Angularjs: sharing state between directives and their controllers

by Benny Bottema - Friday, January 03, 2014

http://www.bennybottema.com/2014/01/03/angularjs-sharing-state-between-directives-and-their-controllers/

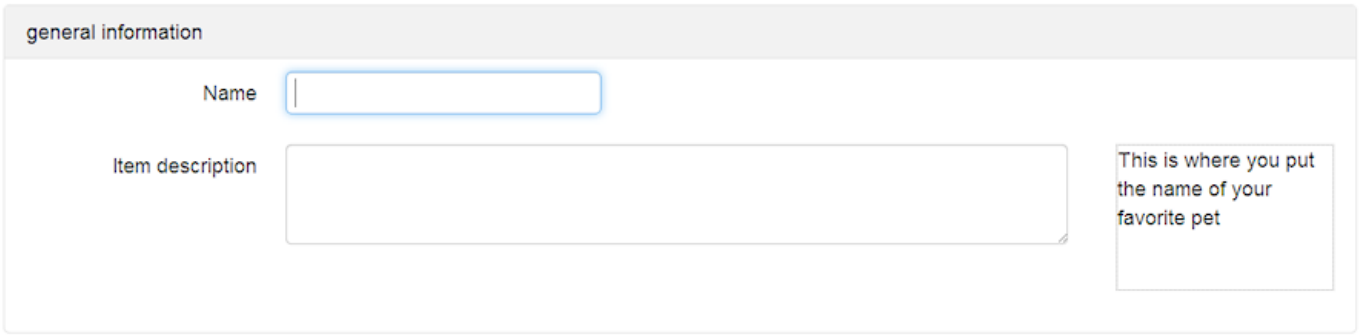
The thing with directives, controllers, scope, link and compile is: they all share keywords and concepts. This puts so much trees in front of the forest, that until you're clear on the individual underlying concepts you have no hope of understanding the abstractions made on top of it. And so it took me a long time to understand how I can make directives/controllers share state, *even if they're not nested*.

That last part is important, because in essence, you can't **share data** unless you also **share an ancestor**. The only exception to this is if you're using decoupled communication through events and eventlisteners.

Contents

The example we're not doing today

Ok, let's dive into today's problem we're not solving: A form with several inputs, and each one have a tooltip, except the tooltip should be displayed on a specific location, the same for all inputs. Like an information box… in fact it's not a tooltip at all.



Now I'm not actually going into the code of this particular form, as it is a real world example that I'm working on that contains all kinds of stuff, like a bootstrap ui accordion and other styling that would get in the way of this example. Instead, I will treat the problem essence with the following example:

The example we *are* doing today

How to share state using three directives in a triangle affair

```
<div ng-app="app">
    <input info="&apos;This is where you put the name of your favorite
 pet&apos;"></input>
    <div>result should come here</div>
</div>
```

The solution is to have three directives and the trick is that one of them functions as parent scope through which the other two talk.

And here it is:

```
/*
 * everything is in info-controller.js
 */

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

// parent scope
app.directive('infomanager', function() {
    return {
        controller: function($scope) {
            this.scope = $scope;
        }
    };
});

// the directive that will set the value on the parent scope
app.directive('info', function() {
    return {
        require: '^infomanager',
        scope : {
            info : '=info'
        },
        link: function(scope, element, attrs, infoManager) {
            infoManager.scope.info = scope.info;
        }
    };
});
```

```
// the directive that (implicitly) reads from the parent scope
app.directive('showinfo', function(){
    return {
        template:'{{info}}',
        require:'^infomanager'
    }
});
```

The HTML then looks like this:

```
<div ng-app="app" infomanager>
    <input info="&apos;This is where you put the name of your favorite
 pet&apos;"></input>
    <div showinfo></div>
</div>
```

How to update the text when we focus on an <input>

Ok great, we now have a way to get information accross the page into the hint box. Now how do we change it when we focus on another input box?

One thing a *link function* is responsible for is listening to DOM changes and reflecting this in an angular scope and then have Angular process the new value. In our case we're listening to *focus* events:

```
// the directive that will set the value on the parent scope
app.directive('info', function() {
    return {
        require: '^infomanager',
        scope : {
            info : '=info'
        },
  link : function(scope, element, attrs, infoManager) {
   element.on('focus', function() {
    // timeout is needed because the focus event fires during already
active $digest
    $timeout(function() {
     infoManager.scope.info = scope.info;
     scope.$apply(); // trigger new $digest to process new model value
    });
```

```
        });
  }
    };
});
```

That’s it, we listening to DOM events and reflecting that in Angular’s lifecycle.

Make the info directive generic by allowing us to define the event to listen to

We can even make this component more generic, by dynamically indicate the type of event which to listen to:

```
// the directive that will set the value on the parent scope
(..)
  link : function(scope, element, attrs, infoManager) {
   element.on(attrs.hintEvent, function() {
    (..)
   });
  }
(..)
```

And the modified HTML:

```
<div ng-app="app" infomanager>
    <input info="&apos;This is where you put the name of your favorite
 pet&apos;" hintEvent="focus"></input>
    <div showinfo></div>
</div>
```

All in all the HTML is very clean. Directives, although numerously enough with just the three of them, have been kept as *lite* as possible. Some would argue it is not a good thing to expose the entire scope in the infomanager. However, I already wished that third parent directive was not necessary, but since it is, let’s not pretend we were planning to have ‘managing behavior’ just to share this state. As far as I’m concerned, the state in the parent *belongs* to the underlying info and showinfo directives.

- example jsfiddle

PDF generated by Kalin's PDF Creation Station