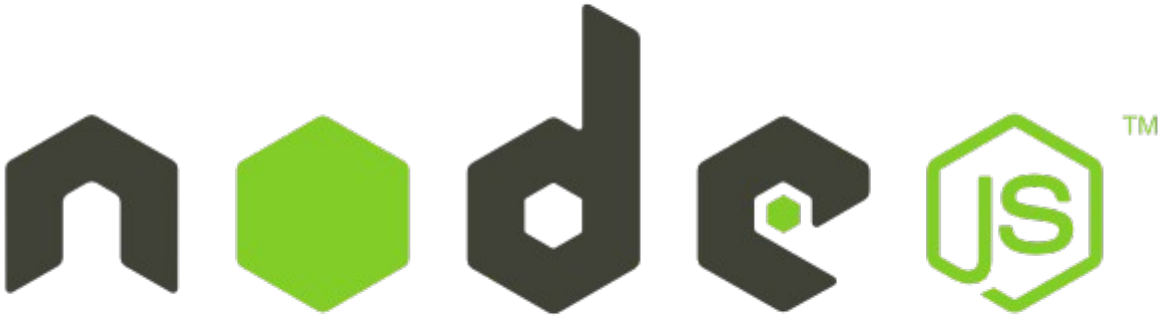
EXPLORATION NODE.JS



date	modification
2022-10-04	Correction client.query() page 6

SOMMAIRE

1 Introduction à Node.JS

Node.JS est un environnement d'exécution de scripts JavaScript.

Lorsque votre environnement Node.JS est installé (voir support de cours "Installation Node.JS"), il est possible de lancer l'interpréteur Node.JS.

1.1 Mode REPL

Utiliser Node.JS en mode REPL permet d'écrire des commandes JavaScript directement, comme dans la console de déboguage des navigateurs comme Chrome ou Firefox. REPL signifie :

1. **Read** the user input
2. **Evaluate** the code
3. **Print** any results
4. **Loop** back to first step

Dans un invite de commande, tapez l'instruction suivante :

```
node
```

Un message devrait apparaître et l'invite de commande de Node.JS doit s'afficher. Si ce n'est pas le cas, consulter l'annexe des erreurs.

```
Welcome to Node.js v16.16.0.
Type ".help" for more information.
>
```

Pour tester ceci, tapez les commandes JavaScript suivantes :

```
let prenom = "Chuck"
let nom = "NORRIS"
console.log("Bonjour "+prenom+" "+nom)
```

Votre affichage devrait être similaire à ceci :

```
Bonjour Chuck NORRIS
undefined
>
```

Utilisez `CTRL`+`C` deux fois pour quitter l'environnement REPL.

Vous pouvez aussi utiliser les commandes `.help` et `.exit` !

1.2 Mode Script

Évidemment, le mode le plus utilisé est en mode script : Node.JS va donc exécuter un fichier contenant des instructions JavaScript.

Créez le fichier script.js et ajoutez-y les commandes suivantes :

```
let a = 8
let b = 2
let oct = 2**8
console.log("un octet contient",oct,"valeurs")
```

puis lancez la commande suivante dans l'invite de commande (au même emplacement que le script) :

```
node ./script.js
```

Le résultat devrait apparaître (la valeur numérique, en couleur)

```
un octet contient 256 valeurs
```

2 ACCÈS BASE DE DONNÉES

⚠ **Démarrez un serveur mySQL avant de tester les codes qui suivent.**

Vous pouvez également utiliser une base de données en ligne : https://www.db4free.net/about.php

2.1 JEU DE DONNÉES

Le jeu de données est un ensemble d'informations permettant de tester les traitements dans un code (affichage, suppression, modification, ajout...). Voici les requêtes pour générer l'exemple :

```
CREATE TABLE `matable` (
  `id` int(11) NOT NULL,
  `name` varchar(50) NOT NULL,
  `address` varchar(300) NOT NULL,
  `email` varchar(50) NOT NULL,
  `phone` varchar(15) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

--
-- Contenu de la table `matable`
--

INSERT INTO `matable` (`id`, `name`, `address`, `email`, `phone`) VALUES
(1, 'NORRIS Chuck', '3 rue des rondins de bois 3750 NORRISTOWN',
'god@norristown.chuck', '555-0000'),
(2, 'KENT Clark', 'Daily Planet', 'ckent@daily-planet.biz', '555-1697'),
(3, 'PARR Hélène', '5569, Indestructible street 6060 GREATTOWN',
'elastigirl@indestructible.com', '555-7998'),
(4, 'DE LA VEGA Don Diego', 'hacienda del signor Don Alexandro (chez papa
quoi)', 'z@or.ro', 'inexistant');
```

id	name	address	email	phone
1	NORRIS Chuck	3 rue des rondins de bois 3750 NORRISTOWN	god@norristown.chuck	555-0000
2	KENT Clark	Daily Planet	ckent@daily-planet.biz	555-1697
3	PARR Hélène	5569, Indestructible street 6060 GREATTOWN	elastigirl@indestructible.com	555-7998
4	DE LA VEGA Don Diego	hacienda del signor Don Alexandro (chez papa quoi)	z@or.ro	inexistant

2.2 CONNEXION BASE DE DONNÉES

Étudions l'accès à une base MySQL depuis Node.JS ; L'utilisation d'une fonction s'écrit comme suit :

```
// On utilise le module node-mysql
var mysql = require('mysql');

var client= mysql.createConnection({
  host     : 'localhost',
  user     : 'root',
  password : 'motdepasse',
  database : 'mabase'
});
```

Mettre ici les paramètres du compte de la base de données.

Si le module 'mysql' n'existe pas dans votre projet, il faut l'installer avec **npm**.

Par la suite, on peut utiliser les fonctions simplement, comme ci-dessous :

```
// On se connecte à la base de données.
client.connect();
```

Et il est important de noter que JavaScript gère les objets, ainsi, les fonctions s'appliquent sur une variable correspondant à un module. L'objet contenant les éléments nécessaires pour se connecter, la fonction connect() n'a ici aucun paramètre.

L'envoi d'une requête et la récupération des tuples sera écrit comme suit :

```
client.query('SELECT * from matable', function(err, rows, fields) {
  if (!err)
    console.log('Le résultat est : ', rows);
  else
    console.log('Erreur durant la requête...');
});
```

Cette fois, on passe une variable (requête SQL sous forme d'une chaîne) en paramètre… et une fonction anonyme (ou un objet contenant les attributs err, rows et fields).

- Pour passer le proxy, il peut être nécessaire de modifier la configuration de npm :
  `npm config set proxy …`
  voir les annexes, proxy

Voici un exemple d'utilisation détaillé de l'accès à cette table avec Node.JS, fichier **mySQLExample1.js** :

```
let mysql = require('mysql');

let cnx = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "root",
  database: "mabase"
});

cnx.connect(function(err) {
  if (err) throw err;
  cnx.query("SELECT name, address FROM maTable;", function (err, result, fields) {
    if (err) throw err;

      console.log(result);               // doit être la liste
      console.log('=======================');
      let rows = JSON.parse(JSON.stringify(result[0]));
      console.log(rows);                        // écrire ligne 0 format JSON
      console.log('=======================');
      for (res of result) {
            console.log(res.name);       // écrire les noms
      }
      console.log('=======================');
      cnx.end(function(err) {
            if (err) throw err;
      });
  });
});
```

Enfin, n'oubliez pas d'installer le module MySQL avec npm :

```
npm install mysql
```

Puis exécutez le code :

```
node mySQLExample1.js
```

3 INTERACTIVITÉ DES REQUÊTES

Jusqu'à maintenant, nous n'avons fait qu'écrire une information figée ("Hello World"). L'objectif est toutefois de savoir ce que veut faire le visiteur.

3.1 PROPRIÉTÉ URL

L'utilisation de la propriété URL sur la requête permet de déterminer le chemin de l'URL demandé :

```
let http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write("je dois afficher l'application sur la page "+req.url);
  res.end();
}).listen(8080);
```

Récupère dans la requête (demande) le contenu de l'URL

sur l'objet req (qui représente la requête transmise au navigateur), on retrouve les propriétés contenues dans l'URL.

Ainsi, interroger la page http://localhost:8080/exemple affichera

```
je dois afficher l'application sur la page /exemple
```

3.2 MÉTHODE URL.PARSE

Nous pouvons aussi récupérer les informations transmises par la méthode GET ou POST dans l'URL. Pour cela, utilisons deux fichiers, un petit formulaire HTML et le code Node.JS.

form.html

```
<html>
 <body>
  <h1>Formulaires POST et GET</h1>
  <form method="GET" action="http://localhost:8080">
    nom : <input type="text" name="nom">
    age : <input type="text" name="age">
    <input type="submit" value="Envoi (post)">
  </form>

  <form method="POST" action="http://localhost:8080">
    identifiant : <input type="text" name="login">
    mot de passe : <input type="text" name="password">
    <input type="submit" value="Envoi (get)">
  </form>

 </body>
</html>
```

index.js

```
let http = require('http');
let url  = require('url');
const { parse } = require('querystring');
let txt = "";

let serveur = http.createServer(function(req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  txt = "";
  if (req.method == "GET") {
    // avec GET, les données sont dans l'URL
     console.log("Appel par méthode GET depuis", req.socket.remoteAddress);
     let q = url.parse(req.url, true).query;
     txt = "<strong>nom : </strong>"+q.nom + "<br>\n<strong>age : </strong>" +
q.age;
     res.end(txt);
  }
  else if (req.method == "POST") {
    // avec POST, il faut lire les données
     console.log("Appel par méthode POST depuis", req.socket.remoteAddress);
     // request.on : 'on' bind an event with an action
     let body="";
     req.on('data', chunk => {
       body = body + chunk.toString();
    });
     req.on('end', () => {
          console.log(parse(body));    // format JSON
     let txt = "login:"+parse(body).login+"<br>\npwd:"+parse(body).password;
          res.end(txt);
     });
  }
  console.log('------');
});

serveur.listen(8080);
```

Listener d'un flux de données

Listener de fin de flux

Le code Node.JS réagit aux deux méthodes (GET et POST), toutefois, les méthodes de lecture sont différentes :

dans le premier cas, il suffit de "parser" (analyser) l'URL,

dans le second cas, il faut récupérer la zone file data (puis l'analyser)

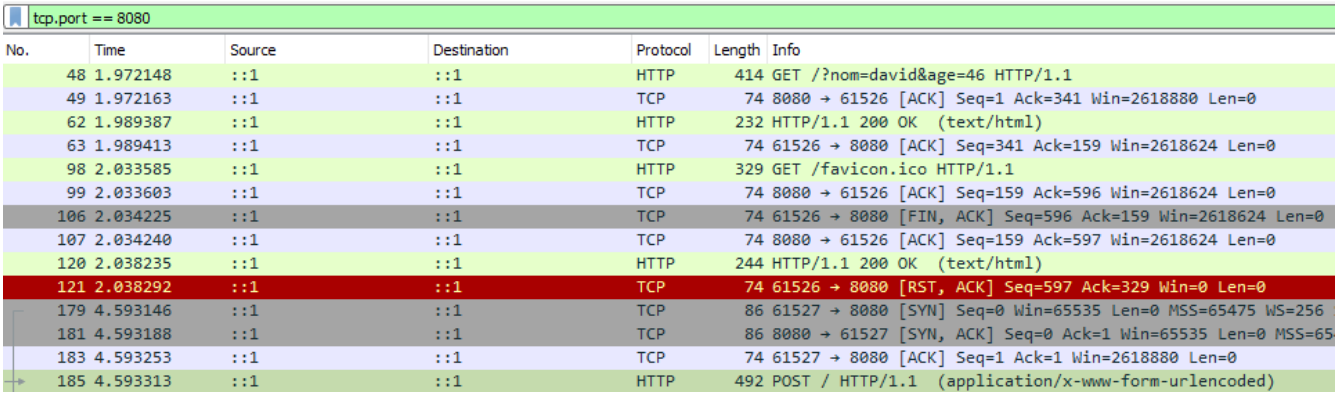
La lecture des données pouvant être lente, on utilise deux événements, que l'on associe à deux actions différentes :

- 'data' ► à la réception de données, on place les données dans l'objet chunk puis on convertit chunk en texte que l'on ajoute à la variable txt,
- 'end' ► lorsque l'événement de fin de fichier survient, on écrit le résultat dans la page de réponse.

4 USAGE DE WIRESHARK

Bien que **Wireshark**↗ soit considéré comme un outil pour les gens "réseaux", il s'avère être un outil pratique pour les développeurs, depuis l'usage de la librairie **npcap** au lieu de winpcap. NPCAP, car il peut désormais lire les flux de l'interface locale (IPv4 127.0.0.1 ou Ipv6 ::1).

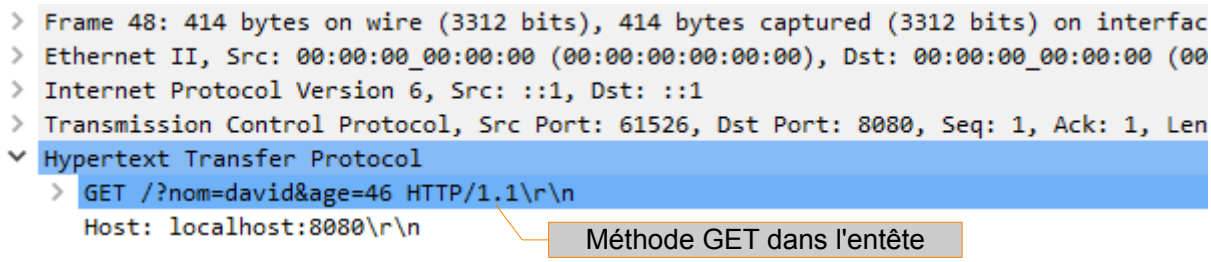
Cliquez sur les deux boutons de la page web précédente et vérifiez que vous capturez les informations de manière similaire aux deux captures d'écran suivante :



tcp.port == 8080

No.	Time	Source	Destination	Protocol	Length	Info
48	1.972148	::1	::1	HTTP	414	GET /?nom=david&age=46 HTTP/1.1
49	1.972163	::1	::1	TCP	74	8080 → 61526 [ACK] Seq=1 Ack=341 Win=2618880 Len=0
62	1.989387	::1	::1	HTTP	232	HTTP/1.1 200 OK (text/html)
63	1.989413	::1	::1	TCP	74	61526 → 8080 [ACK] Seq=341 Ack=159 Win=2618624 Len=0
98	2.033585	::1	::1	HTTP	329	GET /favicon.ico HTTP/1.1
99	2.033603	::1	::1	TCP	74	8080 → 61526 [ACK] Seq=159 Ack=596 Win=2618624 Len=0
106	2.034225	::1	::1	TCP	74	61526 → 8080 [FIN, ACK] Seq=596 Ack=159 Win=2618624 Len=0
107	2.034240	::1	::1	TCP	74	8080 → 61526 [ACK] Seq=159 Ack=597 Win=2618624 Len=0
120	2.038235	::1	::1	HTTP	244	HTTP/1.1 200 OK (text/html)
121	2.038292	::1	::1	TCP	74	61526 → 8080 [RST, ACK] Seq=597 Ack=329 Win=0 Len=0
179	4.593146	::1	::1	TCP	86	61527 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65475 WS=256
181	4.593188	::1	::1	TCP	86	8080 → 61527 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65
183	4.593253	::1	::1	TCP	74	61527 → 8080 [ACK] Seq=1 Ack=1 Win=2618880 Len=0
185	4.593313	::1	::1	HTTP	492	POST / HTTP/1.1 (application/x-www-form-urlencoded)

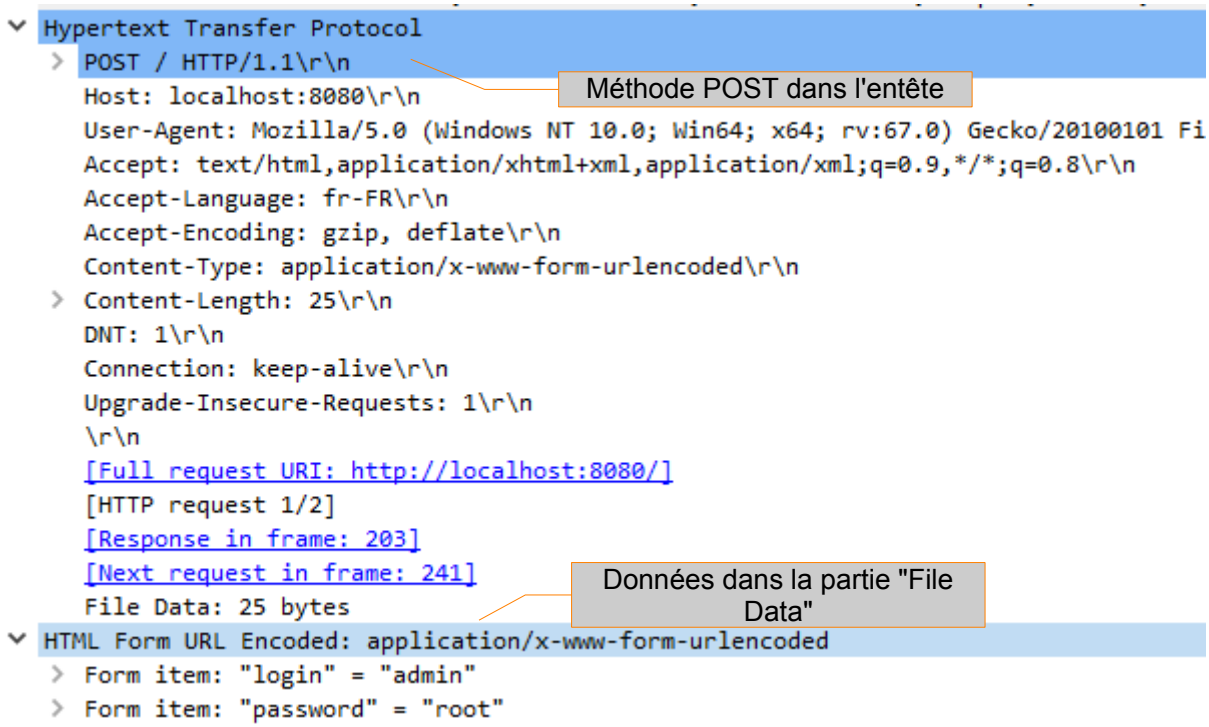
4.1 TRAME GET



Le point faible de la méthode GET concerne la visibilité des informations dans l'URL du navigateur.

Cependant, la plus grande faiblesse, vient de l'utilisation du protocole HTTP : les communications ne sont pas chiffrées. Cela signifie qu'un pirate peut écouter toutes les informations et récupérer les données qui transitent, quelle que soit la méthode (GET/POST) utilisée.

4.2 TRAME POST



On constate ici, que Wireshark décode facilement la trame, et affiche simplement les données transmises : un identifiant et un mot de passe seraient tout aussi faciles à afficher.

Dans le cadre des cours, nous travaillerons globalement en HTTP, pour permettre le décodage des trames et la vérification du bon fonctionnement des échanges. Nous ajouterons la sécurisation des échanges avec le protocole HTTPS, qui nécessite un certificat, plus tard.

5 ANNEXES

5.1 ERREURS D'INSTALLATION

5.1.1 PATH

Lors de l'installation de Node.JS, afin que la commande 'node' soit accessible partout, la variable d'environnement **PATH** doit contenir le chemin vers Node.JS.
Vérifiez que ce soit le cas.

https://www.malekal.com/comment-modifier-la-variable-path-sous-windows-10-11/

5.1.2 Session active

Si la variable d'environnement contient Node.JS, elle n'est peut-être pas prise en compte : fermez et ré-ouvrez votre session.