**AngularJS Controller Tutorial With Example**

Let us talk about one of the AngularJS’s most useful feature which helps in making awesome **single page applications** in JavaScript – The Controller.

This tutorial is part of series of tutorials on AngularJS where we shall touch different aspects of Angular and its features. In our last tutorial we saw [Introduction of AngularJS](http://viralpatel.net/blogs/angularjs-introduction-hello-world-tutorial/) and also created a Hello World example using Angular.

Controllers are nothing but plain JavaScript functions which are bound to a particular scope. Don’t worry if this sounds gibberish. Shortly this all will make sense once we create a small Hello World using controller. Controllers are used to add logic to your view. Views are HTML pages. These pages simply shows the data that we bind to them using two-way data binding in Angular (Note: Two-way data binding is explained in [previous tutorial](http://viralpatel.net/blogs/angularjs-introduction-hello-world-tutorial/)). Basically it is Controllers responsibility to glue the Model (data) with the View.

**1. What are Scopes?**

Before we get into Controllers let us understand Scopes. Scope is nothing but an object that Glues the View with Controller. They hold the Model data that we need to pass to view. Scope uses Angular’s two-way data binding to bind model data to view.

Imaging $scope as an object that links Controller to the View. It is controllers responsibility to initialize the data that the view needs to display. This is done by making changes to $scope.

Let us see a small Hello World application to understand $scope.

|  |
| --- |
| <!DOCTYPE html>  <html ng-app>  <head>  <title>Hello World, AngularJS - ViralPatel.net</title>  <script type="text/javascript"      src="http://ajax.googleapis.com/ajax/libs/angularjs/1.0.7/angular.min.js"></script>  </head>  <body>  <div ng-controller="ContactController">       Email:<input type="text" ng-model="newcontact"/>      <button ng-click="add()">Add</button>      <h2>Contacts</h2>        <ul>          <li ng-repeat="contact in contacts"> {{ contact }} </li>      </ul>    </div>  <script type="text/javascript">      function ContactController($scope) {          $scope.contacts = ["hi@email.com", "hello@email.com"];            $scope.add = function() {          $scope.contacts.push($scope.newcontact);          $scope.newcontact = "";          }      }  </script>  </body>  </html> |

1.1. Online Demo

Play on JSFiddle – <http://jsfiddle.net/viralpatel/aLDJJ/>

In above demo, just write something in the textbox and press Add button. This will add whatever you type in textbox in an array. The content of array is displayed below in a list.

First thing you should note in demo is the attribute ng-controller.

1.2. ng-controller

This attribute defines a Controller to be bound with the view. In this case we defined a controller called ContactController in DIV using ng-controller attribute. Thus whatever we put inside that DIV, the ContactController will have its influence on it.

ContactController is nothing but a plain vanilla JavaScript function. In the demo we defined it as function. Also see the definition of ContactController function. There is an object $scope which we pass as an argument. This object is used to bind the controller with view. When AngularJS initialize this controller, it automatically creates and injects the $scope object to this function using dependency injection (More on dependency injection in coming tutorials). For now just note that the $scope object is created by Angular and injected in this controller function.

1.3. ng-repeat

Notice how we displayed a list of contacts using an attribute ng-repeat.

|  |
| --- |
| <li ng-repeat="contact in contacts">{{ contact }}</li> |

ngRepeat is one of the most used AngularJS attribute. It iterate through an array and bind the view with each element. So in our example it creates <li> tag for each item within contacts array. ngRepeat takes expression as argument. In our case “contact in contacts” where contact is user defined variable and contacts is an array within $scope.

In our final demo in this tutorial, we will use ng-repeat to iterate through an array of objects and paint each property in a table.

2. Initial state of a scope object

Typically, when you create an application you need to set up an initial state for an Angular scope. In our case we need initial state to be list of contacts.

On $scope object, we defined an array called contacts:

|  |
| --- |
| $scope.contacts = ["hi@email.com", "hello@email.com"] |

When Angular initilize this function (ContactController), it automatically creates this array and binds it in$scope object. Then in our view we display the array using ng-repeat attribute.

Thus, $scope provides us with a way to pass/retrieve objects from Controller to View and vice-versa.

2.1. ng-click

It is also possible to define functions on $scope and use the same in View. In our demo, we created a function add() on $scope and use it on Add button click:

|  |
| --- |
| $scope.add = function() {      ...  } |

The function add() is bound to Add button using an attribute ng-click. ng-click binds the click event on the button or link or any clickable element with the function that is defined within $scope. So in this case, whenever Add button is clicked, the add() method on $scope will be called.

In add() method we add (or push) a string in contacts array. This is the string that user types in textbox. Note that we bind textbox using ng-model attribute.

|  |
| --- |
| <input type="text" ng-model="contact" ... |

This textbox’s value we got in $scope.contact as we bind it using ng-model attribute.

Pretty nice, isn’t it!!

3. How to define a Controller

So we got some basic idea of what Controllers are. Just plain vanilla JavaScript functions that we add in an Angular application to add some business logic and bind the view with model.

In the above demo, we defined controller as JavaScript function. While this is the easiest way to define them, but is certainly not advisable one. If your application grows, soon you’ll have bunch of controllers lying here and there in code poluting the JavaScript namespace.

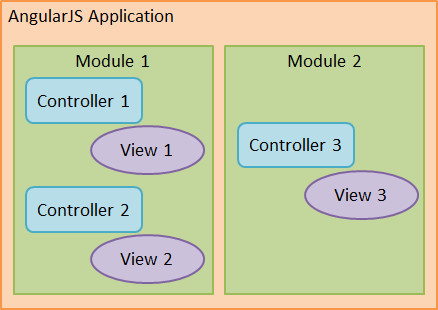
So instead of defining controller as:

|  |
| --- |
| function ContactController($scope) {        //...    } |

We should define Controller within Modules.

3.1. AngularJS Modules

So what-the-heck are modules? Well, modules are the logical entities that you divide you app in. So your app can contain several modules (like Transaction, Report, etc.). Each module represent a logical entity within the app.



Furthermore, each modules have several Controllers. As referred in above diagram, a module can contain one or more controllers and views. We haven’t touch based on Views. We will see how each module can be linked with view using Routing in AngularJS in our next tutorial. For now just assume that each controller has one or more views linked to it.

So in this case, we can add one or more controllers to a module. Let us check the syntax to create a module and add controller to it:

|  |
| --- |
| var myApp = angular.module('myApp',[]);    myApp.controller('ContactController', ['$scope', function($scope) {      $scope.contacts = ["hi@email.com", "hello@email.com"];        $scope.add = function() {          $scope.contacts.push($scope.contact);          $scope.contact = "";      }  }]); |

In above example we created a module called myApp using angular.module() method. Then we added a controller ContactController to this module. This is just an alternate way of defining a controller but recommended one.

Notice how controller is declared using myApp.controller() method. We passed an array to this method. First argument of array is string ‘$scope’ and next is the function itself that representContactController. We explicitly told Angular that we have one argument (dependency) to our ContactController which is $scope. This is useful when you Minify or obfuscate JavaScript for production release. In that case the argument $scope might be renamed to $s, but because we defined string ‘$scope’ as first argument, Angular is aware that first dependency to this controller is $scope object.

To cut it short, always stick to the above way of defining controllers.

4. Nested Controllers

We can declare the scope of controller in our HTML page using ng-controller attribute. For example:

|  |
| --- |
| <div ng-controller="CarController">      ...  </div> |

We can declare number of controllers and nest them within each other. For example:

|  |
| --- |
| <div ng-controller="CarController">      My name is {{ name }} and I am a {{ type }}        <div ng-controller="BMWController">          My name is {{ name }} and I am a {{ type }}            <div ng-controller="BMWMotorcycleController">              My name is {{ name }} and I am a {{ type }}            </div>      </div>  </div>    <script>  function CarController($scope) {        $scope.name = 'Car';      $scope.type = 'Car';    }    function BMWController($scope) {        $scope.name = 'BMW';    }    function BMWMotorcycleController($scope) {        $scope.name = 'BMWMotorade';      $scope.type = 'Motorcycle';    }      </script> |

4.1. Online Demo

Play on JSFiddle – <http://jsfiddle.net/viralpatel/gWz4K/>

In above demo, notice how each nested Controller’s scope override the scope of parents controller. First we defined a controller CarController which defines two variables name and type within scope. NextBMWController is nested within CarController using ng-controller attribute. BMWController overridesname attribute and change it to BMW. It does not change type attribute so type attribute is still Car.

BMWMotorcycleController is the inner-most controller defined within controllers hierarchy. It overrides both name and type attribute of scope.

This way you can nest one controller within another and take advantage of parent controllers attributes whenever needed.

5. Inheritance in Controllers

In order to take advantage of inheritance of scope in Nested controllers, one has to define Controllers one into another using ng-controller attribute. Sometimes you don’t want to define controllers like this but still want to use power of inheritance within controllers. May be you want to put common logic into BaseController and use it in all the child controllers.

In order to achieve this, we must use $injector object that AngularJS provides.

|  |
| --- |
| <div ng-controller="BMWController">        My name is {{ name }} and I am a {{ type }}        <button ng-click="clickme()">Click Me</button>    </div>      <script>  function CarController($scope) {        $scope.name = 'Car';      $scope.type = 'Car';        $scope.clickme = function() {          alert('This is parent controller "CarController" calling');      }    }    function BMWController($scope, $injector) {        $injector.invoke(CarController, this, {$scope: $scope});        $scope.name = 'BMW';    }  </script> |

5.1. Online Demo

Play on JSFiddle – <http://jsfiddle.net/viralpatel/WCZcZ/>

We define two controllers, CarController and BMWController. CarController defines two attributes name and type on $scope and also defines one method clickme().

BMWController just override name attribute. Notice that it does not have name and clickme defined within its body. These attributes are defined by parent controller in this case CarController. Within BMWController, we initialize CarController and bind it into current scope using $injector.invoke() method.

Once this is done, notice how in HTML page the BMWController points to its parent’s attributes.

6. End to end application using AngularJS Controller

Let us apply the knowledge that we acquired so far and create a ContactManager application. Following are some basic requirements of this application:

1. User can add new contact (name, email address and phone number)
2. List of contacts should be shown
3. User can delete any contact from contact list
4. User can edit any contact from contact list

Following is the HTML code which defines a FORM to save new contact and edit contact. And also it defines a table where contacts can be viewed.

6.1. The HTML

|  |
| --- |
| <div ng-controller="ContactController">      <form>      <label>Name</label>          <input type="text" name="name" ng-model="newcontact.name"/>      <label>Email</label>          <input type="text" name="email" ng-model="newcontact.email"/>      <label>Phone</label>          <input type="text" name="phone" ng-model="newcontact.phone"/>          <br/>          <input type="hidden" ng-model="newcontact.id" />       <input type="button" value="Save" ng-click="saveContact()" />      </form>  <table>  <thead>  <tr>      <th>Name</th>      <th>Email</th>      <th>Phone</th>      <th>Action</th>  </tr>  </thead>  <tbody>  <tr ng-repeat="contact in contacts">      <td>{{ contact.name }}</td>      <td>{{ contact.email }}</td>      <td>{{ contact.phone }}</td>      <td>          <a  href="#" ng-click="edit(contact.id)">edit</a> |          <a href="#" ng-click="delete(contact.id)">delete</a>      </td>   </tr>  </tbody>  </table>  </div> |

Just note that we have used ng-model, ng-click and ng-repeat attributes from Angular so far.

To add life to this application, we add following JavaScript code.

6.2. The JavaScript

|  |
| --- |
| var uid = 1;      function ContactController($scope) {        $scope.contacts = [          { id:0, 'name': 'Viral',            'email':'hello@gmail.com',            'phone': '123-2343-44'          }      ];        $scope.saveContact = function() {            if($scope.newcontact.id == null) {          //if this is new contact, add it in contacts array          $scope.newcontact.id = uid++;          $scope.contacts.push($scope.newcontact);          } else {          //for existing contact, find this contact using id          //and update it.          for(i in $scope.contacts) {              if($scope.contacts[i].id == $scope.newcontact.id) {              $scope.contacts[i] = $scope.newcontact;              }          }          }            //clear the add contact form          $scope.newcontact = {};      }          $scope.delete = function(id) {            //search contact with given id and delete it          for(i in $scope.contacts) {              if($scope.contacts[i].id == id) {                  $scope.contacts.splice(i,1);                  $scope.newcontact = {};              }          }        }          $scope.edit = function(id) {      //search contact with given id and update it          for(i in $scope.contacts) {              if($scope.contacts[i].id == id) {                  //we use angular.copy() method to create                  //copy of original object                  $scope.newcontact = angular.copy($scope.contacts[i]);              }          }      }  } |

First thing to note, we created a variable uid and set its initial value to 1. This variable is used to generate unique ids for each new contact we save. In real life application you may want to do this at backend.

We defined one controller ContactController. This controller defines following objects within $scope:

|  |  |  |
| --- | --- | --- |
| **Scope object** | **Type** | **Comment** |
| $scope.contacts | Array | Array to store contact objects. In real life app, this should be maintained at server-side. |
| $scope.saveContact | JavaScript Function | Saves the newcontact object within contacts array. Check if contact is new or is being updated |
| $scope.delete | JavaScript Function | Delete the contact object from contacts list based on id specified |
| $scope.edit | JavaScript Function | Update the contact object in contacts list based on id specified |

6.3. Online Demo

Play on JSFiddle – <http://jsfiddle.net/viralpatel/JFYLH/>

You can add new contact using the above form. Once new contact is saved, the list showing contacts will be updated. Each contact can be edited and deleted. You got the gist :)

That’s All Folks

AngularJS is fun.. isn’t it. We saw what Controllers are. How to create Controllers and use $scopeobject to bind model values with the views. We went through AngularJS attributes such as ng-controller and ng-repeat. Also we saw how Nested controllers and inheritance in controllers work.

Finally we created a Contact Manager application using AngularJS.

We are not done yet!!  where we will explorer world of Views and Routing using AngularJS :)  
**Update:** Next tutorial is published. Read now [AngularJS Routing and Views Tutorial](http://viralpatel.net/blogs/angularjs-routing-and-views-tutorial-with-example/)