Node.js

Day 2

Alexandre Perrin

July 11, 2017

nomades.ch

Today

- Node.js modules
- callback and error handling
- async
- Testing
  - mocha
  - chai
  - supertest
  - Cucumber.js
  - Zombie.js
- Let’s code!

Node.js modules

require()

Modules installed from npm are simply files and directories saved into the `node_modules/` directory. `require` will look for them:

```
const mod = require("mod"); // node_modules/mod.js
                            // node_modules/mod/index.js
```

You can use the same logic inside of your project:

```
const mod = require("./mod"); // ./mod.js
                              // ./mod/index.js
```

module.exports

From a module, visibility is simply controlled by setting `module.exports`:

```
/*
 * mod.js
 */
"use strict";

// i is local to this module, invisible from the "outside".
let i = 0;

// nexti() is local to this module, invisible from the "outside".
function nexti() {
    return i++;
}

// next() is exposed to require().
module.exports = {
    next: function () {
        return 42 + nexti();
    }
};
```

callback and error handling

callback and error handling

Node.js uses closures (callback) extensively. Usually, the first argument given to a callback function is an `Error`, e.g.

```
fs.readFile("/etc/hosts", function callback(err, buf) {
    if (err)
        return console.error(err.message);
    // do something with buf
});
```

callback and error handling

Rather than `throwing`, errors and values are propagated through callback recursively:

```
1  "use strict";
2
3  const fs = require("fs");
4
5  // callback the word count of a given file path.
6  function wc(path, callback) {
7      fs.readFile(path, function (err, buf) {
8          if (err)
9              return callback(err);
10         const count = buf.toString().trim().split(/\s+/).length;
11         return callback(/* no error */null, count);
12     });
13 }
14
15 // node callback-err.js <path>
16 const path = process.argv[2];
17 wc(path, function (err, count) {
18     /* here we're given either (Error, undefined) or (null, Number) */
19     if (err)
20         console.error(err.message);
21     else
22         console.log(count);
23 });
```

callback hell

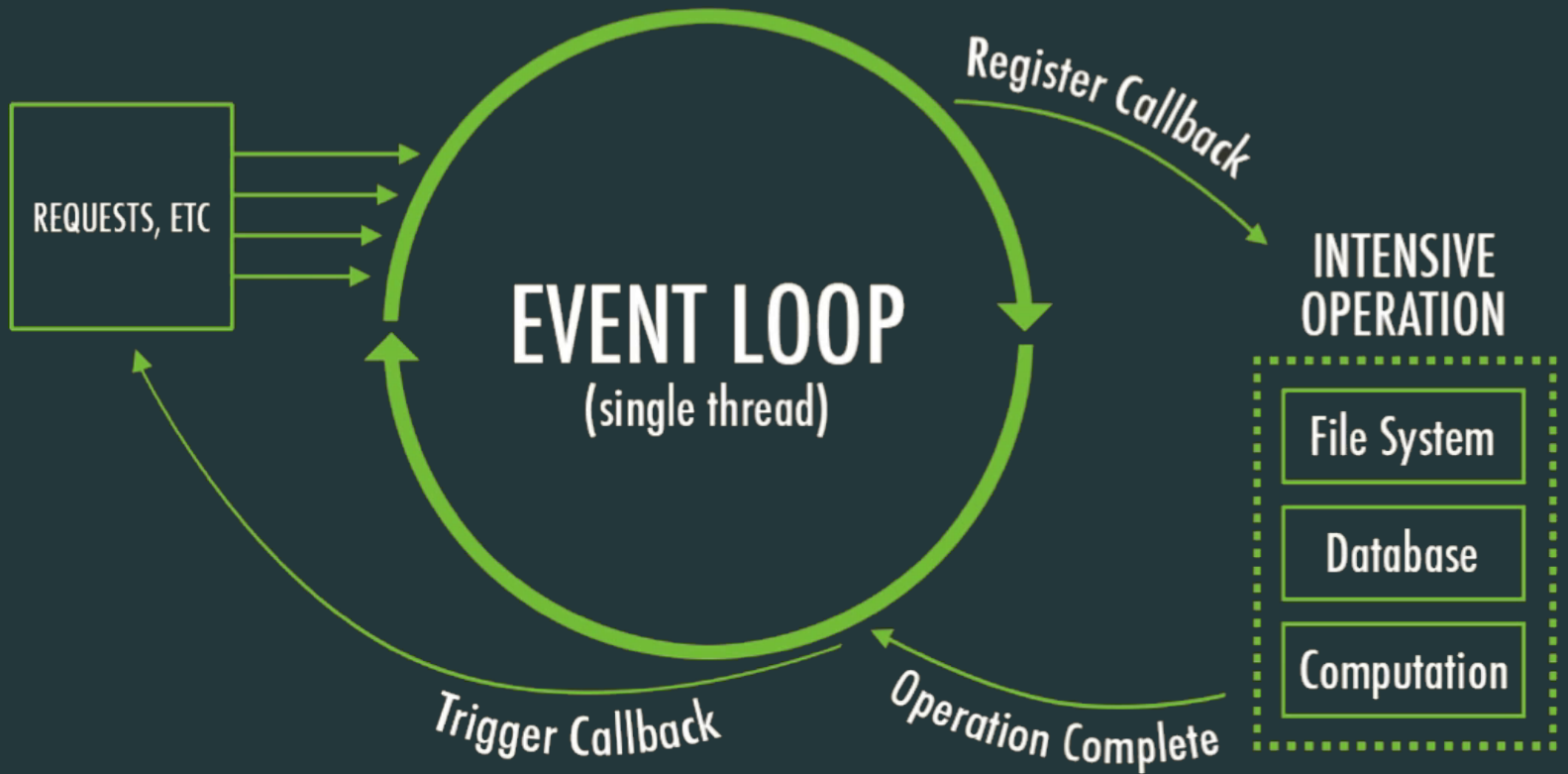
Arguably, ”callback hell” is caused by poor coding practices. callbackhell.com is worth reading on the subject.

async

async loop

Remember that I/O are executed in worker threads asynchronously. As a result, it is sometime tricky to understand in which order callback are executed.

```
1  "use strict";
2
3  const fs = require("fs");
4
5  ["/usr/share/dict/words", "/etc/hosts"].forEach(path => {
6      fs.readFile(path, (err, buf) => {
7          // which file will be printed first?
8          console.log(`${path} contains ${buf.length} bytes.`);
9      });
10 });
```


REQUESTS, ETC
EVENT LOOP
(single thread)
Register Callback
INTENSIVE OPERATION
File System
Database
Computation
Trigger Callback
Operation Complete

async loop

Solution using the async module:

```
1  "use strict";
2
3  const fs    = require("fs");
4  const async = require("async");
5
6  /* build an array of tasks (function) reading the files */
7  const tasks = ["/usr/share/dict/words", "/etc/hosts"].map(path => {
8      return function (done) {
9          fs.readFile(path, (err, buf) => {
10             if (err)
11                 return done(err);
12             const content = buf.toString();
13             return done(null, content);
14         });
15     };
16 });
17
18 /* print the results */
19 async.parallel(tasks, (err, results) => {
20     if (err)
21         console.error(err.message);
22     else
23         console.dir(results);
24 });
```

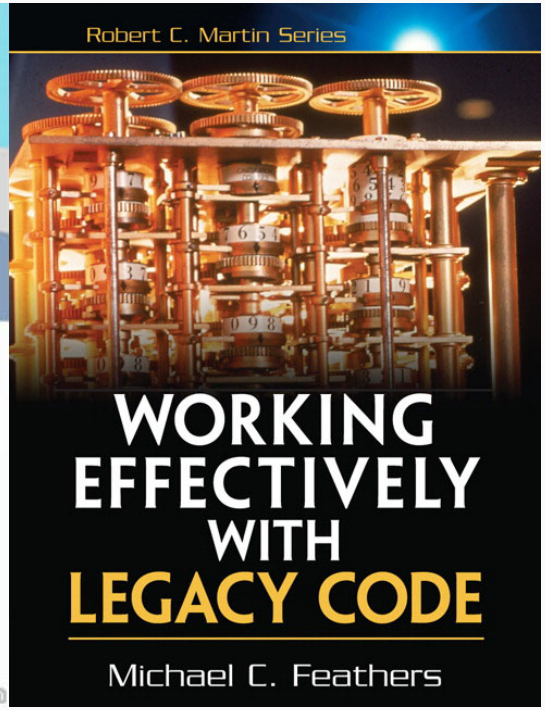
async loop

Solution using `Promise` (ES7 `async` / `await` in Node.js >= 7.6)

```
1  "use strict";
2
3  const fs = require("fs");
4  /* NOTE: no need to require Promise */
5
6  /* build an array of tasks (promises) reading the files */
7  const tasks = ["/usr/share/dict/words", "/etc/hosts"].map(path => {
8      return new Promise((resolve, reject) => {
9          fs.readFile(path, (err, buf) => {
10             if (err)
11                 return reject(err);
12             const content = buf.toString();
13             return resolve(content);
14         });
15     });
16 });
17
18 /* print the results */
19 Promise.all(tasks)
20     .catch(err => {
21         console.error(err.message);
22     }).then(results => {
23         console.dir(results);
24     });
```

Testing

yea what about it?



Michael Feathers describe two coding strategies:

1. Edit and Pray
2. Cover and Modify

The main thing that distinguishes legacy code from non-legacy code is a lack of comprehensive tests.

setting up

Installation

```
% npm install --save-dev mocha chai supertest
```

Setup the `test` script in `package.json`:

```
6   "scripts": {
7     "test": "mocha"
8   },
```

Create the `test` directory and add a blank file:

```
% mkdir test && touch test/index.js
```

You can now run the mocha tests by calling:

```
% npm test
```

testing Hello World

```
1  "use strict";
2
3  const chai    = require("chai");
4  const expect  = chai.expect;
5  const request = require("supertest");
6
7  require("../hello-express.js");
8
9  describe("Hello World app", () => {
10     it("should return 200", done => {
11         request("localhost:3000").get("/").end((err, res) => {
12             expect(res.status).to.eql(200);
13             return done();
14         });
15     });
16     it("should say Hello", done => {
17         request("localhost:3000").get("/").end((err, res) => {
18             expect(err).to.not.exist;
19             expect(res.text).to.eql("Hello World!");
20             return done();
21         });
22     });
23 });
```

mocha

mocha help to give your test a structure and some control flow.

Full documentation at mochajs.org.

```
// describe() group your tests.
describe("a feature", function () {
    // run once before all the tests
    before(function () { ... });
    // run once after all the tests
    after(function () { ... });
    // run before each tests
    beforeEach(function () { ... });
    // run after each tests
    afterEach(function () { ... });
    // context() is simply an alias of describe()
    context("when the sun is shining", function () {
        // it() is a single test.
        it("should work", function () { ... });
        // it() may not be yet implemented (pending)
        it("is not tested yet");
        // it() function can take a callback (most of the time).
        it("can be tested async", function (done) { ... });
        // it.skip() will not execute this test
        it.skip("may work or not", function () { ... });
        // .only() will make mocha run only the tests in the block
        it.only("may work or not", function () { ... });
    });
});
```

chai

chai is an ”assertion” library supporting several interfaces depending on your taste. Full documentation at chaijs.com.

```
1  expect(answer).to.be.a('number');
2  expect(password).to.be.a('string').and.to.equal('Open Sesame');
3  expect(clients).to.have.lengthOf(1000);
4  expect(tea).to.have.property('flavors').with.lengthOf(3);
```

supertest

supertest is a specialized assertion library for HTTP. It is very handy even to only make requests (e.g. POST with body). The documentation is at the GitHub project page.

```
1  const request = require('supertest');
2  const express = require('express');
3
4  const app = express();
5
6  app.get('/user', function(req, res) {
7      res.status(200).json({ name: 'tobi' });
8  });
9
10 request(app)
11 .get('/user')
12 .expect('Content-Type', /json/)
13 .expect('Content-Length', '15')
14 .expect(200)
15 .end(function(err, res) {
16     if (err) throw err;
17 });
```

Cucumber.js

Cucumber is a *Behaviour-Driven Development* testing tool. Gherkin, Cucumber's non-technical and human readable language, is used to define test cases. Documentation and examples at the GitHub project page.

```
# features/documentation.feature
Feature: Example feature
  As a user of Cucumber.js
  I want to have documentation on Cucumber
  So that I can concentrate on building awesome applications

  Scenario: Reading documentation
    Given I am on the Cucumber.js GitHub repository
    When I click on "CLI"
    Then I should see "Running specific features"
```

Zombie.js

Zombie is a very fast, headless full-stack testing using Node.js. It *simulate* a browser environment and is able to evaluate Javascript. More at zombie.js.org.

```
const Browser = require('zombie');
const browser = new Browser();
browser.visit("https://github.com/cucumber/cucumber-js/tree/master", () => {
    browser.clickLink("CLI").then(() => {
        browser.assert.success();
        browser.assert.text('body', /Running specific features/);
        browser.tabs.closeAll();
    });
});
```

Let’s code!

Yet Another (tested!) Blog Engine

1. Cover your blog server with tests, then
2. Refactor your code into functions and modules.

Questions?

Read on later

#NoTDD by Eric Gunnerson.