**the Scope of the Directives in AngularJS**

## Scopes in AngularJS

Unlike the other MVC frameworks, AngularJS doesn’t have specific classes or functions to create modelobjects. Instead, AngularJS extended the raw JavaScript objects with custom methods and properties. These objects, also known as scope in AngularJS terms, work as a glue between the view and other parts ( directives, controllers and services ) inside the AngularJS application.

Whenever the AngularJS application is bootstrapped, a rootScope object is created. Each scope created by controllers, directives and services are prototypically inherited from rootScope. AngularJS documentation is one of the best resources to learn how scope inheritance works: see [Scopes in AngularJS](https://github.com/angular/angular.js/wiki/Understanding-Scopes). Understanding how scope inheritance works will be useful in following sections.

## Scope inside a directive

***Note***: This section assumes you’ve prior knowledge of creating a simple directive

All directives have a scope associated with them. They use this scope for accessing data/methods inside the template and link function. By default, unless explicitly set, directives don’t create their own scope. Therefore, directives use their parent scope ( usually a controller ) as their own.

However, AngularJS allows us to change the default scope of directives by passing a configuration object known as **directive definition object**. A directive definition object –– let’s call it as DDO –– is a simple JavaScript object used for configuring the directive’s behaviour,template..etc

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

app.controller("Ctrl1",function($scope){

$scope.name = "Harry";

$scope.reverseName = function(){

$scope.name = $scope.name.split('').reverse().join('');

};

});

app.directive("myDirective", function(){

return {

restrict: "EA",

scope: false,

template: "<div>Your name is : {{name}}</div>"+

"Change your name : <input type='text' ng-model='name' />"

};

});

<div ng-app="test">

<div ng-controller="Ctrl1">

<h2 ng-click="reverseName()">Hey {{name}}, Click me to reverse your name</h2>

<div my-directive class='directive'></div>

</div>

</div>

<http://jsfiddle.net/shidhincr/eyNYw/4/light/>

If we change the name inside the textbox, notice the header name also gets changed. Since there’s no scope provided in the *DDO*, the directive uses its parent scope. Therefore, any changes we make inside the directive are actually reflected in the parent scope. Similarly, parent Ctrl1 scope has a method to reverse the name and this gets triggered when we click on the header. Now as we expect, clicking on the header should reverse the name inside the directive too.

**Scope : True** ( Directive gets a new scope )

Now it’s time for the directive to get its own scope. This is achieved by setting a “true” value to the scope property of the *DDO*. When directive scope is set to “true”, AngularJS will create a new scope object and assign to the directive. This newly created scope object is prototypically inherited from its parent scope ( the controller scope where it’s been used ).

Confused ? Let’s see the exact differences between setting scope: true and scope: false :

* When scope is set to “**true**”, AngularJS will create a new scope by inheriting parent scope ( usually controller scope, otherwise the application’s rootScope ). Any changes made to this new scope will not reflect back to the parent scope. However, since the new scope is inherited from the parent scope, any changes made in the Ctrl1 ( the parent scope ) will be reflected in the directive scope.
* When scope is set to “**false**”, the controller Ctrl1 and directive are using the same scope object. This means any changes to the controller or directive will be in sync.

Let’s look at the following fiddle to make it more clear :

In the above example, we created a directive by returning a DDO from the function. There are a lot of properties of the DDO to learn, but here we’re just going to discuss the scope property, because, the values of scope property decides how the actual scope is created and used inside a directive. These values can be either **“false”**, **“true”** or **“{}”**. In the following sections, we’ll see how each of these affects directive’s behaviour.

## Different types of directive scopes

**Scope : False** ( Directive uses its parent scope )

Let’s try another example. We’ll create simple directive to render a div and a textbox that can show and change a name.

The name property gets the initial value from the Ctrl1 scope ( parent scope of the directive ).

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

app.controller("Ctrl1",function($scope){

$scope.name = "Harry";

$scope.reverseName = function(){

$scope.name = $scope.name.split('').reverse().join('');

};

});

app.directive("myDirective", function(){

return {

restrict: "EA",

scope: true,

template: "<div>Your name is : {{name}}</div>"+

"Change your name : <input type='text' ng-model='name' />"

};

});

<div ng-app="test">

<div ng-controller="Ctrl1">

<h2 ng-click="reverseName()">Hey {{name}}, Click me to reverse your name</h2>

<div my-directive class='directive'></div>

</div>

</div>

<http://jsfiddle.net/shidhincr/q3kex/3/light/>

First, try clicking on the header. We can see that the name gets reversed inside controller Ctrl1 and the directive. Next, change the name inside the textbox; the parent scope is not at all affected.

***Note:****Clicking on header again, makes no changes to the directive scope. I guess this is because the ng-model will create a new name property only when the textbox value is changed. Before this, name property inside directive was referring to it’s parent scope ( through prototype chain )*

**Scope : { }** ( Directive gets a new isolated scope )

This is the most interesting section. Till now, we saw two situations for directive scope creation. In the third type, we are going to set scope property in *DDO* to an **Object literal**. When an object literal is passed to the scope property, things are bit different. This time, there will be a new scope created for the directive, but it **will not be inherited from the parent scope**. This new scope also known as **Isolated scope** because it is completely detached from its parent scope.

Let’s re-write our original example like this :

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

app.directive("myDirective",function(){

return {

restrict: "EA",

scope: {},

link: function(scope,elem,attr){

// code goes here ...

}

}

});

So far, this is the recommended way of setting the scope on *DDO* while creating custom directives. Why? Because:

* It’ll make sure that our directive is generic, and placed anywhere inside the application. Parent scope is not going to interfere with the directive scope.

Though it’s called as an *Isolated scope*, AngularJS allows to communicate with the parent scope using some special symbols knows as **prefixes**. Because of course there are still situations where the directive needs to be able to exchange data with parent scope. The next section is dedicated to *Isolated scope* and its properties.

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

app.controller("Ctrl1",function($scope){

$scope.name = "Harry";

$scope.reverseName = function(){

$scope.name = $scope.name.split('').reverse().join('');

};

});

app.directive("myDirective", function(){

return {

restrict: "EA",

scope: {},

template: "<div>Your name is : {{name}}</div>"+

"Change your name : <input type='text' ng-model='name'/>"

};

});

<div ng-app="test">

<div ng-controller="Ctrl1">

<h2 ng-click="reverseName()">Hey {{name}}, Click me to reverse your name</h2>

<div my-directive class='directive'></div>

</div>

</div>

<http://jsfiddle.net/shidhincr/q3kex/4/light/>

We just created a directive with an isolated scope. Notice, even the parent scope has a name “Harry”, the textbox inside directive is blank. This is because of the new *Isolated scope* doesn’t know anything about its parent scope.

**But, can we pass some values from the parent scope to the directives now?**

Yes ! Not only that, we might need to handle situations like invoking callbacks in parent scope, two-way binding between parent & directives scope ..etc

To access any parent scope data, we need to pass that to our directive explicitly. This is achieved by setting properties on the scope object in the *DDO*. Imagine these properties as interfaces of the directive to communicate with outside scope. Another important thing is that, these properties also **MUST** be set as the attributes of the directive html element. If this confusing, let me explain with an example:

Just go through the below fiddle, and look at the “HTML”, “JavaScript” and “Results” tabs.

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

app.controller("MainCtrl", function( $scope ){

$scope.name = "Harry";

$scope.color = "#333333";

$scope.reverseName = function(){

$scope.name = $scope.name.split("").reverse().join("");

};

$scope.randomColor = function(){

$scope.color = '#'+Math.floor(Math.random()\*16777215).toString(16);

};

});

app.directive("myDirective", function(){

return {

restrict: "EA",

scope: {

name: "@",

color: "=",

reverse: "&"

},

template: [

"<div class='line'>",

"Name : <strong>{{name}}</strong>; Change name:<input type='text' ng-model='name' /><br/>",

"</div><div class='line'>",

"Color : <strong style='color:{{color}}'>{{color|uppercase}}</strong>; Change color:<input type='text' ng-model='color' /><br/></div>",

"<br/><input type='button' ng-click='reverse()' value='Reverse Name'/>"

].join("")

};

});

<div ng-app="app">

<div class="parent" ng-controller="MainCtrl">

<div class="line">

Name inside parent scope is: <strong>{{name}}</strong>

<input type="button" ng-click="reverseName()" value="Reverse name" />

</div>

<div class="line">

Color inside parent scope is: <strong style="color:{{color}}">{{color|uppercase}}</strong>

<input type="button" ng-click="randomColor()" value="Randomize color" />

</div>

<div class="directive" my-directive

name="{{name}}"

color="color"

reverse="reverseName()"

></div>

</div>

</div>

<http://jsfiddle.net/shidhincr/pJLT8/10/light/>