ES6, ES2015 : la déclaration de variables avec const, let et var

Nyalab • 2015/12/01

ES6 (aussi appelé ES2015) vous apporte de nouvelles façons de déclarer vos variables grâce à `let` et `const` mais garde aussi la déclaration par `var` dans la spécification du langage.

Première étape, on oublie tout ce qu'on sait sur `var`.

Déclarations

const

`const` vous permet de déclarer une variable à assignation unique bindée lexicalement. Bon, ça fait un peu pompeux, alors pour les devs au fond de la salle à côté du radiateur, ça veut simplement dire que vous pouvez déclarer une variable qui ne contiendra qu'une valeur et qui sera scopée au niveau du bloc.

Si vous avez déjà lu des posts ou des ressources parlant de `const`, méfiez-vous

> ce ne sont pas des vraies constantes au sens *valeur* de variable. Ce sont des constantes au niveau référence. C'est à dire que le contenu d'un tableau ou d'un objet déclaré avec `const` bloque la réassignation de la variable, mais ne rend pas la valeur immuable.

```
function fn() {
  const foo = "bar"
  if (true) {
    const foo // SyntaxError, la variable a besoin d'être assignée
    const foo = "qux"
    foo = "norf" // SyntaxError, la variable ne peut pas être réassignée
    console.log(foo)
    // "qux", la variable appartient au scope de son bloc (le "if")
  }
  console.log(foo)
  // "bar", la variable appartient au scope de la fonction "fn"
}
```

Le fonctionnement `const` peut être utilisé de manière cool dans le cas d'itérables :

```
function fn() {
  const arr = [1, 2, 3];
  for (const el of arr) {
    console.log(el);
  }
}
```

En effet, on pourrait croire qu'un `let` doit être utilisé ici, mais la déclaration est évaluée à chaque passage de l'itérateur, `const` est donc un meilleur choix !

let

`let` vous permet de faire pareil que `const` mais sans la contrainte d'assignation unique. Vous devriez donc instinctivement voir que les cas d'utilisation pour `let` sont les mêmes que ceux de `var`, son ancêtre. D'ailleurs, vous entendrez souvent : `let` est le nouveau `var` (*`let` is the new `var`*). C'est en partie vrai car il est capable de faire les mêmes choses, mais en mieux, car il a cette caractéristique d'être scopé au bloc courant.

```
function fn() {
  let foo = "bar";
  var foo2 = "bar";
  if (true) {
    let foo; // pas d'erreur, foo === undefined
    var foo2;
    // Attention, les déclarations "var" ne sont pas scopées au niveau bloc
    // foo2 est en réalité écrasé !
    foo = "qux";
    foo2 = "qux";
    console.log(foo);
    // "qux", la variable appartient au scope de son blocs (le "if")
    console.log(foo2);
    // "qux"
  }
  console.log(foo);
  // "bar", la variable appartient au scope de son bloc (la fonction "fn")
  console.log(foo2);
  // "qux"
}
```

Vous pouvez par exemple utiliser `let` pour vos boucles, la variable servant à l'itération est désormais scopée au niveau de cette boucle et n'entrera pas en conflit avec votre code autour. Plus de problème de `i` déjà pris !

```
function fn2() {
  let i = 0;
  for (let i = 1; i < 10; i++) {
    console.log(i);
  }
  console.log(i);
  // 0

  for (let j = i; j < 10; j++) {}
  console.log(j);
  // j is not defined
}
fn2(); // 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
```

Note : l'exemple avec `const` dans une boucle `for ... of` ne peut être reproduit ici. En effet, la boucle `for` classique est impérative, et la déclaration n'est effectuée qu'une seule fois au début de la boucle. Un `const` n'est donc pas utilisable.

var

On a vu `const`, on a vu `let`. Avec ces deux nouveaux outils, il ne reste pas de grande place pour `var`. À mon avis, le seul cas d'utilisation valable pour `var` est lors de l'utilisation de `try/catch`, et ce n'est pas dans le cadre d'un bug, mais juste de syntaxe et de préférence (exemple).

Piège du hoisting et de la TDZ (*Temporal Dead Zone*)

Pour rappel, JavaScript possède un mécanisme de hoisting, par exemple, vous pouvez écrire :

```
function fn() {
  console.log(foo); // undefined (au lieu de ReferenceError)
  var foo = "bar";
}
```

Concrètement, le moteur d'exécution JavaScript va lire toutes les déclarations et remonter celles avec `var` au début du scope de votre fonction (attention, cela concerne les déclarations, pas les affectations).

`let` et `const` ne bénéficient pas de ce mécanisme de hoisting, ce qui peut mener à des problèmes de TDZ (*Temporal Dead Zone*). Vu que la déclaration de votre variable n'est pas remontée au scope de la fonction, il existe un moment où votre variable n'existe pas. Ce moment, c'est la TDZ.

```
function fn() {
  console.log(foo);
  // ReferenceError, on est dans la TDZ pour la variable foo
  let foo = "bar";
}
```

Outro

Comment choisir quelle déclaration de variable utiliser ? C'est très simple :

- Utilisez une déclaration par `const` (99% du temps, c'est le bon choix)
- Si au fil de votre code vous changez sa valeur, modifiez pour un `let` (1%)
- Si vous avez trouvé le pire cas d'utilisation du monde, changez pour un `var` (je vous laisse faire le calcul)

← Articles

Clem • il y a 6 ans • edited

Juste une petite erreur concernant le TDZ, le `console.log()` n'affichera pas `"bar"` mais `undefined`, en effet la déclaration et remontée en haut du scope mais pas l'instanciation. ce qui revient à écrire ceci :

```
function fn() {
```