

8. Intro à Node.js

Jean-Luc Falcone

Août 2019

Histoire

- Suite à la "Second Browsers War" (2005-2010), les interpréteurs JS deviennent particulièrement rapides.
- En 2009, Ryan Dahl propose Node.js en combinant:
 - V8: le moteur JS de Chrome
 - Une API pour interagir avec les IO (réseau et système de fichier)
 - Une boucle d'événement

Modules Node.js

- L'organisation du code en module permet de travailler avec des petites unités cohérentes et composables.
- Seulement certaines fonctions sont visibles depuis les autres modules.
- Conduit à séparer son code en plusieurs fichiers.
- ES6 introduit un système de module différent mais pas encore bien implanté.

Modules exemple

Dans un fichier 'stats.js'

```
function sum( xs ) {  
  return xs.reduce( (x,y)=>x+y, 0 );  
}
```

```
function avg( xs ) {  
  return sum(xs) / xs.length;  
}
```

```
exports.average = avg;  
exports.defaultConfidence = 0.05;
```

Modules exemple

Dans un fichier 'stats.js'

```
function sum( xs ) {  
  return xs.reduce( (x,y)=>x+y, 0 );  
}
```

```
function avg( xs ) {  
  return sum(xs) / xs.length;  
}
```

```
exports.average = avg;  
exports.defaultConfidence = 0.05;
```

Dans un autre fichier

```
const stats = require("./stats.js");  
let s = stats.average( [1,2,3] );  
let alpha = stats.defaultConfidence;
```

Node Packet Manager (NPM)

Node.js utilise le gestionnaire de dépendances NPM. On peut aisément l'utiliser pour télécharger et installer une librairie dans un projet Node.js:

```
$ npm install twitter #installe les APIs twitter
```

On peut ensuite utiliser le module dans le projet:

```
const Twitter = require("twitter");
```

```
let twitterClient = new Twitter( /* params */ );
```

Autres fonctionnalités

- Configurer un projet `npm init`
- Chercher un packet `npm search <name>`
- Lister les packets installés `npm list`
- Trouver les mises à jour `npm outdated`
- Installer tous les packets d'un projet `npm install`

IO: Fichiers Sync

```
let data = fs.readFileSync("/tmp/in.txt", "utf8");  
fs.writeFileSync("/tmp/out.txt", data, "utf8")
```


IO: Fichiers Sync avec les erreurs

```
let data;
try{
  data = fs.readFileSync("/tmp/in.txt", "utf8");
} catch(err) {
  if(err !== null ) {
    console.error("Could not open file: " + err);
    return;
  }
}
try {
  fs.writeFileSync("/tmp/out.txt", data, "utf8");
} catch(err) {
  if(err !== null ) {
    console.error("Could not open file: " + err);
    return;
  }
}
```

IO: Fichiers Async

```
fs.readFile("/tmp/init.txt", "utf8", function(err, data) {  
  if(err !== null) {  
    console.error("Could not open file: " + err);  
    return;  
  }  
  fs.writeFile("/tmp/out.txt", data, function(err) {  
    if(err !== null) {  
      console.error("Could not write file: " + err);  
    }  
  });  
});
```

Serveur HTTP Basic

```
const http = require("http");

http.createServer(function(request, response) {
  console.log( "REQUESTED: " + request.url );
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello, World !");
  response.end();
}).listen(80);
```

Automatisation de tâches: scripts en JS

```
#!/usr/bin/env node
```

```
"use strict";
```

```
console.log( "CPU architecture:", process.arch );  
console.log( "Platform:", process.platform );
```

Event Loop

- Les performances de `node.js` dépendent d'une boucle unique.
- Il ne faut pas la bloquer, pour éviter le déni de service.
 - Attention à la complexité des calls-backs
 - Utiliser le `worker pool`
 - Utiliser des variantes `safe`

Worker Pool

Le worker pool est utilisé de base pour:

DNS `dns.lookup()`, `dns.lookupService()`

File System Toutes sauf `fs.FSWatcher()` et celles explicitement synchrones.

Crypto `crypto.pbkdf2()`, `crypto.randomBytes()`,
`crypto.randomFill()`

Zlib Toutes les API zlib, sauf celles explicitement synchrones.

Possibilité d'écrire des extensions en C++ pour profiter du worker pool: <https://github.com/nodejs/nan>

Problèmes typiques

- Regex** Eviter les regex compliquées (ou utiliser un package comme `safe-regex`).
- JSON** `JSON.parse` et `JSON.stringify`. Peuvent devenir lentes avec des grandes entrées (cf. `JSONStream`)

Sans partitionnement: $O(n)$

```
for (let i = 0; i < n; i++)  
  sum += i;  
let avg = sum / n;  
console.log('avg: ' + avg);
```


Avec Partitionement: $O(1)$

```
asyncAvg(n, function(avg){  
  console.log('avg of 1-n: ' + avg);  
});
```

Avec Partitionement: $O(1)$

```
function asyncAvg(n, avgCB) {  
  let sum = 0;  
  function help(i, cb) {  
    sum += i;  
    if (i == n) {  
      cb(sum);  
      return;  
    }  
    setImmediate(help.bind(null, i+1, cb));  
  }  
  help(1, function(sum){  
    let avg = sum/n;  
    avgCB(avg);  
  });  
}
```