footer: © NodeProgram.com, Node.University and Azat Mardan 2018
slidenumbers: true
theme: Simple, 1
build-lists: true

[.slidenumbers: false]
[.hide-footer]

Node Advanced

Overview

Azat Mardan @azat_co

Node Advanced

- Videos: http://node.university/p/node-advanced (http://node.university/p/node-advanced)
- Slides: in `*.md` in https://github.com/azat-co/node-advanced (https://github.com/azat-co/node-advanced)
- Code: in code in https://github.com/azat-co/node-advanced (https://github.com/azat-co/node-advanced)

Table of Contents

--

Course Overview

- Table of Contents
- What to expect
- What you need

Module 1: Modules

- How `module.exports` and `require()` actually work

- npm tricks (scripts) and npm scripts

Module 2: Node Event Loop and Async Programming

- Event look
- Call stack
- setTimeout, process.nextTick
- Error-first callback
- Event Emitters
- Promises vs events
- Async/await

Module 3: Streaming

- reading
- writing
- duplex
- transform

Module 4: Debugging

- Debugging
- CPU profiling
- Networking Debugging with DevTools

Module 5: Scaling

- cluster
- Load testing
- Messaging
- spawn, fork, exec
- Offloading CPU-intensive tasks

Outro

- Summary

What to expect

- Pure Node
- Core Node
- ES Next

What not to expect

- No npm modules
- No JavaScript
- No frameworks

Prerequisites

- Node Foundation
- You Don't Know Node
- Node Patterns

What you need

- Node version 8+
- npm version 5+
- Chrome
- Slides&code: [https://github.com/azat-co/node-advanced (https://github.com/azat-co/node-advanced)](https://github.com/azat-co/node-advanced)

Module 2: Modules

- How `module.exports` and `require()` actually work
- npm tricks (scripts) and npm scripts

`require()`

1. Resolving
2. Loading
3. Wrapping
4. Evaluating
5. Caching

---

module-1.js:

```
console.log(module) // console.log(global.module)
```

```
Module {
  id: '.',
  exports: {},
  parent: null,
  filename: '/Users/azat/Documents/Code/node-advanced/code/module-1.js',
  loaded: false,
  children: [],
  paths:
   [ '/Users/azat/Documents/Code/node-advanced/code/node_modules',
     '/Users/azat/Documents/Code/node-advanced/node_modules',
     '/Users/azat/Documents/Code/node_modules',
     '/Users/azat/Documents/node_modules',
     '/Users/azat/node_modules',
     '/Users/node_modules',
     '/node_modules' ] }
```

- local takes precedence
- module can be a file or a folder with index.js (or any file specified in package.json main in that nested folder)
- loaded is true when this file is imported/required by another
- id is the path when this file is required by another
- parent and children will be populated accordingly

require.resolve() – check if the package exists/installed or not but does not execute

1. try name.js
2. try name.json
3. try name.node (compiled addon example)
4. try name folder

`require.extensions`

{ '.js': [Function], '.json': [Function], '.node': [Function] }

Exporting

All the same:

```
global.module.exports.parse = () => {}
module.exports.parse = () => {}
exports.parse = () => {}
```

```
exports.parse = ()=>{} // ok
exports = {parse: ()=>{} } // not ok
module.exports = {parse: ()=>{} } // ok again
```

Function wrapping keeps local vars local

`require('module').wrapper`

exports and require are specific to each module, not true global global, same with `__filename` and `__dirname`

console.log(arguments)

```
module.exports = {
  parse: () => {}
}
```

```
const Parser = {
  parse() {

  }
}
module.exports = Parser
```

```
module.exports = () => { // not the same as object {}
  return {
    parse: () => {}
  }
}
```

`import` and `import()`

Caching

`require.cache`

Global

```
var limit = 1000 // local, not available outside
const height = 50 // local
let i = 10 // local
console = () => {} // global, overwrites console outside
global.Parser = {} // global
max = 999 // global too
```

---

npm

- registry
- cli: folders, git, private registries (self hosted npm, Nexus, Artifactory)

- yarn
- pnpm

---

npm Git

```
npm i expressjs/express -E
npm ls express
```

```
npm i expressjs/express#4.14.0 -E
```

```
npm i --dry-run express
npm ls -g --depth=0
npm ll -g --depth=0
npm ls -g --depth=0 --json
```

npm installs in ~/node_modules (if no local)

```
npm init -y
```

```
 npm config ls
```

```
init-author-name = "Azat Mardan"
init-author-url = "http://azat.co/"
init-license = "MIT"
init-version = "1.0.1"
```

Setting up npm registry

```
npm config set registry "http://registry.npmjs.org/"
```

Setting up npm proxy

```
npm config set https-proxy http://proxy.company.com:8080
npm config set proxy http://proxy_host:port
```

Note: The https-proxy doesn't have https as the protocol, but http.

-S (default in v5)
-D
-O
-E

---

npm update and npm outdated

< and <=
=
.x
~
^
> and >=

```
npm home express
npm repo express
npm docs express
```

```
npm link
npm unlink
```

Module 3: Node Event Loop and Async Programming

--

Event loop

Input and Output

Input/Output Messages are most "expensive" (slow)

- Disk
- Networking

Dealing with Slow I/O

- Synchronous
- Forking
- Threading
- Event loop

https://youtu.be/PNa9OMajw9w?t=5m48s (https://youtu.be/PNa9OMajw9w?t=5m48s)

Call stack

push, pop functions
FILO/LIFO/LCFS - functions removed from top (opposite of queue)

^https://techterms.com/definition/filo

```
const f3 = () => {
  console.log('executing f3')
  undefinedVariableError
}
const f2 = () => {
  console.log('executing f2')
  f3()
}
const f1 = () => {
  console.log('executing f1')
  f2()
}

f1()
```

---

push to call stack

```
f3() // last
f2()
f1()
anonymous() // first
```

```
> f1()
executing f1
executing f2
executing f3
ReferenceError: undefinedVariableError is not defined
    at f3 (repl:3:1)
    at f2 (repl:3:1)
    at f1 (repl:3:1)
    at repl:1:1
    at ContextifyScript.Script.runInThisContext (vm.js:23:33)
    at REPLServer.defaultEval (repl.js:339:29)
    at bound (domain.js:280:14)
    at REPLServer.runBound [as eval] (domain.js:293:12)
    at REPLServer.onLine (repl.js:536:10)
    at emitOne (events.js:101:20)
```

Event Queue

FIFO to push to call stack

```
const f3 = () => {
  console.log('executing f3')
  setTimeout(()=>{
    undefinedVariableError
  }, 100)
}
const f2 = () => {
  console.log('executing f2')
  f3()
}
const f1 = () => {
  console.log('executing f1')
  f2()
}

f1()
```

Different call stack. No f1, f2, f3 for the setTimeout callback (comes from event queue)

```
> f1()
executing f1
executing f2
executing f3
undefined
> ReferenceError: undefinedVariableError is not defined
    at Timeout.setTimeout [as _onTimeout] (repl:4:1)
    at ontimeout (timers.js:386:14)
    at tryOnTimeout (timers.js:250:5)
    at Timer.listOnTimeout (timers.js:214:5)
>
```

---

setTimeout vs. setImmediate vs. process.nextTick

`setImmediate()` similar to setTimeout with 0 but timing is different sometimes, it is recommended when you need to execute on the next cycle
`process.nextTick` - making functions fully async (same cycle)

---

1. timers
2. I/O callbacks
3. idle, prepare
4. poll (incoming connections, data)
5. check
6. close callbacks

^https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/

Error-first callback

define:

```
const myFn = (cb) => {
  // define error and data
  // do something...
  cb(error, data)
}
```

Use:

```
myFn((error, data)=>{

})
```

Argument names don't matter (order does):

```
myFn((err, result)=>{

})
```

Callbacks not always async

```
arr.map((item, index, list)=>{

})
```

Errors first but the callback last

(Popular convention but not enforced by Node)

Promises

- Consume from a module (axios, koa, etc.)
- Create your own using ES6 Promise or a library (bluebird () or q (https://documentup.com/kriskowal/q/))

Usage and consumption of ready promises

Callbacks Syntax

```
asyncFn1((error1, data1) => {
  asyncFn2(data1, (error2, data2) => {
    asyncFn3(data2, (error3, data3) => {
      asyncFn4(data3, (error4, data4) => {
        // Do something with data4
      })
    })
  })
})
```

Promise Syntax Style

```
promise1(data1)
  .then(promise2)
  .then(promise3)
  .then(promise4)
  .then(data4=>{
    // Do something with data4
  })
  .catch(error=>{
    // handle error1, 2, 3 and 4
  })
```

(separation of data and control flow arguments)

Axios Example

```
const axios = require('axios')
axios.get('http://azat.co')
  .then((response)=>response.data)
  .then(html => console.log(html))
```

```
const axios = require('axios')
axios.get('https://azat.co')
  .then((response)=>response.data)
  .then(html => console.log(html))
  .catch(e=>console.error(e))
```

```
Error: Hostname/IP doesn't match certificate's altnames: "Host: azat.co. is not
in the cert's altnames: DNS:*.github.com, DNS:github.com, DNS:*.github.io,
DNS:github.io"
```

Naive Promise: Callback Async Function

```
function myAsyncTimeoutFn(data, callback) {
  setTimeout(() => {
    callback()
  }, 1000)
}

myAsyncTimeoutFn('just a silly string argument', () => {
  console.log('Final callback is here')
})
```

Naive Promise: Implementation

```
function myAsyncTimeoutFn(data) {

  let _callback = null
  setTimeout( () => {
    if ( _callback ) callback()
  }, 1000)

  return {
    then(cb){
      _callback = cb
    }
  }

}

myAsyncTimeoutFn('just a silly string argument').then(() => {
  console.log('Final callback is here')
})
```

Naive Promise: Implementation with Errors

```
const fs = require('fs')
function readFilePromise( filename ) {
  let _callback = () => {}
  let _errorCallback = () => {}

  fs.readFile(filename, (error, buffer) => {
    if (error) _errorCallback(error)
    else _callback(buffer)
  })

  return {
    then( cb, errCb ){
      _callback = cb
      _errorCallback = errCb
    }
  }

}
```

Naive Promise: Reading File

```
readFilePromise('package.json').then( buffer => {
  console.log( buffer.toString() )
  process.exit(0)
}, err => {
  console.error( err )
  process.exit(1)
})
```

Naive Promise: Triggering Error

```
readFilePromise('package.jsan').then( buffer => {
  console.log( buffer.toString() )
  process.exit(0)
}, err => {
  console.error( err )
  process.exit(1)
})
```

```
{ Error: ENOENT: no such file or directory, open 'package.jsan'
  errno: -2,
  code: 'ENOENT',
  syscall: 'open',
  path: 'package.jsan' }
```

ES6/ES2015 Promise

```
Promise === global.Promise
```

(Node version 8+)

Promise take callback with `resolve` and `reject`

Simple Proper Promise Implementation (from ES6/ES2015)

```
const fs = require('fs')
function readJSON(filename, enc='utf8'){
  return new Promise(function (resolve, reject){
    fs.readFile(filename, enc, function (err, res){
      if (err) reject(err)
      else {
        try {
          resolve(JSON.parse(res))
        } catch (ex) {
          reject(ex)
        }
      }
    })
  })
}

readJSON('./package.json').then(console.log)
```

Advanced Proper Promise Implementation (from ES6/ES2015) for both promises and callbacks

```
const fs = require('fs')

const readFileIntoArray = function(file, cb = null) {
  return new Promise((resolve, reject) => {
    fs.readFile(file, (error, data) => {
      if (error) {
        if (cb) return cb(error)
        return reject(error)
      }

      const lines = data.toString().trim().split('\n')
      if (cb) return cb(null, lines)
      else return resolve(lines)
    })
  })
}
```

Example call

```
const printLines = (lines) => {
  console.log(`there are ${lines.length} line(s)`)
  console.log(lines)
}
const FILE_NAME = __filename

readFileIntoArray(FILE_NAME)
  .then(printLines)
  .catch(console.error)

readFileIntoArray(FILE_NAME, (error, lines) => {
  if (error) return console.error(error)
  printLines(lines)
})
```

Event Emitters

1. Import `require('events')`
2. Extend `class Name extends ...`
3. Instantiate new `Name()`
4. Add listeners `.on()`
5. Emit `.emit()`

Promises vs events

- Events are sync
- React to same event from multiple places
- React to same event multiple times

Events are about building extensible functionality and making modular code flexible

- `.emit()` can be in the module and `.on()` in the main program which consumes the module
- `.on()` can be in the module and `.emit()` in the main program
- pass data with `emit()`
- `error` is a special event (if listen to it then no crashes)

- on() execution happen in the order in which they are defined (`prependListener` or `removeListener`)

- Default maximum listeners is 10 (to find memory leaks), `setMaxListeners` (source (https://github.com/nodejs/node/blob/master/lib/events.js#L81))

```
EventEmitter.prototype.setMaxListeners = function setMaxListeners(n) {
  if (typeof n !== 'number' || n < 0 || isNaN(n)) {
    const errors = lazyErrors();
    throw new errors.RangeError('ERR_OUT_OF_RANGE', 'n',
                                'a non-negative number', n);
  }
  this._maxListeners = n;
  return this;
};
```

---

class

```
process.nextTick(()=>{
  this.emit()
})
```

Async/await

```
const axios = require('axios')
const getAzatsWebsite = async () => {
  const response = await axios.get('http://azat.co')
  return response.data
}
getAzatsWebsite().then(console.log)
```

Util.promisify

```
const util = require('util')
const f = async function() {
  try {
    await util.promisify(setTimeout)(()=>{consoleg.log('here')}, 1000)
  } catch(e) {
    await Promise.reject(new Error('test'))
  }
}

f()
```

```
const axios = require('axios')
const {expect} = require('chai')
const app = require('../server.js')
const port = 3004

before(async function() {
  await app.listen(port, ()=>{console.log('server is running')})
  console.log('code after the server is running')
})

describe('express rest api server', async () => {
  let id

  it('posts an object', async () => {
    const {data: body} = await
axios.post(`http://localhost:${port}/collections/test`, { name: 'John', email:
'john@rpjs.co'})
    expect(body.length).to.eql(1)
    expect(body[0]._id.length).to.eql(24)
    id = body[0]._id
  })

  it('retrieves an object', async () => {
    const {data: body} = await
axios.get(`http://localhost:${port}/collections/test/${id}`)
    // console.log(body)
    expect(typeof body).to.eql('object')
    expect(body._id.length).to.eql(24)
    expect(body._id).to.eql(id)
    expect(body.name).to.eql('John')
  })
  // ...
})
```

Project: Koa Server with Mocha (async/await)

Module 3: Networking

---

net

Any server, not just http or https!

```
const server = require('net').createServer()
server.on('connection', socket => {
  socket.write('Enter your command: ') // Sent to client
  socket.on('data', data => {
    // incoming data from a client
  })

  socket.on('end', () => {
    console.log('Client disconnected')
  })
})

server.listen(8000, () => console.log('Server bound'))
```

Chat

chat.js:

```
if (!sockets[socket.id]) {
  socket.name = data.toString().trim()
  socket.write(`Welcome ${socket.name}!\n`)
  sockets[socket.id] = socket
  return
}
Object.entries(sockets).forEach(([key, cs]) => {
  if (socket.id === key) return
  cs.write(`${socket.name} ${timestamp()}: `)
  cs.write(data)
})
```

Client?

```
telnet localhost 8000
```

or

```
nc localhost 8000
```

or write your own TCP/IP client using Node, C++, Python, etc.

Demo: Bitcoin Price Ticker

bitcoin-price-ticker.js

Ticker Server

```
const https = require('https')

const server = require('net').createServer()
let counter = 0
let sockets = {}
server.on('connection', socket => {
  socket.id = counter++

  console.log('Welcome to Bitcoin Price Ticker (Data by Coindesk)')
  console.log(`There are ${counter} clients connected`)
  socket.write('Enter currency code (e.g., USD or CNY): ')

  socket.on('data', data => {
    // process data from the client
  })

  socket.on('end', () => {
    delete sockets[socket.id]
    console.log('Client disconnected')
  })
})

server.listen(8000, () => console.log('Server bound'))
```

---

processing data from the client:

```
    let currency = data.toString().trim()
    if (!sockets[socket.id]) {
      sockets[socket.id] = {
        currency: currency
      }
      console.log(currency)
    }
    fetchBTCPrice(currency, socket)
    clearInterval(sockets[socket.id].interval)
    sockets[socket.id].interval = setInterval(()=>{
      fetchBTCPrice(currency, socket)
    }, 5000)
```

Making request to [Coindesk API (https://www.coindesk.com/api/)](https://www.coindesk.com/api/) (HTTPS!)

API: https://api.coindesk.com/v1/bpi/currentprice/.json

https://api.coindesk.com/v1/bpi/currentprice/USD.json
(https://api.coindesk.com/v1/bpi/currentprice/USD.json)
https://api.coindesk.com/v1/bpi/currentprice/JPY.json
(https://api.coindesk.com/v1/bpi/currentprice/JPY.json)
https://api.coindesk.com/v1/bpi/currentprice/RUB.json

(https://api.coindesk.com/v1/bpi/currentprice/RUB.json)
https://api.coindesk.com/v1/bpi/currentprice/NYC.json
(https://api.coindesk.com/v1/bpi/currentprice/NYC.json)

```
{
  "time": {
    "updated": "Jan 9, 2018 19:52:00 UTC",
    "updatedISO": "2018-01-09T19:52:00+00:00",
    "updateduk": "Jan 9, 2018 at 19:52 GMT"
  },
  "disclaimer": "This data was produced from the CoinDesk Bitcoin Price Index
(USD). Non-USD currency data converted using hourly conversion rate from
openexchangerates.org",
  "bpi": {
    "USD": {
      "code": "USD",
      "rate": "14,753.6850",
      "description": "United States Dollar",
      "rate_float": 14753.685
    }
  }
}
```

```
const fetchBTCPrice = (currency, socket) => {
  const req = https.request({
    port: 443,
    hostname: 'api.coindesk.com',
    method: 'GET',
    path: `/v1/bpi/currentprice/${currency}.json`
  }, (res) => {
    let data = ''
    res.on('data', (chunk) => {
      data +=chunk
    })
    res.on('end', () => {
      socket.write(`1 BTC is ${JSON.parse(data).bpi[currency].rate}
${currency}\n`)
    })
  })
  req.end()
}
```

```
telnet localhost 8000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Enter currency code (e.g., USD or CNY): USD
1 BTC is 14,707.9438 USD
1 BTC is 14,694.5113 USD
1 BTC is 14,694.5113 USD
CNY
1 BTC is 40,202.5000 CNY
RUB
1 BTC is 837,400.5342 RUB
1 BTC is 837,400.5342 RUB
1 BTC is 837,400.5342 RUB
```

---

http

static file server (file-server.js)

```
const url = require('url')
const SECRET = process.env.SECRET
const server = require('http').createServer((req, res) => {
  console.log(`URL is ${req.url} and the method is ${req.method}`)
  const courseId = req.url.match(/courses\/([0-9]*)/)[1] // works for
/courses/123 to get 123
  const query = url.parse(req.url, true).query // works for /?
key=value&key2=value2
  if (courseId && API_KEY===SECRET) {
    fs.readFile('./archive.sql', (error, data)=>{
      if (error) {
        res.writeHead(500)
        res.end()
      } else {
        res.writeHead(200, {'Content-Type': 'text/plain' })
        res.end(data)
      }
    })
  }
}).listen(3000, () => {
  console.log('server is listening on 3000')
})
```

Command to run the server:

```
SECRET=NNN nodemon file-server.js
```

Browser request: http://localhost:3000/courses/123?API_KEY=NNN
[(http://localhost:3000/courses/123?API_KEY=NNN)](http://localhost:3000/courses/123?API_KEY=NNN)

You can use switch...

```
const server = require('http').createServer((req, res) => {
  switch (req.url) {
    case '/api':
      res.writeHead(200, { 'Content-Type': 'application/json' })
      // fetch data from a database
      res.end(JSON.stringify(data))
      break
    case '/home':
      res.writeHead(200, { 'Content-Type': 'text/html' })
      // send html from a file
      res.end(html)
      break
    default:
      res.writeHead(404)
      res.end()
  }
}).listen(3000, () => {
  console.log('server is listening on 3000')
})
```

Find a problem with this server (from [Advanced Node by Samer Buna (https://app.pluralsight.com/player?course=nodejs-advanced&author=samer-buna&name=nodejs-advanced-m5&clip=3&mode=live)](https://app.pluralsight.com/player?course=nodejs-advanced&author=samer-buna&name=nodejs-advanced-m5&clip=3&mode=live)):

```
const fs = require('fs')
const server = require('http').createServer()
const data = {}

server.on('request', (req, res) => {
  switch (req.url) {
  case '/api':
    res.writeHead(200, { 'Content-Type': 'application/json' })
    res.end(JSON.stringify(data))
    break
  case '/home':
  case '/about':
    res.writeHead(200, { 'Content-Type': 'text/html' })
    res.end(fs.readFileSync(`.${req.url}.html`))
    break
  case '/':
    res.writeHead(301, { 'Location': '/home' })
    res.end()
    break
  default:
    res.writeHead(404)
    res.end()
  }
})

server.listen(3000)
```

Always reading (no caching) and blocking!

```
  case '/about':
    res.writeHead(200, { 'Content-Type': 'text/html' })
    res.end(fs.readFileSync(`.${req.url}.html`))
    break
```

Use Status Codes

```
http.STATUS_CODES
```

---

https

Server needs the key and certificate files:

```
openssl req -x509 -newkey rsa:2048 -nodes -sha256 -subj
'/C=US/ST=CA/L=SF/O=NO\x08A/OU=NA' \
  -keyout server.key -out server.crt
```

---

https server:

```
const https = require('https')
const fs = require('fs')

const server = https.createServer({
  key: fs.readFileSync('server.key'),
  cert: fs.readFileSync('server.crt')
}, (req, res) => {
  res.writeHead(200)
  res.end('hello')
}).listen(443)
```

---

https request with streaming

```
const https = require('https')

const req = https.request({
    hostname: 'webapplog.com',
    port: 443,
    path: '/',
    method: 'GET'
  }, (res) => {
  console.log('statusCode:', res.statusCode)
  console.log('headers:', res.headers)

  res.on('data', (chunk) => {
    process.stdout.write(chunk)
  })
})

req.on('error', (error) => {
  console.error(error)
})
req.end()
```

http2

```
openssl req -x509 -newkey rsa:2048 -nodes -sha256 -subj
'/C=US/ST=CA/L=SF/O=NO\x08A/OU=NA' \
  -keyout server.key -out server.crt
```

```
const http2 = require('http2')
const fs = require('fs')

const server = http2.createSecureServer({
  key: fs.readFileSync('server.key'),
  cert: fs.readFileSync('server.crt')
}, (req, res) => {
  res.end('hello')
})
server.on('error', (err) => console.error(err))
server.listen(3000)
```

```
const http2 = require('http2')
const fs = require('fs')

const server = http2.createSecureServer({
  key: fs.readFileSync('server.key'),
  cert: fs.readFileSync('server.crt')
})

server.on('error', (err) => console.error(err))
server.on('socketError', (err) => console.error(err))

server.on('stream', (stream, headers) => {
  // stream is a Duplex
  stream.respond({
    'content-type': 'text/html',
    ':status': 200
  })
  stream.end('<h1>Hello World</h1>')
})

server.listen(3000)
```

```
$ curl https://localhost:3000/ -vik
```

```
 Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 3000 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* Cipher selection:
...
* SSL connection using TLSv1.2 / ECDHE-RSA-AES128-GCM-SHA256
* ALPN, server accepted to use h2
* Server certificate:
*  subject: C=US; ST=CA; L=SF; O=NOx08A; OU=NA
* Using HTTP2, server supports multi-use
* Connection state changed (HTTP/2 confirmed)
```

http2 Server Push

HTTP/1 makes two requests:

1. HTML: index.html refers to static assets
2. Assets: style.css + bundle.js + favicon.ico + logo.png

HTTP/2 with server push just one:

1. HTML and assets are pushed by the server

(Assets are not used unless referred to by HTML)

```
const http2 = require('http2')
const fs = require('fs')

const server = http2.createSecureServer({
  key: fs.readFileSync('server.key'),
  cert: fs.readFileSync('server.crt')
})

server.on('error', (err) => console.error(err))
server.on('socketError', (err) => console.error(err))
```

```
server.on('stream', (stream, headers) => {
  stream.respond({
    'content-type': 'text/html',
    ':status': 200
  })
  stream.pushStream({ ':path': '/myfakefile.js' }, (pushStream) => {
    pushStream.respond({
      'content-type': 'text/javascript',
      ':status': 200
    })
    pushStream.end(`alert('you win')`)
  })
  stream.end('<script src="/myfakefile.js"></script><h1>Hello World</h1>')
})

server.listen(3000)
```

Additional server push articles

- What's the benefit of Server Push? (https://http2.github.io/faq/#whats-the-benefit-of-server-push)
- Announcing Support for HTTP/2 Server Push (https://blog.cloudflare.com/announcing-support-for-http-2-server-push-2)
- Innovating with HTTP 2.0 Server Push

(https://www.igvita.com/2013/06/12/innovating-with-http-2.0-server-push)

Demo: Advanced Express REST API routing in HackHall

Conclusion

Just don't use core http directly. Use Express, Hapi or Koa.

Module 4: Debugging

Debugging

---

console.log is one of the best debuggers

- Not breaking the execution flow
- Nothing extra needed (unlike Node Inspector/DevTools or VS Code)
- Robust: clearly shows if a line is executed
- Clearly shows data

Console Tricks

Streaming logs to files

```
const fs = require('fs')

const out = fs.createWriteStream('./out.log')
const err = fs.createWriteStream('./err.log')

const console2 = new console.Console(out, err)

setInterval(() => {
  console2.log(new Date())
  console2.error(new Error('Whoops'))
}, 500)
```

```
console.log('Step', 2) // Step2
const name = 'Azat'
const city = 'San Francisco'
console.log('Hello %s from %s', name, city)
```

```
const util = require('util')
console.log(util.format('Hello %s from %s', name, city)) // Hello Azat from San Francisco
console.log('Hello %s from %s', 'Azat', {city: 'San Francisco'}) // Hello Azat from [object Object]
console.log({city: 'San Francisco'}) // { city: 'San Francisco' }
console.log(util.inspect({city: 'San Francisco'})) // { city: 'San Francisco' }
```

```
const str = util.inspect(global, {depth: 0})
console.dir(global, {depth: 0})
```

```
info = log
warn = error
trace // prints call stack
assert // require('assert')
```

Console Timers

```
console.log('Ethereum transaction started')
console.time('Ethereum transaction')
web3.send(txHash, (error, results)=>{
  console.timeEnd('Ethereum transaction') // Ethereum transaction: 4545.921ms
})
```

Real Debuggers

- CLI
- DevTools
- VS Code

Node CLI Debugger

```
$ node inspect debug-me.js
< Debugger listening on ws://127.0.0.1:9229/80e7a814-7cd3-49fb-921a-2e02228cd5ba
< For help see https://nodejs.org/en/docs/inspector
< Debugger attached.
Break on start in myscript.js:1
> 1 (function (exports, require, module, __filename, __dirname) { global.x = 5;
  2 setTimeout(() => {
  3   console.log('world');
debug>
```

```
Stepping#
cont, c - Continue execution
next, n - Step next
step, s - Step in
out, o - Step out
pause - Pause running code (like pause button in Developer Tools)
```

Node V8 Inspector

```
$ node --inspect index.js
Debugger listening on 127.0.0.1:9229.
To start debugging, open the following URL in Chrome:
    chrome-devtools://devtools/bundled/inspector.html?
experiments=true&v8only=true&ws=127.0.0.1:9229/dc9010dd-f8b8-4ac5-a510-
c1a114ec7d29
```

better to break right away:

```
$ node --inspect --debug-brk index.js
```

Node V8 Inspector Demo

VS Code Demo

CPU profiling

Networking Debugging with DevTools

Module 4: Scaling

Why You Need to Scale

- Performance (e.g., under 100ms response time)
- Availability (e.g., 99.999%)
- Fault tolerance (e.g., zero downtime)

^Zero downtime
^ Offload the workload: when Node server is a single process, it can be easily blocked
^https://blog.interfaceware.com/disaster-recovery-vs-high-availability-vs-fault-tolerance-what-are-the-differences/

Scaling Strategies

- Forking (just buy more EC2s) - what we will do
- Decomposing (e.g., microservices just for bottlenecks) - in another course
- Sharding (e.g., eu.docusign.com and na2.docusign.net) - not recommended

---

Offload the workload

- spawn() - events, stream, messages, no size limit, no shell
- fork() - Node processes, exchange messages
- exec() - callback, buffer, 1Gb size limit, creates shell
- execFile() - exec file, no shell

- spawnSync()
- execFileSync()
- execSync()
- forkSync()

Executing bash and spawn params

```
const {spawn} = require('child_process')
spawn('cd $HOME/Downloads && find . -type f | wc -l',
  {stdio: 'inherit',
  shell: true,
  cwd: '/',
  env: {PASSWORD: 'dolphins'}
})
```

Executing Python with exec

```

```

---

os Module

```
const os = require('os')
console.log(os.freemem())
console.log(os.type())
console.log(os.release())
console.log(os.cpus())
console.log(os.uptime())
console.log(os.networkInterface())
```

```
{ lo0:
   [ { address: '127.0.0.1',
       netmask: '255.0.0.0',
       family: 'IPv4',
       mac: '00:00:00:00:00:00',
       internal: true },
  ...
  en0:
   [ { address: '10.0.1.4',
       netmask: '255.255.255.0',
       family: 'IPv4',
       mac: '78:4f:43:96:c6:f1',
       internal: false } ],
  ...
```

```
ifconfig | grep "inet " | grep -v 127.0.0.1
```

CPU Usage in %

```
//os-cpu.js
const os = require('os')
let cpus = os.cpus()

cpus.forEach((cpu, i) => {
  console.log('CPU %s:', i)
  let total = 0

  for (let type in cpu.times) {
    total += cpu.times[type]
  }

  for (let type in cpu.times) {
    console.log(`\t ${type} ${Math.round(100 * cpu.times[type] / total)}%`)
  }
})
```

Free Memory

- `free =

Useful Libraries

-
- https://www.npmjs.com/package/systeminformation (https://www.npmjs.com/package/systeminformation) and https://github.com/sebhildebrandt/systeminformation (https://github.com/sebhildebrandt/systeminformation)

---

cluster

Load testing

```
npm i loadtest -g
loadtest -c 10 --rps 100 10.0.1.4:3000
```

Messaging

spawn, fork, exec

Offloading CPU-intensive tasks

Outro

Summary