Angular, Data, and ES2015 Classes

Posted At : November 16, 2015 10:48 AM | Posted By : Cutter
Related Categories: Development, AngularJS, ES2015, JavaScript

I'm really loving the changes introduced in ES2015 (otherwise known as ES6 or the new JavaScript). At work we've transitioned to working in ES2015, and discovering the differences has been both fun and challenging, especially when it comes to changes in how variables are scoped. But we'll save some of that for another day, and talk for a moment about passing data around in an Angular application, and how you can have some fun with ES2015 classes.

In Angular, a Controller can have Services as dependencies. Services are singleton in nature, so a Service shared among multiple components can be used to "share" data, to a degree. First, let's create a simple controller with a few dependencies and a custom variable we want to track both within the controller, and among other bits of the app.

```
'use strict';

 class MyController {
  constructor ($scope, dataService, orderService) {
   this.$scope = $scope;
   this._dataSvc = dataService;
   this._orderSvc = orderService;

   this.myCrazyVar = null;
   dataService.myCrazyVar = this.myCrazyVar;
  }
 }

 myController.$inject = ['$scope', 'dataService', 'orderService'];

 export {MyController};
```

Next, let's create some simple method in the *orderService* that needs access to our custom variable. We can easily do this with the shared dataService:

```
'use strict';

 class OrderService {
  constructor (dataService) {
```

```
   this._dataSvc = dataService;
  }

  add (order) {
   // get our custom var for quick reference
   let forNow = this._dataSvc.myCrazyVar;
   // do a bunch of stuff, then
   forNow.OrderId = order.id;
   // give our changes back to our dataService
   this._dataSvc.myCrazyVar = forNow;
  }
}

orderService.$inject = ['dataService'];

export {OrderService};
```

First, some might ask "Cutter, why didn't you just pass that variable into the method?" Well, sometimes you just can't. Others might ask "Isn't it passed by reference?" Scoping changes have adjusted how this works as well. Changing the variable in the service won't automatically update your controller variable. We'll talk more about that in a moment. Still other's might ask "What is 'let'?" That's a conversation about the differences in variable assignment in ES2015, and is really a discussion for another day. Ultimately I used *let* because I need the variable to be mutable.

But, to explain what I'm doing here, the *add()* method takes an order. What you can see of the method, it gets our custom variable and applies it to a local variable for easy reference. We update it with data from the *order* that was passed in. We then reset the Service property with the updated data.

OK, but that syntax with the Service property is odd, coming off of ES5. How does that work? Well, ES2015 classes allow for implicit getters and setters of properties. Consider the following:

```
'use strict';

 class DataService {
  constructor () {
   this.myCrazyVar = null;
  }
 }

 export {DataService};
```

In the past, to access and change a property of an object would have required us to write some kind of getter or setter method. In this example, you can simply access and change the property directly through dot notation. But let's say you want to do something a bit more complex. There may be some bit of pre- or post-process you want to do, either when setting or getting the variable. For this, ES2015 classes allow you to define custom getters and setters:

```
'use strict';

 class DataService {
  constructor () {
   this._myCrazyVar = null;
  }

  set myCrazyVar (value) {
   // I can do what I want in here
   this._myCrazyVar = value;
  }

  get myCrazyVar () {
   // I could do stuff here too, if I wanted
   return this._myCrazyVar;
  }
 }

 export {DataService};
```

Where this could come in handy is in that inter-app communication. A Controller can call methods on the Service, to set values and stuff, but the Service can't automatically pass data back to the Controller based on actions from other items accessing the Service (like our orderService interaction above). But, in Angular, we can use event handling and binding in this instance. First, let's put a listener on our controller:

```
constructor ($scope, dataService, orderService) {
  this.$scope = $scope;
  this._dataSvc = dataService;
  this._orderSvc = orderService;

  this.myCrazyVar = null;
```

```
  this._dataSvc.myCrazyVar = this.myCrazyVar;
  $scope.$on('crazyUpdated', ($event, newValue) => this.myCrazyVar = newValue);
 }
```

Yes, the arrow function is a different concept for most front-end-only developers. I'm not going in depth on that here, but you can find plenty of info out there about them. The gist here is that if the *crazyUpdated* event is cast it will pass a new value, that we then use to update the Controller variable. This also tells us that *myCrazyVar* will always be changed from outside of the Controller. Now let's do some magic to make sure that event gets cast. In our *dataService*:

```
'use strict';

 class DataService {
  constructor ($rootScope) {
   this._myCrazyVar = null;
  }

  set myCrazyVar (value) {
   this._myCrazyVar = value;
   $rootScope.$broadcast('crazyUpdated', value);
  }

  get myCrazyVar () {
   // I could do stuff here too, if I wanted
   return this._myCrazyVar;
  }
 }

 DataService.$inject = ['$rootScope'];

 export {DataService};
```

So we use our custom property setter. When the value is changed, it automatically broadcasts that change. The Controller then picks up that change, and applies it to it's own internal variable.

So, what have we learned here? Well, first there's some samples on using ES2015 classes as Controllers and Services within Angular. Simple examples, but there you go. Next, we talked about class property getters and setters, both implicit (no need to define, they just work), and explicit. Our explicit example shows where you can apply some additional logic during those processes. This may not be the greatest example,

but it shows you that you can do stuff. Probably the greatest usages here will be in data validation, or in splitting concatenated data (like a full name to first and last, for example).

Using those ES6 classes in your app is a matter of *import*ing the classes into your app:

```
import {MyController} from './MyController';
 import {DataService} from './DataService';
 import {OrderService} from './OrderService';

 // ... other app init stuff
 .controller('MyController', MyController)
 .service('dataService', DataService)
 .service('orderService', OrderService);
```

So, it's still all new to me, these changes to scope and classes and the like. But it's fun, and powerful, and has a ton of possibilities. If I screwed something up just let me know. All feedback is welcome.