Feature :

Two Way data binding

MVC -

Seperation of concerns - code managability

Declartive Style (ng-repeat)

Dependency Injection

Angular Module :

Modules are AngularJS’s way of packaging relevant code under a single name. So A module is consists of collection of services, directives, controllers, filters, and configuration information.The module can also depend on other modules as dependencies, which are defined when the module is instantiated.

Angular Controller :

Controllers JavaScript functions that perform or trigger the majority of our UI-oriented work.

• Fetching the right data from the server for the current UI

• Deciding which parts of that data to show to the user

• Presentation logic, such as how to display elements, which parts of the UI to show, how to style them, etc.

• User interactions, such as what happens when a user clicks something

Angular invoke the controller with $scope object.

Can be nested.

Types : regular directive controller - <div ng-controller="MaintCtrl as ctrl"></div>

component controller - <app

Service :

Service are lazily instantiated singleton constructor functions - used for performing business logic or data manipulation or make a call to API or CDN. Services are used for sharing the state of application between controllers.

Independent of views

Responsible for making server calls, common validation logic, application-level stores, and reusable

business logic

service factory function generates the single object or function that represents the service to the rest of the application

angular.module('someApp',[]).factory('ItemService', [function () {

var items = [{ id: 1, label: 'Item 0' }];

return {

add: function (item) {

items.push(item);

}

};

}]);

angular.module('myModule', []).config(['$provide', function($provide) {

$provide.factory('serviceId', function() {

var shinyNewServiceInstance;

// factory function body that constructs shinyNewServiceInstance

return shinyNewServiceInstance;

});

}]);

Built in Service :

$window - wrapper for the browser's window.document

$location -

$http - service to facilitate theh HTTP communication between code & API or CDN.

$http.get('/someUrl', config).then(successCallback, errorCallback);

$http.post('/someUrl', data, config).then(successCallback, errorCallback);

Config : {

method: string,

url: string,

params: object,

data: string or object,

headers: object,

xsrfHeaderName: string,

xsrfCookieName: string,

transformRequest: function transform(data, headersGetter) or

an array of functions,

transformResponse: function transform(data, headersGetter) or

an array of functions,

cache: boolean or Cache object,

timeout: number,

withCredentials: boolean

}

.config(['$httpProvider', function($httpProvider) {

$httpProvider.defaults.transformRequest.push(function(data) {

return transfromedString;

});

$httpProvider.defaults.headers.post['Content-Type'] = 'application/x-www-form-urlencoded';

}]);

Interceptor :

factory('MyLoggingInterceptor', ['$q', function($q) {

return {

request: function(config) {

console.log('Request made with ', config);

return config;

},

requestError: function(rejection) {

console.log('Request error due to ', rejection);

return $q.reject(rejection);

},

response: function(response) {

console.log('Response from server', response);

return response || $q.when(response);

},

responseError: function(rejection) {

console.log('Error in response ', rejection);

return $q.reject(rejection);

}

}

});

.config(['$httpProvider', function($httpProvider) {

$httpProvider.interceptors.push('MyLoggingInterceptor');

}]);

$q - service that helps you run functions asynchronously, and use their return values (or exceptions) when they are done processing

$locale -

$interval - AngularJS's wrapper for window.setInterval

$timeout - AngularJS's wrapper for window.setTimeout

$log

$filter - Filters are used for formatting data displayed to the user

$parse

$document - wrapper for the browser's window.document

$exceptionHandler - Any uncaught exception in AngularJS expressions is delegated to this service. The default implementation simply delegates to $log.error

$scope - They also provide event emission/broadcast and subscription facility.

Provider : The provider() function allows us to create a configurable service.the whole point of creating a service with Provider is to be able to alter some

variables through the app.config function before the final object is passed to the rest of the application. the provider function is set on

app.provider('customService',[$http,$q, function($htp,$q){

this.$get = function(){

return {}; // service object similar to factory function.

};

}]);

app.config(['customServiceProvider',function(customServiceProvider){

customServiceProvider.XXX = "";

}]);

Provider does not return anything but set the service function to the scope of the module by this.$get

Factories : A factory is a simple function which allows you to add some logic before creating the service object. the object returned by factory method is singleton.

app.factory('myFactory', function ($http, $q) {

var service = {};

var baseUrl = 'https://itunes.apple.com/search?term=';

var \_artist = '';

var \_finalUrl = '';

var makeUrl = function () {

\_artist = \_artist.split(' ').join('+');

\_finalUrl = baseUrl + \_artist + '&callback=JSON\_CALLBACK';

return \_finalUrl;

}

return service;

});

DECORATORS : Decorators are a design pattern that is used to separate modification or decoration of a class without modifying the original source code. In AngularJS,

decorators are functions that allow a service, directive or filter to be modified prior to its usage.

There are two ways to register decorators

$provide.decorator, and

module.decorator

angular.module('myApp', [])

.config([ '$provide', function($provide) {

$provide.decorator('myService', [

'$delegate',

function myServiceDecorator($delegate) {

function helperFn() {

}

$delegate.aHelpfulAddition = helperFn;

return $delegate;

}

]);

}]);

AngularJS Directive :

: ng-app - auto-bootstrap an AngularJS application.

: ng-controller - ngController directive attaches a controller class to the view. its enable us to achieve MVC pattern. it will cause controller function to execute.

ng-controller="SettingsController2"

ng-controller="SettingsController1 as settings"

: ng-bind - ngBind attribute tells AngularJS to replace the text content of the specified HTML element with the value of a given expression

ng-bind-html="expression"

: ng-click - ngClick directive allows you to specify custom behavior when an element is clicked

: ng-repeat

ng-repeat="note in ctrl.notes1"

ng-repeat="note in ctrl.notes2 track by note.id">

ngRepeat directive instantiates a template once per item from a collection. the keys of the object will be sorted in a case-sensitive,

alphabetic order.

Helper Variables in ng-repeat

{{$first}}

{{$last}}

{{$middle}}

{{$odd}}

{{$even}}

{{$index}}

repeat a series of elements

ng-repeat-start

ng-repeat-end

: ng-cloak - used to hide the uncompiled template from being displayed to user while page lodaing.

: ng-show

ng-show="myForm.uname.$error.required"

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

: ng-class - ngClass directive allows you to dynamically set CSS classes on an HTML element by databinding an expression

: ng-model - ngModel directive binds an input,select, textarea (or custom form control) to a property on the scope using

Binding the view into the model,

Providing validation behavior

Keeping the state of the control

Setting related css classes on the element

ngModel watches the model by reference, not value. So when Value are chaged, ngModel will not be notified and so the input will not be re-rendered.

<input type="text" name="userName" ng-model="user.name" ng-model-options="{ getterSetter: true }" />

: ng-include -

:

: ng-submit -

: ng-switch - ngSwitch directive is used to conditionally swap DOM structure on your template based on a scope expression.

<div ng-switch on="mainCtrl.tab">

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

<div ng-switch-when="second">.

Validation

: required

: ng-required

: ng-minlength

: ng-maxlength

: ng-pattern

: type="email"

: type="number"

: type="date"

: type="url"

: ng-disabled

$invalid

$valid

$pristine

$dirty

$error

---

Services :

build in angular service

1. $log

2. $window

3. $location

5. $http

6. $q

Custom Services

1. angular.module('',[]).factory('myService',[function(){}]);

2. function Item(){}

angular.module('',[]).factory('myService',ItemService);

3.

angular.module('',[])..provider('ItemService', function() {

this.$get =[function(){

return new ItemService(optItems);

}]

}).config(['ItemServiceProvider', function(ItemServiceProvider){

}])

AngularJS Filters: Filters format the value of an expression for display to the user. They can be used in view templates, controllers or services

{{ expression | filter1 | filter2 | ... }}

{{greeting|reverse:true | jumbledFilter:true}}

Built in filter :

{{ctrl.amount | number}}

{{ctrl.totalCost | currency}}

{{ctrl.totalCost | currency:'INR '}}

{{ctrl.name | uppercase}}

{{ctrl.name | lowercase}}

{{ctrl.startTime | date:'medium'}}

{{ctrl.startTime | date:'M/dd, yyyy'}}

{{expression | filter1 | filter2}}

{{ctrl.name | lowercase | limitTo:5}}

<li ng-repeat="note in ctrl.notes | filter:ctrl.filterOptions[ctrl.currentFilter] | orderBy:ctrl.sortOrder |

limitTo:5">

{{note.label}} - {{note.type}} - {{note.done}}

.controller('FilterCtrl', [function() {}]);

currency

date

filter

json

limitTo

lowercase

number

orderBy

uppercase

Route : Mapping URL to Component & template.

angular.module('routingApp', ['ngRoute'])

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

$routeProvider.when(url, {

template: string,

templateUrl: string,

controller: string, function or array,

controllerAs: string,

resolve: object<key, function>

}).otherwise({redirectTo: '/'});;

}]);

Custom Directive :

angular.module('stockMarketApp')

.directive('stockWidget', [function() {

return {

templateUrl: 'stock.html',

restrict: 'A',

scope: {

stockData: '=' or '<', '@'

},

link : function(){

}

}

}

Transclusion : Transclusion provides a way to pass in templates to a directive and it is displayed.

unit test

\* karma - test runner framework

Karma is a tool that enables the running of source code (i.e. JavaScript) against real browsers via the CLI. The fact that it runs against real browsers.

karma.conf.js

module.exports = function (config) {

config.set({

// base path that will be used to resolve files and exclude

basePath: '',

// testing framework to use (jasmine/mocha/qunit/...)

frameworks: ['jasmine'],

// list of files / patterns to load in the browser

files: [

'node\_modules/angular/angular.min.js',

'node\_modules/angular/angular-monpmcks.js',

'controller.js',

'controllerSpec.js'

],

// list of files / patterns to exclude

exclude: [],

port: 8080,

// level of logging

// possible values: LOG\_DISABLE || LOG\_ERROR ||

// LOG\_WARN || LOG\_INFO || LOG\_DEBUG

logLevel: config.LOG\_INFO,

// enable / disable watching file and executing tests

// whenever any file changes

autoWatch: true,

browsers: ['Chrome'],

singleRun: false

});

};

\* jasmine - testing framework

describe - Controller ( describe can be nested)

steps :

beforeAll

beforearch

afterEach

afterAll

it

Jsmine Matcher

toEqual

toBe

toBeTruthy

toBeFalsy

toBeDefined

toBeUndefined

toBeNull

toContain

toMatch

it('should return 3 for 1 + 2', function() {

document.getElementById('x').value = 1;

document.getElementById('y').value = 2;

document.getElementById('add').click();

expect(document.getElementById('result').innerHTML).toBe('3');

});

Node JS :

var capitalize = require("../capitalize");

var chai = require("chai");

var expect = chai.expect;

describe("capitalize", function() {

it("capitalizes single words", function() {

expect(capitalize("express")).to.equal("Express");

expect(capitalize("cats")).to.equal("Cats");

});

});

HTML Forms :

<input type="checkbox" ng-model="ctrl.user.agree" ng-true-value="YES" ng-false-value="NO">

<div ng-init="otherGender = 'other'">

<input type="radio" name="gender" ng-model="user.gender" value="male">Male

<select ng-model="ctrl.selectedCountry" ng-options="c.label for c in ctrl.countries">

</select>

<select ng-model="ctrl.selectedCountryId" ng-options="c.id as c.label for c in ctrl.countries">

</select>

modelValue as labelValue for item in array

$q Service :

function asyncGreet(name) {

var deferred = $q.defer();

setTimeout(function() {

deferred.notify('About to greet ' + name + '.');

if (okToGreet(name)) {

deferred.resolve('Hello, ' + name + '!');

} else {

deferred.reject('Greeting ' + name + ' is not allowed.');

}

}, 1000);

return deferred.promise;

}

var promise = asyncGreet('Robin Hood');

promise.then(function(greeting) {

alert('Success: ' + greeting);

}, function(reason) {

alert('Failed: ' + reason);

}, function(update) {

alert('Got notification: ' + update);

});

Waity for promise to finish.

var allPromise = $q.all([async1(),async2(),asyncN()]);

allPromise.then(function(values) {

var value1 = values[0];

});

$q.all(promisesArr) returns a promise that is resolved only when all the promises in promisesArr are resolved.

$q.when :::

$q.when(getDemoInformation(), processDemoInformation(response));

function getDemoInformation() {

return $http.get('https://www.technouz.com');

}

function processDemoInformation(response) {

var temp = response.data;

}

$q.when(value, [successCallback], [errorCallback], [progressCallback]);

Wraps an object that might be a value or a (3rd party) then-able promise into a $q promise. This is useful when you are dealing with an object that might or might not be a promise, or if the promise comes from a source that can't be trusted.

Deffered Object :

A deferred represents the result of an asynchronic operation. It exposes an interface that can be used for signaling the state and the result of the operation it represents. It also provides a way to get the associated promise instance.

A promise provides an interface for interacting with it’s related deferred, and so, allows for interested parties to get access to the state and the result of the deferred operation.

When creating a deferred, it’s state is pending and it doesn’t have any result. When we resolve() or reject() the deferred, it changes it’s state to resolved or rejected.

Built in Services :

$broadcast(name, args);

Dispatches an event name downwards to all child scopes (and their children) notifying the registered $rootScope.Scope listeners.

$emit(name, args);

Dispatches an event name upwards through the scope hierarchy notifying the registered $rootScope.Scope listeners.

$watch(watchExpression, listener, [objectEquality]);

Registers a listener callback to be executed whenever the watchExpression changes.

The watchExpression is called on every call to $digest() and should return the value that will be watched. (watchExpression should not change its value when executed multiple times with the same input because it may be executed multiple times by $digest(). That is, watchExpression should be idempotent.)

The listener is called only when the value from the current watchExpression and the previous call to watchExpression are not equal (with the exception of the initial run, see below). Inequality is determined according to reference inequality, strict comparison via the !== Javascript operator, unless objectEquality == true (see next point)

When objectEquality == true, inequality of the watchExpression is determined according to the angular.equals function. To save the value of the object for later comparison, the angular.copy function is used. This therefore means that watching complex objects will have adverse memory and performance implications.

$exceptionHandler

Any uncaught exception in AngularJS expressions is delegated to this service. The default implementation simply delegates to $log.error which logs it into the browser console.

angular.

module('exceptionOverwrite', []).

factory('$exceptionHandler', ['$log', 'logErrorsToBackend', function($log, logErrorsToBackend) {

return function myExceptionHandler(exception, cause) {

logErrorsToBackend(exception, cause);

$log.warn(exception, cause);

};

}]);

$injector

injector is used to retrieve object instances as defined by provider, instantiate types, invoke methods, and load modules.

Module methods such as factory, service, directive, etc. register these items with the injector. When you inject something (e.g., a service into a controller), the injector will lookup and then instantiate the service (if it wasn't instantiated already -- if it was, it will return the already-instantiated object).