Introduction POO

Introduction à la POO en JavaScript

La programmation orientée objet (POO) est un paradigme de programmation qui utilise des objets pour organiser le code. En JavaScript, la POO a été améliorée avec l'introduction des classes dans ECMAScript 2015 (ES6).

Classes en JavaScript

Les classes sont un moyen de définir des objets et des fonctionnalités associées. Une classe est un modèle pour créer des objets avec des propriétés et des méthodes. Voici comment déclarer une classe en JavaScript :

```
class Animal {
    constructor(nom) {
        this.nom = nom;
    }

    direNom() {
        console.log(`Je m'appelle ${this.nom}`);
    }
}

const monanimal = new Animal("Rex");
console.log(monanimal.nom);
```

Dans cet exemple, la classe `Animal` a un constructeur qui initialise la propriété `nom` et une méthode `direNom` qui affiche le nom de l'animal.

Constructeurs

Un constructeur en programmation orientée objet est une fonction spéciale qui est appelée lors de la création d'une instance d'une classe.

Voici comment définir un constructeur dans une classe JavaScript :

```
class Personne {
    constructor(nom, age) {
        this.nom = nom;
        this.age = age;
    }
}
```

Dans cet exemple, `constructor` est le constructeur de la classe `Personne`. Lorsque vous créez une nouvelle instance de `Personne` à l'aide du mot-clé `new`, le constructeur est automatiquement appelé, et vous pouvez initialiser les propriétés de l'objet.

```
const personne1 = new Personne("Alice", 30);
console.log(personne1.nom); // Affiche "Alice"
console.log(personne1.age); // Affiche 30
```

Maintenant, examinons les éléments clés liés au constructeur :

1. **Appel automatique**: Lorsque vous créez une instance d'une classe avec `new`, le constructeur est appelé automatiquement.
2. **Initialisation des propriétés**: Le constructeur est souvent utilisé pour initialiser les propriétés de l'objet créé. Dans l'exemple ci-dessus, le constructeur initialise les propriétés `nom` et `age` avec les valeurs passées en paramètres.
3. **Référence à l'objet courant (`this`)**: À l'intérieur du constructeur, le mot-clé `this` fait référence à l'instance de l'objet en cours de création. Cela permet d'assigner des valeurs aux propriétés spécifiques de cet objet.
4. **Paramètres du constructeur**: Les paramètres du constructeur sont utilisés pour passer des valeurs spécifiques lors de la création d'une instance. Dans l'exemple, `nom` et `age` sont des paramètres du constructeur.
5. **Pas de mot-clé de retour nécessaire**: Contrairement à une fonction normale, un constructeur n'a pas besoin du mot-clé `return`. L'instance nouvellement créée est implicitement renvoyée.

1. Introduction à l'Héritage en JavaScript

L'héritage est un mécanisme qui permet à une classe d'hériter des propriétés et des méthodes d'une autre classe, créant ainsi une hiérarchie de classes. Dans JavaScript, nous avons deux approches principales pour implémenter l'héritage : les prototypes et les classes (introduites dans ES6).

Héritage en JavaScript

L'héritage permet à une classe d'hériter des propriétés et des méthodes d'une autre classe. Utilisez le mot-clé `extends` pour définir une classe enfant :

```
class Chien extends Animal {
    constructor(nom, race) {
        super(nom);
        this.race = race;
    }

    aboyer() {
        console.log("Woof! Woof!");
    }
}
```

La classe `Chien` hérite de la classe `Animal` en utilisant `extends`. La méthode `super` est utilisée pour appeler le constructeur de la classe parent.

2. Héritage avec les Classes en JavaScript

L'héritage avec les classes en JavaScript permet la création de sous-classes qui héritent des propriétés et des méthodes d'une classe de base. La syntaxe pour définir une classe dérivée est assez simple.

2.1 Héritage Simple

Exemple :

```
// Classe de base
class Forme {
  constructor(couleur) {
    this.couleur = couleur;
  }

  dessiner() {
```

```
    console.log(`Dessin d'une forme de couleur ${this.couleur}`);
  }
}

// Classe dérivée
class Cercle extends Forme {
  constructor(couleur, rayon) {
    // Appel du constructeur de la classe de base
    super(couleur);
    this.rayon = rayon;
  }

  calculerSurface() {
    return Math.PI * this.rayon ** 2;
  }
}

// Utilisation des classes
const cercle = new Cercle("rouge", 5);
cercle.dessiner();
console.log(`Surface du cercle: ${cercle.calculerSurface()}`);
```

Création d'instances

Les instances sont des objets créés à partir d'une classe. Utilisez le mot-clé `new` pour créer une instance :

```
const monChien = new Chien("Buddy", "Labrador");
monChien.direNom(); // Affiche "Je m'appelle Buddy"
monChien.aboyer();   // Affiche "Woof! Woof!"
```