

MDN web docs

[Technologies](#) ▼[Guides et références](#) ▼[Votre avis](#) ▼[Connexion](#) 

Classes

Les classes JavaScript ont été introduites avec ECMAScript 2015. Elles sont un « sucre syntaxique » par rapport à l'héritage prototypal. En effet, cette syntaxe n'introduit pas un nouveau modèle d'héritage dans JavaScript ! Elle fournit uniquement une syntaxe plus simple pour créer des objets et manipuler l'héritage.

Définir des classes

En réalité, les classes sont juste des [fonctions](#) spéciales. Ainsi, les classes sont définies de la même façon que les fonctions : par déclaration, ou par expression.

Les déclarations de classes

Pour utiliser une déclaration de classe simple, on utilisera le mot-clé `class`, suivi par le nom de la classe que l'on déclare (ici « Rectangle »).

```
1 | class Rectangle {  
2 |     constructor(hauteur, largeur) {  
3 |         this.hauteur = hauteur;  
4 |         this.largeur = largeur;  
5 |     }  
6 | }
```

Remontée des déclarations (*hoisting*)

Les [déclarations de fonctions](#) sont remontées dans le code. En revanche, ce n'est pas le cas pour les déclarations de classes. Ainsi, il est nécessaire de déclarer la classe avant de l'instancier. Dans le cas contraire, on obtient une [ReferenceError](#) :

```
1 | const p = new Rectangle(); // ReferenceError
2 |
3 | class Rectangle {}
```



Les expressions de classes

Il est possible d'utiliser des expressions de classes, nommées ou anonymes. Si on utilise un nom dans l'expression, ce nom ne sera accessible que depuis le corps de la classe via la propriété [name](#) (cette valeur ne sera pas directement accessible pour les instances).

```
1 | // anonyme
2 | let Rectangle = class {
3 |     constructor(hauteur, largeur) {
4 |         this.hauteur = hauteur;
5 |         this.largeur = largeur;
6 |     }
7 | };
8 |
9 | // nommée
10 | let Rectangle = class Rectangle {
11 |     constructor(hauteur, largeur) {
12 |         this.hauteur = hauteur;
13 |         this.largeur = largeur;
14 |     }
15 | };
```

Note : Les mêmes règles s'appliquent aux expressions de classes en ce qui concerne la remontée (*hoisting*) des classes (cf. le paragraphe précédent sur les remontées des déclarations de classe).

Corps d'une classe et définition des méthodes

Le corps d'une classe est la partie contenue entre les accolades. C'est dans cette partie que l'on définit les propriétés d'une classe comme ses méthodes et son constructeur.

Mode strict

Le corps des classes, pour les expressions et pour les déclarations de classes, est exécuté en `mode strict` (autrement dit, le constructeur, les méthodes statiques, le prototype, les accesseurs (*getters*) et mutateurs (*setters*) sont exécutés en mode strict).

Constructeur

La méthode `constructor` est une méthode spéciale qui permet de créer et d'initialiser les objet créés avec une classe. Il ne peut y avoir qu'une seule méthode avec le nom "constructor" pour une classe donnée. Si la classe contient plusieurs occurrences d'une méthode `constructor`, cela provoquera une exception `SyntaxError`.

Le constructeur ainsi déclaré peut utiliser le mot-clé `super` afin d'appeler le constructeur de la classe parente.

Méthodes de prototype

Voir aussi les définitions de méthode.

```
1  class Rectangle {
2    constructor(hauteur, largeur) {
3      this.hauteur = hauteur;
4      this.largeur = largeur;
5    }
6
7    get area() {
8      return this.calcArea();
9    }
10
11   calcArea() {
12     return this.largeur * this.hauteur;
13   }
14 }
15
16 const carré = new Rectangle(10, 10);
17
18 console.log(carré.area);
```

Méthodes statiques

Le mot-clé `static` permet de définir une méthode statique pour une classe. Les méthodes statiques sont appelées par rapport à la classe entière et non par rapport à une [instance](#) donnée (ces méthodes ne peuvent pas être appelées « depuis » une instance). Ces méthodes sont généralement utilisées sous formes d'utilitaires au sein d'applications.

```
1  class Point {
2    constructor(x, y) {
3      this.x = x;
4      this.y = y;
5    }
6
7    static distance(a, b) {
8      const dx = a.x - b.x;
9      const dy = a.y - b.y;
10     return Math.hypot(dx, dy);
11   }
12 }
13
14 const p1 = new Point(5, 5);
15 const p2 = new Point(10, 10);
16
17 console.log(Point.distance(p1, p2));
```

Gestion de `this` pour le prototype et les méthodes statiques

Lorsqu'une méthode statique ou une méthode liée au prototype est appelée sans valeur `this`, celle-ci vaudra `undefined` au sein de la fonction. Il n'y aura pas d'autodétermination de `this` (*autoboxing* en anglais). On aura le même résultat si on invoque ces fonctions dans du code non-strict car les fonctions liées aux classes sont exécutées en mode strict.

```
1  class Animal {
2    crie() {
3      return this;
4    }
5    static mange() {
6      return this;
7    }
8  }
9
```

```
10 let obj = new Animal();
11 obj.crie(); // Animal {}
12 let crie = obj.crie;
13 crie(); // undefined
14
15 Animal.mange(); // class Animal
16 let mange = Animal.mange;
17 mange(); // undefined
```

Si on écrit le code avec des fonctions traditionnelles plutôt qu'avec des classes et qu'on utilise le mode non-strict, l'autodétermination de `this` sera faite en fonction du contexte dans lequel la fonction a été appelée. Si la valeur initiale est `undefined`, `this` correspondra à l'objet global.

L'autodétermination de `this` n'a pas lieu en mode strict, la valeur `this` est passée telle quelle.

```
1 function Animal() { }
2
3 Animal.prototype.crie = function() {
4   return this;
5 }
6
7 Animal.mange = function() {
8   return this;
9 }
10
11 let obj = new Animal();
12 let crie = obj.crie;
13 crie(); // l'objet global
14
15 let mange = Animal.mange;
16 mange(); // l'objet global
```

Propriétés des instances

Les propriétés des instances doivent être définies dans les méthodes de la classe :

```
1 class Rectangle {
2   constructor(hauteur, largeur) {
3     this.hauteur = hauteur;
```

```
4 |     this.largeur = largeur;  
5 | }  
6 | }
```



Les propriétés statiques ou les données relatives au prototype doivent être définies en dehors de la déclaration comportant le corps de la classe :

```
1 | Rectangle.largeurStatique = 20;  
2 | Rectangle.prototype.largeurProto = 25;
```

Déclarations de champs

Cette fonction est expérimentale

Puisque cette fonction est toujours en développement dans certains navigateurs, veuillez consulter le [tableau de compatibilité](#) pour les préfixes à utiliser selon les navigateurs. Il convient de noter qu'une fonctionnalité expérimentale peut voir sa syntaxe ou son comportement modifié dans le futur en fonction des évolutions de la spécification.

⚠ Attention ! Les déclarations de champs publics et privés sont une  [fonctionnalité expérimentale actuellement proposée pour être intégrée dans le standard ECMAScript](#). Elle n'est pas implémentée par la majorité des navigateurs mais on peut émuler cette fonctionnalité en utilisant un système de compilation tel que  [Babel](#).

Déclarations de champs publics

En utilisant la syntaxe pour la déclaration des champs, on peut réécrire l'exemple précédent de la façon suivante :

```
1 | class Rectangle {  
2 |     hauteur = 0;  
3 |     largeur;  
4 |     constructor(hauteur, largeur) {  
5 |         this.hauteur = hauteur;  
6 |         this.largeur = largeur;  
7 |     }  
8 | }
```

En déclarant les champs en préalable, il est plus facile de comprendre la classe dans son ensemble. De plus, on s'assure que les champs soient toujours présents.

Comme on peut le voir dans cet exemple, les champs peuvent éventuellement être déclarés avec une valeur par défaut.

Déclarations de champs privés

En utilisant des champs privés, on peut revoir la définition de la façon suivante :

```
1 class Rectangle {  
2   #hauteur = 0;  
3   #largeur;  
4   constructor(hauteur, largeur){  
5     this.#hauteur = hauteur;  
6     this.#largeur = largeur;  
7   }  
8 }
```

Si on utilise les champs privés hors de la classe, cela génèrera une erreur. Ces champs ne peuvent être lus ou modifiés que depuis le corps de la classe. En évitant d'exposer des éléments à l'extérieur, on s'assure que les portions de code qui consomment cette classe n'utilisent pas ses détails internes et on peut alors maintenir la classe dans son ensemble et modifier les détails internes si besoin.

Note : Les champs privés doivent nécessairement être déclarés en premier dans les déclarations de champ.

Il n'est pas possible de créer des champs privés *a posteriori* au moment où on leur affecterait une valeur. Autrement dit, il est possible de déclarer une variable normale au moment voulu lorsqu'on lui affecte une valeur tandis que pour les champs privés, ces derniers doivent être déclarés (éventuellement initialisés) en amont, au début du corps de la classe.

Créer une sous-classe avec `extends`

Le mot-clé `extends`, utilisé dans les déclarations ou les expressions de classes, permet de créer une classe qui hérite d'une autre classe (on parle aussi de « sous-classe » ou de « classe-fille »).

```
1 class Animal {
2   constructor(nom) {
3     this.nom = nom;
4   }
5
6   parle() {
7     console.log(this.nom + ' fait du bruit.');
```

Si on déclare un constructeur dans une classe fille, on doit utiliser `super()` avant `this`.

On peut également étendre des classes plus *traditionnelles* basées sur des constructeurs fonctionnels :

```
1 function Animal (nom) {
2   this.nom = nom;
3 }
4 Animal.prototype.crie = function () {
5   console.log(this.nom + ' fait du bruit.');
```



```
18 | // Ida fait du bruit.  
    | // Ida aboie.
```

En revanche, les classes ne permettent pas d'étendre des objets classiques non-constructibles. Si on souhaite créer un lien d'héritage en un objet et une classe, on utilisera

`Object.setPrototypeOf()` :

```
1 | const Animal = {  
2 |   crie() {  
3 |     console.log(this.nom + ' fait du bruit.');4 |   }  
5 | };  
6 |  
7 | class Chien {  
8 |   constructor(nom) {  
9 |     this.nom = nom;  
10 |  }  
11 |   crie() {  
12 |     super.crie();  
13 |     console.log(this.nom + ' aboie.');14 |   }  
15 | }  
16 | Object.setPrototypeOf(Chien.prototype, Animal);  
17 |  
18 | let d = new Chien('Ida');  
19 | d.crie();  
20 | // Ida fait du bruit  
21 | // Ida aboie.
```

Le symbole `species`

Lorsqu'on souhaite renvoyer des objets `Array` avec une sous-classe `MonArray`, on peut utiliser symbole `species` pour surcharger le constructeur par défaut.

Par exemple, si, lorsqu'on utilise des méthodes comme `map()` qui renvoient le constructeur par défaut et qu'on veut qu'elles renvoient `Array` plutôt que `MonArray`, on utilisera

`Symbol.species` :

```
1 class MonArray extends Array {
2   // On surcharge species
3   // avec le constructeur Array du parent
4   static get [Symbol.species]() { return Array; }
5 }
6 let a = new MonArray(1,2,3);
7 let mapped = a.map(x => x * x);
8
9 console.log(mapped instanceof MonArray); // false
10 console.log(mapped instanceof Array);    // true
```

Utiliser `super` pour la classe parente [↗](#)

Le mot-clé `super` est utilisé pour appeler les fonctions rattachées à un objet parent.

```
1 class Chat {
2   constructor(nom) {
3     this.nom = nom;
4   }
5
6   parler() {
7     console.log(this.nom + ' fait du bruit.');
```

Les *mix-ins*

Les sous-classes abstraites ou *mix-ins* sont des modèles (*templates*) pour des classes. Une classe ECMAScript ne peut avoir qu'une seule classe parente et il n'est donc pas possible, par exemple, d'hériter de plusieurs classes dont une classe abstraite. La fonctionnalité dont on souhaite disposer doit être fournie par la classe parente.





Une fonction peut prendre une classe parente en entrée et renvoyer une classe fille qui étend cette classe parente. Cela peut permettre d'émuler les *mix-ins* avec ECMAScript.

```
1 | let calculetteMixin = Base => class extends Base {  
2 |     calc() { }  
3 | };  
4 |  
5 | let aleatoireMixin = Base => class extends Base {  
6 |     randomiseur() { }  
7 | };
```

Une classe utilisant ces *mix-ins* peut alors être écrite de cette façon :

```
1 | class Toto { }  
2 | class Truc extends calculetteMixin(aleatoireMixin(Toto)) { }
```

Spécifications

Spécification	État	Commentaires
ECMAScript 2015 (6th Edition, ECMA-262) La définition de 'Class definitions' dans cette spécification.	 ST Standard	Définition initiale.
ECMAScript 2016 (ECMA-262) La définition de 'Class definitions' dans cette spécification.	 ST Standard	
ECMAScript 2017 (ECMA-262) La définition de 'Class definitions' dans cette spécification.	 ST Standard	
ECMAScript Latest Draft (ECMA-262) La définition de 'Class definitions' dans cette spécification.	 D Projet	

Compatibilité des navigateurs [↗](#)

🔗 Take this quick survey to help us improve our browser compatibility tables

Support simple	
Chrome	49★
Edge	13
Firefox	45
IE	Non
Opera	36
Safari	9
WebView Android	?
Chrome Android	Oui
Edge Mobile	13
Firefox Android	45
Opera Android	?
Safari iOS	9
Samsung Internet Android	Oui
nodejs	6.0.0
constructor	
Chrome	49★
Edge	13
Firefox	45
IE	Non
Opera	36

Safari	9
WebView Android	?
Chrome Android	Oui
Edge Mobile	13
Firefox Android	45
Opera Android	?
Safari iOS	9
Samsung Internet Android	Oui
nodejs	6.0.0
extends	
Chrome	49★
Edge	13
Firefox	45
IE	Non
Opera	36
Safari	9
WebView Android	?
Chrome Android	Oui
Edge Mobile	13
Firefox Android	45
Opera Android	?
Safari iOS	9
Samsung Internet Android	Oui
nodejs	6.0.0
static	
Chrome	49★

Edge	13
Firefox	45
IE	Non
Opera	36
Safari	9
WebView Android	?
Chrome Android	Oui
Edge Mobile	13
Firefox Android	45
Opera Android	?
Safari iOS	9
Samsung Internet Android	Oui
nodejs	6.0.0



Support complet



Aucun support



Compatibilité inconnue



Voir les notes d'implémentation.



Une action explicite de l'utilisateur est nécessaire pour activer cette fonctionnalité.

Utilisation via l'ardoise dans Firefox [🔗](#)

Une classe ne peut pas être redéfinie. Si vous testez votre code via l'ardoise JavaScript de Firefox (Outils > Développement web > Ardoise JavaScript) et que vous exécutez à plusieurs reprises votre code avec la définition d'une classe, vous obtiendrez une exception `SyntaxError` : *redeclaration of let <class-name>*.

Pour relancer une définition, il faut utiliser le menu Exécuter > Recharger et exécuter. À ce sujet, voir le bug [bug 1428672](#).

Voir aussi

- [Les fonctions](#)
 - [Les déclarations de classes](#)
 - [Les expressions de classes](#)
 - [super](#)
 - [Billet sur les classes \(traduction en français\)](#)
 - [Champs publics et privés pour les classes \(proposition de niveau 3\)](#)
-