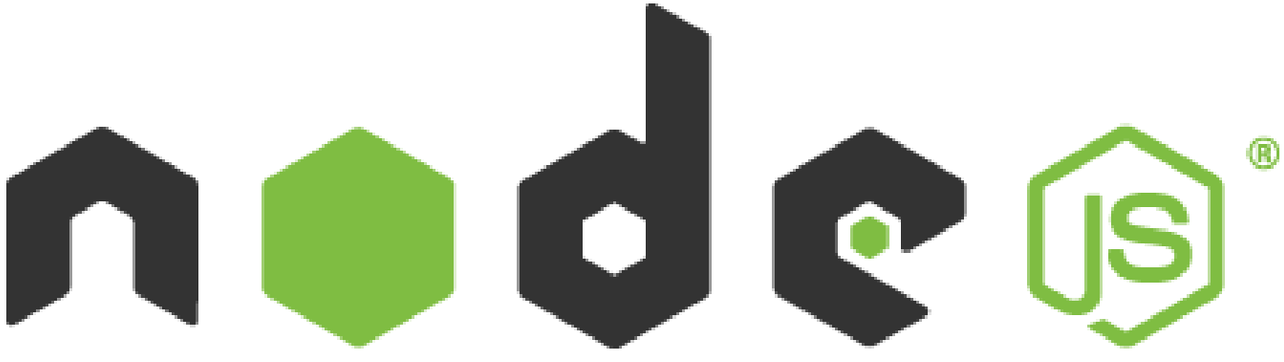


Node.js

Alvin Berthelot

Version 1.0.0

Ce(tte) œuvre est mise à disposition selon les termes de la Licence Creative Commons.

Attribution - Partage dans les Mêmes Conditions 3.0 non transposé.

La licence, ses explications ainsi que les moyens de contribution et réappropriation sont détaillés à la fin.

Introduction à Node.js

Node.js, c'est quoi ?

Node.js est une plateforme disposant d'une API JavaScript pouvant s'executer sur de nombreux environnements.

De ce fait il est possible d'exécuter du code JavaScript non plus uniquement dans un navigateur web (front-end) comme nous avions l'habitude, mais également côté serveur (back-end).

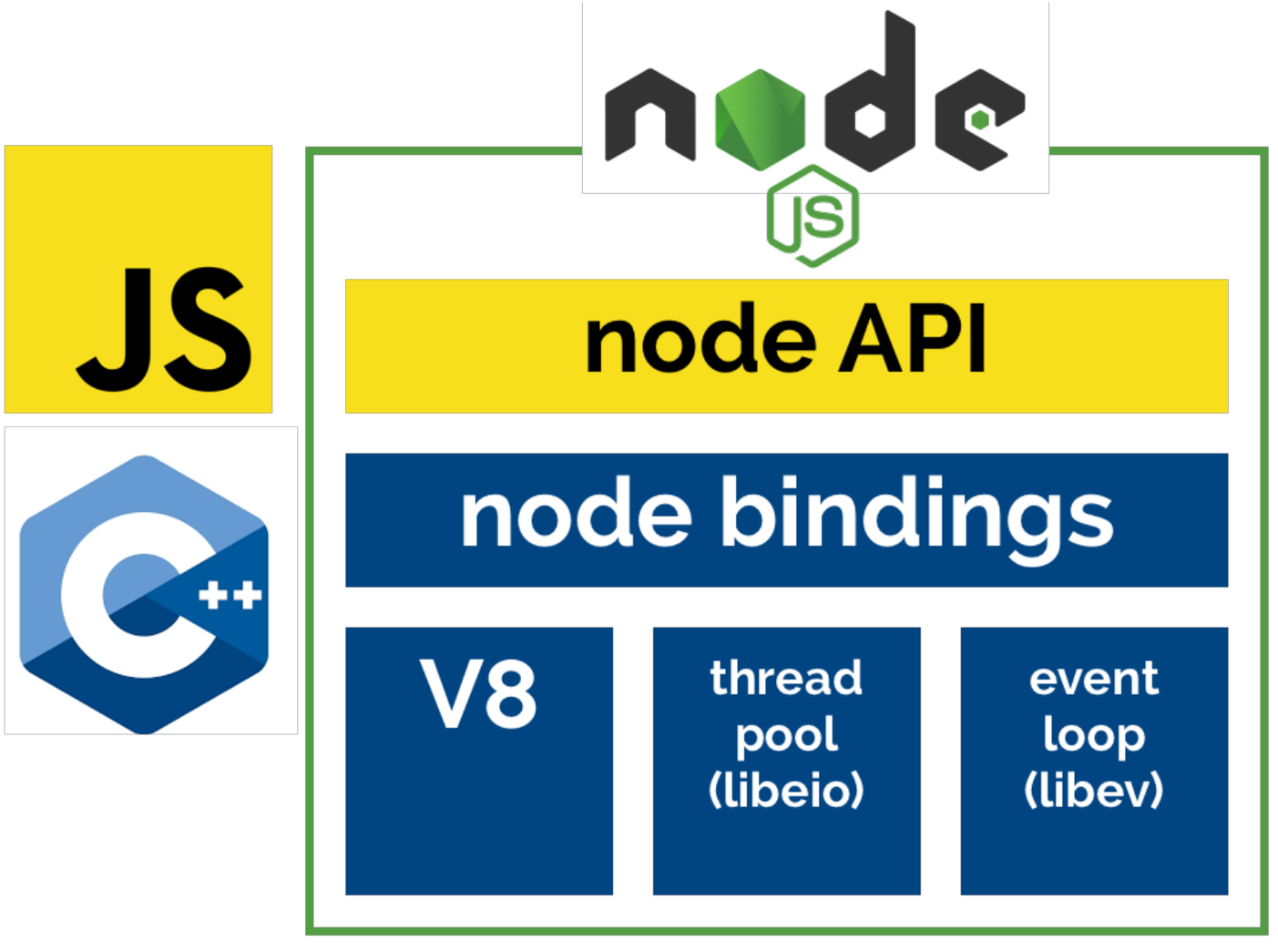
On résume généralement Node.js à un serveur back-end JavaScript, c'est assez réducteur comme définition car si c'est un moyen d'utiliser cette plateforme, elle permet de nombreuses autres possibilités.

Node.js, pourquoi ?

Node.js est capable d'accéder aux ressources système ainsi qu'aux connexions réseau de l'environnement dans lequel il s'execute. En clair, une fois installé sur un environnement il est capable de faire beaucoup de choses :

- Créer un serveur HTTP.
- Automatiser des tâches en éditant des fichiers, le cas typique de la contruction de livrables pour le Web.
- Exécuter des tests.
- Contrôler un IoT "Internet of Things".

Node.js, comment ?



Qui maintient Node.js ?

Initialement l'entreprise Joyent après avoir embauché Ryan Dahl le créateur de Node.js.

L'éco-sytème Node.js est désormais porté depuis 2015 par la fondation non-commerciale Node.js Foundation.

Utilisation basique de Node.js

Installer Node.js

Il existe différentes options pour installer Node.js dans un environnement, les plus populaires étant :

- Télécharger le binaire pour l'environnement concerné et l'installer.
- Utiliser nvm (Node Version Manager) pour installer les versions de Node.js souhaitées.
- Utiliser Docker et récupérer une image officielle Node.js.

Ensuite il convient de vérifier la bonne installation.

```
node -v
```

Exécuter un programme avec Node.js

Une fois installé, exécuter un programme avec Node.js est trivial. Soit en utilisant l'interpréteur en lignes de commandes.

```
node

> console.log('Hello World !');
```

Soit en exécutant un programme depuis un fichier.

```
// index.js
console.log('Hello World !');
```

```
node index.js
```

Arguments d'exécution

Il est possible de passer des arguments en exécutant un programme dans Node.js.

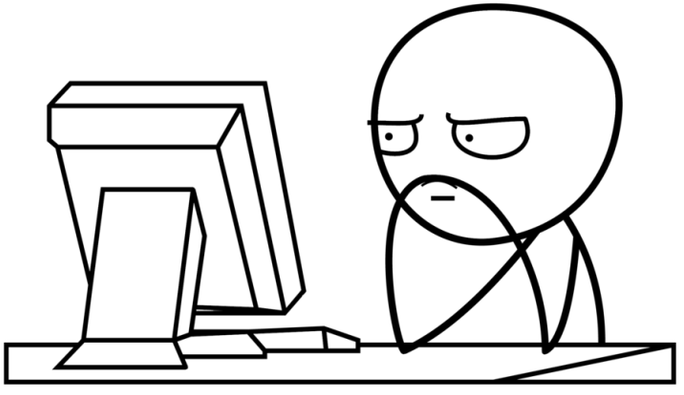
```
node index.js Hello you
```

Ces arguments seront stockés dans la propriété `argv` (tableau de valeurs) du processus `process` accessible depuis n'importe quelle exécution.

```
// index.js
process.argv.forEach(value => { console.log(value) });

// /Users/webyousoon/.nvm/versions/node/v6.9.2/bin/node
// /Users/webyousoon/work/workspace/samples/index.js
// Hello
// you
```

2 arguments sont passés par défaut : le binaire d'exécution et le fichier exécuté.



Exercices

On va s'appuyer sur la plateforme NodeSchool.io, pour cela vous devez installer le TP "learnyounode".

⇒ HELLO WORLD
⇒ BABY STEPS

"Event loop" de Node.js

JavaScript un langage "Single thread"

JavaScript est un langage de programmation qui exécute des opérations selon un fonctionnment "Single thread", cela signifie qu'un seul et unique processus est instancié pour réaliser l'ensemble des opérations.

A contrario, les langages Java et C sont des langages de programmation "Multi thread", cela signifie que plusieurs processus sont instanciés pour réaliser l'ensemble des opérations.

"Blocking" Vs "Non-Blocking"

A priori le fonctionnement du langage JavaScript semble moins performant car il permet de réaliser moins d'opérations dans un même laps de temps du fait qu'un seul processus soit en mesure de les réaliser.

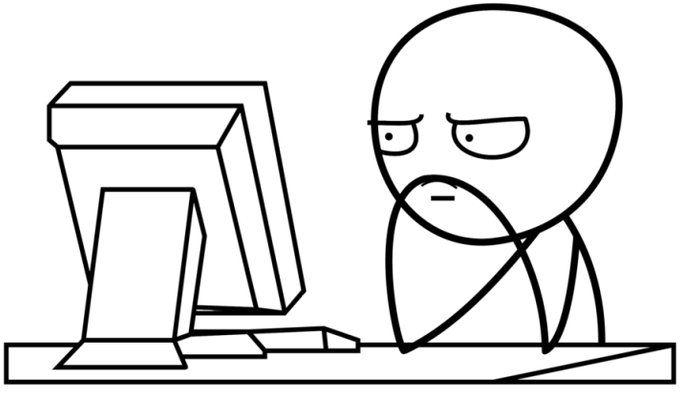
Ce raisonnement n'est vrai que si les opérations à effectuer sont "bloquantes" (Blocking) c'est à dire que l'on attend systématiquement que les opérations soient indiquées comme terminées afin d'en lancer une nouvelle.

Or JavaScript gère la concurrence des opérations grâce à une "boucle d'événements" qui est "non bloquantes" (Non-Blocking) ; c'est à dire qu'elle lance des opérations les unes à la suite des autres sans pour autant attendre qu'elles soient indiquées comme terminées.

Le résultat d'une opération peut-être récupéré selon un fonctionnement de callback et le traitement de ce résultat est lui même ajouté à la "boucle d'événements".

"Blocking" Vs "Non-Blocking"

D'une certaine manière le moteur JavaScript délègue les traitements ; qui ne sont pas de sa responsabilité ; à d'autres processus et récupère le résultat des traitements lorsque ceux-ci sont disponibles.



Exercices

On va s'appuyer sur la plateforme NodeSchool.io, pour cela vous devez installer le TP "learnyounode".

⇒ MY FIRST I/O!
⇒ MY FIRST ASYNC I/O!

Utilisation des modules

Le "scope" global

Rappel

JavaScript dispose d'un "scope" global, il s'agit d'un `object` disposant déjà d'un certain nombre de propriétés et de méthodes.

Dans un navigateur

```
console.log(window);
console.log(this);
```

C'est également vrai pour Node.js qui a un "scope" global se nommant `global`.

Dans Node.js

```
console.log(global);
console.log(this);
```

Modules en JavaScript

Dans Node.js (comme dans un navigateur) il faut limiter l'utilisation du "scope" global pour éviter les risques de collision et optimiser l'usage de la mémoire.

```
console.log(global.newSideEffect);
newSideEffect = 42;
console.log(global.newSideEffect);
```

Rappel
Le fonctionnement des "closures" et des IIFE permettent l'exposition publique vs privée de propriétés d'un objet.

Sur le même principe d'isolement de "scope" que les IIFE, Node.js dispose d'un système de modules natif qui permet de limiter la portée des "scopes" tout en laissant la possibilité d'exporter des états, des fonctions, etc.

Il s'avère qu'il existe plusieurs système de modules, Node.js a commencé avec (et peut toujours utiliser) la syntaxe CommonJS.

CommonJS

CommonJS peut être perçu comme une API pour normaliser et simplifier l'utilisation de modules dans les environnements JavaScript. Elle est implémentée par défaut au sein de Node.js.

On peut ainsi facilement exporter depuis un autre fichier un `object` pouvant contenir des valeurs en JavaScript avec le mot clé `exports`.

```
// mathTools.js
exports.add = (a, b) => a + b;
```

Puis importer cette valeur depuis le fichier avec le mot clé `require`.

```
// program.js
require('./mathTools').add(4, 7); // 11
```

L'extension du fichier ".js" est facultative pour les imports.

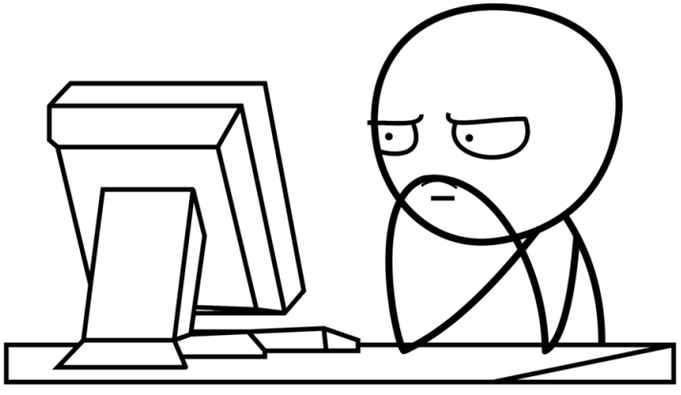
Modules présents par défaut

Certains modules sont présents nativement au sein de Node.js, les principaux étant :

- fs : Module "File System" pour créer / modifier / supprimer des fichiers
- net : Module pour utiliser les protocoles TCP ou IPC
- http : Module pour utiliser le protocole HTTP
- path : Module pour travailler sur les chemins des fichiers / répertoires
- events : Module pour appliquer la programmation évenementielle

```
const http = require('http')

http.get('http://numbersapi.com/42', (res) => {
  res.setEncoding('utf8')
  res.on('data', fact => console.log(fact))
});
```



Exercices

On va s'appuyer sur la plateforme NodeSchool.io, pour cela vous devez installer le TP "learnyounode".

⇒ FILTERED LS
⇒ MAKE IT MODULAR
⇒ HTTP CLIENT

Utilisation de npm

npm le compagnon associé

Une fois Node.js installé, celui-ci ne vient pas seul, il inclut par défaut npm qui est son **gestionnaire de modules**.

```
npm -v
```

Ce gestionnaire de modules permet d'installer et de publier facilement des modules (signifiant "package" en JavaScript) pour Node.js : serveurs HTTP, frameworks, drivers, processeurs, etc.

npm un gestionnaire ET un dépôt de modules

npm est à la fois lui même un module dédié pour Node.js (servant de gestionnaire) et le registre de modules accessible via Internet.

Sa popularité et son fonctionnement avec Node.js en ont fait le registre pour tout l'écosystème JavaScript désormais.

Installer un module via npm

Dans une invite de commandes, l'instruction `npm install` permet d'installer un module. Il existe 2 manières d'installer un module, soit de manière locale au projet (option par défaut), soit de manière globale (avec l'option `-g`).

```
// local
npm install <module_name>

// global
npm install -g <module_name>
```

Dans le premier cas, le module sera installé depuis le répertoire courant où l'instruction a été lancée. Il ne sera donc accessible que depuis ce répertoire courant.

Dans le deuxième cas, le module sera installé dans un répertoire commun lié à l'installation de Node. Il sera donc accessible depuis n'importe quel répertoire de l'environnement.

Installer tous les modules

Lors d'installations locales de modules, npm va rapatrier ceux-ci dans un répertoire `node_modules`.

Pour installer au pré-requis tous les modules nécessaires au bon fonctionnement d'un projet, npm va s'appuyer sur le fichier `package.json` qui liste les dépendances (et leurs versions associées) avec `dependencies`, `devDependencies`, `optionnalDependencies` et `peerDependencies`.

Il suffit ensuite d'executer `npm install` dans le répertoire où se situe le fichier `package.json` pour installer toutes les dépendances listées dans le répertoire `node_modules`.

```
npm install
```

Le fichier `package.json`

Le fichier `package.json` est le point d'entrée de tout projet/module devant s'exécuter dans Node.js, il contient principalement :

- Les informations relatives au projet : `name`, `description`, `version`, `license`, `contributors`, etc.
- Les dépendances : `dependencies`, `devDependencies`, `optionnalDependencies` et `peerDependencies`.
- Les exécutions possibles avec `scripts`.
- Les contraintes de compatibilité Node.js avec `engines`.

Il existe beaucoup d'autres informations que l'on peut renseigner dans ce fichier, la documentation officielle exhaustive à ce sujet.

Un exemple de `package.json`

```
{
  "name": "colourfull-api",
  "version": "1.0.5",
  "description": "Web API to deliver palettes of colours.",
  "dependencies": {
    "body-parser": "1.5.2",
    "cors": "2.8.1",
    "express": "4.2.0"
  },
  "devDependencies": {
    "expect.js": "0.3.1",
    "mocha": "1.21.0",
    "superagent": "0.18.2"
  },
  "scripts": {
    "start": "node index.js",
    "test": "mocha --timeout 15000 test/"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/my-company/colourfull-api"
  },
  "license": "MIT license"
}
```

Enrichir un fichier `package.json`

La commande `npm init` permet de générer facilement un fichier `package.son`.

Une fois le fichier `package.json` créé il conviendra de le mettre à jour régulièrement, notamment sur les dépendances nécessaires au projet. Des options sont disponibles en ce sens.

```
// depencies
npm install --save <module_name>

// devDepencies
npm install --save-dev <module_name>

// optionnalDepencies
npm install --save-optionnal <module_name>
```

(Bien) enrichir un fichier `package.json`

Par défaut les commandes `npm install` ne gèlent pas les dépendances de manière stricte, mais en utilisant `^version` qui correspond à "Compatible with version" (cf. documentation officielle).

```
// package.json
...
  "dependencies": {
    "express": "^4.14.0"
  }
...
```

Honnêtement mieux vaut se passer de ce fonctionnement par défaut et geler strictement toutes les dépendances.

C'est posible facilement avec l'option supplémentaire `--save-exact`.

```
npm install --save --save-exact <module_name>
```

Utiliser un module

L'utilisation des modules dans Node.js se fait via CommonJS. L'import de modules est réalisé avec la commande `require`, il est ainsi possible d'utiliser les modules récupérés de npm, tout comme ses propres modules.

```
// modules from npm
var express = require('express');
var cors = require('cors');
var bodyParser = require('body-parser');

// custom modules
var routes = require('./routes/index');

var app = express();

...

// URL mapping
app.use('/', routes);

app.listen(3000, function () {
  console.log('Server started on port 3000')
});
```

Gèrer les modules

La gestion des modules est une tâche importante pour s'assurer qu'un projet sera opérationnel.

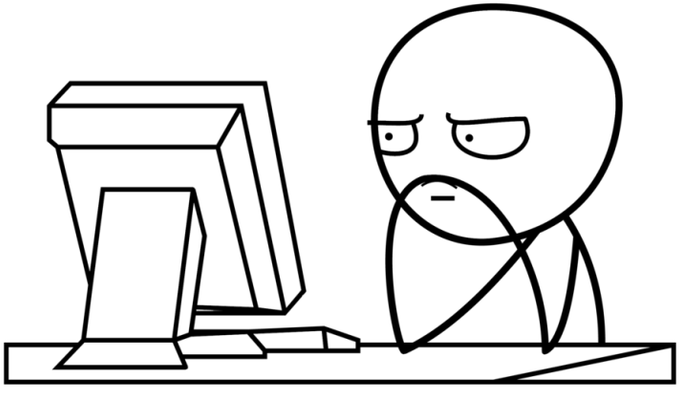
Il ne suffit pas de voir fonctionner un projet sur son environnement local.

Plusieurs commandes npm nous aide en ce sens.

```
// Supprime les modules locaux non déclarés dans package.json
npm prune

// Référencement des modules qui devraient être installés ou mis à jour
npm outdated

// Mise à jour des modules
npm update
```



Exercices

On va s'appuyer sur la plateforme NodeSchool.io, pour cela vous devez installer le TP "learnyounode".

⇒ HTTP COLLECT
⇒ JUGGLING ASYNC
⇒ TIME SERVER
⇒ HTTP FILE SERVER
⇒ HTTP UPPERCASERER
⇒ HTTP JSON API SERVER

Licence

Pour toutes questions, réclamations ou remarques, merci d'envoyer un message à alvin.berthelot@webyousoon.com.

Explications licence CC BY-SA 3.0

Cette licence permet aux autres de remixer, arranger, et adapter votre œuvre, même à des fins commerciales, tant qu'on vous accorde le mérite en citant votre nom et qu'on diffuse les nouvelles créations selon des conditions identiques.

Cette licence est souvent comparée aux licences de logiciels libres, "open source" ou "copyleft".

Toutes les nouvelles œuvres basées sur les vôtres auront la même licence, et toute œuvre dérivée pourra être utilisée même à des fins commerciales.

C'est la licence utilisée par Wikipédia ; elle est recommandée pour des œuvres qui pourraient bénéficier de l'incorporation de contenu depuis Wikipédia et d'autres projets sous licence similaire.

Contribution et réappropriation

Ce fichier PDF est généré avec Asciidoctor à partir d'un dépôt Git se trouvant sous GitHub.

https://github.com/alvinberthelot/slides-node-js

Cela signifie que vous n'avez pas besoin de vous battre avec un fichier binaire (le PDF) pour **contribuer**, **vous réapproprier le contenu** ou **modifier le thème** de présentation.

Contribution

Vous voulez **contribuer au contenu** car :

- Il y a une erreur (ça arrive à tout le monde), de typographie, de compréhension, ou tout autre chose.
- Vous souhaitez apporter une précision.

Il vous suffit de contribuer au projet via Git par le moyen d'une "pull request" sur le dépôt Git.

Réappropriation

N'oubliez pas les conditions de la licence.

Vous voulez vous **réapproprier le contenu** car :

- Vous souhaitez donner un style différent.
- Vous souhaitez enlever/ajouter/modifier des sections dans votre contexte.

Il vous suffit de "forker" le dépôt Git et d'y apporter vos propres modifications, puis de générer par vous même le nouveau PDF.