
SOUNDCLOUD

Introducing AngularJS

Márton Salomváry

@salomvary

salomvary@soundcloud.com

What is AngularJS

A JavaScript *application* framework:

- created by Google (engineers)
- been around since 2009
- stable (1.0.7)
- no dependencies (can use jQuery though)

Why AngularJS?

Isn't Backbone good enough?

Grouping JS frameworks

- low level / DOM — jQuery, Prototype
- MVC — Backbone, Angular
- full fledged — YUI, ExtJS

What is MVC?

- Model - transient/persisted data
- View - HTML template
- Controller - glue, business logic

Aim: single page applications.

Another grouping of MVC frameworks

By me, for now :)

- no data binding
- has **data binding**

What is data binding?

...but first, how is life *without* data binding?

A Handlebars template

```
{{#each sounds}}
<div>
  <h2>{{title}}</h2>
  <button class="play-button">▸</button>
</div>
{{/each}}
```

A Backbone View

```
events: {
  'click .play-button': 'play'
},
initialize: function() {
  this.model.on('change:foo', this.render, this)
  this.$('.something').on('keyup', ...)
},
play: function() {
  this.model.set(...)
  // or manipulate DOM, etc.
}
```

Data binding = ☺

```
<div ng-repeat="sound in sounds">
  <h2>{{ sound.title }}</h2>
  <button ng-click="play()">▸</button>
</div>
```

What happens?

Model changes -> DOM changes

- including loop inserts/removes
- including fancy manipulations eg. visibility
- only the relevant DOM nodes

User interactions -> model changes

- model property update
- function call

- simple and powerful expression language

The MAGIC
is called
DIRTY CHECKING

"Manual" lifecycle

```
factory('socket', function($rootScope) {
  var connection = new WebSocket('ws://localhost')
  // ...

  connection.onmessage = function(message) {
    $rootScope.$apply(function () {
      handler.call(null, message.data)
    })
  }
})
```

Using the socket service

```
socket.onMessage(function(data) {
  $scope.foo = ... //use data to update $scope
})
```

Updates the DOM automagically afterwards.

Modules

Defining modules

```
angular.module('app', []).

controller('Playlist', function($scope, $http, socket) {
  // controller implementation
}).

directive('player', function() { //... }).

filter('toUpperCase', function() { //... })
```

Alternative (ugly) module syntax

Has magic, but breaks when minified:

```
controller('AnotherCtrl', function($window) {
  // ...
})
```

Safe to minify:

```
controller('AnotherCtrl', ['$window', function(the$window) {
  // ...
}])
```

Modules don't do

- script fetching
- lazy loading

..and there is the minification problem

Dependency Injection

aka. Inversion of Control

- code "asks" for dependencies
- "injector" looks up and passes them in
- easy to swap implementation
- useful for testing/mocking

With code

```
controller('Playlist', function ($window, $scope) {
  // at some point...
  $window.alert(‘Playlist is over!’)
})

// test
var mock = { alert: jasmine.createSpy() }, $scope = {}
$provide.value('$window', mock)
$controller(‘Playlist’,  { $scope: $scope })

expect(mock.alert.callCount).toEqual(1)
```

Directives

Directives

"Directives are a way to teach HTML new tricks"

- hook on
  - attribute
  - class name
  - tag
  - comment
- built-in directives, eg. ng-repeat=""
- custom directives, eg. <player>

Powerful and complicated

```
myModule.directive('directiveName', function factory(injectables) {
  var directiveDefinitionObject = {
    priority: 0,
    template: '<div></div>',
    templateUrl: 'directive.html',
    replace: false,
    transclude: false,
    restrict: 'A',
    scope: false,
    controller: ["$scope", "$element", "$attrs", "$transclude", "otherInjectables",
      function($scope, $element, $attrs, $transclude, otherInjectables) { ... }],
    compile: function compile(tElement, tAttrs, transclude) {
      return {
        pre: function preLink(scope, iElement, iAttrs, controller) { ... },
        post: function postLink(scope, iElement, iAttrs, controller) { ... }
      }
    },
    link: function postLink(scope, iElement, iAttrs) { ... }
  };
  return directiveDefinitionObject;
});
```

Controllers and Models

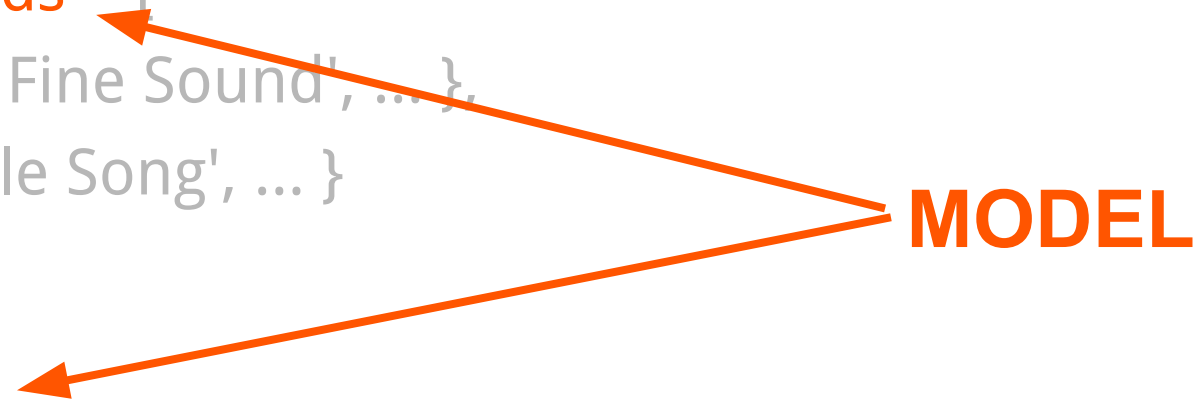
The view

```
<div ng-controller="Playlist">
  <div ng-repeat="sound in sounds">
    <h2>{{ sound.title }}</h2>
    <button ng-click="play()">▸</button>
  </div>
</div>
```

Controllers and models

```
controller('Playlist', function ($scope) {
  $scope.sounds = [
    { title: 'One Fine Sound', ... },
    { title: 'Whale Song', ... }
  ]

  $scope.play = function() { ... }
})
```



"In Angular, a model is any data that is reachable as a property of an angular Scope object."

Scope inheritance

- Organized hierarchically - following the DOM
- Using prototypical inheritance
- Scopes are created
  - for controllers
  - by directives
- Demo: playlist, tricky inheritance

What else?

Built-in services

- $http, $resource
- $route
- $location
- $timeout
- $log
- $cacheFactory

Summary

The good parts

- “full stack”
- data binding
- dependency injection
- project structure (angular-seed)
- $scope vs this

The bad parts

"There are tons of things should be mentioned in the docs lol. I guess this is the kinda the fun part of learning AngularJS, just like treasure hunting." (StackOverflow)

- documentation (example)
- complexity (directives!)
- minification vs. module syntax
- too much logic in templates

Alternatives (for data binding)

- Ember.js - even more sophisticated
- Knockout - ugly template syntax
- Epoxy.JS - for Backbone

Thank you

Márton Salomváry

@salomvary

salomvary@soundcloud.com

soundcloud.com/jobs