**Introduction to Directives**

Before introducing Directives , we should know brief about HTML.As web developers, we’re all familiar with HTML. Let’s take a moment to review and synchronize. Our terminology around this most fundamental of web technologies.

**HTML Document**

An HTML document is a plain text document that contains structure and may be styled through

CSS or manipulated with JavaScript.

**HTML Node**

An HTML node is an element or chunk of text nested inside another element. All elements are also nodes; however, a text node is not an element.

**HTML Element**

An element comprises an opening tag and a closing tag.

**HTML Tag**

An HTML tag is responsible for marking the beginning and end of an element. A tag itself is declared using angle brackets.

An opening tag contains a name that becomes the name of the element. It can also contain attributes, which decorate the element.

**Attributes**

To provide additional information about an element, HTML elements can contain attributes. These attributes are always set in the opening tag. We can set them in a key-value pair, like key="value", or as only a key.

<a href="http://interviewGuly.com">Click me to go to InterviewGully</a>

The <a> tag defines a link between another page on our site or off our site, depending on the contents of the href attribute, which defines the link’s destination.

In summary, we can say that attributes add some behaviors in HTML element.

**What is a directive?**

Directive is used to provide extra functionality on the HTML element.

For instance, the ng-click directive gives an element the ability to listen for the click event and run an Angular expression when it receives the event.

Directives are what makes the Angular framework so powerful, and, as we’ve seen, we can also create them.

A directive is defined using the .directive() method, one of the many methods available on our

applications Angular module.

The directive() method takes two arguments:

**name (string)**

The name of the directive as a string that we’ll refer to inside of our views.

**factory\_function (function)**

The factory function returns an object that defines how the directive behaves. It is expected to return an object providing options that tell the $compile service how the directive should behave when it is invoked in the DOM.

In Angularjs, The HTML attributes and binding controls are called directives.  
  
All HTML attributes attached with prefix "ng-". The directives are  
  
Basic Directives

1. ng-app : nitializes to application
2. ng-init : initialize to application data
3. ng-model : binds HTML controls to application data
4. ng-Controller : Attached a controller class to view.

Others then basic Directives

1. ng-repeat : bind to repeated an HTML elements
2. ng-if : bind an HTML elements with condition
3. ng-grid : bind data collection to an HTML elements
4. ng-show : Used to Show the HTML elements
5. ng-hide : Used to Hide the HTML elements
6. ng-class : CSS binding class
7. ng-src : Used to pass the URL image etc.
8. ng-switch
9. ng-bind etc.

Event Listener Directives

1. ng-click : This is a click event to bind on HTML elements
2. ng-dbl-click
3. ng-mousedown
4. ng-mouseup
5. ng-mouseenter
6. ng-mouseleave
7. ng-mousemove
8. ng-mouseover
9. ng-keydown
10. ng-keyup
11. ng-keypress
12. ng-change

Types of Directive

1. Element directives
2. Attribute directives
3. CSS class directives
4. Comment directives

**The possible options are shown below**

angular.module(‘IG’, [])

.directive('myDirective', **function**() {

**return** {

restrict: String,

priority: Number,

terminal: Boolean,

template: String or Template Function:

**function**(tElement, tAttrs) (...},

templateUrl: String,

replace: Boolean or String,

scope: Boolean or Object,

transclude: Boolean,

controller: String or

**function**(scope, element, attrs, transclude, otherInjectables) { ... },

controllerAs: String,

require: String,

link: **function**(scope, iElement, iAttrs) { ... },

compile: **return** an Object OR

**function**(tElement, tAttrs, transclude) {

**return** {

pre: **function**(scope, iElement, iAttrs, controller) { ... },

post: **function**(scope, iElement, iAttrs, controller) { ... }

}

*// or*

**return function** postLink(...) { ... }

}

};

});

We can also return a function instead of an object to handle this directive definition, but it is best practice to return an object as we’ve done above. When a function is returned, it is often referred to as the postLink function, which allows us to define the link function for the directive. Returning a function instead of an object limits us to a narrow ability to customize our directive and, thus, is good for only simple directives.

**Bootstrapped HTML**

When the browser loads our HTML page along with Angular, we only need one snippet of code to boot our Angular application (we learned about it in the introductory chapter).

In our HTML we need to mark up the root of our app using the built in directive ng-app. This

Directive is meant to be used as an attribute; thus, we could stick it anywhere, but let’s choose the opening <html> tag.

<!DOCTYPE html>

<html ng-app="IG">

<head lang="en">

</head>

<body>

</body>

</html>

**Our First Custom Directive**

The quickest way to get our feet wet is to just dive right in. Let’s go ahead and create a very basic custom directive. Consider the following HTML element, which we’ll define in a moment:

**<my-directive></my-directive>**

Provided we’ve created an HTML document and included Angular as well as the ng-app directive in the DOM to mark the root of our app, when Angular compiles our HTML, it will invoke this directive.

The myDirective directive definition looks like:

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

app.directive('myDirective', function() {

return {

restrict: 'E',

replace: true,

template: '<a href="http://interviewgully.com">Click me to go to InterviewGully</a>'

}

});

<!DOCTYPE html>

<html ng-app="IG">

<head lang="en">

<meta charset="UTF-8">

<title>Directives in AngularJS</title>

<script

src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.6/angular.js">

</script>

<script src="Scripts/external.js"></script>

</head>

<body>

<my-directive></my-directive>

</body>

</html>

Simple directive in action With the .directive() method, provided by the Angular module API, we can register new directives by providing a name as a string and function. The name of the directive should always be pascalCased, and the function we provide should return an object.

**Passing Data into a Directive**

Let’s recall our directive definition:

app.directive('myDirective', function() {

return {

restrict: 'E',

replace: true,

template: '<a href="http://interviewgully.com">Click me to go to InterviewGully</a>'

}

});

Notice that in our template we are hard coding the URL and the text of our link:

template: '<a href="http://interviewgully.com">Click me to go to InterviewGully</a>'

With Angular, we aren’t limited to hard coding strings in the template of our directive.

We can provide a nicer experience for others using our directive if we specify the URL and link

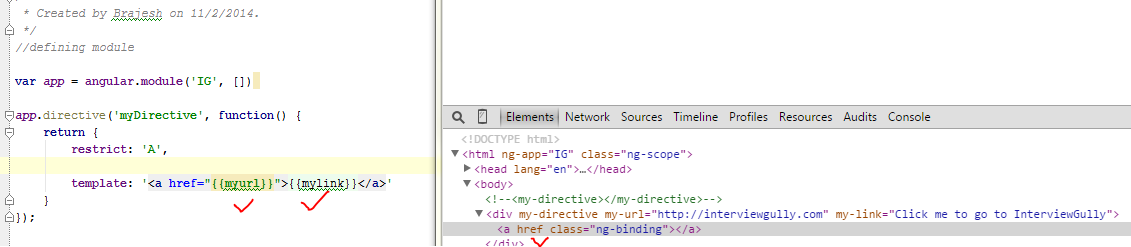
Text without messing with the internal guts of the directive. Our goal here is to pay attention to the public interface of our directive, just as we would in any programming language. In essence, we’d like to turn the above template string into one that takes two variables: one for the URL and one for the link’s text:

template: '<a href="{{myUrl}}">{{myLink}}</a>'

Looking at our main HTML document, we can declare our directive with attributes that will become the properties myUrl and mylink, set on the inner scope of our directive:

<Div my-directive my-Url="http://interviewgully.com" my-Link= "Click me to go to InterviewGully" > </Div>

Reload the page and notice that the div where we declared our directive has been replaced by its template; however, the links href is empty, and there is no text inside the brackets.

<!DOCTYPE html>

<html ng-app="IG">

<head lang="en">

<meta charset="UTF-8">

<title>Directives in AngularJS</title>

<script src="Scripts/Vendor/angular.js"></script>

<script src="Scripts/external.js"></script>

</head>

<body ng-controller="FirstController">

<!--<my-directive></my-directive>-->

<my-directive my-url="www.interviewgully.com" my-link= "ClickInterviewGully" ></my-directive>

</body>

</html>

//defining module

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

app.directive('myDirective', function() {

"use strict";

return {

restrict: 'EA',

template: '<a></a>',

link: function (scope, element, attr, ctrl) {

var myUrl, myLink, handler;

handler = element.find('a')[0];

console.log(attr);

handler.style.backgroundColor = "Red";

handler.setAttribute("href", attr.myUrl);

handler.setAttribute("title", attr.myLink);

handler.innerHTML = attr.myLink;

}

}

});

**Isolate Scope**

Isolate scope is the most confusing of the three options available when setting the scope

Property, but also the most powerful. Isolate scope is based on the ideology present in Object

Oriented Programming and design principles like SOLID have found their way into Angular via directives that use isolate scope.

The main reason for isolation is to increase reusability.

To create a directive with isolate scope we’ll need to set the **scope property** of the directive to an empty object, {}. Once we’ve done that, no outer scope is available in the template of the directive.

app.directive('myDirective', function() {

"use strict";

return {

restrict: 'EA',

scope: {

},

template: '<div>{{fullname}}</div>',

link: function(scope,element,attr,ctrl){

scope.fullname = attr.fullname;

}

}

});

There are three types of isolate scope binding methods in Angularjs.

1. @
2. =
3. &
4. Use of @ 🡺 it accept only string value and provide one way binding (UI to Directive).This operator is used for reading an attribute

<!DOCTYPE html>

<html ng-app="IG">

<head lang="en">

<meta charset="UTF-8">

<title>Directives in AngularJS</title>

<script src="Scripts/Vendor/angular.js"></script>

<script src="Scripts/external.js"></script>

</head>

<body ng-controller="FirstController">

<!--<my-directive></my-directive>-->

<my-directive fullname="Brajesh Kumar Yadav" ></my-directive>

</body>

</html>

//defining module

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

app.directive('myDirective', function() {

"use strict";

return {

restrict: 'EA',

scope: {

},

template: '<div>{{fullname}}</div>',

link: function(scope,element,attr,ctrl){

scope.fullname = attr.fullname;

}

}

});

app.controller('FirstController',['$scope', function($scope){

}])

Instead of the above code we use to optimized as below

app.directive('myDirective', function() {

"use strict";

return {

restrict: 'EA',

scope: {

fullname : '@'

},

template: '<div>{{fullname}}</div>'

}

});

1. Use of = 🡺 It work with object instead of string and provide two way binding (UI to Directive and directive to UI).

<!DOCTYPE html>

<html ng-app="IG">

<head lang="en">

<meta charset="UTF-8">

<title>Directives in AngularJS</title>

<script src="Scripts/Vendor/angular.js"></script>

<script src="Scripts/external.js"></script>

</head>

<body ng-controller="FirstController">

<!--<my-directive></my-directive>-->

ctrl <br>

<input type="text" ng-model="Name" />

<br><br><br>

directive <br>

<my-directive fullname=Name ></my-directive>

</body>

</html>

//defining module

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

app.directive('myDirective', function() {

"use strict";

return {

restrict: 'EA',

scope: {

fullname : '='

},

template: '<div><input type="text" ng-model="fullname" /></div>'

}

});

app.controller('FirstController',['$scope', function($scope){

$scope.Name = "InterviewGully";

}])

1. Use of & 🡺 It is use to invoke an expression or you can say evaluate expression.This operator is used to make a call on controller scope

<!DOCTYPE html>

<html ng-app="IG">

<head lang="en">

<meta charset="UTF-8">

<title>Directives in AngularJS</title>

<script src="Scripts/Vendor/angular.js"></script>

<script src="Scripts/external.js"></script>

</head>

<body ng-controller="FirstController">

<!--<my-directive></my-directive>-->

<my-directive clickhere="clickMe()" ></my-directive>

</body>

</html>

//defining module

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

app.directive('myDirective', function() {

"use strict";

return {

restrict: 'EA',

scope: {

clickhere : '&'

},

template: '<div ng-click="clickhere()">InterviewGully</div>'

}

});

app.controller('FirstController',['$scope', function($scope){

$scope.clickMe = function()

{

alert("Hi! Welcome");

}

}])