Le module Node.js path

Temps de lecture : 4 minutes

Le module `core path` de `Node.js`

Retour sur `Node.js`, nous allons étudier un module `core` qui nous sera très utile pour développer nos applications en utilisant le framework `Express`.

Ce module est utilisé pour manipuler facilement des `path`, c'est-à-dire des chemins vers des dossiers ou des fichiers.

En outre ce module gère tous les différents systèmes de fichiers suivant le système d'exploitation pour nous.

Les chemins ne sont en effet pas les mêmes sur Linux, MacOS et Windows.

Les exemples utiliserons les chemins `posix` sur Linux et MacOS. Mais le fonctionnement sur WIndows est exactement le même, c'est juste les chemins qui seront différents.

Pour utiliser ce module, par besoin de l'installer ni de définir son `path` car il s'agit d'un module `core` de `Node.js` :

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

Obtenir des informations sur un `path`

Plusieurs méthodes permettent d'obtenir des informations sur un `path` donné.

La méthode `basename()`

La méthode `basename()` permet de retourner la dernière portion d'un `path` :

```
path.basename('/a/b/c/d/fichier.html');
// fichier.html
```

La méthode `dirname()`

La méthode `dirname()` permet de retourner le `path` du dossier contenant le dossier ou le fichier :

```
path.dirname('/tmp/package.json');
// /tmp
```

La méthode `extname()`

La méthode `extname()` permet de retourner l'extension :

```
path.extname('/tmp/package.json');
// .json
```

La méthode `isAbsolute()`

La méthode `isAbsolute()` retourne un booléen en fonction du `path` passé en paramètre.

Elle retourne `true` si le `path` est absolu et `false` si il est relatif.

Manipuler un ou plusieurs `path`

D'autres méthodes permettent de facilement manipuler un ou plusieurs `paths` pour les utiliser.

La méthode `join()`

Comme son nom l'indique, la méthode `join()` permet de joindre plusieurs `paths` puis de les normaliser, c'est-à-dire calculer le résultat final :

```
path.join('/a', 'b', 'c/d', 'e', '..');
// '/a/b/c/d
```

Comme la dernière portion est `..`, le path normalisé remonte d'un niveau.

La méthode `normalize()`

Cette méthode permet de seulement normaliser un `path`, c'est-à-dire résoudre les `.` et les `..` contenu dans un `path` passé en paramètre.

```
path.normalize('/a/b/c/d/e/..');
// '/a/b/c/d'
```

La méthode `parse()`

La méthode `parse()` permet de retourner un objet qui contient toutes les propriétés d'un `path`.

Les propriétés sont les suivantes : `dir`, `root`, `base`, `name` et `ext`.

```
path.parse('/home/user/dir/file.txt');
// { root: '/',
//   dir: '/home/user/dir',
//   base: 'file.txt',
//   ext: '.txt',
//   name: 'file' }
```

La méthode `relative()`

La méthode `relative()` prend deux paramètres `from` et `to` et permet de calculer le `path` relatif permettant d'aller du premier chemin vers le second.

Par exemple :

```
path.relative('/a/b/c/d', '/a/b/e/f');
// '../../e/f'
```

La méthode `resolve()`

La méthode `resolve()` prend en paramètre un ensemble de `path` et va les résoudre en un `path` absolu.

A noter que les `paths` sont calculés de droite à gauche jusqu'à ce qu'un `path` absolu puisse être retourné.

```
path.resolve('/a/b', './c');
// Returns: '/a/b/c'

path.resolve('/a/b', '/c/d/');
```

```
// Returns: '/c/d'

path.resolve('root', 'a/b/', '../c/d.txt');
// Si le working directory est /home/bob/node,
// '/home/bob/node/root/a/c/d.txt'
```