Compte-rendu de JavaScript

Sujet : Classes et Modèle Vue/Contrôleur

Quentin DAVIN

7 mai 2024

Modules ES6

A l'ancienne

Dans cette partie, j'ai crée les différents sous-dossiers dans le dossier `js`.

Ensuite, dans le fichier `counter.js`, j'ai juste ajouter une constante `counter` avec comme valeur `0`.

Voici le code de `counter.js` :

```
let counter = 0;
```

Et dans `application.js`, j'ai juste affiché le contenu de `counter` :

```
console.log(counter);
```

Enfin, dans `index.html`, j'ai ajouté les deux scripts à la fin du `body` :

```
…
    <script src="js/models/counter.js"></script>
    <script src="js/application/application.js"></script>
…
```

Le fait que, `counter.js` soit appelé avant `application.js` est important, sinon, `counter` ne sera pas reconnu.

Ainsi, dans la console Web, il sera affiché `0`.

Cela veut dire que, toute variable globale situé dans un script appelé dans `index.html`, est utilisable par les autres scripts de `index.html`.

Avec les modules

Je retire la balise `script` contenant le fichier `counter.js`.

Il y a un problème car, `counter` n'est défini dans aucun script.

J'ajoute le tag `type="module"` dans la balise `script` contenant `application.js`.

Cela veut dire que `application.js` est, maintenant, considéré comme un module.

Pour pouvoir utiliser `counter`, il faut pouvoir l'importer dans `application.js`.

Pour faire cela, il faut faire deux choses :

- La première est d'importer `counter` dans `application.js` avec la ligne :

```
import { counter } from "./../models/counter.js"
```

- La seconde est d'ajouter le mot clé `export` devant `let counter = 0;`, afin de rendre `counter` exportable.

Fun fact (pas rigolo pour moi) : VSCode est très sympa et importer automatiquement les éléments dans les fichiers, mais il oublie d'ajouter l'extension `.js` à la fin des fichiers. Mais, du coup, l'HTML croit qu'il doit faire appel à un URL et non à un fichier JS, ce qui m'a poussé à corriger toutes les importations à la fin.

Et encore, on trouve, encore, `0` dans la console.

Un peu mieux

J'ajoute donc la fonction `getCounter` qui retourne la valeur du compteur.

Pour cela, j'ajoute le code :

```
export function getCounter(){
    return counter;
}
```

J'enlève le `export` devant l'affectation de la variable `counter`.

Mais j'ai une erreur dans la console, `getCounter is not defined`.

Pour corriger cela, j'importe `getCounter` au lieu de `counter`.

Cela donne :

```
import { getCounter } from "./../models/counter.js"

console.log(getCounter());
```

Et là, on obtient un magnifique `0` dans la console.

Première classe

Donc j'ai crée une classe `Counter`.

Voici le code :

```
class Counter{
    #value

    constructor(){
```

```
        this.#value = 0;
    }

    getValue(){
        return this.#value;
    }

    incrementValue(){
        this.#value += 1;
    }

    decrementValue(){
        this.#value -= 1;
    }
}
```

Dans `application.js`, j'ai remplacé l'importation de `getValue` par `Counter` :

```
import { Counter } from "./../models/counter.js"

const counter = new Counter();
console.log(counter.getValue());
```

Et je me retrouve avec ce magnifique `0` dans la console.

Getter et Setter

Je supprime `getValue` et je fais un getter :

```
…
    get value(){ return this.#value; }
…
```

Et dans `application.js`, j'ai fait de même :

```
…
console.log(counter.value);
…
```

Et ça marche, bien sûr.

Sujet/Observateur

Je crée le dossier `pattern` et fichier `observer.js`. Dans celui-ci, je tape ce code :

```
export class Observer{
    notify(){
        throw new Error("You have to define this function !");
    }
}
```

Cela permet de créer une sorte de classe """abstraite""", en faisant planter le programme de toute personne n'héritant pas la classe `Observer` et ne définissant pas les fonctions de celle-ci.

Aussi, il ne faut pas oublier d'ajouter le mot-clé `export`, sinon, aucun script ne peut importer cette classe.

Je crée le fichier `notifier.js` Dans celui-ci, j'hérite cette classe d'`Observer` :

```
import { Observer } from "./pattern/observer.js";

export class Notifier extends Observer{
    #observers

    constructor(){
        this.#observers = [];
    }

    addObserver(observer){
        this.#observers.push(observer);
    }

    notify(){
        this.#observers.forEach( observer => {
            observer.notify();
        }
    }

}
```

Cette classe va permet d'ajouter des observateurs. Lorsqu'un des observers abonnés fera un changement sur la page web, il appellera la fonction `notify` du `Notifier`.

Celui-ci va dire à tous les observateurs abonnées de mettre à jour leurs données.

Modèle/Vue/Contrôleur

Modèle

Rien à ajouter.

Contrôleur

Je crée le dossier `controllers` et le fichier `controller.js`.

Je code cela dans ce fichier :

```
import { Notifier } from "./../pattern/notifier.js"
import { Counter } from "./../models/counter.js"

export class Controller extends Notifier {
    #counter;

    constructor(){
        super();
        this.#counter = new Counter();
    }

    incrementCounter(){
        this.#counter.incrementValue();
        this.notify();
    }

    decrementCounter(){
        this.#counter.decrementValue();
        this.notify();
    }

    getCounterValue(){
        return this.#counter.value;
    }
}
```

Alors, dans cette classe, j'ai dû appelé la fonction `super();`. Elle permet d'appeler le constructeur de la classe mère, et donc "confirmer" l'héritage.

J'ai dû appelé une fonction de la classe mère : `this.notify();` et cela marche, donc l'héritage marche bien.

Vue

Je crée le dossier `views` et le fichier `view.js`. Dans celui-ci, je code cela :

```
import { Observer } from "./../pattern/observer.js";

export class View export Observer{
    #controller

    constructor(controller){
        this.#controller = controller;
        controller.addObserver(this);
        this.#addEventIncrementButton();
    }

    #addEventIncrementButton(){
        const incrementButton = document.querySelector("#btn-increment");
```

```
        incrementButton.addEventListener("click", (event) => {
            this.#controller.incrementValue();
    }    );
    }

    notify(){
        counterElement = document.querySelector("#txt-counter");
        counterElement.innerHTML = this.controller.getCounterValue();
    }

}
```

Dans le `constructor`, j'ai ajouté la ligne :

```
controller.addObserver(this);
```

Cela permet d'abonner notre `View` au contrôleur, et de recevoir les ordres d'actualisation.

La fonction `#addEventIncrementButton`, qui est privé, permet d'ajouter l'évènement `click` au bouton d'incrément, afin de rendre le bouton fonctionnel.

D'ailleurs, la fonction `notify` permet de actualiser la valeur du compteur à chaque modification de celui-ci (grâce au fonctionnement de la classe `Controller`).

Pour permet d'ajouter le fonctionnement du bouton de décrementation, je modifie la fonction `#addEventIncrementButton` (dont son nom) :

```
…
#addEventsToButtons(){
    const incrementButton = document.querySelector("#btn-increment");
    incrementButton.addEventListener("click", (event) => {
            this.controller.incrementValue();
    });
    }

    const decrementButton = document.querySelector("#btn-decrement");
    decrementButton.addEventListener("click", (event) => {
            this.#controller.decrementValue();
    });
    }
}
…
```

Ainsi, la page fonctionne parfaitement !

Vous pouvez retrouver tout le code sur mon dépôt GitHub.