Enhancing $log in AngularJs the simple way

by Benny Bottema - Monday, December 23, 2013

http://www.bennybottema.com/2013/12/23/enhance-logging-in-angularjs-the-simple-way/

16-05-2015: The code of this post was turned into a GitHub project called angular-logger

Contents

Let's start by prepending timestamp and class name

Recently I've been on the lookout for a way to configure Angular's logging module. Specifically, I wanted its output prepended with a date and a context string, say a 'class' name or name of the controller doing the logging.

Something like this (example from java):
"2013-12-23 19:44:39,619 INFO [java.sql.DatabaseMetaData]: Table not found: Employees"

Also, if we enable per-class loggers, how can we mute logging for certain classes?

What I found was a rather… *extensive* a solution utilizing Angular's built in support for decorators. Ofcourse, the fact it actually works is great and it's a nice insight into some advanced javascript and Angular concepts (I especially found the requirejs / angular combo of interest), but I think it can be much *much* simpler.

Here's how I do it:

```
app.run(['$log', function($log) {
   $log.getInstance = function(context) {
      return {
         log   : enhanceLogging($log.log, context),
         info  : enhanceLogging($log.info, context),
         warn  : enhanceLogging($log.warn, context),
         debug : enhanceLogging($log.debug, context),
         error : enhanceLogging($log.error, context)
      };
   };

   function enhanceLogging(loggingFunc, context) {
      return function() {
         var modifiedArguments = [].slice.call(arguments);
         modifiedArguments[0] = [moment().format("dddd h:mm:ss a") + '
::[' + context + ']> '] + modifiedArguments[0];
         loggingFunc.apply(null, modifiedArguments);
      };
   }
}]);
```

Usage:

```
var logger = $log.getInstance('Awesome');
logger.info("This is awesome!");
```

Monday 9:37:18 pm::[Awesome]> This is awesome!

Oh dear, that's it? Actually, it is. I used the terrific Moment.js for datetime formatting, which is like Joda-Time for javascript. The code uses Angular's module run block support to configure the application before anything else starts running.

Now let's add filtering based on context

Ok, so now we need to enable per-class logging suppression, so that we can reduce the logging noise from all the log.debug() statements. Ofcourse Angular has a global switch, which is very crude:

```
// Angular's own crude way of muting log statements
app.config(['$logProvider', function($logProvider) {
   $logProvider.debugEnabled(false); // default is true
}]);
```

We want to be able to do this per class, or rather per context since we can use an arbitrary string as a prepended context. Let's add a logging filter based on context:

First let's move the $log enhancement code into its own function for better reuse accross projects:

```
app.run(['$log', enhanceAngularLog]);

// in log-enhancer.js:
function enhanceAngularLog($log) {
   $log.getInstance = function(context) {
      (..)
   };

   (..)

}
```

Everything is the same, except the code now resides in a function we can easily call. Let's add context-based filtering to it:

```
function enhanceAngularLog($log) {
   $log.enabledContexts = [];
   $log.getInstance = function(context) {
      return {
         log   : enhanceLogging($log.log, context),
         info  : enhanceLogging($log.info, context),
         warn  : enhanceLogging($log.warn, context),
         debug : enhanceLogging($log.debug, context),
         error : enhanceLogging($log.error, context),
         enableLogging: function(enable) {
            $log.enabledContexts[context] = enable;
         }
      };
   };

   function enhanceLogging(loggingFunc, context) {
      return function() {
         var contextEnabled = $log.enabledContexts[context];
         if ($log.enabledContexts[context] == null || contextEnabled)
{
            var modifiedArguments = [].slice.call(arguments);
            modifiedArguments[0] = [moment().format("dddd h:mm:ss a")
+ '::[' + context + ']> '] + modifiedArguments[0];
            loggingFunc.apply(null, modifiedArguments);
         }
      };
   }
}
```

Usage:

```
var notMutedLogger = $log.getInstance('Not Muted');
var mutedLogger = $log.getInstance('Muted');

mutedLogger.enableLogging(false);

notMutedLogger.info("This *will* appear in your console");
mutedLogger.info("This will *not* appear in your console");

// output: Monday 9:37:18 pm::[Not Muted]> This is *will* appear in yo
ur console
```

A little bit more advanced: log enhancer as a configurable provider

A more elegant solution is if we better integrate the log enhancer with Angularjs and make it configurable per app. We can do that by define the log enhancer as a provider:

```
angular.module('app').provider('logEnhancer', function() {
 this.loggingPattern = '%s - %s: ';

 this.$get = function() {
  var loggingPattern = this.loggingPattern;
  return {
   enhanceAngularLog : function($log) {
    $log.enabledContexts = [];

    $log.getInstance = function(context) {
     return {
      log : enhanceLogging($log.log, context, loggingPattern),
      info : enhanceLogging($log.info, context, loggingPattern),
      warn : enhanceLogging($log.warn, context, loggingPattern),
      debug : enhanceLogging($log.debug, context, loggingPattern),
      error : enhanceLogging($log.error, context, loggingPattern),
      enableLogging : function(enable) {
       $log.enabledContexts[context] = enable;
      }
     };
    };

    function enhanceLogging(loggingFunc, context, loggingPattern) {
     return function() {
      var contextEnabled = $log.enabledContexts[context];
      if (contextEnabled === undefined || contextEnabled) {
       var modifiedArguments = [].slice.call(arguments);
       modifiedArguments[0] = [ sprintf(loggingPattern, moment().forma
t("dddd h:mm:ss a"), context) ] + modifiedArguments[0];
       loggingFunc.apply(null, modifiedArguments);
      }
     };
    }
   }
  };
 };
});
```

And then to configure and enhance the log:

```
var app = angular.module('app', []);

app.config(['logEnhancerProvider', function(logEnhancerProvider) {
    logEnhancerProvider.loggingPattern = '%s::[%s]> ';
}]);

app.run(['$log', 'logEnhancer', function($log, logEnhancer) {
    logEnhancer.enhanceAngularLog($log);
}]);
```

Final thoughts

So there you have it. We enhanced Angular’s $log to prepend a date / timestamp and context, and we modified it so that we can turn off logging for a specific context. Finally we made it reusable and configurable as a provider.

- Working jsFiddle example

What’s next?

What else can we do? Well, maybe we can turn can further refine the filter so that we can turn off logging for a context for a specific log-level, say only turn off debug statements. but I’ll leave that as an exercise to the reader, for now…