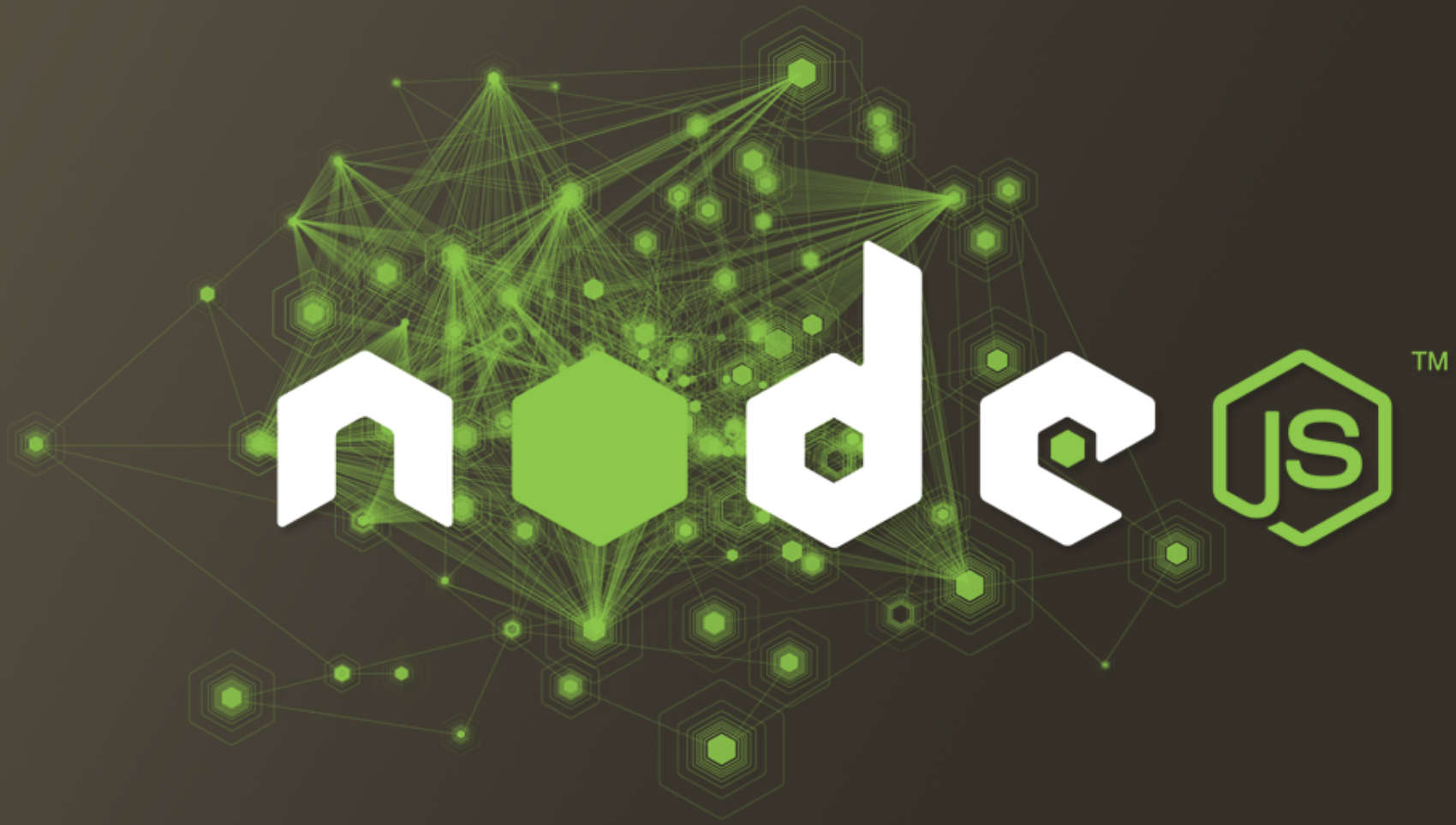


Côté serveur

Sommaire

- Introduction à NodeJS
- Architecture
- L’asynchronisme
- Modules et gestion de dépendances
- Node et le Web (Http, Connect, Express)
- Communication temps réel
- La gestion des streams
- Persistance de données
- Forge
- Node en mode cluster
- Au delà de NodeJS

Introduction à NodeJS
Les origines

- Node.js : plate-forme basée sur un moteur javascript
- Projet initié en 2009 par Ryan Dahl
- Publié en 2011, sponsorisé par Joyent
- Le serveur est écrit en C
- API non bloquante d'I/O en javascript
- https://github.com/joyent/node
- + de 500 commiters

Joyent
smart computing
https://www.joyent.com

Introduction à NodeJS
V8

- V8 est une VM Javascript OpenSource
- Développée en C++ par Google
- Moteur JS de Chrome
- Lancée en 2008
- Une des plus optimisée du marché (avec FireFox)
- Multi OS
- Le JS est compilé et optimisé à l'exécution
- Gestion mémoire, Garbage collector et inline caching
- Permet d'exécuter du C++ extérieur
- Gère ECMAScript 5 et partiellement ES6

https://github.com/v8/v8

Introduction à NodeJS
Asynchronisme

- NodeJS a une architecture asynchrone
- Toute opération est non-bloquante
- Mécanisme de callback
- Le code n'est jamais en attente d'une I/O
- La programmation asynchrone :
  - On peut programmer séquentiellement
  - Découpe du programme en fonctions
  - Les traitements font suite à des événements
- Les événements sont gérés par NodeJS
- Un callback est appelé pour récupérer le résultat du traitement

Introduction à NodeJS
Usages

- NodeJS est souple et permet de nombreux usages :
  - Scripts (ie : commande shell)
    - NPM, Grunt, Bower, Gulp, Karma, Mocha, …
  - Serveur Web
  - Robotique
    - NodeBots, NodeCopter, Rosnodejs
  - NodeOS
  - tessel.io
  - JS.everywhere

Introduction à NodeJS
Écosystème

- NodeJS a un écosystème riche
  - Intégralement OpenSource
  - Complet et varié
  - Réactif
- Dépôts officiels
  - npmjs.org
  - nodejsmodules.org
- Plus de 120 000 packages
- … la quantité ne fait pas la qualité

Introduction à NodeJS
Écosystème

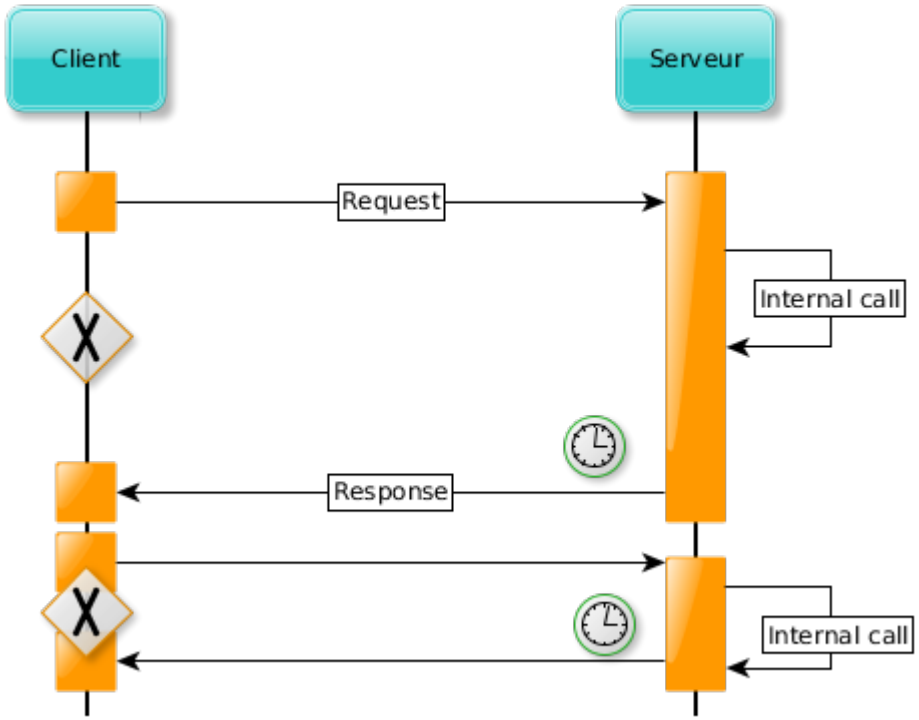
- Utilitaires
  - Underscore / Lodash
  - Async
  - Coffee
  - Q
  - Bluebird
- Web
  - Request
  - Connect
  - Express
  - Jade
  - Passport
- Test
  - Mocha
  - NodeUnit
  - Chai
  - Cucumber
- Build
  - Grunt
  - Bower
  - Gulp
  - Nodemon

Introduction à NodeJS
Premier script

- Fichier helloworld.js
  - `console.log("hello world");`
- Lancer le programme avec Node :
  - `$ node helloworld.js`
    `hello world`
- Le rendre auto exécutable
  - Ajouter au fichier :
    - `#!/usr/bin/env node`
  - Lancer les commandes :
    - `$ chmod +x helloworld.js`
    - `$ ./helloworld.js`
      `hello world`

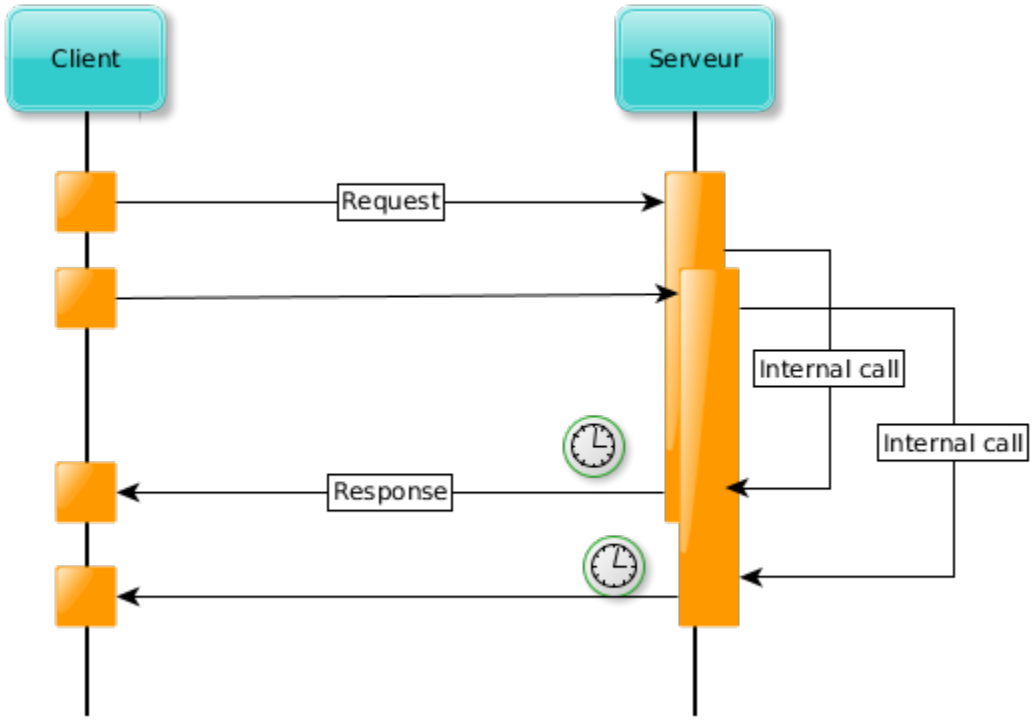
Architecture NodeJS
Asynchronisme

- Appel synchrone



Architecture NodeJS
Asynchronisme

- Appel asynchrone



L’asynchronisme

```
var hello = function() {
  for(var i = 0;
    i <= 100000000000000; i++) {

    if(i === 100000000000000) {
      console.log('fini :)');
    }
  }
};
hello();
console.log('et hop');
//-> fini :)
//-> et hop
```

```
var fs = require('fs');

var hello = null;
fs.readFile('hello.txt', 'utf-8',
  function(err, data) {
  if(err) throw err;
  console.log(data);
  hello = data;
});
console.log(hello, 'World');
//-> null World
//-> Hello
```

Architecture NodeJS
Asynchronisme

- La programmation asynchrone est très impactante
- Les traitements ne retournent pas la réponse
- Il faut réaliser les enchaînements par callback
- Les callbacks sont exécutés plus tard
- Le contexte de l'application aura alors changé
- La programmation fonctionnelle de JavaScript facilite les choses

```
var file = fs.readFile( 'test.json', function(data) {
    console.log( 'async', data);
});
console.log( 'sync', file);
// -> sync undefined
// -> async '[{"nom": "Young", "prenom": "Angus"}]'
```

Architecture NodeJS
Callback

- Fonction déclenchée sur un événement
- N'importe quelle référence à une fonction
- Des paramètres seront passés
- Par convention dans NodeJS :

```
var callback = function(err, result) {
    if(err) {
        console.error(err.message);
    } else {
        console.log(result);
    }
}
```

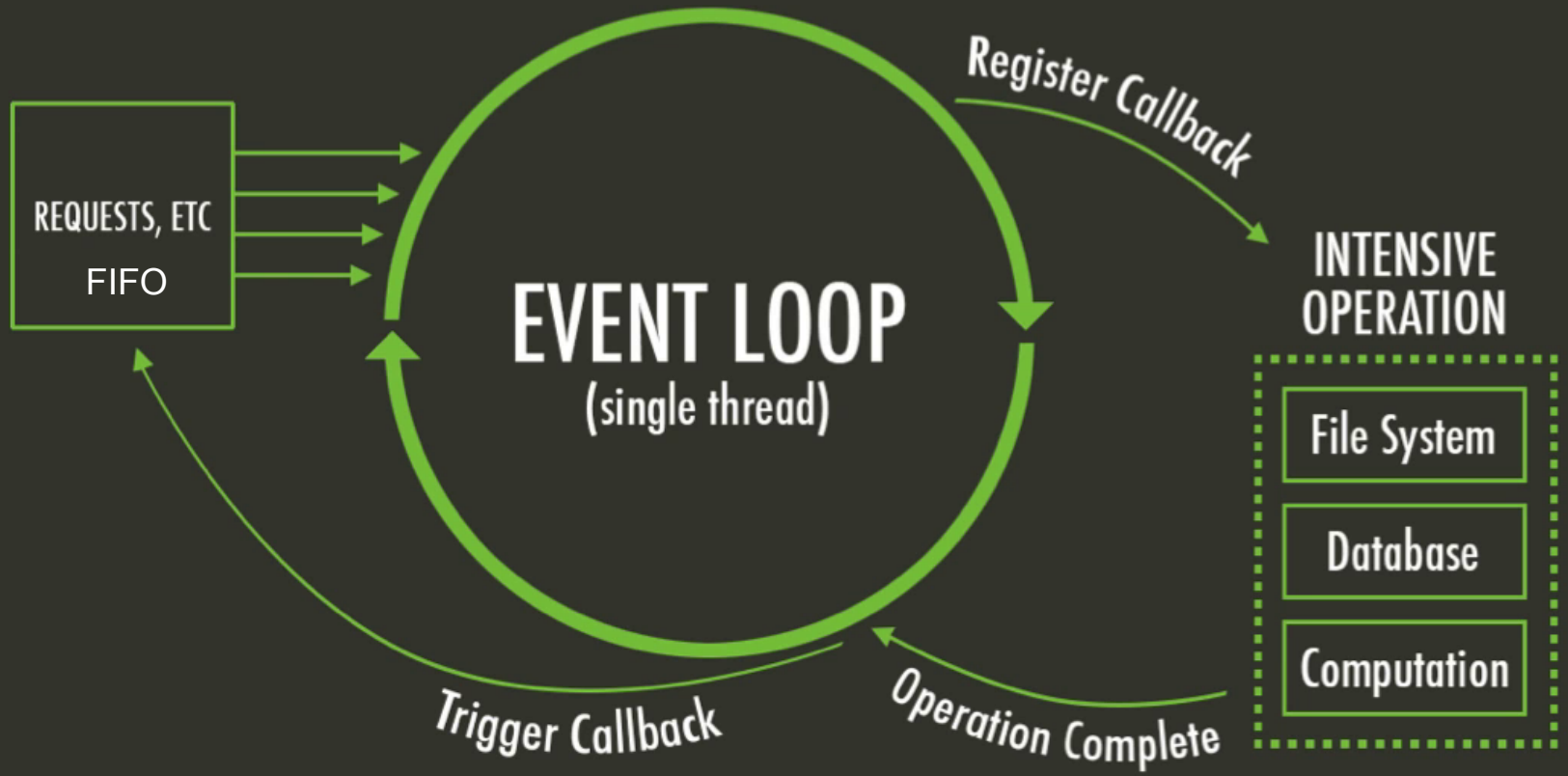
- Les APIs NodeJS

```
module.action(/* args */, callback);
```

- Callback sous forme de fonction anonyme

```
module.action(/* args */, function(err, result) {
    /* du traitement */
});
```

Architecture NodeJS
L'event loop

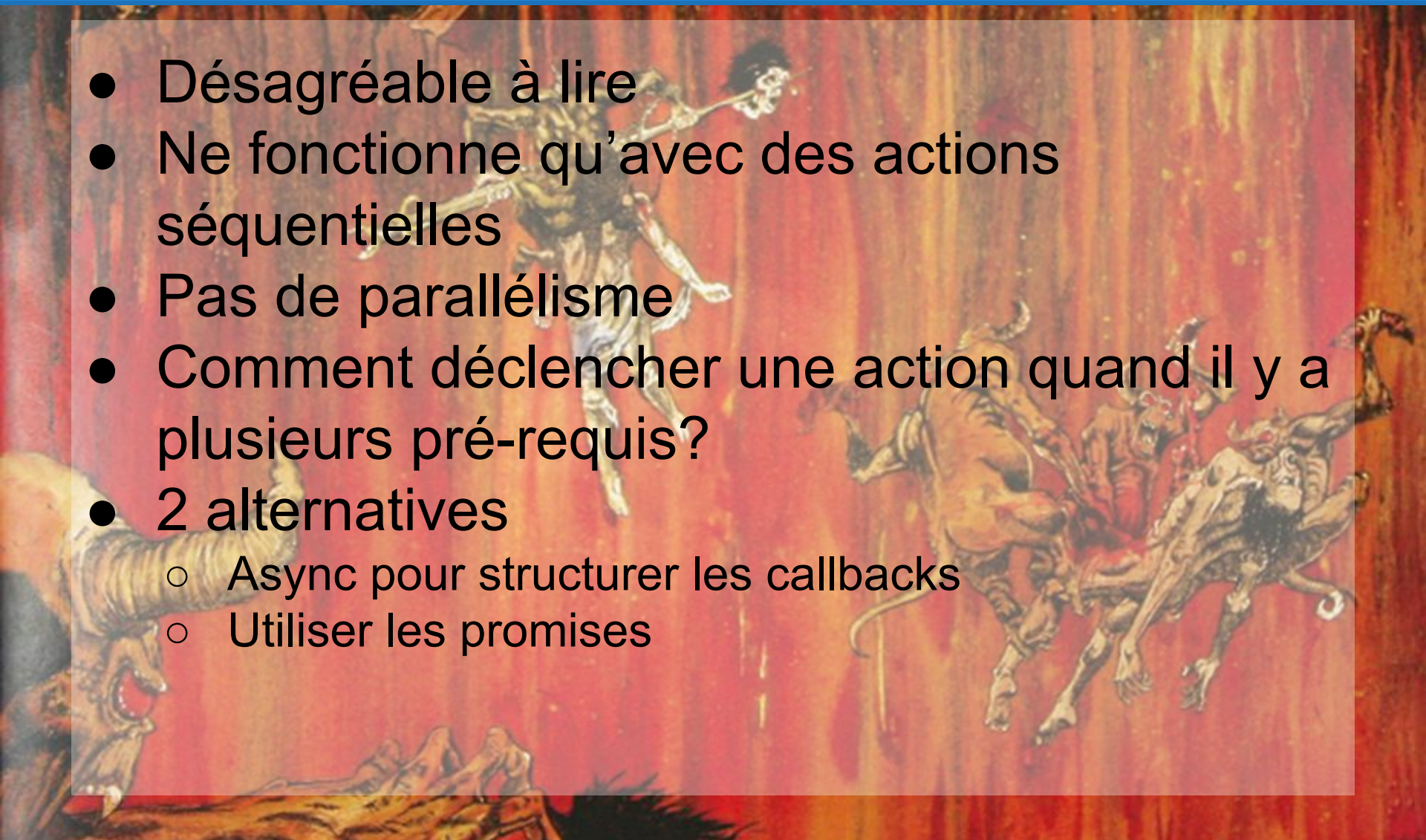


L'asynchronisme
Callback Hell / Pyramid of Doom

```
var fs = require('fs');
var callbackHell = function(jsonData, callback) {
  // On pousse les données dans un fichier
  fs.writeFile('data.json', jsonData, function(err) {
    // 1er callback on ajoute un élément dans un fichier
    fs.readFile('data2.json', 'utf-8', function(err, data2) {
      // 2nd callback
      jsonData.['data2'] = data2;
      // 3ème callback
      callback(err, jsonData);
    });
  });
};

var callbackHell({name: 'In Flames'}, function(err, result) {
  console.log(result);
});
```

L’asynchronisme
Callback Hell / Pyramid of Doom



- Désagréable à lire
- Ne fonctionne qu’avec des actions séquentielles
- Pas de parallélisme
- Comment déclencher une action quand il y a plusieurs pré-requis?
- 2 alternatives
  - Async pour structurer les callbacks
  - Utiliser les promises

L’asynchronisme
Async

- $ npm install async --save
- https://github.com/caolan/async
- Async ne change pas le paradigme des callbacks
- On manipule beaucoup de fonctions
- Quelques outils fournis:
  - map : lance une fonction sur un jeu de paramètres différents
  - parallel : lance un ensemble de fonctions en paralèle
  - series : chaîne un ensemble de fonctions

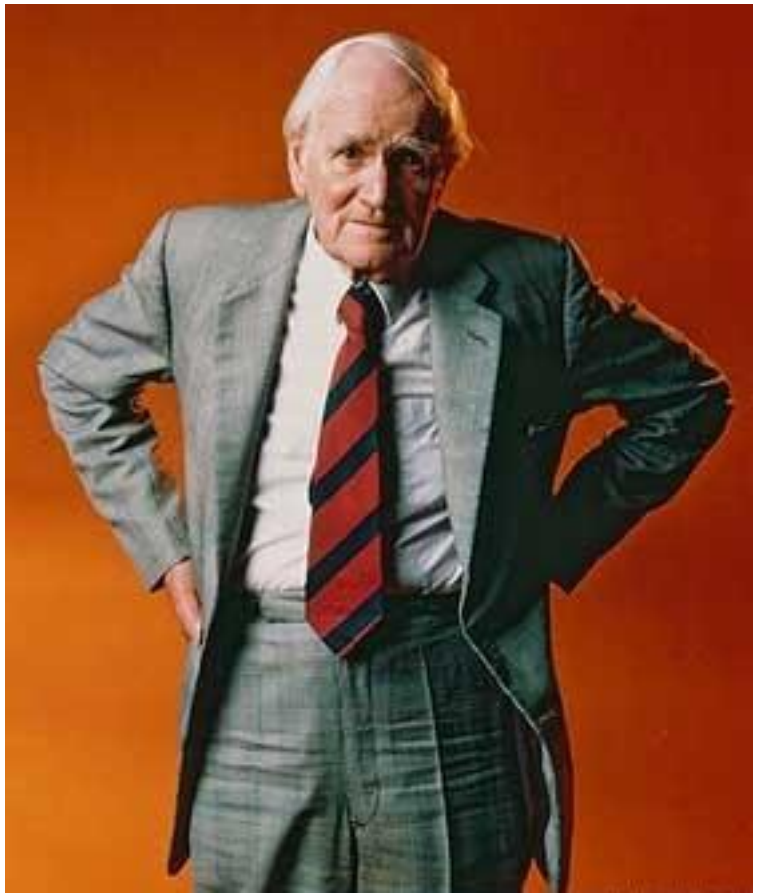
L’asynchronisme
Q

- $ npm install q --save
- http://github.com/kriskowal/q
- Nouveau paradigme : les promesses
- Remplace les callbacks
- Une promesse est un objet résolu de manière asynchrone
- On écoute l’événement de résolution avec un callback
- Alternative : bluebird

L'asynchronisme
Q

```
var funcPromise = function(param) {
  var deferred = Q.defer();
  /* … */
  if(condition) {
    deferred.reject(new Error( 'et zut!' ));
  } else {
    deferred.resolve(result);
  }
};

funcPromise( 'toto' ).then(function(result) {
  console.log( 'success' );
}, function(err) {
  console.error(err);
});
```



L’asynchronisme
Q

Utilisable avec les API de NodeJS :

```
Q.nfcall(module.action, param1, param2, …).then(function(err, result) {
  /* … */
});
```

On peut chaîner :

```
aPromise.then(function(result) {}, function(err) {})
    .then(function(result) {}, function(err) {})
    .then(function(result) {}, function(err) {});
```

On peut paralléliser :

```
Q.all([promise1, promise2, …]).then(function(result) {}, function(err)
{});
```

Gestion des erreurs :

```
aPromise.then(function(result) {})
    .then(function(result) {})
    .catch(function(err) {})
    .done();
```

Modules et gestion de dépendances
Les modules

- JavaScript n'apporte pas la notion de modules
- Cela change avec ECMAScript6
- NodeJS apporte les modules
- Chaque fichier JS est un module
- Principes :
  - Zéro passe plat : traitement ou accès direct
  - KISS (Keep It Stupid Simple) : chaque module doit réaliser une fonctionnalité simple
  - Toute action doit être explicite

Modules et gestion de dépendances
Les modules

- Chaque fichier JS est un module
- Charger un module (synchrone)
  - Par ordre d'appel
  - Dépendances circulaires gérées
  - Appelé deux fois = chargé une fois en cache
  - **require**
  - `var fs = require('fs');`
  - `var monModule = require('./dir/monModule.js');`
- Publier une API
  - **exports**
  - ```
    exports.maFonction = function() {
        console.log('ça déchire');
    };
    ```
    ```

Modules et gestion de dépendances
Les modules : exemple

```
// MonApp.js
var fs = require('fs'); // core
var colors = require('colors'); // npm
var stringUtils = require('./stringUtils'); // local
fs.readFile('./participants.txt', function(err, data) {
    var upper = stringUtils.upperCase(data.toString());
    console.log(upper);
});

// stringUtils.js
exports.upperCase = function(string) {
    return string.toUppercase();
}
// ou module.exports = {upperCase : function …}
```

Modules et gestion de dépendances Core API

- Il existe un ensemble de modules noyau permettant de :
  - Accéder au système et aux processus
  - Manipuler le filesystem
  - Manipuler des interfaces réseau
  - Ces API sont écrites en C et dépendent de l'OS
- Elles sont asynchrone

http://nodejs.org/api

Modules et gestion de dépendances
Core API

- APIs importées de manière implicite :
  - module
  - console
    - console.log
    - console.error
  - process
    - process.exit (sortie propre)
    - process.abort (sortie brutale)
    - process.stdout
    - process.stderr
    - process.stdin

Modules et gestion de dépendances
Core API

- APIs standard :
  - os : informations sur le système
  - path : manipulation des paths
  - util : utilitaires (format, is*, inspect, ...)
  - fs : accès au système de fichier
    - fs.readFile, fs.writeFile
    - fs.rename, fs.chmod, fs.rmdir
  - net : accès réseau
  - http, https
  - dns
  - child_process
  - ...

Modules et gestion de dépendances NPM

- NPM : système de gestion des paquets
- En ligne de commande (basé sur NodeJS)
- Télécharge les paquets depuis npmjs.org
  - `$ npm install express`
  - `$ npm install grunt-cli -g`
  - `$ npm update`
  - `$ npm remove connect`
- Gère la description (package.json)
  - `$ npm init`
  - `$ npm docs`
  - `$ npm install mocha --save-dev`
  - `$ npm install bluebird --save`

Modules et gestion de dépendances NPM

- Les modules sont isolés
- Ils sont identifiés par leur fichier JS
- On peut charger un même module dans différentes versions
- On peut publier librement ses modules sur npmjs.org

- module1/
  - node_modules/
    - dep1/
      - node_modules/
        - subdep/ (1.0)
    - dep2/
      - node_modules/
        - subdep/ (1.5)

NodeJS et le Web
HTTP

- Créé pour faire du Web
- NodeJS n'est pas un serveur Web en lui même
- NodeJS est une plate-forme de développement
- programmer un serveur Web est simple :

```
var http  = require('http');

http.createServer(function(request, response) {
    res.writeHead(200, {'Content-Type' : 'text/plain'});
    res.end('hello world');
}).listen(8080);
```

NodeJS et le Web
HTTP

- APIs bas niveau :
  - `var request = http.request(options, callback);`
  - `var server = http.createServer(requestListener).listen(port);`
- Les requêtes et réponses sont des streams
- La requête ne part qu'avec :
  - `req.end();`
- Le module request propose une couche d'abstraction

NodeJS et le Web
HTTP

- NodeJS est mono-threadé
- Performant : Il fait majoritairement des I/O
- Délicat : une requête peut bloquer le serveur si il y a un traitement synchrone
  - Découpler les traitements en asynchrone
  - Monter un cluster
- Le module HTTP est de trop bas niveau
  - Le callback doit gérer
    - headers
    - paramètres
    - types
    - ...
  - director, connect, express, dispatch

NodeJS et le Web
Connect

- Framework HTTP pour NodeJS
- Extensible avec des middlewares
- Rarement utilisé directement

```
var connect = require( 'connect' );
var http = require( 'http' );

var app = connect()
     .use(connect.logger( 'dev' ))
     .use(connect.static( 'public' ))
     .use(function(req, res) { res.end( 'hello world' );});

http.createServer(app).listen(3000);
```

NodeJS et le Web
Connect

- Les middlewares
  - static : sert des ressources statiques
  - favicon : favicon du site
  - logger : trace les requêtes
  - query : décode une requête
  - errorHandler : gère un traitement d'erreur
  - directory : visualisation de dossiers
  - bodyParser : décode les corps de requêtes (JSON)
  - session : gestion des sessions
- Ne gère pas le routage !

NodeJS et le Web
Express

- Même développeur que Connect
- S'appuie sur Connect
- Intègre un système de routage
- Fournit un générateur d'application (express-generator)
  - `$ express -h`

```
var express = require('express');
var app = express();

app.get('/hello', function(req, res) {
    res.send('Hello World');
});
app.listen(3000);
```

NodeJS et le Web
Express

```
var express = require('express');
var logger = require('morgan');
var app = express();
/* Chargement des middlewares connect */
app.use(express.static('./public'));
app.use(logger());

/* Ajout des routes */
app.get('/hello', function(req, res) {
    res.send('Hello World');
});
app.listen(3000);
```

NodeJS et le Web
Express

Réagir aux requêtes :

- Sous forme de middleware :
  - `app.use(function(req, res, next) {});`
- Avec méthode HTTP et URL :
  - `app.get('/resource', function(req, res) {});`
- Réponde à toutes les méthodes :
  - `app.all('/resources', function(req, res) {});`
- Capturer des paramètres d'URL :

```
var callback = function(req, res) {
    /* res.params */
};

app.get('/resource/:id', callback);
app.get(\^\/resource\/(/d+)$/, callback);
```

NodeJS et le Web
Express

Le moteur de templates d'Express

```
app.set('views', __dirname + '/views');
app.set('view engine', 'jade');
res.render('index', {message : 'Hello World'});
```

- Jade :
  - Le choix par défaut
  - Notation sans balise avec indentation significative
  - Variables préfixées par =
- EJS
  - HTML classique
  - Syntaxe type JSP
    - <% operation %>
    - <%= expression %>

NodeJS et le Web
Express

La requête :

- request.params : les paramètres capturés dans l'url
- request.query : les paramètres de la query string
- request.body : le corps de la requête
  - parsé si on utilise bodyParser
- request.cookies : les cookies

La réponse :

- res.status(code) : code retour
- res.send(status, body) : envoi du contenu sans stream
- res.sendFile(path) : sert un fichier
- res.redirect(url)
- res.cookie(name, value)

Communication temps réel

- Le web temps réel = push
- Le serveur doit pouvoir contacter ses clients
- NodeJS n'utilise qu'un seul thread
- La technologie : Websockets
  - Bloquée par certains proxys
  - Disponible sur des navigateurs récents
  - Système de fallback :
    - Server Event Send
    - Flash Socket
    - Ajax long pooling
    - Forever IFrame
    - JSONP pooling

Communication temps réel Socket.IO

- Socket.IO
  - Une API simple
  - Existe depuis 2010
  - Fallbacks natifs et transparents
  - Librairie côté client
  - Concurrent de SockJS
- Étapes
  - Créer un serveur HTTP
  - Associer Socket.IO
  - Traiter l'événement de connexion
  - Une fois la socket établie
    - Emmètre des événements
    - Écouter des événements

socket.io

Communication temps réel Socket.IO

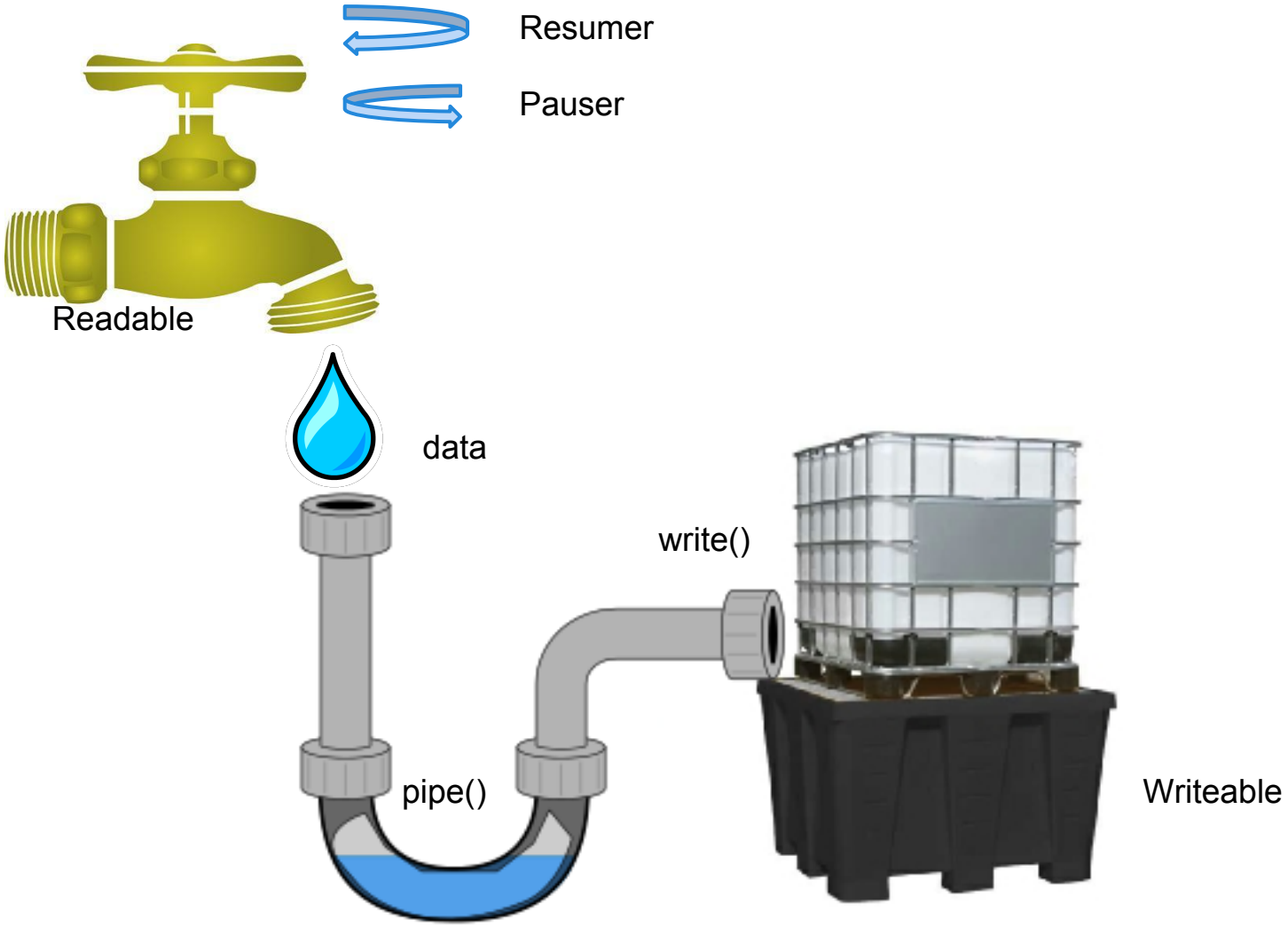
```
// Côté serveur
var io = require('socket.io');
io.listen(server);
// connexion d'un client
io.sockets.on('connection', function(socket) {});
// émission d'un message pour tous les clients
io.sockets.emit(name, content);
// émission d'un message pour un client
socket.emit(name, content);
// émission d'un message pour tous les clients sauf lui
socket.broadcast(name, content);
// écoute d'un message
socket.on(name, function(data) {});
```

```
// Côté client
// connexion au serveur
var socket = io.connect(url);
// écoute de la réception d'un message
socket.on(name, function(data) {});
// émission d'un message
socket.emit(name, content);
```

Gestion des streams

- Les streams sont centraux dans NodeJS
- On peut
  - Chaîner les streams : .pipe()
  - Transformer les streams
  - écouter leurs événements
- Sont des streams :
  - Les requêtes et réponses HTTP
  - Websockets
  - Les manipulations de fichiers
- Prototypes :
  - Readable, Writable, Duplex, Transform

Gestion des streams



Gestion des streams
Through2

- Through2 est une librairie de plus haut niveau pour gérer les streams
- Quelques outils :
  - Through2-map : applique une fonction à chaque chunk
  - Through2-filter : flitre les chunks
  - Through2-reduce : agrège les chunks
  - Through2-spy : espionne les chunks

Gestion des streams
Exemple avec Through2

```
var childProcess = require('child_process');
var through2 = require('through2');
var spawned = childProcess.spawn('ls', ['-al']);

var transform = through2(function(chunk, enc, done) {
  this.push(chunk.toString().toUppercase());
  done();
});

spawned.stdout
  .pipe(transform)
  .pipe(process.stdout);
```

Gestion des streams
Exemple avec Through2

```
var map = require('through2-map');
var through2 = require('through2');
var colors = require('colors');

var flowerPower = map(function(chunk) {
    return chunk.toString().rainbow;
});

process.stdin.pipe(flowerPower).pipe(process.stdout);
```

Gestion des streams

- Gulp est un outil de build JavaScript/NodeJS
- Alternative à Grunt
- Basé sur les streams
- Un plugin Gulp est un stream Transform

Persistance de données

- NodeJS peut communiquer avec des bases de données
- NodeJS peut travailler avec :
  - Oracle
  - MySQL,
  - MSSQL
  - PostgreSQL
  - MongoDB
  - Redis
  - Neo4J
  - CouchDB
  - …
- Des drivers sont disponibles sur npmjs.org

Persistance de données
Exemple : MySQL

```
var MySQL = require('MySQL');

var pool = MySQL.createPool({
    host     : 'localhost',
    user     : 'root',
    password : 'secret'
});

pool.query('SELECT * FROM users', function(err, rows, fields) {
    if(err) throw err;
    console.log(rows);
    pool.end();
});
```

Persistance de données
Exemple : MongoDB

```
var MongoClient = require( 'mongodb' ).MongoClient;

MongoClient.connect( 'mongodb://localhost:27017/madb' , function(err, db) {
    if(err) {return console.error(err);}
    var collection = db.collection( 'users' );
    var docs = [{_id : 1}, {_id: 2}, {_id: 3}];
    collection.insert(docs, {w:1}, function(err, result) {
        collections.find().toArray(function(err, items) {});
        var streams = collection.find({_id:{$ne 2}}).stream();
        stream.on( 'data' , function(item) {});
        stream.on( 'end' , function() {});
        collection.findOne({_id: 1}, function(err, item) {});
    });
});
```

Persistance de données
Couche d'abstraction

- dbtool : pour MySQL, MsSQL et Oracle
- jsdbc : pour MySQL, PostgreSQL, Oracle et SQLite et transactions
- Any-DB : modulaire
  - any-db-MySQL
  - any-db-postgre
  - any-db-SQLite3
  - any-db-transaction

Persistance de données
ORM

- Un ORM : Object Relationship Mapping
  - Gère les accès aux données à plus haut niveau
- Diverses solutions dans l'écosystème
  - ORM : MySQL, PostgreSQL, Amazon Redshift, SQLite
  - light-orm : MySQL, MsSQL, PostgreSQL, …
  - Sequelize : MySQL, PostgreSQL, SQLite, MariaDB
  - CORMO : MySQL, MongoDB, SQLite3, PostgreSQL
  - Model : PostgreSQL, MySQL, SQLite, Riak, MongoDB, LevelDB, In-memory, FileSystem
  - persist : SQLite3, MySQL, PostgreSQL, Oracle
  - streamsql : MySQL, SQLite3

Persistance de données Sequelize

```
var Sequelize = require('sequelize');
var sequelize = new Sequelize('madb', 'root', 'secret', {
     dialect : 'MySQL', // ou SQLite, postgres, mariadb
     port : 3306,
});
sequelize.autheticate()
     .complete(function(err) {
          if(err) throw err;
          console.log('connexion ok');
     });
var User = sequelize.define('User', {
     username: {
          Sequelize.STRING,
          allowNull : false
     },
     password : Sequelize.STRING
}, {});
```


```
sequelize.sync({force: true})
.complete(function(err) {
  if(err) throw err;
  User.create({
    username : 'Gilbert', password : 'Ranu'})
    .complete(function(err, user1) {
      if(err) throw err;
      User.find({username : 'Gilbert'})
      .complete(function(err, user2) {
        if(err) throw err;
        console.log(user1.value, user2.value);
      })
    })
  })
});
```

Persistance de données Mongoose

```
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017:madb');

var User = mongoose.model('User', {
  username : String,
  password : String
});

var user = new User({username : 'Gilbert', password : 'Ranu'});
user.save(function(err, user) {
  if(err) throw err;
  console.log(user);
});

User.find(function(err, users) {
    console.log(users);
});
```

mongoose

elegant mongodb object modeling for node.js

Forge
IDE

- Différents “IDE”
  - NotePad++, Geany, SublimeText
  - WebStorm/IntelliJ, Netbeans, Eclipse avec JSDT, Cloud9
  - Brackets
- Debugger
  - Utilisation de l’IDE
  - Debugger de NodeJS
    - `$ node debug server.js`
    - `repl, next, cont, step, out, ...`
  - Node inspector
    - `$ npm install -g node-inspector`
    - `$ node-debug server.js`

Forge
Tests unitaires

- Mocha - le plus répandu
- Chai Assertion Library
  - Should, Expect, Assert
- Sinon.js
  - des mocks
  - des stubes (implémentations retournant toujours la même chose
  - des spies (tracer les appels aux méthodes)

Sinon.JS

Forge
Tests unitaires : Exemple Mocha


```
var assert = require('assert');
describe('Test avec Mocha', function() {
  var rh;
  beforeEach(function() {
    // Avant chaque test
    rh = new RequestHandler();
  });
  // premier test
  it('should handle the request and return a number between 0 and 1' ,
function() {
    var result = rh.handleRequest();
    assert(result < 1 && result >=0);
  });
});

// $mocha test.js
//-> 1 passing (4ms)
```

Forge
Nodemon

Surveille les fichiers source

Relance l'appli à chaque changement

```
$ npm install -g nodemon
$ nodemon server.js
```

NodeJS en mode Cluster

- NodeJS étant mono-threadé, il utilise un coeur de processeur
- Il faut lancer autant de threads que de coeurs
- Il faut un système de gestion du cluster
- Il faut que les threads partagent les mêmes ressources
- Il existe un module Cluster
  - cluster.setupMaster(config)
    - config principale
    - appelé une seule fois
    - configure les workers
  - cluster.fork(env)
    - démarre un nouveau worker
    - un sous processus

NodeJS en mode Cluster Exemple

```
var cluster = require( 'cluster' );
var cpuCount = require( 'os' ).cpus().length;

if(cluster.isMaster) {
    for(var i = 0; i < cpuCount; i++) {
        cluster.fork();
    }
} else {
    var http = require( 'http' );
    server = http.createServer(function(req, res) {
        res.writeHead(200);
        res.end( 'Hello World' );
    });
    server.listen(3000);
}
```

NodeJS en mode Cluster

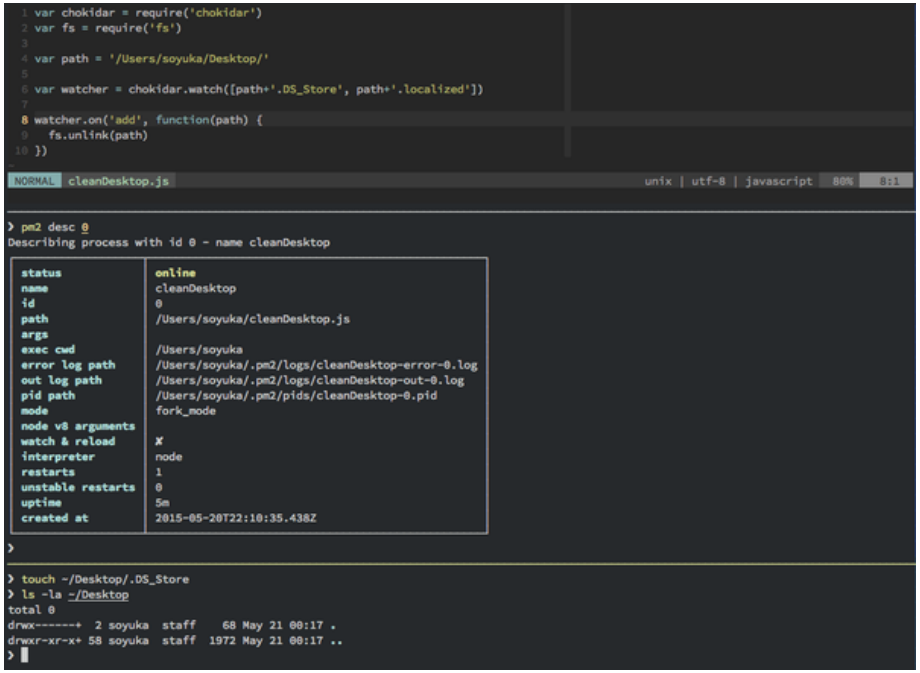
- worker.send(message[, sendHandle])
  - Envoie un message depuis le master vers un worker
  - C’est un alias de childProcess.send
  - sendHandle peut être un serveur TCP
- process.send(message)
  - Envoie un message au processus parent
  - donc du worker au master
- process.on(‘message’, callback)
  - Écoute un message IPC (master et worker)

Au delà de NodeJS
MEAN

- Stack JavaScript
- MongoDB + Express + AngularJS + Node.js
  - Passport (oAuth, SSO, etc …)
  - Mongoose
- http://mean.io
- http://meanjs.org

Au delà de NodeJS Monitoring

- AppDynamics
- NewRelic
- PM2
  - $ npm install -g pm2
  - monitore un cluster



Prêts à construire un nouveau monde?