Working with Directives

- Introduction & Core Directives
- Conditional Directives
- Styles Directives
- Mouse and Keyboard Events Directives

Introduction

At a high level, directives are markers on a 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.

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:

1. Strip x- and data- from the front of the element/attributes.
2. Convert the :, -, or _-delimited name to camelCase.

For example, the following forms are all equivalent and match the **ngBind** directive:

Eg: In the below example, all will produce same result:

```
<div ng-app>
   Hello <input ng-model='name'> <hr />
   <span ng-bind="name"></span> <br />
   <span ng:bind="name"></span> <br />
   <span ng_bind="name"></span> <br />
   <span data-ng-bind="name"></span> <br />
   <span x-ng-bind="name"></span> <br />
 </div>
```

ng-app directive

- The **ng-app** directive defines the **root element** of an AngularJS application.
- The **ng-app** directive will **auto-bootstrap** (automatically initialize) the application when a web page is loaded.
- Later we will learn how **ng-app** can have a value (like ng-app="myModule"), to connect code modules.

ng-init directive:

- The **ng-init** directive defines **initial values** for an AngularJS application.
- Normally, you will not use ng-init. You will use a controller or module instead.

```
<div ng-app ng-init="qty=1;cost=2">
  <div>
    <b>Total:</b> {{qty * cost | currency}}
  </div>
</div>
```

ng-model directive:

The **ng-model** directive binds the value of HTML controls (input, select, textarea) to application data.

The **ng-model** directive can also:

- Bind HTML elements to HTML forms
- Provide CSS classes for HTML elements.
- Provide status for application data (invalid, dirty, touched, error).
- Provide type validation for application data (number, email, required).

```
<div ng-app="">
  <p>
    First Name: <input type="text" ng-model="fname">
    Last Name: <input type="text" ng-model="lname">
  </p>
  Hello {{fname + " " + lname}}
</div>
```

The ng-bind & ng-bind-template Directives:

*Unlike ngBind, the ngBindTemplate can contain multiple {{ }} expressions. This directive is needed since some HTML elements (such as TITLE and OPTION) cannot contain SPAN elements.

```
<div ng-app>
  FirstName: <input type="text" ng-model="FirstName"><br>
  Name: <input type="text" ng-model="LastName"><br>
  <p>Hello <span ng-bind="FirstName"></span> <span ng-bind="LastName"></span></p>
  <p>Hello <span ng-bind-template="{{FirstName}} {{LastName}}"></span></p>
</div>
```

Condional Directives

ng-show & ng-hide directives:

Conditionally show or hide an element, depending on the value of a boolean expression. Show and hide is achieved by setting the CSS **display** style.

```
<div ng-app>
   <input type="checkbox" name="chkShow" value="" ng-model="blnShow" /> Show Block
   <div ng-Show="blnShow">This is a block</div>
</div>
```

The ng-if directive

Basic if statement directive which allow to show the following element if the conditions are true. When the condition is false, the element is **removed** from the DOM tree. When true, a clone of the compiled element is re-inserted

```
<div ng-app>
   <input type="checkbox" name="chkShow" value="" ng-model="blnShow" /> Show Block
   <div id="div1" ng-show="blnShow">This is a block</div>
   <div id="div2" ng-if="blnShow"> This is If block</div>
   <input type="button" value="Div1 in DOM Tree" onclick="alert(document.getElementById('div1')!=null);">
   <input type="button" value="Div2 in DOM Tree" onclick="alert(document.getElementById('div2')!=null);">
</div>
```

The ng-switch, ng-switch-when, ng-switch-default Directives:

Conditionally instantiate one template from a set of choices, depending on the value of a selection expression.

```
<div ng-app>
   <select ng-model="selection">
      <option value="1">One</option>
      <option value="2">Two</option>
      <option value="3">Three</option>
      <option value="4">Four</option>
      <option value="5">Five</option>
   </select>
   <hr />
   <div ng-switch on="selection">
      <div ng-switch-when="1">This is One</div>
      <div ng-switch-when="2">This is Two</div>
      <div ng-switch-when="3">This is Three</div>
      <div ng-switch-default>This is Default</div>
```

```
  </div>
 </div>
```

The ng-repeat Directive

The **ng-repeat** directive **clones HTML elements** once for each item in a collection (in an array).

```
<div ng-app="" ng-
init="names=[{FirstName:'Shahrukh',LastName:'Kahn'},{FirstName:'Virat',LastName:'Kohli'},{FirstName:'Bruce',Last
Name:'Lee'}]">
  <table>
    <tr ng-repeat="name in names">
      <td>{{name.FirstName}}</td>
      <td>{{name.LastName}}</td>
    </tr>
  </table>
</div>
```

Styles Directives

The ng-class Directives:

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

The directive operates in three different ways, depending on which of three types the expression evaluates to:

1. If the expression evaluates to a **string**, the string should be one or more space-delimited class names.
2. If the expression evaluates to an **array**, each element of the array should be a string that is one or more space-delimited class names.
3. If the expression evaluates to an **object**, then for each key-value pair of the object with a truthy value the corresponding key is used as a class name.

The directive won't add duplicate classes if a particular class was already set.

When the expression changes, the previously added classes are removed and only then the new classes are added.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title></title>
  <script src="angular.js"></script>
  <style>
    .strike {
```

```
      text-decoration: line-through;
    }

    .bold {
      font-weight: bold;
    }
    .red {
      color: red;
    }
  </style>
</head>
<body>
  <div ng-app>
    <p ng-class="{strike: deleted, bold: important, red: error}">Map Syntax Example</p>
    <input type="checkbox" ng-model="deleted"> deleted (apply "strike" class)<br>
    <input type="checkbox" ng-model="important"> important (apply "bold" class)<br>
    <input type="checkbox" ng-model="error"> error (apply "red" class)
    <hr>
    <p ng-class="style">Using String Syntax</p>
    <input type="text" ng-model="style" placeholder="Type: bold strike red">
    <hr>
    <p ng-class="[style1, style2, style3]">Using Array Syntax</p>
    <input ng-model="style1" placeholder="Type: bold, strike or red"><br>
    <input ng-model="style2" placeholder="Type: bold, strike or red"><br>
    <input ng-model="style3" placeholder="Type: bold, strike or red"><br>
  </div>
</body>
</html>
```

The ng-class-even & ng-class-odd Directives

The ngClassOdd and ngClassEven directives work exactly as ngClass, except they work in conjunction with ngRepeat and take effect only on odd (even) rows.

This directives can be applied only within the scope of an ngRepeat.

```
<div ng-app>
  <ol ng-init="names=['John', 'Mary', 'Cate', 'Suz']">
    <li ng-repeat="name in names">
```

```
        <span ng-class-odd="'odd'" ng-class-even="'even'" ng-bind="name"></span>
      </li>
    </ol>
</div>
```

The ng-style Directive

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

Value must be an expression which evaluates to an object whose keys are CSS style names and values are corresponding values for those CSS keys.

Note: Since some CSS style names are not valid keys for an object, they must be quoted. For example: 'background-color'.

```
<input type="button" ng-click="mystyles={color:'red', 'background-color': 'yellow', 'font-weight':'bold'}"
value="Styles1" />
 <input type="button" ng-click="mystyles={color:'green', 'background-color': 'black', 'font-weight':'bold'}"
value="Styles1" />
 <input type="button" ng-click="mystyles={color:'blue', 'background-color': 'white', 'font-weight':'bold'}"
value="Styles1" />
 <div ng-style="mystyles">This is style demo</div>
```

Events related Directives

The ng-click directive

Evaluate the given expression based on action performed:

```
<div ng-app>
   <button ng-click="count = count + 1" ng-init="count=0">
      Increment
   </button>
   <span>
      count: {{count}}
   </span>
 </div>
```

The ng-change directive:

Evaluate the given expression when the user changes the input. The expression is evaluated immediately, unlike the JavaScript onchange event which only triggers at the end of a change (usually, when the user leaves the form element or presses the return key).

The ngChange expression is only evaluated when a change in the input value causes a new value to be committed to the model.

Note: This directive requires ngModel to be present.

```
<body ng-app ng-init="counter=0">
  <input type="text" name="txt1" ng-model="data" ng-change="counter = counter+1" />
  <input type="checkbox" ng-model="data" ng-change="counter = counter+1" />
  <br />
  <span>{{data + " " + counter}}</span>
</body>
```

The ng-checked directive:

Their presence of checked attribute means true and its absence means false. If we put an Angular interpolation expression into such an attribute then the binding information would be lost when the browser removes the attribute.

The ngChecked directive solves this problem for the checked attribute. This complementary directive is not removed by the browser and so provides a permanent reliable place to store the binding information.

```
Check me to check both:
<input type="checkbox" ng-model="master"><br />
<input id="checkSlave" type="checkbox" ng-checked="master">
```

Mouse Events:

The ng-mouseover, ng-mouseenter, ng-mouseleave, ng-mousedown, ngmousemove directives

```
<div ng-app ng-init="hover=true">
  <div ng-mouseenter="hover=true" ng-mouseleave="hover=false" ng-class="{red:hover}">
    This is a block
  </div>
  <hr />
  <button ng-mouseover="count = count + 1" ng-init="count=0">
    Increment (when mouse is over)
  </button>
  count: {{count}}
</div>
```

Keyboard Events:

The ng-keydown, ng-keyup, ng-keypress directives:

Event object is available as $event and can be interrogated for keyCode, altKey, etc.

```
<div ng-app>
  <input ng-keypress="key1=$event.key; key2=$event.keyCode">
  Key Pressed: {{key1}}, Its Code = {{key2}}
  <hr />
  <input ng-keydown="key3=$event.key; key4=$event.keyCode">
  Key Down: {{key3}}, Its Code = {{key4}}
</div>
```

The ng-cut, ng-copy, ng-paste directives:

Expressions are evaluated when the action occurs:

```
<div ng-app ng-init="cut=false; copy=false; paste=false">
  <input ng-cut="cut=true" ng-copy="copy=true" ng-paste="paste=true">
  <br /><br />
  Cut: {{cut}} <br />
  Copy: {{copy}}<br />
  Paste: {{paste}}
</div>
```

The ng-Src directive:

Using Angular markup like {{hash}} in a src attribute doesn't work right: The browser will fetch from the URL with the literal text {{hash}} until Angular replaces the expression inside {{hash}}. The ngSrc directive solves this problem.

```
<img ng-src="http://www.gravatar.com/avatar/{{hash}}" />
```

The ng-href directive:

Using Angular markup like {{url}} in an href attribute will make the link go to the wrong URL if the user clicks it before Angular has a chance to replace the {{url}} markup with its value. Until Angular replaces the markup the link will be broken and will most likely return a 404 error. The ngHref directive solves this problem.

```
<div ng-app ng-init="hover=true">
   <input ng-model="url" /><br />
   URL: <a ng-href="http://{{url}}">{{url}}</a> <br />
 </div>
```