Javascript Callbacks : Dossier MDN Web Docs

Comprendre les fonctions de rappel JavaScript (callback) et les utiliser

(Apprendre les fonctions d'ordre supérieur JavaScript, alias fonctions de rappel)

En JavaScript, les fonctions sont des objets de première classe, c'est-à-dire que les fonctions sont du type Object et peuvent être utilisées de la même manière que n'importe quel autre objet (String, Array, Number, etc.) puisqu'elles sont elles-mêmes des objets. Elles peuvent être "stockées dans des variables, transmises comme arguments à des fonctions, créées dans des fonctions et retournées par des fonctions "1.

Les fonctions étant des objets de première classe, nous pouvons passer une fonction en tant qu'argument dans une autre fonction et exécuter ultérieurement cette fonction passée ou même la renvoyer pour qu'elle soit exécutée plus tard. C'est l'essence même de l'utilisation des fonctions de rappel en JavaScript. Dans la suite de cet article, nous allons tout apprendre sur les fonctions callback en JavaScript. Les fonctions de rappel sont probablement la technique de programmation fonctionnelle la plus utilisée en JavaScript, et vous pouvez les trouver dans à peu près chaque morceau de code JavaScript et jQuery, pourtant elles restent mystérieuses pour de nombreux développeurs JavaScript. Le mystère n'existera plus, lorsque vous aurez terminé la lecture de cet article.

Les fonctions de rappel sont dérivées d'un paradigme de programmation connu sous le nom de programmation fonctionnelle. À un niveau fondamental, la programmation fonctionnelle spécifie l'utilisation de fonctions comme arguments. La programmation fonctionnelle était - et est toujours, bien que dans une moindre mesure aujourd'hui - considérée comme une technique ésotérique réservée à des programmeurs chevronnés spécialement formés.

Heureusement, les techniques de programmation fonctionnelle ont été élucidées afin que les simples mortels comme vous et moi puissent les comprendre et les utiliser facilement. L'une des principales techniques de la programmation fonctionnelle est celle des fonctions de rappel. Comme vous le lirez bientôt, l'implémentation des fonctions de rappel est aussi simple que de passer des variables ordinaires comme arguments. Cette technique est si simple que je me demande pourquoi elle est surtout abordée dans les sujets avancés sur le JavaScript.

Qu'est-ce qu'une fonction de rappel (callback) ou une fonction d'ordre supérieur (higher-order) ?

Une fonction de rappel, également appelée fonction d'ordre supérieur, est une fonction qui est transmise à une autre fonction (appelons cette autre fonction "otherFunction") en tant que paramètre, et la fonction de rappel est appelée (ou exécutée) dans l'otherFunction. Une fonction de

rappel est essentiellement un modèle (une solution établie à un problème commun), et par conséquent, l'utilisation d'une fonction de rappel est également connue comme un modèle de rappel.

Considérez cette utilisation courante d'une fonction de rappel dans jQuery :

```
////Notez que l'élément du paramètre de la méthode click est une fonction et non une variable.
//L'élément est une fonction de rappel (dite callback en anglais)
$("#btn_1").click(function() {
  alert("Btn 1 Clicked");
});
```

Comme vous le voyez dans l'exemple précédent, nous passons une fonction comme paramètre à la méthode click. Et la méthode click appellera (ou exécutera) la fonction de rappel que nous lui avons passée. Cet exemple illustre une utilisation typique des fonctions de rappel en JavaScript, et une fonction largement utilisée dans jQuery.

Ruminons cet autre exemple classique d'utilisation des fonctions de rappel en JavaScript de base :

```
var friends = ["Mike", "Stacy", "Andy", "Rick"];

friends.forEach(function (eachName, index){
console.log(index + 1 + ". " + eachName); // 1. Mike, 2. Stacy, 3. Andy, 4. Rick
});
```

Une fois encore, notez la façon dont nous passons une fonction anonyme (une fonction sans nom) à la méthode forEach en tant que paramètre.

Jusqu'à présent, nous avons passé des fonctions anonymes comme paramètre à d'autres fonctions ou méthodes. Comprenons maintenant comment fonctionnent les fonctions de rappel avant de nous pencher sur des exemples plus concrets et de commencer à créer nos propres fonctions de rappel.

Comment fonctionnent les fonctions de rappel ?

Nous pouvons faire circuler les fonctions comme des variables, les retourner dans des fonctions et les utiliser dans d'autres fonctions. Lorsque nous transmettons une fonction de rappel comme argument à une autre fonction, nous ne transmettons que la définition de la fonction. Nous n'exécutons pas la fonction dans le paramètre. En d'autres termes, nous ne transmettons pas la fonction avec la paire de parenthèses d'exécution () qui suit, comme nous le faisons lorsque nous exécutons une fonction.

Et puisque la fonction contenant la fonction de rappel a la fonction de rappel dans son paramètre comme une définition de fonction, elle peut exécuter le rappel à tout moment.

Notez que la fonction de rappel n'est pas exécutée immédiatement. Elle est "rappelée" (d'où son nom) à un moment précis dans le corps de la fonction contenant la fonction. Ainsi, même si le premier exemple de jQuery ressemblait à ceci :

```
//La fonction anonyme n'est pas exécutée à cet endroit dans le paramètre.
//L'élément est une fonction de rappel
$("#btn_1").click(function() {
  alert("Btn 1 Clicked");
});
```

la fonction anonyme sera appelée plus tard dans le corps de la fonction. Même sans nom, il est toujours possible d'y accéder ultérieurement via l'objet d'arguments de la fonction qui la contient.

Les fonctions de rappel sont des fermetures

Lorsque nous passons une fonction de rappel comme argument à une autre fonction, le rappel est exécuté à un moment donné dans le corps de la fonction qui le contient, comme si le rappel était défini dans la fonction qui le contient. Cela signifie que le callback est une fermeture. Pour en savoir plus sur les fermetures, lisez mon article intitulé "Understanding JavaScript Closures With Ease". Comme nous le savons, les fermetures ont accès à la portée de la fonction contenante, de sorte que la fonction de rappel peut accéder aux variables des fonctions contenantes, et même aux variables de la portée globale.

Principes de base pour l'implémentation des fonctions de rappel

Bien qu'elles ne soient pas compliquées, les fonctions de rappel comportent quelques principes importants à connaître lors de leur mise en œuvre.

Utilisez des fonctions nommées OU anonymes comme fonctions de rappel

Dans les exemples précédents de jQuery et de forEach, nous avons utilisé des fonctions anonymes définies dans le paramètre de la fonction qui les contient. C'est l'un des modèles courants d'utilisation des fonctions de rappel. Un autre modèle populaire consiste à déclarer une fonction nommée et à transmettre le nom de cette fonction au paramètre. Considérez ceci :

```
// variable globale
var allUserData = [];

// fonction générique logStuff qui imprime sur la console
function logStuff (userData) {
    if ( typeof userData === "string")
    {
        console.log(userData);
    }
    else if ( typeof userData === "object")
    {
        for (var item in userData) {
            console.log(item + ": " + userData[item]);
        }

    }

}

// Une fonction qui prend deux paramètres, le dernier étant une fonction de rappel.
function getInput (options, callback) {
    allUserData.push (options);
    callback (options);

}

// Lorsque nous appelons la fonction getInput, nous passons logStuff comme paramètre.
// Ainsi, logStuff sera la fonction qui sera rappelée (ou exécutée) dans la fonction getInput.
getInput ({name:"Rich", speciality:"JavaScript"}, logStuff);
//  name: Rich
// speciality: JavaScript
```

Passer des paramètres aux fonctions de rappel

Puisque la fonction de rappel est juste une fonction normale lorsqu'elle est exécutée, nous pouvons lui passer des paramètres. Nous pouvons passer n'importe quelle propriété de la fonction conteneur (ou propriété globale) comme paramètre à la fonction de rappel. Dans l'exemple précédent, nous passons des options comme paramètre à la fonction de rappel. Passons une variable globale et une variable locale :

```
//Global variable
var generalLastName = "Clinton";

function getInput (options, callback) {
    allUserData.push (options);
// Passez la variable globale generalLastName à la fonction de rappel.
    callback (generalLastName, options);
}
```

Assurez-vous que la fonction de rappel est une fonction avant de l'exécuter

Il est toujours sage de vérifier que la fonction de rappel passée en paramètre est bien une fonction avant de l'appeler. En outre, il est bon de rendre la fonction de rappel facultative.

Refactorons la fonction getInput de l'exemple précédent pour nous assurer que ces vérifications sont en place.

```
function getInput(options, callback) {
    allUserData.push(options);

    // Assurez-vous que le callback est une fonction
    if (typeof callback === "function") {
    // Appelez-le, puisque nous avons confirmé qu'il est appelable.
        callback(options);
    }
}
```

Sans cette vérification, si la fonction getInput est appelée sans la fonction de rappel en tant que paramètre ou si une non-fonction est passée à la place de la fonction, notre code produira une erreur d'exécution.

Problème lors de l'utilisation de méthodes avec l'objet this comme callback

Lorsque la fonction de callback est une méthode qui utilise l'objet this, nous devons modifier la façon dont nous exécutons la fonction de callback pour préserver le contexte de l'objet this. Sinon, l'objet this pointera vers l'objet global de la fenêtre (dans le navigateur), si le callback a été transmis à une fonction globale. Ou bien il pointera vers l'objet de la méthode qui le contient. Explorons cela en code :

```
// Define an object with some properties and a method
// We will later pass the method as a callback function to another function
var clientData = {
    id: 094545,
    fullName: "Not Set",
    // setUserName is a method on the clientData object
    setUserName: function (firstName, lastName)  {
        // this refers to the fullName property in this object
      this.fullName = firstName + " " + lastName;
    }
}

function getUserInput(firstName, lastName, callback)  {
    // Faites d'autres choses pour valider le prénom et le nom ici.

    // Maintenant, enregistrez les noms
    callback (firstName, lastName);
}
```

Dans l'exemple de code suivant, lorsque clientData.setUserName est exécuté, this.fullName ne définit pas la propriété fullName de l'objet clientData. Au lieu de cela, il définira fullName sur l'objet

window, puisque getUserInput est une fonction globale. Cela se produit parce que l'objet this de la fonction globale pointe vers l'objet window.

```
getUserInput ("Barack", "Obama", clientData.setUserName);

console.log (clientData.fullName);// Non défini

// La propriété fullName a été initialisée sur l'objet window.
console.log (window.fullName); // Barack Obama
```

Utiliser la fonction Call ou Apply pour préserver this

Nous pouvons résoudre le problème précédent en utilisant la fonction Call ou Apply (nous en parlerons plus tard dans un article de blog complet). Pour l'instant, sachez que chaque fonction en JavaScript possède deux méthodes : Call et Apply. Ces méthodes sont utilisées pour définir l'objet this dans la fonction et pour passer des arguments aux fonctions.

Call prend la valeur à utiliser comme objet this dans la fonction comme premier paramètre, et les autres arguments à passer à la fonction sont passés individuellement (séparés par des virgules bien sûr). Le premier paramètre de la fonction Apply est également la valeur à utiliser comme objet this dans la fonction, tandis que le dernier paramètre est un tableau de valeurs (ou l'objet arguments) à transmettre à la fonction.

Cela semble complexe, mais voyons comment il est facile d'utiliser Apply ou Call. Pour résoudre le problème de l'exemple précédent, nous allons utiliser la fonction Apply :

```
//Notez que nous avons ajouté un paramètre supplémentaire pour l'objet de rappel, appelé "callbackObj".
function getUserInput(firstName, lastName, callback, callbackObj)  {
    // Faire d'autres choses pour valider le nom ici

    // L'utilisation de la fonction Apply ci-dessous définira cet objet comme étant callbackObj.
    callback.apply (callbackObj, [firstName, lastName]);
}
```

La fonction Apply ayant correctement défini cet objet, nous pouvons maintenant exécuter correctement la fonction de rappel et faire en sorte qu'elle définisse correctement la propriété fullName de l'objet clientData :

```
// Nous passons la méthode clientData.setUserName et l'objet clientData comme paramètres.
// L'objet clientData sera utilisé par la fonction Apply pour définir l'objet this.
getUserInput ("Barack", "Obama", clientData.setUserName, clientData);

// la propriété fullName sur le clientData a été correctement définie
console.log (clientData.fullName); // Barack Obama
```

Nous aurions également utilisé la fonction Call, mais dans ce cas, nous avons utilisé la fonction Apply.

Plusieurs fonctions de rappel autorisées

Nous pouvons passer plus d'une fonction de rappel dans le paramètre d'une fonction, tout comme nous pouvons passer plus d'une variable. Voici un exemple classique avec la fonction AJAX de jQuery :

```
function successCallback() {
    // Faites les choses avant de les envoyer
}

function successCallback() {
    // Faire quelque chose si le message de succès est reçu
}

function completeCallback() {
    // Faire les choses une fois terminées
}

function errorCallback() {
    // Faire quelque chose si une erreur est reçue
}

$.ajax({
    url:"http://fiddle.jshell.net/favicon.png",
    success:successCallback,
    complete:completeCallback,
    error:errorCallback

});
```

Problème et solution de l'enfer des callbacks (Callback Hell)

Dans l'exécution de code asynchrone, qui est simplement l'exécution de code dans n'importe quel ordre, il est parfois courant d'avoir de nombreux niveaux de fonctions de rappel au point d'avoir un code qui ressemble à ce qui suit. Le code désordonné ci-dessous est appelé l'enfer des callbacks en

raison de la difficulté à suivre le code en raison des nombreux callbacks. J'ai pris cet exemple dans le node-mongodb-native, un pilote MongoDB pour Node.js. [2]. L'exemple de code ci-dessous est juste pour la démonstration :

```
var p_client = new Db('integration_tests_20', new Server("127.0.0.1", 27017, {}), {'pk':CustomPKFactory});
p_client.open(function(err, p_client) {
    p_client.dropDatabase(function(err, done) {
        p_client.createCollection('test_custom_key', function(err, collection) {
            collection.insert({'a':1}, function(err, docs) {
                collection.find({'_id':new ObjectID("aaaaaaaaaaaa")}, function(err, cursor) {
                    cursor.toArray(function(err, items) {
                        test.assertEquals(1, items.length);

                        // Fermons le db
                        p_client.close();
                    });
                });
            });
        });
    });
});
```

Il est peu probable que vous rencontriez souvent ce problème dans votre code, mais lorsque c'est le cas - et ce le sera de temps en temps - il existe deux solutions à ce problème. [3]

1. Nommez vos fonctions, déclarez-les et passez juste le nom de la fonction comme callback, au lieu de définir une fonction anonyme dans le paramètre de la fonction principale.
2. Modularité : Séparez votre code en modules, afin de pouvoir exporter une section de code qui effectue un travail particulier. Vous pouvez ensuite importer ce module dans votre application plus vaste.

Créez vos propres fonctions de rappel

Maintenant que vous avez tout compris (je pense que c'est le cas, sinon vous pouvez relire rapidement :)) sur les fonctions de rappel JavaScript et que vous avez vu que l'utilisation des fonctions de rappel est plutôt simple mais puissante, vous devriez examiner votre propre code pour trouver des occasions d'utiliser les fonctions de rappel, car elles vous permettront de :

- Ne pas répéter le code (DRY-Do Not Repeat Yourself)
- Mettre en œuvre une meilleure abstraction où vous pouvez avoir des fonctions plus génériques qui sont polyvalentes (peuvent gérer toutes sortes de fonctionnalités)
- Avoir une meilleure maintenabilité
- Avoir un code plus lisible
- Avoir des fonctions plus spécialisées.

Il est assez facile de créer ses propres fonctions de rappel. Dans l'exemple suivant, j'aurais pu créer une seule fonction pour faire tout le travail : récupérer les données de l'utilisateur, créer un poème générique avec les données et accueillir l'utilisateur. Cela aurait été une fonction désordonnée avec

beaucoup d'instructions if/else et, même ainsi, elle aurait été très limitée et incapable d'exécuter d'autres fonctionnalités dont l'application pourrait avoir besoin avec les données de l'utilisateur.

Au lieu de cela, j'ai laissé la mise en œuvre de la fonctionnalité ajoutée aux fonctions de rappel, de sorte que la fonction principale qui récupère les données de l'utilisateur peut effectuer pratiquement n'importe quelle tâche avec les données de l'utilisateur en passant simplement le nom complet et le sexe de l'utilisateur comme paramètres à la fonction de rappel, puis en exécutant la fonction de rappel.

En bref, la fonction getUserInput est polyvalente : elle peut exécuter toutes sortes de fonctions de rappel avec une myriade de fonctionnalités.

```
// Tout d'abord, configurez la fonction générique de création de poèmes ; elle sera la fonction de rappel dans la fonction getUserInput ci-dessous.
function genericPoemMaker(name, gender) {
    console.log(name + " is finer than fine wine.");
    console.log("Altruistic and noble for the modern time.");
    console.log("Always admirably adorned with the latest style.");
    console.log("A " + gender + " of unfortunate tragedies who still manages a perpetual smile");
}

//Le callback, qui est le dernier élément du paramètre, sera la fonction genericPoemMaker que nous avons définie ci-dessus.
function getUserInput(firstName, lastName, gender, callback) {
    var fullName = firstName + " " + lastName;

    // Assurez-vous que le callback est une fonction
    if (typeof callback === "function") {
    // Exécuter la fonction de rappel et lui passer les paramètres
    callback(fullName, gender);
    }
}
```

Appelez la fonction getUserInput et passez la fonction genericPoemMaker en tant que callback :

```
getUserInput("Michael", "Fassbender", "Man", genericPoemMaker);
// Output
/* Michael Fassbender is finer than fine wine.
Altruistic and noble for the modern time.
Always admirably adorned with the latest style.
A Man of unfortunate tragedies who still manages a perpetual smile.
*/
```

Comme la fonction getUserInput ne s'occupe que de la récupération des données, nous pouvons lui passer n'importe quelle fonction de rappel. Par exemple, nous pouvons passer une fonction greetUser comme ceci :

```
function greetUser(customerName, sex)  {
   var salutation  = sex && sex === "Man" ? "Mr." : "Ms.";
  console.log("Hello, " + salutation + " " + customerName);
}

// Passez la fonction greetUser comme callback à getUserInput.
getUserInput("Bill", "Gates", "Man", greetUser);

// Et voici le résultat
Hello, Mr. Bill Gates
```

Nous avons appelé la même fonction getUserInput que précédemment, mais cette fois-ci, elle a effectué une tâche complètement différente.

Comme vous le voyez, les fonctions de rappel offrent une grande polyvalence. Et même si l'exemple précédent est relativement simple, imaginez le travail que vous pouvez vous épargner et l'abstraction de votre code si vous commencez à utiliser les fonctions de rappel. Allez-y. Faites-le le matin ; faites-le le soir ; faites-le quand vous êtes fatigué ; faites-le quand vous êtes fatigué.

Notez les façons suivantes dont nous utilisons fréquemment les fonctions de rappel en JavaScript, en particulier dans le développement d'applications Web modernes, dans les bibliothèques et dans les frameworks :

- Pour l'exécution asynchrone (comme la lecture de fichiers et les requêtes HTTP)
- Dans les récepteurs/manipulateurs d'événements
- Dans les méthodes setTimeout et setInterval
- Pour la généralisation : concision du code

Mot de la fin

Les fonctions de rappel JavaScript sont merveilleuses et puissantes à utiliser et elles apportent de grands avantages à vos applications Web et à votre code. Vous devriez les utiliser quand le besoin s'en fait sentir ; cherchez des moyens de refactoriser votre code pour l'abstraction, la maintenabilité et la lisibilité avec les fonctions de rappel.

A la prochaine fois, et n'oubliez pas de revenir car JavaScriptIsSexy.com a beaucoup à vous apprendre et vous avez beaucoup à apprendre.

Références

1. http://c2.com/cgi/wiki?FirstClass
2. https://github.com/mongodb/node-mongodb-native
3. http://callbackhell.com/
4. JavaScript Patterns by Stoyan Stefanov (Sep 28, 2010)