AngularJS Framework

Bao Nguyen

May 20, 2014

Course Objectives

- Understanding AngularJS Framework
- Know how to get started building AngularJS applications

Agenda

- Introduction to AngularJS
- Dependency Injection
- Directives
- Routing
- Unit Test

Course Audience and Prerequisite

- Audience: Software Engineer
- Prerequisites:
  - Basic understanding of Javascript and HTML/CSS
  - Hand on working with Javascript

Assessment Disciplines

- Class Participation: 100%
- Final Exam: 70%

Course Duration

- Course duration: 9 hrs

Course Administration

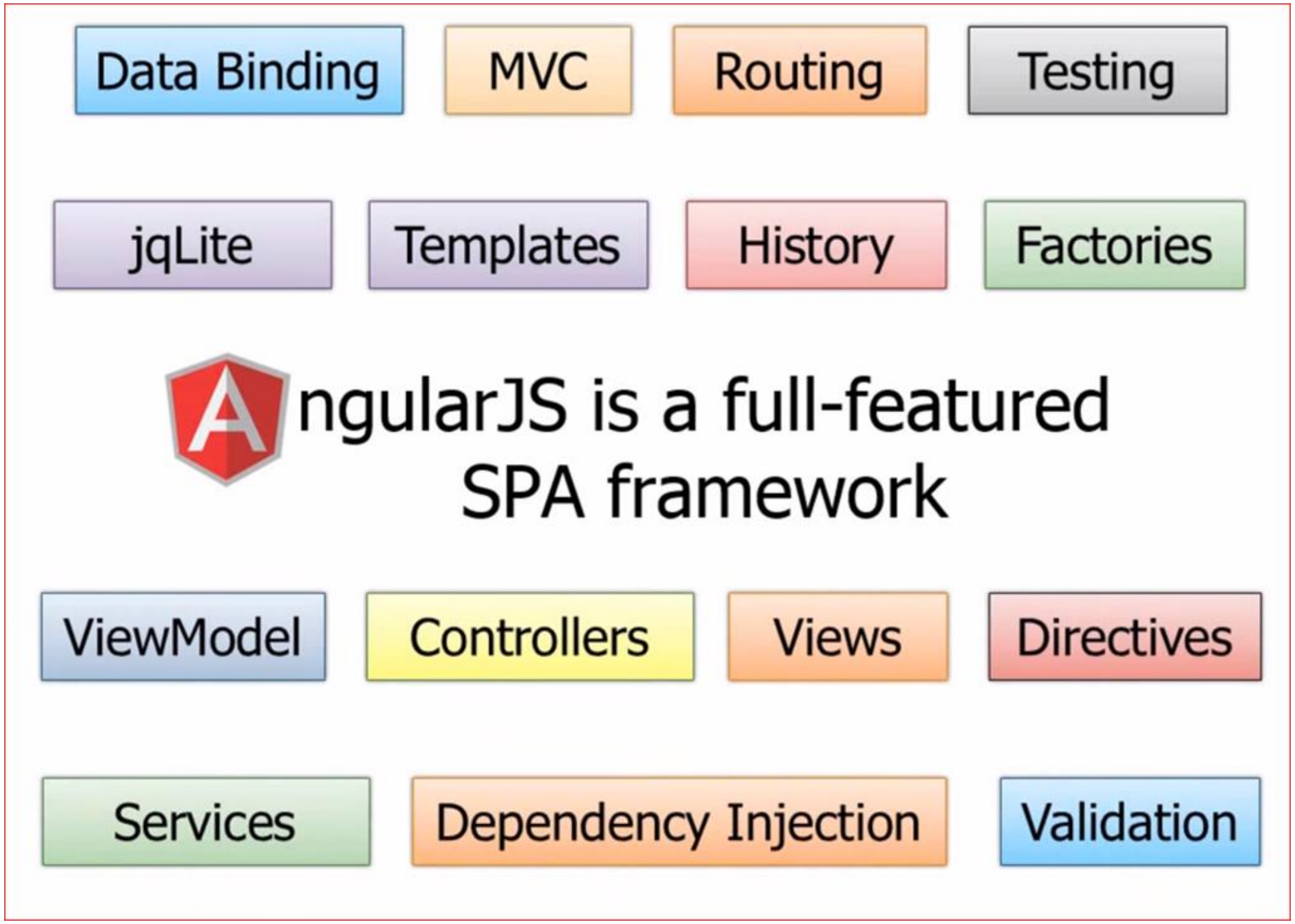
- In order to complete the course you must:
  - Sign in the Class Attendance List
  - Participate in the course
  - Provide your feedback in the End of Course Evaluation

Introduction to AngularJS Framework

What is AngularJS?

- JavaScript framework used for making SPA web applications
- It runs on plain JavaScript and HTML/CSS
- Developed in 2009 by Miško Hevery and Adam Abrons
- Maintained by Google

What is AngularJS?



Data Binding
MVC
Routing
Testing
jqLite
Templates
History
Factories

AngularJS is a full-featured SPA framework

ViewModel
Controllers
Views
Directives
Services
Dependency Injection
Validation

Problem

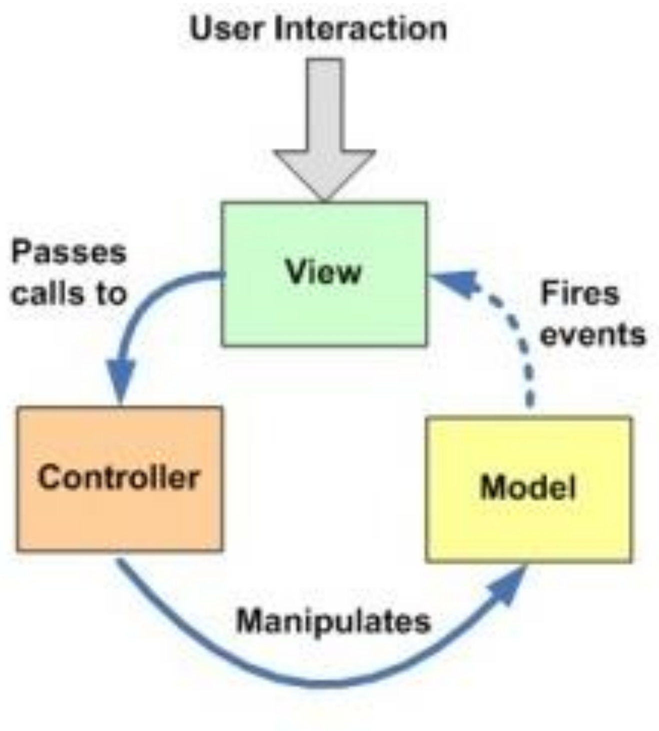
- The problem – HTML is great for static pages, but has no tools for web applications
- Traditional JavaScript programming (jQuery,…) change DOM directly

```
$("#statusBar").hide();
```

- The AngularJS solution – extend and adapt HTML vocabulary with some additional declarations that are useful for web applications

AngularJS is MVC

- MVC = Model-View-Controller
- Less dependencies
- Improves maintainability
- It is easier to read and understand code



Model-View-Controller

AngularJS is MVC

- Model
  - Holds the data
  - Notifies the View and the Controller for changes in the data
  - Does not know about the View and the Controller
- View
  - Displays stuff (buttons, labels, …) what your users will see
  - Knows about the Model
  - Usually displays something related to the current state of the Model
- Controller
  - Controls everything
  - Knows about both the Model and the View
  - The “logic” resides in it
  - What to happen, when and how

AngularJS is MVC

Model:

```
{  “name": “World"  }
```

View:

```
<div ng-controller="HelloController">
  Name:
  <input type="text" ng-model="name" />

  <h1>Hello {{ name }}</h1>

</div>
```

Controller:

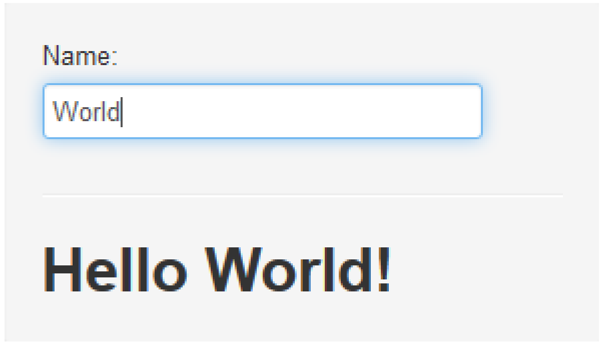
```
app.controller('HelloController', function($scope){
    $scope.name = 'World';
});
```

An AngularJS Hello World

```
<div ng-controller="HelloController">
  Name:
  <input type="text" ng-model="name" />

  <h1>Hello {{ name }}</h1>

</div>
<script>
   var app = angular.module('myApp', []);
   app.controller('HelloController',
function($scope){
      $scope.name = 'World';
   });
</script>
```



Name:

World

Hello World!

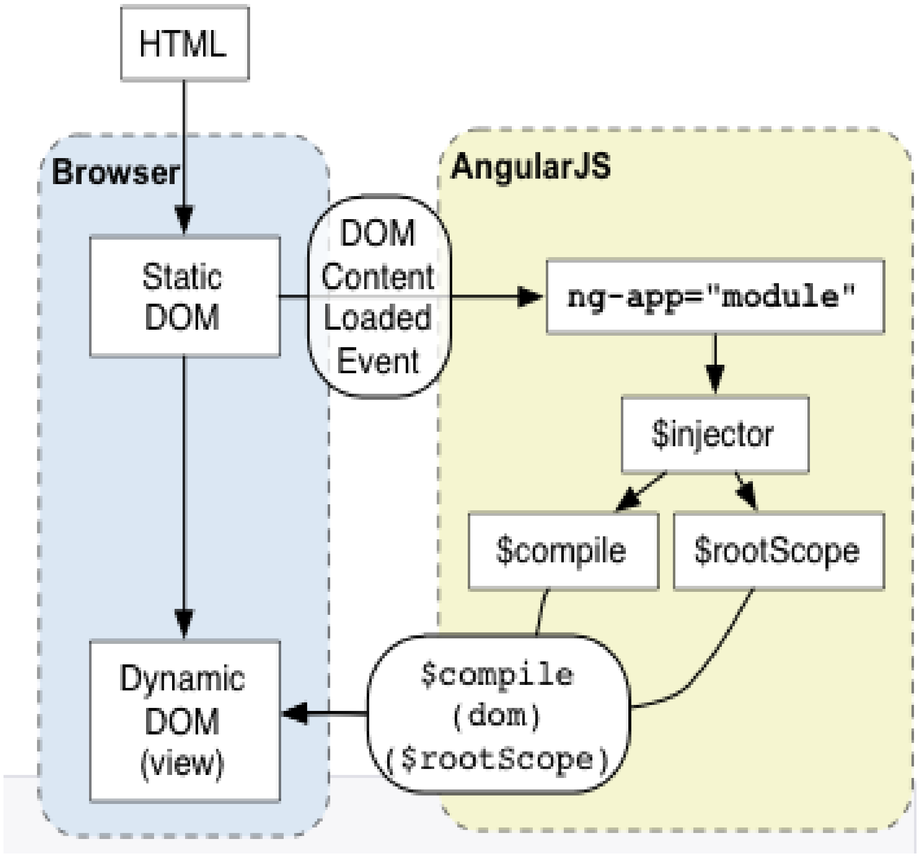
Bootstrap

Angular initializes automatically upon DOMContentLoaded event

1. The $injector that will be used for dependency injection is created.

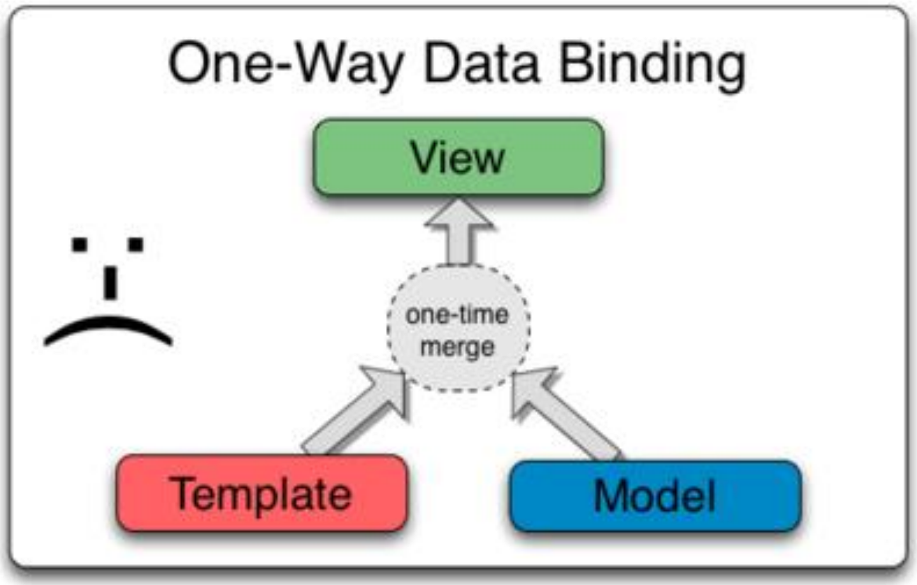
2. The injector will then create the root scope that will become the context for the model of our application.

3. Angular will then "compile" the DOM starting at the ngApp root element, processing any directives and bindings found along the way.



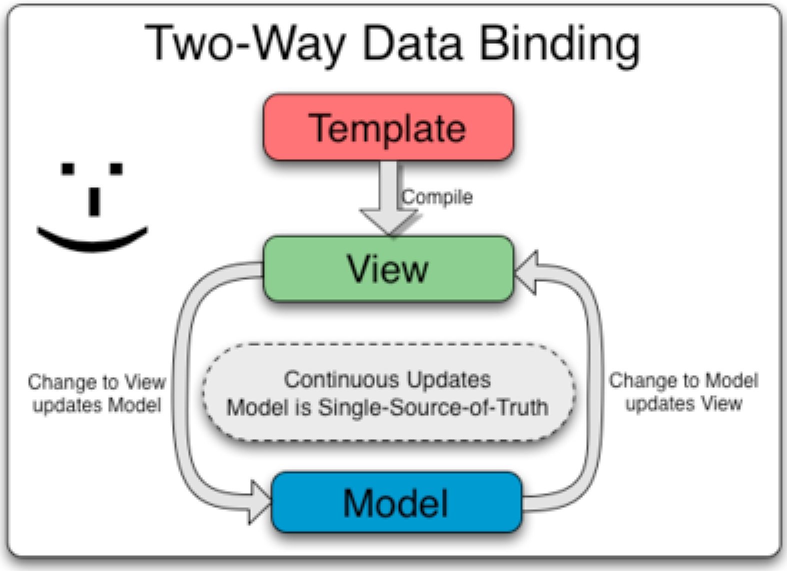
Data Binding

- Automatic synchronization of data between the model and view
- One-way data binding
  - Merge Template and Model into a View one time
  - Need to code to reflect change in Model to View and vice versa



Data Binding

- Two way Data Binding (AngularJS way)
  - View is updated automatically when the Model is changed
  - Model is updated automatically when a value in the View has changed
  - No DOM manipulation boilerplate needed! The model is the single source of truth.

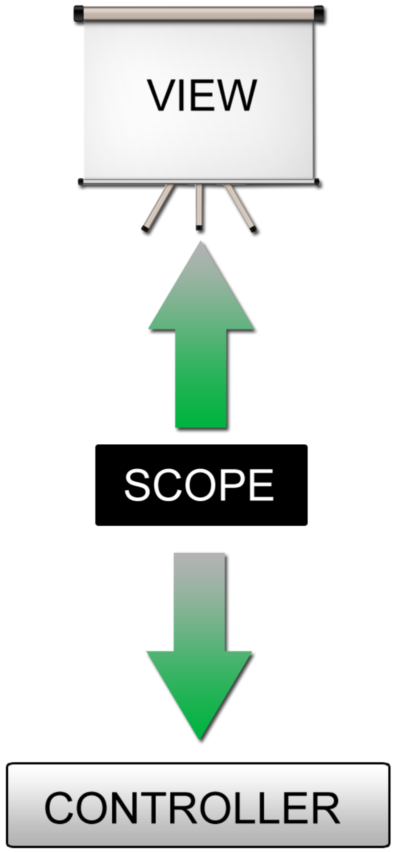

Two-Way Data Binding
Template
Compile
View
Change to View updates Model
Continuous Updates
Model is Single-Source-of-Truth
Change to Model updates View
Model

Controllers

- Define the application’s behavior
- Set up the initial state of the $scope object.
- Add behavior to the $scope object.
- Controllers use scopes to expose controller methods to templates
- Unlike services, there can be many instances in an app

Scopes

- $scope is an object that refers to the Model.
- A hash of key/values
- One scope for each controller
- The GLUE between the View and the Controller

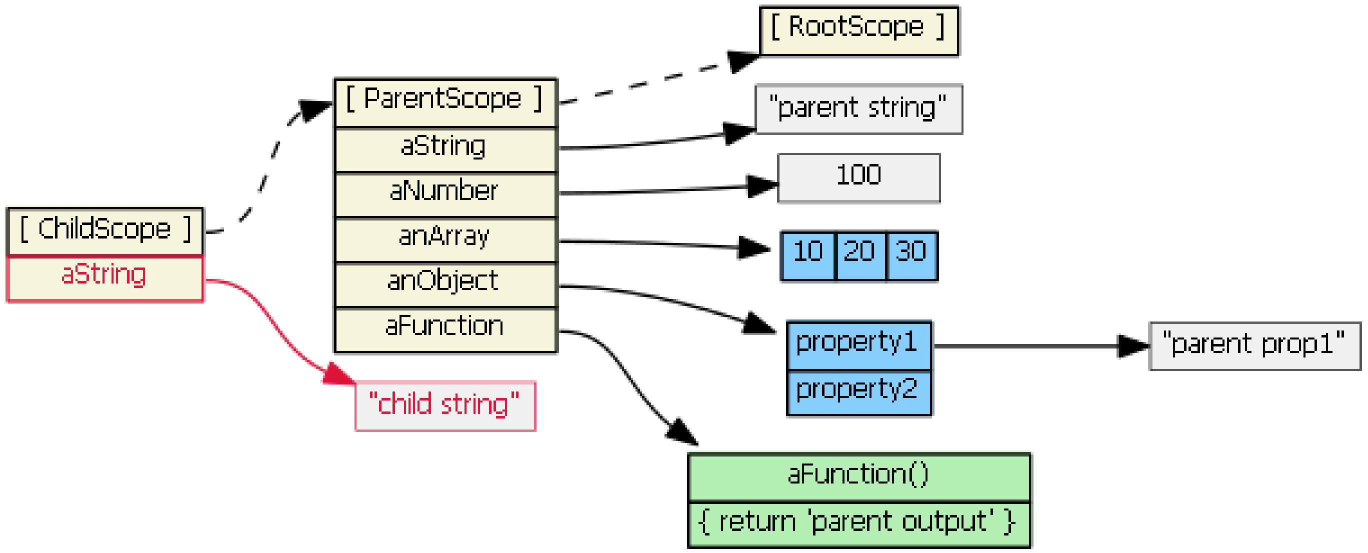


Scopes

- An execution context for expressions.
- Scope can watch expressions and propagate events.
- Directives set up $watch expressions on the scope
- Both controllers and directives have reference to the scope but not to each other.
- Scope can propagate events in similar fashion to DOM events.
  - Broadcast to the children scopes
  - Emit to the parent scope.

Scopes

- Scopes are arranged in hierarchical structure which mimic the DOM structure of the application.
- Child scopes prototypically inherit from their parents
- Exactly one root scope, but may have several child scopes.
- If a property is not find, angular searches in parent scope and so on.
- Some directives creates new child scopes



Scopes

- scope.$watch to observe model mutations.
- scope.$apply to propagate any model changes through the system into the view from outside of the "Angular realm" .
- Scopes can propagate events in similar fashion to DOM events
  - $scope.$broadcast ('paid', data): propagate to child scopes
  - $scope.$emit('paid', data): propagate to parent scope
  - $scope.$on('paid', fn): capture event in destination scope

Services

- To organize and share code across your app
- Dependency for the other components (controller, service, filter or directive)
- Lazily instantiated

```
myApp.service('mathService', function(win) {
  this.pi = 3.14;
  this.add = function(x, y) {
    return x + y;
  }

  this.multiply = function(x, y) {
    return x * y;
  }
});
```

Service Injection

- Inline array injection annotation

```
myApp.controller('MyController', ['mathService',
function(mathService) { ... }]);
```

- $inject property

```
var MyController = function(mathService) { ... };
MyController.$inject = ['mathService'];
myApp.controller('MyController', MyController);
```

- Implicit injection: determine the dependency from the name of the parameter.

```
myApp.controller('MyController', function($scope,
mathService) {
    $scope.sum = function(a, b) {
      return mathService(a, b);
    };
  });
```

Built-in services

- Name begins with $
- Available to inject into any components
- $document: a jQuery or jqLite wrapper for the browser's window.document object.
- $window: a reference to the browser's global window object
- $http: consuming REST services
  - Returns a promise object with two $http specific methods: success and error
  - Response status code 200 and 299 is considered a success status
  - Redirect is transparently followed by XMLHttpRequest

```
$http({method: 'GET', url: '/someUrl'}).
  success(function(data, status, headers, config) {
    // this callback will be called asynchronously
    // when the response is available
  }).
  error(function(data, status, headers, config) {
    // called asynchronously if an error occurs
    // or server returns response with an error status.
});
```

Expressions

- An “expression" in a template is a JavaScript-like code snippet that allows to read variables.

```
{{1+2}}
{{a+b}}
{{user.name}}
```

- In Angular, expressions are evaluated against a scope object, JavaScript expressions are evaluated against the global window
- Use filters within expressions to format data before displaying it.

```
{{ expression | filter }}
```

Filters

- A filter formats the value of an expression for display to the user
  - Uppercase a value,
  - Filter search results, etc.

```
{{user.name || upperCase}}
```

- Can be used in view templates, controllers or services
- Create filter by registering a factory function to $injector

```
module.filter(‘upperCase’, function() {
     return  function(text){
         return text.toUpperCase();
     }
});
```

Filters

- Filters can be chained in pipe

```
{{user.name || filter1 || filter2 || ...}}
```

- Filters with over two arguments
  - The first is output of the pipe
  - The second and following ones are explicit declared

```
{{user.fullname || displayFullname: ‘locale’}}
```

- Use filter as a service

```
$scope.items = $filter('orderBy')(itemsArray, "Name");
```

```
myApp.controller('MyCtrl', function(orderByFilter){
  $scope.items= orderByFilter(itemsArray, "Name");
});
```

CSS classes

- Angular automatically sets below CSS classes
  - ng-scope angular applies this class to any element for which a new scope is defined
  - ng-binding: angular applies this class to any element that is attached to a data binding
  - ng-invalid, ng-valid: whether input does not pass validation
  - ng-pristine, ng-dirty: whether user has interaction on input widget element

Animation

- Animations in AngularJS are completely based on CSS classes
- Animation hooks for common directives such as ngRepeat, ngSwitch, and ngView…
- Custom directives via the $animate service
- Demo for ngShow and ngHide

Dependency Injection

Module

- Module is container for
  - Services
  - Directives
  - Factories
  - Filters
  - Configuration information
- Each Angular JS app contains at least one module
- Be reusable container for different feathers of your app
- Where to do the DI configuration for your app
- Module can be become dependence of another module in app

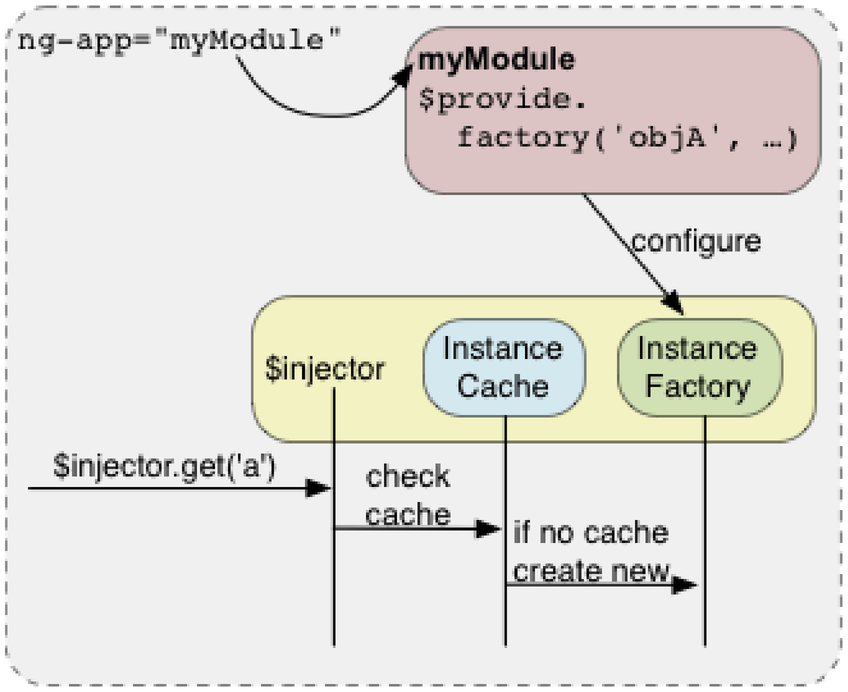
```
angular.module('moduleA', ['moduleB', 'moduleC']);
```

Module

- Each module can only be loaded once per injector
- Module configuration includes 2 block types
- Configuration Blocks
  - get executed during the provider registrations and configuration phase
  - Only providers and constants can be injected into configuration blocks
- Run Blocks
  - A run block is the code which needs to run to kickstart the application
  - It is executed after all of the service have been configured and the injector has been created
  - Only instances and constants can be injected into run blocks

Injector

- $injector service instantiate/lookup objects and wire them together for the app to work



Injector

- The injector creates two types of objects
  - **Services**: custom objects by developer
  - **Specialized objects:** Controller, Filter, Directive,…
- All services in Angular are singletons
- Registry of "recipes"
  - An identifier of the object
  - The description of how to create that object.
- Support for recipe types
  - Value
  - Factory
  - Provider
  - Service
  - Constant

Injector

- The Factory recipe
  - The return value of a factory function is the service instance

```
myApp.factory('unicornLauncher', ["apiToken",
function(apiToken) {
  return new UnicornLauncher(apiToken);
}]);
```

- The Service recipe
  - Injector invokes a constructor function with the new operator to create service instance

```
myApp.service('unicornLauncher', ["apiToken", UnicornLauncher]);
```

Injector

- The Provider recipe.
  - Define $get method directly
  - Allow create services with configurable argument values

```
 myApp.provider('unicornLauncher', function UnicornLauncherProvider() {
  var useTinfoilShielding = false;
  this.useTinfoilShielding = function(value) {
    useTinfoilShielding = !!value;
  };

  this.$get = ["apiToken", function unicornLauncherFactory(apiToken) {
    return new UnicornLauncher(apiToken, useTinfoilShielding);
  }];
});

myApp.config(["unicornLauncherProvider",
function(unicornLauncherProvider) {
  unicornLauncherProvider.useTinfoilShielding(true);
}]);
```

Injector

- The Value recipe
  - Define simple object

```
myApp.value('clientId', 'a12345654321x');

myApp.controller('DemoController', ['clientId',
function DemoController(clientId) {
  this.clientId = clientId;
}]);
```

- The Constant recipe.
  - Don't have dependencies
  - Registered object can be called in the configuration phase of module

```
myApp.constant('planetName', 'Greasy Giant');
myApp.config(['unicornLauncherProvider', 'planetName',
function(unicornLauncherProvider, planetName) {
  unicornLauncherProvider.stampText(planetName);
}]);
```

Directives

Directives

- Directives are markers on a DOM element
  - Attach a specified behavior to that DOM element or
  - Transform the DOM element and its children.
- Two types
  - Observing directives
  - Listener directives
- The HTML compiler traverses the DOM matching directives against the DOM elements.
- Should be the only place in AngularJS app do manipulate DOM

Directives

- Matching directives are called normalization
  - Strip x- and data- from the front of the element/attributes.
  - Convert the :, -, or _-delimited name to camelCase.

```
<input ng-model="foo">

<input data-ng:model="foo">
```

- Best Practice: use the dash-delimited format (e.g. ng-bind for ngBind)
- $compile can match directives based on element names, attributes, class names, as well as comments.

```
<my-dir></my-dir>
<span my-dir="exp"></span>
<!-- directive: my-dir exp -->
<span class="my-dir: exp;"></span>
```

Create Directives

- Why we create custom directives
  - To create specific HTML elements for our project
  - To wrap a repeatly used templates
  - To adapt other javascript third party libraries into an AngularJS app
- Directives are registered on modules

```
module.directive('normalizedName', factoryFn);
```

- factoryFn is either
  - Return a "Directive Definition Object“ that defines the directive properties
  - Return the postLink function

Directive to wrap a repeatly used template

- Used to apply DRY principle
- This is same idea of using ngInclude

Directive to wrap a repeatly used template

- Demo

```
angular.module('docsRestrictDirective', [])
  .controller('Controller', ['$scope', function($scope)
{
    $scope.customer = {
      name: 'Naomi',
      address: '1600 Amphitheatre'
    };
  }])
  .directive('myCustomer', function() {
    return {
      restrict: 'E',
      templateUrl: 'my-customer.html'
    };
  });
```

```
<div ng-controller="Controller">
  <my-customer></my-customer>
</div>
```

Directive to wrap a repeatly used template

- restrict: attribute versus an element?
  - Use an element when you are creating a component that is in control of the template
  - Use an attribute when you are decorating an existing element with new functionality.

Directive with Isolated Scope

- By default new directive’s scope prototypically inherit from parent element’s scope
- Isolate the directive template from parent element (except pass in)
- Enforce reusability

```
angular.module('docsRestrictDirective', [])
  .directive('myCustomer', function() {
    return {
      restrict: 'E',
      scope: {
        customerInfo: '=info'
      },
      templateUrl: 'my-customer-iso.html'
    };
  });
```

```
<div ng-controller="Controller">
  <my-customer info="naomi"></my-customer>
  <hr>
  <my-customer info="igor"></my-customer>
</div>
```

Directive with Isolated Scope

- If set to true or { }:
  - New scope will be created for the directive
  - If multiple directives on the same element request a new scope, only one new scope is created
- Notation for property in scope:
  - @ or @attr - bind a local scope property to the value of DOM attribute
  - = or =attr - set up bi-directional binding between a local scope property and the parent scope property
  - & or &attr - provides a way to execute an expression in the context of the parent scope
- If no attr name is specified then the attribute name is assumed to be the same as the local name.

Directive that Manipulates the DOM

- Directives that want to modify the DOM typically use the link option

```
function link(scope, element, attrs) { ... }
```

- Signature
  - scope is an Angular scope object.
  - element is the jqLite-wrapped element that this directive matches.
  - attrs is a hash object with key-value pairs of normalized attribute names and their corresponding attribute values.

Directive that Manipulates the DOM

- Demo

Directive that Wraps Other Elements

- When we want to pass in entire template not string or object
- Using transclude to true compile the content of the element and make it available to the directive (where place ng-transclude)

```
angular.module('docsTransclusionDirective',
[])
  .directive('myDialog', function() {
    return {
      restrict: 'E',
      transclude: true,
      templateUrl: 'my-dialog.html'
    };
  });
```

- Use &attr to allow to pass in function which also execute within outside scope context

Directive ngModel

- The most popular directive of AngularJS
- Augments the HTML controls (input, radio,…) by providing the two-way data-binding
- Support CSS classes:
  - Class attribute of control is changed according to state
  - ng-valid; ng-invalid; ng-pristine; ng-dirty
- Binding to form and control state
  - State of control is automatically stored/updated in properties
  - $dirty; $error; $invalid; $pristine; $valid
- Custom triggers
  - Any change to the content trigger a model update and form validation
  - This behavior can be overridden by using ng-model-options
- AngularJS provides some validation directives for using with ngModal (required, pattern, minlength, maxlength, min, max)

Custom Form Validate

- Adds a custom validation function to the ngModel.
- The validation can occur in two phases
- Model to View update phase:
  - Whenever the bound model changes
  - All functions in NgModelController#$formatters array are pipe-lined
  - Format the value and change validity state of the form control through NgModelController#$setValidity.
- View to Model update phase:
  - Whenever a user interacts with a control
  - NgModelController#$setViewValue is called → NgModelController#$parsers array are pipe-lined
  - Format the value and change validity state of the form control through NgModelController#$setValidity.
- Demo

Custom Form Controls using ngModel

- Write your own form control as a directive using along with ngModal.
- Implement $render () method
  - To render the data
  - Call after it passed the NgModelController#$formatters
- Call $setViewValue() method
  - Whenever the user interacts with the control and model needs to be updated.
  - Usually done inside a DOM Event listener.

Custom Form Controls using ngModel

```
angular.module('form-example2',
[]).directive('contenteditable', function() {
  return {
    require: 'ngModel',
    link: function(scope, elm, attrs, ctrl) {
      // view -> model
      elm.on('blur', function() {
        scope.$apply(function() {
          ctrl.$setViewValue(elm.html());
        });
      });
      // model -> view
      ctrl.$render = function() {
        elm.html(ctrl.$viewValue);
      };
      // load init value from DOM
      ctrl.$setViewValue(elm.html());
    }
  };
});
```

Routing

Deep Linking and AJAX app

- Deep Linking is a URL that allows navigate directly to a specific resource like a web page or a file.
  - http://example.com/path/page
  - http://example.com/
- AJAX application
  - Contents of AJAX sites are loaded by script
  - Can’t bookmark or history browsing (back or forward)
- Solutions: use ‘#’ in the URL
  - http://www.example.com/home.html#introduction
  - Browser would not refresh the page when the hash is changed.
  - JavaScript need to listen for the change of the hash in the navigation bar and do actions.

```
<a href="#C4">See also Chapter 4.</a>
```

AngularJS Route

- Help to create a layout style for application
- Application routes in Angular are provided by $route service
  - Wire controllers, view templates, and the current URL location in the browser
  - It watches $location.url() and map the path to an existing route definition
  - Used in conjunction with the ngView directive

```
<body>
  <div ng-view></div>
</body>
```

- Configure $route service
  - Via $routeProvider in config block of module
  - when(path, route): adds a new route definition to the $route service.
  - otherwise(params): sets default route definition

AngularJS Route

- Example path:
  - http://www.myapp.com/index.html#phones/ip5
  - http://www.myapp.com/index.html#phones/

```
phonecatApp.config(['$routeProvider',
  function($routeProvider) {
    $routeProvider.
      when('/phones', {
        templateUrl: 'partials/phone-list.html',
        controller: 'PhoneListCtrl'
      }).
      when('/phones/:phoneId', {
        templateUrl: 'partials/phone-detail.html',
        controller: 'PhoneDetailCtrl'
      }).
      otherwise({
        redirectTo: '/phones'
      });
  }]);
```

AngularUI Router

- Routing framework for AngularJS
- Built by the AngularUI team
- Organize routing by a state machine, rather than a simple URL route
- Allows for nested views and multiple named views
- Be ngRoute with more additional features

Unit Test

Unit Test

- Why need to test
  - Ensure reliability in production
  - If it's hard to test, maybe it needs refactored?
  - Let developers refactor with confidence if any
- Angular Mock
  - Inject and mock Angular services into unit tests
  - Extends various core ng services
- Recommended Testing Suite: Jasmine
- Recommended Test Runner: Karma

Unit Test Steps

- Install Node.js
- Install Karma via npm
- Configure Karma
- Write Unit test using Jasmine
- Run Karma to connect to a browser to run unit test

Karma Test Runner

- Install Karma and command line

```
npm install -g karma;

npm install -g karma-cli;
```

- Init Karma configuration for our app, from command line type

```
karma init my.conf.js;
```

- Start Karma server

```
karma start my.conf.js;
```

Jasmine Framework

- A behavior-driven development (BDD) framework for testing JavaScript code
  - Describe a suite → describe
  - Describe a spec → it
  - Setup and teardown → beforeEach and afterEach

```
describe("A suite", function() {
  var foo;
  beforeEach(function() {
    foo = 0;
    foo += 1;
  });
  it("contains spec with an expectation", function() {
    expect(foo).toEqual(1);
  });
  afterEach(function() {
    foo = 0;
  });
});
```

Unit Test Controller/Service

- Controller is the most important thing to unit test in your angular app
- Create controller mock and service mock, for instance

```
service = $injector.get('basicService');
ctrl = $controller('MainCtrl', {$scope: $scope,
basicService: service});
```

- Demo

Unit Test Directive

- Directive involves to DOM element
- Using $compile and angular.element() for testing
- Demo

Reference

- AngularJS official site
  - https://docs.angularjs.org/guide
- Others
  - https://egghead.io/
  - https://github.com/angular-ui/ui-router/wiki
  - http://stackoverflow.com/questions/14049480/what-are-the-nuances-of-scope-prototypal-prototypical-inheritance-in-angularjs

CSC

Q&A

Thank You !