1 Session

Recommendations

Node Training

nearForm

1 Session

Recommendations

Node Training

Professional Node.js

- Web Frameworks
- Production-level modules
- Code-level best practices
- Performant Code
- Unit Testing

nearForm

Web Frameworks: Express

express
web application framework for node

- http://expressjs.com
- Most popular framework
- Defined the “middleware” pattern
- Version 4, almost certainly end-of-life
- Now used a basis for more complete frameworks

nearForm

Web Frameworks: Kraken

krakenjs

- http://krakenjs.com
- Developed by Paypal; based on Express
- Great support for page-templating, and asset management
- Also supports express middleware with declarative configuration
- Version 1, in production use by PayPal and others
- Maintainer: PayPal team, well supported

nearForm

Web Frameworks: Hapi

- http://hapijs.com
- Developed by Walmart; very much it's own eco-system
- "Batteries-included" philosophy, but also has a plugin system
- Most suitable for building web API's, e.g. for mobile apps
- *Warning!* Does not support middleware pattern
- Version 6; proven under high load - Black Friday 2013 - #nodebf
- Maintainer: Walmart, principally Eran Hammer

Web Frameworks: Sails

- http://sailsjs.org
- This is the best "Rails" for Node
- Follows traditional web Model-View-Controller
- Packages an ORM, so time to "first working version" is fast
- Version 0.10, "not-quite" production ready
- Maintainer: Mike McNeil

Production-level Modules

require	author	description
request	mikeal	HTTP client
hyperquest	substack	Lighter HTTP client
needle	tomas	Lighter HTTP client
minimist	substack	Command line options
async	caolan	Async patterns
mocha	tjholowaychuk	Unit testing

nearForm

Production-level Modules

require	author	description
tape	substack	Unit testing
chai	jakeluer	Assertions
underscore	jashkenas	Functional patterns
lodash	jdalton	Faster functional patterns
nconf	indexzero	Configuration
passport	jaredhanson	Login and authentication

nearForm

Production-level Modules

require	author	description
browserify	substack	Browser distribution
event-strm	dominictarr	Stream utilities
JSONStream	dominictarr	Stream utilities
through2	rvagg	Stream utilities
split2	matteo.collina	Stream utilities
pump	mafintosh	Stream utilities

nearForm

Production-level Modules

require	author	description
duplexify	mafintosh	Stream utilities
xml2js	leonidas	XML to JavaScript
bunyan	trentm	Logging
express	tjholowaychuk	Server framework
hapi	hueniverse	Server framework
restify	mcavage	REST API builder

nearForm

Production-level Modules

require	author	description
moment	timrwood	Date manipulation
debug	tjholowaychuk	Debug printer
ws	einaros	Websockets
socket.io	rauchg	Realtime
levelup	rvagg	LevelDB
mongodb	christkv	MongoDB

nearForm

Production-level Modules

require	author	description
redis	mjr	Redis
pg	brianc	Postgres
mysql	felixge	MySQL
gulp	contra	Build system
grunt	cowboy	Build system

nearForm

Production-level Modules

require	author	description
minimatch	isaacs	Glob matching
glob	isaacs	Glob matching
bl	rvagg	Binary parsing
dockerode	apocas	Docker management

nearForm

Code Best Practices

- Follow JavaScript best practices
  - Read "The Good Parts", again!
- Understand the Node.js environment
  - it's not the browser

try-catch

- Avoid try-catch in async code
  - every caller needs to handle exceptions properly
    - otherwise they "disappear"
  - domains are not widely supported
- Pass errors through callbacks

```
doMainThing(function (data, done) {
  queryDatabase(data.query, function(err, result) {
    if (err) return done(err);
    ...
  });
});
```

nearForm

wrapper functions

- Reduce code noise
- Don't be afraid to build functions dynamically
  - `return function() { ... }`

```
function errHandler(fail, win) {
  return function(err, result) {
    if (err) { return fail(err); }
    win(result)
  }
}

queryDatabase(data.query,
  errHandler(done, function(result) {
    ...
  })
)
```

nearForm

Use async

- The async library helps avoid callback hell
- Important use case: rate limiting async loops

```
function dbInsert(item, done) {
  var sql = makeSQL(item)
  dbapi.execute(sql, function(err, result) {
    if(err) return done(err);
    done(null, setID(item, result))
  })
}

async.mapLimit([ entry, ...], 5, dbInsert,
  function(err, results) {
    if (err){ ... }
    console.dir(results);
  })
```

nearForm

Return Return Return

- Get out of callbacks as soon as you can
- Avoid additional indentation

```
//PREFER THIS:
doStuff(data, function(err, result) {
  if (err) { return console.log(err); }
  doWork(result)
})

//TO THIS:
doStuff(data, function(err, result) {
  if (err) {
    console.log(err)
  } else {
    doWork(result)
  }
})
```

nearForm

The for-loop closure Trap

loops containing functions behave unexpectedly

```
var letterprinters = [];
var letters = ['h','e','l','l','o'];

for(var i = 0; i < 5; i++) {
  letterprinters[i] = function(){
    console.log(letters[i])
  }
}

letterprinters[0](); // undefined !!!
```

nearForm

The for-loop closure Trap

Use `forEach` instead

```
var letterprinters = [];
var letters        = ['h','e','l','l','o'];

var i = 0
letters.forEach(function(letter) {
  letterprinters[i++] = function(){
    console.log(letter)
  }
})

letterprinters[0]();  // h
```

nearForm

Use a linter !

- jshint.com
- `npm install jshint –save`
- `package.json`:

```
{
  "scripts": {
    "jshint": "./node_modules/.bin/jshint *.js",
    "test": "npm run jshint && mocha"
  }
}
```

Node.js Performance

- Get in, and Get out, as fast as possible
  - don’t tax the CPU
- Don’t serve static assets
  - Use nginx, or a CDN
- Be careful with server-side templates
  - they cost CPU
  - prefer client-side rendering, if possible

nearForm

Node.js Sessions

- Express provides traditional HTTP client sessions
  - Don't use them!
- Don't have sessions at all
  - Prefer a robust caching layer and compartmentalize your data
  - In particular, this lets your Node process crash and restart safely
- If a third-party library needs sessions (e.g. passport), use externally persisted sessions

nearForm

Work with V8

- V8 Optimization consists of pure machine code rather than function calls, where possible
  - CPU *add* instruction versus runtime addition function
- It's easy to write code that V8 can't optimise:
  - `try-catch` - put these in a separate function
  - `debugger;`- don't leave these in your code, even if guarded by a debug flag
  - `eval` and `with` - what did you expect!
  - `arguments` - almost all usage kills the optimiser, but direct index-based access is OK

nearForm

Unit Testing

- **Mocha**: http://visionmedia.github.io/mocha/
    - most common unit testing framework
    - behaviour-driven testing style
- **Jasmine**: https://github.com/pivotal/jasmine
    - Node and Browser testing
    - also follows Behaviour style

nearForm

Unit Testing

- **NodeUnit**: https://github.com/caolan/nodeunit
  - more traditional style (like good old JUnit)
  - Written by same developer as async
- **TAP**: https://github.com/isaacs/node-tap
  - Implementation of the Test Anything Protocol for Node.js
    - see http://testanything.org
    - Used by many core Node developers. Traditional in style.

nearForm

Unit Testing: Mocks

- SinonJS: http://sinonjs.org/
  - ensure callbacks actually get called
  - provides fake timers - very useful!
- Proxyquire: https://github.com/thlorenz/proxyquire
  - replace require result with your mock
  - means you don't have to change any code
- Nock: https://github.com/pgte/nock
  - Intercepts HTTP calls on Core API
  - Again, no changes to your own code

nearForm

Unit Testing: Mocks

- SinonJS: http://sinonjs.org/
  - ensure callbacks actually get called
  - provides fake timers - very useful!
- Proxyquire: https://github.com/thlorenz/proxyquire
  - replace require result with your mock
  - means you don't have to change any code
- Nock: https://github.com/pgte/nock
  - Intercepts HTTP calls on Core API
  - Again, no changes to your own code

nearForm

Unit Testing: Coverage

- Istanbul: https://github.com/gotwarlost/istanbul
  - by far the most commonly used
  - works well with Mocha