Les classes

Description Sommaire

À propos de ce tutoriel

Dans ce chapitre nous allons faire le point sur le fonctionnement interne des objets en JavaScript et on va parler de la notion de prototype.

Prototype ?

Si vous inspectez un objet avec votre navigateur vous remarquerez qu'il y a une propriété particulière `[[Prototype]]`. Cette propriété est un objet qui contient aussi des propriétés et des méthodes.

```
console.log({a: 1})
/**
a:1
[[Prototype]]: Object
    constructor:ƒ Object()
    hasOwnProperty: ƒ hasOwnProperty()
    isPrototypeOf: ƒ isPrototypeOf()
    propertyIsEnumerable: ƒ propertyIsEnumerable()
    toLocaleString: ƒ toLocaleString()
    toString: ƒ toString()
    valueOf: ƒ valueOf()
    __defineGetter__: ƒ __defineGetter__()
    __defineSetter__: ƒ __defineSetter__()
    __lookupGetter__: ƒ __lookupGetter__()
    __lookupSetter__: ƒ __lookupSetter__()
    __proto__: (...)
    get __proto__: ƒ __proto__()
    set __proto__: ƒ __proto__()
```

Cet objet `[[Prototype]]` peut être obtenu à l'aide de la méthode `Object.getPrototypeOf`

```
console.log(Object.getPrototypeOf({a: 1}))
```

Et on remarque que tous les types que l'on a évoqués dans cette formation possèdent cet objet prototype

```
console.log(Object.getPrototypeOf(1))
console.log(Object.getPrototypeOf("hello"))
console.log(Object.getPrototypeOf([]))
console.log(Object.getPrototypeOf(true))
```

Si on regarde le prototype d'une chaîne de caractères ou d'un tableau, on retrouve des méthodes que l'on a déjà utilisées dans des chapitres précédents. Les prototypes peuvent aussi avoir des prototypes. Mais qu'est ce que ça veut dire ?

Lorsque l'on utilise une méthode ou une propriété sur un objet, le moteur JavaScript va regarder si la méthode / propriété existe sur l'objet en question. Si ce n'est pas le cas il va regarder le prototype. Si il ne trouve toujours rien alors il regardera le prototype du prototype jusqu'à arriver au bout de la chaîne (l'élément qui n'a plus de prototype). Ce système permet d'avoir un système d'héritage au sein du langage javascript. Par exemple, une chaîne de caractères a comme prototype String, qui lui même à le prototype Object.

Créer son propre prototype

Vous pouvez aussi utiliser ce système de prototype pour vous créer des types de variables personnalisés.

Structure de base

```
class Student {

    // La propriété sera automatiquement placée sur l'objet lors de sa construction
    notes = []

    // La méthode constructor permet d'indiquer comment le type sera construit
```

```
    constructor (firstname, lastname) {
        // this permettra d'accéder à l'objet construit
        this.firstname = firstname
        this.lastname = lastname
    }

    // On peut ajouter des méthodes, this fera référence à l'objet sur lequel on uti
    addNote (note) {
        this.notes.push(note)
    }

    // On peut créer des getters, ce sont des sortes de propriétés magiques qui ont
    get fullname () {
        return `${this.firstname} ${this.lastname}`
    }

    get moyenne () {
        let sum = 0
        for (let note of this.notes) {
            sum += note
        }
        return sum / this.notes.length
    }

    // On peut aussi créer des setters pour ajouter de la logique
    set name (str) {
        const items = str.split(' ')
        this.firstname = items[0]
        this.lastname = items[1]
    }

}
```

On peut ensuite utiliser ce prototype pour créer des objets qui auront les méthodes et propriétés requises.

```
const john = new Student('John', 'Doe')
john.notes // []
john.fullname // "John Doe"
john.addNote(18)
john.moyenne // 18
```

```
john.name = 'Jane Doe'
john.firstname = "Jane"
```

Les setters / getters ne peuvent pas avoir le même nom qu'une propriété

Propriété privées

Vous pouvez définir une propriété privée qui ne sera pas accessible en dehors de la class

```
class Student {

    #notes = []

    get notes () {
        return this.#notes // On a le droit d'y accéder depuis la classe
    }

}
const a = new Student()
a.#notes // Private field '#notes' must be declared in an enclosing class
a.notes // []
```

Héritage

Vous pouvez aussi créer une classe qui aura comme prototype un autre prototype à l'aide du mot clef extends.

```
class CheaterStudent extends Student {

    set notes (note) {
        this.notes.push(20) // On "remplace" le setter parent, le tricheur aura tou
    }

}
```

Dans le cas d'un extends vous pouvez aussi appeler les méthodes parentes à l'aide du mot clef `super`. On pourra l'utiliser dans le constructeur ou n'importe quelle méthode.

```
class CheaterStudent extends Student {

    addNote (note) {
        super.addNote(20)
    }

}
```