Angular JS

Dependency Injection is a software design pattern in which components are given their dependencies instead of hard coding them within the component.

**Application Module**

This Module is used to initialize an application with controller(s)

**Steps to create AngularJS Application:**

**Step 1 − Load framework**

Being a pure JavaScript framework, It can be added using <Script> tag.

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

**Step 2 − Define AngularJS Application using ng-app directive – This is one of the main DIRECTIVE in Angular JS**

<div ng-app = ""> ... </div>

**Step 3 − Define a model name using ng-model directive**

<p>Enter your Name: <input type = "text" ng-model = "name"></p>

**Step 4 − Bind the value of above model defined using ng-bind directive.**

<p>Hello <span ng-bind = "name"></span> !

**AngularJS DIRECTIVES:−**

AngularJS directives are used to extend HTML. These are special attributes starting with ng- prefix.

**ng-app** − This directive starts an AngularJS Application and defines  the root element and links the application to HTML. It automatically initializes or bootstraps the application when web page containing AngularJS Application is loaded. It is also used to load various AngularJS modules in AngularJS Application.

**ng-init −** This directive initializes application data and also used to put values to the variables to be used in the application

**ng-model −** This directive defines the variable(or model) to be used in AngularJS and binds the values of the application data to HTML input controls.

**ng-bind − This directive binds the AngularJS Application data to HTML tags.**

**ng-repeat − This directive repeats html elements for each item in a collection.**

**Include AngularJS:-** Need to Include AngularJS JavaScript file in the HTML page

<head>

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

</head>

**Point to AngularJS app:**

**NG-APP**

AngularJS app is done by adding the attribute to the root HTML element. You can either add it to html element or body:

<body ng-app = "myapp"> … </body>

**NG-INIT**

<div ng-app = "" ng-init = "countries = [{locale:'en-US',name:'United States'}, {locale:'en-GB',name:'United Kingdom'}, {locale:'en-FR',name:'France'}]">

...

</div>

**NG-MODEL**

<div ng-app = "">

...

<p>Enter your Name: <input type = "text" ng-model = "name"></p>

</div>

**NG-REPEAT**

<div ng-app = "">

...

<p>List of Countries with locale:</p>

<ol>

<li ng-repeat = "country in countries">

{{ 'Country: ' + country.name + ', Locale: ' + country.locale }}

</li>

</ol>

</div>

**Controller Module:**

**This Module is used to define the controller.**

<div ng-controller = "HelloController" >

<h2>Welcome {{helloTo.title}}</h2>

</div>

ng-controller tells AngularJS what controls to be used with this view. helloTo.title tells AngularJS to write the "model" value named helloTo.title to the HTML at this location.

**Registering a Controller part:**

<script>

angular.module("myapp", [])

.controller("HelloController", function($scope) {

$scope.helloTo = {};

$scope.helloTo.title = "AngularJS";

});

</script>

This code registers a controller function named HelloController in the angular module named myapp. We will study more about modules and controllers in their respective chapters. The controller function is registered in angular via the angular.module(...).controller(...) function call.

The $scope parameter passed to the controller function is the model. The controller function adds a helloTo JavaScript object, and in that object it adds a title field.

**AngularJS Components - The framework can be divided into following three major parts**

ng-app − This directive defines and links an AngularJS application to HTML.

ng-model − This directive binds the values of AngularJS application data to HTML input controls.

ng-bind − This directive binds the AngularJS Application data to HTML tags.

**AngularJS Forms**

**Events in Angular JS**

AngularJS provides more events for associating with HTML controls.

**Following are the events in Angular JS.**

ng-click - ng-click is normally associated with button.

ng-dbl-click

ng-mousedown

ng-mouseup

ng-mouseenter

ng-mouseleave

ng-mousemove

ng-mouseover

ng-keydown

ng-keyup

ng-keypress

ng-change

**ng-click event / directive:**

For example: Reset data of a form using on-click directive of a button.

<input name = "firstname" type = "text" ng-model = "firstName" required>

<input name = "lastname" type = "text" ng-model = "lastName" required>

<input name = "email" type = "email" ng-model = "email" required>

<button ng-click = "reset()">Reset</button>

<script>

function empController($scope) {

$scope.reset = function(){

$scope.firstName = "aarif";

$scope.lastName = "mohammad";

$scope.email ="aarif.mohammad@infovisionlabs.com";

}

$scope.reset();

}

</script>

**Validate data:**   
Examples for tracking error.

$dirty − states that value has been changed.  
$invalid − states that value entered is invalid.  
$error − states the exact error.

<td>

<input name = "firstname" type = "text" ng-model = "firstName" required>

<span style = "color:red" ng-show = "empForm.firstname.$dirty && empForm.firstname.$invalid">

<span ng-show = "emp.firstname.$error.required">First Name is required.</span>

</span>

</td>

**Following directives can be used to bind application data to attributes of HTML DOM Elements**

|  |  |  |
| --- | --- | --- |
| S.No | Name | Description |
| 1 | ng-disabled | disables a given control. |
| 2 | ng-show | shows a given control. |
| 3 | ng-hide | hides a given control. |
| 4 | ng-click | represents a AngularJS click event. |

**ng-disabled directive**

Add ng-disabled attribute to a HTML button and pass it a model. Bind the model to a checkbox:

<input type = "checkbox" ng-model = "enableDisableButton">Disable Button

<button ng-disabled = "enableDisableButton">Click Me!</button>

**ng-show directive**

Add ng-show attribute to a HTML button and pass it a model. Bind the model to a checkbox

<input type = "checkbox" ng-model = "showHide1">Show Button

<button ng-show = "showHide1">Click Me!</button>

**ng-hide directive**

Add ng-hide attribute to a HTML button and pass it a model. Bind the model to a checkbox

<input type = "checkbox" ng-model = "showHide2">Hide Button

<button ng-hide = "showHide2">Click Me!</button>

**ng-click directive**

Add ng-click attribute to a HTML button and update a model. Bind the model to html

<p>Total click: {{ clickCounter }}</p>

<button ng-click = "clickCounter = clickCounter + 1">Click Me!</button>

**Using AngularJS, we can embedded HTML pages within a HTML page using ng-include directive.**

<div ng-app = "" ng-controller = “empController">

<div ng-include = "'main.htm'"></div>

<div ng-include = "'department.htm'"></div>

</div>

**Auto Completer:**

The Auto Completer provides the front-end logic for text-entry suggestion and completion functionality. Auto completion can be done using one of the feature named **DOJO**

**The Dojo Toolkit is a full featured JavaScript library. It gives you complete DOM manipulation tools, AJAX, UI components, graphing and charting, mobile components, etc.**

The Combo Box is a hybrid between the combo box and text field. It provides a list of acceptable values. This is good for open-ended multiple choice questions. Rather than having two fields - a combo box and an Other text box - you can use just one field.

<html> <head> <title>Auto Completer Combo</title>

<style type="text/css">

@import "../resources/dojo.css";

@import "../dijit/themes/tundra/tundra.css";

</style>

<script type="text/javascript" src="dojo.xd.js" djConfig="parseOnLoad: true"/>

<!-- combo box -->

<script type="text/javascript">

dojo.require("dojo.parser");

dojo.require("dijit.form.FilteringSelect");

</script> </head>

<body class="tundra">

<h2>Auto Completer Combo box</h2>

<select dojoType="dijit.form.FilteringSelect" name="sname"

autocomplete="false" value="Vinod">

<option value="Vinod">Vinod</option>

<option value="Vikash" >Vikash</option>

<option value="Deepak" >Deepak</option>

<option value="DeepakSir" >Aarif</option>

<option value="Arun" >Arun</option>

<option value="Amar" >Amar</option><

<option value="Aman" >Aman</option>

</select> </body> </html>

**Scope Object**:

Scope is a special javascript object which plays the role of joining controller with the views. Scope contains the model data, inside controllers, model data is accessed via $scope object.

$scope.message and $scope.type are the models which are to be used in the HTML page.

**Scope Inheritance** - Scope are controllers specific. If we defines nested controllers then child controller will inherit the scope of its parent controller.

<script>

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

mainApp.controller("shapeController", function($scope) {

$scope.message = "In shape controller";

$scope.type = "Shape";

});

mainApp.controller("circleController", function($scope) {

$scope.message = "In circle controller";

});

</script>

**Points to be considered in above example:**

1. Setting values to models in shapeController.
2. Overridden message in child controller circleController. When "message" is used within module of controller circleController, the overridden message will be used.

**Dependency Injection:**

AngularJS provides a great Dependency Injection mechanism.

It provides these core components which can be injected into each other as dependencies :

Value

Factory

Service

Provider

Constant

**Value**

Value is a simple javascript object, used to pass values to controller during config phase.

//define a module

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

//create a value object as "defaultInput" and pass it a data.

mainApp.value("defaultInput", 5);

...

//inject the value in the controller using its name "defaultInput"

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

$scope.number = defaultInput;

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

$scope.square = function() {

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

}

});

**Factory**

Factory is a function used to return a value. It creates value on demand whenever a service or controller requires. It normally uses a factory function to calculate and return the value.

//define a module

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

//create a factory "MathService" which provides a method multiply to return multiplication of two numbers

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

var factory = {};

factory.multiply = function(a, b) {

return a \* b

}

return factory;

});

//inject the factory "MathService" in a service to utilize the multiply method of factory.

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

this.square = function(a) {

return MathService.multiply(a,a);

}

});

**Service**

Service is a singleton javascript object containing a set of functions to perform certain tasks. Services are defined using service() functions and then injected into controllers.

//define a module

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

...

//create a service which defines a method square to return square of a number.

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

this.square = function(a) {

return MathService.multiply(a,a);

}

});

//inject the service "CalcService" into the controller

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

$scope.number = defaultInput;

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

$scope.square = function() {

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

}

});

**Provider**

Provider is used by AngularJS internally to create services, factory etc. During config phase (phase during which AngularJS bootstraps itself). Below mention script can be used to create MathService that we've created earlier. Provider is a special factory method with a method get() which is used to return the value/service/factory.

//define a module

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

...

//create a service using provider which defines a method square to return square of a number.

mainApp.config(function($provide) {

$provide.provider('MathService', function() {

this.$get = function() {

var factory = {};

factory.multiply = function(a, b) {

return a \* b;

}

return factory;

};

});

});

**Constant**

Constants are used to pass values at config phase considering the fact that value cannot be used to be passed during config phase.

mainApp.constant("configParam", "constant value");

**Expressions**

Expressions are used to bind application data to html. Expressions are written inside double braces like {{ expression}}. Expressions behaves in same way as ng-bind directives. AngularJS application expressions are pure javascript expressions and outputs the data where they are used.

Using numbers

<p>Expense on Books : {{cost \* quantity}} Rs</p>

Using strings

<p>Hello {{student.firstname + " " + student.lastname}}!</p>

Using object

<p>Roll No: {{student.rollno}}</p>

Using array

<p>Marks(Math): {{marks[3]}}</p>

**Filters**

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

Uppercase - converts a text to upper case text.

Lowercase - converts a text to lower case text.

Currency - formats text in a currency format.

Filter - filter the array to a subset of it based on provided criteria.

Orderby - orders the array based on provided criteria.

**Uppercase filter:**

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

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

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

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

**Lowercase filter**

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

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

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

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

**currency filter**

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

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

fees: {{student.fees | currency}}

**filter filter**

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

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

Subject:

<ul>

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

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

</li>

</ul>

**orderby filter**

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

Subject:

<ul>

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

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

</li>

</ul>