NodeJS

Boughaleb Kamel

Présentation

- Créé par Ryan Lienhart Dahl
- Développé et maintenu par la société Joyent
- Plateforme logicielle événementielle en Javascript
- Utilise **V8 Javascript engine** de Google

Présentation

- Javascript devenu un language de développement majeur

- Capitalisation des connaissances

- Léger et Ultra rapide

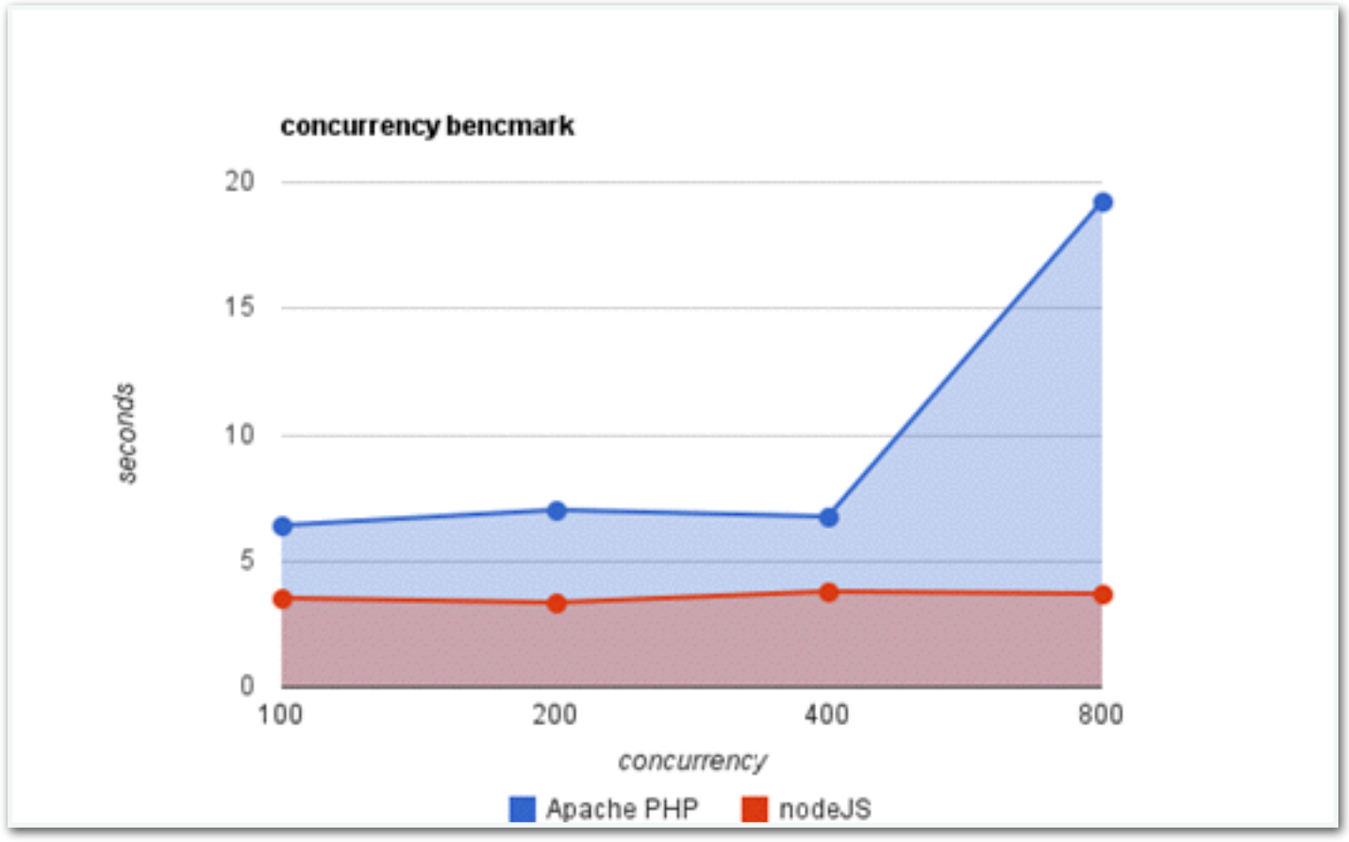
Node.js n’est pas un framework. C’est avant tout un environnement bas niveau permettant d’exécuter du Javascript. Il est plus proche du C que du PHP par exemple pour ce qui est du niveau de programmation qu’il permet?

Présentation

- 2009 : création officielle de NodeJS
- 2010 : Sortie de Nodejitsu (service cloud d’application Node.js)
- 2010: Sortie de Express.js (librairie js basée sur Node.js pour créer des appui web)
- 2011: Sortie de NPM (gestionnaire de packages)
- 2011 : Microsoft prend en charge Node.js sur Azure (cloud MS)
- 2013: Ebay, Paypal, LinkedIn, Walmart, IBM, …

Présentation

- NodeJS en mode serveur VS Apache



En temps de réponse par requête simultané, Node.js est plus performant qu'Apache.

Présentation

- Environnement d'exécution asynchrone

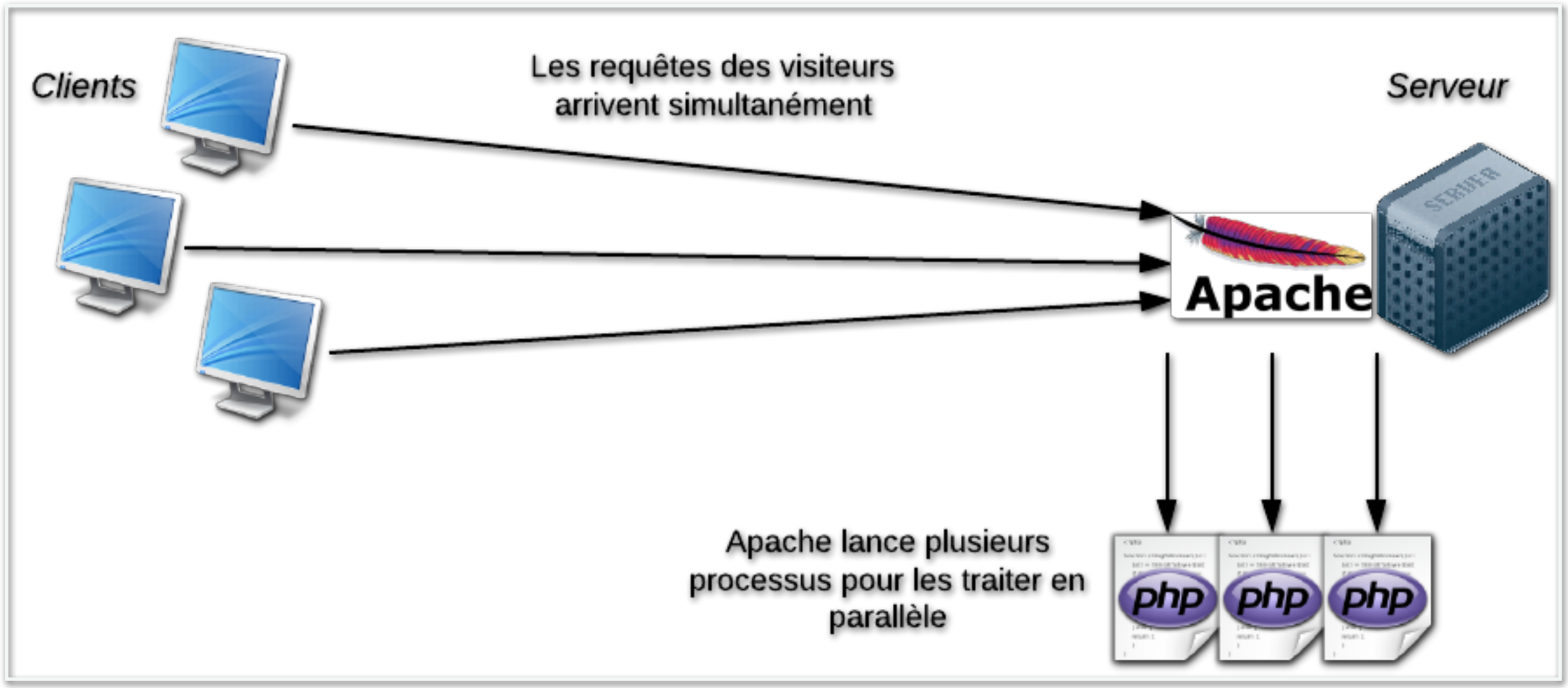
```
1  var request = require('request');
2
3  console.log("Téléchargement commencé");
4
5  request('http://www.flexilivre.com/fichier.zip',
6          function(error, response, body){
7              console.log("Téléchargement fini !");
8          });
9
10 console.log("Instructions suivantes");
```

```
MacBook-Pro-de-kamel:nodejs kamelboughaleb$ node hello.js
Téléchargement commencé
Instructions suivantes
Téléchargement fini !
```

L'exécution des instructions Node.js n'est pas séquentielle. Toutes les instructions asynchrone sont traitées en parallèle, et le flow d'exécution suit sont cours sans s'arrêter.

Présentation

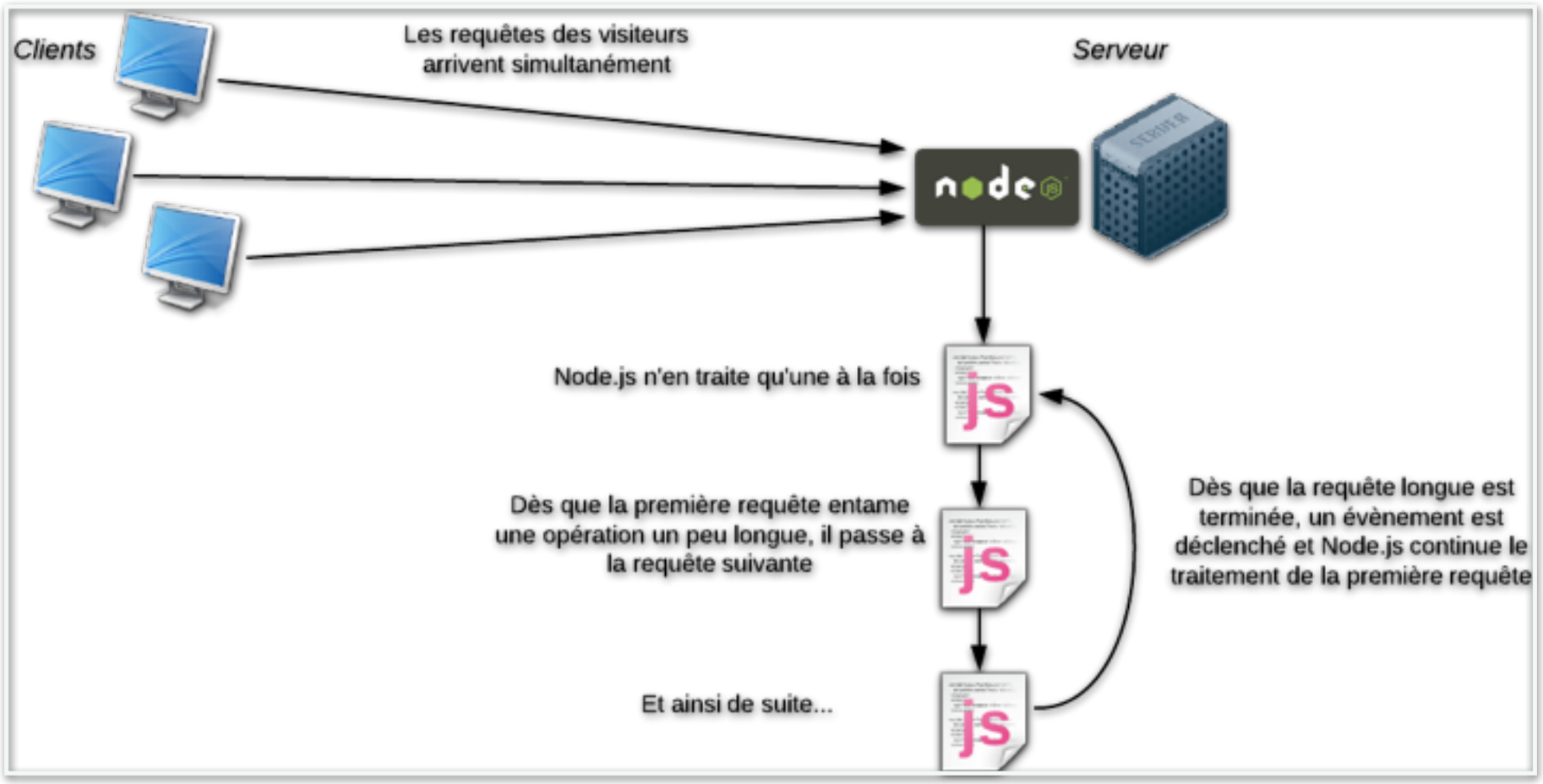
- Apache : multithread



Pour chaque requête Apache créer un nouveau thread. Cependant, chaque theard est séquentiel. Toutes les instructions sont exécutées dans l'ordre, et tant qu'une instruction n'est pas terminée, le serveur ne passe pas à la suivante.

Présentation

- Node.js : monothread



Les requêtes sur Node.js sont mis en file d'attente, et une seul requête est traité à la fois. Cependant, l'environnement Node.js étant asynchrone, les instructions ne sont pas exécutées de manière séquentielle. Aussi, chaque requête est exécuté plus rapidement.

Présentation

- Grâce à NPM, une gigantesque librairie de plugins et fonctionnalités sont facilement disponibles

- Tout est configurable : port de communication, entête HTTP, etc…

- Permet de faire des requêtes aynchrônes (idéal pour le temps réel)

Présentation

- En tant que serveur, nécessité de tout gérer soit même (tout coder)

- Problème d'incompatibilité entre version Node.js et paquets NPM

- Maturité des librairies à disposition moins avancée que celle des équivalents PHP, Java, etc…

Installation

Téléchargement :

https://nodejs.org/download/

Node et NPM sont ensuite installés sur le système.

Première application

- Créer un fichier hello.js

```
1 console.log("hello à tous!");
```

```
MacBook-Pro-de-kamel:nodejs kamelboughaleb$ node hello.js
hello à tous!
```

Node permet d'exécuter du Javascript de la même manière que sur un navigateur. La console permet d'afficher directement en sortie.

Créer un serveur HTTP

- le module http permet de créer un serveur HTTP

```
1 var http = require('http');
2
3 var server = http.createServer(function(req, res) {
4   res.writeHead(200);
5   res.end('Voici mon premier serveur');
6 });
7 server.listen(8080);
```

Node.js possède de nombreux modules pour créer des serveurs différents (TCP, FTP, File, etc…)

Personnaliser son serveur HTTP

- Le serveur est totalement personnalisable.
- ***process.argv*** permet de récupérer les paramètres passés en ligne de commande

```
30 var http = require("http");
31
32 var server = http.createServer(function(request, response){
33
34     response.writeHead(200, {'content-type':'text/html'});
35     response.end('<h1>Salut tout le monde !<h1>');
36
37 });
38
39 server.listen(process.argv[2]);
```

Créer son module Node.js

- **module.exports** permet de créer sont propre module

Fichier filterfiles.js

```
2 var path = require('path');
3
4 module.exports = function(folder, ext
5
6     fs.readdir(folder, function(err, l
7     if (err)
8         callback(err);
9     else
```

Utilisation du module

```
1 var mymodule = require('./filterfiles.js');
2
3 mymodule(process.argv[2], process.argv[3],
```

La propriété module.exports permet de définir des fonctions à importer. Ce mécanisme permet de créer ses propres modules Node.js. Lorsque de l'utilisation du module, il faut alors l'inclure à l'aide de la fonction require(). De manière générale le fichier doit être indiqué complètement. Cependant, l'extension n'est pas obigatoire. Dans l'exemple ci-dessus, il aurait été possible de seulement indiquer : ***require('./filterfiles')***

NPM

- Site internet : npmjs.com
- Noyaux de Node.js très limité
- Modules Node.js permettant d'ajouter des fonctionnalités

Il existe des milliers de modules Node.js regroupé sur npmjs.com : de la gestion de fichiers uploadés, à la connexion à la base de données MySQL, à des frameworks entiers, des templates engine, des modules de communication en temps réel, etc…

NPM

- Installer un module

```
npm install nomdumodule
```

- Mettre à jour ses modules

```
npm update
```

Lors d'un *nom update*, nom va chercher des mises à jours pour tout les modules installés. Il supprimera toutes les anciennes version.

Malgré l'importante dynamique du projet Node.js, celui-ci est encore un projet jeune. Les modules proposés sont nombreux, mais ne couvrent pas encore tous les besoins déjà comblés pas les technologies plus traditionnelles (PHP, C#, Java, etc…).

NPM

- Publier son module

- Se créer un compte sur NPM et publier !

```
npm adduser
```

```
npm publish
```

Un module n'est rien d'autre qu'une application Node.js qui contient des instructions *exports* pour partager des fonctionnalités.

Avant la publication de votre projet il est recommandé d'ajouter un fichier json (package.json) comportant au moins le nom de votre module (et éventuellement une description) sa version et ses dépendances. Un fichier README.md et un mini-tutoriel sont aussi fortement recommandé.

NPM

- Utiliser NPM pour son projet : package.json

```
{
  "name": "node-ejs",
  "main": "server.js",
  "dependencies": {
    "ejs": "^1.0.0",
    "express": "^4.6.1"
  }
}
```

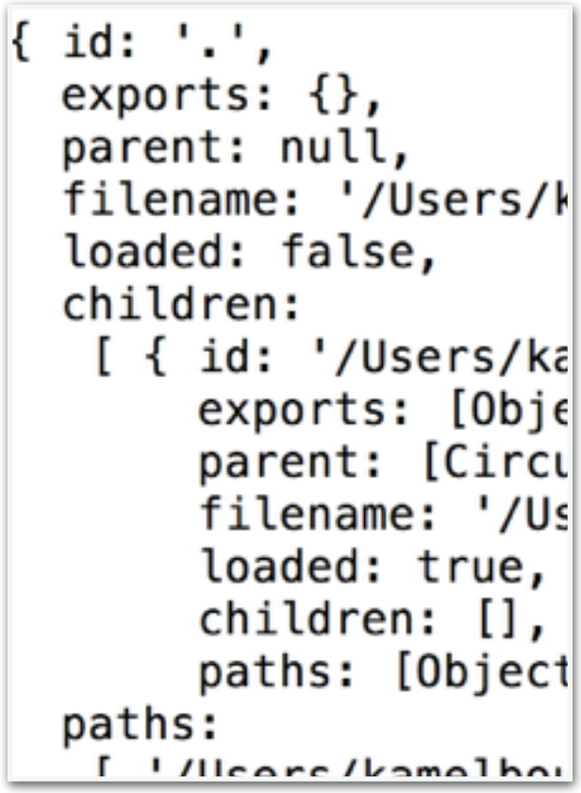
Créez un fichier package.json dans votre projet. Puis exécutez la commande :
npm install

Modules

- Unité de base dans l'organisation du code NODEJS

- Tout fichier Javascript est un module

- Chaque module contient plusieurs propriétés : ***id, filename, loaded, parent, children, exports…***



Exports / Require

Utiliser ***module.exports*** pour mettre à disposition vos données/fonctionnalités

```
1 var utilisateur = {
2     nom: 'James',
3     prenom: 'Brown',
4 };
5
6 module.exports = utilisateur;
```

```
1 var utilisateur = require("./utilisateur")
2
3 console.log(utilisateur.nom);
4 console.log(utilisateur.prenom);
```

Modules de base

- Plus d'une trentaine (traitement de fichier, streaming, etc…)

- La liste évolue rapidement

- http://nodejs.org/api/

Exemple de Module : url

- http://nodejs.org/api/url.html
- Manipulation d'url
- Indice de stabilité 3 (donc stable)

```
var url = require('url')

url.parse(URL);
url.format(urlObj);
url.resolve(from, to);
```

Variables globales

- ***global*** : espace global (this).

- ***process*** : processus courant mettant à disposition toutes les informations nécessaires.
  - *exit* : évènement envoyé lors de la fermeture
  - *uncaughtException* : événement envoyé lorsqu'une exception n'a pas été traitée

Variables globales

- Exemple : process

```
2 process.on('exit', function () {
3     console.log('le programme se termine');
4 });
5
6
7 process.on('uncaughtException', function (exception) {
8     console.error('Erreur', exception);
9
10    process.exit(1);
11 });
```

Variable globales

- Contexte d'exécution
  - **`argv`** vaut `node` + nom du script + paramètres du script courant.
  - **`env`** est un tableau contenant les variables d'environnement.
  - **`pid`** est le numéro du processus (identifiant système).
  - **`arch`** représente l'architecture courante (`arm`, `ia32`, `x64`).
  - **`platform`** est une chaîne représentant le système d'exploitation courant (`darwin`, `freebsd`, `linux`, `sunos`, `win32`).
  - **`chdir`**`(directory)` modifie le répertoire courant.
  - **`cwd`**`()` retourne le répertoire courant.
  - **`exit`**`([code])` arrête le processus potentiellement avec un code d'erreur.
  - **`getuid`**`()` retourne et `setuid(uid)` définit l'utilisateur courant.
  - **`getgid`**`()` retourne et `setgid(gid)` définit le groupe courant.

Gestion des paramètres

- **process.argv** comporte tous les paramètres

```
$ node cli.js --foo bar --baz
[ 'node', '/home/me/dev/args.js', '--foo', 'bar', '--baz' ]
```

On constate qu'en plus de contenir les différents paramètres, ce tableau contient également l'exécutable `node` en première position suivi du chemin absolu du script en cours d'exécution.

Gestion des paramètres

- Analyser les paramètres : module **minimist**

```
$ npm install --save minimist
```

```
var minimist = require('minimist');
```

```
var opts = minimist(process.argv.slice(2), {
    boolean: ['help'],
    string: ['port'],

    default: {
        port: '80'
    }
});

console.log(opts);
```

Console & trace

- console.log() : affichage d'infos dans la sortie
  Si le premier paramètre est une chaîne et si elle contient des formateurs (`%s` pour convertir en chaîne, `%d` en nombre, `%j` en JSON) alors la fonction fait le remplacement des valeurs :

```
14 console.log('Valeur %d: %j', 1, 'foo');
15 //   Valeur 1: "foo"
```

- console.error() : fonctionne de la même manière, mais écrit sur la sortie d'erreur

- trace() : affiche la pile d'appel courante

Programmation Asynchrone

- API Asynchrone de Node
- Continuation-Passing Style (CPS)

```
1 // La fonction 'continuation' n'est appelée que lorsque
2 // l'opération asynchrone est terminée.
3 asyncOperation(arg1, arg2, function continuation(result) {
4     // 'result' est le résultat.
5 });
6
7
8 var readFile = require('fs').readFile;
9
10 readFile('toto.txt', function (error, result) {
11   console.log(result);
12 });
13
```

Programmation Asynchrone et erreurs

- Par convention les erreurs dans un système Asynchrone sont retournées dans le premier paramètre

```
1  var readFile = require('fs').readFile;
2
3  readFile('texte.md', function (error, content) {
4      // Il ne faut pas oublier de vérifier l'existence d'une erreur.
5      if (error) {
6          console.error(error);
7          return;
8      }
9
10     console.log(content);
11 });
```