

AngularJS

Notes for Professionals

Chapter 3: Components

Parameter
for using two-way data binding. This means that if you update that component scope, the change will be reflected on the parent scope and vice versa. One-way bindings when we just want to read a value from a parent scope.

LifeCycle Hooks

ngOnInit
Called on each controller after all the controllers on an element and their bindings have been initialized. This is a good place to put your initialization logic.

ngOnChanges(changedData)
Called whenever one-way bindings are updated. The changes are the names of the bound properties that have changed. If you are using two-way data binding, this hook is called before the digest cycle.

ngOnDestroy
Called on a controller when its containing scope is destroyed, releasing external resources, watches and event handlers.

ngPostLink
Called after this controller's element and its children have been linked to the DOM.

ngDoCheck
Called on each turn of the digest cycle. Provides an opportunity to perform additional checks on the controller's state. Any actions that you wish to take in response to changes must be invoked from this hook; implementing this hook is optional.

Section 3.1: Basic Components and Lifecycle

What's a component?
A component is basically a directive that uses a simpler configuration and based architecture, which is what Angular 2 is all about. Think of a component as a piece of code that you can reuse in several different places in your web application.

Component

```
angular.module('myApp', [])  
  .component('helloWorld', {  
    templateUrl: 'app/views/helloWorld.html'  
  });
```

Markup

```
<div ng-app="myApp">  
  <hello-world>  
</div>
```

Live Demo

Using External data in Component:
We could add a parameter to pass a name to our component, which would be used as follows:

Chapter 4: Built-in directives

Section 4.1: Angular expressions - Text vs. Number

This example demonstrates how Angular expressions are evaluated when using type="text" and type="number" for the input element. Consider the following controller and view:

Controller

```
var app = angular.module('app', []);  
app.controller('ctrl', function($scope) {  
  $scope.textInput = {  
    value: 5  
  };  
  $scope.numberInput = {  
    value: 5  
  };  
});
```

View

```
<div ng-app="app" ng-controller="ctrl">  
  <input type="text" ng-model="textInput.value">  
  <input type="number" ng-model="numberInput.value">  
</div>
```

- When using `+` in an expression bound to text input, the operator will concatenate the strings (first example), displaying 55 on the screen.
- When using `+` in an expression bound to number input, the operator return the sum of the numbers (second example), displaying 10 on the screen.

That is until the user changes the value in the input field, afterward the display will change accordingly.

Working Example

Section 4.2: ngIf

`ngIf` is a directive similar to `ngShow` but inserts or removes the element from the DOM instead of simply hiding it. Angular 1.3.5 introduced `ngIfElse` directive. You can use `ngIfElse` directive above 1.1.5 versions. This is useful because Angular will not process digests for elements inside a removed `ngIf` reducing the workload of Angular especially for complex data bindings.

Unlike `ng-show`, the `ng-if` directive creates a child scope which uses prototypical inheritance. This means that a primitive value on the child scope will not apply to the parent. To set a primitive on the parent scope the property on the child scope will have to be used.

JavaScript

```
angular.module('myApp', [])  
angular.module('myApp').controller('myController', ['$scope', '$rootScope', function  
myController($scope, $rootScope) {  
  $scope.currentUser = $rootScope.localStorage.getItem('username');  
});
```

AngularJS Notes for Professionals

Chapter 5: Use of in-built directives

Section 5.1: Hide/Show HTML Elements

This example hide show HTML elements.

```
<doctype html>  
<html ng-app="ngDemoApp">  
  <head>  
    <script src="https://code.angularjs.org/1.3.0/angular.min.js"></script>  
    <script>  
      function hideShowController() {  
        var vm = this;  
        vm.show = false;  
        vm.toggle = function() {  
          vm.show = !vm.show;  
        };  
      }  
    </script>  
  </head>  
  <body>  
    <div ng-controller="hideShowController as vm">  
      <input type="checkbox" ng-model="vm.show" /> Show / Hide  
    </div>  
  </body>  
</html>
```

```
angular.module('ngDemoApp', [/* module dependencies go here */])  
  .controller('hideShowController', [hideShowController]);
```

```
<div ng-controller="hideShowController as vm">  
  <input type="checkbox" ng-model="vm.show" /> Show / Hide  
</div>
```

Live Demo

Step by step explanation:

- `ng-app="ngDemoApp"`, the `ngApp` directive tells angular that a DOM element is controlled by a specific `angular.module` named "ngDemoApp".
- `script src="https://code.angularjs.org/1.3.0/angular.min.js"></script>` - include angular.js.
- `hideShowController` function is defined containing another function named `toggle` which help to hide show the element.
- `angular.module(...)` creates a new module.
- `hideShowController` and returns the module for chaining.
- `ng-controller="hideShowController"` is key aspect of how angular supports the principles behind the Model View Controller design patterns.
- `ng-show` directive shows the given HTML element if expression provided is true.
- `ng-hide` directive hides the given HTML element if expression provided is true.
- `ng-click` directive fires a toggle function inside controller.

AngularJS Notes for Professionals

100+ pages
of professional hints and tricks

Contents

About	1
Chapter 1: Getting started with AngularJS	2
Section 1.1: Getting Started	6
Section 1.2: Showcasing all common Angular constructs	7
Section 1.3: The importance of scope	8
Section 1.4: Minification in Angular	10
Section 1.5: AngularJS Getting Started Video Tutorials	11
Section 1.6: The Simplest Possible Angular Hello World	11
Chapter 2: Modules	13
Section 2.1: Modules	13
Section 2.2: Modules	13
Chapter 3: Components	15
Section 3.1: Basic Components and LifeCycle Hooks	15
Section 3.2: Components In angular JS	17
Chapter 4: Built-in directives	19
Section 4.1: Angular expressions - Text vs. Number	19
Section 4.2: ngIf	19
Section 4.3: ngCloak	20
Section 4.4: ngRepeat	21
Section 4.5: Built-In Directives Cheat Sheet	24
Section 4.6: ngInclude	25
Section 4.7: ng-model-options	25
Section 4.8: ngCopy	26
Section 4.9: ngPaste	26
Section 4.10: ngClick	27
Section 4.11: ngList	27
Section 4.12: ngOptions	28
Section 4.13: ngSrc	30
Section 4.14: ngModel	30
Section 4.15: ngClass	31
Section 4.16: ngDbclick	31
Section 4.17: ngHref	32
Section 4.18: ngPattern	32
Section 4.19: ngShow and ngHide	33
Section 4.20: ngRequired	34
Section 4.21: ngMouseenter and ngMouseleave	34
Section 4.22: ngDisabled	34
Section 4.23: ngValue	35
Chapter 5: Use of in-built directives	36
Section 5.1: Hide/Show HTML Elements	36
Chapter 6: Custom Directives	37
Section 6.1: Creating and consuming custom directives	38
Section 6.2: Directive Definition Object Template	39
Section 6.3: How to create reusable component using directive	40
Section 6.4: Basic Directive example	42
Section 6.5: Directive decorator	42
Section 6.6: Basic directive with template and an isolated scope	43

Section 6.7: Building a reusable component	44
Section 6.8: Directive inheritance and interoperability	45
Chapter 7: How data binding works	47
Section 7.1: Data Binding Example	47
Chapter 8: Angular Project - Directory Structure	49
Section 8.1: Directory Structure	49
Chapter 9: Filters	51
Section 9.1: Accessing a filtered list from outside an ng-repeat	51
Section 9.2: Custom filter to remove values	51
Section 9.3: Custom filter to format values	51
Section 9.4: Using filters in a controller or service	52
Section 9.5: Performing filter in a child array	52
Chapter 10: Custom filters	54
Section 10.1: Use a filter in a controller, a service or a filter	54
Section 10.2: Create a filter with parameters	54
Section 10.3: Simple filter example	54
Chapter 11: Constants	56
Section 11.1: Create your first constant	56
Section 11.2: Use cases	56
Chapter 12: Custom filters with ES6	58
Section 12.1: FileSize Filter using ES6	58
Chapter 13: Directives using ngModelController	59
Section 13.1: A simple control: rating	59
Section 13.2: A couple of complex controls: edit a full object	61
Chapter 14: Controllers	64
Section 14.1: Your First Controller	64
Section 14.2: Creating Controllers, Minification safe	65
Section 14.3: Using ControllerAs in Angular JS	66
Section 14.4: Creating Minification-Safe Angular Controllers	67
Section 14.5: Creating Controllers	68
Section 14.6: Nested Controllers	68
Chapter 15: Controllers with ES6	69
Section 15.1: Controller	69
Chapter 16: The Self Or This Variable In A Controller	70
Section 16.1: Understanding The Purpose Of The Self Variable	70
Chapter 17: Services	72
Section 17.1: Creating a service using angular.factory	72
Section 17.2: Difference between Service and Factory	72
Section 17.3: \$sce - sanitize and render content and resources in templates	75
Section 17.4: How to create a Service	75
Section 17.5: How to use a service	76
Section 17.6: How to create a Service with dependencies using 'array syntax'	76
Section 17.7: Registering a Service	77
Chapter 18: Distinguishing Service vs Factory	78
Section 18.1: Factory VS Service once-and-for-all	78
Chapter 19: Angular promises with \$q service	80
Section 19.1: Wrap simple value into a promise using \$q.when()	80
Section 19.2: Using angular promises with \$q service	80

Section 19.3: Using the \$q constructor to create promises	82
Section 19.4: Avoid the \$q Deferred Anti-Pattern	83
Section 19.5: Using \$q.all to handle multiple promises	84
Section 19.6: Deferring operations using \$q.defer	85
Chapter 20: Dependency Injection	86
Section 20.1: Dynamic Injections	86
Section 20.2: Dynamically load AngularJS service in vanilla JavaScript	86
Chapter 21: Events	87
Section 21.1: Using angular event system	87
Section 21.2: Always deregister \$rootScope.\$on listeners on the scope \$destroy event	89
Section 21.3: Uses and significance	89
Chapter 22: Sharing Data	92
Section 22.1: Using ngStorage to share data	92
Section 22.2: Sharing data from one controller to another using service	92
Chapter 23: Form Validation	94
Section 23.1: Form and Input States	94
Section 23.2: CSS Classes	94
Section 23.3: Basic Form Validation	94
Section 23.4: Custom Form Validation	95
Section 23.5: Async validators	96
Section 23.6: ngMessages	96
Section 23.7: Nested Forms	97
Chapter 24: Routing using ngRoute	98
Section 24.1: Basic example	98
Section 24.2: Defining custom behavior for individual routes	99
Section 24.3: Route parameters example	100
Chapter 25: ng-class directive	102
Section 25.1: Three types of ng-class expressions	102
Chapter 26: ng-repeat	104
Section 26.1: ng-repeat-start + ng-repeat-end	104
Section 26.2: Iterating over object properties	104
Section 26.3: Tracking and Duplicates	105
Chapter 27: ng-style	106
Section 27.1: Use of ng-style	106
Chapter 28: ng-view	107
Section 28.1: Registration navigation	107
Section 28.2: ng-view	107
Chapter 29: AngularJS bindings options (=, @, & etc.)	109
Section 29.1: Bind optional attribute	109
Section 29.2: @ one-way binding, attribute binding	109
Section 29.3: = two-way binding	109
Section 29.4: & function binding, expression binding	110
Section 29.5: Available binding through a simple sample	110
Chapter 30: Providers	111
Section 30.1: Provider	111
Section 30.2: Factory	111
Section 30.3: Constant	112
Section 30.4: Service	112
Section 30.5: Value	113

Chapter 31: Decorators	114
Section 31.1: Decorate service, factory	114
Section 31.2: Decorate directive	114
Section 31.3: Decorate filter	115
Chapter 32: Print	116
Section 32.1: Print Service	116
Chapter 33: ui-router	118
Section 33.1: Basic Example	118
Section 33.2: Multiple Views	119
Section 33.3: Using resolve functions to load data	120
Section 33.4: Nested Views / States	121
Chapter 34: Built-in helper Functions	123
Section 34.1: angular.equals	123
Section 34.2: angular.toJson	123
Section 34.3: angular.copy	124
Section 34.4: angular.isString	124
Section 34.5: angular.isArray	124
Section 34.6: angular.merge	125
Section 34.7: angular.isDefined and angular.isUndefined	125
Section 34.8: angular.isDate	126
Section 34.9: angular.noop	126
Section 34.10: angular.isElement	126
Section 34.11: angular.isFunction	127
Section 34.12: angular.identity	127
Section 34.13: angular.forEach	128
Section 34.14: angular.isNumber	128
Section 34.15: angular.isObject	128
Section 34.16: angular.fromJson	129
Chapter 35: digest loop walkthrough	130
Section 35.1: \$digest and \$watch	130
Section 35.2: the \$scope tree	130
Section 35.3: two way data binding	131
Chapter 36: Angular \$scopes	133
Section 36.1: A function available in the entire app	133
Section 36.2: Avoid inheriting primitive values	133
Section 36.3: Basic Example of \$scope inheritance	134
Section 36.4: How can you limit the scope on a directive and why would you do this?	134
Section 36.5: Using \$scope functions	135
Section 36.6: Creating custom \$scope events	136
Chapter 37: Using AngularJS with TypeScript	138
Section 37.1: Using Bundling / Minification	138
Section 37.2: Angular Controllers in Typescript	138
Section 37.3: Using the Controller with ControllerAs Syntax	140
Section 37.4: Why ControllerAs Syntax?	140
Chapter 38: \$http request	142
Section 38.1: Timing of an \$http request	142
Section 38.2: Using \$http inside a controller	142
Section 38.3: Using \$http request in a service	143
Chapter 39: Prepare for Production - Grunt	145

[Click here to download full PDF material](#)