AngularJS

Advanced Topics

Topics

- Scope & Controllers
- Services & DI
- Directives
- Filters

AngularJS

Scope & Controllers

Scope

- Scope is the context in which model data is stored
- View controls watch for changes in the scope model
- Model data are associated with the scope object as properties
- Scope can have methods defined
- A scope will be created and attached with a controller
- A rootScope will be created during the application bootstrap
- In case of nested controllers scopes will have parent child relation ship and children can access the data in the parent scope
- Let us see some samples…

Nested Controllers

- Html template

```
<div ng-controller="OuterController">
    Simple <input type="text" ng-model="name"> <br/>
    Object <input type="text" ng-model="user.name"> <br/>

    <div ng-controller="InnerController">
      Simple <input type="text" ng-model="name"><br/>
      Object <input type="text" ng-model="user.name"><br/>
      Parent <input type="text" ng-model="$parent.name">
             <br/>
    </div>

</div>
```

Nested Controllers

- The script

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

app.controller("OuterController", function($scope){
    $scope.name="Peter";
    $scope.user = {name : "John"};
});

app.controller("InnerController", function($scope){
    $scope.name="Jill"; //create a new name in scope
    $scope.user.name="Alpha"; //change outer object
});
```

Nested Controller

- Outer Scope
  - Has a name property
  - Has a user object
- Inner Scope
  - Has its own name property
  - Inherits the outer scope name object and changes the name - it reflects in both scope
  - Can access the outer scope's name property using $parent
- Demo
  - Run the app and see how the edit works
  - When the inner controller changes something it reflects on the outer controller

Root Scope

- In case of nested controllers the outer scope properties are available by default in inner scope
- Simple values can be accessed but not changed when assigned they create a new property
- Object values can be changed (same as ref Vs value)

- What if controllers are nested and want to share properties and methods?
- Use Root scope
- Root scope is available to all controllers and modules and can be injected the way scope is injected

Root Scope

- The page

```
<div ng-controller="ListController">
   <ul>
     <li ng-repeat="fruit in fruits">
        {{fruit}}
     </li>
   </ul>
</div>

<div ng-controller="AddController">
   <input type="text" ng-model="newFruit">
   <input type="button" value="Add" ng-click="addFruit()">
</div>
```

Root Scope

- The script

```
app.controller("ListController",
                function($scope, $rootScope){
  $rootScope.fruits = ["Apple", "Orange", "Banana"];
  $scope.fruits = $rootScope.fruits;
});

app.controller("AddController",
                function($scope, $rootScope){
  $scope.addFruit = function() {
      $rootScope.fruits.push($scope.newFruit);
      $scope.newFruit = "";
  };
 });
```

Root Scope

- Demo

Events & Scope

- Events can be propagated across scopes
- emit event will go from a scope to all its parents
- broadcast will fire events all child nodes in the hierarchy

- Let us define 3 level of nested controllers
- Each scope has its own count variable and a handler for an event
- Let us propagate the event from the middle scope

Events & Scope

- The page

```
<div ng-controller="stemController">
  count at stem level : {{count}}<br/>

  <div ng-controller="branchController">
   count at branch level : {{count}}<br/>
   <input type="button" ng-click="$emit('incrEvent')"
              value="emit">
   <input type="button" ng-click="$broadcast('incrEvent')"
              value="broadcast">
      <div ng-controller="leafController">
          count at leaf level : {{count}}<br/>
      </div>
  </div>
</div>
```

Events & Scope

- The script

```
app.controller("stemController", function($scope){
  $scope.count = 0;
  $scope.$on('incrEvent', function() { $scope.count++; });
});

app.controller("branchController", function($scope){
  $scope.count = 0;
  $scope.$on('incrEvent', function() { $scope.count++ });
});

app.controller("leafController", function($scope){
  $scope.count = 0;
  $scope.$on('incrEvent', function() { $scope.count++; });
});
```

Demo

- $emit and $broadcast

AngularJS

Services & Dependency Injection

Services - Why?

- Self contained logic
- Modularised code
- Improves cohesion
- Easy to maintain
- Reduce duplicate code
- Components with Single and well defined responsibility
- Managed by dependency Injection

Dependency Injection - What is it?

- Components in a software depend on each other
- One component (Dependent - Higher level) wants to use another component (Servicing - Lower level) creates it - old way
- The servicing components are created by the framework and injected into dependent component - DI
- Controller components depend upon the Scope object
- But we just declare a function which takes $scope as parameter
- AngularJS takes care of creating the scope and passing it to the controller object function

AngularJS Services

- Singleton objects
- Can be injected into other modules (app or controller or even other services)
- AngularJS provides a set of built-in services for various application tasks ($http - dealt later)

- The fruits application rewritten using a service

Services

- The page - revisited from rootScope app

```
<div ng-controller="ListController">
   <ul>
     <li ng-repeat="fruit in fruits">
        {{fruit}}
     </li>
   </ul>
</div>

<div ng-controller="AddController">
   <input type="text" ng-model="newFruit">
   <input type="button" value="Add" ng-click="addFruit()">
</div>
```

Services

- The service object

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

app.factory("FruitService", function() {
        //Data inside the service object
        var fruits = ["Apple", "Banana", "Orange"];
        //Methods list() and add()
        return {
          list : function() {
            return fruits;
          },
          add : function(aFruit) {
            fruits.push(aFruit);
          }
         };
});
```

Services

- FruitService contains the list of fruits as data
- It exposes 2 methods by returning list & add
- We use the factory function to return a service object
- This is the most preferred way of creating a service

- There is an alternate way to use a constructor function (see later)

Services

- Controllers - Injection of services along with $scope

```
app.controller("ListController", function($scope,
                FruitService){
    $scope.fruits = FruitService.list(); //ref set
});

app.controller("AddController", function($scope,
                FruitService){
    $scope.addFruit = function() {
      FruitService.add($scope.newFruit);
      $scope.newFruit = "";
    };
});
```

Demo

- Service Factory

Service - using service

- Using a constructor function

```
app.service("FruitService", function(){
    var fruits = ["Apple", "Banana", "Orange"];
    return {
    list : function() {
        return fruits;
    },
    add : function(aFruit) {
        fruits.push(aFruit);
    }
};
```

Services - using a provider

- You need to add a $get method to expose the interface

```
app.provider("FruitService", function() {
      var fruits = ["Apple", "Banana", "Orange"];

      this.$get = function() {
        return {
          list : function() {
            return fruits;
          },
          add : function(aFruit) {
            fruits.push(aFruit);
          }
        };
      };
});
```

AngularJS

Directives

Directives - DIY

- Directives are used to create reusable components
- Add to HTML vocabulary
- Can be a
  - Attribute
  - Element /Tag
  - Style
  - Comment
- Directives embed a lot of UI logic
- Can be used for repetitive simple ones or highly complex ones

Directives - How to

- Attach the directive to an angular module
- Similar to controllers - Syntax

```
someModule.directive('dierctive-name', function() {
    return {
        template : "Some html or a snippet file",
        ... //Other properties if any
    };
});
```

- Returns an object with properties and methods
- This will be with some specific names understood by Angular
- template stands for a property which has the html snippet to include / replace

Directives - Sample #1

- Create a Directive called greet which writes “Hello, world!” inside a paragraph
- The script

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

app.directive('greet', function() {
  return {
    template : "<p>Hello, world!</p>"
  };
});
```

- used in html

```
<p greet></p>
```

- The greet directive used as an attribute (default behaviour)

Directives - Sample #1

- This produces the output in html page as

```
<p greet="">
     <p>Hello, world!</p>
</p>
```

- The inner p came from the template of the directive
- Outer p is from the html page

Directives - Sample #2

- To replace the original tag use the replace property

```
app.directive('greet', function() {
  return {
    template : "<p>Hello, world!</p>",
    replace : true
  };
});
```

- Now, the output will be only one p tag

```
<p greet="">Hello, world!</p>
```

Directives - Sample #3

- When we replace the tag (Element) why cant we use greet as a tag?
- Like this in html

```
<greet/>
```

- Just add restrict in directive

```
return {
  restrict : 'E', //E stands for element & A is default
  template : "<p>Hello, world!</p>",
   replace : true
};
```

- Now, the output is

```
<p>Hello, world!</p>
```

- Other restrict values (A - Attribute, C - Class, M - Comment)

Directives - Sample #4

- How to pass values (through attributes) to the greet element?
- <greet how="Hello" who="World"/>
- To handle the strings that are passed (even variables inside the moustache template) we need to add scope to directive

```
return {
  restrict : 'E',
  scope : {
      how : '@how',
      who : '@who'
    },
  template : "<p>{{how}}, {{who}}!</p>",
  replace : true
};
```

- Left hand side (how) is the variable inside directive, RHS (@how) is the attribute name in html

Directives - Sample #5

- If you don't want to use the cluttering {{ }} for binding data
- And still want 2 way binding then instead of @ use =
- The html

```
<div ng-controller="greetController">
  <div>
    <greet how="greetHow" who="greetWho"/>
  </div>
  <input type="button" ng-click="changeLanguage()"
                value="Spanish">
</div>
```

- There is a button to change the values in controller

Directives - Sample #5

- The script - Controller

```
app.controller('greetController', function($scope){
    $scope.greetHow = "Hello";
    $scope.greetWho = "World";

    $scope.changeLanguage = function() {
      $scope.greetHow = "Hola";
      $scope.greetWho = "World";
    };
});
```

Directives - Sample #5

- The script - Controller

```
app.directive('greet', function() {
  return {
    restrict : 'E',
    scope : {
      how : '=how',
      who : '=who'
    },
    template : "<p>{{how}}, {{who}}!</p>",
    replace : true
  };
});
```

Directives - scope variables

- If the name of the scope variable inside the controller is same as attribute name
- Drop the RHS name in directive

```
scope : {
            how : '=',
            who : '='
        }
```

- or

```
scope : {
            how : '@',
            who : '@'
        }
```

Directives - Sample #6

- Pass a controller function into directive
- Have a change colour function inside the controller to change colour of greeting
- Colour changes between black & red

```
app.controller('greetController', function($scope){
  $scope.greetHow = "Hello";
  $scope.greetWho = "World";
  $scope.isRed = false;
  $scope.textColor = "black";

  $scope.changeColor = function() {
    $scope.isRed = !$scope.isRed;
    $scope.textColor = $scope.isRed? "red": "black";
  };
});
```

Directives - Sample #6

- Get this function called during the click of greeting <p>

```
scope : {
 how : '=',
 who : '=',
 colors : '&colors'
},
template: '<p ng-click="colors()">{{how}},{{who}}!</p>',
```

- Now, the colors gets the chancellor passed by html & style controlled

```
<greet how="greetHow" who="greetWho"
       colors="changeColor()" style="color:{{textColor}}"/>
```

- In scope we can also write colors : '&'

Directives - Sample #7

- What if the greet element contains some content?
- At this stage anything between <greet> and </greet> will be gobbled up by our directive
- Now, we want the following to print the Hello & How are you messages

```
<greet how="Hello" who="World">
  How are you?
</greet>
```

- This can be done by transclusion

Directives - Sample #7

- Change the directive code to include this

```
template : '<span><p>{{how}}, {{who}}!</p>
           <p ng-transclude="true"></p></span>',
transclude : true
```

- Provide a location in the template where the content must be copied (ng-include)

Directives - Sample #8

- Use greet as an attribute but provide a value (message to display)

```
<div greet="Hello, World!"/>
```

- To access the attributes and modify DOM etc. we need to use a link function

```
link(scope, element, attributes)
```

- scope : Angular scope object
- element : that the directive matches
- attributes : hashmap of attributes (name - value pairs)
- There is a compile function which takes only element and attributes (no scope)

Directives - Sample #8

- The directive

```
app.directive('greet', function() {
  var linkFunction = function(scope, element, attributes){
    scope.text = attributes['greet'];
  };
  return {
    restrict : 'A',
    template : '{{text}}',
    link : linkFunction
  };
});
```

Directives - Sample #9

- How can directives communicate with each other?
- Let us rewrite greet to work with english and spanish
- `<greet english who="World"/>` should become
- Hello, World!
- `<greet spanish who="Mundo"/>` should become
- Hola, Mundo!
- x
- greet is a directive (element/tag)
- english and spanish are new directives as attributes to only greet
- And they alter the how text of greet

Directives - Sample #9

- This a greet directive to start with (how is dropped from scope)

```
app.directive('greet', function() {
  return {
    restrict : 'E',
    scope : {
      who : '@who'
    },
    template : "<p>{{how}}, {{who}}!</p>",
    replace : true
  };
});
```

- greet is a parent tag so we need some way of english and spanish tags to set the “how” variable

Directives - Sample #9

- To introduce how in greet we create a controller property in the returned object (after restrict property)

```
controller : function($scope, $element, $attrs) {
              $scope.how = 'xx';
              this.speakEnglish = function() {
                $scope.how = 'Hello';
              };
              this.speakSpanish = function() {
                $scope.how = 'Hola';
              };
    },
```

- controller takes same parameters as a link function but can have additional functions (notice that $ in parameters)
- This controller is going to be injected into other directives

Directives - Sample #9

- The english directive gets the greet controller in the link function
- This directive requires the greet directive

```
app.directive('english', function(){
  return {
    restrict : 'A',
    require : 'greet',
    link : function(scope, element, attrs, greetCtrl) {
          greetCtrl.speakEnglish();
        }
    };
  });
```

AngularJS

Filters

Filters - DIY

- Filters are created similar to directives
- Filter to split a string and produce an array

```
{{text | split: '#'}}
```

- split filter takes a separator (here it is #)
- assume a text variable containing 'a#b#c' split will produce an array ["a", "b", "c"]

```
app.filter('split', function(){
//return filter function object here
};
```

- Filter function takes parameters like input and the filter parameter (here it called separator - just a variable name)

Filters - DIY

- Filter code

```
app.filter('split', function(){
    return function(input, separator) {
      if (separator === undefined)
        separator = ',' ; //assume default comma
      var result = input.split(separator);
      return result;
    };
});
```

- if separator is not given - `{{text | split}}`
- default separator comma (,) is assumed

Filters - DIY

- Write a filter to join an array and produce a string

- Filters can be chained

```
{{text | split | join}}
```

- Will produce the text back (assuming comma as a separator)

Thank You!

Bala Sundarasamy
bala@ardhika.com