = '<div ng-bind="exp"></div>';

//Step 1: parse HTML into DOM element

var template = angular.element(html);

//Step 2: compile the template

var linkFn = $compile(template);

//Step 3: link the compiled template with the scope.

var element = linkFn(scope);

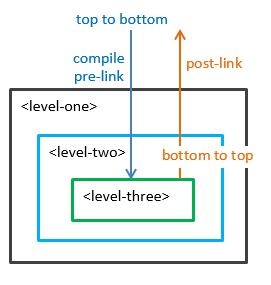
//Step 4: Append to DOM (optional)

parent.appendChild(element);

## What are Compile, Pre, and Post linking in AngularJS?

**Compile –** This compiles an HTML string or DOM into a template and produces a template function, which can then be used to link scope and the template together.

Use the compile function to change the original DOM (template element) before AngularJS creates an instance of it and before a scope is created.



**Post-Link** –This is executed after the child elements are linked. It is safe to do DOM transformation in the post-linking function.

Use the post-link function to execute logic, knowing that all child elements have been compiled and all pre-link and post-link functions of child elements have been executed.

**Pre-Link** –This is executed before the child elements are linked. Not safe to do DOM transformation since thecompiler linking function will fail to locate the correct elements for linking.

Use the pre-link function to implement logic that runs when AngularJS has already compiled the child elements, but before any of the child element's post-link functions have been called.

<html>

<head>

<title>Compile vs Link</title>

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

<script type="text/javascript">

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

function createDirective(name) {

return function () {

return {

restrict: 'E',

compile: function (tElem, tAttrs) {

console.log(name + ': compile');

return {

pre: function (scope, iElem, attrs) {

console.log(name + ': pre link');

},

post: function (scope, iElem, attrs) {

console.log(name + ': post link');

}

}

}

}

}

}

app.directive('levelOne', createDirective('level-One')); app.directive('levelTwo', createDirective('level-Two')); app.directive('levelThree', createDirective('level-Three'));

</script>

</head>

<body ng-app="app">

<level-one>

<level-two>

<level-three>

Hello {{name}}

</level-three>

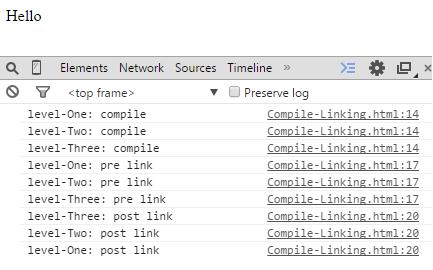
</level-two>

</level-one>

</body>

</html>

**Output**



## What directives are used to show and hide HTML elements in AngularJS?

**ng-show and ng-hide** directives are used to show and hide HTML elements in the AngularJS based on an expression. When the expression is true for ng-show or ng-hide, HTML element(s) are shown or hidden from the page. When the expression is false for ng-show or ng-hide, HTML element(s) are hidden or shown on the page.

<div ng-controller="MyCtrl">

<div ng-show="data.isShow">

ng-show Visible

</div>

<div ng-hide="data.isHide">

ng-hide Invisible

</div>

</div>

<script>

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

app.controller("MyCtrl", function ($scope) {

$scope.data = {};

$scope.data.isShow = true;

$scope.data.isHide = true;

});

</script>

## Explain directives ng-if, ng-switch and ng-repeat?

**ng-if –** This directive can add / remove HTML elements from the DOM based on an expression. If theexpression is true, it add HTML elements to DOM, otherwise HTML elements are removed from the DOM.

<div ng-controller="MyCtrl">

<div ng-if="data.isVisible">

ng-if Visible

</div>

</div>

<script>

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

app.controller("MyCtrl", function ($scope) {

$scope.data = {};

$scope.data.isVisible = true;

});

/script>

**ng-switch –** This directive can add / remove HTML elements from the DOM conditionally based on scopeexpression.

<div ng-controller="MyCtrl">

<div ng-switch on="data.case">

<div ng-switch-when="1">

Shown when case is 1

</div>

<div ng-switch-when="2">

Shown when case is 2

</div>

<div ng-switch-default>

Shown when case is anything else than 1 and 2

</div>

</div>

</div>

<script>

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

app.controller("MyCtrl", function ($scope) {

$scope.data = {};

$scope.data.case = true;

});

</script>

**ng-repeat** - This directive is used to iterate over a collection of items and generate HTML from it.

<div ng-controller="MyCtrl">

<ul>

<li ng-repeat="name in names">

{{ name }}

</li>

</ul>

</div>

<script>

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

app.controller("MyCtrl", function ($scope) {

$scope.names = ['Shailendra', 'Deepak', 'Kamal'];

});

</script>

## 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:

* $index
* $first
* $middle
* $last

<html>

<head>

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

<script>

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

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

$scope.friends = [{ name: 'shailendra', gender: 'boy' },

{ name: 'kiran', gender: 'girl' },

{ name: 'deepak', gender: 'boy' },

{ name: 'pooja', gender: 'girl' } ];

});

</script>

</head>

<body ng-app="app">

<div ng-controller="ctrl">

<ul class="example-animate-container">

<li ng-repeat="friend in friends">

<div>

[{{$index + 1}}] {{friend.name}} is a {{friend.gender}}.

<span ng-if="$first">

<strong>(first element found)</strong>

</span>

<span ng-if="$middle">

<strong>(middle element found)</strong>

</span>

<span ng-if="$last">

<strong>(last element found)</strong>

</span>

</div>

</li>

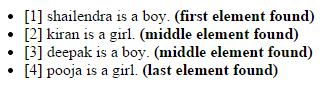
</ul>

</div>

</body>

</html>

**Output**



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.

## What are Templates in AngularJS?

AngularJS templates are just plain old HTML that contains Angular-specific elements and attributes.AngularJS used these templates to show information from the model and controller.

**Creating an AngularJS template**

<html ng-app>

<!-- body tag with ngController directive -->

<body ng-controller="MyController">

<input ng-model="txtName" value="shailendra"/>

<!-- button tag with ng-click directive & string expression 'btnText' wrapped in "{{ }}" markup -->

<button ng-click="changeName()">

{{btnText}}

</button>

<script src="angular.js">

</body>

</html>

## What is ng-include and when to use it?

**ng-include** is a directive which is used to include external HTML fragments from other files into the view'sHTML template.

**For example**,*index.html*file can be added inside the*div*element by using ng-include directive as an attribute.

<div ng-controller="MyCtrl">

<div ng-include="'index.html'"></div> </div>

ng-include directive is limited to load HTML fragments file from same domain but it doesn’t work for cross-domain i.e. it can’t load the HTML fragments file from different domains.

## What angular components can be defined within AngularJS templates?

AngularJS templates can have following angular elements and attributes:

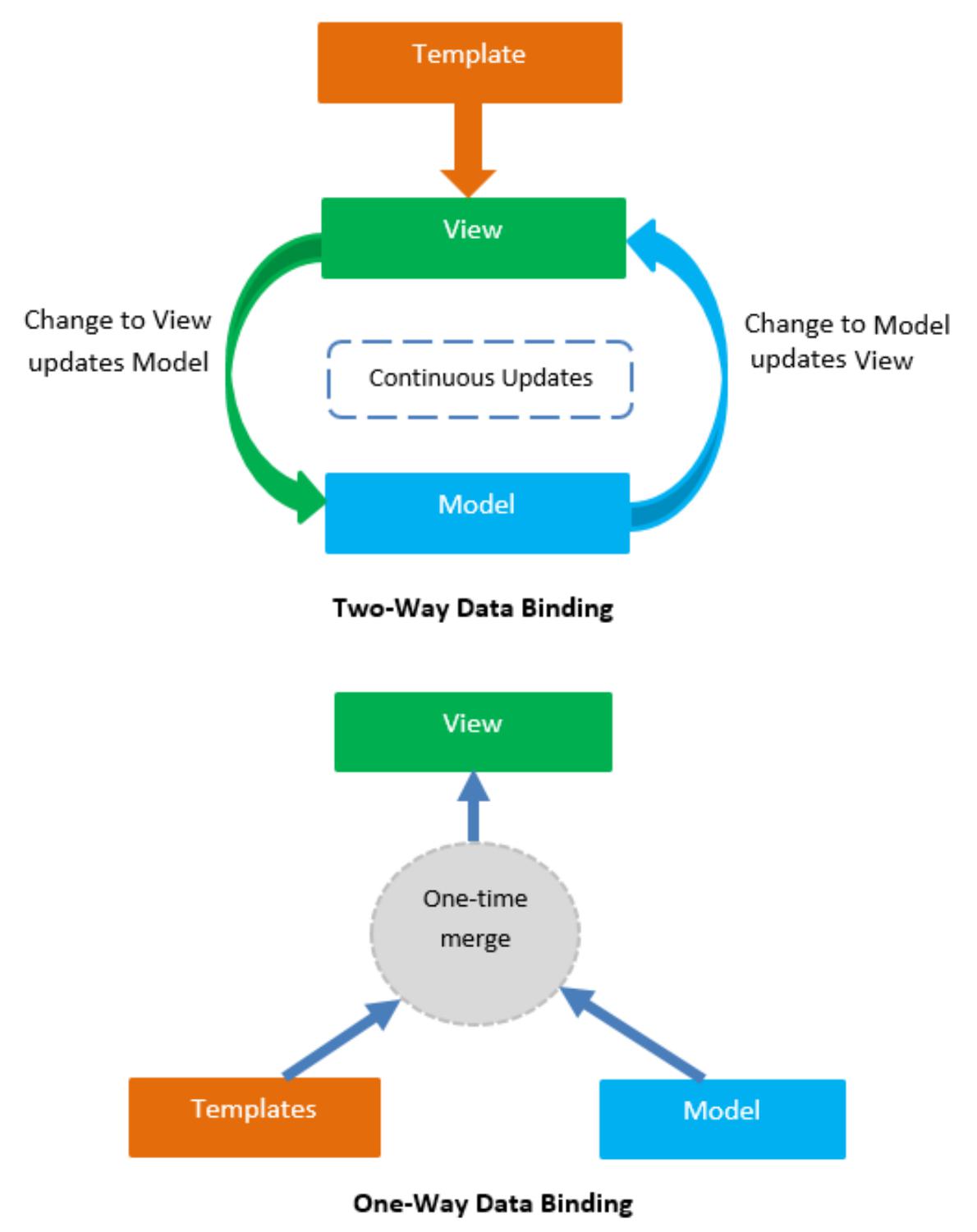
* Directive
* Angular Markup ('{{}}')
* Filters
* Form Controls

## What is data binding in AngularJS?

AngularJS data-binding is the most useful feature which saves you from writing boilerplate code (i.e. the sections of code which is included in many places with little or no alteration). Now, developers are not responsible for manually manipulating the DOM elements and attributes to reflect model changes. AngularJS provides two-way data-binding to handle the synchronization of data between model and view.

## Explain Two-way and One-way data binding in AngularJS?

Two-way data binding - It is used to synchronize the data between model and view. It means, any change in model will update the view and vice versa. ng-model directive is used for two-way data binding.



**One-way data binding -** This binding is introduced in **Angular 1.3**. An expression that starts with double colon (:**:**),is considered a one-time expression i.e. one-way binding.

**Two-Way and One-Way data binding Example**

<div ng-controller="MyCtrl">

<label>Name (two-way binding):

<input type="text" ng-model="name" />

</label>

<strong>Your name (one-way binding):</strong> {{::name}}<br />

<strong>Your name (normal binding):</strong> {{name}}

</div>

<script>

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

app.controller("MyCtrl", function ($scope) {

$scope.name = "Shailendra Chauhan"

})

</script>

## What is issue with two-way data binding? OR Why one-way data binding is introduced?

In order to make data-binding possible, 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.

This automatic change happens because of $digest cycle is triggered. Hence, Angular processes all registered *watchers* on the current scope and its children and checks for model changes and calls dedicated *watch listeners* until the model is stabilized and no more listeners are fired. Once the $digest loop finishes the execution, the browser re-renders the DOM and reflects the changes.

By default, every variable on a scope is observed by the angular. In this way, unnecessary variable are also observed by the angular that is time consuming and as a result page is becoming slow.

**Hence** to avoid **unnecessary observing** of variables on scope object, angular introduced one-way data binding.

## How AngularJS handle data binding?

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.

## What is the difference between $watch, $digest and $apply?

$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.

$watch(watchExpression, listener, [objectEquality])

<html>

<head>

<title>AngularJS Watch</title>

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

<script>

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

var myController = myapp.controller("myController", function ($scope) {

$scope.name = 'dotnet-tricks.com';

$scope.counter = 0;

//watching change in name value

$scope.$watch('name', function (newValue, oldValue) {

$scope.counter = $scope.counter + 1;

});

});

</script>

</head>

<body ng-app="myapp" ng-controller="myController">

<input type="text" ng-model="name" />

<br /><br />

Counter: {{counter}}

</body>

</html>

**$digest()** - This function iterates through all the watches in the $scope object, and its child $scope objects (if it hasany). 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.

$digest()

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.

<html>

<head>

<script src="lib/jquery-1.11.1.js"></script>

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

</head>

<body ng-app="app">

<div ng-controller="Ctrl">

<button class="digest">Digest my scope!</button>

<br />

<h2>obj value : {{obj.value}}</h2>

</div>

<script>

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

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

$scope.obj = { value: 1 };

$('.digest').click(function () {

console.log("digest clicked!");

console.log($scope.obj.value++);

//update value

$scope.$digest();

});

});

</script>

</body>

</html>

**$apply()** - Angular do auto-magically updates only those model changes which are inside AngularJS context. Whenyou 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.

$apply([exp])

**Example**

<html>

<head>

<title>AngularJS Apply</title>

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

<script>

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

var myController = myapp.controller("myController", function ($scope) {

$scope.datetime = new Date();

$scope.updateTime = function () {

$scope.datetime = new Date();

}

//outside angular context document.getElementById("updateTimeButton")

.addEventListener('click',function () {

//update the value

$scope.$apply(function () {

console.log("update time clicked");

$scope.datetime = new Date();

console.log($scope.datetime);

});

});

});

</script>

</head>

<body ng-app="myapp" ng-controller="myController">

<button ng-click="updateTime()">Update time - ng-click</button>

<button id="updateTimeButton">Update time</button>

<br />

{{datetime | date:'yyyy-MM-dd HH:mm:ss'}}

</body>

</html>

## Which one is fast between $digest and $apply?

$digest() is faster than $apply(), since $apply() triggers watchers on the entire scope chain i.e. on thecurrent scope and its parents or children (if it has) while $digest() triggers watchers on the current scope and its children(if it has).

## Which one handles exception automatically between $digest and $apply?

When error occurs in one of the watchers, $digest() cannot 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.

**Pseudo-Code of $apply()**

function $apply(expr) {

try {

return $eval(expr);

}

catch (e) {

$exceptionHandler(e);

}

finally {

$root.$digest();

}

}

## Explain $watch(), $watchgroup() and $watchCollection() functions of 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.

$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.

scope.name = 'shailendra'; scope.counter = 0;

scope.$watch('name', function (newVal, oldVal) {

scope.counter = scope.counter + 1;

});

**$watchgroup -** This function is introduced in **Angular1.3**. This works the same as $watch() function except thatthe first parameter is an array of expressions to watch.

$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.

$scope.teamScore = 0;

$scope.time = 0;

$scope.$watchGroup(['teamScore', 'time'], function(newVal, oldVal) {

if(newVal[0] > 20)

{

$scope.matchStatus = 'win';

}

else if (newVal[1] > 60)

{

$scope.matchStatus = 'times up';

}

});

**$watchCollection -** This function is used to watch the properties of an object and fires whenever any of theproperties change. It takes an object as the first parameter and watches the properties of the object.

$watchCollection(obj, listener)

The listener is called whenever anything within the obj has been changed.

$scope.names = ['shailendra', 'deepak', 'mohit', 'kapil']; $scope.dataCount = 4;

$scope.$watchCollection('names', function (newVal, oldVal) {

$scope.dataCount = newVal.length;

});

## Explain AngularJS scope life-cycle?

Scope data goes through a life cycle when the angular app is loaded into the browser. Understanding thescope life cycle will help you to understand the interaction between scope and other AngularJS components.

The scope data goes through the following life cycle phases:

* **Creation** –This phase initialized the scope. The root scope is created by the $injector when the applicationis bootstrapped. During template linking, some directives create new child scopes.
* A digest loop is also created in this phase that interacts with the browser event loop. This digest loop is responsible to update DOM elements with the changes made to the model as well as executing any registered watcher functions.
* **Watcher registration -** This phase registers watches for values on the scope that are represented in the template. These watches propagate model changes automatically to the DOM elements.
* You can also register your own watch functions on a scope value by using the *$watch()* function.
* **Model mutation -** This phase occurs when data in the scope changes. When you make the changes in your angular app code, the scope function *$apply()* updates the model and calls the $digest() function to update the DOM elements and registered watches.
* When you do the changes to scope inside your angular code like within controllers or services, angular internally call *$apply()* function for you. But when you do the changes to the scope outside the angular code, you have to call *$apply()* function explicitly on the scope to force the model and DOM to be updated correctly.
* **Mutation observation –** This phase occurs when the $digest() function is executed by the digest loop atthe end of *$apply()* call. When $digest() function executes, it evaluates all watches for model changes. If a value has been changed, $digest() calls the $watch listener and updates the DOM elements.
* **Scope destruction –** This phase occurs when child scopes are no longer needed and these scopes areremoved from the browser’s memory by using $destroy() function. It is the responsibility of the child scope creator to destroy them via scope.$destroy() API.

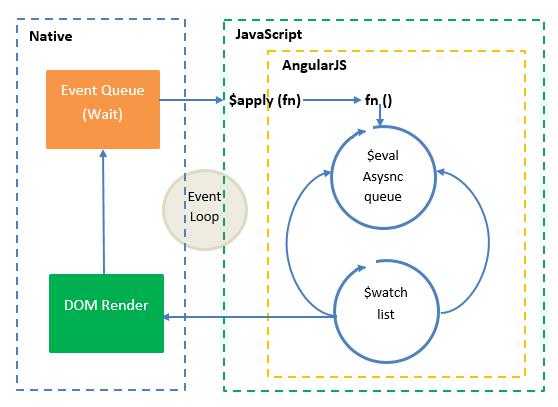
This stop propagation of $digest calls into the child scopes and allow the memory to be reclaimed by the browser garbage collector.

## Explain digest life-cycle in AngularJS?

The digest loop is responsible to update DOM elements with the changes made to the model as well asexecuting any registered watcher functions.

**The $digest loop** is fired when the browser receives an event that can be managed by the angular context. This loop is made up of two smaller loops. One processes the $evalAsync queue and the other one processes the $watch list.

**The $digest loop** keeps iterating until the $evalAsync queue is empty and the $watch list does not detect anychanges in the model.



**The $evalAsync queue** contains those tasks which are scheduled by*$evalAsync()*function from a directive orcontroller.

The **$watch list** contains watches correspondence to each DOM element which is bound to the $scope object. These watches are resolved in the $digest loop through a process called dirty checking. The dirty checking is a process which checks whether a value has changed, if the value has changed, it set the $scope as dirty. If the $scope is dirty, another $digest loop is triggered.

When you do the changes to the scope outside the angular context, you have to call *$apply()* function explicitly on the scope to trigger $digest cycle immediately.

## When to use $destroy() function of scope?

**$destroy() -** This function permanently detached the current scope with all of its children from the parent scope. This is required when child scopes are no longer needed. Hence, $destroy() function is called to remove these scopes from the browser’s memory.

When **$destroy()** is called all the watchers and listeners get removed and the object which represented the scope becomes eligible for garbage collection.

## What is difference between $evalAsync and $timeout?

$evalAsync - This executes the expression on the current scope on later time. The $evalAsync makes no guarantees as to when the expression will be executed, only that:

1. If code is queued using $evalAsync from a directive, it will run after the DOM has been manipulated by Angular, but before the browser renders.

2. If code is queued using $evalAsync from a controller, it will run before the DOM has been manipulated by Angular and before the browser renders.

$timeout - This also executes the expression on the current scope on later time. When the code is queued using $timeout, it will run after the DOM has been manipulated by Angular and after the browser renders which may cause flicker in some cases.

## What is the difference between $watch and $observe?

**$watch is a method on the scope object** which is used to watch expressions. The expression can be either strings or functions. It can be called wherever you have access to scope (a controller or a directive linking function).

**$observe is a method on the attrs** object which is only used to observe the value change of a DOM attribute. It is only used inside directives.

Note - All $observes and $watches are checked on every digest cycle.

## What is the difference between $parse and $eval?

$parse and $eval both operate on angular expressions i.e. {{ expression }}.

***$eval is a scope method*** ***which executes an expression*** on the current scope and returns the result.

scope.a = 1; scope.b = 2;

scope.$eval('a+b'); // 3

***$parse is an Angular service which converts an expression into a function***. Then function can be invoked and passed a context (usually scope) in order to retrieve the expression's value.

<div my-attr="obj.name" my-directive>

dotnet-tricks.com

</div>

<script type="text/javascript">

myApp.directive('myDirective', function( $parse, $log ) {

return function( scope, el, attrs ) {

//parse the "my-attr" attribute value into a function

var myFn = $parse( attrs.myAttr );

//"myFn" is now a function which can be invoked to get the expressions

//value;

//the following line logs the value of obj.name on scope:

$log.log(myFn(scope) ); // dotnet-tricks.com

el.bind('click', function() {

//"myFn.assign" is also a function; it can be invoked to

//update the expresssion value

myFn.assign(scope, "New name");

scope.$apply();

})

}

});

</script>

Also, if the expression is assignable then the returned function will have an assign property. The assign property is a function that can be used to change the expression's value on the given context.

## What is Isolate Scope and why it is required?

By default, directives have access to the parent scope in AngularJS apps. Hence, you can write your custom directive code based on parent scope. If the parent scope changes at all the directive is no longer useful.

angular.module('mydirective').directive('sharedScope', function () {

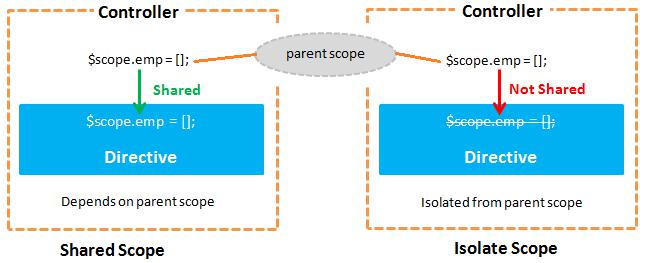
return {

template: 'Name: {{emp.name}} Address: {{emp.address}}'

};

});

The shared scope allows the parent scope to flow down into the directive but the Isolate scope doesn’t allow the parent scope to flow down into the directive.



**Creating Isolate Scope**

You can isolate the scope in a directive by adding a scope property into the directive as given below:

angular.module('mydirective').directive('sharedScope', function () {

return {

scope: {},

template: 'Name: {{emp.name}} Address: {{emp.address}}'

};

});

## Does AngularJS support MVC?

AngularJS is a MVC framework. It does not implement MVC in the traditional way, but rather something closer to MVVM Model-View-ViewModel).

## What is Model in AngularJS?

Models are plain old JavaScript objects that represent data used by your app. Models are also used to represent your app's current state.

## What is ViewModel in AngularJS?

A viewmodel is an object that provides specific data and methods to maintain specific views. Basically, it is a $scope object which lives within your AngularJS app's controller. A viewmodel is associated with a HTML element with the ng-model and ng-bind directives.

## What is Controller in AngularJS?

The controller defines the actual behavior of your app. It contains business logic for the view and connects the model to view with the help of $scope. A controller is associated with a HTML element with the ng-controller directive.

**Creating Controller**

<script type="text/javascript"> //defining main controllerapp.controller('mainController', function ($scope) {

//defining book viewmodel

$scope.book =

{

id: 1,

name: 'AngularJS Interview Questions and Answers',

author: 'Shailendra Chauhan',

};

});

</script>

**Using Controller**

<div ng-controller="mainController">

Id: <span ng-bind="book.id"></span> <br />

Name:<input type="text" ng-model="book.name" /> <br />

Author: <input type="text" ng-model="book.author" />

</div>

## How to share information between controllers in AngularJS? OR What are the ways to communicate between controllers in AngularJS?

***There are various different ways to share data between controllers in an AngularJS app. The most commonly used are Scope, Service, Factory and Providers.***

## What is $emit, $broadcast and $on in AngularJS?

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.

<!DOCTYPE html>

<html>

<head>

<title>Broadcasting</title>

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

<script>

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

app.controller("firstCtrl", function ($scope) {

$scope.$on('eventName', function (event, args) {

$scope.message = args.message;

console.log($scope.message);

});

});

app.controller("secondCtrl", function ($scope) {

$scope.handleClick = function (msg) {

$scope.$emit('eventName', { message: msg });

};

});

</script>

</head>

<body ng-app="app">

<div ng -controller="firstCtrl"

style="border:2px solid #E75D5C; padding:5px;">

<h1 >Parent Controller</h1>

<p>Emit Message : {{message}}</p> <br />

<div ng-controller="secondCtrl"

style="border:2px solid #428bca;padding:5px;">

<h1>Child Controller</h1>

<input ng-model="msg">

<button ng-click="handleClick(msg);">Emit</button>

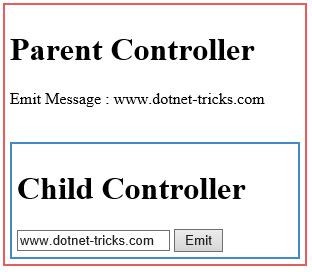
</div>

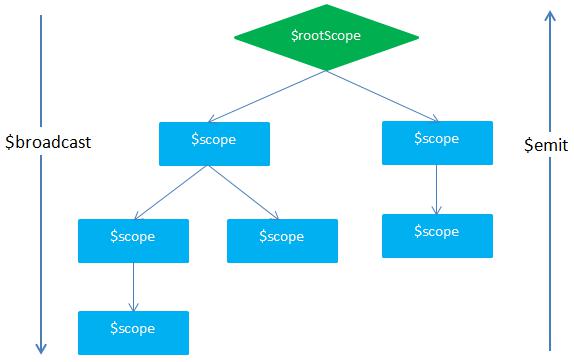
</div>

</body>

</html>

**Output**





$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.

<!DOCTYPE html>

<html>

<head> <title>Broadcasting</title>

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

<script>

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

app.controller("firstCtrl", function ($scope) {

$scope.handleClick = function (msg) {

$scope.$broadcast('eventName', { message: msg });

};

});

app.controller("secondCtrl", function ($scope) {

$scope.$on('eventName', function (event, args) {

$scope.message = args.message;

console.log($scope.message);

});

});

</script>

</head>

<body ng-app="app">

<div ng-controller="firstCtrl"

style="border:2px solid #E75D5C; padding:5px;">

<h1>Parent Controller</h1>

<input ng-model="msg">

<button ng-click="handleClick(msg);">Broadcast</button> <br /><br />

<div ng-controller="secondCtrl"

style="border:2px solid #428bca;padding:5px;">

<h1>Child Controller</h1>

<p>Broadcast Message : {{message}}</p>

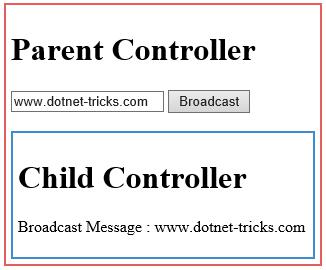
</div>

</div>

</body>

</html>

**Output**



**$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 andbroadcast 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.

## What is View in AngularJS?

The view is responsible for presenting your models data to end user. Typically it is the HTML markup which exists after AngularJS has parsed and compiled the HTML to include rendered markup and bindings.

## How to apply validation in AngularJS?

AngularJS provides you built-in validation directives to validate form client side. This makes your life prettyeasy to handle client-side form validations without adding a lot of extra effort. AngularJS form validations are based on the HTML5 form validators.

**AngularJS directives for form validation**

Here is a list of AngularJS directive which can be applied on an input field to validate its value.

<input type="text" ng-model="{ string }" [name="{ string }"]

[ng-required="{ boolean }"] [ng-minlength="{ number }"] [ng-maxlength="{ number }"] [ng-pattern="{ string }"] [ng-change="{ string }"]>

</input>

## How to do custom form validation in AngularJS?

AngularJS allows you to create your own custom validation directives. For example, you have to compare password and confirm password fields. To achieve this task, you have to make a custom directive that will be fired whenever the password or confirm password changes.

**Creating custom directive to compare password and confirm password fields**

//defining module

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

//creating custom directive

myapp.directive('ngCompare', function () {

return {

require: 'ngModel',

link: function (scope, currentEl, attrs, ctrl) {

//getting first element

var comparefield= document.getElementsByName(attrs.ngCompare)[0];

compareEl = angular.element(comparefield);

//current field key up

currentEl.on('keyup', function () {

if (compareEl.val() != "") {

var isMatch = currentEl.val() === compareEl.val(); ctrl.$setValidity('compare', isMatch); scope.$digest();

}

});

//Element to compare field key up

compareEl.on('keyup', function () {

if (currentEl.val() != "") {

var isMatch = currentEl.val() === compareEl.val(); ctrl.$setValidity('compare', isMatch); scope.$digest();

}

});

}

}

});

**Using above created custom directive**

<form name="userForm" ng-submit="submitForm()" novalidate>

<!-- Password -->

<div>

<label>Password</label>

<input type="Password" name="password" ng-required="true"

ng-model="user.password">

<p ng-show="userForm.password.$invalid">

Your password is required.

</p>

</div>

<!-- Confirm Password -->

<div>

<label>Confirm Password</label>

<input type="Password" name="confirmPassword" ng-compare="password"

ng-required="true" ng-model="user.confirmPassword" >

<p ng -show="userForm.confirmPassword.$error.compare &&

!userForm.confirmPassword.$error.required">

Confirm password doesn’t match.

</p>

</div>

<!-- Other code has been removed for clarity-->

</form>

Bottom of Form

## What are different Angular form properties?

Angular provides properties on form which help you to get information about a form or its inputs and tovalidate them.

**$valid -** It is a boolean property that tells whether the form or it's inputs are valid or not. If all containing form andcontrols are valid, then it will be true, otherwise it will be false.

**Syntax:** formName.$valid formName.inputFieldName.$valid

**$invalid -** It is a boolean property that tells whether the form or it's inputs are invalid or not. If at least onecontaining form and control is invalid then it will be true, otherwise it will be false.

**Syntax:** formName.$invalid formName.inputFieldName.$invalid

**$pristine -** It is a boolean property that tells whether the form or it's inputs are unmodified by the user or not. Ifthe form or its inputs are unmodified by the user, then it will be true, otherwise it will be false.

**Syntax:** formName.inputFieldName.$pristine

**$dirty -** It is a boolean property that is actually just reverse of pristine i.e. it tells whether the form or it's inputsare modified by the user or not. If the form or its inputs are modified by the user, then it will be true, otherwise it will be false.

**Syntax:** formName.$dirty formName.inputFieldName.$dirty

**$error -** This is an object hash which contains references to all invalid controls or forms. It has all errors as keys:where keys are validation tokens (such as required, url or email) and values are arrays of controls or forms that are invalid with given error. For a control, if a validation fails then it will be true, otherwise it will be false.

**Syntax:** formName.$error formName.inputFieldName.$error

## What are different states of a form in AngularJS?

The AngularJS form goes to the following states, starting from the form rendering and when the user hasfinished the filling of form.

**State 1: pristine and invalid -** When the form is first time rendered and the user has not interacted with the formyet.

**State 2: dirty and invalid -** User has interacted with the form, but form validity has not been satisfied, yet.

**State 3: dirty and valid** - User has finished the filling of form and the entire form validations has been satisfied

## What is Service in AngularJS?

***A service is a reusable singleton object*** 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.

## What are different ways to create service in AngularJS?

There are five ways to create a service as given below

* Service
* Factory
* Provider
* Value
* Constant

## What is the difference between Factory, Service and Provider?

**Factory** - A factory is a simple function which allows you to add some logic before creating the object. It returns the created object

**Example**

//define a factory using factory() function

app.factory('MyFactory', function () {

var serviceObj = {};

serviceObj.function1 = function () {

//TO DO:

};

serviceObj.function2 = function () { //TO DO:

};

return serviceObj;

});

**When to use:** It is just a collection of functions like a class. Hence, it can be instantiated in different controllerswhen you are using it with constructor function.

**Service -** A service is a constructor function which creates the object using new keyword. You can add propertiesand functions to a service object by using *this* keyword. Unlike factory, it doesn’t return anything.

**Example**

//define a service using service()

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

this.function1 = function () {

//TO DO:

};

this.function2 = function () {

//TO DO:

};

});

**When to use:** It is a singleton object. Use it when you need to share a single object across the application.

**For example**, authenticated user details.

**Provider -** A provider is used to create a configurable service object. It returns value by using*$get()*function.

**Example**

//define a provider using provider()

function app.provider('configurable', function () {

var privateName = '';

this.setName = function (newName) {

privateName = newName;

};

this.$get = function () {

return {

name: privateName

};

};

});

//configuring provider using config()

function app.config(function (configurableService) {

configurableService.setName('www.dotnet-tricks.com');

});

**When to use:** When you need to provide module-wise configuration for your service object before making itavailable.

**Example**

<html>

<head>

<title>AngularJS Service, Factory and Providers</title>

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

</head>

<body>

<div class="container" style="padding-top:20px;">

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

<p>From Service: {{serviceName}}</p>

<p>From Factory: {{factoryName}}</p>

<p>From Provider: {{providerName}}</p>

</div>

</div>

<script>

//defining module

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

//defining service

app.service('myService', function () {

this.name = '';

this.setName = function (newName) {

this.name = newName;

return this.name;

};

});

//defining factory

app.factory('myFactory', function () {

var serviceObj = {}; serviceObj.name = '';

serviceObj.setName = function (newName) {

serviceObj.name = newName;

};

return serviceObj;

});

//defining provider

app.provider('configurable', function () {

var privateName = '';

this.setName = function (newName) {

privateName = newName;

};

this.$get = function () {

return {

name: privateName

};

};

});

//configuring provider

app.config(function (configurableProvider) {

configurableProvider.setName("Saksham Chauhan");

});

//defining controller

app.controller('myController', function ($scope, myService, myFactory, configurable) {

$scope.serviceName = myService.setName("Saksham Chauhan");

myFactory.setName("Saksham Chauhan");

$scope.factoryName = myFactory.name;

$scope.providerName = configurable.name;

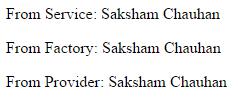
});

</script>

</body>

</html>

**Output**



## What is difference between value and constant?

Value and Constant are simple objects which are used to share data globally with in a module.

**Value -** A value can be a number, string,date-time, array orobject. You can also register a function as a value.Values are typically used as configuration which is injected into factories, services or controllers.

//define module

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

//define value app.value("numberValue", 100);

app.value("stringValue", "dotnet-tricks.com");

app.value("objectValue", { name: "dotnet-tricks.com", totalUsers: 120000 });

**Constant -** A constant is like as value. The difference between a value and a constant service is that constantservice can be injected into a module configuration function i.e. *config()* but value service cannot be.

//define module

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

//define constant

app.constant("mynumberValue", 200);

app.constant("mystringValue", "webgeekschool.com");

//module configuration function

app.config(function (mynumberValue) {

//here value objects can't be injected

console.log("Before:" + mynumberValue);

mynumberValue = "New Angular Constant Service";

console.log("After:" + mynumberValue);

});

## What is the difference between $http and $resource?

$http service is a core Angular service which allows you to make AJAX requests by using GET, HEAD, POST, PUT, DELETE, JSONP and PATCH methods. It is very much like the $.ajax() method in jQuery. It can be used with RESTful and Non-RESTful server-side data sources.

$http is good for quick retrieval of server-side data that doesn’t really need any specific structure or complex behaviors.

$resource warps $http and allows you to interact with RESTful server-side data sources. It requires the

ngResource module to be installed which exist in angular-resource.js

$http is good for retrieval of RESTful server-side data sources that might need any specific structure or complex behaviors.

## What methods $http service support?

The $http service supports the following methods:

* $http.get()
* $http.head()
* $http.post()
* $http.put()
* $http.delete()
* $http.jsonp()
* $http.patch()

## How to enable caching in $http service?

You can enable caching in $http service by setting configuration property*cache*to*true*. When cache isenabled, $http service stores the response from the server in local cache. In this way, next time the response will be served from the cache without sending request to server.

$http.get("http://server/myserviceapi",{ cache:true

}).sucess(function(){ //TO DO:

});

## What methods $resource service object support?

The $resources service method supports the following method

* get()
* query()
* save()
* remove()
* delete()

## What is $q service and when to use it?

**$q** is a service that helps you to run functions asynchronously, and use their return values when they havedone processing.

**$q service** is said to be inspired by*Chris Kowal's Q library*which allow users to monitor asynchronous progress byproviding a "promise" as a return from a call.

It is good when you need to process a number of asynchronous activities simultaneously. The *$q.all()* function lets you trigger several callbacks at the same time, and use a single then function to join them all together.

var first = $http.get("/app/data/first.json"),

second = $http.get("/app/data/second.json"),

third = $http.get("/app/data/third.json");

$q.all([first, second, third])

.then(function (result) {

var tmp = [];

angular.forEach(result, function (response) {

tmp.push(response.data);

});

return tmp; })

.then(function (tmpResult) {

$scope.combinedResult = tmpResult.join(", ");

});

## What is the difference between Kris Kowal's Q and $q?

There are two main differences between Kris Kowal's Q and $q:

1. $q is integrated with the $rootScope.Scope Scope model observation mechanism in angular, which means faster propagation of resolution or rejection into your models and avoiding unnecessary browser repaints, which would result in flickering UI.
2. Q has many more features than $q, but that comes at a cost of bytes. $q is tiny, but contains all the important functionality needed for common async tasks.

## What is Restangular?

Restangular is an Angular service specifically designed simply to fetch data from the rest of the world. Touse Restangular you need to link the *restangular.js* file and include restangular resource as a dependency within your angular app.

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

app.controller('MainController',function ($scope, Restangular) {

//TO DO:

});

## What are the advantages of Restangular over $resource and $http?

Restangular has several features that distinguish it from $resource:

1. It uses promises.
2. You can use this in $routeProvider.resolve.
3. It doesn't have all those $resource bugs like trailing slashes, additional: in the URL, escaping information, expecting only arrays for getting lists, etc.
4. It supports all HTTP methods.
5. It supports ETag out of the box.
6. It supports self-linking elements if you receive from the server some item that has a link to itself, you can use that to query the server instead of writing the URL manually.
7. You don't have to create one $resource object per request.
8. You don't have to write or remember ANY URL like $resource. You just write the name of the resource you want to fetch and that's it.
9. It supports nested RESTful resources.
10. Restangular lets you create your own methods.
11. Support for wrapped responses.

## What is difference between $window and window in AngularJS?

**$window** is an Angular service which reference to the browser's **window** object. The **window** object isglobally available in JavaScript; it causes testability problems, because it is a global variable. Angular refer to it through the **$window** service, so that it can be overridden, removed or mocked for testing.

<script>

var app = angular.module('windowExample', []); app.controller('ExampleController',function ($scope, $window) {

$scope.greeting = 'Hello, World!';

$scope.doGreeting = function (greeting) {

$window.alert(greeting);

};

});

</script>

<div ng-app="windowExample" ng-controller="ExampleController">

<input type="text" ng-model="greeting" />

<button ng-click="doGreeting(greeting)">ALERT</button>

</div>

## What is difference between $document and window.document in AngularJS?

**$document** is an Angular service which reference to the browser's **window.document** object.

<script>

var app = angular.module('documentExample', []); app.controller('ExampleController', ['$scope', '$document', function ($scope,

$document) {

$scope.title = $document[0].title;

$scope.windowTitle = angular.element(window.document)[0].title; }]);

</script>

<div ng-app="documentExample" ng-controller="ExampleController">

<p>$document title: <b ng-bind="title"></b></p>

<p>window.document title: <b ng-bind="windowTitle"></b></p>

</div>

## What is difference between $timeout and window.setTimeout in AngularJS?

$timeout is an Angular service which wraps the browser's window.setTimeout() function into a try/catch block and delegates any exceptions to $exceptionHandler service. It is used to call a JavaScript function after a given time delay. The $timeout service only schedules a single call to the function.

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

app.controller("MyController", function ($scope, $timeout) {

$timeout(callAtTimeout, 1000);

});

function callAtTimeout() {

console.log("Timeout occurred");

}

## What is difference between $interval and window. setInterval in AngularJS?

**$interval** is an Angular service which wraps the browser's **window. setInterval()** function. It is also usedto call a JavaScript function repeatedly within a time interval.

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

app.controller("MyController", function ($scope, $interval) {

$interval(callAtInterval, 3000);

});

function callAtInterval() {

console.log("Interval occurred");

}

## What is Routing in AngularJS?

AngularJS Routing helps you to divide your app into multiple views and bind different views to Controllers.The magic of Routing is taken care by an AngularJS service *$routeProvider*. *$routeProvider* service provides method *when()* and *otherwise()* to define the routes for your app. Routing has dependency on ngRoute module.

<script type="text/javascript">

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

myApp.config(['$routeProvider', function ($routeProvider) {

$routeProvider. when('/products', {

//route

templateUrl: 'views/products.html', controller: 'productController'

}).

when('/product/:productId', {

//route with parameter

templateUrl: 'views/product.html',controller: 'productController'

}).

otherwise({ //default route redirectTo: '/index' });

}]);

</script>

## What is AngularUI router and how it is different from ngRoute?

The UI-Router is a routing framework for AngularJS built by the*AngularUI*team. Unlike ngRoute, itchanges your angular app views based on state of the app and not based on the route URL (ngRoute).

The ui-router helps you to create nested views, use multiple views on the same page, have multiple views that control a single view, and more.

To use it you need to include reference of *ui-router.js* file into your angular app.

## What is $injector and $inject?

$injector is a service which is used to invoke controller functions, service functions, filter functions, and any other function that might need dependencies as parameters. Angular creates only a single $injector object when an application is bootstrapped and uses that object for invoking.

<script>

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

app.service('s1', function () {

this.value = 22;

});

app.controller("MyCtrl", function ($scope, $injector) {

$scope.doSomething = function () {

var s1 = $injector.get('s1') s1.value += 10

}

$scope.value = function () {

var s1 = $injector.get('s1')

return s1.value

}

});

</script>

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

<button ng-click="doSomething()">increment</button>

{{value()}}

</div>

**$inject** is property which is used to inject the dependencies of a function as an array of strings.

<script type="text/javascript">

var MyController = function(renamed$scope, renamedGreeter) {

// ...

}

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

//inject dependencies as an array of strings

</script>

## What is Dependency Injection in AngularJS?

Dependency Injection (DI) is a software design pattern that deals with how components get hold of theirdependencies. AngularJS comes with a built-in dependency injection mechanism. You can divide your AngularJS app into multiple different types of components which AngularJS can inject into each other.

There are following three ways of injecting dependencies into your code:

**1.** Implicitly from the function parameter names

<script type="text/javascript">

function MyController($scope, greeter) {

// ...

}

</script>

**2.** Using the $inject property annotation

<script type="text/javascript">

var MyController = function(renamed$scope, renamedGreeter) {

// ...

}

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

</script>

**3.** Using the inline array annotation

<script type="text/javascript">

someModule.factory('greeter', ['$window', function (renamed$window) {

// ...

}]);

</script>

## How to do Language 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. These languages specific JSON files can be lazy-loads from the server only when necessary. The angular-translate library (*angular-translate.js*) also provides built-in directives and filters that make the process of internationalizing simple.

<!DOCTYPE html> <html>

<head>

<title>AngularJS Internalization</title>

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

<script src="lib/angular-translate.js"></script>

<script>

var app = angular.module('myApp', ['pascalprecht.translate']); app.config(function ($translateProvider) {

$translateProvider.translations('en', {

TITLE: 'Hello',

PARA: 'This is a paragraph.',

BUTTON\_LANG\_EN: 'english',

BUTTON\_LANG\_DE: 'german'

})

.translations('de', {

TITLE: 'Hallo',

PARA: 'Dies ist ein Paragraph.',

BUTTON\_LANG\_EN: 'englisch',

BUTTON\_LANG\_DE: 'deutsch'

});

//setting default language $translateProvider.preferredLanguage('en');

});

app.controller('TranslateController', function ($translate, $scope) {

$scope.changeLanguage = function (langKey) {

$translate.use(langKey);

};

});

</script>

</head>

<body ng-app="myApp">

<div ng-controller="TranslateController">

<h1>{{ 'TITLE' | translate }}</h1>

<p>{{ 'PARA' | translate }}</p>

<button ng-click="changeLanguage('en')" translate="BUTTON\_LANG\_EN"></button> <button ng-click="changeLanguage('de')" translate="BUTTON\_LANG\_DE"></button>

</div>

</body>

</html>

## What is i18n and L10n?

i18n means Internationalization, where 18 stands for the number of letters in word Internationalization between the first i and last n. It is the process of developing products in such a way that they can be localized for languages and cultures easily.

L10n means Localization, where 10 stand for the number of letters in word Localization between the first L and last n. It is the process of adapting applications and text to enable their usability in a particular culture.

## What is $locale service?

$locale service provides localization rules for various Angular components. As of now the only public api is: id – {string}

For example, a locale id is formatted as: languageId-countryId (e.g. en-us)

## What is a locale ID?

A locale is a specific geographical, political, or cultural region. The most commonly used locale ID consistsof two parts: language code and country code. For example, en-US, en-AU, hi-IN are all valid locale IDs that have both language codes and country codes.

## How to manage cookie in AngularJS?

AngularJS provides*ngCookies*module for reading and writing browser cookies. To use it include the ***angular-cookies.js*** file and set*ngCookies*as a dependency in your angular app. This module provides two servicesfor cookie management: $cookies and $cookieStore.

## What is difference between $cookies and $cookieStore service?

**$cookies -** This service provides read/write access to browser's cookies.

If you want to use existing cookie solution, say read/write cookies from your existing server session system then use $cookie.

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

app.controller('ExampleController', function ($cookies) {

// Retrieving a cookie

var favoriteCookie = $cookies.myFavorite;

// Setting a cookie

$cookies.myFavorite = 'meal';

});

**$cookiesStore -** $cookieStore is a thin wrapper around $cookies. It provides a key-value (string-object) storagethat is backed by session cookies. The objects which are put or retrieved from this storage are automatically serialized or deserialized by angular to JSON and vice versa.

If you are creating a new solution which will persist cookies based on key/value pairs, use $cookieStore.

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

app.controller('ExampleController',function ($cookieStore) {

//Put cookie

$cookieStore.put('myFavorite', 'meal');

//Get cookie

var favoriteCookie = $cookieStore.get('myFavorite');

//Removing a cookie $cookieStore.remove('myFavorite');

});

## How to handle mobile browsers/devices events in AngularJS?

Mobile browsers/devices deal with events differently than desktop browsers. The AngularJS provide **ngTouch library** (*angular-touch.js*) to detect mobile browsers/devices events.

**For example**, Mobile browsers detect a tap event and then wait for second event about 300 ms if any. Soif we’redouble-tapping the device then after this delay the browser fires a click event.

In this way this delay can make our apps unresponsive. Hence, instead of dealing with the click event, we can detect touch event using *ngTouch library*. It handles touch detection for us through the ng-click directive. Hence it will take care of calling the correct click event for mobile.

<button ng-click="save()">Save</button>

## How to detect swipe event in mobile browsers/devices in AngularJS?

The **ngTouch library** provides **swipe** directives to capture user swipes, either left or right across the screen.These events are useful when the user want to swipe to the next photo gallery photo or to a new portion of our app.

The **ngSwipeLeft** directive detects when an HTML element is swiped from the right to the left and the **ngSwipeRight** directive detects when an HTML element is swiped from the left to the right.

## How to do animation with the help of AngularJS?

AngularJS 1.2 comes with animation support via*ngAnimate*module. To enable animations within yourangular app, you need to link the **angular-animate.js** file and include *ngAnimate* module as a dependency within your angular app.

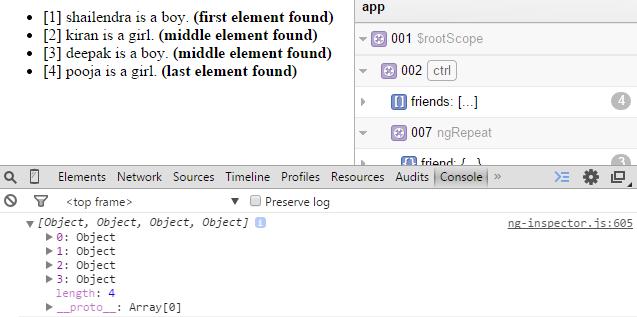
## What directives support animations?

The following directives support animations:

1. **ngRepeat** –It supports enter, leave and move animations.
2. **ngInclude** –It supports enter and leave animations.
3. **ngView** –It supports enter and leave animations.
4. **ngIf** –It supports enter and leave animations.
5. **ngSwitch** –It supports enter and leave animations.
6. **ngClass** –It supports addClass and removeClass animations.
7. **ngShow & ngHide** –These support addClass and removeClass animations.

## How to debug AngularJS app in browser?

**AngularJS Batarang** and **ng-inspector** are browser extensions for Chrome that adds an inspector pane into browser to debug and understand your AngularJS app. ng-inspector also works with safari browser. **ng-inspector Extension:**



## How to securely parse and manipulate your HTML data in AngularJS?

AngularJS provides ngSanitize module to securely parse and manipulate HTML data in your application. To use it include the angular-sanitize.js file and set ngSanitize as a dependency in your angular app.

var app = angular.module('sanitizeExample', ['ngSanitize']); app.controller('ExampleController',function ($scope, $sce) {

var snippet ='<p style="color:blue">an html\n' +

'<em onmouseover="this.textContent=\'PWN3D!\'">click here</em>\n snippet</p>';

$scope.trustedSnippet = $sce.trustAsHtml(snippet);

//sce=Strict Contextual Escaping

});

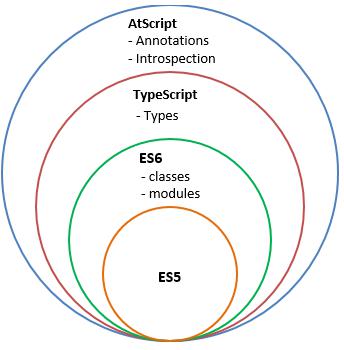
The snippet may contain HTML, CSS, URLs and JavaScript code which you want to safely render in your app.

## What is Angular 2.0?

Angular 2.0 is being written with AtScript, but that doesn't mean you have to write your application codewith AtScript or know anything about AtScript to use Angular 2.0. You can easily write with TypeScript, ES6, ES5, CoffeeScript etc. whatever you like. It is still in development phase (at the time of writing this book).

## What is AtScript?

AtScript isGoogle’s new supersetfor JavaScript**.** It enhances JavaScript with new features to make it morerobust. It is not only designed to run on top of ECMAScript 5 and ECMAScript 6, but on the top of Microsoft’s superset TypeScript language as well.



The aim of AtScript is to make type annotation data available at runtime to enhance JavaScript with type, field and metadata annotations.

## ****What are the basic steps to unit test an AngularJS filter?****

1. Inject the module that contains the filter.
2. Provide any mocks that the filter relies on.
3. Get an instance of the filter using $filter('yourFilterName').
4. Assert your expectations.

Dependency injection is a powerful software design pattern that Angular employs to compose responsibilities through an intrinsic interface. However, for those new to the process, it can be puzzling where you need to configure and mock these dependencies when creating your isolated unit tests. The open-source project “Angular Test Patterns” is a free resource that is focused on dispelling such confusion through high-quality examples.

This question is useful since it can give you a feel for how familiar the candidate is with automated testing (TDD, BDD, E2E), as well as open up a conversation about approaches to code quality.

## ****What should be the maximum number of concurrent “watches”? Bonus: How would you keep an eye on that number?****

To reduce memory consumption and improve performance it is a good idea to limit the number of watches on a page to 2,000. A utility called ng-stats can help track your watch count and digest cycles.

**Jank** happens when your application cannot keep up with the screen refresh rate. To achieve 60 frames-per-second, you only have about 16 milliseconds for your code to execute. It is crucial that the scope digest cycles are as short as possible for your application to be responsive and smooth. Memory use and digest cycle performance are directly affected by the number of active watches. Therefore, it is best to keep the number of watches below 2,000. The open-source utility ng-stats gives developers’ insight into the number of watches Angular is managing, as well as the frequency and duration of digest cycles over time.

Caution: Be wary of relying on a “single magic metric” as the golden rule to follow. You must take the context of your application into account. The number of watches is simply a basic health signal. If you have many thousands of watches, or worse, if you see that number continue to grow as you interact with your page. Those are strong indications that you should look under the hood and review your code.

This question is valuable as it gives insight into how the candidate debugs runtime issues while creating a discussion about performance and optimization.

## ****How do you share data between controllers?****

Create an AngularJS service that will hold the data and inject it inside of the controllers.

Using a service is the cleanest, fastest and easiest way to test.  
However, there are couple of other ways to implement data sharing between controllers, like:

– Using events  
– Using $parent, nextSibling, controllerAs, etc. to directly access the controllers  
– Using the $rootScope to add the data on (not a good practice)  
The methods above are all correct, but are not the most efficient and easy to test.

What is the difference between ng-show/ng-hide and ng-if directives?

ng-show/ng-hide will always insert the DOM element, but will display/hide it based on the condition. ng-if will not insert the DOM element until the condition is not fulfilled.

ng-if is better when we needed the DOM to be loaded conditionally, as it will help load page bit faster compared to ng-show/ng-hide.

We only need to keep in mind what the difference between these directives is, so deciding which one to use totally depends on the task requirements.

## ****What is a digest cycle in AngularJS?****

In each digest cycle Angular compares the old and the new version of the scope model values. The digest cycle is triggered automatically. We can also use $apply() if we want to trigger the digest cycle manually.

For more information, take a look in the ng-book explanation:

Let’s take a peek at how Angular works underneath the hood. How do we get this magical data binding to work in only a few lines of code? It’s important that we understand how the $digest loop works and how to use the $apply() method.

In the normal browser flow, a browser executes callbacks that are registered with an event when that event occurs (e.g., clicking on a link).

Events are fired when the page loads, when an $http request comes back, when the mouse moves or a button is clicked, etc.

When an event is fired/triggered, JavaScript creates an event object and executes any functions listening for the specific events with this event object. This callback method then runs inside the JavaScript function, which returns to the browser, potentially updating the DOM.

No two events can run at the same time. The browser waits until one event handler finishes before the next handler is called.

In non-Angular JavaScript, we can attach a function callback to the click event on a div. Any time that a click event is found on an element, the function callback runs:

var div = document.getElementById("clickDiv");

div.addEventListener("click", function(evt) {

console.log("evt", evt);

});

Open the Chrome developer tools, and copy and paste the previous code inside of any web page and click around the page.

Any time the browser detects a click, the browser calls the function registered with the addEventListener on the document.

When we mix Angular into the flow, it extends this normal browser flow to create an Angular context. The Angular context refers specifically to code that runs inside the Angular event loop, referred to as the $digest loop. To understand the Angular context, we need to look at exactly what goes on inside of it. There are two major components of the $digest loop:

* The $watch list
* The $evalAsync list

**$watch List**

Every time we track an event in the view, we are registering a callback function that we expect to be called when an event happens in the page. Recall our first example:

<!DOCTYPE html>

<html ng-app>

<head>

<title>Simple app</title>

<script

src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.js">

</script>

</head>

<body>

<input ng-model="name" type="text" placeholder="Your name">

<h1>Hello {{ name }}</h1>

</body>

</html>

Any time a user updates the input field, {{ name }} changes in the UI. This change happens because we bind the input field in the UI to the $scope.name property. In order to update the view, Angular needs to track the change. It does so by adding a watch function to the $watch list.

Properties that are on the $scope object are only bound if they are used in the view. In the case above, we’ve added a single function to the $watch list.

Remember: For all UI elements that are bound to a $scope object, a $watch is added to the $watch list.

These $watch lists are resolved in the $digest loop through a process called dirty checking.

**Dirty Checking**

Dirty checking is a simple process that boils down to a very basic concept: It checks whether a value has changed that hasn’t yet been synchronized across the app.

The dirty checking strategy is commonly used in plenty of different applications, beyond Angular. Game engines, database engines, and Object Relational Mappers (ORMs) are some examples of such systems.

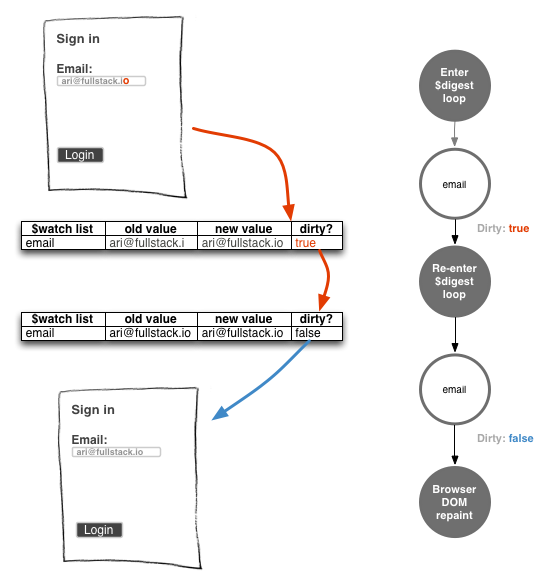
Our Angular app keeps track of the values of the current watches (in the watch object, for those who are curious). Angular walks down the $watch list, and, if the updated value has not changed from the old value, it continues down the list. If the value has changed, the app records the new value and continues down the $watch list.

Once Angular has run through the entire $watch list, if any value changed, the app will fall back into the $watch loop until it detects that nothing has changed.

Why run the loop all over again? If we update a value in the $watch list that updates another value, Angular won’t detect the update unless we rerun the loop.

If the loop runs ten times or more, our Angular app throws an exception, and the app dies. If Angular doesn’t throw this exception, our app could launch into an infinite loop, with bad results.

In future versions of Angular, the framework will use the native browser specification Object.observe(), which will speed up the dirty checking process considerably.



**$watch**

The $watch method on the $scope object sets up a dirty check on every call to $digest inside the Angular event loop. The Angular $digest loop always returns if it detects changes on the expression.

The $watch function itself takes two required arguments and a third optional one:

* watchExpression

The watchExpression can either be a property of a scope object or a function. It runs on **every** call to $digest in the $digest loop.

If the watchExpression is a string, Angular evaluates it in the context of the $scope. If it is a function, then Angular expects it to return the value that should be watched.

* listener/callback

The callback listener function is only called when the current value of the watchExpression and the previous value of the expression are not equal (except during initialization on the first run).

* objectEquality (optional)

The objectEquality parameter is a comparison boolean that tells Angular to check for strict equality.

The $watch function returns a deregistration function for the listener that we can call to cancel Angular’s watch on the value.

// ...

var unregisterWatch =

$scope.$watch('newUser.email',

function(newVal, oldVal) {

if (newVal === oldVal) return; // on init

});

// ...

// later, we can unregister this watcher

// by calling

unregisterWatch();

If we are done watching the newUser.email in this example, we can clean up our watcher by calling the deregistration function it returns.

For instance, let’s say we want to parse an input field value from a full name to split on spaces and find a simple first and last name. Given that our view looks like:

<input type="text" ng-model="full\_name" placeholder="Enter your full name" />

We should **never** use $watch in a controller: It makes it difficult to test the controller. We’re making an allowance here for the sake of illustration, and we’ll move these watches into services later.

We want to set up a $watch listener on the full\_name property and detect any changes to the value. We also want to set the $watch function on the full\_name property.

angular.module("myApp")

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

$scope.$watch('full\_name', function(newVal, oldVal, scope) {

// the newVal of the full\_name will be available here

// while the oldVal is the old value of full\_name

});

}]);

In our example, we’re setting an AngularJS expression that tells our Angular app to “watch the full\_name property for any potential changes on it, and run the function if you detect any changes”.

The listener function is called once on initialization, so the first time around, the value of newVal and oldVal will be undefined (and will be equal). That being the case, it’s generally good to check inside the expression if we’re in the initialization phase or if there is an update to the previous value. We can easily accomplish this check inside the function, like so:

$scope.$watch('full\_name',

function(newVal, oldVal, scope) {

if (newVal === oldVal) {

// This will run only on the initialization of the watcher

} else {

// A change has occurred after initialization

}

});

The $scope.$watch() function sets up a watchExpression on the $scope for ‘full\_name’.

## ****Where should we implement the DOM manipulation in AngularJS?****

In the directives. DOM Manipulations should not exist in controllers, services or anywhere else but in directives.

[Best Practice - Dom Manipulations](http://ng-learn.org/2014/01/Dom-Manipulations/)

Lets talk about best practices and roles in AngularJS. A controller is the middle man. Its main role is to talk to the Service to get the model and then make sure this model is available to our presentation layer(html). Even in large applications, the controller should be small, compact and dumb!

Dom Manipulations should not exist in controllers, services or anywhere else but in directives.

This came up today. We have two products with its own results list controller.

productAResultsCtrl.coffee and productBResultsCtrl.coffee were both presenting this code:

$(".search-details-form").hide()

$("#more").click ->

if $(".search-details-form").is(":hidden")

$(".search-details-form").slideDown()

else

$(".search-details-form").slideUp()

This is meant to hide a search bar with advanced options and if the user clicks on a button we display all the options with a classic jquery slide effect.

This works but

* It is not reusable
* It is not testable
* It include css hard coded selectors dependencies

So in order to resolve these problems, we should always:

write a directive for dom manipulation

the directive should not have dependencies on other html blocks or scope’s parents objects

the directive should not have any hard coded css selector

We are going to write a directive to be applied in the div that should slide down or up based on an external variable.

To avoid dependency problems, we are going to pass this variable as a parameter. We dont care who changes this variable. We only care to react when the variable changes.

Our directive should look something like this:

<div data-my-slide="showDetails"> details content goes here</div>

We use ‘data-’ prefix to make sure html validates and we do not cause IE7, 8 browsers to go into quirks mode. Very Important!!!!

We use ‘my-’ prefix to determine this is a custom directive that belongs to our team.

Lets write the code for this. As always the test comes first:

describe("Unit testing jquery directive", function() {

var $compile, $scope, element;

$scope = element = $compile = void 0;

// Load the module, which contains the directive

beforeEach(module("jqueryDirectives"));

// Store references to $rootScope and $compile so they are available to all

// tests in this describe block

beforeEach(inject(function(\_$compile\_, \_$rootScope\_) {

// The injector unwraps the underscores (\_) from around the parameter names

// when matching

$scope = \_$rootScope\_;

return $compile = \_$compile\_;

}));

it("should slide Down a block", function() {

// Create html fragment

element = angular.element('<div class="form" data-my-slide="showForm">Text

</div>');

// Set variable

$scope.showForm = true;

// Compile a piece of HTML containing the directive

$compile(element)($scope);

$scope.$digest();

// Set expectation

return expect(element.css('height')).toBe('1px');

});

it("should slide Up a block", function() {

// Create html fragment

element = angular.element('<div class="form" data-my-slide="showForm">Text

</div>');

// Set variable

$scope.showForm = false;

// Compile a piece of HTML containing the directive

$compile(element)($scope);

$scope.$digest();

// Set expectation

return expect(element.css('height')).toBe('0px');

});

});

We have created two tests where after compiling the code, our directive applies slideDown or slideUp.

Based on jquery’s documentation the divs’ height gets primarily affected. We take this fact so as to evaluate the success of our directive.

Now lets write the directive

// Here we create a module to group these directives jquery related

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

// Here we add a directive to the module. camelCase naming in this file

// (mySlide) and dash separated in html (my-Slide)

jqueryDirectives.directive("mySlide", [

function() {

return {

// This means the directive can be used as an attribute only. Example <div data-my-slide="variable"> </div>

restrict: "A",

// This is the functions that gets executed after Angular has

//compiled the html

link: function(scope, element, attrs) {

// We dont want to abuse on watch but here it is critical to

//determine if the parameter has changed.

scope.$watch(attrs.mySlide, function(newValue, oldValue) {

// This is our logic. If parameter is true slideDown otherwise

//slideUp.

// TODO: This should be transformed into css transition or angular

// animator if IE family supports it

if (newValue) {

return element.slideDown();

}

else {

return element.slideUp();

}

});

}

};

}

]);

So now on both productAResults.html and productBResults.html we say

<div class="search-details-form" data-my-slide="showRedoSearchDetails">

And we remove the jquery dom maniuplations out of our controllers.

Now we have a reusable, independent, tested directive.

## ****Is it a good or bad practice to use AngularJS together with jQuery?****

It is definitely a bad practice. We need to stay away from jQuery and try to realize the solution with an AngularJS approach. JQuery takes a traditional imperative approach to manipulating the DOM, and in an imperative approach, it is up to the programmer to express the individual steps leading up to the desired outcome.

AngularJS, however, takes a declarative approach to DOM manipulation. Here, instead of worrying about all of the step by step details regarding how to do the desired outcome, we are just declaring what we want and AngularJS worries about the rest, taking care of everything for us.

AngularJS shouldn't be looked at as a replacement to jQuery, in my opinion. Though if you use AngularJS and you are using jQuery as well (which you are allowed to do), that's a clear sign that you are probably using AngularJS incorrectly. So in that sense, I can say that you will naturally use one or the other but I wouldn't call it a replacement.   
  
With that said, before we look at when AngularJS should be used, it might be important to state why I think they are naturally mutually exclusive.  
  
**Declarative vs Imperative Approach to DOM Manipulation**  
jQuery takes a traditional imperative approach to manipulating the DOM. In an imperative approach, it is up to the programmer to express the individual steps leading up to the desired outcome. What do I mean by this? So if we want an action to occur when a user types say 150 characters into an input, in jQuery we would say, "every time the user hits a key, check how many characters are in the input, if it exceeds 150 characters, do the action." Every step is addressed along the way.  
  
AngularJS however takes a declarative approach to DOM manipulation. Here instead of worrying about all of the step by step details regarding how to do the desired outcome, AngularJS abstracts that and allows you to just say what you want done, in this case, "AngularJS, when the state of the input is at 150 characters, do this." We are just declaring what we want and AngularJS worries about the rest, taking care of everything for us.  
  
It might seem like I'm just splitting hairs here, but it's really an important distinction. AngularJS wants you basing your actions around the data models you create. It's how the entire framework works and how your applications will be structured. To simply begin writing side scripts in jQuery where you are plucking out elements and setting up side event listeners just goes against the AngularJS approach in my opinion.  
  
The AngularJS approach does have the one downside that most abstracted layers have, though it is simpler, it gives you less direct control. Often times the control is not needed and when it is there are things like Angular-UI and ngAnimate that often can solve the problem you're having in a more "Angular way", but in the case that it is required, AngularJS actually comes prepackaged with jqLite, which is a subset of jQuery. This was done intentionally to eliminate another dependency, which makes sense given that AngularJS is an attempt to be an all encompassing front-end framework. So this gives you even more reason not to use jQuery with AngularJS.   
  
With all of this said, if you still feel the need to use jQuery, all you need to do is include the jQuery script tag in your page before the AngularJS script tag. AngularJS is smart enough to know that you did this and it will switch to using jQuery for DOM manipulation. This alone shows you that they aren't inherent replacements of each other, but as per the above, they naturally don't go together in many ways.  
  
Okay so that was long winded... but now onto the question, why should you use AngularJS in the first place?  
  
**When to Use AngularJS... That is the Question.**  
The intention of jQuery is to be a library to simplify DOM manipulation. The intention of AngularJS is to be a full framework that provides structure and addresses all aspects of modern web applications that contain heavy front-ends, including DOM manipulation. So when do you use what?  
My opinion is that it depends on the type of application you are building. If you are building a very heavy front end, where most of the heavy lifting is going to be done in the front and all the back end is really going to be is a REST API that you can make calls to... than I would say a full framework like AngularJS (or Backbone.js or Ember.js or React.js) might be a good idea as it will help you with way more than just DOM manipulation.  
  
However if you are creating a good old fashioned site, with a very light front end where most of the work is being done in the back end, a large front end framework might be of much lesser use to you. You could still use it, but if the front end isn't doing a whole lot and is pretty small, it seems like it might not be worth it in my opinion.

Don't design your page, and then change it with [DOM](http://en.wikipedia.org/wiki/Document_Object_Model) manipulations

In jQuery, you design a page, and then you make it dynamic. This is because jQuery was designed for augmentation and has grown incredibly from that simple premise.  
  
But in AngularJS, you must start from the ground up with your architecture in mind. Instead of starting by thinking "I have this piece of the DOM and I want to make it do X", you have to start with what you want to accomplish, then go about designing your application, and then finally go about designing your view.

Don't augment jQuery with AngularJS

Similarly, don't start with the idea that jQuery does X, Y, and Z, so I'll just add AngularJS on top of that for models and controllers. This is really tempting when you're just starting out, which is why I always recommend that new AngularJS developers don't use jQuery at all, at least until they get used to doing things the "Angular Way".  
  
I've seen many developers here and on the mailing list create these elaborate solutions with jQuery plugins of 150 or 200 lines of code that they then glue into AngularJS with a collection of callbacks and $applys that are confusing and convoluted; but they eventually get it working! The problem is that inmost cases that jQuery plugin could be rewritten in AngularJS in a fraction of the code, where suddenly everything becomes comprehensible and straightforward.  
  
The bottom line is this: when solutioning, first "think in AngularJS"; if you can't think of a solution, ask the community; if after all of that there is no easy solution, then feel free to reach for the jQuery. But don't let jQuery become a crutch or you'll never master AngularJS.

Always think in terms of architecture

First know that [single-page applications](http://en.wikipedia.org/wiki/Single-page_application) are applications. They're not webpages. So we need to think like a server-side developer in addition to thinking like a client-side developer. We have to think about how to divide our application into individual, extensible, testable components.  
  
So then how do you do that? How do you "think in AngularJS"? Here are some general principles, contrasted with jQuery.  
  
**The view is the "official record"**  
In jQuery, we programmatically change the view. We could have a dropdown menu defined as a ullike so:

<ul class="main-menu">

    <li class="active">

        <a href="#/home">Home</a>

    </li>

    <li>

        <a href="#/menu1">Menu 1</a>

        <ul>

            <li><a href="#/sm1">Submenu 1</a></li>

            <li><a href="#/sm2">Submenu 2</a></li>

            <li><a href="#/sm3">Submenu 3</a></li>

        </ul>

    </li>

    <li>

        <a href="#/home">Menu 2</a>

    </li>

</ul>

In jQuery, in our application logic, we would activate it with something like:

$('.main-menu').dropdownMenu();

When we just look at the view, it's not immediately obvious that there is any functionality here. For small applications, that's fine. But for non-trivial applications, things quickly get confusing and hard to maintain.  
In AngularJS, though, the view is the official record of view-based functionality. Our ul declaration would look like this instead:

<ul class="main-menu" dropdown-menu> ...</ul>

These two do the same thing, but in the AngularJS version anyone looking at the template knows what's supposed to happen. Whenever a new member of the development team comes on board, she can look at this and then *know* that there is a directive called dropdownMenu operating on it; she doesn't need to intuit the right answer or sift through any code. The view told us what was supposed to happen. Much cleaner.  
  
Developers new to AngularJS often ask a question like: how do I find all links of a specific kind and add a directive onto them. The developer is always flabbergasted when we reply: you don't. But the reason you don't do that is that this is like half-jQuery, half-AngularJS, and no good. The problem here is that the developer is trying to "do jQuery" in the context of AngularJS. That's never going to work well. The view *is* the official record. Outside of a directive (more on this below), you never, ever, *never*change the DOM. And directives are applied *in the view*, so intent is clear.  
  
Remember: don't design, and then mark up. You must architect, and then design.  
  
**Data binding**  
This is by far one of the most awesome features of AngularJS and cuts out a lot of the need to do the kinds of DOM manipulations I mentioned in the previous section. AngularJS will automatically update your view so you don't have to! In jQuery, we respond to events and then update content. Something like:

$.ajax({  url: '/myEndpoint.json',  success: function ( data, status ) {

    $('ul#log').append('<li>Data Received!</li>');

  }

});

For a view that looks like this:

<ul class="messages" id="log"></ul>

Apart from mixing concerns, we also have the same problems of signifying intent that I mentioned before. But more importantly, we had to manually reference and update a DOM node. And if we want to delete a log entry, we have to code against the DOM for that too. How do we test the logic apart from the DOM? And what if we want to change the presentation?  
This a little messy and a trifle frail. But in AngularJS, we can do this:

$http( '/myEndpoint.json' ).then( function ( response ) {

    $scope.log.push( { msg: 'Data Received!' } );

});

And our view can look like this:

<ul class="messages">

    <li ng-repeat="entry in log">{{ entry.msg }}</li>

</ul>

But for that matter, our view could look like this:  
<div class="messages">

    <div class="alert" ng-repeat="entry in log">

        {{ entry.msg }}

     </div>

</div>

And now instead of using an unordered list, we're using Bootstrap alert boxes. And we never had to change the controller code! But more importantly, no matter where or howthe log gets updated, the view will change too. Automatically. Neat!  
  
Though I didn't show it here, the data binding is two-way. So those log messages could also be editable in the view just by doing this: <input ng-model="entry.msg" />. And there was much rejoicing.  
  
**Distinct model layer**In jQuery, the DOM is kind of like the model. But in AngularJS, we have a separate model layer that we can manage in any way we want, completely independently from the view. This helps for the above data binding, maintains [separation of concerns](http://en.wikipedia.org/wiki/Separation_of_concerns), and introduces far greater testability. Other answers mentioned this point, so I'll just leave it at that.  
  
**Separation of concerns**  
And all of the above tie into this over-arching theme: keep your concerns separate. Your view acts as the official record of what is supposed to happen (for the most part); your model represents your data; you have a service layer to perform reusable tasks; you do DOM manipulation and augment your view with directives; and you glue it all together with controllers. This was also mentioned in other answers, and the only thing I would add pertains to testability, which I discuss in another section below.  
  
**Dependency injection**  
To help us out with separation of concerns is [dependency injection](http://en.wikipedia.org/wiki/Dependency_injection) (DI). If you come from a server-side language (from [Java](http://en.wikipedia.org/wiki/Java_%28programming_language%29) to [PHP](http://en.wikipedia.org/wiki/PHP)) you're probably familiar with this concept already, but if you're a client-side guy coming from jQuery, this concept can seem anything from silly to superfluous to hipster. But it's not. :-)  
  
From a broad perspective, DI means that you can declare components very freely and then from any other component, just ask for an instance of it and it will be granted. You don't have to know about loading order, or file locations, or anything like that. The power may not immediately be visible, but I'll provide just one (common) example: testing.  
  
Let's say in our application, we require a service that implements server-side storage through a [REST](http://en.wikipedia.org/wiki/Representational_State_Transfer) API and, depending on application state, local storage as well. When running tests on our controllers, we don't want to have to communicate with the server - we're testing the controller, after all. We can just add a mock service of the same name as our original component, and the injector will ensure that our controller gets the fake one automatically - our controller doesn't and needn't know the difference.  
Speaking of testing...  
  
**Test-driven development – always**

This is really part of section 3 on architecture, but it's so important that I'm putting it as its own top-level section.  
Out of all of the many jQuery plugins you've seen, used, or written, how many of them had an accompanying test suite? Not very many because jQuery isn't very amenable to that. But AngularJS is.  
  
In jQuery, the only way to test is often to create the component independently with a sample/demo page against which our tests can perform DOM manipulation. So then we have to develop a component separately and then integrate it into our application. How inconvenient! So much of the time, when developing with jQuery, we opt for iterative instead of test-driven development. And who could blame us?  
  
But because we have separation of concerns, we can do test-driven development iteratively in AngularJS! For example, let's say we want a super-simple directive to indicate in our menu what our current route is. We can declare what we want in the view of our application:

<a href="/hello" when-active>Hello</a>

Okay, now we can write a test for the non-existent when-active directive:  
  
it( 'should add "active" when the route changes',

inject(function() {

    var elm = $compile( '<a href="/hello" when-active>Hello</a>' )( $scope );

    $location.path('/not-matching');

    expect( elm.hasClass('active') ).toBeFalsey();

    $location.path( '/hello' );

    expect( elm.hasClass('active') ).toBeTruthy();

}));

And when we run our test, we can confirm that it fails. Only now should we create our directive:

.directive( 'whenActive', function ( $location ) {

    return {

        scope: true,

        link: function ( scope, element, attrs ) {

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

                if ( $location.path() == element.attr( 'href' ) ) {

                    element.addClass( 'active' );

                }

                else {

                    element.removeClass( 'active' );

                }

            });

        }

    };

});

Our test now passes and our menu performs as requested. Our development is bothiterative and test-driven. Wicked-cool.

**Conceptually, directives are not packaged jQuery**

You'll often hear "only do DOM manipulation in a directive". This is a necessity. Treat it with due deference!  
  
But let's dive a little deeper...  
  
Some directives just decorate what's already in the view (think ngClass) and therefore sometimes do DOM manipulation straight away and then are basically done. But if a directive is like a "widget" and has a template, it should also respect separation of concerns. That is, the template too should remain largely independent from its implementation in the link and controller functions.  
AngularJS comes with an entire set of tools to make this very easy; with ngClass we can dynamically update the class; ngModel allows two-way data binding; ngShow and ngHide programmatically show or hide an element; and many more - including the ones we write ourselves. In other words, we can do all kinds of awesomeness without DOM manipulation. The less DOM manipulation, the easier directives are to test, the easier they are to style, the easier they are to change in the future, and the more re-usable and distributable they are.  
  
I see lots of developers new to AngularJS using directives as the place to throw a bunch of jQuery. In other words, they think "since I can't do DOM manipulation in the controller, I'll take that code put it in a directive". While that certainly is much better, it's often still wrong.  
  
Think of the logger we programmed in section 3. Even if we put that in a directive, we stillwant to do it the "Angular Way". It still doesn't take any DOM manipulation! There are lots of times when DOM manipulation is necessary, but it's a lot rarer than you think! Before doing DOM manipulation anywhere in your application, ask yourself if you really need to. There might be a better way.  
  
Here's a quick example that shows the pattern I see most frequently. We want a toggle-able button. (Note: this example is a little contrived and a skosh verbose to represent more complicated cases that are solved in exactly the same way.)

.directive( 'myDirective', function () {

    return {

        template: '<a class="btn">Toggle me!</a>',

        link: function ( scope, element, attrs ) {

            var on = false;

            $(element).click( function () {

                on = !on;

                $(element).toggleClass('active', on);

            });

        }

    };

});

There are a few things wrong with this:

1. First, jQuery was never necessary. There's nothing we did here that needed jQuery at all!
2. Second, even if we already have jQuery on our page, there's no reason to use it here; we can simply use angular.element and our component will still work when dropped into a project that doesn't have jQuery.
3. Third, even assuming jQuery was required for this directive to work, jqLite (angular.element) will always use jQuery if it was loaded! So we needn't use the $ - we can just use angular.element.
4. Fourth, closely related to the third, is that jqLite elements needn't be wrapped in $ - the element that is passed to the link function would already be a jQuery element!
5. And fifth, which we've mentioned in previous sections, why are we mixing template stuff into our logic?

This directive can be rewritten (even for very complicated cases!) much more simply like so:

.directive( 'myDirective', function () {

    return {

        scope: true,

        template: '<a class="btn" ng-class="{active: on}" ng-click="toggle()">Toggle me!</a>',

        link: function ( scope, element, attrs ) {

            scope.on = false;

            scope.toggle = function () {

                scope.on = !scope.on;

            };

        }

    };

});

Again, the template stuff is in the template, so you (or your users) can easily swap it out for one that meets any style necessary, and the logic never had to be touched. Reusability - boom!  
  
And there are still all those other benefits, like testing - it's easy! No matter what's in the template, the directive's internal API is never touched, so refactoring is easy. You can change the template as much as you want without touching the directive. And no matter what you change, your tests still pass.  
  
w00t!  
So if directives aren't just collections of jQuery-like functions, what are they? Directives are actually extensions of HTML. If HTML doesn't do something you need it to do, you write a directive to do it for you, and then use it just as if it was part of HTML.  
  
Put another way, if AngularJS doesn't do something out of the box, think how the team would accomplish it to fit right in with ngClick, ngClass, et al.

**Summary**

Don't even use jQuery. Don't even include it. It will hold you back. And when you come to a problem that you think you know how to solve in jQuery already, before you reach for the $, try to think about how to do it within the confines the AngularJS. If you don't know, ask! 19 times out of 20, the best way to do it doesn't need jQuery and to try to solve it with jQuery results in more work for you.

## ****If you were to migrate from Angular 1.4 to Angular 1.5, what is the main thing that would need refactoring?****

Changing .directive to .component to adapt to the new Angular 1.5 components

## ****How would you specify that a scope variable should have one-time binding only?****

By using “::” in front of it. This allows the check if the candidate is aware of the available variable bindings in AngularJS.

## ****What is the difference between one-way binding and two-way binding?****

– One way binding implies that the scope variable in the html will be set to the first value its model is bound to (i.e. assigned to)  
– Two way binding implies that the scope variable will change its value every time its model is assigned to a different value.

## ****Explain how**** $scope.$apply() ****works****

$scope.$apply re-evaluates all the declared ng-models and applies the change to any that have been altered (i.e. assigned to a new value)  
Explanation: $scope.$apply() is one of the core angular functions that should never be used explicitly, it forces the angular engine to run on all the watched variables and all external variables and apply the changes on their values.

## ****What directive would you use to hide elements from the HTML DOM by removing them from that DOM not changing their styling?****

The ngIf Directive, when applied to an element, will remove that element from the DOM if it’s condition is false.

## ****What makes the**** angular.copy() ****method so powerful?****

It creates a deep copy of the variable.

A deep copy of a variable means it doesn’t point to the same memory reference as that variable. Usually assigning one variable to another creates a “shallow copy”, which makes the two variables point to the same memory reference. Therefore if we change one, the other changes as well.

## ****How would you make an Angular service return a promise? Write a code snippet as an example****

To add promise functionality to a service, we inject the “$q” dependency in the service, and then use it like so:

angular.factory('testService', function($q){

return {

getName: function(){

var deferred = $q.defer();

//API call here that returns data

testAPI.getName().then(function(name){

deferred.resolve(name)

})

return deferred.promise;

}

}

})

The $q library is a helper provider that implements promises and deferred objects to enable asynchronous functionality

## ****What is the role of services in AngularJS and name any services made available by default?****

– AngularJS Services are objects that provide separation of concerns to an AngularJS app.  
– AngularJS Services can be created using a factory method or a service method.  
– Services are singleton components. All components of the application (into which the service is injected) will work with single instance of the service.  
– An AngularJS service allows developing of business logic without depending on the View logic which will work with it.

Few of the inbuilt services in AngularJS are:  
– the $http service: The $http service is a core Angular service that facilitates communication with the remote HTTP servers via the browser’s XMLHttpRequest object or via JSONP  
– the $log service: Simple service for logging. Default implementation safely writes the message into the browser’s console  
– the $anchorScroll: it scrolls to the element related to the specified hash or (if omitted) to the current value of $location.hash()  
Why should one know about AngularJS Services, you may ask. Well, understanding the purpose of AngularJS Services helps bring modularity to AngularJS code.  
Services are the best may to evolve reusable API within and AngularJS app

Overview:

* AngularJS Services help create reusable components.
* A Service can be created either using the service() method or the factory() method.
* A typical service can be injected into another service or into an AngularJS Controller.

## ****When creating a directive, it can be used in several different ways in the view. Which ways for using a directive do you know? How do you define the way your directive will be used?****

When you create a directive, it can be used as an attribute, element or class name. To define which way to use, you need to set the restrict option in your directive declaration.

The restrict option is typically set to:

‘A’ – only matches attribute name  
‘E’ – only matches element name  
‘C’ – only matches class name

These restrictions can all be combined as needed:

‘AEC’ – matches either attribute or element or class name

## ****When should you use an attribute versus an element?****

Use an element when you are creating a component that is in control of the template. Use an attribute when you are decorating an existing element with new functionality.

This topic is important so developers can understand the several ways a directive can be used inside a view and when to use each way.

## ****How do you reset a**** $timeout****,**** $interval()****, and disable a**** $watch()****?****

To reset a timeout and/or $interval, assign the result of the function to a variable and then call the .cancel() function.

var customTimeout = $timeout(function () {

// arbitrary code

}, 55);

$timeout.cancel(customTimeout);

to disable $watch(), we call its deregistration function. $watch() then returns a deregistration function that we store to a variable and that will be called for cleanup

var deregisterWatchFn = $scope.$on(‘$destroy’, function () {

// we invoke that deregistration function, to disable the watch

deregisterWatchFn();

});

## ****Explain what is a**** $scope ****in AngularJS****

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. Scopes are objects that refer to the model. They act as glue between controller and view.

This question is important as it will judge a persons knowledge about a $scope object, and it is one of the most important concepts in AngularJS. Scope acts like a bridge between view and model.

## ****What are Directives?****

Directives are markers on a DOM element (such as an attribute, element name, comment or CSS class) that tell AngularJS’s HTML compiler ($compile) to attach a specified behavior to that DOM element (e.g. via event listeners), or even to transform the DOM element and its children. Angular comes with a set of these directives built-in, like ngBind, ngModel, and ngClass. Much like you create controllers and services, you can create your own directives for Angular to use. When Angular bootstraps your application, the HTML compiler traverses the DOM matching directives against the DOM elements.

This question is important because directives define the UI while defining a single page app. You need to be very clear about how to create a new custom directive or use the existing ones already pre-build in AngularJS.

## ****What is DDO Directive Definition Object?****

“DDO is an object used while creating a custome directive. A standard DDO object has following parameters.

var directiveDefinitionObject = {

priority: 0,

template: '<div></div>', // or // function(tElement, tAttrs) { ... },

// or

// templateUrl: 'directive.html', // or // function(tElement, tAttrs) { ... },

transclude: false,

restrict: 'A',

templateNamespace: 'html',

scope: false,

controller: function($scope, $element, $attrs, $transclude, otherInjectables) { ... },

controllerAs: 'stringIdentifier',

bindToController: false,

require: 'siblingDirectiveName', // or // ['^parentDirectiveName', '?optionalDirectiveName', '?^optionalParent'],

compile: function compile(tElement, tAttrs, transclude) {

return {

pre: function preLink(scope, iElement, iAttrs, controller) { ... },

post: function postLink(scope, iElement, iAttrs, controller) { ... }

}

// or

// return function postLink( ... ) { ... }

},

// or

// link: {

// pre: function preLink(scope, iElement, iAttrs, controller) { ... },

// post: function postLink(scope, iElement, iAttrs, controller) { ... }

// }

// or

// link: function postLink( ... ) { ... }

};"

This question mainly judges whether candidate knows about creating custom directives.

## **What is a singleton pattern and where we can find it in Angularjs?**

Is a great pattern that restricts the use of a class more than once. We can find singleton pattern in angular in dependency injection and in the services.

In a sense, if you do 2 times ‘new Object()‘ without this pattern, you will be alocating 2 pieces of memory for the same object. With singleton pattern, if the object exists, you reuse it.

## ****What is an interceptor? What are common uses of it?****

An interceptor is a middleware code where all the $http requests go through.

The interceptor is a factory that are registered in $httpProvider. You have 2 types of requests that go through the interceptor, request and response (with requestError and responseErrorrespectively). This piece of code is very useful for error handling, authentication or middleware in all the requests/responses.

## ****How would you programatically change or adapt the template of a directive before it is executed and transformed?****

You would use the compile function. The compile function gives you access to the directive’s template before transclusion occurs and templates are transformed, so changes can safely be made to DOM elements. This is very useful for cases where the DOM needs to be constructed based on runtime directive parameters.

## ****How would you validate a text input field for a twitter username, including the @ symbol?****

You would use the ngPattern directive to perform a regex match that matches Twitter usernames. The same principal can be applied to validating phone numbers, serial numbers, barcodes, zip codes and any other text input.

## ****How would you implement application-wide exception handling in your Angular app?****

Angular has a built-in error handler service called $exceptionHandler which can easily be overriden as seen below:

myApp.factory('$exceptionHandler', function($log, ErrorService) {

return function(exception, cause) {

if (console) {

$log.error(exception);

$log.error(cause);

}

ErrorService.send(exception, cause);

};

});

This is very useful for sending errors to third party error logging services or helpdesk applications. Errors trapped inside of event callbacks are not propagated to this handler, but can manually be relayed to this handler by calling $exceptionHandler(e) from within a try catch block.

## ****How do you hide an HTML element via a button click in AngularJS?****

You can do this by using the ng-hide directive in conjunction with a controller we can hide an HTML element on button click.

<div ng-controller="MyCtrl">

<button ng-click="hide()">Hide element</button>

<p ng-hide="isHide">Hello World!</p>

</div>

function MyCtrl($scope){

$scope.isHide = false;

$scope.hide = function(){

$scope.isHide = true;

}

}

## ****How would you react on model changes to trigger some further action? For instance, say you have an input text field called**** email ****and you want to trigger or execute some code as soon as a user starts to type in their email.****

We can achieve this using $watch function in our controller.

function MyCtrl($scope) {

$scope.email = "";

$scope.$watch("email", function(newValue, oldValue) {

if ($scope.email.length > 0) {

console.log("User has started writing into email");

}

});

}

## ****How do you disable a button depending on a checkbox’s state?****

We can use the ng-disabled directive and bind its condition to the checkbox’s state.

<body ng-app>

<label><input type="checkbox" ng-model="checked"/>Disable Button</label>

<button ng-disabled="checked">Select me</button>

</body>

## What is Angular Expression? How do you differentiate between Angular expressions and JavaScript expressions?

Angular expressions are code snippets that are usually placed in binding such as {{ expression }} similar to JavaScript.

The main differences between Angular expressions and JavaScript expressions are:

* **Context :** The expressions are evaluated against a scope object in Angular, while Javascript expressions are evaluated against the global window
* **Forgiving:** In Angular expression, the evaluation is forgiving to null and undefined whereas in JavaScript undefined properties generate TypeError or ReferenceError
* **No Control Flow Statements:** We cannot use loops, conditionals or exceptions in an Angular expression
* **Filters:** In Angular unlike JavaScript, we can use filters to format data before displaying it

## What is the difference between link and compile in Angular.js?

* Compile function is used for template DOM Manipulation and to collect all the directives.
* Link function is used for registering DOM listeners as well as instance DOM manipulation and is executed once the template has been cloned.

## What are the characteristics of ‘Scope’?

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. The characteristics of Scope are:

* Scopes provide APIs ($watch) to observe model mutations.
* Scopes provide APIs ($apply) to propagate any model changes through the system into the view from outside of the “Angular realm” (controllers, services, Angular event handlers).
* Scopes can be nested to limit access to the properties of application components while providing access to shared model properties. Nested scopes are either “child scopes” or “isolate scopes”. A “child scope” (prototypically) inherits properties from its parent scope. An “isolate scope” does not. See isolated scopes for more information.
* Scopes provide context against which expressions are evaluated. For example {{username}} expression is meaningless, unless it is evaluated against a specific scope which defines the username property.

## What are the advantages of using Angular.js framework?

Angular.js framework has the following advantages:

* Supports two way data-binding
* Supports MVC pattern
* Support static template and angular template
* Can add custom directive
* Supports REST full services
* Supports form validations
* Support both client and server communication
* Support dependency injection
* Applying Animations
* Event Handlers

## What is the difference between AngularJS and backbone.js?

AngularJS combines the functionalities of most third party libraries and supports individual functionalities required to develop HTML5 Apps.  While Backbone.js does these jobs individually.

## Explain what is injector in AngularJS?

An injector is a service locator, used to retrieve object instance as defined by provider, instantiate types, invoke methods, and load modules.

## What is factory method in AngularJS?

Factory method is used for creating a directive.  It is invoked when the compiler matches the directive for the first time. We can invoke the factory method using

$injector.invoke.

Syntax: module.factory( 'factoryName', function );  
Result: When declaring factoryName as an injectable argument you will be provided with the value that is returned by invoking the function reference passed to module.factory.

## What is ng-app, ng-init and ng-model?

* ng-app : Initializes application.
* ng-model : Binds HTML controls to application data.
* ng-Controller : Attaches a controller class to view.
* ng-repeat : Bind repeated data HTML elements. Its like a for loop.
* ng-if : Bind HTML elements with condition.
* ng-show : Used to show the HTML elements.
* ng-hide : Used to hide the HTML elements.
* ng-class : Used to assign CSS class.
* ng-src : Used to pass the URL image etc.

## Does Angular use the jQuery library?

Yes, Angular can use jQuery if it’s present in the app when the application is being bootstrapped. If jQuery is not present in the script path, Angular falls back to its own implementation of the subset of jQuery that we call jQLite.

## Can AngularJS have multiple ng-app directives in a single page?

No. 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. If another ng-app directive has been placed then it will not be processed by AngularJS and we will need to manually bootstrap the second app, instead of using second ng-app directive.

## Can angular applications (ng-app) be nested within each other?

No. AngularJS applications cannot be nested within each other.

## What is internationalization and how to implement it in AngularJS?

Internationalization is a way in which you can show locale specific information on a website. AngularJS supports inbuilt internationalization for three types of filters: currency, date and numbers. To implement internalization, we only need to incorporate corresponding js according to locale of the country. By default it handles the locale of the browser.

## On which types of component can we create a custom directive?

AngularJS provides support to create custom directives for the following:

* Element directives − Directive activates when a matching element is encountered.
* Attribute − Directive activates when a matching attribute is encountered.
* CSS − Directive activates when a matching css style is encountered.
* Comment − Directive activates when a matching comment is encountered.

## What is $rootscope in AngularJS?

Every application has a single root scope. All other scopes are descendant scopes of the root scope. Scopes provide separation between the model and the view, via a mechanism for watching the model for changes. They also provide event emission/broadcast and subscription facility.

## Can we have nested controllers in AngularJS?

Yes, we can create nested controllers in AngularJS. Nested controllers are defined in hierarchical manner while using in View.

## What is bootstrapping in AngularJS?

Bootstrapping in AngularJS is nothing but initializing, or starting the Angular app. AngularJS supports automatic and manual bootstrapping.

* Automatic Bootstrapping: this is done by adding ng-app directive to the root of the application, typically on the tag or tag if you want angular to bootstrap your application automatically. When angularJS finds ng-app directive, it loads the module associated with it and then compiles the DOM.
* Manual Bootstrapping:Manual bootstrapping provides you more control on how and when to initialize your angular App. It is useful where you want to perform any other operation before Angular wakes up and compile the page.

## What does SPA (Single Page Application) mean? How can we implement SPA with Angular?

Single Page Applications (SPAs) are web apps that load a single HTML page and dynamically update that page as the user interacts with the app. In an SPA the page never reloads, though parts of the page may refresh. This reduces the round trips to the server to a minimum.

It’s a concept where we create a single shell page or master page and load the webpages inside that master page instead of loading pages from the server by doing post backs. We can implement SPA with Angular using Angular routes. You can read up about SPAs [here](http://www.edureka.co/blog/spa-using-angularjs).

## Why AngularJS?

AngularJS lets us extend HTML vocabulary for our application resulting in an expressive, readable, and quick to develop environment. Some of the advantages are:

* MVC implementation is done right.
* It extends HTML using directives, expression and data binding techniques to define a powerful HTML template.
* Two way data-binding, form validations, routing supports, inbuilt services.
* REST friendly.
* Dependency injection support.
* It helps you to structure and test your JavaScript code.

## Is AngularJS compatible with all browsers?

Yes AngularJS is compatible with the following browsers: Safari, Chrome, Firefox, Opera 15, IE9 and mobile browsers (Android, Chrome Mobile, iOS Safari).

## How to implement routing in AngularJS?

It is a five-step process:

* Step 1: – Add the “Angular-route.js” file to your view.
* Step 2: – Inject “ngroute” functionality while creating Angular app object.
* Step 3: – Configure the route provider.
* Step 4: – Define hyperlinks.
* Step 5: – Define sections where to load the view.

## Explain $q service, deferred and promises.

* ‘Promises’ are post processing logics which are executed after some operation/action is completed whereas ‘deferred’ is used to control how and when those promise logics will execute.
* We can think about promises as “WHAT” we want to fire after an operation is completed while deferred controls “WHEN” and “HOW” those promises will execute.
* “$q” is the angular service which provides promises and deferred functionality.

## Interview Questions for recruiting experienced AngularJS Developers

AngularJS is one of the most popular JavaScript frameworks available today. AngularJS is a great framework that continues to evolve with the community. The AngularJS community is not only large; it’s highly engaged as well.

Choosing the right AngularJS developer is essential to your organization’s digital progress. The right talent can be pivotal to your software development team. If you find the finest of developers, you can succeed in securing the top development team.

Here, this article offers a set of insightful questions to help you identify amazing AngularJS developers who are experts in their field.

Before jumping in the questions, let’s take a look at what skills you should look for in an AngularJS developer.

### Must have AngularJS Skills

Below are the must have skills you should look for when hiring an AngularJS developer.

#### Proficiency with JavaScript, HTML, and CSS.

I know this is obvious, but if you want to hire a great AngularJS professional he needs to know JavaScript and HTML. The efficiency with JavaScript ensures better performance and a cleaner modular code structure. He should be well-versed in the best practices that are critical to building AngularJS web applications properly.

#### Good understanding of MVVM and MVC.

If you want to hire a professional AngularJS developer, he must know MVC/MVVM model, design patterns. The candidate should have a good grasp of –

* Separation of concerns
* Data binding
* Command binding
* Validation
* Unit testing

#### Build an App.

Ask whether the candidate has built an app?

This implies that this candidate is much more valuable than someone who says they know AngularJS but hasn’t actually built anything. Ask questions in details here

#### Strong Communication Skills.

Lastly and maybe most important. An Angular developer might very well become the main communication node of your team. They will need to constantly cooperate with the back-end team, the HTML coders, and the designers. See whether the candidate can explain technical concepts effectively to coworkers, and less technically savvy clients.

Make sure your AngularJS developer is an **outstanding problem solver** with communication skills who is capable of taking your applications to the next level.

#### Important AngularJS Topics

Grill candidates on the below topics, go more in depth. This will provide you a fair view of candidate’s knowledge.

* Modules
* Directives
* Filters
* Services
* Routing
* Templates
* Scope
* Data Binding
* Dependency Injection
* Expressions
* MVC (Model, View, Controller)
* Unit Testing

## AngularJS Interview Questions

Now here is the list of AngularJS interview questions for experienced. Read them, bookmark them, comment on them, or even contribute your own.

1. Mention some AngularJS directives and their purpose.
2. How to create Directive using AngularJS. Give an example.
3. Can we use the open-source Closure Library with Angular? How?
4. How could one create a single-page application using AngularJS? Which provider is used to achieve this objective?
5. Explain how MVC is achieved with AngularJS? What are the benefits of client-side MVC, in general?
6. What is restrict option in directive? Can you define multiple restrict options on a directive?
7. What are the differences between AngularJS module’s Service, Provider and Factory?
8. The following HTML is given
9. <button ng-click=’showAlert()”>ALERT</button>
10. How can the attached controller look like, to show an alert window when clicking on the button?
11. What is the difference between “.$digest()” and “.$apply()”? Why would you ever call “.$digest()” on a scope?
12. Explain AngularJS boot process.
13. How can be the permission denied errors handled with AngularJS? Explain with example.
14. At framework level, how does Angular retrieve the matching elements for processing? Give example.
15. List a few ways to improve performance in an AngularJS app.
16. Can we have nested controllers in AngularJS? In the case of nested controllers, does the $scope object is shared across all controllers?
17. With which methods you can bootstrap your angular app for multiple modules? Explain the methods in detail.
18. How AngularJS is compiled? How is AngularJS compilation different from other JavaScript’s framework?
19. What is dependency injection and how does it work in Angular? How does DI benefit in Angular?
20. What is routing? How can we implement routing in Angular?
21. How does Angular JS know when to perform dirty checking and update DOM output?
22. How can we dynamically create forms in Angular?
23. What are the different custom directive types in AngularJS? How can custom directives  be applied to an element as well as attributes?
24. Describe an approach to testing directives.
25. Explain $q service, deferred and promises.
26. What is Isolate Scope and why it is required?
27. Which means of communication between modules of your application are easily testable?
28. With more than one ng-app in an HTML document (an HTML page), are they automatically initialized? Describe the AngularJS application initialization process with multiple ng-app in an HTML document/page.
29. How to do Language Internationalization in AnglarJS?
30. Explain digest life-cycle in AngularJS. What happens when you do the changes to the scope outside the angular context?
31. What is the difference between Angular and Polymer? What problems does Polymer solve that AngularJS has not or will not? Are there plans to tie Polymer in with AngularJS in the future?
32. The most popular e2e testing tool for AngularJS is Protractor. There are also others which rely on similar mechanisms. Describe how e2e testing of AngularJS applications work?
33. Ask questions on Angular 2.0, AtScript and its up gradations. Asking questions on up gradations or upcoming technologies gives you a better idea of candidate’s passion for technology.