Technologies Web et XML
TP2 — Manipulation de données avec JavaScript

Polytech Nantes — Info 3

Objectif du TP

L'objectif de ce TP est de manipuler des données semi-structurées en JavaScript.
À l'issue de ce TP, vous saurez :
— produire du code JavaScript qui respecte des bonnes pratiques et un style imposés,
— construire des données hiérarchiques en JavaScript,
— utiliser les prototypes en JavaScript pour donner un comportement commun à des objets similaires,
— sérialiser et désérialiser des données hiérarchiques vers et depuis le format JSON,
— sérialiser et désérialiser des structures cycliques vers et depuis le format JSON.

Rappels et compléments de JavaScript

Dans le langage JavaScript, les objets sont essentiellement des dictionnaires (liste d'associations clé/valeur) dont les clés sont des chaînes de caractères. Les tableaux (`Array`) sont des objets particuliers dont les clés représentent des indices entiers.

Le format d'échange JSON représente des données de manière hiérarchique en associant listes (`Array`), dictionnaires et types simples (chaînes de caractères, booléens, nombres...).

`Array.prototype.map` (MDN)

S'applique à un tableau. Prend une fonction en paramètre et renvoie un nouveau tableau en transformant chaque élément par la fonction.

Exemple `[0, 1, 2, 3].map(function (x) {return x * x;})` renvoie `[0, 1, 4, 9]`.

`Object.entries` (MDN)

Cette fonction fabrique un tableau avec toutes les propriétés d'un objet, chacune décrite comme un tableau à deux éléments : la clé et la valeur.

Exemple `Object.entries({a: 'a', b: 1})` renvoie `[['a', 'a'], ['b', 1]]`.

Exemple `Object.entries([0,, 2])` renvoie `[['0', 0], ['2', 2]]`

Opérateur de décomposition (MDN)

Permet de décomposer (*spread* = tartiner ?) les éléments d'un tableau comme arguments d'une fonction appelée ou comme éléments d'un tableau littéral et les propriétés d'un objet pour construire un nouvel objet littéral.

Exemple 1 : dans un appel de fonction Supposons qu'on veuille extraire la valeur minimale dans le tableau `t = [1, 3, 3, 7]`, il faudrait exécuter `Math.min(1, 3, 3, 7)`, *i.e.*, séparer les valeurs de `t` à l'appel de `Math.min` :

```
const t = [1, 3, 3, 7];
Math.min(...t); // produit 1
```

Exemple 2 : dans un tableau littéral

```
const t = [1, 3, 3, 7];
[37, ...t, 73]; // produit [37, 1, 3, 3, 7, 73]
```

Exemple 3 : dans un objet littéral

```
const o = {a: 1, b: 2};
const t = [,, 'deux'];
{...t, c: 'cé', ...o, a: 'un'}
```

produit `{ '2': 'deux', c: 'cé', a: 'un', b: 2}`

Outils

`node`

Sans arguments, `node` lance un shell interactif (REPL) qui vous permet d'expérimenter facilement. Vous pouvez également utiliser la console JavaScript de votre navigateur.

`npm`

Le sujet est accompagné d'un fichier `package.json` qui décrit les dépendances. Installez-les avec la commande `npm install`.

`gulp`

`gulp` sera utilisé pour vérifier la syntaxe de vos productions en association avec `eslint`, un *linter* JavaScript. Lancez `./node_modules/.bin/gulp` dans une console. Les fichiers ajoutés et modifiés seront scrutés et passés à la moulinette par `eslint`.

Bibliothèques logicielles

L'utilisation de bibliothèques logicielles externes autres qu'expressément stipulées n'est pas autorisée.

Documentation de référence

- Documentation normative (standards du groupe de travail TC39)
  - ECMAScript 5.1
  - ECMAScript 6
  - ECMAScript 7
- Documentation Mozilla Developer Network

Travail à rendre

Vous rendrez une archive comprenant les fichiers produits. **Tous les fichiers** (à l'exception de `albums.json`) doivent mentionner le noms des auteurs.

L'archive doit s'extraire dans un répertoire. Le nom de ce répertoire doit inclure vos noms et prénoms. L'archive est à rendre sous forme tarball comprimée ou zip (formats acceptés `.tar.gz` ou `.tgz` ou `.zip`). Tout autre format sera considéré comme un défaut de remise de travaux.

Les sources JavaScript devront se conformer aux bonnes pratiques et au style de codage mis en œuvre dans le fichier `gulpfile.js` qui accompagne ce sujet. Le respect de syntaxe et de la forme du rendu comptera pour la moitié de la note.

Liste des fichiers à inclure

1 Prise en main des outils

1.1 Analyse de code source JavaScript – exemple 1/2

```
(function () {
    var a = b = 5;
}());

console.log(typeof a);
console.log(typeof b);
```

Dans ce programme, une expression fonction est invoquée immédiatement.

1. Anticipez le résultat de ce programme.
2. Critiquez-en la lisibilité.

1.2 Analyse de code source JavaScript – exemple 2/2

Un programmeur imprudent (inconscient ?) a écrit le code suivant. Il s'agit d'une mise en œuvre de l'algorithme de tri par sélection. L'objectif de la fonction `sortTable` est de trier progressivement le tableau en sélectionnant de manière itérative la valeur minimale dans la partie droite, non triée, du tableau. Pour cela, elle s'appuie sur la fonction `findMinIndex` qui renvoie l'indice de la valeur minimale dans la partie désignée du tableau.

```
var t = [0, 3, 2, 5];
console.log('Plus petite valeur ' +
    t[findMinIndex(t, 0, t.length)]);
console.log('Plus petite valeur parmi les trois dernières ' +
    t[findMinIndex(t, 1, t.length)]);
sortTable(t);
console.log(t);

/**
 * return the index of the minimal value in the array 't' from index
 * 'from' to index 'to' (excluded)
 */
function findMinIndex(t, from, to) {
    var j = from;
    for (i = from + 1; i < to; i += 1) {
        if (t[j] > t[i]) {
            j = i;
        }
    }
    return j;
}

/**
 * sort the table 't'
 */
function sortTable(t) {
    var j, s;
    for (i = 0; i < t.length - 1; i += 1) {
        // Find the index of the minimal value in the unsorted part of
        // the array
        j = findMinIndex(t, i, t.length);
        // Swap the ith minimal value
        s = t[j];
        t[j] = t[i];
        t[i] = s;
    }
}
```

La fonction `findMinIndex` semble fonctionner (essayez vous-mêmes), la fonction `sortTable` est apparemment correcte mais le résultat faux.

1. Expliquez la nature du problème.
2. Expliquez de quelle manière très simple on aurait pu détecter le problème.
3. Corrigez ce code défaillant (le code corrigé doit être le plus proche possible du code initial).

⇒ À rendre : `tp2.1.2.js` (réponses aux questions en commentaires)

Album	Artist	Année
Fresh Cream	Cream	1966
Hot Rats	Frank Zappa	1969
Space Oddity	David Bowie	1969
Merry Christmas	Mariah Carey	1994
Songs from a Room	Leonard Cohen	1969
Ummagumma	Pink Floyd	1969
Camembert Électrique	Gong	1971
The Piper at the Gates of Dawn	Pink Floyd	1967

TABLE 1 – La collection de vinyles (et un CD, trouvez-le) de Matthieu Positivement Défini Symétrique

1.3 Outillage

Pour éviter de reproduire les erreurs précédentes, nous nous équipons d'un analyseur de syntaxe (`eslint`) qui sera lancé sur chaque fichier JavaScript modifié.

`npm install` Installation des packages listés dans `package.json`.

`node_modules/.bin/gulp` Lancement de `gulp`.

1.4 Hello World

Écrivez un script qui affiche « Hello World! ». Assurez-vous que sa syntaxe est correcte avec `eslint`.
⇒ À rendre : `tp2.1.4.js`

2 Objets JavaScript

2.1 Objets littéraux

Le moyen le plus simple de fabriquer un objet est d'utiliser une expression littérale entre accolades.

```
'use strict';

var a = {t: [2, 3, 5]};
console.log(a);
```

Listing 1 – Objet litéral

Écrivez un programme qui :

1. fabrique, à partir d'une expression littérale, un objet `albums` dont les propriétés ont pour clés les noms des albums listés dans le **??** et pour valeurs les noms des groupes.

2. affiche le nom d'un artiste en consultant l'objet `albums`.

⇒ À rendre : `tp2.2.1.js`

2.2 Structure hiérarchique

Apportez les modifications suivantes au programme précédent :

1. les valeurs associées aux noms des albums sont des objets (propriétés `artist` : nom de l'artiste, `year` : année et `title` : titre de l'album),
2. les fonctions `albumTitle`, `albumArtist` et `albumYear` qui prennent un objet album en paramètre et renvoient respectivement son titre, le nom de l'artiste et l'année de parution. *e.g.*, `albumYear(albums['Ummagumma'])` renvoie `1969`.

⇒ À rendre : `tp2.2.2.js` dans lequel les données sont écrites dans un objet littéral.

2.3 Sérialisation

À partir de cette question, les données seront stockées dans un/des fichiers JSON. La sérialisation de données hiérarchiques se fait avec la fonction `JSON.stringify`. La désérialisation avec la fonction `JSON.parse`, ou, dans un module node, avec la fonction (synchrone) `require`.

1. Écrivez le fichier `albums.json` qui reprend la structure de données de la question précédente et chargez ce fichier dans votre programme.

⇒ À rendre : `albums.json`

⇒ À rendre : `tp2.2.3.js` qui charge les données depuis `albums.json` et se comporte comme le programme précédent.

2.4 POO par prototype

Nous allons maintenant créer les objets albums à partir d'une fonction `Album` utilisée comme constructeur :

```
new Album({
    title: 'Fresh Cream',
    artist: 'Cream',
    year: 1966,
});
```

1. Écrivez ce constructeur. Vous pourrez vous aider de la fonction `Object.assign`
2. Ajoutez trois accesseurs `getTitle`, `getArtist` et `getYear` qui renvoient respectivement son titre, le nom de l'artiste et l'année de parution d'un album.
3. Créez un objet album avec ce constructeur et testez ses méthodes.

⇒ À rendre : `tp2.2.4.js`

2.5 Désérialisation de nos données 1/2

Nous allons transformer les données lues depuis `albums.json` en plusieurs étapes :

1. transformation en liste de paires `[clé, valeur]` à l'aide de la fonction `Object.entries`.
2. construction d'objets `Album`. Il s'agit de transformer chaque paire `[key, value]` fournie par `Object.entries` en `[key, new Album(value)]`. Vous utiliserez à cet effet la méthode `Array.prototype.map`.
3. construction d'objets `{[title]: album}`. Cette étape fournit un tableau d'objets, un par album, chacun contenant une propriété dont la clé est le nom de l'album et la valeur est l'album lui-même. À nouveau, c'est la méthode `Array.prototype.map` que vous utiliserez.
4. concaténation des entrées avec `Object.assign`. Pour passer les éléments d'un tableau comme arguments de la fonction `Object.assign`, vous utiliserez l'opérateur de décomposition.

⇒ À rendre : `tp2.2.5.js`

2.6 Désérialisation de nos données 2/2

1. Créez un constructeur `Artist` qui prend en paramètre le nom d'un artiste. Pour un accès facile aux artistes par leur nom, on crée une « méthode statique » `Artist.withName` qui renvoie l'artiste dont le nom est donné en paramètre. Cette méthode s'appuie sur une liste globale d'artistes stockée dans un « attribut de classe » de `Artist` : `Artist.list`. Chaque instance `Artist` est destinée à conserver un lien vers les albums produits sous forme d'un dictionnaire dont la clé est le nom de l'album.
2. Ajoutez une méthode `Artist.prototype.addAlbum` pour ajouter un album à la production d'un artiste.
3. Modifiez le constructeur `Album` de façon à lier chaque album avec l'instance qui représente l'artiste (qu'on créera si besoin) et non son nom et invoquez la méthode `Artist.prototype.addAlbum` depuis ce constructeur.

⇒ À rendre : `tp2.2.6.js`

2.7 Sérialisation de données avec des références circulaires

Notre structure de données est plus simple à exploiter mais elle contient des références circulaires (`artist` → `album` → `artist`). Il n'est plus possible de la sérialiser directement avec `JSON.stringify`. De surcroît, cette structure contient de nombreuses redondances.
Pour résoudre ce problème, `JSON.stringify` accepte en second paramètre une fonction `replacer` qui peut remplacer à la volée les valeurs à sérialiser. `replacer` reçoit deux arguments `key` et `value` et renvoie une valeur de remplacement pour `value`.

1. Écrivez une fonction `replacer` qui brise les références circulaires et supprime les redondances puis affichez le résultat de `JSON.stringify(albums, replacer, 4)`.

⇒ À rendre : `tp2.2.7.js`

3 Musicbrainz

1. Récupérez des données musicales sur le site Musicbrains [1],
2. Écrivez le code qui permet de les formater à notre schéma de données.

⇒ Bonus : `tp2.3.js`

1. *e.g.* : http://musicbrainz.org/ws/2/artist/5927990e-34bb-493f-b5da-b7b28d43698f?inc=releases&fmt=json