CitiusTech

Controllers

May 2015

Agenda

- **Controller Introduction**
- Controller & $scope
- Controller Inheritance
- Correct Usage Of Controller
- Minification Safe Dependency Injection
- AngularJS Lifecycle

Controller Introduction

- Controller is a **JavaScript** constructor **function**
- It is used to wire up model data and functions & assign them to the view via scope.
- Use controller to place the model data & business logic related to view

- $scope glues the View with the Controller
- $scope is a container for business data and functions (i.e. all model data)
- $scope can hold variables, functions, arrays, json any form of business data

- Basic Demo on Controller:
  http://ctgit/silviap/angularjsdemosandlabs/blob/master/Day1/day1_Demos/2_BasicController.html

Agenda

- Controller Introduction
- **Controller & $scope**
- Controller Inheritance
- Correct Usage Of Controller
- Minification Safe Dependency Injection
- AngularJS Lifecycle

Controller & $scope (1/2)

- $scope has the model data & its behaviour(functions) as shown below:

```
<div ng-controller="ContactController">
          Greet: <input type="text" ng-model="myname"/>
          <button ng-click="greet()">Greet</button>
           <h2>  {{ myname1 }}   </h2>
        </div>
          function ContactController($scope) {
            // myname,myname1 ,greet is model data and function assigned to $scope
       $scope.greet = function() {     $scope.myname1 = "hi"+$scope.myname;  }
    }
```

- **ng-model** – directive to indicate a model, available automatically in $scope
- **ng-controller –** directive is used to define controller to be bound with the view
- When controller is attached to a DOM element using 'ng-controller' directive, AngularJS will do the following:
  - Instantiate a new controller object using the controller's constructor function
  - Instantiate the $scope object and inject it inside the controller
  - Invoke the controller

Controller & $scope (2/2)

- The scope is used to expose the model to the view, but the scope is not the model. The model is the business data that is put into the scope
- {{ .. }} expression is used in View to access the model data assigned to $scope
- Every controller when instantiated get it's own $scope object (injected into it)
- Data is passed from view to $scope using ng-model directive
- Variables and functions not assigned to $scope are not accessible in View. They are for controller functions internal use

- Demo on Controller & Scope

  http://ctgit/silviap/angularjsdemosandlabs/blob/master/Day1/day1_Demos/3_ControllerAndScope.html

  http://ctgit/silviap/angularjsdemosandlabs/blob/master/Day1/day1_Demos/4_ControllerAndScopeWithArgument.html

Agenda

- Controller Introduction
- Controller & $scope
- **Controller Inheritance**
- Correct Usage Of Controller
- Minification Safe Dependency Injection
- AngularJS Lifecycle

Controller Inheritance

Reusability in AngularJS

- Reusability: Need to share common functionality across different parts of the application
- There are different approaches to maximize code reuse and clarity:
  - **Services**: Move functionality into services, where it can be appropriately factored
  - **Custom Directives**: Refactor repeated functionality into directives and aim for directive reuse
  - **Controller Inheritance**: Create an inheritance hierarchy of page controllers and place common functionality in ancestor controllers
- All of these are valid approaches and can be mixed and matched to suit the scenario.
- **Two Approaches to AngularJS Controller Inheritance**
  - Controller Inheritance by nesting of scopes
  - Controller Inheritance via $injector

Controller Inheritance by Nesting of Scopes

```
/*  The 'SpecialistDoctorController' inherits the scope of parent controller 'GenralDoctorController'
    Its done by nesting of controllers in view   */
<body ng-app>
<div ng-controller="GenralDoctorController">
   My name is {{ name }} and I am a {{ type }}
     <div ng-controller="SpecialistDoctorController">
      My name is {{ name }} and I am a {{ type }}
     </div>
</div>
 <script>
 function GenralDoctorController($scope) {
     $scope.name = 'Rahul';
     $scope.type = 'General Doctor'; }
 function SpecialistDoctorController($scope) {
    $scope.type = 'Specialist Doctor';
 }
</script>
</body>
```

Controller Inheritance via $injector

```
/*  The 'SpecialistDoctorController' inherits the scope of parent controller 'GenralDoctorController' .
                Its done via $injector service of Angular*/
<body ng-app>
<div ng-controller="SpecialistDoctorController">
   My name is {{ name }} and I am a {{ type }}
   <button ng-click="clickme()">Click Me</button>
 </div>
 <script>
function GeneralDoctorController($scope) {
   $scope.name = 'Namrata';
   $scope.type = 'Genral Docotor';
   $scope.clickme = function()  {
alert('This is parent controller "GeneralDoctorController" calling');     }
}
function SpecialistDoctorController($scope, $injector) {
    $injector.invoke(GeneralDoctorController, this, {$scope: $scope});
    $scope.type = 'Specialist Doctor';
}
</script>
</body>
```

*Note: refer service ppt for info on $injector

Controller Inheritance

- Demo Link on Controller Inheritance by Nesting Of Scopes

  The ‘SpecialistDoctorController’ inherits the scope of parent controller ‘GenralDoctorController’ . Its done by nesting of controllers in view .

  http://ctgit/silviap/angularjsdemosandlabs/blob/master/Day1/day1_Demos/5_NestedController.html

- Demo Link on Controller Inheritance via $injector

  The ‘SpecialistDoctorController’ inherits the scope of parent controller ‘GenralDoctorController’ . Its done via $injector service of Angular.

  http://ctgit/silviap/angularjsdemosandlabs/blob/master/Day1/day1_Demos/6_ControllerInheritance.html

Agenda

- Controller Introduction
- Controller & $scope
- Controller Inheritance
- **Correct Usage Of Controller**
- Minification Safe Dependency Injection
- AngularJS Lifecycle

Correct Usage of Controllers

- Key points on usage of Controllers:
  - Controller should not do too much, it should only contain the business logic used to represent single view
  - Need to keep controller code as small as possible
  - Refactor business logic into service instead of writing into controller & inject that service into the controller via dependency injection
- Do not use controllers to:
  - Manipulate DOM — Controllers should contain only business logic
- Angular has data binding for most cases and directives to encapsulate manual DOM manipulation
  - Format input — Use angular form controls instead
  - Filter output — Use angular filters instead
  - Share code or state across controllers — Use angular services instead
  - Manage the life-cycle of other components

Agenda

- Controller Introduction
- Controller & $scope
- Controller Inheritance
- Correct Usage Of Controller
- **Minification Safe Dependency Injection**
- AngularJS Lifecycle

Minification Safe Dependency Injection

- Angular infers the controller's dependencies from the names of arguments passed to controller's constructor function
- If you were to minify the JavaScript code for controller, all of its function arguments would be minified as well
- The dependency injector would not be able to identify dependant services correctly as names of dependencies would also get minified
- We can overcome this problem by annotating the function with the names of the dependencies, provided as strings in [] , which will not get minified

```
phonecatApp.controller('PhoneListCtrl', ['$scope', '$http', function($scope, $http)
                                                                { ... } ] );
```

- In above code:
  - When javascript code is minified, dependencies passed in function arguments get minified
  - dependency names passed as strings '$scope' and '$http' in [] bracket won't get minified
  - So, dependency injector would be able to identify dependant service correctly

Agenda

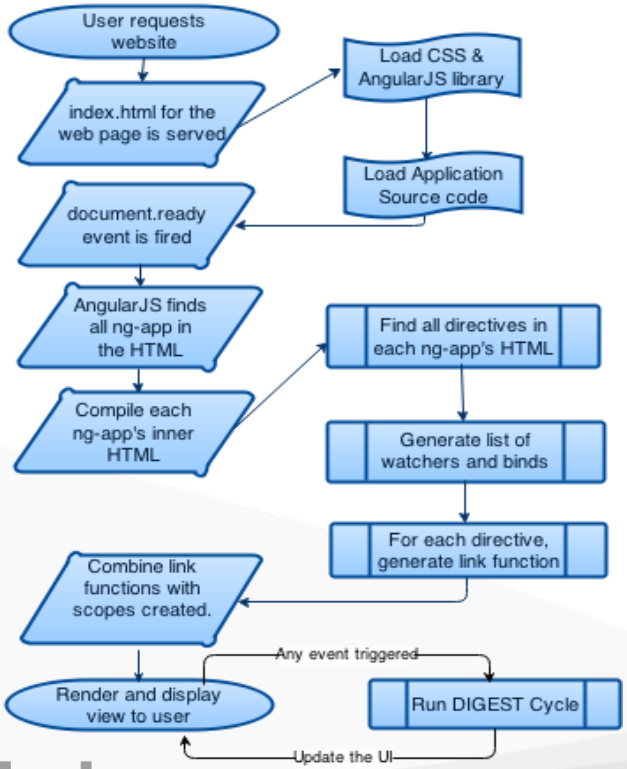
- Controller Introduction
- Controller & $scope
- Controller Inheritance
- Correct Usage Of Controller
- Minification Safe Dependency Injection
- **AngularJS Lifecycle**

AngularJS Lifecycle

- When an AngularJS application is loaded in browser window, the following events are executed in order:
- HTML page loaded.
  - Loads the AngularJS source code
  - Loads the application's JavaScript code
- When the document ready event is fired, AngularJS bootstraps and searches for any/all instances of the ng-app attribute in the HTML.
  - In case AngularJS is bootstrapped manually, then this needs to be triggered by the code we write.
- Within the context of each ng-app, AngularJS starts its magic by running the HTML content inside the ng-app through the **compile** step.
  - The compile step goes through each line of HTML and looks for AngularJS directives.
  - For each directive, it executes the necessary code as defined by that directive's definition object.
  - At the end of the compile step, a link function is generated for each directive .It has access to all elements and attributes in UI that need to be controlled by AngularJS.

AngularJS Lifecycle

- AngularJS takes the link function and combines it with a scope.
- AngularJS will take the variables in the scope, and display them in the UI if the HTML view refers to its associated controller function (ng-controller)
  - AngularJS adds **watchers** and **listeners** for all the directives and bindings, to ensure it knows which fields it needs to track , bind and update in UI
- At the end of this, we have a live, interactive view with the content filled in for the user



Controllers - Finding your way (1/2)

Technical Questions:

1. What is $injector?
2. What is $rootScope?

As you start to work with AngularJS Controllers, you will frequently encounter technical issues which are not covered by this training.

How will you resolve these technical issues on AngularJS Controllers?

Controllers - Finding your way (2/2)

Resources	Remarks
http://viralpatel.net/blogs/angularjs-controller-tutorial/	Good blog post on Angularjs controllers
https://docs.angularjs.org/guide/controller	AngularJS official doc site
http://tutorials.jenkov.com/angularjs/scope-hierarchy.html	Good blog on Controllers
CurioCT - https://interct/SitePages/CurioCT.aspx	
CTCourses – Controllers	In addition to updated course material, CTCourse contains reference sites (Library) and list of project teams with expertise on geolocation

CurioCT - CitiusTech's Technology Q & A Forum

- In case of any questions please log on to https://interct/SitePages/CurioCT.aspx

Thank You