Le mot-clé this

Qu'est-ce que `this` ?

`this` est un mot clé `JavaScript` permettant d'accéder à l'objet représentant le contexte d'exécution.

Attention ! Ici, la valeur de `this` dépend du contexte d'exécution et non pas de l'environnement lexical !

Autrement dit, la valeur de `this` dépend de l'endroit de l'exécution, c'est-à-dire de l'endroit où il est appelé.

Le mode `strict`

Le mode `strict` existe depuis `ECMAScript 5.1` (année 2011).

Il active le mode restrictif de `JavaScript`, qui se comporte différemment dans certains cas. C'est une amélioration recommandée du langage.

Les principales différences sont : des erreurs silencieuses ont été éliminées (c'est-à-dire que vous avez une erreur dans votre console et le code stoppe son exécution ce qui est une bonne chose pour pouvoir prédire son comportement), il permet aux moteurs `JavaScript` d'être plus performants en leur donnant la possibilité de faire des optimisations supplémentaires, enfin il permet de nombreuses sécurisations du langage pour l'exécution du code de librairies tierces.

Quelques exemples qui sont possibles en mode normal mais lèvent des erreurs en mode `strict` :

```
"use strict";

a = 17; // ReferenceError
// En mode normal une var est déclarée automatiquement.

undefined = 42; //  Cannot assign to read only property.
// En mode normal pas d’erreur, mais pas d’affectation.

delete Object.prototype; // Cannot delete property 'prototype'
// En mode normal pas d’erreur, mais pas de suppression.

function somme(a, a, b) { // Duplicate parameter name not allowed
}
// En mode normal pas d’erreur, a aura la valeur du second argument
```

```
false.true = ""; // TypeError: Cannot create property
// En mode normal, pas d'erreur. Simplement ignoré
```

Pour l'utiliser il suffit d'ajouter `"use strict";` en haut de votre programme `JavaScript`.

A noter que par défaut, la modularisation utilisée par `Webpack` active le mode strict.

`this` dans le contexte global

Dans le contexte global, `this` a pour valeur l'objet global. Dans un navigateur, ce sera `window`. Dans module `Node.js` cela retournera un objet vide.

C'est totalement indépendant du mode strict :

```
"use strict";

console.log(this === window);
```

`this` avec des fonctions

Il y a deux manières principales d'utiliser des fonctions en `JavaScript`, la valeur de `this` va dépendre du contexte d'exécution.

Appel simple

Un appel simple est l'exécution d'une fonction, et ce, peu importe que la fonction soit imbriquée dans une autre fonction.

Dans le mode normal, `this` sera l'objet global.

Donc nous aurons dans un navigateur :

```
function test () {
    console.log(this === window);
}

test(); // true
```

En mode `strict`, ce ne sera pas le cas. Pour éviter des failles de sécurité lors de l'utilisation de librairies tierces, `this` sera `undefined`. En effet, dans le cas contraire, les librairies tierces pourraient accéder à l'objet global et injecter du code malveillant ou récupérer des informations.

```
"use strict";
```

```
function test () {
    console.log(this === undefined);
}

test(); // true
```

Méthode d'objet

Lorsque la fonction invoquée est une méthode d'un objet, `this` aura pour valeur la référence de l'objet, en mode strict et en mode normal :

```
"use strict";

const objet = {
  maMethode() {
    console.log(this === objet);
  }
};

objet.maMethode(); // true
```

C'est le cas même si la fonction est déclarée d'abord sur l'objet global. Ce qui importe c'est qu'au moment de l'exécution elle soit appelée comme méthode d'objet :

```
"use strict";

function test() {
  console.log(this === objet);
}

const objet = {
  maMethode: test
};

objet.maMethode(); // true
```

Méthode passée en argument d'une fonction

En revanche, lorsqu'une méthode est passée comme argument d'une fonction, nous avons une invocation simple.

En effet, la fonction est passée par référence et elle perd son attachement à son objet :

```
"use strict";

const objet = {
  maMethode() {
    console.log(this === undefined);
  }
};

function a(fonction) {
  fonction();
}

a(objet.maMethode); // true
```

Notes

Trois choses à retenir : premièrement, nous sommes très loin d'avoir vu tout `this`, nous verrons d'autres comportements lorsque nous aurons vu les classes, les prototypes, les constructeurs etc.

Deuxièmement, comme nous sommes en mode `strict` avec `Webpack` la valeur de `this` est assez prévisible.

Troisièmement nous verrons que la valeur de `this` est différente avec les fonctions fléchées.