Javascript : Les classes

1. Introduction

Javascript reste un langage prototype, ça ne change pas le langage mais permet une nouvelle écriture

Ça fonctionne comme les méthodes constructor.

1. Déclaration d’une class

Ceci avec la méthode fonction constructeur :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Produit la même chose que cette nouvelle écriture :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

D’ailleurs on retrouve bien les méthodes dans les deux cas qui sont prototypées.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Il est aussi à note qu’on peut faire comme les fonctions, des class expressions au même titre que les functions expressions.

* get / set

se retrouve aussi simplifié :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

* les properties :

Elles ne peuvent se faire que via le constructor avec le mot clef this.

Il n’est pas encore prevu dans la spec de les déclarer dans la class.

* Différence avec fonction constructeur :

Par défaut elle est enumerable = false ; donc pas de for…in possible par defaut.

1. Héritage

* Avec fonction constructeur :

Ca change du fonctionnement avec les objets litteraux sur ce que l’on recupere et aussi l’utilisation des méthodes et du this et la problématique du context lexical.

Quand on hérite en utilisant les fonctions constructeurs :

* Passer le prototype à la fonction qui hérite (attention les slots doivent être précisés après car ils sont ecrasés par la redifinition du protype.)
* On appelle la fonction constructeur depuis la fonction constructeur qui hérite.
  + Il faut utiliser la méthode .call pour changer le contexte lexical et que le this point bien sur le this du car… autrement cela ne fonctionnerait pas.
  + Si on l’invoque sans call le this ne pointe pas sur la bonne référence.

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

* Avec une class

Beaucoup plus simple :

* On utilise le mot clef extends avec la class dont on herite
* On utilise la méthode super() dans le constructeur de l’enfant
  + Cette méthode permettra de positionner le this parent sur la bonne reference (comme le call)

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Ceci donne exactement le même le résultat.

Information :

Si on ne définit pas de constructor dans l’enfant JS fabrique un constructor par defaut :

constructor(…args){

super(…args) ;

}

1. Méthodes statiques :

Sur une fonction constructor :

Dans ce cas la on déclare une methode qui appartient à Car() mais pas à ses instances. (on a pas utilisé prototype)

//Pas disponible

car.description() ;

//Disponible

Car.description() ;

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Pour avoir le même comportement dans une class on utilise le mot clef static.

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

Important :

Les méthodes statiques sont héritées.

1. Les propriétés privées

Par défaut toutes les méthodes et propriétés sont disponibles partout.

Il n’y a pas de mot clef private encore en JS, cependant pour rendre une méthode uniquement utilisable au sein de sa class on lui rajouter simplement un # devant sa définition.

Exemple ici, il sera impossible d’appeler startEngine depuis l’exterieur de la class.

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

Ce fonctionnement est étendu sur les properties. Mais il faudra déclarer la propriété en amont dans la class.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

On peut enfin maintenant avoir une bonne encapsulation avec getter et setter adéquates.

1. Les champs publics

C’est une écriture très récente. On peut parfaitement déclarer maintenant les propriétés qui seront publiques directement dans la class.

Elles seront accessibles sur les instances.

Idem pour les privates en ajoutant les # devant.

Du coup on peut mettre des valeurs par defaut.

Aujourd’hui cette écriture est préconisée pour les développements futurs.

Une déclaration public de cette manière n’empêche pas l’initialisation par constructeur.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

1. Étendre les objets natifs avec extends.

On peut parfaitement étendre des objets natifs ! (\_\_proto\_\_)

Exemple avec une classe qui génère un tableau de valeurs aléatoires qaund on appelle random() :

(il faut se rappeler que comme on ne déclare par un constructeur il va en générer un par défaut et il appellera le constructeur de Array avec le …args ☺)

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

1. L’opérateur instanceof

instanceof va rechercher si un objet à le meme protoype que la class / fonction constructeur.

Il permet aussi de remonter la chaine de prototype en cas d’héritage (polymorphisme)

<une instance> instanceof <UneClass> => renvoi true ou false.

1. Mixin

C’est un ensemble de propriétés qu’on souhaite partager à un ensemble de class.

Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement

On utilise Object.assign pour assigner un objet contenant des méthodes à une class.

Cela n’est pas un héritage mais permet d’ajouter des fonctionnalités à une classe.

Ici quand on va créer une nouvelle instance de car on aura automatiquement les fonctionnalités.