Le wrapper module

Temps de lecture : 4 minutes

Le `wrapper module`

Avant que le code d'un module ne soit exécuté, `Node.js` va ajouter une fonction englobante, appelée `wrapper`.

Ce `wrapper` est le suivant :

```
(function(exports, require, module, __filename, __dirname) {
  // Le code du module contenu dans le fichier.
});
```

Pourquoi `Node.js` utilise ce `wrapper` ?

Premièrement, cela permet de **restreindre la portée des variables au module plutôt que sur l'objet global**.

Retenez, que par défaut cela empêche une contamination de l'objet global par les variables : imaginez que vous ayez deux variables `x` dans deux modules qui soient importés à des endroits totalement différents et n'aient pas du tout le même objectif : vous ne voulez pas que ces deux variables se retrouvent sur l'objet global et s'écrasent !

Deuxièmement, cela permet de définir exactement ce que vous souhaitez exporter pour rendre disponible à d'autres modules et ce qui n'est utilisé que dans le module.

Le paramètre `exports`

Nous n'allons pas revenir dessus car nous l'avons expliqué en détails dans la leçon précédente.

Il s'agit d'une référence qui pointe vers le même objet que `module.exports`.

Il s'agit donc **uniquement d'un raccourci syntaxique**.

Le paramètre `require`

Il s'agit de la fonction utilisée pour importer d'autres modules.

Vous pouvez lui passer un chemin relatif ou absolu vers le module à importer.

Vous pouvez préciser l'extension `.js` ou `.json`. Les fichiers `.js` seront interprétés comme des fichiers JavaScript et les fichiers `.json` seront parsés en JSON.

Vous pouvez ne pas préciser l'extension et dans ce cas, `Node.js` va automatiquement essayer de charger le fichier `monModule.js` puis si il ne le trouve pas il va chercher `monModule.json`.

Nous verrons plus en détails comment sont importés les fichiers en fournissant un `id` plus tard.

Pour le moment retenez que `Node.js` a des algorithmes avancés de chargement et de mise en cache des modules.

Le paramètre `module`

Il s'agit d'une référence au module courant, c'est-à-dire au code englobé par le `wrapper`.

Il contient notamment la propriété `module.exports` qui permet de définir les exports du module.

Le paramètre `__dirname`

Il s'agit d'une chaîne de caractères qui contient le chemin vers le dossier contenant le fichier, c'est-à-dire le module courant.

Par exemple :

```
/Users/monApp/
```

Le paramètre `__filename`

Il s'agit d'une chaîne de caractères qui contient le chemin absolu vers le module courant.

Par exemple :

```
/Users/monApp/monModule.js
```