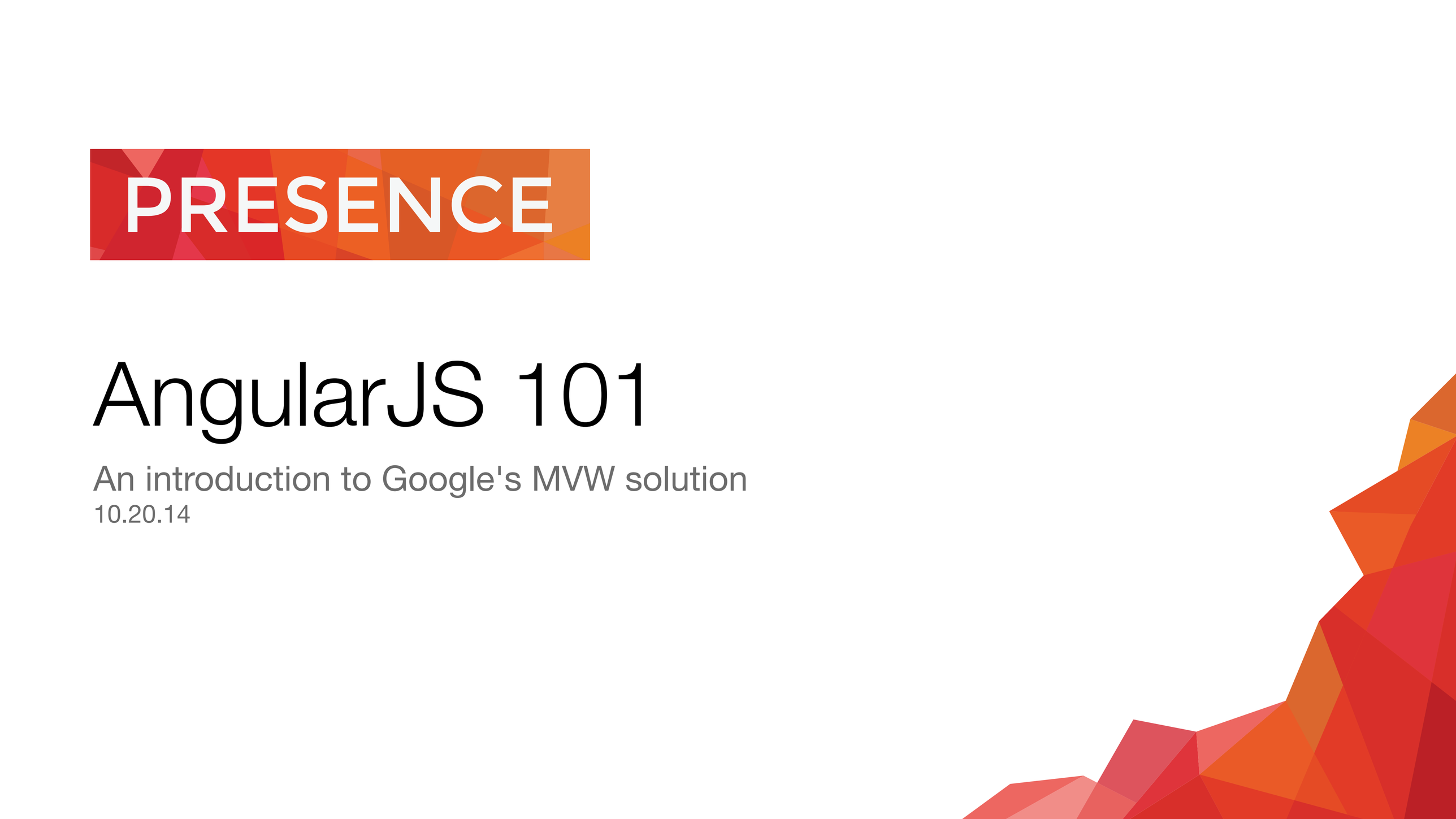
PRESENCE

AngularJS 101

An introduction to Google's MVW solution

10.20.14



PRESENCE

Sean Cannon

Brief Introduction

Agenda

1. What is AngularJS?
2. The Scaffold
3. ng-app
4. Controllers
5. Directives
6. Filters
7. Services
8. Animations
9. Modules
10. Demo

1. What is AngularJS?

1. MVW (Model View Whatever)
2. Created by Google
3. Organized
4. Declarative
5. Modular

2. The Scaffold

```
/app
    /images
    /scripts
        /animations
        /controllers
        /directives
        /filters
        /modules
        /services
    /styles
    /views
        /pages
        /partials
/bower_components
```

A proposed directory structure

HELPFUL DEPENDENCY CONSIDERATIONS:

- Bower
- Grunt
- Protractor
- Karma
- Batarang
- Yeoman

3. ng-app

```
<!-- index.html -->
<!DOCTYPE html>
<html>
    <head></head>

    <body ng-app="ng101" ng-controller="RootCtrl as rootCtrl">
        <div ng-view></div>

        <!-- Bower scripts go here -->

        <!-- Our concatenated script, built by Grunt -->
        <script src="concat/scripts/build.js"></script>
    </body>
</html>
```

- `ng-app` is a directive. It defines the scope of your application and accepts your app name.
- `ng-101` is this app's name.
- `ng-view` is where the router will render its output.

app.js

```
angular.module('ng101', []);
```

- Angular will map your app name here when you create your module.
- The above code is all you need to begin using controllers.

```
angular.module('ng101', [
    'ngAnimate',
    'ngCookies',
    'ngResource',
    'ngRoute',
    'ngSanitize',
    'ngTouch',
    'ngjQuery' // Custom module
])
.config(['$routeProvider', function ($routeProvider) {

    $routeProvider
        .when('/', {
            templateUrl : 'app/views/pages/main.html',
            controller  : 'MainCtrl as mainCtrl'
        })
        .otherwise({
            redirectTo : '/'
        });

}]);
```

app.js

4. Controllers

1. Create scope
2. Are instances, and can be instantiated from routes, directives, or the markup
3. Should contain application logic and avoid containing DOM manipulation
4. Can be inherited through scope propagation or through directives via `require`

Routing

```
$routeProvider
    .when('/', {
        templateUrl : 'app/views/pages/main.html',
        controller  : 'MainCtrl as mainCtrl'
    })
    .otherwise({
        redirectTo : '/'
    });
```

```
angular.module('ng101').controller('MainCtrl', [
function () {

    this.message = 'Welcome to HTML5DevConf!';

}]);
```

APP.JS

Here we defined a "/" route that would instantiate the `MainCtrl` controller and declare that it should be referenced in the templates as `mainCtrl`.

CONTROLLERS/MAIN.JS

- `this` is the scope reference
- `message` is a property on the scope which we can now reference in the templates as `mainCtrl.message`
- `$scope` can be used instead of `this`, and also offers exposed API methods like `$watch`, `$digest`, `$apply` and more

controllers/root.js

```
angular.module('ng101').controller('RootCtrl', ['$rootScope', '$window',
function ($rootScope, $window) {

    $rootScope.$on('$routeChangeError', function (event, current, previous, rejection) {
        $window.alert(rejection);
    });

}]);
```

- It is a good practice to have a "catch-all" controller on the `ng-app` level to handle route-change errors and any other `$rootScope` handling logic.
- Notice the minification-proof dependency injection.
- Notice the lack of global scope clutter.
- `$window` instead of `window`

5. Binding

```
angular.module('ng101').controller('MainCtrl', [
function () {

    this.className = 'AngularJS 101';
    this.message   = 'Welcome to HTML5DevConf!';

}]);
```

CONTROLLERS/MAIN.JS

```
<!-- One-time binding example -->
<h1>{{::mainCtrl.className}}</h1>

<!-- Two-way, watched binding example -->
<h3>{{mainCtrl.message}}</h3>

<input type="text" ng-model="mainCtrl.message" />
```

VIEWS/PAGES/MAIN.HTML

5. Binding

```
angular.module('ng101').controller('MainCtrl', ['$interval',
function ($interval) {

    $interval(function () {
        this.now = new Date().toString();
    }.bind(this), 1000);

}]);
```

CONTROLLERS/MAIN.JS

```
<!-- Two-way, watched binding example -->
<h3>{{mainCtrl.now}}</h3>
```

VIEWS/MAIN.HTML

5. Directives

1. Components or Containers
2. Core or Custom
3. Can inherit scope or isolate it
4. Support multiple binding types passed in as attributes
5. Are self-aware and discourage DOM traversal
6. Can be tricky when starting out

directives/movie.js

```
angular.module('ng101').directive('movie', [ // Inject dependencies like any other module
function () {

   return {
       restrict : 'E', // or A, C, M
       scope    : {
           title : '@', // Literal value
           meta  : '=', // Two-way binding
           click : '&'  // Expression
       },
       template : '<h3>{{title}}</h3>',
       link     : function (scope, element, attrs) { // For directives that manipulate the DOM
           // constructor logic goes here.

           scope.$on('$destroy', function () {
               // Garbage collect, remove bindings, etc.
           });
       }
   };

}]);
```

Directive restrictions

```
<movie title="{{mainCtrl.movies[0].title}}"></movie>
```

ELEMENT

Use `restrict : 'E'`

```
<div movie title="{{mainCtrl.movies[0].title}}"></div>
```

ATTRIBUTE

Use `restrict : 'A'`

```
<div class="movie" title="{{mainCtrl.movies[0].title}}"></div>
```

CLASS

Use `restrict : 'C'`

```
<!-- directive:movie -->
<div title="{{mainCtrl.movies[0].title}}"></div>
```

COMMENT

Use `restrict : 'M'`

Common core directives

```
<p ng-repeat="comment in myCtrl.comments">{{comment}}</p>
```

NG-REPEAT

Appends DOM nodes for each item in a collection

```
<button id="logout" ng-if="myCtrl.user.isLoggedIn">Logout</button>
```

NG-IF

Keeps inapplicable DOM out of memory

```
<menu ng-show="myCtrl.menu.isOpen"></menu>
```

NG-SHOW

CSS display toggle

```
<button ng-click="myCtrl.menu.isOpen = !myCtrl.menu.isOpen">
    Toggle menu
</button>
```

NG-CLICK

Evaluates an expression

ng-repeat example

```
angular.module('ng101').controller('MainCtrl', [
function () {

    this.movies = [
        {title : 'Bloodsport', year : 1988},
        {title : 'Kickboxer',  year : 1989},
        {title : 'Cyborg',     year : 1989}
    ];

}]);
```

CONTROLLERS/MAIN.JS

A simple array of objects is assigned to a property on the scope.

```
<movie ng-repeat="movie in mainCtrl.movies"
       title="{{movie.title}}">
</movie>
```

VIEWS/PAGES/MAIN.HTML

- Here we instantiate a new `movie` directive for each movie found in the collection.

ng-repeat example

```
angular.module('ng101').controller('MainCtrl', [
function () {

    this.movies = [
        {title : 'Bloodsport', year : 1988},
        {title : 'Kickboxer',  year : 1989},
        {title : 'Cyborg',     year : 1989}
    ];

}]);
```

CONTROLLERS/MAIN.JS

A simple array of objects is assigned to a property on the scope.

```
<movie ng-repeat="movie in mainCtrl.movies"
       title="{{movie.title}}">
    So, what about this?
</movie>
```

VIEWS/PAGES/MAIN.HTML

- Here we instantiate a new `movie` directive for each movie found in the collection.

Transclusion

1. Tell a directive you care about what's inside with `transclude : true`
2. Angular will copy the `innerHTML` of the directive.
3. ... and paste it in the directive's template into a provided `ng-transclude` directive.

directives/movie.js

```
angular.module('ng101').directive('movie', [
function () {

    return {
        restrict   : 'E', // or A, C, M
        transclude : true,
        scope      : {
            title : '@', // Literal value
            meta  : '=', // Two-way binding
            click : '&'  // Expression
        },
        template   : '<h3>{{title}}</h3><div ng-transclude></div>',
        link       : function (scope, element, attrs) {
            // constructor logic goes here.
        }
    };

}]);
```

templateUrl

```
// Before
template : '<h3>{{title}}</h3><div ng-transclude></div>',
```

```
// After
templateUrl : 'app/views/partials/movie.html',
```

```
<h3>{{title}}</h3><div ng-transclude></div>
```

DIRECTIVES/MOVIE.JS

Abstracting views into their own files greatly reduces directive clutter and concatenation requirements.

VIEWS/PARTIALS/MOVIE.HTML

- Here we instantiate a new `movie` directive for each movie found in the collection.

templateUrl

```
templateUrl : 'app/views/partials/movie.html',
```

DIRECTIVES/MOVIE.JS

The view URI will become the `$templateCache` key so Angular will only make one http request. Grunt can seed the `$templateCache` for you during your build process.

6. Filters

```
angular.module('ng101').controller('MainCtrl', ['$filter',
function ($filter) {

    this.message = 'Hello world';

    // HELLO WORLD
    this.capsMessage = $filter('uppercase', this.message);

}]);
```

```
<movie ng-repeat="movie in mainCtrl.movies | limitTo:2"
       title="{{movie.title}}">
</movie>
```

CORE FILTERS

currency	date
filter	json
limitTo	lowercase
number	orderBy
uppercase	

Custom filters

```
angular.module('ng101').filter('reverse', [
function () {
    return function (text) {
        return text.split('').reverse().join('');
    };
}]);
```

FILTERS/REVERSE.JS

A John Lindquist goodie.

```
<h1>{{mainCtrl.message | reverse}}</h1>
```

VIEWS/PAGES/MAIN.HTML

Displays "!fnoCveD5LMTH ot emocleW"

7. Services

1. Persist data across controllers
2. Singletons
3. Use AngularJS's factory pattern
4. Can be injected into controllers, directives, filters, animations, and other services
5. Can not circular-reference

Service injection and property persistence

```
angular.module('ng101').service('MovieSvc', [
function () {
    var svc = this;

    svc.favoriteMovie = 'Bloodsport';

    return svc;
}]);
```

SERVICES/MOVIE.JS

A service is a portable, shared scope. It is commonly used to make AJAX requests and to normalize and persist response data to controller scopes.

```
angular.module('ng101').controller('MainCtrl', ['MovieSvc',
function (MovieSvc) {

    this.movieSvc = MovieSvc;

    this.favoriteMovie = this.movieSvc.favoriteMovie;

    this.movies = [
        {title : 'Bloodsport', year : 1988},
        {title : 'Kickboxer',  year : 1989},
        {title : 'Cyborg',     year : 1989}
    ];

}]);
```

VIEWS/PAGES/MAIN.HTML

- Here we instantiate a new `movie` directive for each movie found in the collection.

views/pages/main.html

```
<select ng-options="movie.title for movie in mainCtrl.movies | orderBy:'year'"
        ng-model="mainCtrl.movieSvc.favoriteMovie">
    <option value=""></option>
</select>

<movie ng-repeat="movie in mainCtrl.movies"
       title="{{movie.title}}"
       ng-class="{'favorite' : mainCtrl.movieSvc.favoriteMovie.title === movie.title}">
</movie>
```

- Core directives and filters such as `ng-options`, `ng-model`, `ng-class`, and `orderBy` can be leveraged to drastically streamline your code.

8. ngAnimate

1. AngularJS 1.2 overhauled animations.
2. You can leverage built-in hooks or create your own.
3. Uses JavaScript, CSS3 Transitions and CSS3 Keyframe Animations
4. Core : ngRepeat, ngInclude, ngIf, ngSwitch, ngShow, ngHide, ngView and ngClass.
5. Custom directives can take advantage of animation by using the $animate service.
6. Angular will wait for two digest cycles until enabling animations

Animation example, straight from the docs

```
<style type="text/css">
    .slide.ng-enter, .slide.ng-leave {
        -webkit-transition : 0.5s linear all;
        transition         : 0.5s linear all;
    }

    .slide.ng-enter { }                 /* starting animations for enter */
    .slide.ng-enter.ng-enter-active { } /* terminal animations for enter */
    .slide.ng-leave { }                 /* starting animations for leave */
    .slide.ng-leave.ng-leave-active { } /* terminal animations for leave */
</style>

<!--
    the animate service will automatically add .ng-enter and .ng-leave to the element
    to trigger the CSS transition/animations
-->
<ANY class="slide" ng-include="..."></ANY>
```

9. Modules

```
angular.module('ngjQuery', []).service('jQuery', ['$window',
function ($window) {
  return $window.jQuery || {};
}]);
```

```
angular.module('ng101', ['njQuery']);
```

```
angular.module('ng101').directive('squarify', ['jQuery'
function (jQuery) {
    return function (scope, element, attrs) {
        var $this = jQuery(element[0]);
        $this.height($this.width());
    };
}]);
```

MODULES/JQUERY.JS

Eliminates assumptions like "`window.$` exists and points to `window.jQuery`.

APP.JS

- Inject modules into the app definition.

A DIRECTIVE

- *jqLite* is missing functions like `height()` and `width()`.

10. Demo

Sean Cannon

sean@presencepg.com