



Press Space for next page →



# Présentation Node.js (1)

- Node.js, plateforme événementielle aux entrées-sorties non-bloquantes (asynchrone) pour développer des applications en JavaScript "côté serveur"
- Moteur léger et écologique !
  - (peu gourmand en ressources, CPU, RAM)
- Noyau en JavaScript et C++ par-dessus le moteur JavaScript V8 de Google Chrome
  - (moteur V8 régulièrement mis à jour dans Node.js)
- Logiciel libre à licence permissive (licence MIT) avec la quasi totalité des modules également à licence permissive (licence MIT ou BSD)

# Présentation Node.js (2)

- Les méthodes de l'API sont asynchrones par défaut
- Le streaming au coeur de l'API
- Manipule nativement le JSON
- Sait gérer nativement toutes les requêtes HTTP
  - (pour réaliser l'équivalent d'appels AJAX côté serveur)
- Développement aisément d'applications (grande agilité)
- Les applications réalisées sont extrêmement rapides
- Mettre frontal Nginx (basé événements) à la place de Apache (basé processus), sinon performances bridées

# Utilisations de Node.js

- Node.js excellent pour :
  - API web (REST, services JSON)
  - Sites web (notamment streaming de bout en bout : par exemple streaming de fichiers depuis base de données puis streaming de ces fichiers dans archive ZIP puis streaming HTTP vers navigateur web)
  - Très nombreuses connexions et montée en charge (gère des milliers de connexions simultanées à faible charge sur un seul processus)
  - Communications temps réel (sockets, polling, etc.)

# Limitations

- Node.js inadapté pour :
  - Usage intensif du CPU : 3D, transcodage vidéo, et en général tout calcul mathématique non fourni par l'API du cœur (les fonctions cryptographiques font, elles, partie du cœur et sont rapides, en C++)
  - Mise en mémoire de grande quantité de données (pas plus de 1Go augmentable à 1,7Go max). Mais de par programmation événementielle cette limite est moins gênante qu'elle le serait avec d'autres langages comme Java par exemple (qui consomme une très grande quantité de mémoire).

# Philosophie de Node.js

- [http://substack.net/node\\_aesthetic](http://substack.net/node_aesthetic)
- La philosophie de Node.js :
  - Bibliothèque centrale minimale
  - Programmabilité et composabilité maximale
  - Réutilisabilité radicale

{ Agilité maximale

Innovation rapide car décentralisée

# Callback Node.js normalisé

- Un callback normalisé doit être passé comme argument à toutes les méthodes asynchrones

```
function(err, res) {  
  if (err) {  
    console.error("Failed:", err);  
    return;  
  }  
  console.info("Success with produced result:", res);  
}
```

Mais on peut utiliser des promesses en « promisifiant » les méthodes de Node.js (cf. infra)

# Recherche de modules

- <https://www.npmjs.com/>
- <https://github.com/>
  - Comparer la popularité grâce :
    - aux étoiles ( $>10 \rightarrow$  c'est bien)
    - au nombre des contributeurs ( $>3 \rightarrow$  c'est bien)

# Modules serveur utiles (1)

- **\*\* Tracer \*\*** : logs serveur
- **\*\* Commander \*\*** : Gestion de la ligne de commande
- **\*\* Node-convict \*\*** : lecture / enregistrement de configuration
- **\*\* Express \*\***: cadriel web/REST (web framework)
- **\*\* Koa\*\*** : cadriel web/REST (web framework) utilisant les promesses et les générateurs

# Modules serveur utiles (2)

- **Bookshelf.js** : ORM fonctionnant avec PostgreSQL, MySQL, SQLite3, Oracle
- **Ldapjs** : Client et serveur LDAP
- **Nodemailer** : Envoi de courriels
- **Archiver** : Génération d'archives ZIP, TAR en streaming
- **Fast-csv** : création / lecture de CSV

# Modules client et serveur utiles

- **Create-error** : création nouveaux types d'erreur
- **Bluebird** : Module pour promesses, ultra-rapide, avec plein d'utilitaires
- **Moment** : affichage, manipulations dates et durées
- **Validator.js** : Validation et nettoyage de chaînes de caractères
- **Nunjucks** : template engine proposant layouts et streaming (développé par Mozilla)
- **Base64url** : encode, decode, escape, unescape

# Modules client utiles

- **Loglevel** : logs en console débraillables à l'exécution et compatible avec tout navigateur web (idéal pour assurer du débogage et du support)
- **Dexie** : API IndexedDB avec des promesses, permettant notamment la recherche plein texte

# Bonnes pratiques avec modules NPM (1)

- Trouvez les modules pouvant être mis à jour

```
npm install -prefix ~/ -g npm-check-updates  
npm-check-updates
```

- Mettez à jour tous les modules (+ validez par tests) :

```
npm-check-updates -u
```

- Utilisez des numéros de version strictes de module (pas de ^ ou ~)

```
npm install loglevel --save --save-exact  
npm install jshint --save-dev --save-exact
```

# Bonnes pratiques avec modules NPM (2)

- Factorisez les dépendances communes :

```
npm dedupe
```

- Définissez strictement l'arbre de dépendances :

```
npm shrinkwrap  
git add npm-shrinkwrap.json
```

# Bonnes pratiques avec modules NPM (3)

- Vérifiez les licences de tous les modules utilisés (y compris les licences de toutes les dépendances)

```
npm install -prefix ~/ -g licence-checker  
license-checker  
license-checker | grep licences | sort | uniq
```

On trouve essentiellement des licences :

- MIT
- BSD
- Apache

# Factoriser le code avec modules NPM (1)

- Avec NPM on peut utiliser (installer) un module présent dans le registre NPM mais pas que ...
- Avec NPM on peut aussi utiliser (installer) un module présent sur un dépôt Git public, comme GitHub, ou privé



# Factoriser le code avec modules NPM (2)

- Forme des URL de module sur dépôt Git :

```
git://github.com/user/project.git#commit-ish  
git+ssh://user@hostname:project.git#commit-ish  
git+ssh://user@hostname/project.git#commit-ish  
git+http://user@hostname/project/blah.git#commit-ish  
git+https://user@hostname/project/blah.git#commit-ish
```

- Exemple pour utiliser (installer) un module privé :

```
npm install git+ssh://git@git.entreprise.fr:module1.git --save-dev --save-exact
```

- Suppression du besoin d'utilisation de git-submodules dans beaucoup de cas

# Modules CommonJS

- NPM et Node.js utilisent le standard CommonJS pour gérer les modules. La spec modules ES6 en est inspirée.

Exemple de module fourni (repository.js) et de module importé (behavior.js) :

```
// repository.js
exports.getDocsCount = getDocsCount;
function getDocsCount() {
    return db.documents.count();
}
// behavior.js
var repository = require('./repository');
repository.getDocsCount();
```

# Modules CommonJS pour code client (navigateur)

- Grâce à Browserify on peut utiliser les modules CommonJS avec tous les avantages que cela représente au niveau du code JavaScript déployé sur le client (navigateur)
- On peut ainsi disposer d'environ 80% des 120.000 modules NPM disponibles à ce jour côté client.
- Les développeurs peuvent utiliser les mêmes modules qu'ils connaissent côté client que côté serveur (et inversement :-))
- Pour mettre en pratique lire absolument : <https://github.com/substack/browserify-handbook>

# Autres gestionnaires de modules (bower, yeoman)

- Il y a actuellement plusieurs gestionnaires de modules JavaScript :
  - NPM
  - Bower
  - Yarn
- Bower et Yeoman sont moins puissants que NPM et sont généralement utilisés par les développeurs connaissant peu (la puissance de) NPM et Browserify
- NPM a vocation à devenir l'unique gestionnaire de modules JavaScript. Mais Yarn est très populaire

# Outils de construction d'applications

- **Grunt** : Outil très célèbre, basé sur la configuration, pour construire des applications web.
  - Facile à comprendre et à utiliser.
  - **Désavantage** : ne tire pas partie du streaming, fichiers lus et modifiés sur disque à chaque tâche.
- **Gulp** : Outil de construction basé sur le code, plutôt que la configuration, et utilisant le streaming
- **NPM** : NPM seul peut aussi servir à construire des applications (minification, etc.)

# Promesses

- On peut gérer le code asynchrone avec des callbacks. C'est la pratique par défaut dans Node.js
- Mais peut devenir très pénible quand plusieurs actions asynchrones s'enchaînent ou se déroulent en parallèle
  - La meilleure solution → les promesses (aussi appelées engagements ou futures)
  - En anglais on parle de promises ou de thenables

# Promesses

Exemple de code asynchrone classique :

```
fs.readFile('file.json', function(err, val) {
  if (err) {
    console.error("Unable to read file");
  } else {
    try {
      val = JSON.parse(val);
      console.log(val.success);
    } catch( e ) {
      console.error("Invalid JSON in file");
    }
  }
});
```

# Promesses

```
var P = require('bluebird');
var fs = P.promisifyAll(require('fs'));

fs.readFileAsync('file.json').then(JSON.parse)
.then(function(val) {
    console.log(val.success);
})
.catch(SyntaxError, function(err) {
    console.error("Invalid JSON in file", err);
})
.catch(function(err) {
    console.error("Unable to read file", err) ;
});
```

# Veille sécurité et correctifs

- Audit permanent de TOUS les modules JavaScript sur NPM et émissions d'alertes :
  - <http://nodesecurity.io/>
- Nouvelles versions de Node.js avec correctifs de sécurité :
  - <http://nodejs.org/>

# NSP

```
npm install --prefix ~/ -g nsp
nsp audit-package
  Name      Installed  Patched  Vulnerable  Dependency
  connect    2.7.5     >=2.8.1  nodesecurity-jobs > kue > express
nsp audit-shrinkwrap
  Name      Installed  Patched  Vulnerable  Dependency
  connect    2.7.5     >=2.8.1  nodesecurity-jobs > kue > express
```

# Mon premier programme en Node.js

```
// bonjour.js  
console.log('Bonjour');
```

```
$ node bonjour.js
```