B.Sc.ECS-III
Sem-VI
AngularJS
(Additional Notes)

The Four Functions of the AngularJS Directive Life Cycle

There are many options that can be configured and how those options are related to each other is important. Each directive undergoes something similar to a life cycle as AngularJS compiles and links the DOM. The directive lifecycle begins and ends within the AngularJS bootstrapping process, before the page is rendered. In a directive's life cycle, there are four distinct functions that can execute if they are defined. Each enables the developer to control and customize the directive at different points of the life cycle.

The four functions are: compile, controller, pre-link and post-Link.

The compile function allows the directive to manipulate the DOM before it is compiled and linked thereby allowing it to add/remove/change directives, as well as, add/remove/change other DOM elements.

The controller function facilitates directive communication. Sibling and child directives can request the controller of their siblings and parents to communicate information.

The pre-link function allows for private $scope manipulation before the post-link process begins.

The post-link method is the primary workhorse method of the directive.

In the directive, post-compilation DOM manipulation takes place, event handlers are configured, and so are watches and other things. In the declaration of the directive, the four functions are defined like this.

```
.directive("directiveName",function () {
                return {
            controller: function()    {
                        // controller code here...
                    },
                compile:   {
                            // compile code here...
                  return   {
              pre: function()   {
                        // pre-link code here...
                    },
                post: function() {

                    // post-link code here...
                     }
                  };
                }
              }
            })
```

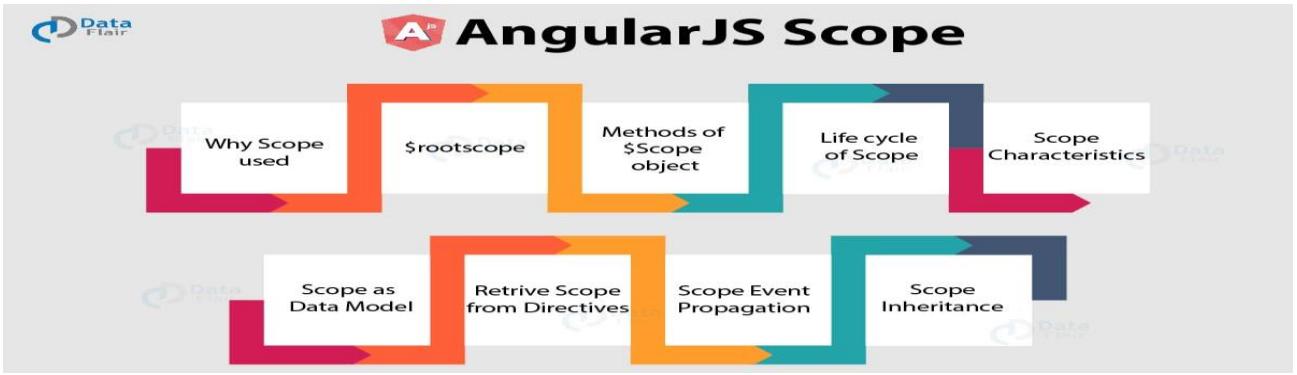
Commonly, not all of the functions are needed. In most circumstances, developers will simply create a controller and post-link function following the pattern below.

AngularJS Scope Life Cycle – Characteristics, Inheritance, $Scope Object

As we know that **AngularJS directives** can interact with the controller of a page but how it is possible? What made it possible to pass an object through directive? Well, directives have a scope associated with it which made all these things possible in angular. Let us explore the new topic, AngularJS Scope.

In this AngularJS Scope tutorial, we will learn why scope is used in AngularJS with its lifecycle and characteristics. Along with this, we will learn $rootscope, methods of $scope object, Scope as data model and inheritance.

So, let's start the AngularJS Scope.



What is AngularJS Scope?

An AngularJS Scope is a built-in object. It is available for both view (HTML part) and controller. For every controller, angular creates or injects $scope object in angular. Therefore, the behavior attached in one controller cannot be accessed by another **controller in AngularJS**.

During the constructor definition, the first argument passed through controller is the $scope.

Syntax-

```
<script>
var mainApp = angular.module("mainApp", []);
mainApp.controller("shapeController", function($scope) {
$scope.message = function(Fname,Lname)
{
return Fname + Lname;
}
});
</script>
```

Why AngularJS Scope is used?

To define member variables, AngularJS scope is used within a controller. It can contain variables that are the application data and methods.

Syntax-

```
<script>
var mainApp = angular.module("mainApp", []);
mainApp.controller("shapeController", function($scope) {
$scope.message = "Property is defined here";
```

```
});
</script>
```

In the above code “message ” is the member variable and its value is defined using $scope object. So wherever we display a message in view part its value will now get displayed as “property is defined here”.

- It is used to transfer data from view to controller as well as to transfer data from a controller to view.
- It acts as a linking between view and controller.
- Functions can be defined using a $scope.

Do you know How to implement View in AngularJS?

```
<script>
var mainApp = angular.module("mainApp", []);
mainApp.controller("shapeController", function($scope) {
$scope.message = function(Fname,Lname)
{
return Fname + Lname;
}
});
</script>
```

$rootscope

An angular application contains a single $rootscope. All other $scope present in an angular application are the child objects. The $rootscope object is the parent of all other $scope object.

The behavior attached to $rootscope is available to all the controllers present in that particular angular application. It means the properties and methods attached to it can be accessed by all controllers.

NOTE- The difference between $scope object and $rootscope object is that behavior attached with $rootscope is available for all the controllers while behavior attached with $scope is attached with a specific controller in which it is defined.

```
<!DOCTYPE html>
<html>
<head>
<title>AngualrJS Controller</title>
<script src="~/Scripts/angular.js"></script>
</head>
<body ng-app="myNgApp">
<div ng-controller="parentController">
Controller Name: {{controllerName}} <br />
Message: {{msg}} <br />
<div style="margin:10px 0 10px 20px;" ng- controller="childController">
Controller Name: {{controllerName}} <br />
Message: {{msg}} <br />
</div>
</div>
<div ng-controller="siblingController">
Controller Name: {{controllerName}} <br />
Message: {{msg}} <br />
</div>
```

```
<script>
var ngApp = angular.module('myNgApp', []);
ngApp.controller('parentController', function ($scope, $rootScope) {
$scope.controllerName = "parentController";
$rootScope.msg = "Common property of all controller ";
});
ngApp.controller('childController', function ($scope) {
$scope.controllerName = "childController";
});
ngApp.controller('siblingController', function ($scope) {
$scope.controllerName = "siblingController";
});
</script>
</body>
</html>
```

Output-

```
Controller Name: parentController
Message: This property is same for all controller.
Controller Name: childController
Message: This property is same for all controller.
Controller Name: siblingController
Message: Common property of all controller.
```

Explore the Dependency Injection in AngularJS

Explanation-

In the above code the property message is defined in parent controller only but since it is attached with $rootscope object, therefore, it remains the same for all the controller.

For Example-

In a piece of university information, the name of the student is a property that is distinct for every student so its behavior is attached with $scope object while the university name remains common for all the students so its behavior can be attached with a $rootscope object.

Various Methods of $Scope Object

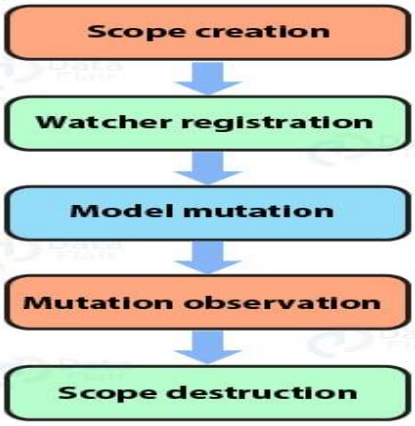
1. **$eval()** – The current scope expression is executed and the result is displayed.
2. **$apply() –** Using this method an expression can be executed in angular outside the angular framework.
3. **$new() –** A new child can be created using a $new method of $scope object.
4. **$on() –** A callback for an event is registered using $on a method of $scope object.
5. **$destroy() –** From parent scope, a current scope can be removed using $destroy method. As well as all its child scope is also removed using from parent scope.
6. **$emit() –** A particular event is dispatched upwards till $rootscope using $emit method.
7. **$broadcast() –** A particular event is dispatched downwards till $rootscope using a $broadcast method.

Life Cycle of AngularJS Scope

In the flow of **javascript code**, whenever an event is received by browser then it executes a javascript callback corresponding to it. And after the completion of callback, DOM objects are re-rendered by a

browser. If any javascript code is executed outside the context of angularJS by browser then angularJS remain unaware with the changes in the model. To detect the model change in angularJS $apply API is used.

1. **Creation –** During bootstrap of an application using $injector, root scope is created. And during the linking of a template, many of the directives creates new child scope.
2. **Watcher Registration –** Model values can propagate to DOM using a $watch
3. **Model Mutation –** To observe mutation in a proper way, an API known as $apply is used.
4. **Mutation Observation –** After $apply, a $digest is performed on the root scope in angular JS.
5. **Scope Destruction –** If child scope is no longer in use, then it should be destroyed. Child scope creator is responsible to destroy the child scope if no longer in use. Using API $scope.$destroy() child scope creator can do so. Destroying an unused child scope will release memory that is being used by it and stop model propagation too.



You must read – AngularJS Global API with types and examples

AngularJS Scope Characteristics

The various characteristics of the AngularJS scope are as follows:

- Context is provided by the scope for expression. For example – We are displaying {{name}} in view part so it will be displayed as it is to a user until a value is attached to it and this work is done by scope. Scope provides value to property by $scope.name = “daisy”.
- To observe model like a $watch, APIs are provided by scope.
- Any model changes that is outside of “angular realm” can be propagated through the API provided by $scope.

AngularJS Scope as Data Model

AngularJS Scope is a link between the controller and view part. It is because values are attached with attributes through a scope in the controller part and it gets displayed in view part.

```
angular.module('scopeApp', [])
.controller('ScopeController', ['$scope', function($scope) {
$scope.tutorialname = 'angularJs';}]);
```

$WATCH

A $watch is an expression that notifies about any change in the property. So that an updated value gets render to DOM.

Recommended Reading – DOM in AngularJS

Scope Hierarchies

Every angularJS application can have only one root scope as it is the parent scope. But it can have more than one child scope. New scopes can be created by directives which are added to parent scope as child scope. This is how a hierarchical structure such as a tree of scope is created.

How to Retrieve AngularJS Scope from the Directives

As a $scope data property, scopes are attached with DOM elements. Location of ng-app directive defines the location where root scope is attached to DOM. Generally, we put ng-app in the root element but if we want to control a particular part of our application we can also provide ng-app directive for that particular part. But it is declared only once in an angular application.

How to identify scope in a debugger

- Select the element of your choice then right click on it you will get an option of 'inspect element' select it. You can now see a browser debugger along with a highlighted element that is the element selected by you for inspection.
- The currently selected element can be accessed as $0 variable in a console, it is allowed by the debugger.
- You have to execute angular.element($0).scope() to retrieve the associated scope in a console.

To make the scope function available $compileProvider.debugInfoEnabled() should be set true. And it is already set as a true value by default.

Scope Event Propagation

Like the DOM elements, **AngularJS events** can be propagated by scope too. Events can either be broadcasted to the child scope or emitted to the parent scope

- $EMIT

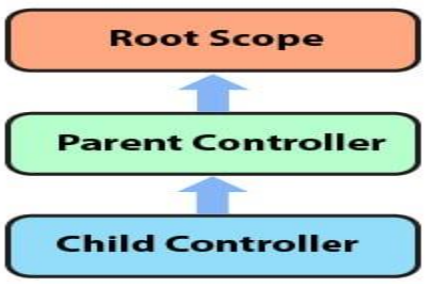
It is a function used to propagate events. $EMIT propagates its upper side of the hierarchy of scope. It means if the event is in child scope it will get propagates at upwards means in the parent scope.

- $BROADCAST

It is a function used to propagate events. $BROADCAST propagates it down the side of the hierarchy of scope. It means if the event is in child scope it will get propagates at downwards means in the second child scope.

AngularJS Scope Inheritance

Scope is basically controller specific. Whenever we nest a controller, that means to declare a controller inside a controller then the child controller inherits the scope of the parent controller.




```
<!Doctype html>
```

```
<html>
<head>
<title>Angular JS Forms</title>
</head>
<body>
<div ng-app = "studentApp" ng-controller = "studentOne">
<p>{{name}} <br/> {{grade}} </p>
<div ng-controller = "studentTwo">
<p>{{name}} <br/> {{grade}} </p>
</div>
<div ng-controller = "studentThree">
<p>{{name}} <br/> {{grade}} </p>
</div>
</div>
<script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
</script>
<script>
var t = angular.module("studentApp", []);
t.controller("studentOne", function($scope) {
$scope.name = "John";
$scope.grade = "A";
});
t.controller("studentTwo", function($scope) {
$scope.name = "jay";
});
t.controller("studentThree", function($scope) {
$scope.name = "diva";
$scope.grade = "B";
});
</script>
</body>
</html>
```

Output:

John

A

jay

A

diva

B

Summary

Therefore we can conclude that $scope is a built-in object that is used to define (assign value) member variable. As well as it can also be used to change the value of the member variable. A number of $scope object can be used inside a controller and it remains distinct for every controller.

Types of AngularJS Services with Examples – Creating/Registering Services

We offer you a brighter future with FREE online courses Start Now!!

Earlier we talked about **Filters in AngularJS**. In this article, we will discuss what is AngularJS Services with its types: built-in and custom services. Along with this, we will learn how to create and register services in AngularJS with an example.

AngularJs service is a function, which can use for the business layer of an application.

What is AngularJS Services?

AngularJs service is a function, which can use for the business layer of an application. It is like a constructor function that will invoke only once at runtime with new. Services in AngularJS are stateless and singleton objects because we get only one object in it regardless of which application interface created it.

We can use it to provide functionality in our web application. Each service performs a specific task.

When to use Services in AngularJS?

We can use AngularJS services when we want to create things that act as an application interface. It can be used for all those purposes for which constructor is used.

Types of AngularJS Services

There are two types of services in angular:

1. **Built-in services** – There are approximately 30 built-in services in angular.
2. **Custom services –** In angular if the user wants to create its own service he/she can do so.

Do you know What are AngularJS Events and list of HTML Event Directives?

Built-in Services in AngularJS

They are pre-built services in AngularJS. These services get registered automatically at runtime with the dependency injector. Therefore, by using dependency injector we can easily incorporate these built-in services in our angular application.

The various built-in services are as follows:

i. $http

It is a service to communicate with a remote server. It makes an ajax call to the server.

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>
<div ng-app="demoApp" ng-controller="myController">
<p>Today's welcome message is:</p>
<h1>{{message}}</h1>
</div>
<p>A page is requested on server by $http, and the response is set as the value of the "message" variable.</p>
<script>
var httpmodule = angular.module('demoApp', []);
```

```
httpmodule.controller('myController', function($scope, $http) {
$http.get("demo.htm").then(function (response) {
$scope.message = response.data;
});
});
</script>
</body>
</html>
```

Output:

Message is:

Hello AngularJS Students

A page is requested on a server by $http, and the response is set as the value of the “message” variable.

ii. $interval

It is a wrapper in angular for window.setInterval.

```
<!DOCTYPE html>
<html ng-app="intrApp">
<head>
<meta charset="utf-8" />
<title>$interval angularjs</title>
<script src="https://code.angularjs.org/1.3.15/angular.js">
</script>
<script src="script.js"></script>
</head>
<body ng-controller="IntervalController">
<br /><br />The value of counter is: {{cnt}}
</body>
<script>
var sample = angular.module("intrApp", []);
sample.controller("IntervalController", function($scope, $interval) {
$scope.cnt = 0;
$scope.increment = function() {
$scope.cnt += 1;
};
$interval(function() {
$scope.increment();
}, 2000);
});
</script>
</html>
```

Output:

The value of the counter is: 0
But the value gets increased by 1 in every 2milisecond.

iii. $timeout

It is the same as setTimeout function in javascript. To set a time delay on the execution of a function $timeout is used.

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
<script src="//code.angularjs.org/snapshot/angular.min.js"></script>
```

```
<script src="script.js"></script>
</head>
<body ng-controller="timeController">
<div>
<div>Message is displayed afer 3 seconds : {{msg}}</div>
</div>
<script>
var sample = angular.module('myApp', []);
sample.controller('timeController', function($scope, $timeout) {
$timeout( function(){
$scope.msg = "Hey!";
}, 3000 );
});
</script>
</body>
</html>
```

Output:

The message is displayed after 3 seconds: Hey!

iv. $anchorscroll

The page which specifies by an anchor in $location.hash() scrolls using $anchorscroll.

v. $animate

It consists of many DOM (Document Object Model) utility methods that provide support for animation hooks.

Also, see Major Directives for AngularJS HTML DOM

vi. $animateCss

It will perform animation only when ngAnimate includes, by default.

vii. $compile

We can compile HTML string or DOM in the template by it. Also, it produces a template function which will use to link template and scope together.

viii. $controller

By using a $controller, we can instantiate Angular controller components.

ix. $document

J-query wrapped the reference to the window. Document element is specified by it.

x. $locale

For various angular components, localization rules can be provided using $locale.

xi. $location

URL in the address bar of a browser is parsed by it and then the URL is made available to your application. Changes to the URL in the address bar is reflected in $location service and vice-versa.

xii. $q

A function can run asynchronously using $q and its return value, which will use when they finish the processing.

xiii. $rootElement

It is a root element of the angular application.

xiv. $rootScope

Use in an angular application.

Customs Services in AngularJS

We can create our own service by connecting it with a **module in AngularJS**. And to use it add it as a dependency while defining controller.

Here, in the below code we created a custom service demo.

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="myCtrl">
<p>The octal value of 225 is:</p>
</div>
<p>A custom service with a method that converts a given number into a octal number.</p>
<script>
var app = angular.module('myApp', []);
app.service('demo', function() {
this.myFunc = function (x) {
return x.toString(8);
}
});
app.controller('myCtrl', function($scope, demo) {
$scope.hex = demo.myFunc(225);
});
</script>
</body>
</html>
```

Output:

The octal value of 225 is:

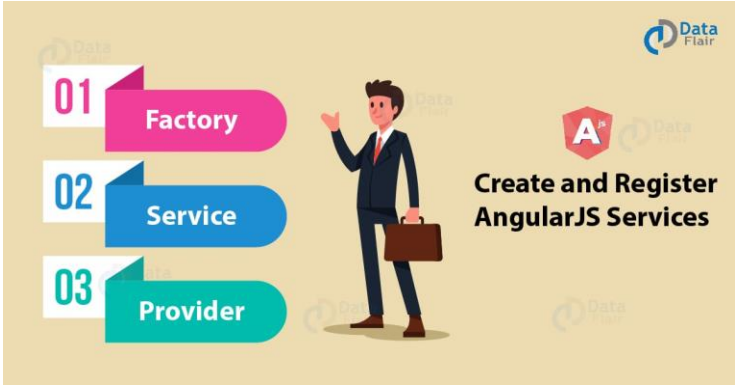
377

A custom service with a method that converts a given number into an octal number.

How to Create and Register AngularJS Services?

We can create angular services by registering it into a module. The module operates services.

In angular JS, there are **three ways to create service in AngularJS.**



1. Factory

Using Modules factory API, we can create service. We can use a factory method to create an object, properties add to it and return the same object.

Have a Look at 8 Types of Global API's with examples

Syntax:

```
var sample = angular.module("app", []);
sample.factory('factoryName',function(){
return factoryObj;
});
```

2. Service

We can use a new keyword to instantiate it, an instance of the function will pass by service. The instance of an object is the service object. Angular JS registers service object. Here, we can inject this object into various other components of angular application such as controller, directives, services, filters etc.

Syntax:

```
var sample = angular.module("app", []);
sample.service('serviceName',function(){ });
```

3. Provider

We can pass only $provider into the .config function. It uses $get function to return value. It will use before making a service available you want to provide module –wise configuration for the service object.

Syntax:

```
var sample = angular.module("app", []);
// syntax to create a provider
sample.provider('providerName',function(){ });
//syntax to configure a provider
sample.config(function(providerNameProvider){});
```

AngularJS Service v/s Factory

Service and factory both are singletons.

AngularJS .Service is like a constructor while .factory is a method that returns a value.

AngularJS .factory() provides us much more power and flexibility, whereas a .service() is the “end result” of a .factory() call.

Angular Services Example

```
<!Doctype HTML>
<html>
<head>
<title>AngularJS Services Tutorial</title>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.12/angular.min.js"></script>
<script src="main.js"></script>
</head>
<body>
<div>
<div ng-app="mainApp" ng-controller="myController">
<p><b>Message From Service: </b>{{serviceMsg}}</p>
<p><b>Message From Factory: </b>{{factoryMsg}}</p>
<p><b>Message From Provider:</b>{{providerMsg}}</p>
</div>
</div>
<script>
var sample = angular.module('mainApp', []);
//define a service named myService
sample.service('myService', function () {
this.message = '';
```

```
this.setMessage = function (newMsg) {
this.msg = newMsg;
return this.msg;
};
});
//define factory named 'myFactory'
sample.factory('myFactory', function () {
var obj = {};
obj.message = '';
obj.setMessage = function (newMsg) {
obj.message = newMsg;
};
return obj;
});
//Defining a provider 'configurable'
sample.provider('configurable', function () {
var returnMessage = '';
this.setMessage = function (newMsg) {
returnMessage = newMsg;
};
this.$get = function () {
return {
message: returnMessage
};
};
});
//configuring provider
sample.config(function (configurableProvider) {
configurableProvider.setMessage("Hello, This is Provider here!");
});
//defining controller
sample.controller('myController', function ($scope, myService, myFactory,
configurable) {
$scope.serviceMsg = myService.setMessage("Hello,This is Service here!");
myFactory.setMessage("Hello,This is Factory here! ");
$scope.factoryMsg = myFactory.message;
$scope.providerMsg= configurable.message;
});
</script>
</body>
</html>
```

Output:

A message From Service: Hello, This is Service here!

Message From Factory: Hello, This is Factory here!

Message From Provider: Hello, This is Provider here!

Dependencies in Servies

Like you declare dependencies in a controller, similarly, you can declare dependencies in AngularJS services too. We can declare the dependency in service by specifying them in a factory function signature of service.

```
var batch = angular.module('batchModule', []);
batch.factory('batchLog', ['$interval', '$log', function($interval, $log) {
```

```
var msgQueue = [];
function log() {
if (msgQueue.length) {
$log.log('batchLog messages: ', msgQueue);
msgQueue = [];
}
}
$interval(log, 50000);
return function(message) {
msgQueue.push(message);
}
}]);
batch.factory('routeTemplateMonitor', ['$route', 'batchLog', '$rootScope',
function($route, batchLog, $rootScope) {
return {
startMonitoring: function() {
$rootScope.$on('$routeChangeSuccess', function() {
batchLog($route.current ? $route.current.template : null);
});
}
};
}]);
```