BOOTSTRAPPING

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

Bootstrapping

- Angular initialization process
- Converting a plain HTML page to a full functional application
- Loading all relevant modules
- Angular allows the developer to participate in the process

Common Angular App


```
<!DOCTYPE html>

<html ng-app="MyApp">
 <head>
     <meta name="viewport" content="width=device-width" />
     <title>Index</title>
 </head>
 <body>
     <div ng-controller="HomeCtrl">
         {{name}}
     </div>

     <script src="~/Scripts/angular.js"></script>

     <script>
         angular.module("MyApp", [])
             .config(function () {
             })
             .run(function () {
             })
             .controller("HomeCtrl", function ($scope) {
                 $scope.name = "Hello Ng";
             });
     </script>
 </body>
</html>
```

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

Facts you already know

- Module is a container of providers/services/controllers/directives
- The directive ng-app is used to bootstrap Angular
- config block is used to configure providers
- run block is used to initialize app/services
- ng-controller directives causes Angular to instantiate the specified controller
- Controller is responsible for attaching state into the $scope

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

Did you ever wonder ?

- How is the directive ng-app defined ?
- What does Angular exactly do when being loaded
- Which comes first
  - config/run blocks
  - jQuery.ready (legacy code)
- Why does {{name}} flicker ?
- Where can I see the list of loaded modules ?
- Lets handle these questions …

Loading Angular's Script

- Immediately when loaded, Angular performs the following actions
  - Binds to jQuery (if exist)
  - Publishes its public API
  - Creates the ng module
  - Registers for DOMContentLoaded event

```
bindJQuery();

publishExternalAPI(angular);

jqLite(document).ready(function() {
     angularInit(document, bootstrap);
});
```

Binding to jQuery

- Initializes angular.element with jQuery object (if present) or Angular JQLite implementation
- Extends jQuery object with Angular specific functions like controller, scope and injector

```
function bindJQuery() {
     jQuery = window.jQuery;

     if (jQuery && jQuery.fn.on) {
         jqLite = jQuery;
         extend(jQuery.fn, {
             controller: JQLitePrototype.controller,
             ...
         });
     }
     else {
         jqLite = JQLite;
     }

     angular.element = jqLite;
}
```

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

ngJq – Version 1.4

- Documentation says:

"Use this directive to force the angular.element library. This should be used to force either jqLite by leaving ng-jq blank or setting the name of the jquery variable under window (eg. jQuery)."

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

JQLite

- Supports only subset of jQuery API

addClass	after	append	attr	bind
children	clone	contents	css	data
detach	empty	eq	find	hasClass
html	next	on	off	one
parent	prepend	prop	ready	remove
removeAttr	removeClass	removeData	replaceWith	text
toggleClass	triggerHandler	unbind	val	wrap

Publishing Public API

- angular object is extended with public API
- angular.module (named angularModule below) function is defined
- ngLocale and ng modules are created (not loaded)

```
function publishExternalAPI(angular) {
   extend(angular, {...});

   angularModule = setupModuleLoader(window);

   try { angularModule('ngLocale');}
   catch (e) {
       angularModule('ngLocale', []).provider('$locale', $LocaleProvider);
   }

   angularModule('ng', ['ngLocale'], ['$provide', function ngModule($provide) {
          $provide.directive({...});

          $provide.provider({...});
     }]);
}
```

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

Publishing Public API

- Angular must publish its own API before DOMContentLoaded event since our application uses this API immediately after Angular script is loaded

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

<script>
     angular.module("MyApp", []);
</script>
```

angular.module

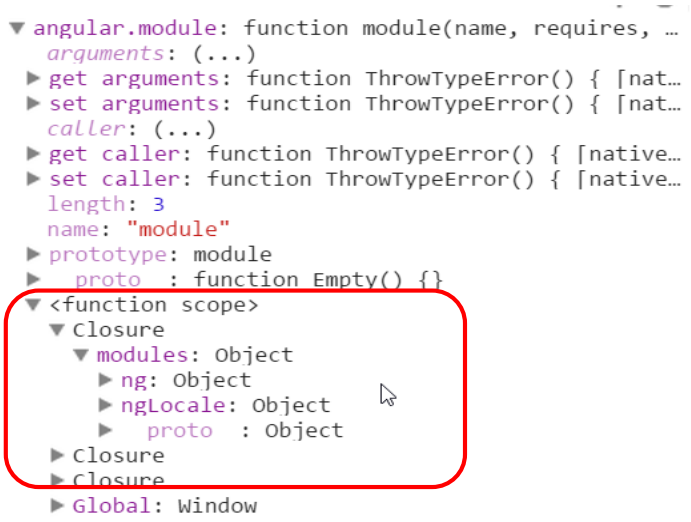
- Holds a reference to a global array named modules which holds all registered modules

```
function setupModuleLoader(window) {
     return ensure(angular, 'module', function () {
         var modules = {};

         return function module(name, requires, configFn) {
             ...
         }
     });
}
```

- The modules reference is a local variable and therefore not available on the angular global object

Analyzing all Loaded Modules

- According to previous slide we cannot easily determines at runtime the list of all loaded modules
  - Unfortunately, angular.modules does not exist
- However, angular.module holds private reference (closure) to the modules array
- Using Chrome DEV tools:



angular.module

- Receives the following parameters
  - name
  - requires
  - configFn
- The requires parameter is overloaded
  - If undefined, angular.module behaves as a getter
  - If non empty, angular.module creates a new module

```
function module(name, requires, configFn) {
     return ensure(modules, name, function () {
         if (!requires) {
             throw $injectorMinErr('nomod', "...", name);
         }
     });

     ...
}
```

Module Redefinition is allowed

- Not common but Angular allows for module redefinition
  - All registered providers/services/controllers are lost

```
function module(name, requires, configFn) {
     if (requires && modules.hasOwnProperty(name)) {
         modules[name] = null;
     }

     return ensure(modules, name, function() {
        ...
     });
}
```

```
angular.module("MyApp", [], function () {
     console.log("Config 1");
});

angular.module("MyApp", [], function () {
     console.log("Config 2");
});
```

Module

- Offers public API to register services/factories/providers and others
- Holds private state
  - Registration queue
  - Config queue
  - Run queue

```
ensure(modules, name, function() {
    var invokeQueue = [];
    var configBlocks = [];
    var runBlocks = [];

    var moduleInstance = {
        provider: invokeLater('$provide', 'provider'),
        factory: invokeLater('$provide', 'factory'),
        config: invokeLater('$injector', 'invoke', 'push', configBlocks),
        run: function(block) {
            runBlocks.push(block);
            return this;
        }
    };

    if (configFn) {
        config(configFn);
    }

    return moduleInstance;
});
```

Registering a Service (Recap)

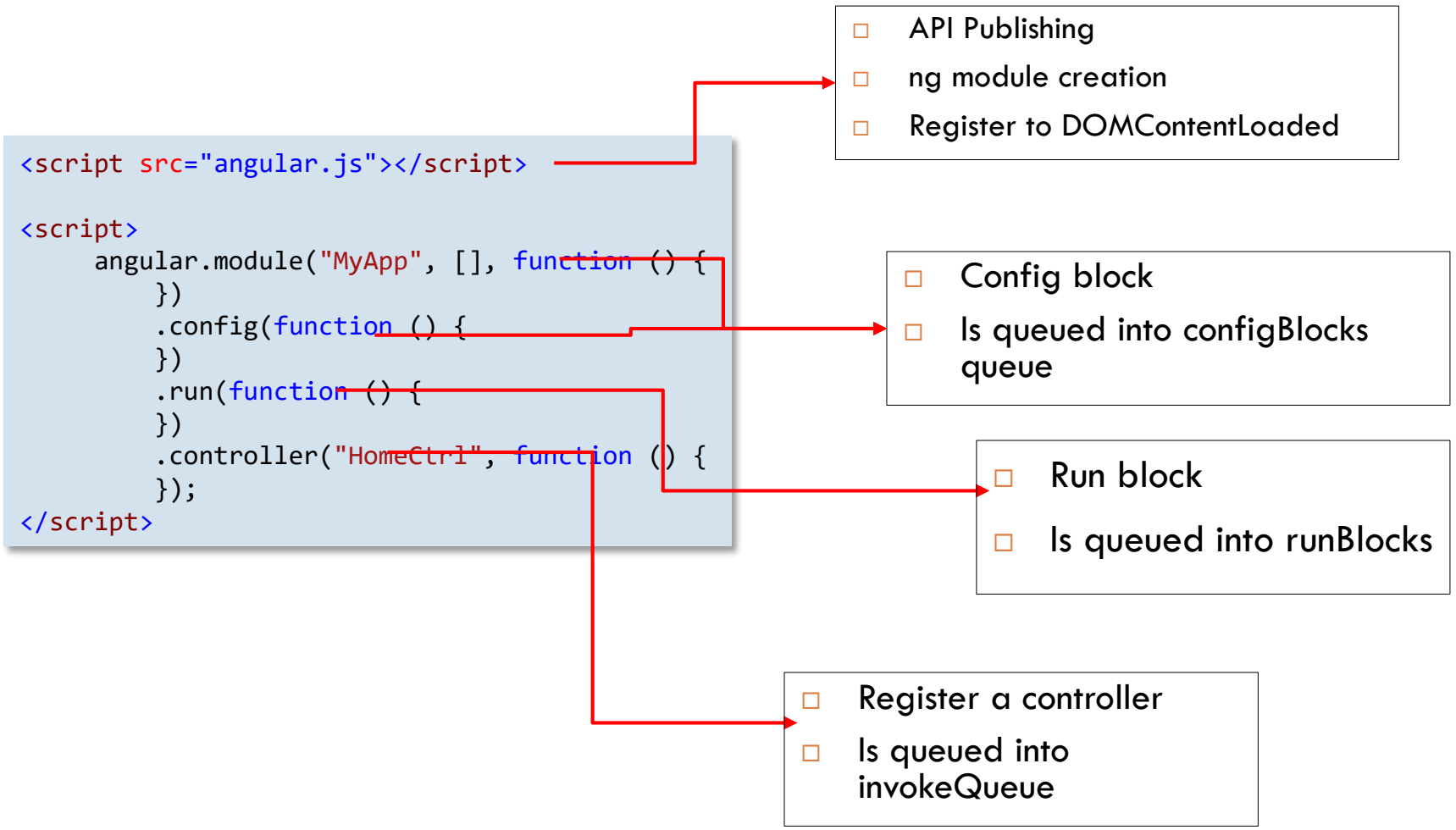

```
angular.module("MyApp", []).factory("OfflineStorage", function () {
     var contacts = [];

     function getAllContacts() {
     ...
     }

     function deleteContact(id) {
     ...
     }

     return {
         getAllContacts: getAllContacts,
         deleteContact: deleteContact,
     };
})
```

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

provider/service/factory Registration

- When application registers a controller/service/… using module.controller/service/… nothing really interesting happens
- Angular queues the request inside an internal queue named invokeQueue
- Later, Angular scans all queues and executes the raw registration implementation
- When?
  - See next slides

Global Initialization (Recap)



angularInit

- Up until now Angular just queued all requests without actually doing anything with them
- Now, it waits for DOMContentLoaded using plain jQuery code and executes angularInit
- angularInit is the place where the real action happens

```
bindJQuery();

publishExternalAPI(angular);

jqLite(document).ready(function() {
    angularInit(document, bootstrap);
});
```

ng-app

- A bit confusing, but ng-app is actually NOT a directive
- It is just a marker which Angular looks for and it serves as the root element for DOM compilation

```
function angularInit(element, bootstrap) {
     var ngAttrPrefixes = ['ng-', 'data-ng-', 'ng:', 'x-ng-'];
     var appElement, module, config = {};

     forEach(ngAttrPrefixes, function (prefix) {
         var name = prefix + 'app';
         var candidate;
         if (!appElement && (candidate = element.querySelector('[' + name.replace(':', '\\:') + ']'))) {
             appElement = candidate;
             module = candidate.getAttribute(name);
         }
     });

     if (appElement) {
         bootstrap(appElement, module ? [module] : [], config);
     }
}
```

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

Challenge

- Suppose you manage a classical web page which is server side rendered
- Then, you want to add Angular magic into it
- The magic need to go into 3 different places inside the DOM while other content must not be effected by Angular
- ng-app is not an option since it can only be used once

Manual Bootstrapping

- Decorate multiple DOM elements with your own my-app directive
- Look for the special attribute using plain DOM selection
- Initialize Angular once per element

```
<script src="~/Scripts/jquery.js"></script>
<script src="~/Scripts/angular.js"></script>

<script>
     $("[my-app]").each(function () {
          var element = $(this);
          angular.bootstrap(element, []);
     });
</script>
```

```
<body>
     <div my-app>
          <h1>NG1</h1>
          <span>1 + 2 =</span><span>{{1+2}}</span>
     </div>
     <div my-app>
          <h1>NG2</h1>
          <span>3 + 4 =</span><span>{{3+4}}</span>
     </div>
</body>
```

Multiple Bootstrapping

- Continuing previous technique
- What happens if we bootstrap with the same module multiple times

```
angular.module("MyApp", []).
     config(function () {
         console.log("Config block");
     });

$("[my-app]").each(function () {
     var element = $(this);
     angular.bootstrap(element, ["MyApp"]);
});
```

- The answer lies inside the bootstrap method

angular.bootstrap

- Publishes the root element of the application into Angular value named $rootElement
- Creates special service named $injector and attaches it to $rootElement
- $injector is responsible for initializing all modules
  - Which means that all built-in services are now available (like $compile)
- Compiles the DOM starting from $rootElement

angular.bootstrap


```
function bootstrap(element, modules) {
     element = jqLite(element);
     if (element.injector()) {
         throw ngMinErr("...");
     }

     modules.unshift(['$provide', function ($provide) {
         $provide.value('$rootElement', element);
     }]);

     modules.unshift('ng');

     var injector = createInjector(modules);
     injector.invoke(['$rootScope', '$rootElement', '$compile', '$injector',
        function bootstrapApply(scope, element, compile, injector) {
            scope.$apply(function () {
                element.data('$injector', injector);
                compile(element)(scope);
            });
        }]
     );

     return injector;
};
```

$injector

- Responsible for dependency injection mechanism
  - Knows how to invoke a function while resolving all its parameters
  - Will cover this mechanism later

- Is also responsible for loading the main module and all its dependencies
  - Collects providers/service from all modules into central cache

createInjector


```
function createInjector(modulesToLoad, strictDi) {
     var loadedModules = new HashMap([], true),
         providerCache = {
            ...
         },

         providerInjector = providerCache.$injector =
             createInternalInjector(providerCache, function (serviceName, caller) {
                 throw $injectorMinErr('unpr', "Unknown provider: {0}", path.join(' <- '));
             }),

         instanceCache = {},

         instanceInjector = instanceCache.$injector =
             createInternalInjector(instanceCache, function (serviceName, caller) {
                 var provider = providerInjector.get(serviceName + providerSuffix, caller);
                 return instanceInjector.invoke(provider.$get, provider, undefined, serviceName);
             }));

     forEach(loadModules(modulesToLoad), function (fn) { instanceInjector.invoke(fn || noop); });

     return instanceInjector;
}
```

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

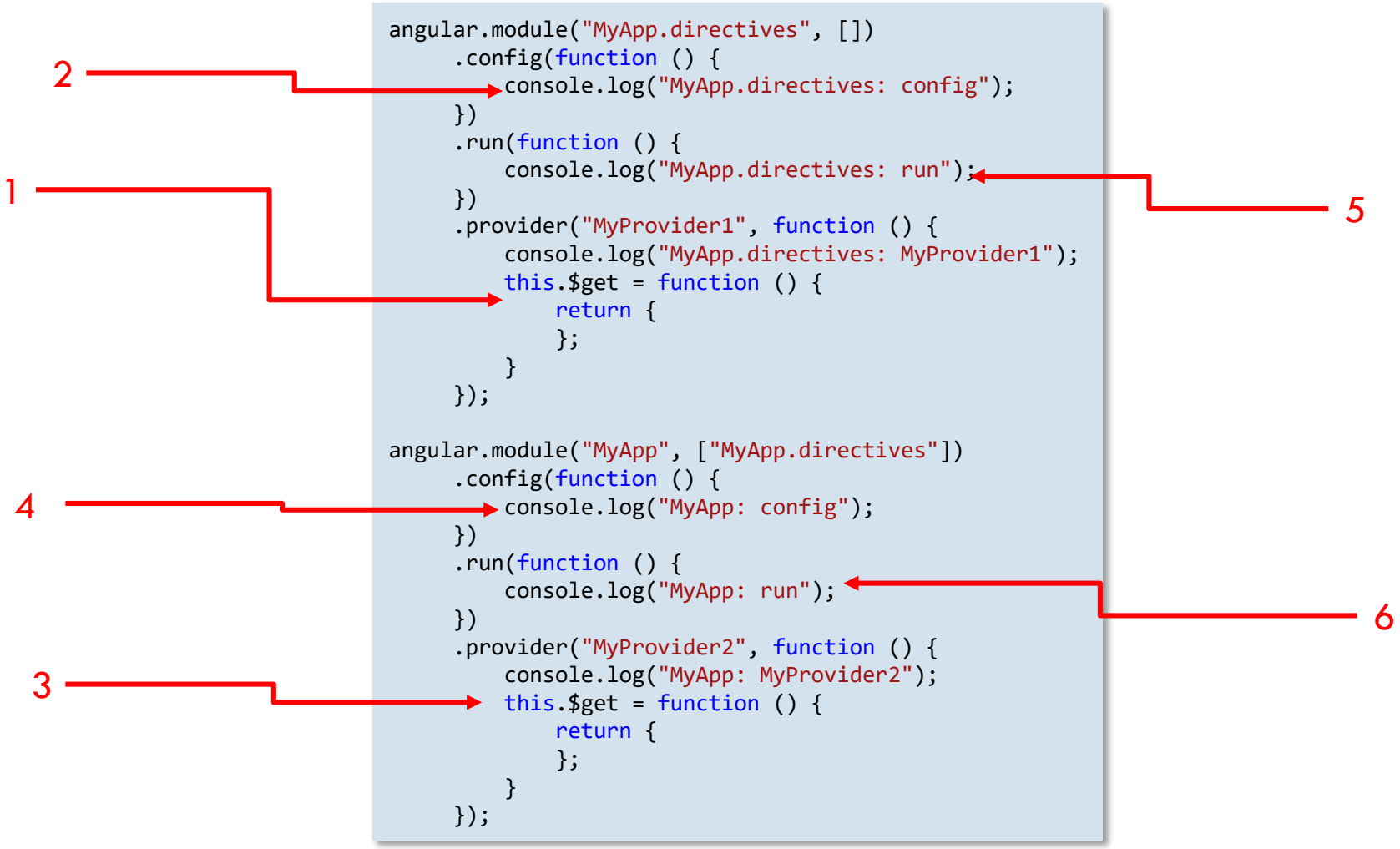
Loading Modules

- Loading a module means
  - Load its dependencies (other modules)
  - Instantiating providers
  - Execute config blocks
  - Execute run blocks
- Order is important
- Config blocks must be executed after all providers were instantiated

loadModules


```
function loadModules(modulesToLoad) {
     var runBlocks = [], moduleFn;

     forEach(modulesToLoad, function (module) {
         if (loadedModules.get(module)) return;
         loadedModules.put(module, true);

         try {
             moduleFn = angularModule(module);

             runBlocks = runBlocks.concat(loadModules(moduleFn.requires)).concat(moduleFn._runBlocks);
             runInvokeQueue(moduleFn._invokeQueue);
             runInvokeQueue(moduleFn._configBlocks);
         } catch (e) {
             throw $injectorMinErr('modulerr', "...");
         }
     });

     return runBlocks;
}
```

Modules Loading Order



Angular Initialization (Cont)


```
bindJQuery();

publishExternalAPI(angular);

jqLite(document).ready(function () {
     var injector = createInjector(modules, config.strictDi);

     injector.invoke(['$rootScope', '$rootElement', '$compile', '$injector',
        function bootstrapApply(scope, element, compile, injector) {

            scope.$apply(function () {
                element.data('$injector', injector);
                compile(element)(scope);
            });

        }]
     );
});
```

$rootScope

- Don't confuse it with $rootElement
  - A DOM element annotated by ng-app
- $rootScope is a service created by a provider named $RootScopeProvider

```
function $RootScopeProvider() {
     this.$get = ['$injector', '$exceptionHandler', '$parse', '$browser',
         function ($injector, $exceptionHandler, $parse, $browser) {
             function Scope() {
             }

             Scope.prototype = {
                 constructor: Scope,
             };

             var $rootScope = new Scope();
             return $rootScope;
         }];
}
```

Angular Initialization


```
bindJQuery();

publishExternalAPI(angular);

jqLite(document).ready(function () {
     var injector = createInjector(modules, config.strictDi);

     injector.invoke(['$rootScope', '$rootElement', '$compile', '$injector',
        function bootstrapApply(scope, element, compile, injector) {

            scope.$apply(function () {
                element.data('$injector', injector);
                compile(element)(scope);
            });

        }]
     );
});
```

Accessing $injector from the DOM

- $injector is attached to the root element using jQuery data method
- This means that 3rd party library can "reach" into Angular using the DOM
- Angular extends jQuery with the injector method

```
$(function () {
     var $injector = $("body div").injector();

     var $rootScope = $injector.get("$rootScope");

     console.log("$rootScope: " + !!$rootScope);
});
```

Angular Initialization - Compilation


```
bindJQuery();

publishExternalAPI(angular);

jqLite(document).ready(function () {
     var injector = createInjector(modules, config.strictDi);

     injector.invoke(['$rootScope', '$rootElement', '$compile', '$injector',
        function bootstrapApply(scope, element, compile, injector) {

            scope.$apply(function () {
                element.data('$injector', injector);
                compile(element)(scope);
            });

        }]
     );
});
```

$compile Service

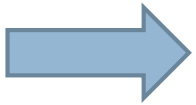
- Compiles an HTML string or DOM into template
- Produces a template function
- The template can be linked to a scope
- Compilation is the process of walking the DOM tree and matching elements to directives

```
function bootstrapApply(scope, element, compile, injector) {
    scope.$apply(function () {
        element.data('$injector', injector);

        compile(element)(scope);
});
```

$compile Usage

- Angular compiles DOM differently then common client side template libraries (Handlebars, jQuery)
- The template function does not produce “clean” HTML

```
var template = angular.element("<span>{{name}}</span>");

var templateFn = $compile(template);

var scope = $rootScope.$new();

var compiledNode = templateFn(scope);
console.log(compiledNode[0].outerHTML);
```

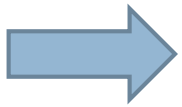


```
<span class="ng-binding ng-scope">
    {{name}}
</span>
```

Live Template

- $compile creates live template
- Live means that the template changes whenever the scope changes
- However, the developer must initiate a “detect changes” request

```
var template = angular.element("<span>{{name}}</span>");
var templateFn = $compile(template);
var scope = $rootScope.$new();
templateFn(scope);

scope.name = "Ori";

scope.$digest();

console.log(template[0].outerHTML);
```



```
<span class="ng-binding ng-scope">
    Ori
</span>
```

scope.$digest

- Walks the list of scope's watchers
- Asks every watcher for the current value
- Compares it to previous value
- If value changed, notifies the watcher
- Works in a recursive manner
  - Through the scope tree
- If a change detected, runs another cycle
  - Stops if no change occurred
  - Or, if maximum allowed cycles was reached (10)

Angular Initialization - Compilation


```
bindJQuery();

publishExternalAPI(angular);

jqLite(document).ready(function () {
     var injector = createInjector(modules, config.strictDi);

     injector.invoke(['$rootScope', '$rootElement', '$compile', '$injector',
        function bootstrapApply(scope, element, compile, injector) {

            scope.$apply(function () {
                element.data('$injector', injector);
                compile(element)(scope);
            });

        }]
     );
});
```

scope.$apply

- A thin wrapper around scope.$digest
- Delegates the request to $rootScope
- Adds exception handling
- Issues dirty checking for all the Scope tree

```
function $apply(expr) {
     try {
         return this.$eval(expr);
     }
     catch (e) {
         $exceptionHandler(e);
     } finally {
         try {
             $rootScope.$digest();
         }
         catch (e) {
             $exceptionHandler(e);
             throw e;
         }
     }
}
```

A to Z

- Creates the public global angular object
- Lets application register its providers/services
- Waits for DOM ready
- Load all modules by instantiating providers and services
- Invoke config and run blocks
- Compiles the DOM
- Initiates digest cycle and let directives do their magic

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

Summary

- Bootstrapping is Angular initialization process
- Converting a plain HTML page to a full functional application