ENHANCING VIEWS

Ori calvo @2015

http://oricalvo.com
ori.calvo@gmail.com

Objectives

- Build complex views
- Get familiar with useful Angular's directives
- Filters
- Conditional Display
- ngRepeat
- Handling DOM Events
- Filters
- Validation

http://oricalvo.com
ori.calvo@gmail.com

Directive Naming Convention

- A single directive can be referenced using different syntax
- Directive is documented using camel-case name
  - For example, ngModel
- In template we can use
  - ng-model
  - ng:model
  - ng_model
  - Each can be prefixed with x or data
    - x-ng-model
    - data:ng-model

Interpolation Directive

- Expression delimited by pair of curly braces
- The directive evaluates the expression and render its result as a string

```
<div class="home-view" ng-controller="HomeCtrl">
      {{message}}
</div>
```

```
function HomeCtrl($scope) {
     $scope.message = "Hello Interpolation";
}
```

Interpolating an Object

- The expression to be interpolated should be of type string
- Angular has special behavior for object type expression
- It converts the object into JSON like representation
  - Great for logging
  - Private fields (which start with $) are ignored

```
<div class="home-view" ng-controller="HomeCtrl">
     <div>{{obj}}</div>
</div>
```

```
{"id":1,"name":"Ori"}
```

Configuring Interpolation

- The start and end symbols can be changed
- Probably to support old templates

```
angular.module("myApp", [])
     .config(function ($interpolateProvider) {
         $interpolateProvider.startSymbol("{");
         $interpolateProvider.endSymbol("}");
     });
```

```
<div class="home-view" ng-controller="HomeCtrl">
      {message}
</div>
```

Problem

- The interpolation expression is rendered by the browser as a plain text
- Only when Angular loads the interpolation expression is transformed into the real value
- Angular does not loads immediately
  - It waits for DOM ready
- → User sees “ugly” text

http://oricalvo.com
ori.calvo@gmail.com

Solution: ng-bind

- Using curly braces {{ }} is easy
- However, it might be visible to the end user
  - Try to press F5 multiple times
- Solution: Use attribute instead of content
  - The browser does not try to render the attribute

```
<div class="home-view" ng-controller="HomeCtrl">
     <span ng-bind="message"></span>
</div>
```

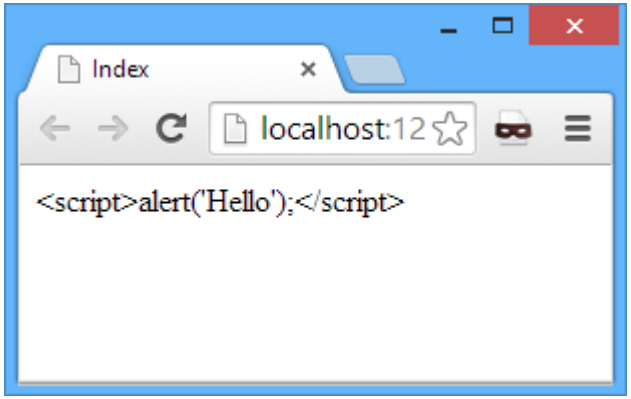
http://oricalvo.com
ori.calvo@gmail.com

HTML Encoding

- Interpolation expression are HTML encoded
- Prevent HTML injection attacks

```
function HomeCtrl($scope) {
     $scope.message = "<script>alert('Hello');</script>";
}
```

```
<div class="home-view" ng-controller="HomeCtrl">
      {{message}}
</div>
```



Disable HTML Encoding

- Old Angular supported ng-bind-html-unsafe
  - Was removed starting V1.2
- Use **ngSanitize** module
  - Can be downloaded separately

```
<div class="home-view" ng-controller="HomeCtrl">
     <span ng-bind-html="message"></span>
</div>
```

```
function HomeCtrl($scope) {
     $scope.message = "<h1>Hello</h1>";
}
```

```
angular.module("myApp", ["ngSanitize"]);
```

http://oricalvo.com
ori.calvo@gmail.com

Conditional Display

- Showing/Hiding parts of he DOM based on some condition
- Use one of the following
  - ng-show/ng-hide
  - ng-switch
  - ng-if
  - ng-include

ng-show/ng-hide

- Show/hide DOM element based on model a property

```
<div class="home-view" ng-controller="HomeCtrl">
     <span ng-show="showDiv">Hello World</span>
</div>
```

```
function HomeCtrl($scope) {
     $scope.showDiv = true;
}
```

- The directive adds/removes a CSS class named ng-hide
- Angular injects this class with display: none;
  - See last line of code inside Angular.js script

Problem

- The ng-show directive is compiled as part of Angular bootstrapping (DOM ready event)
- Up until then the DOM is visible
  - Try hit F5 multiple times
- Solution: Manually add **ng-hide** class on the relevant DOM element
  - CSS styles are processed during HTML loading
- Surprisingly, Chrome still flickers
  - Solution: Manually define the ng-hide CSS class as display: none

Prevent Flickering


```
<!DOCTYPE html>

<html ng-app="MyApp">
 <head>
     <style>
         .ng-hide {
             display: none;
         }
     </style>
 </head>
 <body>
     <div ng-controller="HomeCtrl">
         <span ng-show="showMessage" class="ng-hide">{{message}}</span>
     </div>

     <script src="~/Scripts/angular.js"></script>
     <script src="~/Scripts/HomeCtrl.js"></script>
 </body>
</html>
```

ng-switch

- Completely removes the DOM element

```
<div class="home-view" ng-controller="HomeCtrl">
     <div class="pages" ng-switch on="current">
         <div class="welcome-page" ng-switch-when="1">Welcome</div>
         <div class="content-page" ng-switch-when="2">Content</div>
         <div class="finish-page" ng-switch-when="3">Finish</div>
     </div>
     <div class="buttons">
         <button ng-click="prev()">Prev</button>
         <button ng-click="next()">Next</button>
         <button ng-click="finish()">Finish</button>
     </div>
</div>
```

```
$scope.current = 1;

$scope.next = function () {
     if (++$scope.current > PAGES) {
         $scope.current = PAGES;
     }
}

$scope.prev = function () {
     if (--$scope.current < 1) {
         $scope.current = 1;
     }
}

$scope.finish = function () {
     $scope.current = 3;
}
```

Flickering Problem (Again)

- ng-switch suffers from the same flickering problem like ng-show
- However, adding ng-hide CSS class does not help
  - Since, ng-switch does use this technique
- Solution: use ng-cloak attribute
- This is a special attribute that is automatically removed by Angular as part of DOM compilation

ng-cloak


```
<html ng-app="MyApp">
 <head>
     <title>Index</title>
     <style>
         [ng-cloak] {
             display: none;
         }
     </style>
 </head>
 <body>
     <div ng-controller="HomeCtrl as ctrl">
         <div class="pages" ng-switch on="ctrl.current" ng-cloak>
             <div class="welcome-page page" ng-switch-when="1">Welcome</div>
             <div class="content-page page" ng-switch-when="2">Content</div>
             <div class="finish-page page" ng-switch-when="3">Finish</div>
         </div>
     </div>
     <script src="~/scripts/angular.js"></script>
     <script src="~/scripts/HomeCtrl.js"></script>
 </body>
</html>
```

ng-if

- Similar behavior as ng-switch
  - Adds/removes DOM element
- Syntax is simpler

```
<div ng-controller="MainCtrl">
     <div class="admin-view" ng-controller="AdminCtrl" ng-if="isAdmin">
         <h1>Admin</h1>
     </div>

     <div class="non-admin-view" ng-controller="NonAdminCtrl" ng-if="!isAdmin">
         <h1>Non Admin</h1>
     </div>

     <button ng-click="switch()">Switch</button>
</div>
```

ng-include

- Dynamically load HTML from server based on model property
  - For example, load different HTML based on user role

```
<div ng-controller="MainCtrl">
     <div ng-include="!isAdmin && '/Main/NonAdmin'"></div>
     <div ng-include="isAdmin && '/Main/Admin'"></div>
     <button ng-click="switch()">Switch</button>
</div>
```

```
function MainCtrl($scope) {
     $scope.switch = function () {
         $scope.isAdmin = !$scope.isAdmin;
     }
}
```

ng-repeat

- Iterates over a collection of items
- Instantiates a template for each item
- Monitor the whole collection and each item separately
- Thus, any change is automatically detected and applied to the DOM

```
<ul>
     <li ng-repeat="contact in contacts">
         {{contact.name}}
     </li>
</ul>
```

```
function MainCtrl($scope) {
     $scope.contacts = [
         { id: 1, name: "Ori" },
         { id: 2, name: "Roni" },
     ];
}
```

ng-repeat Special Variables

- A set of variables created for each item scope
  - $index
  - $first, $middle, $last
  - $even, $odd

```
<ul>
    <li ng-repeat="contact in contacts" class="contact" ng-class="{even: $even}">
        {{contact.name}}
    </li>
</ul>
```

http://oricalvo.com
ori.calvo@gmail.com

ng-repeat Over an Object

- ng-repeat knows how to iterate object properties
- The syntax is different
- ng-repeat variables are available too ($index ...)

```
<div ng-controller="MainCtrl">
     <ul>
         <li ng-repeat="(name, value) in obj">
             <span>{{name}}</span> = <span>{{value}}</span>
         </li>
     </ul>
</div>
```

http://oricalvo.com
ori.calvo@gmail.com

ng-repeat is optimized

- ng-repeat tries to minimize DOM element creation
- When moving items inside the collection ng-repeat moves the DOM elements too
- How can ng-repeat differentiate between item reposition and deletion+creation ?
  - Comparing references might be too slow
- Solution: ng-repeat set a unique key for each item
  - The key is named **$$hashKey**
  - Allows ng-repeat to track items using a dictionary

http://oricalvo.com
ori.calvo@gmail.com

ng-repeat Track by

- By default ng-repeat generates its own unique keys
- This might be problematic from application POV
  - Storage
  - Networking
- Consider using the special syntax “track by” to eliminate Angular key generation

```
<ul>
     <li ng-repeat="contact in contacts track by contact.id">
         {{contact.name}}
     </li>
</ul>
```

track by $index

- There are cases were there is no unique key
- For example, when displaying a list of primitive values with duplications
- Consider using the special syntax “track by $index”

```
<ul>
     <li ng-repeat="num in numbers track by $index">
         {{num}}
     </li>
</ul>
```

http://oricalvo.com
ori.calvo@gmail.com

ng-repeat-start/ng-repeat-end

- ng-repeat is attached to one duplicated element
- But what if we want to duplicate multiple elements

```
<div ng-controller="MainCtrl">
     <div class="contacts">
         <input type="checkbox"
                ng-model="contact.checked"
                ng-repeat-start="contact in contacts" />
         <span class="content">{{contact.name}}</span>
         <button>Delete</button>
         <hr ng-repeat-end />
     </div>
</div>
```

```
function MainCtrl($scope, $element) {
     $scope.contacts = [
         { id: 1, name: "Ori" },
         { id: 2, name: "Roni" },
         { id: 3, name: "Udi" },
     ];
}
```

XXX-start & XXX-end

- Not only ng-repeat supports the idea of start and end tags
- Every directive that is defined as multiElement supports the same behavior

```
<div ng-controller="MainCtrl">
     <div class="contacts">
         <input type="checkbox"
                ng-model="contact.checked"
                ng-show-start="show" />
         <span class="content">{{contact.name}}</span>
         <button>Delete</button>
         <hr ng-show-end />
     </div>
</div>
```

http://oricalvo.com
ori.calvo@gmail.com

ng-repeat & Digest Cycle

- ng-repeat installs only one watcher
  - $watchCollection
- However, inside the repeated HTML you probably use interpolation expressions
- Each interpolation expression installs an additional watcher → a total of N watchers → Longer digest cycle
- When dealing with large array you might note performance degradation because of the large amount of watchers

bind once

- A special syntax :: applied inside Angular expression
- One time data binding
- Angular monitors the expression using a watcher
- Once a change is detected the watcher is removed

```
<div ng-controller="MainCtrl">
     <span>{{::name}}</span>
</div>
```

- Therefore, amount of watchers are reduced

DOM Events

- Use one of the built-in directives like: **ngClick, ngMousedown, ngKeydown** and **ngChange**
- You may pass arguments to the handler

```
<div ng-controller="MainCtrl">
     <input type="text" ng-model="name" />
     <button ng-click="sayHello(name)">Say Hello</button>
</div>
```

- Can send the special $event parameter

```
<button ng-click="sayHello(name, $event)">Say Hello</button>
```

http://oricalvo.com
ori.calvo@gmail.com

$event

- Special named parameter which hold a reference to the browser DOM event object
  - Or jQuery event object
- Should be used sparsely
  - Breaks testability

```
<div ng-controller="MainCtrl">
     <input type="text" ng-model="name" />
     <button ng-click="sayHello(name, $event)">Say Hello</button>
</div>
```

```
function MainCtrl($scope, $element) {
     $scope.sayHello = function (name, e) {
          console.log(e.which);
     }
}
```

Problem

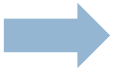
- Interpolation expression applied on a JavaScript object returns a JSON like string
- This is not appropriate behavior for Date object

```
<div ng-controller="MainCtrl">
     <span>{{date}}</span>
</div>
```

```
"2014-04-20T22:07:43.938Z"
```

- Solution – Use filter

```
<div ng-controller="MainCtrl">
     <span>{{date | date}}</span>
</div>
```



```
Apr 21, 2014
```

Filter

- A global named function
- Invoked using a pipe syntax |
- Parameters are separated by colon
- Doesn't require registration of function on the scope
  - Available for every view
- Offers more convenient syntax
- Several filters can be combined to form a transformation pipeline

```
<div ng-controller="MainCtrl">
     <span>{{message | limitTo: 5 | uppercase}}</span>
</div>
```

http://oricalvo.com
ori.calvo@gmail.com

Built in Filters

- **currency** – Formats a number as a currency
  - Globalization ?
- **date** – Formats a date into a string
- **number** – Formats a number into a string
- **filter** – Select subset of an array
- **json** – Same as interpolating an object
- **limitTo** – Shrinks an array
- **lowercase/uppercase**
- **orderBy** – Accepts a comparison expression and reverse flag

Filters inside JavaScript

- All filters can be consumed directly from JavaScript code
  - Can be injected using the name xxxFilter
  - Or using the $filter service
- Just invoke the function with the relevant parameters

```
function MainCtrl($scope, $element, currencyFilter) {
     $scope.salary = currencyFilter(1000);
}
```

```
function MainCtrl($scope, $element, $filter) {
     var currencyFilter = $filter("currency");
     $scope.salary = currencyFilter(1000);
}
```

filter Filter

- Angular offers a filter to manipulate arrays
- Unfortunately, its name is **filter**
- Allows us to easily implement a search criteria
- Supports multiple criteria's

```
<tr ng-repeat="contact in contacts | filter:search">
     <td>{{contact.id}}</td>
     <td>{{contact.name}}</td>
     <td>{{contact.email}}</td>
</tr>
```

```
<div class="filter">
     <input type="text" autofocus ng-model="search" />
     <button>Search</button>
</div>
```

Be aware of filters

- Angular assumes filters are stateless
- Allows for optimization of not calling the filter during dirty checking but rather only the inputs
- However, this optimization only applies for primitive data types
- When using filter with array Angular always invoke the filter function for each digest cycle
- You should consider not using arrays with filters

Custom Filter

- Registration is done using the **filter** method
- The factory function should return a filter function
  - First argument is the value
  - Other arguments are optional

```
angular.module("myApp").filter("trim", function () {
     return function (str) {
         return str.trim();
     }
});
```

```
<div ng-controller="MainCtrl">
     <span>{{message | trim}}</span>
</div>
```

Validation

- Angular offers validation logic through a list of directives
- Directive names are based on HTML5 input types and validation attributes

```
<form name="form">
     Name: <input type="text" ng-model="name" required name="name" />
     E-Mail: <input type="email" ng-model="email" name="email" />
     <button ng-click="save()">Save</button>
</form>
```

- Angular set the bits, you need to check for them

http://oricalvo.com
ori.calvo@gmail.com

Validation

- Validation is initiated only if using **form** tag
- The validation process is managed by **ngFormController** and **ngModelController**
- Both controllers manage a list of flags that can be analyzed by the application to understand current validation state
- In addition Angular attaches CSS classes based on these flags

CSS Validation Classes

- When using ng-model directive Angular tracks changes applied to the input field
- It dynamically attaches CSS classes
  - **ng-pristine/ng-dirty:** Modified state
  - **ng-valid/ng-invalid:** Valid state
  - **ng-touched/ng-untouched:** Blur/focus state

```
<style>
     .ng-dirty.ng-invalid {
         border-color: red;
     }
</style>
```

```
<div ng-controller="MainCtrl">
     <input type="text" ng-model="name" />
</div>
```

ngFormController

- An object that is created by the form directive
- Is attached to the $scope object
  - Property name is defined according to HTML
- Manages the validity state of the whole form
- Has the following fields
  - $valid/$invalid
  - $pristine/$dirty
  - $error
  - $submitted

```
<form name="form">
```

ngModelController

- An object that is created by the ng-model directive
- Is attached to ngFormController instance
  - Property name is specified by HTML
- Manages the validity state of a single input element
- Like ngFormController supports the following
  - $valid/$invalid
  - $pristine/$dirty
  - $error
  - $touched/$untouched

```
<input type="text" name="name">
```

ngFormController & ngModelController


```
<div ng-controller="MainCtrl">
     <form name="form">
          <div>
               <label>Name</label>
               <input type="text" ng-model="name" required name="name" />
          </div>
          <button ng-click="save()">Save</button>
     </form>
</div>
```

```
function MainCtrl($scope, $element, $log) {
     $scope.save = function () {
          if ($scope.form.$invalid) {
               alert("Some fields are invalid");
          }
          if ($scope.form.name.$invalid) {
               alert("Name field is invalid");
          }
     }
}
```

Validation Messages

- Angular does not generate validation messages
- Instead each ngModelController has $error property
- $error is a simple object
  - The key is an error validation type. For example, number, url and email
  - The value is true when validation fails
- Angular supports the following error validation types
  - number, url, email
  - required, pattern, minlength, maxlength, min, max

Validation Messages


```
<div class="form-group" ng-class="getValidationCss('name')">
     <label>Name</label>
     <input type="text" ng-model="name" required name="name" class="form-control" />
     <p class="help-block" ng-show="getValidationVisibility('email', 'required')">
         Please specify a non empty name
     </p>
</div>
```

```
$scope.getValidationCss = function (field) {
     return {
         'has-error': $scope.form[field].$invalid && $scope.form[field].$dirty
     };
}
```

```
$scope.getValidationVisibility = function (field, errorType) {
     return $scope.form[field].$dirty && $scope.form[field].$error[errorType];
}
```

HTML5 Validation

- Angular validation logic might conflict with native browser behavior
- You may consider disabling browser support by using the **novalidate** attribute

```
<form name="form" novalidate>
     <div class="form-group">
         <label>Name</label>
         <input type="text" ng-model="name" required name="name" class="form-control" />
     </div>
     <button class="btn btn-primary" ng-click="save()">Save</button>
</form>
```

Custom Validation

- Use **$setValidity** to register custom validation errors

```
<div class="form-group" ng-class="getValidationCss('min')">
     <label>Min</label>
     <input type="number" ng-model="min" required name="min" class="form-control" />
    <p class="help-block" ng-show="getValidationVisibility('min', 'minmax')">
         Minimum must be less than maximum
     </p>
</div>
```

```
$scope.save = function () {
     $scope.ignoreDirty = true;
     if (!$scope.form.min.$pristine && !$scope.form.max.$pristine && $scope.min >= $scope.max) {
         $scope.form.min.$setValidity("minmax", false);
     }
     else {
         $scope.form.min.$setValidity("minmax", true);
     }
}
```

http://oricalvo.com
ori.calvo@gmail.com

Summary

- Angular offers many directive
  - Most of them are straightforward
  - Knowing the list if crucial when doing Angular development
  - Probably you will have to create your own too
- Filters allow formatting of content
- Validation is supported using HTML form tag