ES2015, Promises, and Fun With Scope

Posted At : November 19, 2015 10:26 AM | Posted By : Cutter
Related Categories: Development, AngularJS, ES2015, JavaScript

I've been using Promises for some time now. JQuery has acted as a shim for some time, and several other libraries have been around as well. ES2015 includes Promises natively. If you're unfamiliar with Promises, I strongly suggest you **read this great post** by **Dave Atchley**.

Like I said though, I've been using Promises for a while now. So, when I started moving to ES2015 it was a bit of a kick in the pants to find issues with implementing my Promises. Let me give you an example of how something might've been written before:

```
'use strict';

 module.exports = [
  '$scope', 'orders', 'dataService',
  function ($scope, orders, dataService) {
   var self = this;

   self.orders = orders;

   self.addOrder = function (order) {
    // ... do some stuff
    // get original
    dataService.get(order.id)
     .then(self._updateOrders)
     .catch(function (error) {
      // do something with the error
     });
   };

   // faux private function, applied to 'this' for unit testing
   self._updateOrders = function (order) {
    // ... some process got our order index from orders, then...
    orders[index] = $.extend(true, orders[index], order);
   };
  }
 ];
```

Seems pretty straightforward, right? *addOrder()* gets called, which does some stuff and then retrieves an order from the service. When the service returns the order, that's passed to the *_updateOrders()* method, where it finds the correct item in the array and updates it (I know, it's flawed, but this is just an example to show the real problem).

So, what's the problem? That works great. Has for months (or even years). Why am I writing this post? Fair question. Let's take a look at refactoring this controller into an ES2015 class. Our first pass might look like this:

```
'use strict';

 class MyController {
  constructor ($scope, orders, dataService) {
   this.$scope = $scope;
   const myOrders = [];
   this.orders = myOrders.push(orders);
   this._dataService = dataService;
  }

  addOrder (order) {
   // ... do some stuff
   // get original
   this._dataService.get(order.id)
    .then(this._updateOrders)
    .catch(function (error) {
     // do something with the error
    });
  }

  _updateOrders (order) {
   // ... some process got our order index from orders, then...
   this.orders[index] = $.extend(true, this.orders[index], order);
  }
 }

 MyController.$inject = ['$scope', 'orders', 'dataService'];

 export {MyController};
```

That looks good, right? Well....

When *MyController.addOrder()* gets called, with this code, the *get()* method is called on the service, and... BOOM! Error. It says there is no *_updateOrders()* on *this*. What? What happened?

Well, it's not on your scope. Why? Because ES2015 has changed how scope is handled, especially within the context of a class. "this" is not the same in the context of the Promise's *then()*, at this point. But then, how are you supposed to reference other methods of your class?

Bom! Bom! BAAAAHHHHH! Use an arrow function. "Wait? What?" (lot's of confusion today) That's right, an arrow function. From MDN:

> An **arrow function expression** (also known as **fat arrow function**) has a shorter syntax compared to function expressions and lexically bind the this value (does not bind its own this, arguments, super, or new.target). Arrow functions are always anonymous.

If you aren't still confused at this point you are a rockstar. Basically what it says is that *this* will become of the context from which you're using the arrow function. So, in terms of a Promise, if we change our *addOrder()* method like this:

```
addOrder (order) {
  // ... do some stuff
  // get original
  this._dataService.get(order.id)
   .then((order) => this._updateOrders(order))
   .catch(function (error) {
    // do something with the error
   });
 }
```

This then fixes our *this* scoping problem within our *then*. Now, I know this isn't much in the way of an explanation into "How" it fixes it (other than setting the right *this*), and I know I'm not explaining what an **arrow function** is either. Hopefully this is enough to stop you from banging your head against the wall anymore, provides a solution, and gives you some clues on what to search for in additional information.

So, as always I welcome your feedback/suggestions/criticisms. Feel free to add a comment below, or drop me a direct line through the "contact" links on this page.