NodeJS : introduction

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

NodeJS

NodeJS ou **Node.js** ou **Node**

- Plateforme de développement : ensemble de librairies **JavaScript**
- Permettant l'exécution de **JavaScript** côté serveur et de réaliser des tâches comme :
  - Persistance de données
  - Manipulation de fichiers
  - Sécurité
  - ...
- Créé par Ryan Lienhart Dahl en 2010

NodeJS

Remarque

- Pour vérifier la compatibilité de **NodeJS** avec une version ES20** : aller à `https://node.green/#ES2015`.
- Pas besoin d'utiliser un traducteur comme **Babel**.

NodeJS

Quelques frameworks **NodeJS**

- **ExpressJS**
- Ionic
- NestJS
- ...

NodeJS

Pour **NodeJS**, il faut

- aller à `https://nodejs.org/en/`
- choisir la dernière version, télécharger et installer

NodeJS

npm : Node Package Manager

- Gestionnaire officiel de paquets **NodeJS** (modules de la communauté)
- Gestionnaire de dépendances
- Créé par Isaac Z. Schlueter
- Installé automatiquement avec **NodeJS**
- Documentation officielle : `www.npmjs.com`

NodeJS

Autres gestionnaires de paquets pour **NodeJS**

- **Yarn**
- **Bower**
- ...

NodeJS

Pour configurer (initialiser) un nouveau projet ou un projet existant

```
npm init
```

Ensuite

- Répondre aux différentes questions
- Vérifier la génération d'un fichier `package.json`

NodeJS

Exemple du contenu de `package.json`

```
{
  "name": "cours-nodejs-intro",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

NodeJS

Pour initialiser un nouveau projet ou un projet existant sans répondre aux questions

```
npm init -y
```

NodeJS

Pour initialiser un nouveau projet ou un projet existant sans répondre aux questions

```
npm init -y
```

Ou

```
npm init --yes
```

NodeJS

Pour installer globalement un package

```
npm install --global nom-package
```

NodeJS

Pour installer globalement un package

```
npm install --global nom-package
```

Ou

```
npm install -g nom-package
```

NodeJS

Pour installer localement un package

```
npm install --save nom-package
```

NodeJS

Pour installer localement un package

```
npm install --save nom-package
```

Ou

```
npm install nom-package
```

NodeJS

Explication

- Installation globale d'un package ⇒
  - package ajouté dans `C:\Users\admin\AppData\Roaming\npm\node_modules`
  - package pouvant être utilisé dans tout autre projet **NodeJS**.
- Installation locale d'un package ⇒
  - package ajouté dans `node_modules` du projet
  - package déclaré dans la section `dependencies` de `package.json`
  - package pouvant seulement être utilisé dans le projet actuel.

NodeJS

Remarques

- `npm install nom-package` ≡ `npm i nom-package`
- `npm i --save nom-package` ≡ `npm i --save-prod nom-package`
- `npm i --save-prod nom-package` ⇒ package déclaré dans la section `devDependencies` de `package.json`

NodeJS

Remarques

- `npm install nom-package` ≡ `npm i nom-package`
- `npm i --save nom-package` ≡ `npm i --save-prod nom-package`
- `npm i --save-prod nom-package` ⇒ package déclaré dans la section `devDependencies` de `package.json`

`npm install` sans arguments

Installe les dépendances listées dans la section `dependencies` du `package.json` dans le répertoire `node_modules` local.

NodeJS

Pour désinstaller un package (et le supprimer de `package.json`)

```
npm uninstall nom-package
```

NodeJS

Pour désinstaller un package (et le supprimer de `package.json`)

```
npm uninstall nom-package
```

Pour désinstaller un package (sans le supprimer de `package.json`)

```
npm uninstall nom-package --no-save
```

NodeJS

Pour désinstaller un package (et le supprimer de `package.json`)

```
npm uninstall nom-package
```

Pour désinstaller un package (sans le supprimer de `package.json`)

```
npm uninstall nom-package --no-save
```

Pour désinstaller un package global

```
npm uninstall nom-package -g
```

NodeJS

Pour mettre à jour un package local

```
npm update nom-package
```

NodeJS

Pour mettre à jour un package local

```
npm update nom-package
```

Pour mettre à jour tous les packages locaux (listés dans `package.json`

```
npm update
```

NodeJS

Pour mettre à jour un package global

```
npm update nom-package -g
```

NodeJS

Pour mettre à jour un package global

```
npm update nom-package -g
```

Pour mettre à jour tous les packages globaux

```
npm update -g
```

NodeJS

Versions d'un module

- Tout projet (ou package) **NodeJS** peut évoluer
- Et on marque cette évolution par des numéros de version souvent écrits : $x.y.z$
  - z : numéro de la correction (patch)
  - y : version mineure
  - x : version majeure

NodeJS

Considérons le contenu suivant de la section de `package.json`

```
"dependencies": {
    "mongoose": "^4.6.7",
    "mysql": "~2.3.0"
}
```

NodeJS

Considérons le contenu suivant de la section de `package.json`

```
"dependencies": {
    "mongoose": "^4.6.7",
    "mysql": "~2.3.0"
}
```


Lancer la commande suivante

```
npm install
```

NodeJS

Constats

- Aller dans `node_modules/mongoose/package.json` et vérifier que `4.13.21` est la version installée.
- Aller dans `node_modules/mysql/package.json` et vérifier que `2.3.2` est la version installée.

NodeJS

Constats

- Aller dans `node_modules/mongoose/package.json` et vérifier que `4.13.21` est la version installée.
- Aller dans `node_modules/mysql/package.json` et vérifier que `2.3.2` est la version installée.

Explication

- `4.13.21` est la dernière version avant `5.0.0`
- `2.3.2` est la dernière version avant `2.4.0`

NodeJS

Pour lister les modules qui ne sont pas à jour

```
npm outdated
```

NodeJS

Pour lister les modules qui ne sont pas à jour

```
npm outdated
```

Résultat

```
Package    Current   Wanted   Latest  Location
mongoose   4.13.21  4.13.21  5.12.13  cours-nodejs-intro
mysql        2.3.2    2.3.2   2.18.1  cours-nodejs-intro
```

NodeJS

Pour chercher un module selon un mot-clé

```
npm search mot-clé
```

NodeJS

Pour chercher un module selon un mot-clé

```
npm search mot-clé
```

Remarque

On peut aussi chercher sur `https://www.npmjs.com/`.

NodeJS

Pour lister les modules locaux

```
npm ls
```

NodeJS

Pour lister les modules locaux

```
npm ls
```

Pour lister les modules globaux

```
npm ls -g
```

NodeJS

Pour avoir de l'aide sur une commande

```
npm help nom-commande
```

NodeJS

Pour avoir de l'aide sur une commande

```
npm help nom-commande
```

Pour avoir de l'aide sur un package

```
npm view nom-package
```

NodeJS

Pour exécuter la commande définie dans la section `start` (définie dans `scripts`) de `package.json`

```
npm start
```

NodeJS

Pour exécuter la commande définie dans la section `start` (définie dans `scripts`) de `package.json`

```
npm start
```

Pour exécuter la commande définie dans la section `stop` (définie dans `scripts`) de `package.json`

```
npm stop
```

NodeJS

Trois types de modules en **NodeJS**

- modules prédéfinis : pour les utiliser, il faut écrire `require("nomModule")`.
- modules de la communauté **NodeJS** : pour les utiliser, il faut les télécharger puis soit écrire `require("nomModule")` soit les utiliser via un terminal.
- modules personnalisés : pour les utiliser, il faut les exporter puis les importer avec `require("./nomModule")`.

NodeJS

Module

Un fichier pouvant contenir des variables, constantes, fonctions, classes...

NodeJS

Module

Un fichier pouvant contenir des variables, constantes, fonctions, classes...

Propriétés

- Il est possible d'utiliser des éléments définis dans un autre fichier : variable, constante, fonction, classe...
- Pour cela, il faut l'importer là où on a besoin de l'utiliser
- Pour importer un élément, il faut l'exporter dans le fichier source

NodeJS

Étant donné le fichier `fonctions.ts` dont le contenu est

```
const somme = (a = 0, b = 0) => {
    return a + b;
}

const produit = (a = 0, b = 1) => {
    return a * b;
}
```

NodeJS

Pour exporter les deux fonctions `somme` et `produit` de `fonction.ts` : première syntaxe

```
const somme = (a = 0, b = 0) => {
    return a + b;
}

const produit = (a = 0, b = 1) => {
    return a * b;
}

module.exports = { somme, produit }
```

NodeJS

Deuxième syntaxe

```
exports.somme = (a = 0, b = 0) => {
    return a + b;
}

exports.produit = (a = 0, b = 1) => {
    return a * b;
}
```

NodeJS

Troisième syntaxe

```
module.exports = {
    somme: (a = 0, b = 0) => {
        return a + b;
    },

    produit: (a = 0, b = 1) => {
        return a * b;
    }
}
```

NodeJS

Pour importer et utiliser une fonction

```
const { somme } = require('./fonctions');

console.log(somme(2, 5));
// affiche 7
```

NodeJS

Pour importer et utiliser une fonction

```
const { somme } = require('./fonctions');

console.log(somme(2, 5));
// affiche 7
```

Pour importer plusieurs éléments

```
const { somme, produit } = require('./fonctions');

console.log(somme(2, 5));
// affiche 7

console.log(produit(2, 5));
// affiche 10
```

NodeJS

Pour tout importer dans une constante `f`

```
const f = require('./fonctions');

console.log(f.somme(2, 5));
// affiche 7

console.log(f.produit(2, 5));
// affiche 10
```

NodeJS

On peut aussi renommer les éléments exportés

```
const somme = (a = 0, b = 0) => {
    return a + b;
}

const produit = (a = 0, b = 1) => {
    return a * b;
}

module.exports = { addition: somme , produit }
```

NodeJS

On peut aussi renommer les éléments exportés

```
const somme = (a = 0, b = 0) => {
    return a + b;
}

const produit = (a = 0, b = 1) => {
    return a * b;
}

module.exports = { addition: somme , produit }
```


Pour importer

```
const { addition } = require('./fonctions');

console.log(addition(2, 5));
// affiche 7
```

NodeJS

`exports, require...`

Syntaxe native de **NodeJS** définie dans **CommonJS** : système de gestion de module de **NodeJS**.

NodeJS

`exports`, `require`...

Syntaxe native de **NodeJS** définie dans **CommonJS** : système de gestion de module de **NodeJS**.

Question

Peut-on utiliser la syntaxe **ES6** (`import`, `export`...) avec **NodeJS** ?

NodeJS

`exports`, `require`...

Syntaxe native de **NodeJS** définie dans **CommonJS** : système de gestion de module de **NodeJS**.

Question

Peut-on utiliser la syntaxe **ES6** (`import`, `export`...) avec **NodeJS** ?

Réponse

Oui avec un peu de configuration pour changer le système natif de **NodeJS**

NodeJS

Démarche

- Utiliser l'extension `.mjs` pour le fichier `fonctions`
- Modifier la configuration par défaut dans `package.json`

Pour générer `package.json`, lançons la commande

```
npm init -y
```

Pour générer `package.json`, lançons la commande

```
npm init -y
```

Ajoutons `type": "module` pour remplacer `CommonJS` dans `package.json`

```
{
  "name": "cours-nodejs",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "type": "module",
  "dependencies": {
  },
  "devDependencies": {},
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

TypeScript

Utilisons la syntaxe ES6 dans `fonction.ts`

```
const somme = (a = 0, b = 0) => {
    return a + b;
}

const produit = (a = 0, b = 1) => {
    return a * b;
}

export { somme, produit };
```

TypeScript

Utilisons la syntaxe ES6 dans `fonction.ts`

```
const somme = (a = 0, b = 0) => {
    return a + b;
}

const produit = (a = 0, b = 1) => {
    return a * b;
}

export { somme, produit };
```


Et dans `app.js`

```
import { somme, produit } from "./fonctions.mjs";

console.log(somme(2, 5));
// affiche 7

console.log(produit(2, 5));
// affiche 10
```

NodeJS

Étapes

- Créer un utilisateur NPM (`npm addUser`)
- Créer un répertoire contenant le(s) fichier(s) de notre module
- Transformer ce répertoire en répository git (`git init`) et associer ce module à un repository Github (`git remote add origin url_repository_github`, `git fetch`, `git add .` , `git commit -m "init"`, `git rebase origin/master` et `git push origin HEAD:master`)
- Initialiser votre projet `npm init`
- Ajouter le `package.json` sur github : `git add .` , `git commit -m "add package.json"` et `git push origin HEAD:master`

NodeJS

Étapes

- Publier sur npm (`npm publish`)
- Aller vérifier sur `npmjs.com`