Mapping Study Documents

This resource presents the documents generated as a result of the two mapping studies. Each document contains a list of the most relevant quotations found in the different sources of information. The listed quotations use an identifier to reference the source from which they were taken. In the first sections, we present the considered sources of information with their respective identifiers. These identifiers use the prefix *"BLOG"* to indicate that the source is a blog, *"Q&A"* for communities of questions and answers, and *"DOC"* for AngularJS official documentation.

1.1 Document for Survey About General Aspects of AngularJS

1.1.1 Sources of Information

1.

Identifier: DOC1
Title: Developer Guide: Unit Testing
Access date: September-2015
URL: `https://docs.angularjs.org/guide/unit-testing`

2.

Identifier: DOC2
Title: API Reference: `$compile`
Access date: September-2015
URL: `https://docs.angularjs.org/api/ng/service/$compile`

3.

Identifier: BLOG1
Title: Choosing a JavaScript MVC Framework

Access date: June-2015
URL: `http://www.funnyant.com/choosing-javascript-mvc-framework/`
Author: Craig McKeachie
Author information: Microsoft Certified Solutions Developer.

4.

Identifier: BLOG2
Title: AngularJS vs. Backbone.js vs. Ember.js
Access date: August-2015
URL: `https://www.airpair.com/js/javascript-framework-comparison`
Author: Uri Shaked
Author information: The author works for WatchDox, lectures at the Israel Institute of Technology, and organizes the Tel Aviv Google Developers Group.

5.

Identifier: BLOG3
Title: Unit Testing AngularJS Directives With External Templates
Access date: September-2015
URL: `http://www.bluesphereinc.com/blog/unit-testing-angularjs-directives`
Author: Chris Hooker
Author information: The author is a graduate of MIT, and has over 15 years of software development experience.

6.

Identifier: BLOG4
Title: What's wrong with Angular
Access date: September-2015
URL: `http://www.quirksmode.org/blog/archives/2015/01/the_problem_wit.html`
Author: Daniel Steigerwald
Author information: JavaScript Developer and Consultant. Member of the Google Developers Expert Program.

7.

Identifier: BLOG5
Title: Optimizing AngularJS: 1200ms to 35ms
Access date: August-2015

URL: http://blog.scalyr.com/2013/10/angularjs-1200ms-to-35ms/
Author: Steven Czerwinksi
Author information: Staff Software Engineer at Scalyr.

8.

Identifier: BLOG6
Title: Speeding Up AngularJS Apps With Simple Optimizations
Access date: August-2015
URL: http://www.binpress.com/tutorial/speeding-up-angular-js-with- simple-optimizations/135
Author: Todd Motto
Author information: Director of Web Development at Mozio. He works with JavaScript and AngularJS.

9.

Identifier: BLOG7
Title: AngularJS: My Solution to the `ng-repeat` Performance Problem
Access date: July-2015
URL: http://www.williambrownstreet.net/blog/2013/07/angularjs-my-solution-to-the-ng-repeat-performance-problem/
Author: Thierry Nicola
Author information: Computer Scientist, working with Grails, AngularJS, BackboneJS, and ChaplinJS.

10.

Identifier: BLOG8
Title: AngularJS Performance in Large Applications
Access date: August-2015
URL: https://www.airpair.com/angularjs/posts/angularjs-performance-large-applications
Author: Abraham Polishchuk
Author information: NodeJS/AngularJS engineer at Lanetix.

11.

Identifier: BLOG9
Title: Why You Should Not Use Angularjs
Access date: June-2015
URL: https://medium.com/@mnemon1ck/why-you-should-not-use-

angularjs- 1df5ddf6fc99
Author: Egor Koshelko
Author information: Fullstack NodeJS Developer.

12.

Identifier: BLOG10
Title: AngularJS Critique
Access date: October-2015
URL: http://tutorials.jenkov.com/angularjs/critique.html
Author: Jakob Jenkov
Author information: Founder and CEO at Jenkov Aps.

13.

Identifier: BLOG11
Title: Debugging AngularJS
Access date: October-2015
URL: https://www.ng-book.com/p/Debugging-AngularJS/
Author: Ari Lerner
Author information: Developer with more than 20 years of experience, and co-founder of Fullstack.io. The author has been using AngularJS for a long time and is a recognized expert in the field.

14.

Identifier: BLOG12
Title: A Guide To Transclusion in AngularJS
Access date: October-2015
URL: http://teropa.info/blog/2015/06/09/transclusion.html
Author: Tero Parviainen

15.

Identifier: BLOG13
Title: An Intervention: Why AngularJS is Worse Than a New ASP.NET WebForms
Access date: October-2015
URL: https://medium.com/@benastontweet/an-intervention-4535d835e836
Author: Ben Aston
Author information: Consultant developer specializing in web applications.

16.

Identifier: BLOG14
Title: Scope Creep, a Deep Dive Into Angular's Scope
Access date: October-2015
URL: `http://jonathancreamer.com/working-with-all-the-different-kinds-of-scopes-in-angular/`
Author: Jonathan Creamer
Author information: JavaScript, Ruby, C#, Node, Rails, .NET MVC, MS MVP, Telerik Developer Expert, and IEuserAgent.

17.

Identifier: BLOG15
Title: Adding Clarity to Scope Inheritance in Angular
Access date: October-2015
URL: `http://jonathancreamer.com/adding-clarity-to-scope-inheritance-in-angular/`
Author: Jonathan Creamer
Author information: JavaScript, Ruby, C#, Node, Rails, .NET MVC. MS MVP, Telerik Developer Expert, and IEuserAgent.

18.

Identifier: BLOG16
Title: Mastering AngularJS directives
Access date: October-2015
URL: `http://code.tutsplus.com/tutorials/mastering-angularjs-directives--cms-22511`
Author: Hüseyin Babal
Author information: NodeJS, PHP, Java, Elasticsearch, WordPress Plug-in/Widget Development, MySQL, MongoDB, SEO, Agile Software Development, Cloud Integration, SCRUM.CSM.

19.

Identifier: BLOG17
Title: Creating Custom AngularJS Directives Part I - The Fundamentals
Access date: September-2015
URL: `http://weblogs.asp.net/dwahlin/creating-custom-angularjs-directives-part-i-the-fundamentals`

Author: Dan Wahlin
Author information: JavaScript/ES6, Node.js, Angular, HTML5, jQuery, Node.js, ASP.NET MVC, C#.

20.

Identifier: BLOG18
Title: Directive Controller And Link Timing In AngularJS
Access date: September-2015
URL: `http://www.bennadel.com/blog/2603-directive-controller-and-link-timing-in-angularjs.htm`
Author: Ben Nadel
Author information: Adobe Community Expert as well as an Adobe Certified Professional in Advanced ColdFusion.

21.

Identifier: Q&A1
Title: Template Type Conversion Errors are Difficult to Debug
Access date: September-2015
URL: `https://github.com/angular/angular.js/issues/1974`

22.

Identifier: Q&A2
Title: Why Don't AngularJS Errors in the HTML Show Up in the Console?
Access date: September-2015
URL: `http://stackoverflow.com/q/26227596`

23.

Identifier: Q&A3
Title: Separating JS Logic From Angular Templates
Access date: October-2015
URL: `https://groups.google.com/forum/#!topic/angular/pHSMrphDJhM`

24.

Identifier: Q&A4
Title: AngularJS Use of Inline JavaScript in HTML Attributes is not Bad Practice?
Access date: October-2015
URL: `http://stackoverflow.com/q/14793692`

25.

Identifier: Q&A5
Title: AngularJS: Is `ngClick` a Good Practice Why is There no `ng-event` in AngularJS?
Access date: September-2015
URL: `http://stackoverflow.com/q/14346073`

26.

Identifier: Q&A6
Title: How to not Let Angular Spoil All Your HTML With Logic Code?
Access date: October-2015
URL: `http://www.reddit.com/r/angularjs/comments/2b15dv/how_to_not_let_angular_spoil_all_your_html_with/`

27.

Identifier: Q&A7
Title: Should AngularJS Logic Be Placed in HTML File?
Access date: September-2015
URL: `http://stackoverflow.com/a/24845815`

28.

Identifier: Q&A8
Title: Using `scope.$watch` and `scope.$apply` in AngularJS
Access date: September-2015
URL: `http://stackoverflow.com/q/15112584`

29.

Identifier: Q&A9
Title: What Does `$scope.$apply()` Do?
Access date: September-2015
URL: `http://stackoverflow.com/q/18710478`
Author: Dan Prince

30.

Identifier: Q&A10
Title: Correct Way to Integrate jQuery Plugins in AngularJS
Access date: October-2015
URL: `http://stackoverflow.com/q/16935095`

31.

Identifier: Q&A11
Title: Difference Between the 'controller', 'link' and 'compile' Functions When Defining a Directive
Access date: October-2015
URL: `http://stackoverflow.com/q/12546945`

1.1.2 Relevant Quotations

- *DOC1:* One of the most useful parts of `ngMock` is `$httpBackend`, which lets us mock XHR requests in tests, and return sample data instead.

- *DOC1:* Angular is written with testability in mind, but it still requires that you do the right thing. We tried to make the right thing easy, but if you ignore these guidelines you may end up with an untestable application.

- *DOC1:* Angular also provides the `ngMock` module, which provides mocking for your tests. This is used to inject and mock Angular services within unit tests. In addition, it is able to extend other modules so they are synchronous. Having tests synchronous keeps them much cleaner and easier to work with.

- *DOC2:* There are many different options for a directive.

- *DOC2:* The 'isolate' scope object hash defines a set of local scope properties derived from attributes on the directive's element. These local properties are useful for aliasing values for templates. The keys in the object hash map to the name of the property on the isolate scope; the values define how the property is bound to the parent scope, via matching attributes on the directive's element

- *BLOG1:* With Angular and Ember you usually have to live with the choices made by the authors of the frameworks, which may or may not suit your project needs and personal style.

- *BLOG1:* It's important to understand how big a download of each of these frameworks is and what you are getting for that extra weight in your application.

- *BLOG2:* Angular helps you categorize your application building blocks into several types: Controllers, Directives, Factories, Filters, Services and Views (templates).

- *BLOG2:* The Digest Cycle of angular, which takes care of the "Magical" dirty checking, has the tendency to surprise developers. It is easy to forget to call `$digest()` when running in non-Angular context.

- *BLOG2:* Putting logic inside the templates makes it harder to test, as it becomes impossible to test it in isolation.

- *BLOG2:* Mistakes such as misspelling a directive name or calling an undefined scope function are silently ignored and can be challenging to find.

- *BLOG2:* Failing to understand scope inheritance causes many cases of frustrated developers.

- *BLOG2:* Wrapping your head around all the concepts such as compiling function, pre/post linking functions, the different scope kinds (transclusion / isolate / child scope) and all the other configuration settings for directives takes some time to master.

- *BLOG2:* Promises play a main role in the Angular cast.

- *BLOG3:* I hate including HTML inline as a string in Javascript, so I definitely wanted to use `templateUrl`. This was all fine and good, and I got my directive up and running. Then I decided it was time to unit test it. That's where the real fun began.

- *BLOG3:* I discovered that while getting some simple unit tests running wasn't too hard using just Jasmine, unit testing my `templateUrl` directive was no simple matter.

- *BLOG3:* The problem with using `templateUrl` is that Angular uses an HTTP request to go get the file. However, in a unit-testing environment, you don't have the full web server environment and can't actually make the HTTP request. So, you'll get an error when you try to test the directive.

- *BLOG3:* You want to pre-process your HTML template and convert in into Javascript, which can be testing without any need for HTTP requests.

- *BLOG4:* Dirty checking, accessors (Ember and Backbone), `Object.observe` and all that stuff. Wrong! It's slow and brittle and it will consume mobile battery like hungry dog, for no reason.

- *BLOG4:* Angular is HTML parser. I really don't want to debug any string based parser instead of my code.
- *BLOG5:* This shows the power of AngularJS for encapsulation and separation of concerns.
- *BLOG6:* You'll get an error thrown from Angular if you're calling `$scope.$apply` in the "wrong" place, usually too high up the call stack.
- *BLOG7:* I started with BackboneJS as frontend MVC, but soon switch to AngularJS because of the concept AngularJS adds to HTML.
- *BLOG7:* AngularJS offers just the right structure for code organization. After some effort to learn to use directives and services, AngularJS forces me to get my code clean.
- *BLOG8:* Isolate Scope and Transclusion are some of the most exciting things about Angular. They allow the building of reusable, encapsulated components, they are syntactically and conceptually elegant and a core part of what makes Angular Angular.
- *BLOG9:* Errors in bindings don't fire at all.
- *BLOG9:* You can't put a breakpoint inside {{ this expression }}.
- *BLOG9:* When you write in AngularJS you put your logic into your HTML (`ng-repeat`, `ng-show`, `ng-class`, `ng-model`, `ng-init`, `ng-click`, `ng-switch`, `ng-if`). Existence of such logic is not as bad as the fact that it is impossible to test this logic with unit tests, this logic can't be debugged and errors don't fire from markup (but this code contains very important logic).
- *BLOG9:* Errors that occurred in JavaScript are caught by the internal angular interceptor, and interpreted by browser as caught errors (everything that happens in AngularJS, stays in AngularJS).
- *BLOG9 (comment):* The null reference exceptions (in templates (HTML)) are swallowed in order to not break the first renders working properly.
- *BLOG9:* It's unclear why it was necessary to introduce several ways to do the same thing.
- *BLOG9:* It is without a doubt the most common error that absolutely every AngularJS developer faces (Scope inheritance).

- *BLOG9:* There is no logical reason to separate logic for 3 methods (`compile`, `link`, `controller`), all this can be easily implemented in a single method.
- *BLOG9:* Even in order to integrate some code in the angular world, for example some jQuery plugin, you need to wrap it in a directive.
- *BLOG9:* Since server side rendering adds logic into your HTML and AngularJS writes logic in HTML, there is no clear separation of concerns and as a result you get spaghetti code.
- *BLOG9:* The only good thing that AngularJS has is that it forces developers to break their logic into modules, and code becomes more granulated.
- *BLOG10:* Before AngularJS it was "best practice" to keep function calls out of the HTML. For instance, you should not use the `onclick` event attributes on HTML elements, but rather attach event listeners via JavaScript. Somehow that was forgotten with AngularJS, and now we are back to embedding JavaScript function calls in the HTML.
- *BLOG10:* In order to "teach HTML new tricks" you end up with HTML full of non-HTML elements and attributes.
- *BLOG10:* We are back to embedding JavaScript function calls in the HTML.
- *BLOG11:* Angular Batarang is a Chrome extension developed by the Angular team at Google that integrates very nicely as a debugging tool for Angular apps.
- *BLOG12:* Based on what I've heard, I'm not alone in this. Transclusion is one of those things people often mention when they talk about their difficulties with Angular.
- *BLOG12:* I think the bigger problems (with transclusion) are tangential:
  - The API is tricky, with its higher-order functions and "magic arguments".
  - Parts of the API are deprecated, and it can be hard to figure out which parts.
  - The documentation uses lots of big words ("a transclude linking function pre-bound to the correct transclusion scope").
  - There are actually two separate features provided by the same API: Regular transclusion and element transclusion.

- *BLOG13:* AngularJS brings with it a domain specific language: transclusion, scope, directives, modules, factories and services.
- *BLOG13:* You could probably sum up AngularJS with this single word: it makes the inclusion of DOM fragments in your page sound novel and complicated.
- *BLOG14:* Getting used to the concept of scope in Angular is among the more difficult concepts to fully grok when first being introduced to the magical land of Angular.
- *BLOG14:* Because of the fact that the scopes do inherit from other scopes, if you create a primitive value (string, number, boolean) on a parent scope, the child scope will have an be able to manipulate the value.
- *BLOG15:* The fact is prototypical inheritance in JavaScript presents confusion to a lot of folks out there. In terms of Angular.js that may explain why the concept of `$scope` is difficult to grok.
- *BLOG16:* Directives are one of the most powerful components of AngularJS, helping you extend basic HTML elements/attributes and create reusable and testable code.
- *BLOG16:* So, what is the link function there? Simply, the link function is the function that you can use to perform directive-specific operations. The directive is not only rendering some HTML code by providing some inputs. You can also bind functions to the directive element, call a service and update the directive value, get directive attributes if it is an `E` type directive, etc.
- *BLOG16:* Every directive has its own scope, but you need to be careful about the data binding with the directive declaration.
- *BLOG16:* The main advantage of the directive is that it's a reusable component that can be used easily.
- *BLOG16:* When you use directives inside the template, what you see on the page is the compiled version of the directive. Sometimes, you want to see the actual directive usage for debugging purposes. In order to see the uncompiled version of the current section, you can use `ng-non-bindable`.
- *BLOG17:* AngularJS directives can be a bit intimidating the first time you see them. They offer many different options, have a few cryptic features (and cryp-

tic is my politically correct term for "what were they thinking here?"), and are generally challenging at first.

- *BLOG17:* In addition to performing data binding operations with templates, directives can also be used to manipulate the DOM. This is done using the link function shown earlier.

- *BLOG18:* I've talked about the timing of directives in AngularJS a few times before. But, it's a rather complicated topic, so I don't mind digging a bit deeper.

- *BLOG18:* As the DOM (Document Object Model) is compiled by AngularJS, the directive controllers and link functions execute at different parts of the compile lifecycle.

- *BLOG18:* This is an important difference. While you can only access the DOM tree in the bottom-up linking phase, the directive controller can provide a hook into the top-down lifecycle. This can be critical if you have to handle DOM events based on when elements of the DOM tree came into existence. The linking phase can never give you that because it's executed in reverse.

- *Q&A1:* Template type conversion errors are difficult to debug.

- *Q&A1:* FYI the only line in the stack trace that references my actual document (index.html, line 500), line 500 is past the end of the source document, so it's referencing some angular script injected into the header, thus does not seem to be useful for debugging.

- *Q&A2:* Expressions are meant to handle very simple logic, and are optimized for simplicity and not breaking your app.

- *Q&A8:* I don't understand how to use `$scope.$watch` and `$scope.$apply`. The official documentation isn't helpful.

- *Q&A3:* It seems as though the developers built the entire framework and wrote the documentation promoting the idea that it's A-OK to frankenstein JavaScript expressions into your HTML.

- *Q&A3:* Architecturally, the way jQuery solves problems is not the best. Clayton pointed out that it's difficult to test, but it also leaves intent nebulous. Directives extend DOM functionality (and we can test them within and without the view) and so belong within the DOM. When you use jQuery to programmatically insert

on-click behavior, whoever is reading the view (designer or developer) doesn't know what something does. With AngularJS, that becomes quite clear.

- *Q&A4:* As I read through the Angular tutorials, I really like a lot of it, but isn't "`ng-click`" the equivalent of an inline `onClick`? My understanding was that the JavaScript community had determined inline JavaScript event handlers in your HTML was "bad practice".

- *Q&A5:* I understand that `ng-click` is technically not the same as `onclick`, but both are placed in the markup. I thought that would be a "bad practice"? Why is this one of the core concepts of AngularJS, if most people say this is "bad"? I thought it would be better to select the DOM element from JavaScript and not to place logic in the markup.

- *Q&A6:* I'm relatively new to angular, and the more I use it the more I insert ng-this and ng-that into my HTML and soon I have a mess like this.

- *Q&A7:* I am very new to AngularJS. Now when I saw the code, I was very curious about all the logic that is placed in the HTML code.

- *Q&A7:* Now about adding logic to the views, if we are talking about business logic then it's a big no no. Use a method on your controller that will evaluate stuff using the service.

- *Q&A7:* If we are talking about `ng-if`/`ng-show` conditions then only if they are small and "readable" conditions I would add them to the view. When it's more than that, I move them to the controller for debugging issues and since I believe the HTML should be readable.

- *Q&A9:* I've been using `$scope.$apply()` to update the bindings for my models when I receive data through websockets in my Angular apps and it works. But what does it actually do and why does it need to be called to achieve the update?

- *Q&A10:* I was wondering what is the correct way to integrate jQuery plugins into my angular app. I've found several tutorials and screen-casts but they seem catered to a specific plugin.

- *Q&A10:* If you are using a jQuery plugin, do not put the code in the controller. Instead create a directive and put the code that you would normally have inside the link function of the directive.

- *Q&A11:* Some places seem to use the controller function for directive logic and other use link. The tabs example on the angular homepage uses controller for one and link for other directive. What is the difference between two?

1.2 Document for Survey About Performance of AngularJS

1.2.1 Sources of Information

1.

Identifier: DOC1
Title: Developer Guide: Scopes
Access date: June-2015
URL: `https://docs.angularjs.org/guide/scope`

2.

Identifier: DOC2
Title: API Reference: `$rootScope.Scope`
Access date: June-2015
URL: `https://docs.angularjs.org/api/ng/type/$rootScope.Scope`

3.

Identifier: DOC3
Title: API Reference: `ngRepeat`
Access date: June-2015
URL: `https://docs.angularjs.org/api/ng/directive/ngRepeat`

4.

Identifier: BLOG1
Title: Why You Should Not Use Angularjs
Access date: **June-2015**
URL: `https://medium.com/@mnemon1ck/why-you-should-not-use-angularjs-1df5ddf6fc99`
Author: Egor Koshelko
Author information: Senior Frontend Developer.

5.

Identifier: BLOG2
Title: AngularJS vs. Backbone.js vs. Ember.js
Access date: June-2015
URL: `https://www.airpair.com/js/javascript-framework-comparison`
Author: Uri Shaked
Author information: The author works for WatchDox, lectures at the Israel Institute of Technology, and organizes the Tel Aviv Google Developers Group.

6.

Identifier: BLOG3
Title: The Problem With Angular
Access date: July-2015
URL: `http://www.quirksmode.org/blog/archives/2015/01/the_problem_wit.html`
Author: Peter-Paul Koch
Author information: Mobile platform strategist, consultant, and trainer.

7.

Identifier: BLOG4
Title: What's wrong with Angular
Access date: July-2015
URL: `http://www.quirksmode.org/blog/archives/2015/01/the_problem_wit.html`
Author: Daniel Steigerwald
Author information: JavaScript Developer and Consultant. Member of the Google Developers Expert Program.

8.

Identifier: BLOG5
Title: Lessons learned from AngularJS
Access date: July-2015
URL: `http://lhorie.github.io/mithril-blog/lessons-learned-from-angular.html`
Author information: Developer in the project Mithril.

9.

Identifier: BLOG6
Title: AngularJS: The Bad Parts
Access date: July-2015
URL: `http://larseidnes.com/2014/11/05/angularjs-the-bad-parts/`
Author: Lars Eidnes
Author information: Developer from Trondheim, Norway.

10.

Identifier: BLOG7
Title: Angular Bad Parts, Part 2: Performance
Access date: July-2015
URL: `http://www.fse.guru/angular-bad-parts-part-2`
Author: Alexey Migutsky
Author information: Passionate software engineer, working with Scala, NodeJS and frontend technologies like Angular, ReactJS, HTML5, CSS.

11.

Identifier: BLOG8
Title: Optimizing AngularJS: 1200ms to 35ms
Access date: June-2015
URL: `http://blog.scalyr.com/2013/10/angularjs-1200ms-to-35ms/`
Author: Steven Czerwinksi
Author information: Staff Software Engineer at Scalyr.

12.

Identifier: BLOG9
Title: Speeding Up AngularJS Apps With Simple Optimizations
Access date: July-2015
URL: `http://www.binpress.com/tutorial/speeding-up-angular-js-with-simple-optimizations/135`
Author: Todd Motto
Author information: Director of Web Development at Mozio. He works with JavaScript and AngularJS.

13.

Identifier: BLOG10
Title: 11 Tips to Improve AngularJS Performance

Access date: July-2015
URL: `http://www.alexkras.com/11-tips-to-improve-angularjs-performance/`
Author: Alex Kras
Author information: Software Engineer at Support.com.

14.

Identifier: BLOG11
Title: AngularJS: My Solution to the `ng-repeat` Performance Problem
Access date: July-2015
URL: `http://www.williambrownstreet.net/blog/2013/07/angularjs-my-solution-to-the-ng-repeat-performance-problem/`
Author: Thierry Nicola
Author information: Computer Scientist, working with Grails, AngularJS, BackboneJS, and ChaplinJS.

15.

Identifier: BLOG12
Title: AngularJS Performance in Large Applications
Access date: July-2015
URL: `https://www.airpair.com/angularjs/posts/angularjs-performance-large-applications`
Author: Abraham Polishchuk
Author information: NodeJS/AngularJS engineer at Lanetix.

16.

Identifier: BLOG13
Title: Why The World Needed Another AngularJS Grid
Access date: July-2015
URL: `http://www.angulargrid.com/why-the-world-needed-another-angularjs-grid/`
Author: Niall Crosby
Author information: The author has been writing software for 20 years, lately focusing on full stack Java/Javascript/AngularJS.

17.

Identifier: BLOG14
Title: Angular Non-trivial Performance Hints

Access date: July-2015
URL: `http://codetunes.com/2015/angular-non-trivial-performance-hints/`
Author: Radek Markiewicz

18.

Identifier: Q&A1
Title: How Does Data Binding Work in AngularJS?
Access date: July-2015
URL: `http://stackoverflow.com/questions/9682092/how-does-data-binding-work-in-angularjs/9693933#9693933`
Author: Misko Hevery
Author information: Contributor to AngularJS.

19.

Identifier: Q&A2
Title: How Do I Measure the Performance of my AngularJS App's Digest Cycle?
Access date: June-2015
URL: `http://stackoverflow.com/q/23066422/5244036`
Author: Gil Birman

20.

Identifier: Q&A3
Title: AngularJS Scaling & Performance
Access date: June-2015
URL: `http://stackoverflow.com/q/17656397/5244036`

21.

Identifier: Q&A4
Title: How To Improve Performance Of `ngRepeat` Over A Huge Dataset (Angular.js)?
Access date: July-2015
URL: `http://stackoverflow.com/q/17348058/5244036`

22.

Identifier: Q&A5
Title: How To Analyze Performance Benchmark In AngularJS Components?

Access date: July-2015
URL: `http://stackoverflow.com/a/27400474/5244036`

23.

Identifier: Q&A6
Title: Ways To Improve AngularJS Performance Even With Large Number Of Dom Elements
Access date: July-2015
URL: `http://stackoverflow.com/a/16128418/5244036`

24.

Identifier: Q&A7
Title: How To Speed Up An AngularJS Application?
Access date: July-2015
URL: `http://stackoverflow.com/q/15643467/5244036`

25.

Identifier: Q&A8
Title: Angular.js Performance Issues
Access date: July-2015
URL: `http://stackoverflow.com/a/14486570/5244036`
Author: Ben Lesh

1.2.2 Relevant Quotations

- *DOC1:* If a `$watch` changes the value of the model, it will force additional digest cycle.
- *DOC1:* Dirty checking the scope for property changes is a common operation in Angular and for this reason the dirty checking function must be efficient. Care should be taken that the dirty checking function does not do any DOM access, as DOM access is orders of magnitude slower than property access on JavaScript object.
- *DOC1:* Dirty checking can be done with three strategies: By reference, by collection contents, and by value.
- *DOC2:* `$destroy()` must be called on a scope when it is desired for the scope and its child scopes to be permanently detached from the parent and thus stop participating in model change detection and listener notification.

- *DOC2:* Usually, you don't call `$digest()` directly in controllers or in directives. Instead, you should call `$apply()` (typically from within a directive), which will force a `$digest()`.

- *DOC3:* The `ngRepeat` directive instantiates a template once per item from a collection.

- *DOC3:* Special properties are exposed on the local scope of each template instance: `$index`, `$first`, `$middle`, `$last`, `$even` and `$odd`. Creating aliases for these properties is possible with ngInit. This may be useful when, for instance, nesting ngRepeats.

- *DOC3:* `ngRepeat` (as well as other ng directives) supports extending the range of the repeater by defining explicit start and end points by using `ng-repeat-start` and `ng-repeat-end`.

- *BLOG1:* It's slow, and it is especially all turns bad on mobile platforms and when you write something complex.

- *BLOG1:* Angular even imposes restrictions on how rich UI you can write and this is not some ephemeral or distant limit you will never experience. It's only 2000 watchers, and if you develop more or less large applications, you will undoubtedly run into this limitation.

- *BLOG1:* When the user loads the page he may see {{expressions in brackets}}. For these purposes AngularJS introduces new directives: ngCloack, ngBind.

- *BLOG1 (comment):* That number is made up. Not true. Originally shared by Misko on StackOverflow. I'll get him to correct it. Complexity of an expression is often more important than the pure number of expressions. But in general when it comes to pure count we are talking about orders of magnitude bigger numbers than 2k (based on our benchmarks).

- *BLOG1 (comment):* That may be the reason why one way data-binding is now on 1.3. Fortunately, AngularJS evolves.

- *BLOG2:* In general, for pages with a lot of interactive elements, Angular becomes really slow. A good rule of thumb is not to have more than 2,000 active bindings on the same page.

- *BLOG2:* Two-way data binding saves a lot of boilerplate code.

- *BLOG2:* The automatic Dirty Checking means that you don't have to access your model data with getters and setters — you can modify any property of an arbitrary scope object and angular will automatically detect the change and notify all the watchers for that property.

- *BLOG3:* All code snippets in are Angular instructions. The problem is that there is no way for Angular to discover these instructions except by parsing the entire DOM — a very expensive process, especially on mobile.

- *BLOG3:* What worries me is that this non-performant mode is Angular's default (talking about `ng-repeat` directive).

- *BLOG4:* Angular is slow. And with dirty checking and HTML parsing always will be. Do I have to tell something about consequences for mobile web apps?

- *BLOG5:* Another problem related to Angular's re-implementation of scope is the steep performance degradation when things like grids grow past a modest size. Often things look ok with test data, but with production-level data volume, you are suddenly forced to discover about Angular internal concepts like watchers, dirty checking, apply cycles, etc, or you need to do massively time consuming audits of the code base in order to figure out the feasibility of adding the `ng-bindOnce` plugin, or maybe you need to completely rewrite a top-level `ng-repeat` to use the `ng-grid` plugin instead.

- *BLOG5:* It's extremely difficult to reason about Angular performance, and standard tools don't help very much.

- *BLOG6:* The digest loop: It scans through everything that has such a binding, and sees if it has changed by comparing its value to a stored copy of its value. It then scans through everything looking for changes again. This keeps going until no more changes are detected.

- *BLOG6:* Changing anything in the application becomes an operation that triggers hundreds or thousands of functions looking for changes.

- *BLOG6:* This is a fundamental part of what Angular is, and it puts a hard limit on the size of the UI you can build in Angular while remaining performant.

- *BLOG6:* It's not hard to end up with more than 2000 bindings.

- *BLOG6:* On mobiles the performance was dreadful.

- *BLOG6:* There are ways to make bindings happen only once, and with Angular version 1.3 these are included by default. It nevertheless requires ditching what is perhaps the most fundamental abstraction in Angular.
- *BLOG7:* Do you know that each time you call `$scope.$apply()` you actually call `$rootScope.$apply()` and this call updates all scopes and run all your watches?.
- *BLOG7:* Moreover, `$rootScope.$apply()` is called each time when:
  - `$timeout` handler is invoked (almost all debounce services are broken by design)
  - `$http` receives a response (yeah, if you have a polling implemented on `$http`...)
  - Any DOM handler is called (have you throttled your `ng-mouseovers`? They actually invoke ALL your `$watches`, and built-in digest phasing does not really help)
- *BLOG8:* In our early tests, we found that advancing to the next log page could take several agonizing seconds of JavaScript execution. Worse, unrelated actions (such as clicking on a navigation dropdown) now had noticeable lag.
- *BLOG8:* We created a directive that "hides" the change watchers of its children. To accomplish this, we had to slightly break the AngularJS abstraction layer.
- *BLOG8:* A straightforward AngularJS implementation of the log view took 1.2 seconds to advance to the next page, but with some careful optimizations we were able to reduce that to 35 milliseconds. We had to break few rules to implement them.
- *BLOG8 (comment):* AngularJS 1.3 introduces one-time binding to reduce watching and digest loops which is relevant to this.
- *BLOG8 (comment):* Yes, with the current implementation's approach, if the AngularJS team changed the `$scope.$watch` implementation to use a different non-public variable that we depend on, our change would break and we would have to fix it..
- *BLOG9:* AngularJS dropped a really interesting feature recently in the beta version of 1.3.0: the ability to render data once and let it persist without being affected by future Model updates.

- *BLOG9:* When `$scope.$apply()` is called it runs `$rootScope.$digest()`. This is what actually kicks off the internal digest cycle.

- *BLOG9:* Instead of `$scope.$apply`, we could turn to `$scope.$digest`, which runs the exact same digest loop, but is executed from the current scope downwards through its children. The only caveat to this approach is that if you're dependent on two-way binding between Objects from the parent scope, the parent scope won't be updated until the next `$rootScope` full digest cycle.

- *BLOG9:* Avoid `ng-repeat` where possible. The `ng-repeat` directive is most likely the worst offender for performance concerns.

- *BLOG9:* Instead of rendering a global navigation using `ng-repeat`, we could create our own navigation using the `$interpolate` provider to render our template against an Object and convert it into DOM nodes.

- *BLOG9:* Another offender that will increase `$$watcher` counts are the core Angular directives such as `ng-show` and `ng-hide`. Although these might not immediately increase watcher counts dramatically, they can easily stack up in the hundreds inside an `ng-repeat`.

- *BLOG9:* Angular also provides us with Directives such as `ng-mouseenter`, these can be more costly too as they're not only binding an event listener, they become a part of the digest cycle adding to the application weight.

- *BLOG9:* Limit DOM filters. Angular includes a `$filter` provider, which you can use to run filters in your JavaScript before parsing into the DOM.

- *BLOG10:* In Angular it is fairly easy to add so many watchers that your app will slow down to a crawl.

- *BLOG10:* Angular 1.3 added `::` notation to allow one time binding. Angular will wait for a value to stabilize after its first series of digest cycles; after that, Angular will remove the watcher forgetting about that binding. If you are on pre 1.3 version of Angular you can use this library to achieve similar results.

- *BLOG10:* Use `$watchCollection` instead of mcode$watch (with a 3rd parameter).

- *BLOG10:* If you know there is going to be a lot of changes coming from an `ng-model`, you can de-bounce the input. For example if you have a search input

like Google, you can de-bounce it by setting the following `ng-model` option: `ng-model-options=''{ debounce: 250 }''`. This will ensure that the digest cycle due to the changes in this input model will get triggered no more than once per 250ms.

- *BLOG10:* Use `ng-if` instead of `ng-show` (but confirm that `ng-if` is actually better for your use case).
- *BLOG10:* Console.time is a great API, and I found it particularly helpful when debugging issues with Angular performance.
- *BLOG10:* Use native JavaScript or Lodash for slow functions: In my tests I got a significant performance boost by simply re-writing some of the basic logic with lodash, instead of relying on built-in Angular methods (which have to account for much more generic use cases).
- *BLOG10:* Use Chrome Timeline and Profiler to identify performance bottlenecks: I like to think of myself as a Chrome Dev Tools power user. But it's not often that I get a to use the Timeline and Profiler views. In this project, both were extremely helpful.
- *BLOG10 (comment):* I just found yesterday a less known easy way to speed up angular : turn off angular's debug features!
- *BLOG11:* After some testing, the response times were very satisfying, however the application was blocking for 500-700ms after each refresh of the view. Quickly I stumble upon the problem: `ng-repeat`.
- *BLOG11:* I was expecting this bad performance due to the internal working of AngularJS.
- *BLOG11:* For `ng-repeat` you often want the items to be rendered and then forget about them as they are not updated from the client side. No two way binding required.
- *BLOG11:* Bindonce (plugin or third-party module) allows to create a template to render that uses jQuery to render the HTML, so no two-way binding. This solved the problem of the slow rendering, but created another one namely that scrolling now had bad performance (on mobile scrolling was not controllable).
- *BLOG11:* Virtual Scrolling just renders the visible elements and removes invisible elements from the DOM.

- *BLOG11:* Final solution: Upgrade to AngularJS 1.1.5 and use limitTo together with Infinite scrolling.
- *BLOG11:* I slightly adapted the Infinite Scroll directive to make scrolling within a container possible that does not have height 100% of window.
- *BLOG11:* The limitTo with Infinite Scrolling plays very neat together with the Track By feature for `ng-repeat`.
- *BLOG12:* It is important to understand what causes an AngularJS application to slow down.
- *BLOG12:* A fantastic tool for benchmarking one's code is jsPerf. The Chrome Dev Tools have a fantastic JavaScript profiler.
- *BLOG12:* There are several things to be said about performant Javascript that are not necessarily limited to Angular.
- *BLOG12:* Now that we have discussed JavaScript performance, it is important to understand a few key Angular concepts that are somewhat "under the hood".
- *BLOG12:* Of particular importance to writing Angular JS applications in general and performance in particular is the digest cycle. Effectively, every scope stores an array of functions `$$watchers`.
- *BLOG12:* When any value in scope changes, all watchers in the `$$watchers` array will fire, and if any of them modify a watched value, they will all fire again.
- *BLOG12:* We should think through our data model and attempt to limit the complexity of our objects. Use a custom serializer to only return the subset of keys that your Angular application absolutely must have.
- *BLOG12:* Never watch a function result directly. This function will run on every digest cycle.
- *BLOG12:* Every time `$watch` is called on a scope value, or a value is bound from the DOM with interpolation, an `ng-repeat`, an `ng-switch`, and `ng-if`, or any other DOM attribute/element, a function gets added to the `$$watchers` array of the innermost scope.
- *BLOG12:* If non-Angular code is run through `$scope.$apply()`, this will immediately kickstart the digest cycle.

- *BLOG12:* Design with Angular in mind.
- *BLOG12:* Angular provides the ability to watch entire objects by passing a third, optional true parameter to `$scope.$watch`. This is a terrible idea.
- *BLOG12:* `ng-repeat` does some pretty heavy DOM manipulation (not to mention polluting `$$watchers`). Keep any lists of rendered data small whether through pagination or infinite scroll.
- *BLOG12:* Avoid using filters if at all possible. They are run twice per digest cycle and do not actually remove any part of the collection from memory.
- *BLOG12:* It is also important to avoid a global list refresh when using `ng-repeat`. Doing something like `$scope.listBoundToNgRepeat = serverFetch()` will cause a complete recalculation of the entire list, causing transcludes to run and watchers to fire for every individual element. This is a very expensive proposition. There are two ways around this. One is to maintain two collections and `ng-repeat` over the filtered set. The other is to use *track by* to specify your own key.
- *BLOG12:* Track by approach only works when a field on the repeated object can be guaranteed to be unique in the set.
- *BLOG12:* `ng-hide` and `ng-show` simply toggle the CSS display property: Any scopes will exist, all `$$watchers` will fire, etc. `ng-if` and `ng-switch` actually remove or add the DOM completely. Unfortunately this results in a case by case judgment call.
- *BLOG12:* A common source of slow Angular applications is incorrect use of `ng-hide` and `ng-show` over `ng-if` or `ng-switch`. The distinction is nontrivial, and the importance can not be overstated in the context of performance.
- *BLOG12:* The problem arises in the fact that `$scope.$apply` starts at `$rootScope` and walks the entire scope chain causing every scope to fire every watcher.
- *BLOG12:* In general, `$scope.$watch` is indicative of bad architecture.
- *BLOG12:* `$on`, `$broadcast` , and `$emit`, like `$watch`, are slow as events (potentially) have to walk your entire scope hierarchy.
- *BLOG12:* `evalAsync` can greatly improve your page's performance.

- *BLOG12:* By creating a new scope with Isolate Scope or Transclusion, we are creating a new object to track, adding new watchers, and therefore slowing down our application.
- *BLOG12:* Angular provides many pre-rolled DOM event directives. `ng-click`, `ng-mouseenter`, `ng-mouseleave`, etc. All of these call `$scope.$apply()` every time the event occurs. A much more efficient approach is to bind directly with addEventListener, and then use `$scope.$digest` as necessary.
- *BLOG13:* You see, Angular JS is great for form based applications, where you don't have hundreds of bound values laid out in a grid and scrolling.
- *BLOG13:* The extra logic that AngularJS puts in behind the scenes doesn't work in your favor when building something like a complex grid.
- *BLOG14:* Avoid complicated watcher's evaluation.
- *BLOG14:* There is no way to think about the higher level of Angular performance without understanding a few things:
  - How digest cycle works in general
  - Difference between `$scope.$apply()` and `$scope.$digest()`
  - List of Angular core features triggering digest cycle
- *Q&A1:* You can't really show more than about 2000 pieces of information to a human on a single page. Anything more than that is really bad UI, and humans can't process this anyway.
- *Q&A1:* Unfortunately it is way too easy to add a slow comparison into AngularJS, so it is easy to build slow applications when you don't know what you are doing.
- *Q&A2:* What is a simple way to measure the duration of the angularjs digest cycle?
- *Q&A3:* You need to create custom directives in order to curb the performance issues with angular. Unlike ember angular comes with all the bells and whistles turned on and it's up to you to tone it down.
- *Q&A4:* Most straightforward approach (fetch data, put it into `$scope`, let `ng-repeat=“”` do its job) works fine, but it freezes the browser for about half of a minute when it starts inserting nodes into DOM.

- *Q&A5:* The `$apply` function itself is relatively light (you can inspect it in the angular source). It is the process of evaluating watchers and comparing values (dirty-checking) during a `$digest` that can get expensive.

- *Q&A6:* I think that the best way to solve performance issues is to avoid using high level abstractions (AngularJS `ng-repeat` with all corresponding background magic) in such situations. AngularJS is not a silver bullet and it's perfectly working with low level libraries.

- *Q&A7:* The thing you can do that will speed up your Angular app the most is to reduce those bindings where you can. One way to do this would be to create a directive that built out the table for you with DOM manipulation rather than using `ng-repeat`s.

- *Q&A8:* Every `ngRepeat` you set up, sets up a watch. Every `{{binding}}` or `ngModel` you do inside of that repeat, sets up another watch, and so on. Each of those creates function and object instances, and also needs to be processed on every `$digest`. So, if your running into performance issues, you may need to implement a custom directive that writes out your data without setting up those superfluous watches, so you're a little more performant.