Programmation Web
TP n° 6 : Services Web et AJAX

Dans ce TP, dans une première partie on réalisera des services Web en node.js qui communiquent au format Json. Dans une seconde partie on utilisera le paradigme AJAX pour la communication asynchrone entre le navigateur (javascript coté client) et le serveur Web qui fournit ces services.

Services Web avec Json

Un service Web est une fonctionnalité proposée par un serveur et utilisable par d'autres programmes via le protocole HTTP. Le programme client transfère, via HTTP, au serveur les paramètres d'*input* et le serveur lui renvoie sa réponse.
Le format JSON : (JavaScript Object Notation) utilisé pour les transferts[1] est d'aspect très proche de la déclaration directe d'objets en JavaScript : un objet défini en javascript par `{name: "nom", age: 24}` sera codé en JSON par la chaîne de caractères `'{"name": "nom", "age": 24}'` (seuls les guillemets diffèrent). Un tableau `[42, "abc"]` sera vu comme `'[42, "abc"]'`.

Les objets Javascript sont donc facilement sérialisés (c.-à-d. représentés de manière portable dans le format JSON) pour pouvoir être transférés et ensuite reconstruits par le programme qui les reçoit.

JavaScript dispose des fonctions `JSON.stringify(`*Object*`)` pour transformer un objet en sa représentation JSON, et `JSON.parse(`*string*`)` pour transformer une chaîne JSON en un objet JavaScript. Cependant la plupart des fonctions qui opèrent sur le format JSON effectuent une conversion automatique ; il sera donc peu fréquent de devoir utiliser explicitement ces fonctions.

Exercice 1 : un service web pour les dictionnaires

Nous allons mettre au point un petit service Web de gestion de dictionnaire permettant :
- d'ajouter des mots au dictionnaire ;
- de rechercher les mots commençant par un préfixe donné ;
- récupérer l'ensemble des mots du dictionnaire.

1. Dans un fichier `Dico.js` écrivez une fonction/constructeur Javascript `function Dico(){...}`. On rappelle que c'est sous la forme `this.var=... ;` et `this.maMethode=function(...) {...}` que vous définissez les attributs et méthodes de l'objet.

Les mots d'un dictionnaire seront simplement stockés dans un attribut de type tableau de chaînes. On souhaite disposer des méthodes suivantes :
- `search(word)` qui renvoie un booléen indiquant si le dictionnaire contient ou non `word`
- `insert(word)` qui insère un mot dans le dictionnaire s'il ne s'y trouve pas déjà
- `words()` qui renvoie le contenu courant du dictionnaire trié.
- `prefixSearch(query)` qui renvoie un tableau contenant tous les mots du dictionnaire dont `query` est un préfixe

Pour les écrire vous pourrez utiliser les méthodes de `Array.prototype`, invocables sur tous tableaux, et suivre ces indications :
- `search(word)` : vous pouvez utiliser une simple boucle, ou encore mieux invoquer `findIndex` avec un prédicat approprié.
- `insert(word)` : combinez `search`, `push`, et `sort` (pour conserver un dictionnaire trié).

1. XML serait une alternative

– `prefixSearch(query)` : vous pouvez utiliser une boucle, ou mieux untiliser `filter` combiné avec `String.prototype.search` qui retourne l'indice de la première correspondance cherchée

Terminez votre fichier `Dico.js` par l'instruction : `module.exports = new Dico();`, elle vous permettra, dans un second fichier, d'importer l'objet construit par ce `new` via l'instruction : `const dico = require('./Dico');`

2. Écrivez à présent un serveur Node `dictionnaire.js` :

– Commencez par récupérer l'objet `dico` créé dans `Dico.js`, et remplissez manuellement ce dictionnaire dans votre code par quelques mots, dont certains auront un prefixe commun.

– Ajoutez le paramétrage d'express suivant [2] :

```
const express = require("express");
const server = new express();
server.use(express.json());
server.use(express.urlencoded({extended:false}));
```

On rappelle qu'alors :

(i) Le serveur pourra recevoir et émettre des données JSON.

(ii) Dans le traitement `(req,res)` des routes en `GET` d'URL à paramètres (c.à.d qui terminent sous la forme `?word=abc`) : `req.query` est un objet de la forme `{word='abc'}`, dont les noms de propriétés sont les noms des paramètres, et les valeurs de ces propriétés les valeurs des paramètres sous forme de chaînes.

(iii) Dans les routes en `POST` dont le corps contient une chaîne JSON (`'{"word" : "abc"}'`), `req.body` sera une référence vers l'objet résultant du parsing de cette chaîne (`{word='abc'}`).

(iv) La réponse du serveur doit être émise via `res.json(`*réponse*`)`

Voici le comportement attendu du serveur :

a. Une requête en `GET` sur `/dictionary` renvoie, au format JSON, l'ensemble des mots du dictionnaire.

b. Une requête en `GET` sur `/dictionary/search` avec un paramètre `word` renvoie, au format JSON, les mots du dictionnaire dont la valeur de `word` est un préfixe.

c. Une requête en `POST` sur `/dictionary`, dont le corps contient une chaîne JSON de la forme `'{"word":` *chaîne*`}'` insère *chaîne* dans le dictionnaire si elle ne s'y trouve pas déjà. La réponse du serveur sera un simple message de réussite ou d'erreur. [3]

Pour tester votre serveur il est très facile de simuler l'usage du service à l'aide d'un navigateur (`http://localhost:8080/dictionary/search?word=ab`) sur les requêtes en `GET`.

Pour tester le `POST` on peut simuler l'usage du service par l'envoi explicite de requêtes depuis le terminal à l'aide du la commande `curl` :

– `curl -X POST -H 'Content-Type:application/json' -d '{"word": "abc"}' localhost:8080/dictionary`

– et pour get : `curl -X GET localhost:8080/dictionary/search?word=abc`

2. à condition que votre version d'`express` soit suffisamment récente. Dans le cas où `npm view express version` ne renvoie pas un numéro de version au moins égal à `4.16.0`, il faudra commencer par installer `body-express` avec `npm`, puis initialiser le serveur par :

```
const express = require("express");
const bodyParser = require("body-parser");
const server = new express();
server.use(bodyParser.json());
server.use(bodyParser.urlencoded(extended : true));
```

3. vous pouvez utiliser la constante `undefined`

Appel aux services avec AJAX

Nous allons à présent utiliser (coté client) AJAX [4] qui a été développé pour permettre d'interagir confortablement avec un serveur de service. Il s'agit d'une surcouche de jQuery, dans laquelle on dispose en particulier des méthodes suivantes :

- `$.get(url, params, callback)`
- `$.post(url, params, callback)`

Le nom de la méthode indique le type de la connexion utilisée, l'`url` est l'adresse du serveur de service, `params` est un objet javascript (un ensemble de couples de noms de propriétés et de valeurs, ou une variable), et `callback` est une fonction de la forme `function (data) {....}` qui sera appelée lorsque le serveur aura renvoyé sa réponse `data`.

Notez que Ajax est asynchrone, et que la donnée attendue n'est disponible que dans le corps de la fonction de callback.

Voici un exemple d'utilisation : ici on interroge le serveur de l'exercice précédent avec une requête de type GET, on récupère le tableau des mots ayant "abc" comme préfixe, puis on l'affiche sur la console.

```
$.get("http://localhost:8080/dictionary/search", {word : "abc"},
  function(data) {
    console.log(data);
);
```

Un autre exemple avec une requête de type POST : il s'agit ici d'insérer le mot `abc` s'il n'y était pas déjà.

```
$.post("http://localhost:8080/dictionary", {word : "abc"},
  function(data) {
    console.log(data);
  }
);
```

Exercice 2 : Autocomplétion

Le but de cet exercice est d'utiliser le service de dictionnaire précédent afin de proposer l'auto-complétion d'un champ de texte sur votre page.

1. Dans le répertoire où vous avez écrit votre serveur, créez un sous-répertoire `public` dans lequel vous placerez une page web `dico.html` liée à un script Jquery `dico_interface.js` comme nous l'avons déjà fait dans les TP précédents.

   La page html contient un simple formulaire avec un input type de type texte nommé "query", un bouton de validation `Ajouter`, et une section `div` ayant pour identifiant "results".

2. Ajoutez au serveur une route `GET "/"` qui renvoie sur `dico.html`.
   (Vous utiliserez `res.sendFile("dico.html", {root: 'public'});`)
   Le serveur aura besoin de naviguer dans le répertoire `public` pour trouver `dico_interface.js`, précisez lui : `server.use(express.static('public'));` **Dans toute la suite, passez par le serveur localhost :8080 pour accéder à cette page, et jamais en l'ouvrant à partir de file ://**

4. Acronyme de Asynchronous Javascript And XML. Contrairement à ce que son nom suggère, AJAX permet également de travailler avec JSON.

3. Complétez `dico_interface.js` de manière à ce qu'à chaque fois que l'utilisateur modifie le contenu du champ de texte (`.on('input', ...)`), une requête `GET` est envoyée au service du dictionnaire, avec comme paramètre de recherche le contenu courant du champ texte (`.val()`).

   Le résultat est alors placé dans la section `div` "results", sous la forme d'une liste contenant chaque mot de la réponse.

4. L'appuis sur "Ajouter" aura pour effet d'ajouter le contenu du champ de texte au dictionnaire. Le `div` "results" devra bien sûr être mis à jour.

5. Faites en sorte que lorsque l'utilisateur clique sur un mot suggéré, celui-ci soit affiché dans le champ de texte et que le menu disparaisse[5].

6. Ajoutez un peu de couleur lorsque vous survolez un mot, et une animation (`slideDown()`) sur la liste des propositions.

Prefix tree (bonus)

Améliorez votre structure de dictionnaire pour qu'elle supporte efficacement l'opération de recherche de préfixe. Un type abstrait adapté à ce genre de recherche est l'arbre de préfixes (*prefix tree* ou *trie* en anglais).

Un arbre de préfixes est simplement un arbre enraciné, dont chaque sommet peut être marqué ou non, et dont chaque arête est indexée par une lettre de l'alphabet. Il possède de plus la propriété qu'il existe un chemin étiqueté $(a_1, \ldots, a_p)$ de la racine à un sommet marqué si et seulement si le dictionnaire contient le mot $a_1 a_2 \ldots a_p$.

Exercice 3 :

1. Après avoir pris soin de spécifier le type abstrait de données pour représenter un sommet, créez un constructeur Vertex pour créer des sommets de l'arbre. (Indice : on pourra utiliser un tableau à 26 éléments pour stocker les éventuels sommets fils, et utiliser la méthode `charCodeAt()` de `String` pour obtenir le code ASCII d'un caractère).
2. Créez un constructeur `Trie()` pour créer des arbres de préfixes. Implementez les méthodes `insert`, `search`, `list`, et `tt prefixSearch`.
3. Insérez une liste de mots, et assurez-vous que vos méthodes fonctionnent correctement.

5. Rappel : on peut construire un objet jQuery représentant un élément du DOM en fournissant explicitement son contenu HTML à la fonction `$( )` sous forme de chaîne de caractères. Par ailleurs, lorsque l'on fournit un handler à la méthode `click` d'un tel objet, `this` dans ce handler désigne l'objet lui-même.