AngularJS: Lesson 3

Custom Directives

The New York Code + Design Academy

AngularJS by Google

Custom Directives

For creating reusable UI components
or reusable, composable UI behaviors

Defining a directive

Prefix directive names with your own short namespace to avoid collisions

```
angular.module('angularjsTutorial')
.directive('ajstDirective', function() {
     return {
          // Directive definition object
          // https://docs.angularjs.org/api/ng/service/$compile#directive-definition-object
     };
});


// Can also dependency-inject here
.directive('ajstDirective', ['$log', function($log) {
});
```

Including a directive

JS: words in directive name are camelCase
HTML: words in directive name are dash-separated

```
// JS
.directive('ajstDirective', function() {
});

// HTML
<ajst-directive></ajs-directive>
<div data-ajst-directive></div>
<div ajst-directive></div>
```

Directive Template

HTML can be inline or retrieved through a URL

```
// Inline. Use this for small templates
.directive('ajstDirective', function() {
     return {
          template : '<div></div>'
     }
});

// Async
.directive('ajstDirective', function() {
     return {
          templateUrl : 'components/component-name/directive-template.html'
     }
});
```

‘restrict’

Define how a directive should be included

```
.directive('ajstDirective', function() {
    restrict : 'E' // only as an element : <ajst-directive></ajst-directive>
});

.directive('ajstDirective', function() {
    restrict : 'A' // only as an attribute : <div ajst-directive></div>
});

.directive('ajstDirective', function() {
    restrict : 'C' // only as a classname: <div class="ajst-directive"></div>
});
```

‘scope’

2 options:
Inherited Scope
Isolate Scope

```
// inherit scope from parent
.directive('ajstDirective', function() {
     // omit the 'scope' property
});

// create isolated scope
.directive('ajstDirective', function() {
     scope : {}
});
```

Passing information to a directive

Done through isolate scope & HTML attributes

```
.directive('ajstDirective', function() {
     scope : {
          string : '@' // match an attribute of the same name, expect string
          anotherStr : '@differentName, // match an attribute named 'different-name'
          boundVariable : '=', // two-way binding. Can also have a different name like above
          expression : '&' // evaluatable expression. Can also have a different name like above
     }
});


// HTML
<ajst-directive
     string="Title"
     different-name="Subtitle"
     bound-variable="mainCtrl.newTodoTitle"
     expression="mainCtrl.doSomething()"
></ajst-directive>
```

‘controller’

Custom injections come after

```
.directive('ajstDirective', function() {
     return {
          controller :function ($scope){

          }
     }
```

‘link’

Called once template has been instantiated.

At this point, you have access to do DOM manipulation, add event listeners, etc.

Passed the following 4 parameters:
scope, element, attrs, controller

```
.directive('ajstDirective', function() {
     link : function (scope, element, attrs, controller){
          // element is the angular-wrapped DOM element of your template
          element.on('click', ...);
     }
});
```

‘transclude’

When you want to pass HTML content to a directive

```
.directive('ajstDirective', function() {
     transclude: true
});


// Directive template
<div>
     <p>some directive content</p>
     <ng-transclude></ng-transclude>
</div>
```

Animation

Can take advantage of animation by using the $animate service inside the ‘link’ function.

https://docs.angularjs.org/guide/animations

```
.directive('ajstDirective', ['$animate', function($animate) {
     return {
          link : function(scope, element, attrs) {
               element.on('click', function() {
                    if(element.hasClass('clicked')) {
                         $animate.removeClass(element, 'clicked');
                    } else {
                         $animate.addClass(element, 'clicked');
                    }
               });
          }
```

Further Reading

Guide to directives
https://docs.angularjs.org/guide/directive

In-depth docs on directive options
https://docs.angularjs.org/api/ng/service/$compile

How directive compilation works
https://docs.angularjs.org/guide/compiler

Services

Features that you want to work across components & controllers

What goes in them?

- Business logic
- Data persistence
- Server communication
- Shared state
- Caching
- Factories
- Third-party JS libraries (wrapped in order to be injectable and testable)

Services

Other interesting features

- Not dependent on views
- Each is created as a singleton - one instance is shared across the module

Services

5 methods to create your own service

- Value
- Constant
- Factory
- Service
- Provider

Services: Value

When all you need is to store a value (any type) and retrieve it at run time

```
// Defining the value
angular.module('angularjsTutorial')
     .value('googleApiKey', '12345');

// Retrieve the value
angular.module('angularjsTutorial')
     .controller('MainCtrl', ['$scope', 'googleApiKey', function($scope, googleApiKey){
```

Services: Constant

Just like Value, but also available at config time

```
// Defining the value
angular.module('angularjsTutorial')
     .constant('googleApiKey', '12345');

// Retrieve the value at module config time
angular.module('angularjsTutorial')
     .config(['googleApiKey', function(googleApiKey) {
```

Services: Factory

Simplest way to create more advanced services

Service API is the return object

Equivalent to Service, just a different style

```
angular.module('angularjsTutorial')
     .factory('MyMessageService', [function(){
          var privateMessage = 'hello';
          return {
               getMessage : function(){
                    return privateMessage;
               },
               setMessage : function(value){
                    privateMessage = value;
               }
          }
```

Services: Service

Equivalent to Factory, just a different style

Uses a constructor
Service API is the “this” from the constructor

```
angular.module('angularjsTutorial')
     .service('MyMessageService', [function MyMessageService(){
          var privateMessage = 'hello';

          this.getMessage = function(){
               return privateMessage;
          };
          this.setMessage = function(value){
               privateMessage = value;
          };
```

Services: Provider

The grandaddy.

More complex, but lets you customize the service at configuration time.
The rest of the Angular services use this underneath.

```
angular.module('angularjsTutorial')
     .provider('MyMessageService', [function MyMessageService(){
          var privateMessage = 'hello';

          this.getMessage = function(){
               return privateMessage;
          };
          this.setMessage = function(value){
               privateMessage = value;
          };
     }]);
```

Unit-testing services

1. Declare a variable scoped to your describe() block to hold the service
2. In the first beforeEach, initialize your module
3. In a following beforeEach, inject your service
    a. Injection uses “underscore wrapping” to refer to your service so you

```
// src/components/todo/todo.service.js
angular.module('angularjsTutorial')
     .factory('TodoService', [function () {
     }]);

// test/unit/components/todo/todo.service.js
describe('TodoService', function(){
     var TodoService;

     beforeEach(module('angularjsTutorial'));

     // underscore-wrap the service name for the injection
```

And now, exercise