Comprendre les fermetures JavaScript avec facilité (Javascript Closures)

Les fermetures permettent aux programmeurs JavaScript d'écrire un meilleur code. Créatives, expressives et concises. Nous utilisons fréquemment les fermetures en JavaScript et, quelle que soit votre expérience en JavaScript, vous les rencontrerez sans doute à plusieurs reprises. Bien sûr, les fermetures peuvent sembler complexes et hors de votre portée, mais après avoir lu cet article, les fermetures seront beaucoup plus faciles à comprendre et donc plus attrayantes pour vos tâches quotidiennes de programmation JavaScript.

Il s'agit d'un article relativement court (et agréable) sur les détails des fermetures en JavaScript. Avant de poursuivre votre lecture, vous devez vous familiariser avec la portée des variables en JavaScript, car pour comprendre les fermetures, vous devez comprendre la portée des variables en JavaScript.

Qu'est-ce qu'une fermeture ?

Une fermeture est une fonction interne qui a accès à la chaîne de portée des variables de la fonction externe (englobante). La fermeture a trois chaînes de portée : elle a accès à sa propre portée (variables définies entre ses crochets), elle a accès aux variables de la fonction externe et elle a accès aux variables globales.

La fonction interne a accès non seulement aux variables de la fonction externe, mais aussi aux paramètres de cette dernière. Notez toutefois que la fonction interne ne peut pas appeler l'objet arguments de la fonction externe, même si elle peut appeler directement les paramètres de la fonction externe.

Vous créez une fermeture en ajoutant une fonction à l'intérieur d'une autre fonction.

```
function showName (firstName, lastName) {
var nameIntro = "Your name is ";
    // cette fonction interne a accès aux variables de la fonction externe, y compris le paramètre
function makeFullName () {
return nameIntro + firstName + " " + lastName;
}

return makeFullName ();
}

showName ("Michael", "Jackson"); // Your name is Michael Jackson*
```

Les fermetures sont largement utilisées dans Node.js ; elles sont les chevaux de bataille de l'architecture asynchrone et non bloquante de Node.js. Les fermetures sont également fréquemment utilisées dans jQuery et dans à peu près tous les morceaux de code JavaScript que vous lisez.

Un exemple classique de fermeture en jQuery :

```
$(function() {

var selections = [];
$(".niners").click(function() { // cette fermeture a accès à la variable de sélection
selections.push (this.prop("name")); // mettre à jour la variable des sélections dans la portée de la fonction externe
});

});
```

Règles et effets secondaires des fermetures

1. **Les fermetures ont accès aux variables de la fonction externe même après le retour de cette dernière :**
   L'une des caractéristiques les plus importantes et les plus délicates des fermetures est que la fonction interne a toujours accès aux variables de la fonction externe même après le retour de cette dernière. Oui, vous avez bien lu. Lorsque les fonctions JavaScript s'exécutent, elles utilisent la même chaîne de portée que celle qui était en vigueur lors de leur création. Cela signifie que même après le retour de la fonction externe, la fonction interne a toujours accès aux variables de la fonction externe. Par conséquent, vous pouvez appeler la fonction interne plus tard dans votre programme. Cet exemple le démontre :

```
function celebrityName (firstName) {
    var nameIntro = "This celebrity is ";
    // cette fonction interne a accès aux variables de la fonction externe, y compris le paramètre
   function lastName (theLastName) {
        return nameIntro + firstName + " " + theLastName;
    }
    return lastName;
}

var mjName = celebrityName ("Michael"); // A ce stade, la fonction extérieure du nom de la célébrité est revenue.

// La fermeture (lastName) est appelée ici après le retour de la fonction externe ci-dessus.
// Cependant, la fermeture a toujours accès aux variables et aux paramètres de la fonction externe.
mjName ("Jackson"); // This celebrity is Michael Jackson
```

2. **Les fermetures stockent les références aux variables de la fonction externe ;**
   elles ne stockent pas la valeur réelle. Les fermetures deviennent plus intéressantes lorsque la valeur de la variable de la fonction externe change avant que la fermeture ne soit appelée. Cette puissante fonctionnalité peut être exploitée de manière créative, comme dans cet exemple de variables privées démontré pour la première fois par Douglas Crockford :

```
function celebrityID () {
    var celebrityID = 999;
    // Nous retournons un objet avec quelques fonctions internes
    // Toutes les fonctions internes ont accès aux variables de la fonction externe.
    return {
        getID: function ()  {
            // Cette fonction interne renverra la variable CelebrityID mise à jour.
            // Elle renvoie la valeur actuelle de celebrityID, même après que la fonction changeTheID l'ait modifiée.
          return celebrityID;
        },
        setID: function (theNewID)  {
            // Cette fonction interne va changer la variable de la fonction externe à tout moment.
            celebrityID = theNewID;
        }
    }

}

var mjID = celebrityID (); // A ce stade, la fonction extérieure de la célébrité est revenue.
mjID.getID(); // 999
mjID.setID(567); // Change la variable de la fonction externe
mjID.getID(); // 567: Il retourne la variable CelebrityId mise à jour
```

3. **Des fermetures qui tournent mal**

Parce que les fermetures ont accès aux valeurs mises à jour des variables de la fonction externe, elles peuvent également entraîner des bogues lorsque la variable de la fonction externe change avec une boucle for. Ainsi :

```
// Cet exemple est expliqué en détail ci-dessous (juste après cette case de code).
function celebrityIDCreator (theCelebrities) {
    var i;
    var uniqueID = 100;
    for (i = 0; i < theCelebrities.length; i++) {
      theCelebrities[i]["id"] = function ()  {
        return uniqueID + i;
      }
    }

    return theCelebrities;
}

var actionCelebs = [{name:"Stallone", id:0}, {name:"Cruise", id:0}, {name:"Willis", id:0}];

var createIdForActionCelebs = celebrityIDCreator (actionCelebs);

var stalloneID = createIdForActionCelebs [0];console.log(stalloneID.id()); // 103
```

Dans l'exemple précédent, au moment où les fonctions anonymes sont appelées, la valeur de i est 3 (la longueur du tableau, puis elle s'incrémente). Le nombre 3 a été ajouté à l'uniqueID pour créer 103 pour TOUS les celebritiesID. Ainsi, chaque position dans le tableau retourné obtient id = 103, au lieu des 100, 101, 102 prévus.

La raison pour laquelle cela s'est produit est que, comme nous l'avons vu dans l'exemple précédent, la fermeture (la fonction anonyme dans cet exemple) a accès aux variables de la fonction externe par référence, et non par valeur. Ainsi, tout comme l'exemple précédent montrait que nous pouvions accéder à la variable mise à jour avec la fermeture, cet exemple a également accédé à la variable i lorsqu'elle a été modifiée, puisque la fonction externe exécute la totalité de la boucle for et renvoie la dernière valeur de i, qui est 103.

Pour corriger cet effet secondaire (bogue) dans les fermetures, vous pouvez utiliser une expression de fonction immédiatement invoquée (IIFE), telle que la suivante :

```
function celebrityIDCreator (theCelebrities) {
    var i;
    var uniqueID = 100;
    for (i = 0; i < theCelebrities.length; i++) {
        theCelebrities[i]["id"] = function (j)  { // la variable paramétrique j est le i passé lors de l'invocation de ce IIFE
            return function () {
                return uniqueID + j; // chaque itération de la boucle for passe la valeur courante de i dans ce IIFE et il sauvegarde la valeur correcte dans le tableau.
            } () // En ajoutant () à la fin de cette fonction, nous l'exécutons immédiatement et retournons juste la valeur de uniqueID + j, au lieu de retourner une fonction.
        } (i); // invoquer immédiatement la fonction en passant la variable i comme paramètre
    }

    return theCelebrities;
}

var actionCelebs = [{name:"Stallone", id:0}, {name:"Cruise", id:0}, {name:"Willis", id:0}];

var createIdForActionCelebs = celebrityIDCreator (actionCelebs);

var stalloneID = createIdForActionCelebs [0];
console.log(stalloneID.id); // 100

var cruiseID = createIdForActionCelebs [1];console.log(cruiseID.id); // 101
```