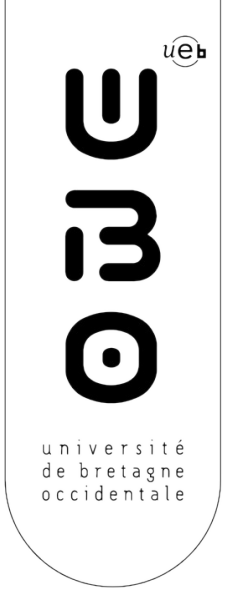
Université de Bretagne Occidentale
2013



Node.js

UFR Sciences et Techniques

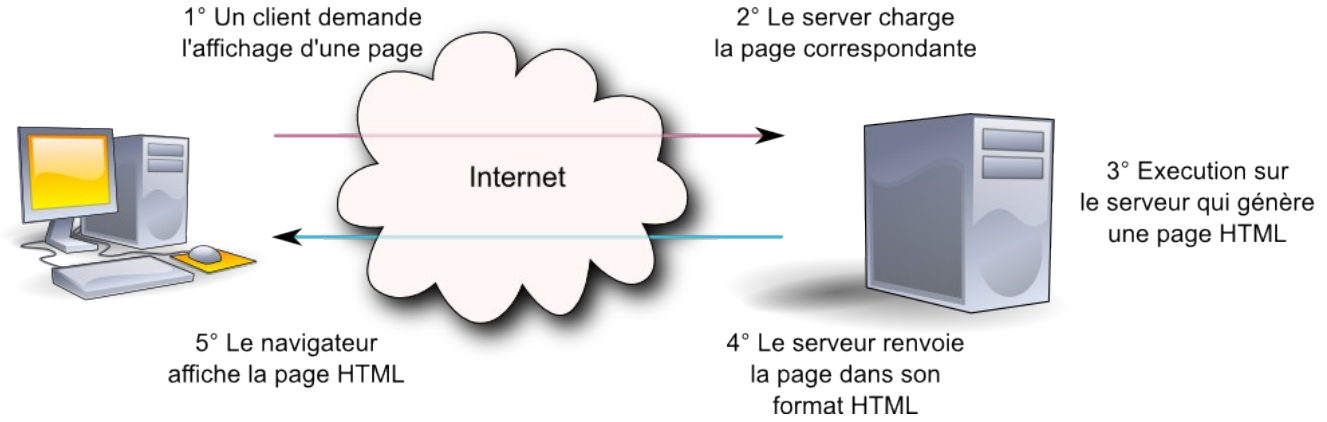
Guillaume Bourel
bourel@univ-brest.fr

Node.js ?

- Présentation
  - Qu'est ce que c'est ?
  - Pourquoi est-ce qu'on en parle maintenant ?
- Comment ça marche ?
  - Les « mécanismes »
- Comment l'utiliser ?
  - NPM
  - Des librairies et des méthodes

Qu'est ce que c'est ?

- Du Javascript **coté serveur** !
  - Exécuté en dehors du navigateur
  - Utilise le moteur de Google Chrome : V8
- Un ensemble de librairies



Qu'est ce que c'est ?

- Créé par Ryan Dahl en 2009
  - A la base l'idée était un serveur permettant le *push*
  - Open source
  - Maintenance assurée par Joyent
- Documentation http://nodejs.org/
  - Version actuelle v0.10.20
- Une présentation par Ryan Dahl :
  - http://www.youtube.com/watch?v=jo_B4LTHi3I

Pourquoi l'utiliser ?

- ✔ Un langage commun pour le client et le serveur :-)
- ✔ Un projet très actif et à succès
- ✗ Un projet jeune
- ✗ Des choix controversés

« *Node is no freaking unicorn that will come and do your work for you, sorry.* »

Felix Geisendörfer

- Avez-vous déjà écrit un serveur web ?
  - Comment faites vous ?

Un serveur web « classique »

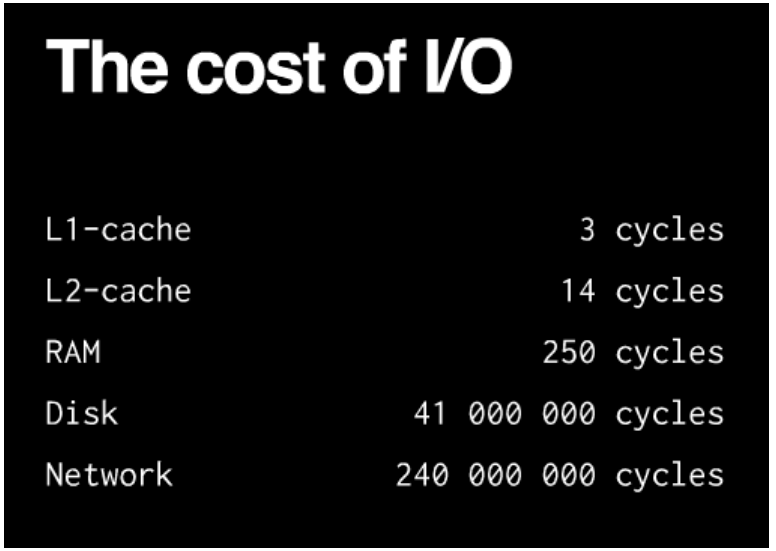
- Le serveur est chargé de renvoyer la page HTML à partir de la demande (URL) du client.
  - Dans le cas du web statique la page est renvoyée directement
  - Dans le cas du web dynamique la page est renvoyée après avoir été générée (par du PHP, Java, Ruby ou autre)
- La solution classique : …

Il était une fois...

1. Au millénaire dernier, l'apparition du Javascript
2. Début du siècle présent, l'enrichissement des fonctionnalités
   - Web 2.0
   - Librairies Javascript comme JQuery, etc.
3. Cette décennie, le Javascript devient très répandu et des solutions d'optimisations apparaissent

Les mécanismes de Node.js

- Après ces différentes évolutions et pour répondre à de nouvelles demandes, de nouveau modes de fonctionnements les principes de Node.js ont été pensés différemment



Un peu plus qu'un interpréteur

- De manière basique, Node peut être utilisé comme un interpréteur
  - Un fichier source « essai.js » :

```
console.log("Bonjour le monde");
```

- Peut être lancé avec :

```
node essai.js
```

- Et affiche

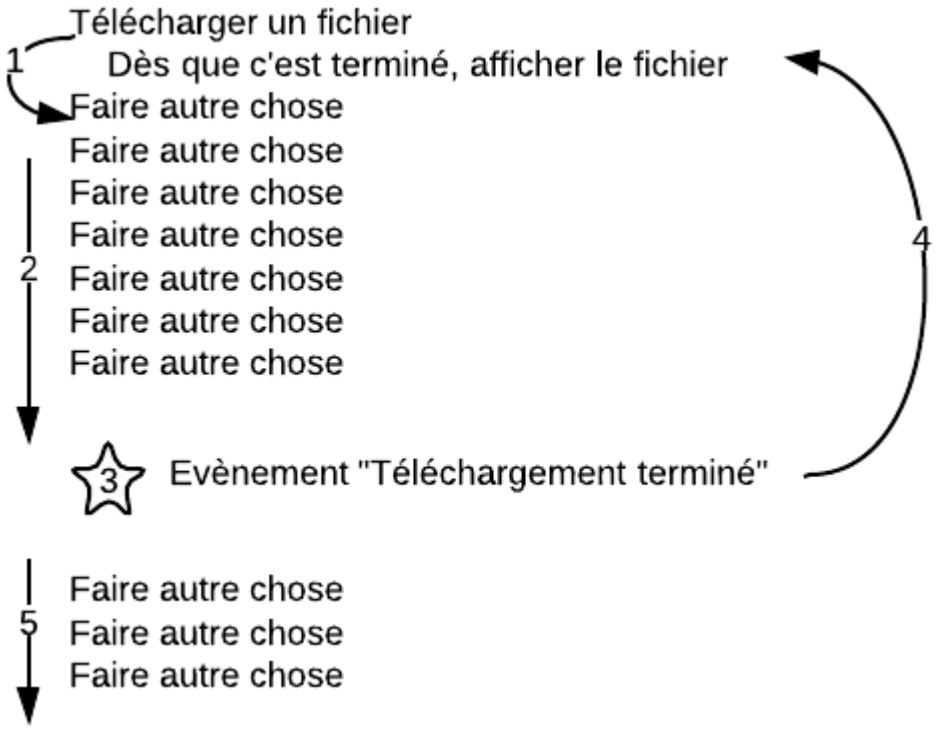
```
Bonjour le monde
```

Un seul flot d'exécution

- Mais Node.js est « mono-thread »
  - *Attention, ce n'est pas vraiment vrai !*
- La solution multi-thread n'est pas utilisée
- Au profit d'une solution basées sur les événements
  - Une solution bien adaptée au Javascript
  - Voir V8
- Mais un serveur écrit en C++

Le principe des événements

- Node utilise le principe Javascript des **événements**



http://fr.openclassrooms.com

Le principe des événements

```
var callback = function (error, response, body) {
    console.log("Fichier téléchargé !");
};

request('http://site.com/fichier.zip', callback);
console.log("Faire autre chose");
console.log("Faire autre chose");
console.log("Faire autre chose");
…
```

- => même principe que le Javascript coté client

Un serveur web à la mode de chez Node

- Un module existe pour le HTTP

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

var server = http.createServer(function(req, res) {
    res.writeHead(200, {"Content-Type": "text/plain"});
    res.end('Bonjour le monde HTTP !');
});
server.listen(3000);
```

Les modules

- Comme Javascript ne fournit pas d'espaces de nommage (namespaces) le principe des modules est utilisé
- Chaque fichier correspond à un module indépendant
- Un module peut être importé avec la fonction « require » et le nom du fichier
  - Si le fichier n'est pas trouvé directement, il est recherché avec .js, .json et .node
  - Un module non natif sans chemin relatif est recherché dans le répertoire node_modules (ou un répertoire du même nom dans les répertoires parents)

Les modules

- Toutes les variables et fonctions d'un module sont « privées » et uniquement utilisables dans ce module
- Sauf si ajoutées à l'objet « exports »

```
var PI = Math.PI;

exports.aire = function (r) {
     return PI * r * r;
};

exports.perimetre = function (r) {
     return 2 * PI * r;
};
```

Exemple de module

- Pour émettre des événements :

```
var EventEmitter = require('events').EventEmitter;

var monEmetteur = new EventEmitter();
monEmetteur.emit('grand', 'Un grand événement !');
```

Un exemple de serveur HTML

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

var server = http.createServer(function(req, res) {
    res.writeHead(200, {"Content-Type": "text/html"});
    res.write('<!DOCTYPE html>'+
'<html>'+
'    <head>'+
'        <meta charset="utf-8" />'+
'        <title>Node.js</title>'+
'    </head>'+
'    <body>'+
'       <p>Bonjour le monde du HTML !</p>'+
'    </body>'+
'</html>');
    res.end();
});

server.listen(8080);
```

Pas d'accès à l'arbre DOM

- Puisque le Javascript n'est pas exécuté dans le navigateur, pas d'accès à l'arbre DOM
- …
- Tout du moins par défaut, par exemple le module jsdom ou le navigateur headless PhantomJS permettent le chargement du DOM

Chacun sa route, chacun son chemin

- Avec le serveur HTTP précédent peut importe l'URL la réponse est toujours la même
- Il serait intéressant de connaître la page et les paramètres

  http://monsite.com/ma/page?param=8

- Le module => url (http://nodejs.org/api/url.html)

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

var server = http.createServer(function(req, res) {
     var chemin = url.parse(req.url).pathname;
     console.log(chemin);
     …
}
```

- Querystring pour les paramètres (http://nodejs.org/api/querystring.html)

NPM

- Node Packaged Modules
  - http://npmjs.org/
- Node permet la création d'un module à partir d'un répertoire
  - Il faut y créer un fichier package.json
- Quelques commandes
  - npm search unModule
  - npm install unAutreModule
  - npm install -g unModuleGlobal
  - npm update

Exemple de fichier package.json

```
{
  "name": "MonProjet",
  "version": "0.1.0",
  "description": "Mon Projet A Moi",
  "author": "G.Bourel",
  "destName": "monProjet",

  "repository": {
    "type": "git",
    "url": "git@monsite.com:MonProjet"
  },

  "dependencies": {
    "express": "3.x",
    "underscore": "~1.5.1"
  },

  "devDependencies": {
    "grunt": "~0.4.1",
    "grunt-contrib-jshint": "~0.5.3"
  },

  "license": "GPL"

}
```

Comment utiliser Node.js ?

- Node.js est écrit en C++
- Le but est d'avoir un serveur simple et efficace
- L'utilisation basique est assez fastidieuse et complexe
- Mais une utilisation avec les bons outils est particulièrement efficace

Des fonctions spécifiques

- Les fonctions de Node enrichissent celles de Javascript
- Il est notamment possible d'accéder au système de fichier, grâce au module « fs »
  - Permet la lecture et l'écriture
- Accès à des fonctions systèmes élaborés
  - watchFile : permet d'exécuté une action lorsqu'un fichier est modifié

Exemple d'utilisation de fs

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

fs.mkdir('./testDir', 0777, function(err) {
     if (err)
          throw err;
     fs.writeFile('./testDir/fichier.txt', 'Salut Fichier',
function(err) {
          if (err)
               throw err;
          fs.readFile('./testDir/fichier.txt', 'UTF-8',
function(err, data) {
               if (err)
                    throw err;
               console.log(data);
          });
     });

});
```

Express.js

- Express est un module répandu qui fourni un ensemble d'éléments pour faciliter le développement d'une application web
- C'est un exemple d'un petit framework utilisable avec Node.js
- Pour l'installer soit utiliser package.json, soit
  npm install express

Organiser les routes

- Afin de répondre aux différentes routes (ie. Les différents chemins demandés) Express propose un mécanisme de callback simplifié

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

var app = express();

app.get('/', function(req, res) {
    res.setHeader('Content-Type', 'text/plain');
    res.end('Page d\'accueil');
});

app.get('/apropos', function(req, res) {
    res.setHeader('Content-Type', 'text/plain');
    res.end('A propos de cette page');
});
```

Les paramètres

- Il est aussi possible de définir des chemins contenant des « paramètres » soit directement dans le chemin, soit des paramètres HTML

```
app.get('/utilisateur/:nom/affiche', function(req, res) {
    res.setHeader('Content-Type', 'text/plain');
    res.end('Utilisateur' + req.params.nom);
});
```

Les vues

- Produire du HTML avec du Javascript n'est pas idéal
- Pour résoudre ce problème Express propose l'utilisation de vues
- Ces vues sont des fichiers *templates* qui contiennent la page HTML et des éléments à remplacer
- Il existe de nombreux *moteurs de template*

Embedded JavaScript

- EJS est un système de template simple et répandu
- Le principe est de placer du Javascript dans du HTML :

```
<h3>Bonjour <%= nom %></h3>
```

- Appelé par :

```
app.get('/utilisateur/:nom/affiche', function(req, res) {
      res.render('user.ejs', {nom: req.params.nom});
});
```

- Comme en JSP
  - < % …%> permet l'exécution
  - <%= …%> permet l'affichage

Pour EJS dans Express

- Express permet d'utiliser différents systèmes de template

- Pour utiliser EJS il faut l'initialiser

```
app.set('views', __dirname + '/views');
app.set('view engine', 'ejs');
```

Activation de fonctionnalités

- Express permet d'activer ou non un grand nombre de fonctionnalité
- Le principe des middlewares est d'ajouter des fonctionnalités au noyau d'Express
- Par exemple :

```
app.use(app.router);
app.use(express.static(path.join(__dirname, 'public')));
```

Des liens intéressants

- The Node Beginner Book
  - http://www.nodebeginner.org/
  - En français : http://nodejs.developpez.com/tutoriels/javascript/node-js-livre-debutant/
- Un tutoriel intéressant :
  - http://fr.openclassrooms.com/informatique/cours/des-applications-ultra-rapides-a
- Mastering Node.js : http://visionmedia.github.io/masteringnode/
- Sources http://github.com/joyent/node
- Podcast http://nodeup.com/
- http://nodejs.org/