making your JavaScript debuggable

Patrick Mueller **@pmuellr**, **muellerware.org**
senior node engineer at NodeSource

http://pmuellr.github.io/slides/2015/11-debuggable-javascript
http://pmuellr.github.io/slides/2015/11-debuggable-javascript.pdf
http://pmuellr.github.io/slides/ (all of Patrick's slides)

code reading

> I'm doing 90% maintenance and 10% development, is this normal?

Stack Overflow

> ... more than 50% of the global software population is engaged in modifying existing applications rather than writing new applications." -

Capers Jones

> In 1949 as soon as we started programming, we found to our surprise that it wasn't as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realised that a large part of my life from then on was going to be spent in finding mistakes in my own programs.

Maurice Wilkes

you will write a little code

you will read a lot of code

pyramid of doom

```
fs.readdir(".", function(err, files){
  files.forEach(function(file) {
    fs.stat(file, function(err, stats){
      if (!stats.isFile()) return
      fs.readFile(file, "utf8", function(err, data){
        console.log(file, data.length)
      })
    })
  })
})
```

Die! Die! Die!

pyramid of doom fixed - I

```
fs.readdir(".", cbReadDir)
function cbReadDir(err, files) {
  files.forEach(eachFile)
}
function eachFile(file) {
  fs.stat(file, (err, stats) => cbStatFile(err, stats, file))
}
function cbStatFile(err, stats, file) {
  if (!stats.isFile()) return
  fs.readFile(file, "utf8", (err, data) => cbReadFile(err, data, file))
}
function cbReadFile(err, data, file) {
  console.log(file, data.length)
}
```

pyramid of doom fixed - 2

```
fs.readdir(".", cbReadDir)
function cbReadDir(err, files) {
  files.forEach(eachFile)
}
function eachFile(file) {
  fs.stat(file, cbStatFile)
  function cbStatFile(err, stats) {
    if (!stats.isFile()) return
    fs.readFile(file, "utf8", cbReadFile)
  }
  function cbReadFile(err, data) {
    console.log(file, data.length)
  }
}
```

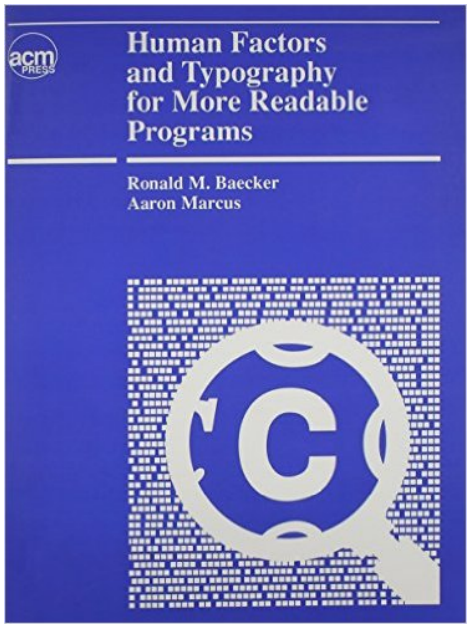
pyramid of doom - see also

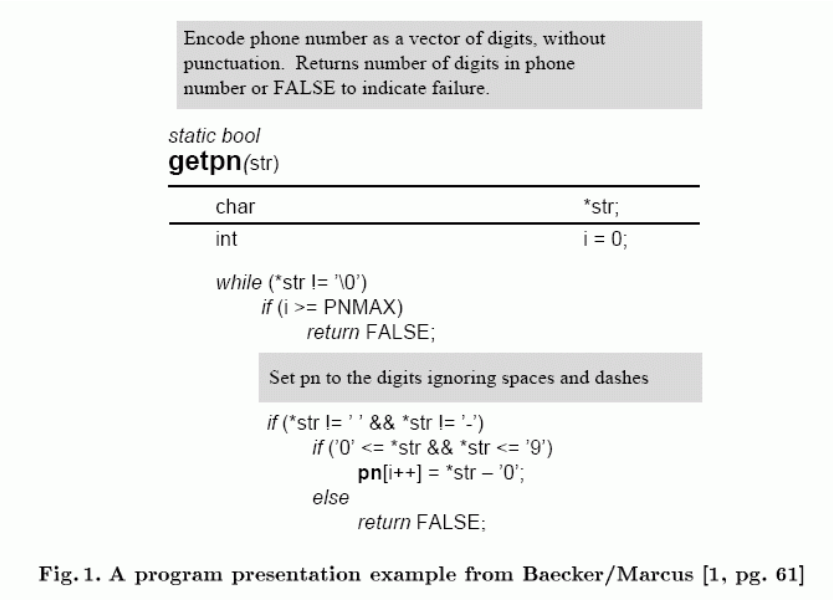
- async - npm - Caolan McMahon
- Promises - Axel Rauschmayer

Human Factors and Typography for More Readable Programs

- 1990
- Ronald M. Baecker, Aaron Marcus

ISBN 0201107457





Encode phone number as a vector of digits, without punctuation. Returns number of digits in phone number or FALSE to indicate failure.

```
static bool
getpn(str)

    char                    *str;
    int                     i = 0;

    while (*str != '\0')
        if (i >= PNMAX)
            return FALSE;

        Set pn to the digits ignoring spaces and dashes

        if (*str != ' ' && *str != '-')
            if ('0' <= *str && *str <= '9')
                pn[i++] = *str – '0';
            else
                return FALSE;
```

Fig. 1. A program presentation example from Baecker/Marcus [1, pg. 61]

logging

> The most effective debugging tool is still careful thought, coupled with judiciously placed print statements."

Brian W. Kernighan

console.log()

```
console.log("foo")
// prints: foo

console.log("foo", "bar")
// prints: foo bar

console.log({x:1, y:2})
// prints: { x: 1, y: 2 }

console.log("a-%s-b %j", 1, {x:1})
// prints: a-1-b {"x":1}

console.log(process)
// prints: { title: 'node', ...many lines... }
```

console.time()

```
console.time("foo")
doStuff()
console.timeEnd("foo")

function doStuff() {
    // takes a long time
}

// prints: foo: 1121ms
```

console.trace()

```
function a() { b() }
function b() { c() }
function c() { console.trace("foo") }

a()

// Trace: foo
//     at c (<program>:3:24)
//     at b (<program>:2:16)
//     at a (<program>:1:78)
//     at ...
```

npm debug

```
const debugA = require("debug")("thing-A")
const debugB = require("debug")("thing-B")

function a() { debugA("thrashing") }
function b() { debugB("churning") }

setInterval(a, 500); setInterval(b, 333)
```

```
$ DEBUG=* node debug.js
thing-B churning +0ms
thing-A thrashing +0ms
thing-B churning +339ms
thing-A thrashing +501ms
...
```

npm winston

```
var winston = require("winston")

winston.remove(winston.transports.Console)
winston.add(winston.transports.Console, { level:"warn" })
winston.add(winston.transports.File, { filename: "x.log" })

winston.info("info message")
winston.warn("warning message")
winston.error("error message")

// prints:
// warn: warning message
// error: error message
```

npm bunyan

```
const bunyan = require("bunyan")

const log = bunyan.createLogger({name: "myapp"})
log.level("info")

log.info("hi")

// prints
// {"name":"myapp","hostname":"my-hostname","pid":49675,
//  "level":30,"msg":"hi","time":"2015-10-27T03:49:14.759Z","v":0}

// du -h bunyan - 2.5M
```

npm bole

```
const bole = require("bole")

const log = bole("myapp")
bole.output({ level: "info", stream: process.stdout })

log.info("hi")

// prints
// {"time":"2015-10-27T03:56:45.762Z","hostname":"my-hostname",
//  "pid":53014,"level":"info","name":"myapp","message":"hi"}

// du -h bole - 144K
```

error handling

builtin process events

```
process.on("exit", code => console.log("exiting with code: " + code))
process.on("uncaughtException", err =>
    console.log("uncaught exception: " + err.stack
)
function a() { throw new Error("die die die") }

a()

// prints:
//
// uncaught exception: Error: die die die
//    at a (/path/to/process-events.js:9:22)
//    at Object.<anonymous> (/path/to/process-events.js:11:1)
//    ... more stack trace lines
// exiting with code: 0
```

`Error.prepareStackTrace()` - before

```
try { a() } catch(err) { console.log(err.stack) }
function a() { b() }
function b() { c() }
function c() { throw new Error("foo blatz") }

// Error: foo blatz
//     at c (/path/to/snippets/v8_prepareStackTrace-before.js:5:22)
//     at b (/path/to/snippets/v8_prepareStackTrace-before.js:4:16)
//     at a (/path/to/snippets/v8_prepareStackTrace-before.js:3:16)
//     at Object.<anonymous> (/path/to/snippets/v8_prepareStackTrace-before.
//     at Module._compile (module.js:456:26)
//     at Object.Module._extensions..js (module.js:474:10)
//     ...
```

Error.prepareStackTrace() - after

```
Error.prepareStackTrace = function(err, stackTrace) { ... }
try { a() } catch(err) { console.log(err.stack) }
function a() { b() }
function b() { c() }
function c() { throw new Error("foo blatz") }

// Error: foo blatz
//     v8_prepareStackTrace-after.js  13 - c()
//     v8_prepareStackTrace-after.js  12 - b()
//     v8_prepareStackTrace-after.js  11 - a()
```

`Error.prepareStackTrace()`

reference: javascript_stack_trace_api.md

at:

- https://chromium.googlesource.com/v8/v8/+/master/docs

npm `Q.longStackSupport` - before

```
var Q = require("q")


function a() { Q.delay(100).done(b) }
function b() { throw new Error("foo") }

a()

// Error: foo
//     at b (/path/to/snippets/q-longStack-before.js:5:22)
//     at _fulfilled (/path/to/snippets/node_modules/q/q.js:787:54)
//     at self.promiseDispatch.done (/path/to/snippets/node_modules/q/q.js:8
//     at Promise.promise.promiseDispatch (/path/to/snippets/node_modules/q/
//     at /path/to
```

npm `Q.longStackSupport` - after

```
var Q = require("q")
Q.longStackSupport = true

function a() { Q.delay(100).done(b) }
function b() { throw new Error("foo") }

a()

// Error: foo
//     at b (/path/to/snippets/q-longStack-after.js:5:22)
// From previous event:
//     at a (/path/to/snippets/q-longStack-after.js:4:29)
//     at Object.<anonymous> (/path/to/snippets/q-longStack-after.js:7:1)
```

early warning systems

If debugging is the process of removing bugs, then programming must be the process of putting them in.

-- Edsger W. Dijkstra

testing

```
describe("Array", function(){
  describe("#indexOf()", function(){
    it("should return -1 when the value is not present", function(){
      assert.equal(-1, [1,2,3].indexOf(1));
    })
  })
})

//  0 passing (3ms)
//  1 failing
//
//  1) Array #indexOf() should return -1 when the value is not present:
//     AssertionError: -1 == 0
//      at Context.<anonymous> (path/to/testing.js:7:14)
//      ...
```

testing

- mocha - http://visionmedia.github.io/mocha/
- jasmine - http://jasmine.github.io

linting

```
> jshint snippets/*.js
snippets/alert.js: line 1, col 17, Missing semicolon.

snippets/console_log.js: line 1, col 19, Missing semicolon.
snippets/console_log.js: line 4, col 26, Missing semicolon.
snippets/console_log.js: line 7, col 25, Missing semicolon.
snippets/console_log.js: line 10, col 35, Missing semicolon.
snippets/console_log.js: line 13, col 21, Missing semicolon.

snippets/console_time.js: line 1, col 20, Missing semicolon.
snippets/console_time.js: line 2, col 10, Missing semicolon.
snippets/console_time.js: line 3, col 23, Missing semicolon.
snippets/console_time.js: line 8, col 21, Missing semicolon.

... repeats ad nauseum ...
```

linting

- jshint - http://jshint.com/
- jslint - http://jslint.com/

etc

builtin module `repl`

```
var repl = require("repl")

function a(i) {
    var context = repl.start("repl> ").context
    context.pi  = 3.14
    context.arg = i
}

a(3)

// repl> pi
// 3.14
// repl> arg
// 3
// repl>
```

builtin debugger

```
function a() {
    debugger
    var x = 1
    var y = 2
    console.log(x + " + " + y + " = " + (x+y))
}

setTimeout(a, 1000)
```

builtin debugger

```
> node debug debugger.js
< debugger listening on port 5858
connecting... ok
...
debug> watch("x")
debug> cont
break in debugger.js:2
Watchers:
  0: x = undefined

  1 function a() {
  2     debugger
  3     var x = 1
  4     var y = 2
debug> next
```

```
...
debug> next
break in debugger.js:4
Watchers:
  0: x = 1

  2     debugger
  3     var x = 1
  4     var y = 2
  5     console.log(x + " + " ...
  6 }
debug> cont
< 1 + 2 = 3
program terminated
debug>
```

npm hooker

```
const hooker = require("hooker")

function log(prefix, name, value) {
    console.log("%s Math.%s: %j", prefix, name, value)
}

hooker.hook(Math, Object.getOwnPropertyNames(Math), {
    passName: true,
    pre: function (name) {
        log("->", name, [].slice.call(arguments,1))
    },
    post: function (result, name) {
        log("<-", name, result)
    }
})

Math.max(5, 6, 7)
Math.sqrt(2)
```

npm hooker

prints:

```
-> Math.max: [5,6,7]
<- Math.max: 7
-> Math.sqrt: [2]
<- Math.sqrt: 1.4142135623730951
```

also provides

- filtering arguments
- overriding results
- https://github.com/cowboy/javascript-hooker

heap snapshots

cpu profiles

fin