Creating Custom AngularJS Directives Part 3 - Isolate Scope and Function Parameters

by Dan Wahlin

In Part 2 this series I introduced isolate scope and how it can be used to make directives more re-useable. A big part of isolate scope is **local scope properties** that rely on characters such as @, = and & to handle data binding as well as function delegation. By using these properties you can pass data into and out of AngularJS directives. If you're new to local scope properties in directives I'd recommend reading the first two articles in this series. In this article I'll explore passing parameters out of a directive into an external function.

Isolate Scope and Function Parameters

By using local scope properties you can pass an external function (such as a $scope function defined within a controller) into a directive. This is done using the **&** local scope property. Here's an example of defining a local scope property named **add** that can store a reference to an external function:

```
angular.module('directivesModule').directive(
  'isolatedScopeWithController', function () {

    return {

        restrict: 'EA',

        scope: {

            datasource: '=',
            add: '&',
        },
        controller: function ($scope) {

            ...

            $scope.addCustomer = function () {
                //Call external scope's function
                var name = 'New Customer Added by Directive';
                $scope.add();

                //Add new customer to directive scope
                $scope.customers.push({
                    name: name
                });
            };
        },
        template: '<button ng-click="addCustomer()">' +
                  'Change Data</button><ul>' +
                  '<li ng-repeat="cust in customers">' +
                  '{{ cust.name }}</li></ul>'
    };
});
```

A consumer of the directive can pass an external function into the directive by defining an **add** property as shown next:

```
<div isolated-scope-with-controller datasource="customers" add="
addCustomer()"></div>
```

In this case the **addCustomer()** function will be invoked when the user clicks the button created by the directive. No parameters are passed so it's a relatively straightforward operation to perform.

How do you pass parameters to **addCustomer()** or another function though? For example, assume that the**addCustomer()** function shown in the following controller needs to have a **name** parameter value passed when the directive invokes the function:

```
var app = angular.module('directivesModule', []);

app.controller('CustomersController', ['$scope',
  function ($scope) {
    var counter = 0;
    $scope.customer = {
        name: 'David',
        street: '1234 Anywhere St.'
    };

    $scope.customers = [];

    $scope.addCustomer = function (name) {
        counter++;
        $scope.customers.push({
            name: (name) ? name : 'New Customer' + counter,
            street: counter + ' Cedar Point St.'
        });
    };
    $scope.changeData = function () {
        counter++;
        $scope.customer = {
            name: 'James',
            street: counter + ' Cedar Point St.'
        };
    };
}]);
```

Passing a parameter out of the directive to the external function is quite straightforward once you know the trick but requires a bit of upfront knowledge to get it working properly. Here’s what most devs try to do initially:

```
angular.module('directivesModule').directive('isolatedScopeWithC
ontroller', function () {
    return {
        restrict: 'EA',
        scope: {
            datasource: '=',
            add: '&',
        },
        controller: function ($scope) {
            ...

            $scope.addCustomer = function () {
                //Call external scope's function
                var name = 'New Customer Added by Directive';
```

```
                 $scope.add(name);

                 //Add new customer to directive scope
                 $scope.customers.push({
                     name: name                 });
             };
         },
         template: '<button ng-click="addCustomer()">' +
                   'Change Data</button><ul>' +
                   '<li ng-repeat="cust in customers">' +
                   '{{ cust.name }}</li></ul>'
     };
});
```

Notice that the directive's controller calls **$scope.add(name)** to try to call the external function with a parameter value. That should work right? It turns out that the parameter value will be **undefined** in the external function which will leave you scratching your head as to why. So what do we do?

Option 1: Using an Object Literal

One technique that can be used in this scenario is to pass an object literal. Here's an example of how a name value can be passed out of the directive to an external function:

```
angular.module('directivesModule').directive(
  'isolatedScopeWithController', function () {
    return {
        restrict: 'EA',
        scope: {
            datasource: '=',
            add: '&',
        },
        controller: function ($scope) {
            ...

            $scope.addCustomer = function () {
                //Call external scope's function
                var name = 'New Customer Added by Directive';
                $scope.add({ name: name });

                //Add new customer to directive scope
                $scope.customers.push({
                    name: name,
                    street: counter + ' Main St.'
                });
            };
        },
        template: '<button ng-click="addCustomer()">' +
                  'Change Data</button>' +
                  '<ul><li ng-repeat="cust in customers">' +
                  '{{ cust.name }}</li></ul>'
    };
});
```

Notice that the call made with **$scope.add()** now includes an object literal. This still won't work though unfortunately! So what's the trick? The **name** property defined in the object literal must also be defined when the external function is assigned to the directive. It's important that the parameter name match with the property name defined in the object literal. Here's an example of doing that:

```
<div isolated-scope-with-controller datasource="customers"
     add="addCustomer(name)"></div>
```

You can see that a **name** parameter is added to **addCustomer()** which is assigned to the directive's **add** local scope property. Since the parameter name matches the property name in the object literal everything works as expected. Not exactly intuitive but it works.

Option 2: Storing a Function Reference and Invoking It

The challenge with the previous option is that the name of the parameter has to be defined when the directive is used **and** it has to match the object literal property name passed out of the directive. If anything doesn't match up properly the technique won't work. Although this option gets the job done, there's a lot of what I like to call "tribal knowledge" involved and if the directive isn't well documented it would be hard to figure out what to name the parameter (or parameters if multiple are passed) without looking through the directive's source code.

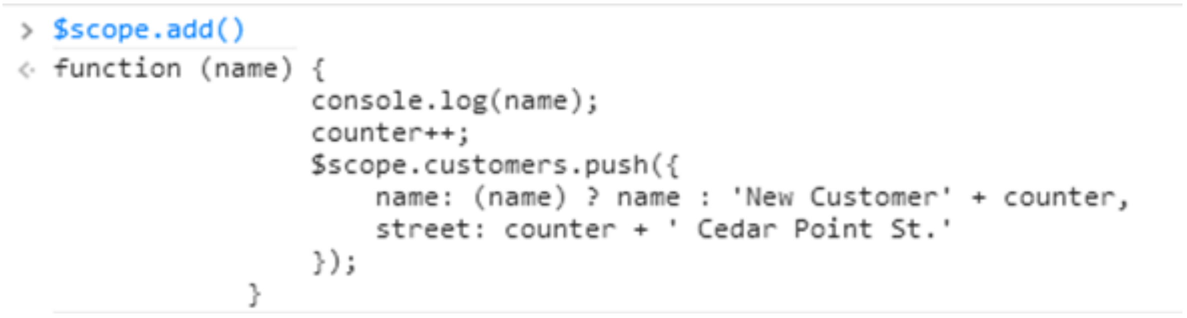
Another technique that can be used is to define the function on the directive without adding the parenthesis after the function name as shown next:

```
<div isolated-scope-with-controller-passing-parameter2
     datasource="customers" add="addCustomer"></div>
```

To pass the parameter value to the external **addCustomer** function you can do the following in the directive. Locate the **$scope.add()(name)** code below to see how the **addCustomer** function can be invoked while also passing a **name** parameter value:

```
angular.module('directivesModule').directive(
  'isolatedScopeWithControllerPassingParameter2', function () {
    return {
        restrict: 'EA',
        scope: {
            datasource: '=',
            add: '&',
        },
        controller: function ($scope) {

            ...

            $scope.addCustomer = function () {
                //Call external scope's function
                var name = 'New Customer Added by Directive';

                $scope.add()(name);

                ...
            };
        },
        template: '<button ng-click="addCustomer()">' +
                  'Change Data</button><ul>' +
                        '<li ng-repeat="cust in customers">'
                        '{{ cust.name }}</li></ul>';
      };
});
```

How and why does this technique work? Calling **$scope.add()** gets you to the function in this case as shown by the following console output. That's because the parenthesis weren't added when the function was assigned to the **add** local scope property shown earlier.



```
> $scope.add()
< function (name) {
                      console.log(name);
                      counter++;
                      $scope.customers.push({
                          name: (name) ? name : 'New Customer' + counter,
                          street: counter + ' Cedar Point St.'
                      });
                  }
```

The second set of parenthesis (**$scope.add()(name))** then invokes the function and passes the **name** parameter value . This of course requires the consumer of the directive to leave off the parenthesis (which may or may not be "normal" in your organization) but it really cleans things up compared to option 1 in my opinion. The directive documentation would still need to define the parameters that can be passed to the external function but at least the code to make that possible is simplified. Kudos to **Andy Jackson** for suggesting this technique.

& Local Properties - Behind the Scenes

If you're interested in some of the internal processes that run to make this happen, when an **&** local scope property is invoked from within a directive (such as the **add** local scope property in the previous examples) the following code is executed:

```
case '&':
    parentGet = $parse(attrs[attrName]);
    isolateScope[scopeName] = function(locals) {
        return parentGet(scope, locals);
    };
break;
```

The **attrName** variable shown above represents the **add** local scope property from the directive in this example. The **parentGet** function that is returned from the call to **$parse** looks like the following.

```
function (scope, locals) {
      var args = [];
      var context = contextGetter ? contextGetter(scope, locals)
          : scope;

      for (var i = 0; i < argsFn.length; i++) {
        args.push(argsFn[i](scope, locals));
      }
      var fnPtr = fn(scope, locals, context) || noop;

      ensureSafeObject(context, parser.text);
      ensureSafeObject(fnPtr, parser.text);
      // IE doesn't have apply for some native functions
      var v = fnPtr.apply
            ? fnPtr.apply(context, args)
            : fnPtr(args[0], args[1], args[2], args[3], args[4])
;

      return ensureSafeObject(v, parser.text);
}
```

This code handles mapping the object literal properties to the external function parameters and then invokes the function. This isn't something you need to know to work with **&** local scope properties of course, but it's always fun to dive into what AngularJS is doing behind the scenes.

Conclusion

You can see that passing parameters is a bit tricky and not exactly intuitive. However, once you know how the process works it's not too bad to work with and use.

Check out my AngularJS Custom Directives course for additional information about building your own directives.