7. ADVANCED ANGULAR -JS

array

string

date functions

Regular expressions

JSON (JavaScript Object Notation)-methods

closures

scope basics:

---

1. Initializing an app and controller

2. angular data bindings and usage

3. $scope and controllers

4. $scope built in variables and methods

5.$scope prototype inheritance

6. use controllerAs syntax

7.controllerAs naming conventions

8.The built-in directive problems

1. Initializing an app and controller

```
<script src="angular.min.js"></script>
<script>
  var myApp = angular.module('myApp', []);
  myApp.controller("mycontroller", function ($scope) {
    $scope.name = "chandra";
  });
</script>
```

2. angular data bindings and usage

```
<script src="angular.min.js"></script>
<script>
  var myApp = angular.module('myApp', []);
  myApp.controller("mycontroller", function ($scope) {
    $scope.name = "chandra";
  });
</script>
<body ng-app="myApp">
  <div ng-controller="mycontroller">
    {{name}}<br />
      </div>
</body>
```

3. $scope and controllers

```
<script src="angular.min.js"></script>
<script>
  var myApp = angular.module('myApp', []);
  myApp.controller("mycontroller", function ($scope) {
    $scope.name = "chandra";
  });

  myApp.controller("childController", function ($scope) {
    $scope.address = "ap";
  });
</script>
```

4. $scope.$$distroyed

5.$scope prototype inheritance

```
<script src="angular.min.js"></script>
<script>
  var myApp = angular.module('myApp', []);
  myApp.controller("mycontroller", function ($scope) {
    $scope.name = "chandra";
  });

  myApp.controller("childController", function ($scope) {
    $scope.address = "ap";
  });
</script>

<body ng-app="myApp">
  <div ng-controller="mycontroller">
    {{name}}<br />
    parent : <input type="text" ng-model="name" />
    <div ng-controller="childController">
      {{name}}<br />
      child <input type="text" ng-model="name" />
    </div>

  </div>
</body>
```

```
<body ng-app="myApp">
  <div ng-controller="mycontroller">
    {{name}}<br />
    parent : <input type="text" ng-model="name" />
    <div ng-controller="childController">
      {{name}}<br />
      child <input type="text" ng-model="name" />
    </div>

  </div>
</body>
```

4. $scope built in variables and methods

$(public) vs $$(private)

1. $scope.$parent
2. $scope.$$watchers(internally)
3. $scope.$watch
4. $scope.$on
5. $scope.$root
6. $scope.$apply

$$-->not part of API...It is part of internal implementation of angualr

Good to know:

---

1. $scope.$$watchers
2. $scope.$$postDigest
3. $scope.$$phase

in above example, $scope.name is the property for mycontroller(parent)

what is prototype inheritance ?

in above example, if you change name property in parent text box..it will effect in all parent and child.because name is own property for mycontroller.

if you change in child scope it will crate own scope in child, so will not effect in the parent scope.

then if you change parent it will not effect in child because child already having scope.

6. use controllerAs syntax

The problem was caused because of the way prototype inheritance works with scopes. To solve this issue, there is a best practice. We can use an object on the scope, and always access variables on top of that object. It is very common in Angular to use the controller itself for such an object, and then bind the controller to the scope.

var vm = this; This variable vm is the object that we're going to use in order to store all the other variables we need on top of that specific scope.

```
<script src="angular.min.js"></script>
<script>
  var myApp = angular.module('myApp', []);
  myApp.controller("mycontroller", function ($scope) {
    var vm =this;
    $scope.address = vm;
    vm.name ="chandra";

  });

  myApp.controller("childController", function ($scope) {
```

```
  });
</script>

<body ng-app="myApp">
  <div ng-controller="mycontroller">
    {{address.name}}<br />
    parent : <input type="text" ng-model="address.name" />
    <div ng-controller="childController">
      {{address.name}}<br />
      child <input type="text" ng-model="address.name" />
    </div>

  </div>
</body>
```

this is called controller as

```
<script src="angular.min.js"></script>
<script>
  var myApp = angular.module('myApp', []);
  myApp.controller("mycontroller", function ($scope) {
    var vm =this;
    vm.name ="chandra";

  });

  myApp.controller("childController", function ($scope) {

  });
```

```
</script>

<body ng-app="myApp">
  <div ng-controller="mycontroller as address">
    {{address.name}}<br />
    parent : <input type="text" ng-model="address.name" />
    <div ng-controller="childController">
      {{address.name}}<br />
      child <input type="text" ng-model="address.name" />
    </div>

  </div>
</body>
```

7.controllerAs naming conventions

```
<script src="angular.min.js"></script>
<script>
  var myApp = angular.module('myApp', []);
  myApp.controller("mycontroller", function ($scope) {
    var vm =this;
    vm.name ="chandra";

  });

  myApp.controller("childController", function ($scope) {
    var vm =this;
    vm.collecge = "abc "+$scope.address.name;
  });
```

```
</script>

<body ng-app="myApp">
  <div ng-controller="mycontroller as address">
    {{address.name}}<br />
    parent : <input type="text" ng-model="address.name" />
    <div ng-controller="childController as child">
      {{child.collecge}}<br />
      {{address.name}}<br />
      child <input type="text" ng-model="address.name" />
    </div>

  </div>
</body>

----------
assign parent scope with same alias
----
<script src="angular.min.js"></script>
<script>
  var myApp = angular.module('myApp', []);
  myApp.controller("mycontroller", function ($scope) {
    var vm =this;
    vm.name ="chandra";

  });

  myApp.controller("childController", function ($scope) {
    var vm =this;
```

```
    vm.address = $scope.$parent.vm;//assign parent scope
    vm.collecge = "abc "+vm.address.name;
  });
</script>

<body ng-app="myApp">
  <div ng-controller="mycontroller as vm">
    {{vm.name}}<br />
    parent : <input type="text" ng-model="vm.name" />
    <div ng-controller="childController as vm">
      {{vm.collecge}}<br />
      {{vm.address.name}}<br />
      child <input type="text" ng-model="vm.address.name" />
    </div>

  </div>
</body>
```

8.The built-in directive problems

```
<script src="angular.min.js"></script>
<script>
  var myApp = angular.module('myApp', []);
  myApp.controller("mycontroller", function ($scope) {
    $scope.editdata = true;
    $scope.name ="chandra";
  });
</script>
```

```
<body ng-app="myApp">
  <div ng-controller="mycontroller">
    {{name}}<br />
    <div ng-if="editdata">
        parent : <input type="text" ng-model="name" />
    </div>
  </div>
</body>
```

in above ng-if will create separate scope..because of that if we change anything in textbox..it will not effet at line number 182

so the solution is use controllerAs i.e vm

```
<script src="angular.min.js"></script>
<script>
  var myApp = angular.module('myApp', []);
  myApp.controller("mycontroller", function ($scope) {
    var vm =this;
    vm.editdata = true;
    vm.name ="chandra";
  });
</script>
<body ng-app="myApp">
  <div ng-controller="mycontroller as vm">
    {{vm.name}}<br />
    <div ng-if="vm.editdata">
        parent : <input type="text" ng-model="vm.name" />
    </div>
```

```
    </div>
</body>
```

---

2. Manual Data Bindings

---

1. $scope.$watch

2. $watch function expressions

3. How Binding works

4. deep $watch

5. How deep $watch works

6. Destroying a $watch

1. $scope.$watch

---

let's go deeper and learn how the angular two way binding mechanism work. First I'll introduce you to a very important scope built in method called $watch. The $watch method can be used when you want to check if a certain value has changed and ran some code when it changes.

```
<script src="angular.min.js"></script>
<script>
  var myApp = angular.module('myApp', []);
  myApp.controller("mycontroller", function ($scope) {
    var vm =this;
    vm.editdata = true;
    vm.name ="chandra";
    vm.reset =function (){
      vm.name ="";
    }
    //let create watch variable on vm.name expression
```

```
    $scope.$watch("vm.name",function(){
        console.log("vm.name changed to "+vm.name);
    })
  });
</script>
<body ng-app="myApp">
  <div ng-controller="mycontroller as vm">
    {{vm.name}}<br />
    <div ng-if="vm.editdata">
        parent : <input type="text" ng-model="vm.name" />
    </div>
    <input type="button" ng-click="vm.reset()" value="Reset" />
  </div>
</body>
```

in above example if we change name it will automatically notify the $scope.$watch method.

if you want capture new changed value and old value of that expression we will use

```
$scope.$watch("vm.name",function(newValue,oldValue){
        console.log("vm.name changed to "+newValue + ' ' +oldValue);
    });
```

2. $watch function expressions

---

in previous practice we verified expression like vm.name

anothet way is there to watch with functions .

```
<script src="angular.min.js"></script>
<script>
  var myApp = angular.module('myApp', []);
  myApp.controller("mycontroller", function ($scope) {
    var vm =this;
    vm.editdata = true;
    vm.age =30;
    vm.reset =function (){
      vm.age ="";
    }
    //let create watch variable on vm.name expression
    $scope.$watch(function(){ return vm.age % 2 === 0}, function(newValue,oldValue){
      console.log(newValue ? "even" : "odd");
    })
  });
</script>
<body ng-app="myApp">
  <div ng-controller="mycontroller as vm">
    {{vm.age}}<br />
    <div ng-if="vm.editdata">
        parent : <input type="text" ng-model="vm.age" />
    </div>
    <input type="button" ng-click="vm.reset()" value="Reset" />
  </div>
</body>
```

3. How Binding works

---

ng-bind will help here

Here I will create my own binding ..for this i will create one div with id and will push textbox value via watch fucntion..

```
<script src="angular.min.js"></script>
<script>
  var myApp = angular.module('myApp', []);
  myApp.controller("mycontroller", function ($scope) {
    var vm =this;
    vm.editdata = true;
    vm.age =30;
    vm.reset =function (){
      vm.age ="";
    }
    //let create watch variable on vm.name expression
    $scope.$watch("vm.age", function(newValue,oldValue){
      document.getElementById("bindme").innerHTML = vm.age;
    })
  });
</script>
<body ng-app="myApp">
  <div ng-controller="mycontroller as vm">
    {{vm.age}}<br />
    <div ng-bind="vm.age"></div>
    <div id="bindme"></div>
    <div ng-if="vm.editdata">
        parent : <input type="text" ng-model="vm.age" />
    </div>
    <input type="button" ng-click="vm.reset()" value="Reset" />
  </div>
```

```
</body>
```

4. deep $watch:

Up until now, we looked at the watch expression that returns a simple value, a string, or a number. But what will happen if we try to watch a full object?

```
<script src="angular.min.js"></script>
<script>
  var myApp = angular.module('myApp', []);
  myApp.controller("mycontroller", function ($scope) {
    var vm =this;
    vm.userProperties = {
      color:"white",
      nickname:"chanti",
      age:30
    };
    vm.editdata = true;
/*
    vm.reset =function (){
      vm.userProperties = {
      color:"white",
      nickname:"chanti",
      age:20
    };
    }
    //let create watch variable on vm.name expression
    $scope.$watch(" vm.userProperties", function(newValue,oldValue){
      console.log("User properties changed"+vm.userProperties);
```

```
    })
    */
//above commented code tells if you cahnge entire object then only it will trigger..if you want to change some property then it wont effect for that we will use deep watch
    vm.reset = function(){
      vm.age=0;
    }
    $scope.$watch("vm.userProperties",function(){
      console.log("user properties changed",vm.userProperties);
    },true)
//the third parameter true tells angualr to create deep watch on object
  });
</script>
<body ng-app="myApp">
  <div ng-controller="mycontroller as vm">
    {{vm.userProperties.age}}<br />
    <div ng-bind=" vm.userProperties.age"></div>

    <div ng-if="vm.editdata">
        parent : <input type="text" ng-model=" vm.userProperties.age" />
    </div>
    <input type="button" ng-click="vm.reset()" value="Reset" />
  </div>
</body>
```

5. How deep $watch works

Notice what really happens behind the scenes when you use a deep watch. Angular actually creates a deep copy of the object and check if something has changed. It does a deep equal that goes over all the fields recursively. Notice that this feature should be used carefully. It might be easier for a developer to

use deep watches in certain places. But it has a high performance cost. If you decide to use a deep watch in your code, make sure that the objects you are using are not huge so it won't be such a high performance cost to go over them and try to deep equal them.

Your application is small enough so it doesn't really affect performance or the experience of the user. Using a deep watch really saves you a lot of code that had to be written if you would do it in any other way. Deep watch is a feature that angular developers used to use heavily when angular had just started. Leaving us today with legacy code that has usages of these watches. That is why it is a feature we must understand deeply when dealing with a mature angular app.

6. Destroying a $watch

Watchers should be destroyed when they are no longer needed. The watchers are costly. They make the digest look slower, which makes the whole app slower. We will talk about the digest look later on in the course. Sometimes we might want to stop a watcher because it make sense in their application.

we'll learn when watchers are being destroyed and how we can manually destroy them. Notice, that when we use the $watch function, we use it on a certain scope.

So whenever the scope is destroyed, all the watchers on that scope are also destroyed. Scopes that were generated because of an ng-controller directive or other directives are destroyed when Angular removes them from the dump.

```
<script src="angular.min.js"></script>
<script>
  var myApp = angular.module('myApp', []);
  myApp.controller("mycontroller", function ($scope) {
    var vm =this;
    vm.reset = function(){
      vm.age=0;
    }

  });
```

```
myApp.controller("internalController",function($scope){
  var vm =this;
  vm.address = $scope.$parent.vm;
  $scope.$watch("vm.address.age",function(){
     console.log("vm.address.age is strong"+vm.address.age);
  })
});

</script>
<body ng-app="myApp">
  <div ng-controller="mycontroller as vm">
     {{vm.age}}<br />
     parent : <input type="text" ng-model=" vm.age" />
     <input type="button" ng-click="vm.reset()" value="Reset" />

      <div ng-if="vm.age %2 === 0">
        <div ng-controller="internalController">
          Yeah! Controller arrived
        </div>
      </div>
    </div>
  </body>
  in above we lost watch if condition not mets
  ----------
  <script src="angular.min.js"></script>
  <script>
    var myApp = angular.module('myApp', []);
    myApp.controller("mycontroller", function ($scope) {
      var vm =this;
```

```
    vm.reset = function(){
      vm.age=0;
    }

});

myApp.controller("internalController",function($scope){
  var vm =this;
  vm.address = $scope.$parent.vm;
  var firstvalueofage = vm.address.age;
  $scope.$parent.$parent.$watch("vm.age",function(){
    console.log("vm.address.age is strong",firstvalueofage,vm.address.age);
  });
  //here watch will create for two times if value is even..
        //if even the extra watch will create for internal controller
 });

</script>
<body ng-app="myApp">
  <div ng-controller="mycontroller as vm">
    {{vm.age}}<br />
    parent : <input type="text" ng-model=" vm.age" />
    <input type="button" ng-click="vm.reset()" value="Reset" />

    <div ng-if="vm.age %2 === 0">
      <div ng-controller="internalController">
        Yeah! Controller arrived
      </div>
    </div>
```

```
</div>
</body>
```

Destroy watch manually:

```
<script src="angular.min.js"></script>
<script>
  var myApp = angular.module('myApp', []);
  myApp.controller("mycontroller", function ($scope) {
    var vm = this;
    vm.age = 30;
    vm.reset = function () {
      vm.age = 0;
    }

  });
  myApp.controller("internalController", function ($scope) {
    var vm = this;
    vm.address = $scope.$parent.vm;

    remainwatch = 3;
    var watchDestroyer = $scope.$watch("vm.address.age", function () {
      console.log("vm.address.age is strong", vm.address.age);
      remainwatch--;
      if (remainwatch == 0) {
        watchDestroyer();
      }
    });
  });
```

```
</script>

<body ng-app="myApp">
  <div ng-controller="mycontroller as vm">
    {{vm.age}}
    <br /> parent :
    <input type="text" ng-model=" vm.age" />
    <input type="button" ng-click="vm.reset()" value="Reset" />

    <div ng-if="vm.age % 2 == 0">
      <div ng-controller="internalController as vm">
        Yeah! Controller arrived
      </div>
    </div>
  </div>
</body>
```

3.The Digest Loop:

---

1. Basic digest
2. When does angular calls the digest loop
3. Dive into a digest loop
4. Infinite digest loops
5. When the digest loop is not called
6. integrate an external library
7. $apply vs $digest