Explication de la portée des variables JavaScript et du levage

Dans ce billet, nous allons apprendre la portée des variables et le hissage des variables en JavaScript et toutes les particularités de ces deux éléments.

Nous devons comprendre le fonctionnement de la portée des variables et de la collecte des variables en JavaScript, si nous voulons bien comprendre JavaScript. Ces concepts peuvent sembler simples, mais ils ne le sont pas. Il existe des subtilités importantes que nous devons comprendre si nous voulons prospérer et exceller en tant que développeurs JavaScript.

Portée de la variable

La portée d'une variable est le contexte dans lequel la variable existe. L'étendue précise à partir de quel endroit vous pouvez accéder à une variable et si vous avez accès à la variable dans ce contexte.

Les variables ont soit une portée locale, soit une portée globale.

Variables locales (portée au niveau de la fonction)

Contrairement à la plupart des langages de programmation, JavaScript n'a pas de portée au niveau du bloc (variables dont la portée est limitée aux accolades environnantes) ; JavaScript a plutôt une portée au niveau de la fonction. Les variables déclarées dans une fonction sont des variables locales et ne sont accessibles que dans cette fonction ou par des fonctions à l'intérieur de cette fonction. Consultez mon article sur les fermetures pour en savoir plus sur l'accès aux variables des fonctions externes à partir des fonctions internes.

Démonstration de la portée du niveau de fonction (Function level scope)

```
var name = "Richard";

function showName () {
    var name = "Jack"; // variable locale ; uniquement accessible dans cette fonction showName
    console.log (name); // Jack
}
console.log (name); // Richard: the global variable
```

Aucune portée au niveau du bloc (No block level scope)

```
var name = "Richard";
// les blocs de cette instruction if ne créent pas de contexte local pour la variable de nom
if (name) {
    name = "Jack"; // ce nom est la variable de nom global et il est changé en "Jack" ici.
    console.log (name); // Jack : toujours la variable globale
}

// Ici, la variable de nom est la même variable de nom global, mais elle a été modifiée dans l'instruction if.
console.log (name); // Jack
```

- **Si vous ne déclarez pas vos variables locales, les ennuis sont proches**

  Déclarez toujours vos variables locales avant de les utiliser. En fait, vous devriez utiliser JSHint pour vérifier votre code afin de détecter les erreurs de syntaxe et les guides de style. Voici le problème que pose la non-déclaration des variables locales :

```
// Si vous ne déclarez pas vos variables locales avec le mot-clé var, elles font partie de la portée globale.
var name = "Michael Jackson";

function showCelebrityName () {
    console.log (name);
}

function showOrdinaryPersonName () {
    name = "Johnny Evers";
    console.log (name);
}
showCelebrityName (); // Michael Jackson

// name n'est pas une variable locale, elle change simplement la variable globale name
showOrdinaryPersonName (); // Johnny Evers

// La variable globale est maintenant Johnny Evers, et non plus le nom de la célébrité.
showCelebrityName (); // Johnny Evers

// La solution consiste à déclarer votre variable locale avec le mot-clé var.
function showOrdinaryPersonName () {
    var name = "Johnny Evers"; // Maintenant, le nom est toujours une variable locale et il ne remplacera pas la variable globale.
    console.log (name);
}
```

- **Les variables locales ont la priorité sur les variables globales dans les fonctions**

  Si vous déclarez une variable globale et une variable locale portant le même nom, la variable locale aura la priorité lorsque vous tenterez d'utiliser la variable dans une fonction (portée locale) :

```
var name = "Paul";

function users () {
    // Ici, la variable de nom est locale et elle a la priorité sur la variable de même nom dans la portée globale.
var name = "Jack";

// La recherche du nom commence ici, à l'intérieur de la fonction, avant de chercher à l'extérieur de la fonction, dans la portée globale.
console.log (name);
}

users (); // Jack
```

Variables globales

Toutes les variables déclarées en dehors d'une fonction se trouvent dans la portée globale. Dans le navigateur, qui nous concerne en tant que développeurs frontaux, le contexte ou la portée globale est l'objet fenêtre (window) (ou le document HTML entier).

- Toute variable déclarée ou initialisée en dehors d'une fonction est une variable globale, et elle est donc disponible pour l'ensemble de l'application. Par exemple :

```
// Pour déclarer une variable globale, vous pouvez effectuer l'une des opérations suivantes :
var myName = "Richard";

// ou même
firstName = "Richard";

// ou
var name; //
name;
```

  Il est important de noter que toutes les variables globales sont attachées à l'objet fenêtre. Ainsi, toutes les variables globales que nous venons de déclarer peuvent être accédées sur l'objet window comme ceci :

```
console.log(window.myName); // Richard;

 // or
console.log("myName" in window); // true
console.log("firstName" in window); // true
```

- Si une variable est initialisée (on lui attribue une valeur) sans avoir été déclarée au préalable avec le mot-clé var, elle est automatiquement ajoutée au contexte global et il s'agit donc d'une variable globale :

```
function showAge () {
    // Age est une variable globale car elle n'a pas été déclarée avec le mot-clé var dans cette fonction.
    age = 90;
    console.log(age);//
}

showAge (); // 90

// L'âge est dans le contexte global, il est donc disponible ici aussi.
console.log(age); // 90
```

Démonstration des variables qui sont dans la portée globale même si elles semblent être autrement :

```
// Les deux variables firstName sont dans la portée globale, même si la seconde est entourée d'un bloc {}.
var firstName = "Richard";
{
var firstName = "Bob";
}

// Pour réitérer : JavaScript n'a pas de portée au niveau du bloc

// La deuxième déclaration de firstName redéclare et écrase simplement la première.
console.log (firstName); // Bob
```

Un autre exemple

```
for (var i = 1; i <= 10; i++) {
    console.log (i); // outputs 1, 2, 3, 4, 5, 6, 7, 8, 9, 10;
};

// La variable i est une variable globale et elle est accessible
// dans la fonction suivante avec la dernière valeur qui lui a été attribuée ci-dessus
function aNumber () {
console.log(i);
}

// La variable i dans la fonction aNumber ci-dessous est la variable globale i qui a
// été modifiée dans la boucle for ci-dessus. Sa dernière valeur était 11,
// définie juste avant la sortie de la boucle for :
aNumber ();  // 11
```

- **Les variables setTimeout sont exécutées dans la portée globale**
  Notez que toutes les fonctions de setTimeout sont exécutées dans la portée globale. C'est un point délicat ; considérez ceci :

```
 // L'utilisation de l'objet "this" dans la fonction setTimeout
 // fait référence à l'objet Window, et non à myObj.

var highValue = 200;
var constantVal = 2;
var myObj = {
    highValue: 20,
    constantVal: 5,
    calculateIt: function () {
 setTimeout (function  () {
    console.log(this.constantVal * this.highValue);
}, 2000);
    }
}

// L'objet "this" dans la fonction setTimeout a utilisé les variables
// globales highValue et constantVal, car la référence à "this" dans
// la fonction setTimeout fait référence à l'objet global window,
// et non à l'objet myObj comme on pourrait s'y attendre.

myObj.calculateIt(); // 400
// C'est un point important à retenir.
```

- **Ne pas polluer la portée globale**
  Si vous voulez devenir un maître du JavaScript, ce que vous souhaitez certainement faire, vous devez savoir qu'il est important d'éviter de créer de nombreuses variables dans la portée globale, comme ceci :

```
// Ces deux variables sont dans la portée globale et elles ne devraient pas être ici.
var firstName, lastName;

function fullName () {
    console.log ("Full Name: " + firstName + " " + lastName );
}
```

Voici le code amélioré et la manière correcte d'éviter de polluer la portée globale.

```
// Déclarez les variables à l'intérieur de la fonction où elles sont des variables locales.

function fullName () {
    var firstName = "Michael", lastName = "Jackson";

    console.log ("Full Name: " + firstName + " " + lastName );
}
```

Dans ce dernier exemple, la fonction fullName se trouve également dans la portée globale.

Hissage des variables

Toutes les déclarations de variables sont hissées (levées et déclarées) au sommet de la fonction, si elles sont définies dans une fonction, ou au sommet du contexte global, si elles sont hors fonction.

Il est important de savoir que seules les déclarations de variables sont hissées au sommet, et non les initialisations ou les affectations de variables (lorsqu'une valeur est attribuée à la variable).

Exemple de levage de variable :

```
function showName () {
console.log ("First Name: " + name);
var name = "Ford";
console.log ("Last Name: " + name);
}

showName ();
// First Name: undefined
// Last Name: Ford

// La raison pour laquelle undefined s'imprime en premier est que le nom de la variable locale a été hissé au sommet de la fonction
// Ce qui signifie que c'est cette variable locale qui est appelée la première fois.
// C'est ainsi que le code est réellement traité par le moteur JavaScript :

function showName () {
    var name; // le nom est hissé (notez que c'est indéfini à ce stade, puisque l'affectation se fait ci-dessous)
console.log ("First Name: " + name); // First Name: undefined

name = "Ford"; // une valeur est attribuée au nom

// now name is Ford
console.log ("Last Name: " + name); // Last Name: Ford
}
```

La déclaration de fonction prévaut sur la déclaration de variable lorsqu'elle est hissée

La déclaration de fonction et la déclaration de variable sont toutes deux hissées au sommet de la portée qui les contient. La déclaration de fonction est prioritaire sur les déclarations de variable (mais pas sur l'affectation de variable). Comme indiqué ci-dessus, l'affectation de variable n'est pas hissée, tout comme l'affectation de fonction. Pour rappel, il s'agit d'une affectation de fonction : var myFunction = function () {}. Voici un exemple de base pour démontrer :

```
 // La variable et la fonction sont toutes deux nommées myName
var myName;
function myName () {
console.log ("Rich");
}

// La déclaration de la fonction remplace le nom de la variable
console.log(typeof myName); // function
```

```
 // Mais dans cet exemple, l'affectation de la variable
 // prévaut sur la déclaration de la fonction.
var myName = "Richard"; // Il s'agit de l'affectation de la variable (initialisation)
//qui prévaut sur la déclaration de la fonction.

function myName () {
console.log ("Rich");
}

console.log(typeof myName); // string
```

Il est important de noter que les expressions de fonctions, comme l'exemple ci-dessous, ne sont pas hissées.

```
var myName = function () {
console.log ("Rich");
}
```

En mode strict, une erreur se produit si vous attribuez une valeur à une variable sans avoir préalablement déclaré la variable. Déclarez toujours vos variables.