

# Présentation de Node.js

## Qu'est-ce que Node.js ?

- Node.js est un environnement de serveur open source
- Node.js est gratuit
- Node.js fonctionne sur différentes plates-formes (Windows, Linux, Unix, Mac OS X, etc.)
- Node.js utilise JavaScript sur le serveur

## Pourquoi Node.js ?

### Node.js utilise la programmation asynchrone !

Une tâche courante pour un serveur Web peut être d'ouvrir un fichier sur le serveur et de renvoyer le contenu au client.

### Voici comment PHP ou ASP gère une demande de fichier :

1. Envoie la tâche au système de fichiers de l'ordinateur.
2. Attend pendant que le système de fichiers s'ouvre et lit le fichier.
3. Renvoie le contenu au client.
4. Prêt à traiter la prochaine demande.

### Voici comment Node.js gère une demande de fichier :

1. Envoie la tâche au système de fichiers de l'ordinateur.
2. Prêt à traiter la prochaine demande.
3. Lorsque le système de fichiers a ouvert et lu le fichier, le serveur renvoie le contenu au client.

Node.js élimine l'attente et continue simplement avec la requête suivante.

Node.js exécute une programmation asynchrone à un seul thread, non bloquante, qui est très économe en mémoire.

# Que peut faire Node.js ?

- Node.js peut générer du contenu de **page dynamique**
- Node.js peut créer, ouvrir, lire, écrire, supprimer et fermer des fichiers sur le serveur
- Node.js **peut collecter** des données de formulaire
- Node.js peut **ajouter, supprimer, modifier** des données dans votre **base de données**

## Qu'est-ce qu'un fichier Node.js ?

- Les fichiers Node.js contiennent des tâches qui seront exécutées sur certains événements
- Un événement typique est quelqu'un essayant d'accéder à un port sur le serveur
- Les fichiers Node.js doivent être lancés sur le serveur avant d'avoir un effet
- Les fichiers Node.js ont l'extension ".js"

---

## Node.js Démarrer

### Télécharger Node.js

Le site Web officiel de Node.js contient des instructions d'installation pour Node.js : <https://nodejs.org>

### Commencer

Une fois que vous avez téléchargé et installé Node.js sur votre ordinateur, essayons d'afficher "Hello World" dans un navigateur Web.

Créez un fichier Node.js nommé "myfirst.js" et ajoutez le code suivant :

myfirst1.js

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

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('Hello World!');
}).listen(8084);
```

Enregistrez le fichier sur votre ordinateur : C:\Users\ *Votre nom* \myfirst1.js

Le code indique à l'ordinateur d'écrire "Hello World!" si quelqu'un (par exemple un navigateur Web) essaie d'accéder à votre ordinateur sur le port 8084.

Pour l'instant, vous n'avez pas besoin de comprendre le code. Il sera expliqué plus tard.

## Interface de ligne de commande

Les fichiers Node.js doivent être lancés dans le programme "Command Line Interface" de votre ordinateur.

La façon d'ouvrir l'interface de ligne de commande sur votre ordinateur dépend du système d'exploitation. Pour les utilisateurs de Windows, appuyez sur le bouton de démarrage et recherchez "Invite de commandes", ou écrivez simplement "cmd" dans le champ de recherche.

Naviguez jusqu'au dossier qui contient le fichier "myfirst1.js", la fenêtre de l'interface de ligne de commande devrait ressembler à ceci :

```
C:\Users\Your Name>_
```

## Lancer le fichier Node.js

Le fichier que vous venez de créer doit être initié par Node.js avant qu'une action puisse avoir lieu.

Démarrez votre interface de ligne de commande, écrivez **node myfirst1.js** et appuyez sur Entrée :

Lancez "myfirst1.js":

```
C:\Users\Your Name>node myfirst1.js
```

Maintenant, votre ordinateur fonctionne comme un serveur !

Si quelqu'un essaie d'accéder à votre ordinateur sur le port 8080, il recevra un "Hello World!" message en retour !

Démarrez votre navigateur Internet et saisissez l'adresse : <http://localhost:8084>



# Modules Node.js

## Qu'est-ce qu'un module dans Node.js ?

Considérez que les modules sont identiques aux **bibliothèques JavaScript**.

Un ensemble de fonctions que vous souhaitez inclure dans votre application.

## Modules intégrés

Node.js possède un ensemble de modules intégrés que vous pouvez utiliser sans aucune autre installation.

Consultez notre [Référence des modules intégrés](#) pour une liste complète des modules.

## Inclure les modules

Pour inclure un module, utilisez la **require()** fonction avec le nom du module :

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

// Votre application a maintenant accès au module HTTP et est capable de créer un
//serveur :

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('Hello World!2 ');
}).listen(8084);
```

## Créez vos propres modules

Vous pouvez créer vos propres modules, et les inclure facilement dans vos applications.

L'exemple suivant crée un module qui renvoie un objet date et heure :

## Exemple

Créez un module qui renvoie la date et l'heure actuelles :

```
exports.myDateTime = function () {  
  return Date();  
};
```

Utilisez le mot- **exports** clé pour rendre les propriétés et les méthodes disponibles en dehors du fichier de module.

Enregistrez le code ci-dessus dans un fichier appelé "myfirstmodule.js"

## Incluez votre propre module

Vous pouvez désormais inclure et utiliser le module dans n'importe lequel de vos fichiers Node.js.

## Exemple

Utilisez le module "myfirstmodule" dans un fichier Node.js :

```
var http = require('http');  
var dt = require('./myfirstmodule');  
  
http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.write("The date and time are currently: " + dt.myDateTime());  
  res.end();  
}).listen(8084);  
console.log('http://localhost:8084');
```

Notez que nous utilisons `./` pour localiser le module, cela signifie que le module est situé dans le même dossier que le fichier Node.js.

Enregistrez le code ci-dessus dans un fichier appelé " myfirst2.js ", et lancez le fichier :

"myfirst2.js":

C:\Users\Your Name>node myfirst2.js

Si vous avez suivi les mêmes étapes sur votre ordinateur, vous verrez le même résultat que l'exemple : <http://localhost:8084>

# Module HTTP Node.js

## Le module HTTP intégré

Node.js possède un module intégré appelé HTTP, qui permet à Node.js de transférer des données via le protocole HTTP (Hyper Text Transfer Protocol).

Pour inclure le module HTTP, utilisez la **require()** méthode :

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

## Node.js en tant que serveur Web

Le module HTTP peut créer un serveur HTTP qui écoute les ports du serveur et renvoie une réponse au client.

Utilisez la **createServer()** méthode pour créer un serveur HTTP :

### Exemple

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

//create a server object:
http.createServer(function (req, res) {
  res.write('Hello World!'); //write a response to the client
  res.end(); //end the response
}).listen(8084); //the server object listens on port 8084
```

La fonction passée dans la **http.createServer()** méthode sera exécutée lorsque quelqu'un tentera d'accéder à l'ordinateur sur le port 8084.

Enregistrez le code ci-dessus dans un fichier appelé " myfirst3.js", et lancez le fichier :

Lancez myfirst3.js :

```
C:\Users\Your Name>node myfirst3.js
```

Si vous avez suivi les mêmes étapes sur votre ordinateur, vous verrez le même résultat que l'exemple : <http://localhost:8084>

# Ajouter un en-tête HTTP

Si la réponse du serveur HTTP est censée être affichée au format HTML, vous devez inclure un en-tête HTTP avec le type de contenu correct :

## Exemple

myfirst4.js

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write('Hello World!');
  res.end();
}).listen(8084);
```

Le premier argument de la `res.writeHead()` méthode est le code d'état, 200 signifie que tout est OK, le deuxième argument est un objet contenant les en-têtes de réponse.

# Lire la chaîne de requête

La fonction passée à `http.createServer()` a une `req` argument qui représente la demande du client, sous la forme d'un objet (objet `http.IncomingMessage`).

Cet objet a une propriété appelée "`url`" qui contient la partie de l'url qui vient après le nom de domaine :

myfirst5.js

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(req.url);
  res.end();
}).listen(8084);
console.log('http://localhost:8084/hiver');
console.log('http://localhost:8084/ete');
```

Enregistrez le code ci-dessus dans un fichier appelé "demo\_http\_url.js" et lancez le fichier :

Lancez myfirst5.js

C:\Users\Your Name>node myfirst5.js

Si vous avez suivi les mêmes étapes sur votre ordinateur, vous devriez voir deux résultats différents lors de l'ouverture de ces deux adresses :

<http://localhost:8084/été>

Produira ce résultat:

/ été

<http://localhost:8084/hiver>

Produira ce résultat:

/ hiver

## Fractionner la chaîne de requête

Il existe des modules intégrés pour diviser facilement la chaîne de requête en parties lisibles, comme le module URL.

### Exemple

Divisez la chaîne de requête en parties lisibles :


=====

en français

`url.parse` est une méthode du module `url` de la bibliothèque standard de Node.js, qui est utilisé pour analyser une chaîne URL en ses parties composantes.

Voici un exemple d'utilisation de `url.parse` :

javascript

 Copy code

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


const myUrl = 'https://www.example.com:8080/path?q=1#hash';
const parsedUrl = url.parse(myUrl);

console.log(parsedUrl);
```

Cela produira un objet qui représente les composants de l'URL, tels que le protocole, le nom d'hôte, le port, le nom de chemin, la chaîne de requête et le fragment.



yaml

 Copy code

```
Url {
  protocol: 'https:',
  slashes: true,
  auth: null,
  host: 'www.example.com:8080',
  port: '8080',
  hostname: 'www.example.com',
  hash: '#hash',
  search: '?q=1',
  query: 'q=1',
  pathname: '/path',
  path: '/path?q=1',
  href: 'https://www.example.com:8080/path?q=1#hash'
}
```

Vous pouvez ensuite accéder aux composants individuels de l'URL en utilisant les propriétés de l'objet retourné, comme `parsedUrl.protocol` OU `parsedUrl.query`.

=====

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

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  var q = url.parse(req.url, true).query;
  var txt = q.year + " " + q.month;
  res.end(txt);
}).listen(8084);
```

Enregistrez le code ci-dessus dans un fichier appelé "demo\_querystring.js" et lancez le fichier :

Lancez myfirst6.js:

C:\Users\Your Name>node myfirst6.js

L'adresse: <http://localhost:8084/?year=2023&month=juillet>

Produira ce résultat:

2023 July

En savoir plus sur le module URL dans le chapitre [Module URL Node.js](#).

## Exemple

URL Comme routage :

```
const http = require("http");

const server = http.createServer((req,resp)=>{
  //console.log(req);
  if(req.url=="/"){
    resp.write("<h1>Bienvenue dans notre premier http response by nodeJS
API</h1>");
  }
  else if(req.url=="/home"){
    console.log("-----");
    resp.write("<h1> thi is the home page</h1>");
  }
  else{
    console.log(req.url);
    resp.write(`<h1>Il existe une erreur d'url : </h1>
      <ul>
        <li>Ressources Not Found</li>
        <li>Erreur 404</li>
        <li><a href="/home">Go Home</li>
      </ul>
    `);
  }
  resp.end();
});

server.listen(8084);
//j'execute node modelHTTP.js puis j'execute dans le browser localhost:5000/home par
exemple
```

# Module de système de fichiers Node.js

## Node.js en tant que serveur de fichiers

Le module de système de fichiers **Node.js** vous permet de travailler avec le système de fichiers de votre ordinateur.

Pour inclure le module **File System**, utilisez la **require()** méthode :

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

Utilisation courante du module File System :

- Lire des fichiers
- Créer des fichiers
- Mettre à jour les fichiers
- Supprimer les fichiers
- Renommer les fichiers

## 1) Lire les fichiers

La **fs.readFile()** méthode est utilisée pour lire des fichiers sur votre ordinateur.

Supposons que nous ayons le fichier HTML suivant (situé dans le même dossier que Node.js) :

```
demofile1.html
```

```
1 <html>
2   <body>
3     <h1>My Header</h1>
4     <p>My paragraph.</p>
5   </body>
6 </html>
7
```

Créez un fichier Node.js qui lit le fichier HTML et renvoyez le contenu :

## Exemple

[Obtenez votre propre serveur Node.js](#)

```
1 var http = require('http');
2 var fs = require('fs');
3 http.createServer(function (req, res) {
4   fs.readFile('demofile1.html', function(err, data) {
5     res.writeHead(200, {'Content-Type': 'text/html'});
6     res.write(data);
7     return res.end();
8   });
9 }).listen(8084);
10 console.log("Le serveur est en cours d'execution");
11 console.log("l'adresse : http://localhost:8084 ");
```

Enregistrez le code ci-dessus dans un fichier appelé " **t1.js** ", et lancez le fichier :

Lancez **t1.js**:

C:\Users\Your Name>**node t1.js**

Si vous avez suivi les mêmes étapes sur votre ordinateur, vous verrez le même résultat que l'exemple : <http://localhost:8084>

## 2) Créer des fichiers

Le module **File System** propose des méthodes pour créer de nouveaux fichiers :

```
. fs.appendFile()  
. fs.open()  
. fs.writeFile()
```

### 1. fs.appendFile()

Cette méthode ajoute le contenu spécifié à un fichier. Si le fichier n'existe pas, le fichier sera créé :

### Exemple

Créez un nouveau fichier en utilisant la méthode **appendFile()** :

```
1 var fs = require('fs');  
2  
3 fs.appendFile('mynewfile1.txt', 'Bonjour javasc et nodeJS', function (err) {  
4   if (err) throw err;  
5   console.log('Le fichier "mynewfile1.txt" est Enregistrer!');  
6 });
```

### 2. fs.open()

Cette méthode prend un **mode d'ouverture** comme deuxième argument, si le mode est "w" pour "écriture", le fichier spécifié est ouvert en écriture. Si le fichier n'existe pas, un fichier vide est créé :

## Exemple

Créez un nouveau fichier vide en utilisant la méthode **open()** :

```
1 var fs = require('fs');
2
3 fs.open('mynewfile2.txt', 'w', function (err, file) {
4   if (err) throw err;
5   console.log('Le fichier "mynewfile2.txt" est Enregistrer!');
6 });
```

### 3. fs.writeFile()

Cette méthode remplace le fichier spécifié et son contenu s'il existe. Si le fichier n'existe pas, un nouveau fichier, contenant le contenu spécifié, sera créé :

## Exemple

Créez un nouveau fichier en utilisant la méthode `writeFile()` :

```
1 var fs = require('fs');
2
3 fs.writeFile('mynewfile3.txt', 'Bonjour javasc et nodeJS 2023', function (err) {
4   if (err) throw err;
5   console.log('Le fichier "mynewfile3.txt" est Enregistrer!');
6 });
```

## 3) Mettre à jour les fichiers

Le module File System dispose de méthodes pour mettre à jour les fichiers :

- **fs.appendFile()**
- **fs.writeFile()**

### 1. fs.appendFile()

Cette méthode ajoute le contenu spécifié à la fin du fichier spécifié :

## Exemple

Ajouter **un retour a la ligne** et **" On ajoute du texte"** à la fin du fichier `'mynewfile1.txt'` :

```

1  var fs = require('fs');
2
3  fs.appendFile('mynewfile1.txt', '\n On ajoute du texte', function (err) {
4      if (err) throw err;
5      console.log('Le fichier "mynewfile1.txt" est Modifié!');
6  });

```

## 2. fs.writeFile()

Cette méthode remplace le fichier et le contenu spécifiés :

### Exemple

Remplacez le contenu du fichier "mynewfile3.txt":

```

1  var fs = require('fs');
2
3  fs.writeFile('mynewfile3.txt', 'Salut C# et microsoft', function (err) {
4      if (err) throw err;
5      console.log('Le fichier "mynewfile1.txt" est Remplacé!');
6  });

```

## 4) Supprimer les fichiers

Pour supprimer un fichier avec le module File System, utilisez la **fs.unlink()** méthode.

La **fs.unlink()** méthode supprime le fichier spécifié :

### Exemple

Supprimez "mynewfile2.txt":

```
1 var fs = require('fs');
2
3 fs.unlink('mynewfile2.txt', function (err) {
4     if (err) throw err;
5     console.log('Le fichier "mynewfile2.txt" est Supprimé!');
6 });
```

## 5) Renommer les fichiers

Pour renommer un fichier avec le module File System, utilisez la **fs.rename()** méthode.

La **fs.rename()** méthode renomme le fichier spécifié :

### Exemple

Renommez "'mynewfile1.txt " en "'myrenamedfile.txt ":

```
1 var fs = require('fs');
2
3 fs.rename('mynewfile1.txt', 'myrenamedfile.txt', function (err) {
4     if (err) throw err;
5     console.log('Le fichier "mynewfile2.txt" est Renommé!');
6 });
```



# Module URL Node.js

## 1) Le module d'URL intégré

Le module URL divise une adresse Web en parties lisibles.

Pour inclure le module URL, utilisez la `require()` méthode :

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

Analysez une adresse avec la `url.parse()` méthode et elle renverra un objet URL avec chaque partie de l'adresse en tant que propriétés.

Le code `url.parse(adr, true)` est utilisé pour analyser une URL et en extraire les différentes parties qui la composent, telles que le protocole, l'hôte, le chemin, les paramètres de requête, etc.

Voici une explication détaillée de chaque paramètre de la méthode `url.parse()` :

- **adr** : Il s'agit de l'URL que nous souhaitons analyser.
- **true** : Ce deuxième paramètre est un booléen qui indique si les paramètres de la requête doivent être analysés en tant qu'objet (`true`) ou en tant que chaîne de requête (`false`).

La méthode `url.parse()` renvoie un objet qui contient les différentes parties de l'URL analysée. Voici les principales propriétés de l'objet renvoyé :

- **protocol** : Le protocole utilisé (http, https, ftp, etc.)
- **host** : L'hôte (nom de domaine ou adresse IP) de l'URL.
- **pathname** : Le chemin de l'URL.
- **query** : Les paramètres de la requête, soit sous forme d'une chaîne de requête (si `true` est `false`), soit sous forme d'un objet (si `true` est `true`).
- **hash** : le fragment d'URL, également appelé **ancree** ou hash, fait référence à la partie optionnelle d'une URL qui suit le symbole "#" et qui peut être utilisée pour identifier une section spécifique d'une ressource Web.

En résumé, le code `url.parse(adr, true)` analyse l'URL `adr` en tant qu'objet et renvoie un objet contenant les différentes parties de l'URL.

**Exemple** Divisez une adresse Web en parties lisibles :

```
1 var url = require('url');
2 var adr = 'http://localhost:8084/default.htm?year=2017&month=february#section2';
3 var q = url.parse(adr, true);
4
5 console.log(q.protocol); //returns 'http:'
6 console.log(q.host); //returns 'localhost:8084'
7 console.log(q.pathname); //returns '/default.htm'
8 console.log(q.search); //returns '?year=2017&month=february'
9
10 var qdata = q.query; //returns an object: { year: 2017, month: 'february' }
11 console.log(qdata.month); //returns 'february'
12 console.log(qdata.year); //returns '2017'
13
14 console.log(q.hash); //returns '#section2:'
```

## 2) Serveur de fichiers Node.js

Nous savons maintenant comment analyser la chaîne de requête et, dans le chapitre précédent, nous avons appris à faire en sorte que Node.js se comporte comme un serveur de fichiers. Combinons les deux, et servons le dossier demandé par le client.

Créez deux fichiers html et enregistrez-les dans le même dossier que vos fichiers node.js.

été.html

```
<!DOCTYPE html>
<html>
<body>
<h1>Summer</h1>
<p>I love the sun!</p>
</body>
</html>
```

hiver.html

```
<!DOCTYPE html>
<html>
<body>
<h1>Winter</h1>
<p>I love the snow!</p>
</body>
</html>
```

Créez un fichier **t3.js** Node.js qui ouvre le fichier demandé et renvoie le contenu au client. Si quelque chose ne va pas, lancez une erreur 404 :

**t3.js :**

```
1 var http = require('http');
2 var url = require('url');
3 var fs = require('fs');
4
5 http.createServer(function (req, res) {
6     var q = url.parse(req.url, true);
7     var filename = "." + q.pathname;
8     //Le point (".") représente le répertoire courant,
9     // donc en ajoutant le point devant le chemin relatif (pathname)
10    //extrait de l'URL, cela crée un chemin de fichier relatif
11    //qui commence à partir du répertoire courant.
12
13    fs.readFile(filename, function(err, data) {
14        if (err) {
15            res.writeHead(404, {'Content-Type': 'text/html'});
16            return res.end("404 Not Found");
17        }
18        res.writeHead(200, {'Content-Type': 'text/html'});
19        res.write(data);
20        return res.end();
21    });
22
23 }).listen(8084);
24 console.log("Le serveur est en cours d'exécution");
25 console.log("l'adresse : http://localhost:8084/summer.html ");
26 console.log("l'adresse : http://localhost:8084/winter.html ");
```

Pensez à initier le fichier :

Lancez **t2.js** :

C:\Users\Your Name>**node t2.js**

Si vous avez suivi les mêmes étapes sur votre ordinateur, vous devriez voir deux résultats différents lors de l'ouverture de ces deux adresses :

<http://localhost:8084/summer.html>

Produira ce résultat:

# Summer

I love the sun!

<http://localhost:8084/winter.html>

Produira ce résultat:

# Winter

I love the snow!

## **SERIE EXERCICES**

### **I) SUR le système de fichiers Node.js :**

- 1) Créez un fichier texte "message.txt" et écrivez-y "Bonjour Node.js !" en utilisant la méthode fs.writeFile().
- 2) Lisez le contenu du fichier "message.txt" à l'aide de la méthode fs.readFile() et affichez-le dans la console.
- 3) Créez un dossier "documents" avec la méthode fs.mkdir().
- 4) Copiez le fichier "message.txt" dans le dossier "documents" avec la méthode fs.copyFile().
- 5) Supprimez le fichier "message.txt" avec la méthode fs.unlink().
- 6) Renommez le fichier "documents/message.txt" en "documents/nouveau-message.txt" avec la méthode fs.rename().
- 7) Affichez la liste des fichiers et dossiers présents dans le dossier "documents" avec la méthode fs.readdir().
- 8) Créez un fichier "data.json" contenant un objet avec des propriétés et des valeurs de votre choix avec la méthode fs.writeFile().
- 9) Lisez le contenu du fichier "data.json" à l'aide de la méthode fs.readFile() et affichez-le dans la console.
- 10) Modifiez une propriété de l'objet contenu dans le fichier "data.json" avec la méthode fs.readFile(), puis écrivez les modifications dans le fichier avec la méthode fs.writeFile().

### **II) SUR 'URL Node.js :**

- 1) Utiliser le module 'url' pour extraire les paramètres de requête d'une URL donnée .
- 2) Utiliser le module 'url' pour extraire le chemin d'une URL donnée.
- 3) Utiliser le module 'url' pour résoudre une URL relative en une URL absolue.
- 4) Utiliser le module 'url' pour parser une URL donnée en ses différentes parties (protocole, hôte, chemin, etc.).
- 5) Utiliser le module 'url' pour générer une URL à partir de ses différentes parties (protocole, hôte, chemin, etc.).
- 6) Utiliser le module 'querystring' pour créer une chaîne de requête à partir d'un objet JavaScript.
- 7) Utiliser le module 'querystring' pour parser une chaîne de requête en un objet JavaScript.
- 8) Utiliser le module 'url' pour ajouter des paramètres de requête à une URL existante.
- 9) Utiliser le module 'url' pour extraire le nom de domaine d'une URL donnée.
- 10) Utiliser le module 'url' pour vérifier si une URL est absolue ou relative.