Angular creating directives

Angular has a strong templating engine with many core directives, directives allows us to add custom html tags, custom html attributes, custom html comments or custom html classes, whenever angular compiles the template and render our directive it will add additional logic to the element rendered, angular directive can manipulate the DOM, add certain behaviour to DOM element communicate with the parent controller/directive and more.
It's highly recommended to split complex pages in smaller components by using directives.
In this tutorial we will cover the main things you need to know when creating your own directives.

Bootstrap your angular app

first we will start a new angular project, create the root directory of your project and cd into that directory and in your terminal type:
> bower init
> bower install angular --save

The first command will create the **bower.json** file and the second one will install angular and place it inside the **bower_components** directory and also place the package version in the **bower.json** file. In case you want to read more about **bower** you can follow our tutorial in this link

Create an **index.html** file with the following HTML:

```
<!DOCTYPE html>
<html ng-app="AppModule">
<head>
  <meta charset="UTF-8">
  <title>Custom Directive</title>
</head>
<body>
  <script src="bower_components/angular/angular.js"></script>
  <script src="src/app.module.js"></script>
</body>
</html>
```

We are simply loading **angular** script and our **app.module.js** script.

Create a folder called **src** and place the file **app.module.js** in that folder with the following code:

```
angular.module('AppModule', []);
```

That's the general bootstrap of our app, let's continue with exploring angular directive api.

Directive name

In our JS code we name our directive in a camel case format, and when angular tries to find a match in the template it looks for the name as dash delimited name. It's also highly recommended to prefix your directive which could avoid collision on existing HTML tags or future tags, similar to what angular is doing by prefixing the core directives with **ng** prefix you should do the same only choose a different prefix and not **ng**. At Nerdeez our naming convention is prefixing our directives with **nz**.
Let's start with a simple directive called **nzHelloWorld** which will greet us with a lovely hello world.

Directive template

you create a directive similar to how you create a controller, you first get your **module** and the call the directive function of the **module**, this will attach a directive to our **module.** the directive function gets as first argument the directive name, and as the second argument a factory method that will only run once. The factory should return a dictionary object with certain keys and values.
In your **src** folder create a directory called **directives** and create another folder inside the **directives** folder named: **hello-world** place a file called **hello-world.directive.js** with the following code in the **hello-world** folder:

```
angular.module('AppModule').directive('nzHelloWorld', function(){
  return {
     template: '<h1>Hello world</h1>'
  }
});
```

now to use this directive we need to add the script to load in the **index.html** and also write the proper html tag. Change your **index.html** to look like this:

```
<!DOCTYPE html>
<html ng-app="AppModule">
<head>
  <meta charset="UTF-8">
  <title>Custom Directive</title>
</head>
```

```
<body>
  <nz-hello-world></nz-hello-world>

  <script src="bower_components/angular/angular.js"></script>
  <script src="src/app.module.js"></script>
  <script src="src/directives/hello-world/hello-world.directive.js"></script>
</body>
</html>
```

It's common to place the template in a seperate file, so replace the **template** key with **templateUrl** and place a string '**src/directives/hello-world/hello-world.template.html'**

create the file **hello-world.template.html** inside the **hello-world** directory with the html:
<h1>hello world</h1>

Let''s examine the scope of the current directive we built.

Parent Scope

by default if you don't specify otherwise the directive will take the scope of the parent controller it's under. to test that it takes the parent scope we will now create a controller that will control the **body** of the index.
In the **src** folder create a **controllers** folder, and in the **controllers** folder create a file called **main.controller.js** with the following code:

```
angular.module('AppModule')
  .controller('MainController', function(){
      var self = this;

      self.greeting = 'hello from main controller';
  });
```

now in the index file we will want to load the script of the controller and also assign this controller to handle the **body** with the **ng-conroller** directive.
change the **index.html** to look like this:

```
<!DOCTYPE html>
<html ng-app="AppModule">
<head>
  <meta charset="UTF-8">
  <title>Custom Directive</title>
</head>
```

```
<body ng-controller="MainController as ctrl">
  <nz-hello-world></nz-hello-world>

  <script src="bower_components/angular/angular.js"></script>
  <script src="src/app.module.js"></script>
  <script src="src/directives/hello-world/hello-world.directive.js"></script>
  <script src="src/controllers/main.controller.js"></script>
</body>
</html>
```

we placed a controller and we placed in our **MainController** a property with a greeting message, we want our directive to have access to the greeting variable we placed in the controller, so change the directive template in **hello-world.template.html** to look like this:

```
<h1>
  {{ctrl.greeting}}
</h1>
```

notice that when you launch your app you see the message: **hello from main controller** which means that by default the template has the scope of it's parent.

But what if we want more isolation to our directive and let the directive have a scope of it's own.

Isolated Scope

In the directive factory method we return an object with keys that will explain angular how to compile our directive, one of these keys is called: **scope** and if presented angular will create an isolated scope for our directive. In the file **hello-world.directive.js** place the following:

```
angular.module('AppModule').directive('nzHelloWorld', function(){
  return {
      templateUrl: 'src/directives/hello-world/hello-world.template.html',
      scope: {

      }
  }
});
```

try to run your app now and the greeting message disappears, the directive has a scope of it's own now.

Directives usually have isolated scope but we do sometime want to pass data from parent to directive. Let's see how we can achieve that.

Passing data to directive

Let's try and pass the **ctrl.greeting** property of the controller to the directive and still maintain an isolated scope.
The **scope** key inside the directive returned object takes key values, where the keys map to directive scope variables, and the value map to attributes located on the template.
For example to pass the greeting property from the controller and place it inside the directive scope variable called **greetingDirective** we can do this:

```
angular.module('AppModule').directive('nzHelloWorld', function(){
  return {
     templateUrl: 'src/directives/hello-world/hello-world.template.html',
     scope: {
        greetingDirective: '=greetingAttr'
     }
  }
});
```

and now we need to make sure we have a **greeting-attr** attribute in the directive.

```
<nz-hello-world greeting-attr="ctrl.greeting"></nz-hello-world>
```

also in the template we need to change the variable we are printing to **greetingDirective.**
change the **hello-world.template.html** to look like this:

```
<h1>
  {{greetingDirective}}
</h1>
```

notice that we don't need to prefix the variable with **ctrl** since the **greetingDirective** variable will be placed directly on the **$scope**.

so basically we are adding a scope variable called **greetingDirective** and binding it to the **greetingAttr** which is passed from an attribute when we are placing the directive in our template. Notice that the name of the attribute is also normalized and **greetingAttr** will map at the template for the attribute called **greeting-attr**.

now when you launch your app you should see the greeting message passed from the controller to the template.

in cases where the directive scope variable name is equal to the attribute name you can just place:

```
scope: {
        greetingDirective: '='
}
```

this will look for the attribute **greeting-directive** in the template on the directive element.

Also note that the controller variable **greeting** is now binded to the directive scope variable **greetingDirective** and if the directive change the value or the controller change the variable will change in the controller and directive accordingly.

Link function

sometimes we will require our directive to change the DOM. To change the DOM in our directive we usually use the **link** function. The **link** function has the following signature.

```
function link(scope, element, attrs, controller, transcludeFn) { ... }
```

- scope - is the scope of the directive
- element - is a jqlite wrapped dom element of the directive
- attrs - contain the attributes we are passing to the directive
- controller - contains the directive controller or other controller instances that we specify according to the **require** property - we will cover this later
- transcludeFn - linking function connected to the transclusion scope (out of scope for this tutorial)

Let's try and experiment with the **link** function by doing a small example with each of the argument the **link** function is getting.
the **scope** is similar to the **scope** we know we can attach events to the **scope** place variables and all will be available in our scope, including the binded variables we are getting from the **scope** key we covered before.

let's try to pass argument to the template and pass event to the template.

```
angular.module('AppModule').directive('nzHelloWorld', function(){
  return {
     templateUrl: 'src/directives/hello-world/hello-world.template.html',
     scope: {
        greetingDirective: '=greetingAttr'
     },
     link: function(scope, element, attrs, controller, transcludeFn){
```

```
        // checking the scope
        scope.messageFromLink = 'hello for link function';
        scope.eventFromLink = function eventFromLink(){
            alert('eventFromLink');
        }

    }
  }
});
```

and inside the template of the directive we can access the scope params:

```
<h1>
  {{greetingDirective}}
</h1>

<h4>
  {{messageFromLink}}
</h4>

<button ng-click="eventFromLink();">
  event from link
</button>
```

so the scope allows us to connect variables and events to be used in our template.

Let's try and check the **element** attribute, we will use the **element** attribute to refer to the DOM and make changes to the DOM using **jqlite** . for this example we will add a style to the directive DOM which will color the text in red

add the following line at the bottom of the **link** function:

```
element.attr('style', 'color: red;');
```

Now let's try and check the **attrs** argument we will try to print the attribute we are getting in the **greeting-attr** add the following line at the bottom of the **link** function.

```
console.log(attrs.greetingAttr);
```

you notice that you can access the string value of the attribute through the **attrs** argument.

We will cover the **controllers** argument later in this tutorial.

Question: You will notice that we didn't put the scope argument with a dollar sign. Can you guess why? Convention wise in angular what does it mean if a function call is marked with arguments with a dollar sign?

Transclude

Let's try and create a simple carousel component, our carousel component contains slides. We want the slides to be passed from the parent that uses the directive to the directive (slides can be dynamic and change in structure so the structure depends on the parent), meaning we want to pass html to the directive, and the scope of each slide we want to remain in the parent.
To demonstrate what we want , in our **MainController** add the following:

```
self.slides = [
  {
    title: 'slide1 title',
    description: 'Lurem Ipsum ... '
  },
  {
    title: 'slide2 title',
    description: 'Lurem Ipsum ... '
  },
  {
    title: 'slide3 title',
    description: 'Lurem Ipsum ... '
  }
];
```

in the **index.html** change the code to look like this:

```
<!DOCTYPE html>
<html ng-app="AppModule">
<head>
  <meta charset="UTF-8">
  <title>Custom Directive</title>
</head>
<body ng-controller="MainController as ctrl">
  {{ctrl.greeting}}
  <nz-hello-world greeting-attr="ctrl.greeting">
    <div class="slide" ng-repeat="slide in ctrl.slides">
      <div class="slide-header">
        {{slide.title}}
      </div>
      <div class="slide-description">
```

```
        {{slide.description}}
      </div>
    </div>
  </nz-hello-world>

  <script src="bower_components/angular/angular.js"></script>
  <script src="src/app.module.js"></script>
  <script src="src/directives/hello-world/hello-world.directive.js"></script>
  <script src="src/controllers/main.controller.js"></script>
</body>
</html>
```

notice that our directive now has a template between the opening tag and closing tag, try and run your app now and you should see that the entire code that we placed between the opening and closing tag of the directive is gone.
Angular renders the template of the directive and place the rendered template between the opening and closing tags and it removes any other HTML that is located there.
If we want to grab the HTML passed we need to use the **transclude** option.

Add the following to the **hello-world.directive.js**

```
angular.module('AppModule').directive('nzHelloWorld', function(){
  return {
    templateUrl: 'src/directives/hello-world/hello-world.template.html',
    scope: {
        greetingDirective: '=greetingAttr'
    },
    transclude: true,
    link: function(scope, element, attrs, controller, transcludeFn){
        ...
    }
  }
});
```

we placed the **transclude true** option, we will now have to tell angular where to inject that HTML so in the **hello-world.template.html**
place the following:

```
<h1>
  {{greetingDirective}}
</h1>

<h4>
```

```
  {{messageFromLink}}
</h4>

<button ng-click="eventFromLink();">
  event from link
</button>

<div class="slides" ng-transclude></div>
```

In the template we are placing the **ng-transclude** directive to tell angular where to place the html content.
notice that the content in the **transclude** has the **scope** of the parent and not the scope of the directive.

Binding parent function to directive action

There are times where we would like to pass a function call from the parent **MainController** to the **scope** of our directive, to bind function calls it's best to use the **&attr** syntax which will mean running the function of the parent from the context of the parent and not the context of the directive, this is the convention when we want our directive to accept functions.

Let's create a function in our **MainController** called print message, add this to the bottom of the controller:

```
self.printMessage = function printMessage(message){
  console.log(message);
}
```

now let's pass this function to our directive in the **index.html** change the directive call to look like this:

```
<nz-hello-world greeting-attr="ctrl.greeting"
print-message="ctrl.printMessage(message)">
        ...
</nz-hello-world>
```

now in the file **hello-world.directive.js** add the following changes:

```
angular.module('AppModule').directive('nzHelloWorld', function(){
  return {
    templateUrl: 'src/directives/hello-world/hello-world.template.html',
    scope: {
```

```
        greetingDirective: '=greetingAttr',
        printMessage: '&'
    },
    transclude: true,
    link: function(scope, element, attrs, controller, transcludeFn){

        // checking the scope
        scope.messageFromLink = 'hello for link function';
        scope.eventFromLink = function eventFromLink(){
            alert('eventFromLink');
        }

        element.attr('style', 'color: red;');

        console.log(attrs.greetingAttr);

    }
  }
});
```

now in the **hello-world.template.html** at the bottom of the template we will add a button that calls the function in the controller and places a message to that function.

```
<button ng-click="printMessage({message: 'this is from the directive'})">
  bind by refrence
</button>
```

It is custom for directive to pass api connected functions this way and also if you don't call functions like this and use the **=attr** in the scope then you will lose the context of the controller, you can check it by refrencing **this** after you call the function

Controller

another option you can pass in your directive is **controller** which get an injected function. In directives most of the logic is placed in the **link** function, we usually place api we want to expose to child directives in the controller and then we can use the **require** option to state that we want refrence to parent controller and we will get the controllers we want in the link function

We will now add a child directive in the hello world directive called: **child.directive.js** this directive will have a button which we will use to call a method in the **hello-world.directive.js**

in the **hello-world** folder add a file called: **child.directive.js**

```
angular.module('AppModule').directive('nzChild', function(){
  return {
     require: '^nzHelloWorld',
     template: '<button ng-click="passToParent()">pass to parent</button>',
     link: function(scope, element, attrs, controllers){
        scope.passToParent = controllers.callMeFromChild;
     }
  }
});
```

make sure to add the script to load in the **index.html** file, then in your **hello-world.directive.js** add a controller to the directive:

```
angular.module('AppModule').directive('nzHelloWorld', function(){
  return {
     templateUrl: 'src/directives/hello-world/hello-world.template.html',
     scope: {
        greetingDirective: '=greetingAttr',
        printMessage: '&',
        printMessage2: '='
     },
     transclude: true,
     link: function(scope, element, attrs, controller, transcludeFn){
        ...
     },
     controller: function(){
        var self = this;

        self.callMeFromChild = function(){
           alert('called from the child');
        }

     }
  }
});
```

we added a controller and added a function to that controller called **callMeFromChild**.
we get a refrence to that controller in the child directive when we added the **require: '^nzHelloWorld'** and we attached the function from the controller to the scope of the child. You can also pass array in the require and get multiple controllers

Student exercise

- in the tasks list page create the following directive
    - place the form and the creation of task logic in the directive: **create-task.directive.js**
    - place the search for tasks in a directive **search-task.directive.js**
    - in the task list iterate and create a directive for a single task: **single-task.directive.js**

Summary

Directives are a great feature of the angular framework, whenever you have a complex page we recommend to split that page to directive component, you will be surprised how easier your app will be to test.