



## Lesson Objectives

- Directives
- Built-In Directives
- Creating Custom Directives
- Digest Cycle



## 2.1 Directives Introduction

## Directives

- Directives are ways to transform the DOM through extending HTML and provides new functionality to it
- As a good practice DOM manipulations need to be done in directives
- Directive is simply a function that we run on a particular DOM element (such as an attribute, element name, comment or CSS class) that tell AngularJS's HTML compiler (\$compile) to attach a specified behavior to that DOM element or even transform the DOM element and its children
- Directives are defined with camelCase in the JavaScript, but applied with a dash to the HTML
- Angular comes with a set of built-in directives.
- It allows to create our own directives



Copyright © Capgemini 2015. All Rights Reserved 3

Angular normalizes an element's tag and attribute name to determine which elements match which directives. We typically refer to directives by their case-sensitive camelCase normalized name (e.g. ngModel). However, since HTML is case-insensitive, we refer to directives in the DOM by lower-case forms, typically using dash-delimited attributes on DOM elements (e.g. ng-model).

The normalization process is as follows:

Strip x- and data- from the front of the element/attributes. Convert the :, -, or \_-delimited name to camelCase. Here are some equivalent examples of elements that match ngBind:

- ☐ ng-bind
- ☐ ng:bind
- ☐ ng\_bind
- ☐ data-ng-bind
- ☐ x-ng-bind

## 2.1 Directives Introduction

## Directives

- Angular directive can be specified in 3 ways
- As a tag
  - `<ng-form />`
- As an attribute
  - `<div ng-form />`
- As a class
  - `<div class="ng-form"/>`
- All the in-built directives in angular cannot be specified with all the 3 ways, some of them can be specified in 1 or 2 ways only

## 2.2 Built-In Directives

## Built-In Directives

- Angular provides a suite of built-in directives

Directives	Descriptions
ng-app	It is added to set the AngularJS section.
ng-init	It sets default variable value.
ng-bind	It is an alternative to {{ }} template.
ng-bind-html	It is used to bind innerHTML property of an HTML element.
ng-checked	It is used to set checkbox checked.
ng-controller	It is used to attach a controller class to the view.
ng-class	It is used to the css class dynamically.



Copyright © Capgemini 2015. All Rights Reserved 5

**ngApp**

Placing ng-app on any DOM element marks that element as the beginning of the \$rootScope.

\$rootScope is the beginning of the scope chain, and all directives nested under the ng-app in your HTML inherit from it.

\$rootScope can be accessed via the run method

Using \$rootScope is like using global scope hence it is not a best practice.

We can use ng-app once per document.

**ngInit**

The ngInit directive is used to set up the state inside the scope of a directive when that directive is invoked.

It is a shortcut for using ng-bind without needing to create an element; therefore, it is most commonly used with inline text

**ngBind**

The ngBind attribute tells Angular to replace the text content of the specified HTML element with the value of a given expression and to update the text content when the value of that expression changes.

It is preferable to use ngBind instead of {{ expression }}. If a template is momentarily displayed by the browser in its raw state before Angular compiles it. Since ngBind is an element attribute, it makes the bindings invisible to the user while the page is loading.

**ngBindHtml**

Creates a binding with innerHTML so that the result of evaluating the expression into the current element comes in a secure way.

ngSanitize (angular-sanitize.js) need to be included as module's dependencies to evaluate in a secure way, else it will throw error "Attempting to use an unsafe value in a safe context".

We can bypass sanitization by binding to an explicitly trusted value via `$sce.trustAsHtml`.

**ngClass**

ngClass directive allows you to dynamically set CSS classes on an HTML element by data binding an expression that represents all classes to be added.

**ngController**

This directive is used to place a controller on a DOM element.

Instead of defining actions and models on \$rootScope, use ng-controller

## 2.2 Built-In Directives

## Built-In Directives

Directives	Descriptions
ng-href	It is used to dynamically bind Angular JS variables to the href attribute.
ng-include	It is used to fetch, compile and include an external HTML fragment to your page.
ng-if	It is used to remove or recreate an element in the DOM depending on an expression
ng-switch	It is used to conditionally switch control based on matching expression.
ng-repeat	It is used to loop through each item in collection to create a new template.



Copyright © Capgemini 2015. All Rights Reserved 6

**ngHref**

Angular waits for the interpolation to take place and then activates the link's behavior.  
It is recommended to use `ng-href` in place of href.

**ngInclude**

ngInclude directive is used to fetch, compile and include an external HTML fragment into your current application.  
By default, the template URL is restricted to the same domain and protocol as the application document. This is done by calling `$sce.getTrustedResourceUrl` on it.  
The URL of the template is restricted to the same domain and protocol as the application document unless white listed or wrapped as trusted values.  
To access file locally in chrome use `chrome.exe -allow-file-access-from-files -disable-web-security`

**ngIf**

ng-if directive is used to completely remove or recreate an element in the DOM based on an expression. If the expression assigned to ng-if evaluates to a false value, then the element is removed from the DOM, otherwise a clone of the element is reinserted into the DOM.  
Using ng-if when an element is removed from the DOM, its associated scope is destroyed. when it comes back into being, a new scope is created

**ngSwitch**

The ngSwitch directive is used to conditionally swap DOM structure on your template based on a scope expression  
It is used in conjunction with ng-switch-when and on="propertyName" to switch which directives render in our view when the given propertyName changes.


**ngRepeat**

ngRepeat directive instantiates a template once per item from a collection. Each template instance gets its own scope, where the given loop variable is set to the current collection item, and \$index is set to the item index or key

2.2 Built-In Directives

## Built-In Directives

Directives	Descriptions
ng-show/ng-hide	It works based on expression, if true then the element is shown or hidden respectively
ng-src	It is used to dynamically bind AngularJS variables to the src attribute.
ng-style	It is used to set CSS style on an HTML element conditionally
ng-classodd/ng-classeven	Take effect only on odd/even rows
ng-form	It is used to the css class dynamically


Copyright © Capgemini 2015. All Rights Reserved

**ngShow**

The ngShow directive shows or hides the given HTML element based on the expression provided to the ngShow attribute.

When the ngShow expression evaluates to a false value then the ng-hide CSS class is added to the class attribute on the element causing it to become hidden. When true, the ng-hide CSS class is removed from the element causing the element not to appear hidden.

**ngHide**

The ngHide directive shows or hides the given HTML element based on the expression provided to the ngHide attribute.

When the ngHide expression evaluates to a truthy value then the .ng-hide CSS class is added to the class attribute on the element causing it to become hidden. When falsy, the ng-hide CSS class is removed from the element causing the element not to appear hidden.

**ngSrc**

Angular will tell the browser not to fetch the image via the given URL until all expressions provided to ng-src have been interpolated.

It is recommended to use `ng-src` in place of `src`.

**ngStyle**

ngStyle directive allows you to set CSS style on an HTML element.

CSS style names and values must be quoted.

**ngClassEven**

Works exactly as ngClass, except they work in conjunction with ngRepeat and take effect only on even row elements i.e. 0,2,4,6 ...

**ngClassOddS**

Works exactly as ngClass, except they work in conjunction with ngRepeat and take effect only on odd row elements i.e. 1,3,5,7...

## 2.2 Built-In Directives

## Built-In Event directives

- When Angular parses the HTML, it look for directive and takes action based on that, when it looks for event directives it register the event on the DOM object.



**Note:** ngChange directive requires the ngModel directive to also be present



Copyright © Capgemini 2015. All Rights Reserved 8

**ngClick:** The ngClick directive allows you to specify custom behavior when an element is clicked.

**ngMousemove:** Expression to evaluate upon mousemove. (Event object is available as \$event)

**ngMouseup** Specify custom behavior on mouseup event.



## Demo

- Angular-01-Built-InDirectives
- Angular-02-Built-InDirectives
- Angular-03-Built-InDirectives
- Angular-04-Built-InDirectives
- Angular-05-Built-InDirectives
- Angular-06-Built-InDirectives



## 2.3 Custom Directives

## Custom Directives

- Directives makes the Angular framework so powerful, we can also create our own directives. A directive is defined using the `.directive()` method on application's Angular module
- Directives can be implemented in the following ways:
  - Element directives : activated when AngularJS finds a matching HTML element in the HTML template
  - Attribute directives : activated when AngularJS finds a matching HTML element attribute
  - CSS class directives : activated when AngularJS finds a matching CSS Class
  - Comment directives : activated when AngularJS finds a matching HTML comment
- AngularJS recommends to use element and attribute directives, and leave the CSS class and comment directives (unless absolutely necessary)



Copyright © Capgemini 2015. All Rights Reserved 10

Much like controllers, directives are registered on modules. To register a directive, we use the `module.directive` API. `module.directive` takes the normalized directive name followed by a **factory function**. This factory function should return an object with the different options to tell \$compile how the directive should behave when matched.

There are many options that can be configured and how those options are related to each other is important. Each directive undergoes something similar to a life cycle as AngularJS compiles and links the DOM. The directive lifecycle begins and ends within the AngularJS bootstrapping process, before the page is rendered. In a directive's life cycle, there are four distinct functions that can execute if they are defined. Each enables the developer to control and customize the directive at different points of the life cycle.

The four functions are: *compile*, *controller*, *pre-link* and *post-Link*.

The **compile** function allows the directive to manipulate the DOM before it is compiled and linked thereby allowing it to add/remove/change directives, as well as, add/remove/change other DOM elements.

The **controller** function facilitates directive communication. Sibling and child directives can request the controller of their siblings and parents to communicate information.

The **pre-link** function allows for private **\$scope** manipulation before the post-link process begins.

The **post-link** method is the primary workhorse method of the directive.

## 2.3 Custom Directives

## Applying restrictions to directives

- Custom directive is restricted to attribute by default
- In order to create directives that are triggered by element, class name & comment we need to use the restrict option
- The restrict option is typically set to:
  - 'A' - only matches attribute name
  - 'E' - only matches element name
  - 'C' - only matches class name
  - 'M' - only matches comment
- These restrictions can also be combined
  - 'AEC' - matches either attribute or element or class name

## 2.3 Custom Directives

## Custom Directives - template

- template is an in-line template specified using html as a string / function which gets appended / replaced (by setting replace:true) within the element where the directive was invoked.
- template has a scope that can be accessed using double curly markup, like {{ expression }}. Backslashes is used at the end of the each line to denote multi line string
- When a template string must be wrapped in a parent element. i.e, a root DOM element must exist.

```
app.directive("helloworldattr",function(){
  return {
    replace:true,
    template: "<h1>Hello World Attribute from
Directive</h1>"
  }
});
```



Copyright © Capgemini 2015. All Rights Reserved 12

When the application bootstraps, Angular starts parsing the DOM using the \$compile service. This service searches for directives in the markup and matches them against registered directives. Once all the directives have been identified, Angular executes their compile functions. compile function returns a link function which is added to the list of link functions to be executed later. This is called the compile phase.

If a directive needs to be cloned multiple times (e.g. ng-repeat), we get a performance benefit as the compile function runs once for the cloned template, but the link function runs for each cloned instance. That's why the compile function does not receive a scope.

After the compile phase is over the linking phase starts, where the collected link functions are executed one by one, starts. This is where the templates produced by the directives are evaluated against correct scope and are turned into live DOM which react to events

## Demo

- Angular-Custom Directives



## Lab

- Lab01-Introduction to Angular JS & Directives



## 2.4 Digest Cycle Overview

## Digest Cycle and \$scope

- Digest cycle can be considered as a loop, during which Angular checks if there are any changes occurred to the variables being watched.
- Angular sets up a watcher on the scope model, which in turn updates the view whenever the model changes.

```
$scope.$watch('modelVariable', function(newValue, oldValue)
{
    //update the DOM with newValue
});
```

- \$digest cycle fires the watchers. When the \$digest cycle starts, it fires each of the watchers. These watchers check whether the current value of the scope model is different from old value. If there is a change, then the corresponding listener function executes. As a result, any expressions in the view get updated



Copyright © Capgemini 2015. All Rights Reserved 15

\$digest cycle starts as a result of a call to \$scope.\$digest(). Angular doesn't directly call \$digest(). Instead, it calls \$scope.\$apply(), which in turn calls \$rootScope.\$digest(). As a result of this, a digest cycle starts at the \$rootScope, and subsequently visits all the child scopes calling the watchers along the way.

AngularJS wraps the function calls (which updates the model) from view within \$scope.\$apply()

The \$apply() function comes in two flavors.

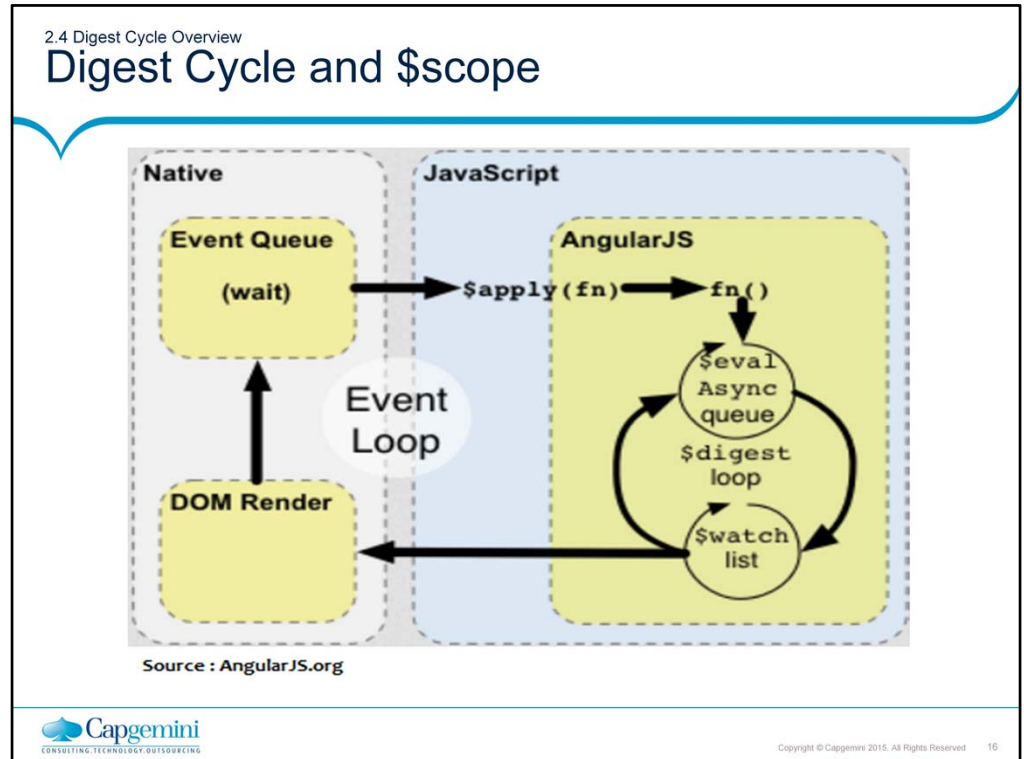
The first one takes a function as an argument, evaluates it, and triggers a \$digest cycle.

The second version does not take any arguments and just starts a \$digest cycle.

Built-in directives/services (like ng-click, ng-repeat, ng-model, \$timeout,\$http etc) which changes the models automatically trigger a \$digest cycle.i.e. It calls \$apply() automatically and creates implicit watches to the model variables.

If we change any model outside of the Angular context, then we need to inform Angular of the changes made by calling \$apply() manually. For instance, if setTimeout() function updates a scope model, Angular won't have any idea about the model change then it becomes our responsibility to call \$apply() manually, which in turn triggers a \$digest cycle.

If a directive that sets up a DOM event listener and changes models inside the handler function, we need to call \$apply() to ensure the changes take effect.



The diagram and the example describe how Angular interacts with the browser's event loop.

1. The browser's event-loop waits for an event to arrive. An event is a user interaction, timer event, or network event (response from a server).
2. The event's callback gets executed. This enters the JavaScript context. The callback can modify the DOM structure.
3. Once the callback executes, the browser leaves the JavaScript context and re-renders the view based on DOM changes.

Angular modifies the normal JavaScript flow by providing its own event processing loop. This splits the JavaScript into classical and Angular execution context. Only operations which are applied in the Angular execution context will benefit from Angular data-binding, exception handling, property watching. We can also use `$apply()` to enter the Angular execution context from JavaScript. Keep in mind that in most places (controllers, services) `$apply` has already been called for you by the directive which is handling the event. An explicit call to `$apply` is needed only when implementing custom event callbacks, or when working with third-party library callbacks.

1. Enter the Angular execution context by calling `scope.$apply(stimulusFn)`, where `stimulusFn` is the work you wish to do in the Angular execution context.
2. Angular executes the `stimulusFn()`, which typically modifies application state.
3. Angular enters the `$digest` loop. The loop is made up of two smaller loops which process `$evalAsync` queue and the `$watch` list. The `$digest` loop keeps iterating until the model stabilizes, which means that the `$evalAsync` queue is empty and the `$watch` list does not detect any changes.
4. The `$evalAsync` queue is used to schedule work which needs to occur outside of current stack frame, but before the browser's view render. This is usually done with `setTimeout(0)`, but the `setTimeout(0)` approach suffers from slowness and may cause view flickering since the browser renders the view after each event.
5. The `$watch` list is a set of expressions which may have changed since last iteration. If a change is detected then the `$watchfunction` is called which typically updates the DOM with the new value.
6. Once the Angular `$digest` loop finishes, the execution leaves the Angular and JavaScript context. This is followed by the browser re-rendering the DOM to reflect any changes.



## Summary

- Scope is not a model actually it contains the model
- We can use JavaScript objects as model in angular
- Double curly brace is the markup indicator for binding data to view
- ngSrc is used to bind and image's src. It delays fetching an image until binding has occurred
- angular directives can be written in three different ways: tag, attribute & class. We cannot write all the inbuilt directives in all the 3 ways
- ngChange directive requires the ngModel directive also to be present



Add the notes here.

## Summary

- ngCloak directive is used to avoid a flash of unbound html
- ngBind does not support multiple bindings where as ngBindTemplate supports multiple bindings
- ngClass directives has two companion directives : ngClassEven & ngClassOdd
- ngForm allow us to Nest forms
- Avoid custom tag name directives to make Angular support with older version of IE
- Two way binding will update on every key stroke. Two way binding is supported by Input, Select and Textarea HTML elements



Add the notes here.

## Summary

- Two way binding will update on every key stroke. Two way binding is supported by Input, Select and Textarea HTML elements
- The property will be created automatically, when we refer a property that doesn't exist in a ngModel directive
- ngPattern directive allow us to create a regex for validation
- form tag should have a name property to check the validity of form
- restrict property of a directive can take the following values : E , A, C and M



Add the notes here.

## Review Question

- ng-model binds the values of AngularJS application data to HTML input controls
  - True
  - False
- On which of the following types of component can we create a custom directive?
  - Element directives
  - Attribute
  - CSS
  - All of the above
- Templates can be a single file (like index.html) or multiple views in one page
  - True
  - False

