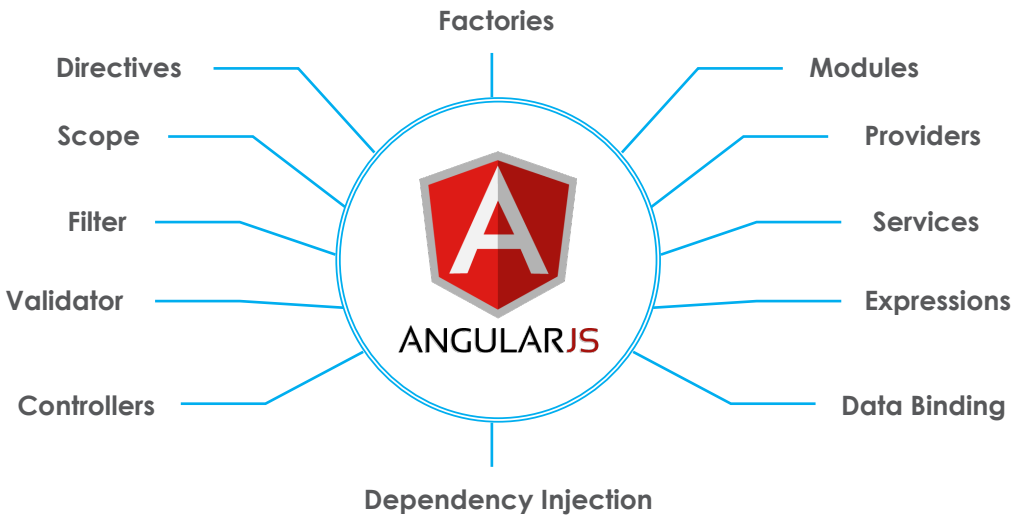
TO THE NEW DIGITAL

HOW TO OPTIMIZE YOUR ANGULARJS APPLICATIONS?

By Rajat Sharma

CONTENT

1. INTRODUCTION



AngularJS is an opinionated Model View Framework developed by Google, which has quickly become one of the most popular JS frameworks today. Unlike Backbone.js that gives you a certain degree of freedom in the way you write your code, Angular has an "Angular" way of doing things the right way.

WHY ANGULAR? -- what's it all about?

- Angular reduces the amount of code you write, thus making development faster and less error-prone .
- It has an extensive community support with a lot of Angular plug-ins available on Github.
- MVC done right!
- Two-way data binding
- Directives (Extending the capabilities of HTML)
- Data Models are plain old Javascript objects, which need any getter and setters unlike Backbone js
- Unit Testing is easier because of the modularization of code that Angular Encourages

At first look, starting with Angular is really easy, with its magical superpowers (Yes, I am talking about 2-way data binding), but building large complex scalable web apps, requires understanding the inner mechanics of Angular. It has a steep learning curve but it is worth the trouble. If one fails to understand these inner workings, the app will get much slower over the course of development. It is known that Angular breaks when the number of bindings near the 2000 mark. This whitepaper aims to address those pitfalls and problems, and share insights on how to overcome them.

2. UNDERSTANDING THE DIGEST LOOP

Two-way data-binding enables AngularJS to update the view if the model is updated, and vice-versa. How does Angular do that? This is probably the most misunderstood concept in AngularJS.

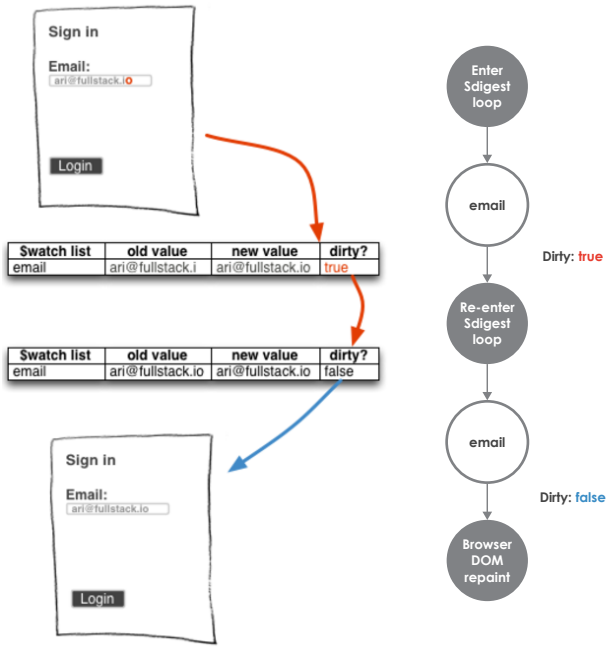
When we write {{abc}} (Angular expression) in an HTML template, Angular implicitly sets up a watcher on the scope model, which in turn updates the view whenever the scope model changes.

This watcher is just like any watcher you set up in AngularJS:

```
$scope.$watch('abc', function(newValue, oldValue) {

  //update the View with newValue

});
```



The Angular app keeps a watch list which stores the current values of all the Angular expressions in the watch functions (implicit or explicit).

Once the digest loop begins, Angular iterates over all the watches in the watch list.

The updated values are matched against the old values, if they don't match, the watcher's callback is executed and the new value is recorded. Once all the items/expression in the watch list have been iterated, the Digest loop starts again until no more changes are detected.

Why run the Digest loop all over again?

Consider the following scenario:
We have two models: A and B. When B changes A would also change.

- Model B changes, Assume there is a watcher callback which in turn causes model A to change.
- Digest loop begins
- Model A is evaluated in the watcher list (no changes detected yet, since the B's watcher hasn't fired yet)
- Model B is evaluated, updated value doesn't match the old value, so this new value is recorded, and the B's watcher callback is fired, which causes Model A's value to change

Now, if the digest loop is not run all over again, Angular wouldn't know how to communicate A's changes throughout the app since the watcher callbacks associated with model A never fired, creating a synchronization problem in the app.

To avoid an infinite digest loop, the digest loop is executed 10 times (infinite loop threshold) at the maximum if not resolved early, before throwing an infinite digest loop error. It is therefore advisable to avoid changing models in the watcher callbacks, as it increases the digest loops. Note that Angular does provide provisions to change the 'infinite loop threshold', but it shouldn't be tampered with.

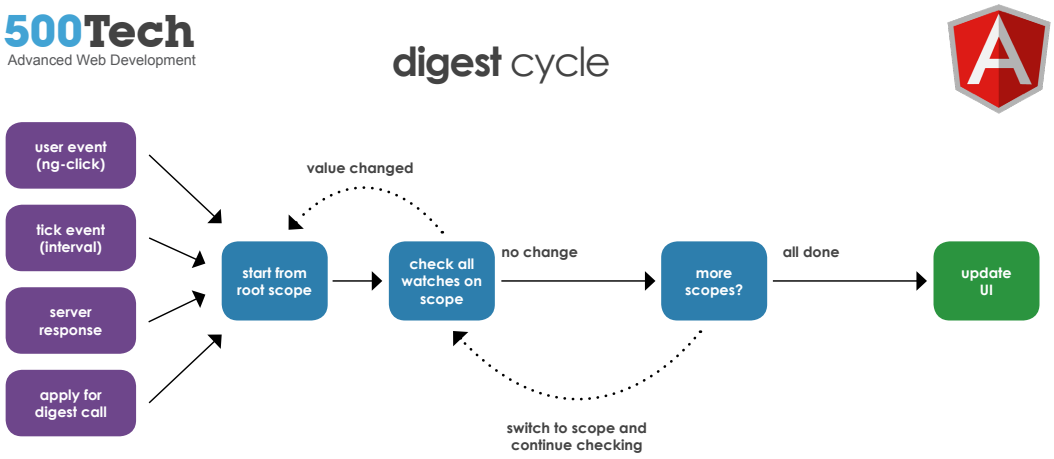
3. WHEN IS THE DIGEST LOOP TRIGGERED?

Angular doesn't work on a polling mechanism which checks the model every few milliseconds and updates the view accordingly. This approach is highly impractical and slow.

Instead Angular recognizes the cases when an App can change:

- **User Interactions through events:** The user clicks on the UI controls like button, which causes the application
- **Ajax:** Request to server is made, response in return may be used to alter the models.
- **Timeouts:** Async timeout operations can possibly change the state of the application.

Angular intercepts the above interactions for us and trigger the digest cycle automatically.



angular source code. line ***13991***

When do we need to explicitly execute the digest loop?

When something happens beyond the Angular realm/context, Angular needs to be notified of the same. It can be done by explicitly calling the $apply()

Certain cases in which Angular manual push is required a digest loop:-

- When we use Javascript's setTimeout() [Please prefer Angular's $timeout for this]
- DOM handler is attached inside a directive.

4. HOW TO OPTIMIZE YOUR ANGULARJS APP?

Enable $httpProvider's useApplyAsync

Consider a scenario in which 5 ajax calls are made through the Angular's $http service during application's bootstrap. As described earlier, each of these requests will inturn cause the digest loop to execute, meaning that Digest loop is executed atleast 5 times. This is costly.

This can be fixed by the following code snippet:-

```
app.config(function ($httpProvider) {
  $httpProvider.useApplyAsync(true); // default value is false.
});
```

This allows Angular to delay the resolution of the Ajax call to the the next tick.

Here is what happens, when an Angular app receives multiple $http responses at around the same time:-

- The call's promise is pushed into a queue.
- An async apply is scheduled, in case none is scheduled yet, by telling the browser to execute a timeout with 0 delay.
- Once timeout, the queue is flushed and the actual $apply is triggered.

It takes about 10 milliseconds for '0-delay-timeout' to execute, if the responses from the 5 $http requests are received within this 10 millisecond window, they are resolved in 1 digest cycle only, instead of 5.

It can result in significant performance improvement for large complex apps, especially during application bootstrap.

Disabling Debug Info

By default, Angular attaches certain CSS classes and information about scope & bindings to the DOM nodes that assist in debugging an Angular app, and are used by tools like Protractor and Batarang. This debugging info should be disabled in the production using the following code snippet:-

```
app.config(function ($compileProvider) {
  $compileProvider.debugInfoEnabled(false)// default value is true.
});
```

Enabling the Strict DI (Dependency Injection) Mode

Strict DI mode was built to help us ensure, that the minified Angular code worked properly. When this mode is enabled, the app will throw errors if the dependencies are not annotated explicitly.

So how does turning this mode on improve performance?

Enabling this mode will improve Angular's performance while injecting the dependencies into injectable functions as it won't need to dynamically discover the function's dependencies.

This mode can be enabled in the following two ways:-

```
<div ng-app="myApp" ng-strict-di>
  <!-- your app here -->
</div>
```

or

```
Angular.bootstrap(document, ['myApp'], {
  strictDi: true
});
```

Caching DOM elements

DOM access and manipulation is the most expensive activity that one can do with Javascript. It needs to be done wisely and artistically.

If one needs to do a lot of work/manipulation with the DOM element, the DOM element should be cached in a variable, rather than querying the DOM again and again for the same element.

Consider the following code snippet:-

```
// bad
document.getElementById('name').style.bottom = "33px";
document.getElementById('name').style.right = "33px";

document.getElementById('name').style.color = "green";


// better
var nameElmStyle = document.getElementById('name').style;
nameElmStyle = "33px";
nameElmStyle = "33px";
nameElmStyle = "green";
```

Optimizing ng Click for mobile devices

When a user taps/clicks in a mobile browser, most of the mobile browsers wait for about 300 milliseconds after the "tap and release" before sending the click event. This can be prevented by including ngTouch module in the Angular app. The ngTouch module comes with its own ngClick directive which overrides default ngClick. However, please note that this functionality has been deprecated ever since Angular 1.5. If you are using Angular >1.5, you may want to check out ng-fastclick by 8bitdesginer on Github.

Use of One-time bindings

This type of bindings are unregistered/removed from the watchlist, once the binding value stabilizes (not undefined) after the first digest cycle. Reducing the number of expressions watched makes the digest loop faster. One-time bindings should be used whenever possible.

Example of one-time binding:-

```
<div>Information: {{::address | uppercase}}</div>
```

ng-if vs ng-show

In large and complex applications, the number of DOM elements in a page/view can increase drastically. Heavy DOM slows down the app. ng-show/ng-hide simply shows or hides the DOM element. Even if the element is hidden, it is still present in the DOM. Problem arises when there are nested ng-repeats present inside the ng-show parent, because this increases the DOM elements quickly (even when there is no need for them to be present). If the ng-if expression evaluates to false, no element(s) are added to DOM, keeping the DOM lightweight.

NOTE:- ng-if creates a new scope of its own, prototypically inheriting from the parent scope.

Using ng-model-options

Consider a scenario in which a user is typing something in a search box. This search box has an ng-model "search" associated with it. On each keystroke, the model "search" gets updated, causing a digest cycle to execute on every keystroke. This is very expensive. Would'nt it be great if we can control when exactly does the digest cycle get triggered? This is what "ng-model-options" brings to the table. With this, we can control after how much time or on completion of which events the model should get updated.

```
<input
  type="search"
  ng-model="search"
  ng-model-options="{ debounce: 500 }">
```

The above code snippet, causes the model to change 500 milliseconds after the user has stopped typing.

Using "track-by" for ng-repeat and ng-options

```
<li ng-repeat="model in collection">
  {{model.name}}
</li>
```

Consider, the above code snippet. If "collection" is updated with the data from the server, by doing something like:-

```
$scope.collection = serverResponse.collection;
```

Even if most of the items in the collection remain exactly the same, all the "li" associated with the ng-repeat will be destroyed and recreated. This is very expensive operation.

"track by" solves this problem. Using track-by we can tell which object property should be used to associate the Javascript object with the DOM node.

```
<li ng-repeat="model in collection track by model.id">
 {{model.name}}
</li>
```

Now even if the collection is updated from the server, since the Javascript object is associated with the DOM node with the model's id, the DOM node simply updates the DOM instead of recreating it.

5. CONCLUSION

A conscious effort should be made to reduce the number of watchers in the Angular app, because it optimizes the Angular digest cycle which is the engine driving an Angular app. Angular is a powerful weapon for the front-end warriors who can conquer the world if used wisely.

Angular 2.0 is just around the corner and it's going to change the Angular world for better. It brings in new features like lazy loading, server side rendering, better dependency injection, better change detection mechanism etc.

Angular 2.0 encourages the use of TypeScript (developed by Microsoft). TypeScript is an extension of ECMA script, which compiles into Javascript. Angular 2.0 is much better in performance when compared to Angular 1.x. One may have to unlearn Angular 1.x to migrate to Angular 2.0, but it needs to be done as Angular 2.0 is the future.

Talk to our experts

6. ABOUT THE AUTHOR

Rajat Sharma
Senior Software Engineer

Rajat has been working on front-end technologies for the past 3+ years and is passionate about Javascript. He has extensive experience in Backbone, AngularJS for single page applications, handlebar as a templating engine for backbone, Grunt for project automation, require for implementing AMD modules and pre-loading javascript files, D3 for charting visualizations, His skill sets include expertise in Javascript , HTML5, CSS3, Angular, Backbone, Grunt, jQuery, Underscore, Handlebars, jsPumb, D3, requireJs, Ajax, Model View Architecture.

7. ABOUT TO THE NEW DIGITAL

TO THE NEW Digital is is a premium digital services company that uniquely combines the power of technology, analytics, marketing and content for digital transformation. TO THE NEW digital has developed more than 200 mission critical web and mobile applications, delivered over 2 billion video views, executed over 1000 digital campaigns, helped over 50 global clients with our social media analytics services and have done more than 100 cloud implementation.

8. Explore Other Resources

Best Practices of Web User Interface Design

Download Ebook

Is It The Right Time to Change Your User Interface?

Download Ebook

TO THE NEW DIGITAL

info@tothenew.com
www.tothenew.com

Lets Connect