node : une plateforme de développement

javascript et son écosystème

Jean-Christophe Routier
Licence mention Informatique
Université Lille – Sciences et Technologies

Université de Lille

Faculté des sciences et technologies
Département Informatique

nodejs.org

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. **Node.js** uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. **Node.js**' package ecosystem, **npm**, is the largest ecosystem of open source libraries in the world.

Node.js (créé en 2009)

- est une plateforme logicielle libre
- intègre une API pour créer un serveur http
- permet d'éxécuter du javascript en dehors du navigateur
- dispose d'un écosystème très large et très actif

projet nodejs

gestion via **npm** : *node package manager*

création du projet

- `npm init`

création d'un fichier `package.json`

- format JSON
- contient les informations sur le projet
- permet son installation et son exécution
- évolue avec l'installation des *packages*

```
{
  "name": "imageslider",
  "version": "1.0.0",
  "description": "my first node...",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: ..."
  },
  "author": "Timo Léon",
  "license": "ISC"
}
```

installation des paquets

- `npm install <package>`
  - installation d'un paquet nécessaire à l'application
    équivaut à `npm install <package> --save`
  - modifie `package.json` pour adapter les `"dependencies"`
  - installation dans le dossier `node_modules`
- `npm install <package> --save-dev`
  - installation d'un paquet correspondant à un « *outil* » de développement
  - modifie `package.json` pour adapter les `"devDependencies"`
- `npm install <package> -g` ou `--global`
  - installation du package globalement
    pour la mise en place de frameworks par exemple

versions

- `npm install`
  provoque l'installation de toutes les dépendances décrites dans le `package.json` présent dans le dossier

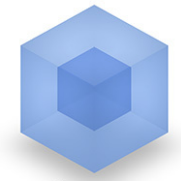
dans `package.json` `"devDependencies": { "webpack": "^4.26.1" }`

- gestion des numéros de versions *compatibles*
- numéro version : ***manor.minor.patch***
- le préfixe ~ est un joker pour *patch*
  "~4.26.1" accepte toute version $4.26.1 \leq version < 4.27.0$
- le préfixe ^ est un joker pour *minor*
  "^4.26.1" on accepte toute version $4.26.1 \leq version < 5.0.0$

quelques autres commandes

- `npm uninstall <package>`
  désinstallation d'un paquet
- `npm outdated`
  vérifie si les paquets sont à jour
- `npm update --save`
  mise à jour des paquets avec les dernières versions compatibles
- `npm prune`
  supprime les paquets inutiles

webpack



webpack

- **webpack** est un *regroupeur* de modules (module bundler)
  - l'objectif premier est d'assembler en un seul paquet plusieurs fichiers javascript pour un usage côté client
  - les dépendances entre les fichiers sont calculées et résolues dans un seul fichier "statique"
  - but : diminuer le temps de chargement
- **webpack** peut aussi transformer (transpiler, minifier, etc.) les fichiers
  utilisation de *loaders* pour définir les tâches exécuter
- **webpack** favorise une **approche modulaire** pour la conception des applis côté client

mise en œuvre

- initialisation du projet

  `npm init`

  webpack v0

- installation

  `npm install webpack webpack-cli --save-dev`

  privilégier une installation locale à chaque projet

- créer un dossier `dist` pour accueillir les fichiers produits par webpack
  - y placer le fichier `html` point d'entrée du projet
  - modifier ce fichier en fonction des bundles créés

    adapter les chargements

    webpack v1.0

modules CommonJS

webpack reconnaît les dépendances entre modules « *à la CommonJS* »

- **exportation**, dans `add.js`

```
var add = (x,y) => x+y;

module.exports = add            // ou exports.add = add;
```

- **importation**, dans `main.js`

```
var add = require('./add.js');    // '.js' facultatif
                                  // ou require('./add').add

add(2,3);
```

webpack v1.5

première *bundle-isation*

```
  ...> ./node_modules/.bin/webpack ./src/scripts/main.js --output
  ./dist/scripts/bundle.js --mode development
Hash: 3b4a27fc3379a73d0735
Version: webpack 4.26.1
Time: 76ms
Built at: 2018-11-05 13:05:37
    Asset      Size  Chunks             Chunk Names
bundle.js  4.71 KiB    main  [emitted]  main
Entrypoint main = bundle.js
[./src/scripts/add.js] 27 bytes {main} [built]
[./src/scripts/main.js] 521 bytes {main} [built]
```

webpack v2

webpack.config.js

- configuration de webpack dans `webpack.config.js`
- préciser le point d'entrée, le dossier et le fichier de sortie

```
const path = require('path');

module.exports = {
  entry: './src/scripts/main.js',
  mode : 'development',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'scripts/bundle.js'
  }
};
```

- utilisation

```
./node_modules/.bin/webpack --config webpack.config.js
```

webpack v2

npm run

possibilité de créer des *commandes* dans `package.json`

- créer une cible pour `npm run`

```
"scripts": {
  "build" : "webpack",          // "--config webpack.config.js" optionnel
},
```

puis : `npm run build`

comparer bundles avec `mode : 'production'`

loader

- les *loaders* sont des extensions qui définissent des traitements sur les fichiers

  pas uniquement fichiers javascript

- les fichiers concernés = cibles d'un `require`

  ou d'un `import` (es6)

- incitation à la modularisation
  - un module peut *requérir*/*importer* tous les fichiers dont il dépend (css, image, etc.) et webpack gère les dépendances

webpack.config.js

```
module.exports = {
  ...
  module: {
     rules: [
       {
         test: filtre des fichiers,
         use: [
           {
             loader: 'nom du loader',
             options: { des options }
           }
         ]
       },
       ... // autre règle
     ]
  }
};
```

file-loader

exemple : gestion des fichiers images

- `npm install file-loader --save-dev`
-

```
module : {
  rules: [
     {
       test: /\.(png|jpg|gif)/,
       use: [
         {
           loader: 'file-loader',
           options: {
             name : '[name].[ext]',
             outputPath : 'images/img'
           }
         }
       ]
     }
  ]}
```

- dans `main.js`

```
var timoleon = require('../images/timoleon.jpg');

var setupListeners = function() {
  ...
  timo.src = timoleon;
}
```

- `npm run build`

```
Hash: b6aeb87f7e90408eeb31
Version: webpack 3.9.1
Time: 759ms
                                  Asset       Size  Chunks  Chunk     Names
images/img/timoleon.jpg    39.3 kB          [emitted]
                       scripts/bundle.js  5.2 kB  0  [emitted]  main
...
```

webpack v3

css-loader

- `npm install --save-dev style-loader css-loader`
-

```
module: {
  rules: [
    {
      test: /\.css$/,
      use: [
        { loader: 'style-loader'},
        { loader: 'css-loader'}
      ]
    },
    ...
  ]
}
```

- dans `main.js`

```
require('../style/main.css');
...
```

webpack v4

- `npm run build`

```
Hash: 7470e9a7f2dcbbae431e
Version: webpack 4.26.1
Time: 288ms
Built at: 2018-11-27 08:17:38
                  Asset      Size  Chunks             Chunk Names
images/img/timoleon.jpg  39.3 KiB          [emitted]
      scripts/bundle.js  26.4 KiB    main  [emitted]  main
Entrypoint main = scripts/bundle.js
[...] ./node_modules/css-loader!./src/style/add.css 226 bytes {main} [built]
[...] ./node_modules/css-loader!./src/style/main.css 326 bytes {main} [built]
[./src/images/timoleon.jpg] 69 bytes {main} [built]
[./src/scripts/add.js] 84 bytes {main} [built]
[./src/scripts/main.js] 596 bytes {main} [built]
[./src/style/add.css] 1.05 KiB {main} [built]
[./src/style/main.css] 1.06 KiB {main} [built]
    + 3 hidden modules
```

- plus de chargement de style dans `index.html`
- chaque module utilisé est accompagné de sa feuille de style

modules ES6

- dans `add.js`

```
import '../style/add.css';

var add = (x,y) => x + y;
export default add;
```

- dans `main.js`

```
import add from './add.js';

import timoleonSrc from '../images/timoleon.jpg';
import '../style/main.css';

var setupListeners = ...
```

- `npm run build`

webpack v5

source map

- le *bundle* est difficile à maintenir
- les **source map** de javascript permettent de garder le lien avec le code original
- avec webpack, dans `webpack.config.js` :

```
module.exports {
  ...
  devtool: 'eval-source-map'
}
```

permet en cas d'erreur de retrouver la source concernée, notamment numéro de ligne du code

webpack v6

watch mode

- *build* automatique à chaque changement d'un des fichiers
- option `--watch` au lancement de webpack
- dans `package.json`

```
"scripts": {
  ...
  "watch" : "webpack --watch"
},
```

- `npm run watch`

hot loading

- build et rechargement automatique de la page
- `npm install --save-dev webpack-dev-server`
- fournit un serveur qui procède à un rechargement à chaque nouveau build
- dans `webpack.config.js`, préciser le point d'entrée du serveur

```
devServer: {
  contentBase : './dist',
  port : 8000                    // default is 8080
},
```

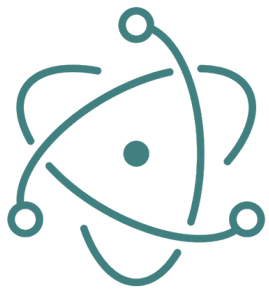
- dans `package.json` définir le script cible

```
"scripts": {
  ...
  "hotloader": "webpack-dev-server"  // --open ouvre la page au lancement
},
```

- `npm run hotloader`
  puis charger l'url `http://127.0.0.1:8000`

electron

https://electron.atom.io



- permet de produire des applications desktop (“hors navigateur”) avec
  - javascript pour le traitement
  - html et css pour la vue

  ex : Atom

mise en œuvre

- installation

  `npm install electron --save-dev --save-exact`

  `--save-exact` car non respect de la gestion de versions de nodejs

- créer le fichier `index.js`
  - voir `main.js` sur `https://electron.atom.io/docs/tutorial/quick-start/`
  - éventuellement changer le fichier `html` à utiliser et modifier les dimensions défaut `index.html`
    cf paramètres de `"mainWindow.loadURL"`
  - et éventuellement
    - supprimer les *DevTools* : retirer `win.webContents.openDevTools()`
    - supprimer la barre de menu : ajouter `win.setMenu(null);`

- exécution :

  `./node_modules/.bin/electron .`

electron-packager

- installation des paquets
  - `npm install --save-dev electron-prebuilt`
  - `npm install --save-dev electron-packager`
- création de l'exécutable

```
electron-packager <sourcedir> <appname> --platform=<platform>
                  --arch=<arch> [optional flags...]
```

  - `electron-packager --help` précise les options
  - par défaut appname est `name` dans `package.json`

exemple :

```
electron-packager .  --platform=linux,win32 --arch=x64 --out=bin
                  --overwrite --icon=./app.ico
```

npm run

- créer des *commandes* dans `package.json`

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "build-exe": "electron-packager .  --platform=linux,win32 --arch=x64
                          --out=bin --overwrite --icon=./app.ico"
},
```

- utilisation

  `npm run build-exe`