Chapitre 2
Le JavaScript et la 3D

1. Bien programmer avec JavaScript

1.1 Optimiser les performances

Bien utilisé et avec les derniers navigateurs, JavaScript peut être seulement trois fois moins rapide que le C, ce qui est réellement exceptionnel pour un langage non compilé, peu typé et de haut niveau.

Il faut cependant veiller à respecter des règles simples dont voici une liste non exhaustive.

Éviter d'utiliser this

En JavaScript, l'objet `this` peut signifier plusieurs choses, et il est déconseillé dans la syntaxe JSLint.

```
var ma_classe=function() {
  this.attribut=0;
  this.ma_methode=function() {
    this.attribut=1;
  }
}
var mon_instance=new ma_classe();
mon_instance.ma_methode();
```

```
alert(mon_instance.attribut);
```

Le code suivant va retourner 1. En effet, lorsque `ma_methode()` est appelée, le `this` de `this.attribut=1` se réfère à `mon_instance`. Maintenant, exécutons le code suivant :

```
var mon_instance2=new ma_classe();
var methode=mon_instance2.ma_methode;
eval("methode();");
alert(mon_instance2.attribut);
```

Ce code va maintenant retourner 0. En effet, lorsque `methode()` est appelée, le `this` se réfère maintenant à la fonction `methode()` et non plus à l'instance `mon_instance2`.

Les codes précédents pourraient avantageusement être remplacés par :

```
var ma_classe=function() {

  var that={
    attribut:0
  };
  that.ma_methode=function() {
    that.attribut=1;
  }
  return that;
}
var mon_instance=ma_classe();
mon_instance.ma_methode();
alert(mon_instance.attribut);

var mon_instance2=ma_classe();
var methode=mon_instance2.ma_methode;
eval("methode();");
alert(mon_instance.attribut);
```

Dans ce dernier cas, les deux alertes affichent bien la valeur 1.

Éviter d'utiliser du code en tant que chaîne de caractères

Dès que le moteur JavaScript rencontre une fonction utilisant du code en tant que chaîne de caractères, il doit transformer celui-ci en code exécutable. C'est une opération pénalisante en termes de performances.

Il faut donc éviter d'utiliser la fonction `eval()`, et remplacer autant que possible le code en tant que chaîne de caractères en code direct.

Exemple avec setTimeout()

```
var ma_fonction=function() {
  console.log("10 secondes se sont écoulées");
}
setTimeout("ma_fonction()", 10000);
```

Dans cette séquence, la dernière instruction sera avantageusement remplacée par :

```
setTimeout(ma_fonction, 10000);
```

Les tableaux et les objets

Évitez de tester un attribut d'un objet s'il est absent. En effet, avant de retourner *undefined*, JavaScript va parcourir tous les attributs et les méthodes de l'objet.

Même si JavaScript autorise d'indexer des tableaux avec d'autres éléments que des nombres entiers, il faut éviter. En effet, dans ce dernier cas, les tableaux sont considérés comme des objets et leur manipulation est beaucoup moins efficace que celle de tableaux numériques.

Déclarez la taille de vos tableaux lors de leur déclaration en utilisant la fonction `Array()` :

```
var mon_tableau=Array(5); //pour un tableau à 5 éléments
```

Pour parcourir un tableau numérique, la structure `for-in` est moins rapide qu'une simple boucle for. Il faut ainsi préférer :

```
for (var i=0; i<mon_tableau.length; i++) {
  ,,,
}
```

à

```
for (var i in mon_tableau) {
  ,,,
}
```

Remarque

Pensez à la fonction native `array.map(fonction)` si vous devez parcourir un tableau. Elle sera plus rapide encore qu'une boucle for !

Éviter les variables globales

Il est nécessaire d'éviter autant que possible les variables globales. En effet, elles sont plus longues d'accès par le moteur JavaScript lorsque ce dernier exécute une portion de code local, et le garbage collector de JavaScript ne les nettoie pas, d'où un risque de fuites mémoire. De plus, avoir trop de variables globales nuit à la lisibilité du code et complexifie le débogage.

Les conversions implicites

Les conversions implicites sont souvent source d'erreurs. Ainsi 0, *null* ou "" peuvent implicitement être convertis en *False*.

Nous vous conseillons d'utiliser au maximum les tests absolus qui vérifient aussi l'égalité des types, par exemple `===` à la place de `=`.

Conseils divers

Il faut éviter autant que possible le recours à la structure `try-catch-finally`, et les *console.log()*, pénalisants pour la performance.

Utilisez au maximum les fonctions natives de JavaScript. Elles sont toujours plus rapides que leur équivalent reprogrammé.

1.2 L'héritage en JavaScript

Le recours à la syntaxe JSLint permet de recréer aisément un héritage simple.

Exemple d'héritage

Considérons une classe Chat, qui hérite de la classe Animal. On prend pour la classe Animal :

```
/*
Animal
```

```
  spec.taille : taille de l'animal en mètres
*/
var Animal = (function () {
  return {
  instance : function (spec) {
               spec.taille=spec.taille || 0;
               return {
                 manger : function() {
                   ...
                 },
                 dormir : function() {
                   ...
                 },
                 get_taille : function() {
                   return spec.taille
                 }
               };
             }
  };
}());
```

Pour créer un animal, il suffira d'appeler la méthode instance :

```
var mamouth=Animal.instance({taille : 9});
mamouth.manger();
```

Créons une classe Chat héritant de la classe Animal et ajoutant une méthode, miauler :

```
/*
Chat
*/
var Chat = (function () {
  return {
  instance : function (spec) {
               var that=Animal.instance({taille: 1}) ;
               // (tous les chats font 1m de long)
               that.miauler=function() {
                 alert("miaou");
               }
               return that;
             }
  };
}());
```

Pour créer une instance de Chat, le faire manger et le faire miauler, il suffit de lancer le code suivant :

```
var mon_chat=Chat.instance({});
mon_chat.miauler();
mon_chat.manger();
```

1.3 Structurer le code avec PHP

Nous conseillons de placer chaque classe JavaScript dans un fichier portant le nom de la classe et ayant pour extension .js. Cela permet d'avoir un code clair et d'éviter les barres de défilement trop longues.

De plus, si vous travaillez à plusieurs développeurs sur un même projet avec un gestionnaire de versions comme *GIT* ou *Mercurial*, vous aurez moins de conflits de fusion à gérer.

Une méthode triviale pour regrouper les scripts serait d'ajouter pour chaque classe une balise :

```
<script type='text/javascript' src='Ma_classe.js'></script>
```

dans le code HTML de la page principale.

Cependant, au niveau de l'optimisation du temps de chargement de la page, ce n'est pas optimal. En effet, le navigateur devra effectuer une requête indépendante pour chaque script.

Il est préférable de regrouper les classes dans quelques fichiers PHP de la forme suivante :

```
<?
header("Content-type: text/javascript");
include("Ma_classe1.js");
include("Ma_classe2.js");
include("Ma_classe3.js");
...
?>
```

et d'inclure le fichier PHP en tant que script dans le code HTML de la page principale :

```
<script type='text/javascript' src='script.php'></script>
```