Gary Chambers Follow
Software Engineer at Football Radar
May 18, 2014 · 3 min read

Better logging in Node.js

Add context to your log messages

Logging in Node.js via ***console.log*** is perfectly adequate for development and debugging, but what should you do if you want more robust logging in production? There are some very good options in the NPM registry: **winston** *(Github: flatiron/winston, MIT)* is a particularly expressive logging library, a strong addition to your Node.js toolbox. In many cases, it might make sense to use something like this in production; what follows is more of a thought exercise. Let's assume for a moment that you want to roll your own logging—how might you go about that?

Enhancing the console

There are a variety of different console methods that build on top of ***process.stdout*** and ***process.stderr***, writable streams available to any Node.js application. While these methods make it easy to write messages to an output stream, they lack the expressiveness to provide *context* about those messages. By augmenting the builtin console, however, we can add some very desirable features.

This code, for example, will prepend any log message with an ISO-8601 formatted date string. When logging in production, this is—without fail—the first thing I look for in an insightful log message.

```
["log", "warn", "error"].forEach(function(method) {
    var oldMethod = console[method].bind(console);
    console[method] = function() {
        oldMethod.apply(
            console,
            [new Date().toISOString()].concat(arguments)
        );
    };
});


console.log("LOG ALL THE THINGS!!1!");
// "2014-05-18T09:00.000Z LOG ALL THE THINGS!!1!"
```

Of course, you may want a more terse, readable date string than this, but the concept is the same. ***moment.js*** (Github: moment/moment, MIT)*, for example, provides an easy-to-use date formatting vocabulary.*

The next thing I would normally like to know is whether the message is an error, a notification, a warning, or something else entirely. Now, I could just do the same as above, prefixing the log message with “info”, “warn”, “error” and so on, but I prefer something a bit more visually accessible: console colours.

My favoured console colour library is **cli-color** *(Github: medikoo/cli-color, MIT)*. This library exposes a simple API for wrapping text strings in one of 255 xterm colours. For convenience, it also defines named colours. So, how can take advantage of this for more expressive logging?

First, we define a mapping between console methods and appropriate colours:

```
var clc = require("cli-color");
var mapping = {
    log: clc.blue,
    warn: clc.yellow,
    error: clc.red
};
```

Then, as above, we override the builtin console methods. In this example, I am only colourising the date string:

```
["log", "warn", "error"].forEach(function(method) {
    var oldMethod = console[method].bind(console);
    console[method] = function() {
        oldMethod.apply(
            console,
            [mapping[method](new Date().toISOString())]
                .concat(arguments)
        );
    };
});
```

Obviously, this is quite a trivial example, but there are far more interesting possibilities: error messages could be colour-coded based on their severity; in an access log, you may wish to distinguish between different status codes or different IP ranges.

How much you choose to extend your log messages depends largely on your requirements: an HTTP server might be concerned with access logs, while an embedded system or background service might care more about runtime errors, bad input data or the availability of other services.

Exploring all these different use cases is beyond the scope of this post, but in my experience, extending log messages in some of these ways can make the console a far more powerful, expressive tool.

. . .

Gary Chambers is a Software Engineer at Football Radar in London, specialising in JavaScript development. Read more about the work done at Football Radar.