Le langage JavaScript

Introduction à la programmation Web (2/3)

Benjamin Canou

9 avril 2018

Cours INE11 @ ENSTA

Plan de ce cours

Cette semaine

- Historique & caractéristiques
- Premiers pas et langage de base
- Structures de données et modèle objet

La semaine prochaine

- Accès à et modification de la page
- Gestion d'évènements

Petit historique

- Les débuts :
  - Introduit dans Netscape en 1995
  - Conçu comme un petit langage de script pour compléter Java
  - Implantation plus tard par Microsoft sous le nom de JScript
- Évolution et adoption :
  - Longue histoire d'incompatibilités
  - Tentatives d'évolution / harmonisation abandonnées
  - Normalisation sous le nom d'ECMAScript
  - Performances originellement très mauvaises, et très variables
- Aujourd'hui :
  - Bientôt ECMAScript 6
  - Performances raisonnables, encore variables
  - Encore des variations entre les bibliothèques
  - Sorti du navigateur : Node.JS

Caractéristiques principales

- Langage principalement impératif
- Traits fonctionnels (principalement pour fonctions de rappel)
- Modèle objet inhabituel, très flexible
- Typage dynamique très laxiste (conversions et valeurs implicites)
- Portée lexicale un peu inhabituelle + portée globale

Comment programmer en JavaScript ?

- **Dans un fichier HTML :**

```
1 : <html>
2 :   <head>
3 :     <script language="JavaScript">
4 :       // code JavaScript
5 :     </script>
6 :   </head>
7 :   <body></body>
8 : </html>
```

- **Dans un fichier JS appelé par un fichier HTML :**

```
1 : <html>
2 :   <head>
3 :     <script language="JavaScript" src="code.js" defer></script>
4 :   </head>
5 :   <body></body>
6 : </html>
```

- **Dans la console du navigateur**
- **Dans un interprète indépendant**

Premiers pas en JavaScript

- **Bonjour tout le monde**

```
1 : alert ("Bonjour") ;
```

- **Bonjour madame**

```
1 : var name = prompt ("Entrez␣votre␣nom") ;
2 : alert ("Bonjour␣" + name) ;
```

Syntaxe de base

À la C / Java :

- **Blocs :** `{ ... }` **(attention, pas des blocs lexicaux)**
- **Expressions arithmétiques et opérateurs usuels,**
- **Affectations :** `x = expr`
- **Contrôle :** `for (;;;){}`, `while (){}`, `if(){}`, **etc.**

Fonctions :

- **Appel de fonction :** `f (x, y)`
- **Définition de fonction :**

```
1 : function f(x, y) {
2 :   return (x + y) ;
3 : }
```

Types de base :

- Nombres (pas de différence flottant / entier)
- Chaînes (non mutables), expressions rationnelles
- Tableaux extensibles / creux
- Objets (pour l'instant, des tables associatives)

Tableaux :

- **Creation :** `[]`, `var t = []`
- **Initialisation :** `[ 1, 2, 4, 8, 16 ]`
- **Lecture :** `t[index]` **(0** $\leq$ **index** $<$ **t.length)**
- **Affectation :** `t[index] = expr`
- **Itération :** `for (index in tab) { ... }`

Objets :

- **Creation :** `{}`, `var t = {}`
- **Initialisation :** `{ x : 12, y : 23 }`
- **Lecture :** `t.x`, `t["x"]`
- **Affectation :** `t.x = expr`, `t["x"] = expr`

Méthodes :

- **Appel :** `obj.m (x, y)` **(cf. la suite du cours)**
- **Définition :**

```
1 : function f (x, y) { return (x + y); } ;
2 : obj = {} ;
3 : obj.m = f ;
4 : obj.m (2, 3) ; // pour appeler
```

- **Définition (alternative) :**

```
1 : obj = { f: function (x, y) { return (x + y); } }
```

Portée globale dynamique :

- **Variables définies partout dés leur première affectation**
- **Masquées par les variables locales**
- **Toujours accessibles par** `window.variable`

Exemple :

```
1 : function f() { alert (xx); }
2 : f() ; // affiche undefined ou lève une erreur
3 : xx = 3;
4 : f() ; // affiche 3
```

Portée locale lexicale :

- **Mot-clef `var` pour définir une locale**
- **Portée : la fonction courante**
- **Par défaut, les variables sont globales !**

Exemple :

```
1 : function f() {
2 :   x = 3 ; // référence la globale x
3 :   var y = 2 ; // crée une nouvelle locale y
4 : }
5 : f() ;
6 : // la globale x est affectée
```

Fonctions anonymes :

```
1 : var f = function () { ... } ;
2 : (function () {...}) () ;
```

Fermetures :

```
1 : function add_const (x) {
2 :   return (function (y) {
3 :     return (x + y) ;
4 :     // la variable x de la fonction parente
5 :     // est enregistrée dans la fermeture
6 :   }) ;
7 : }
8 : var add_2 = add_const (2)
9 : var add_4 = add_const (4)
10 : var five = add_2 (3)
11 : var seven = add_4 (3)
```

Attention ! les variables de l'environnement ne sont pas figées.

```
1 : function what () {
2 :   var fs = [] ;
3 :   for (var i = 0; i < 10; i++) {
4 :     fs[i] = function () { alert (i) ; }
5 :   }
6 :   fs[4]() ;
7 : }
```

Quel est le résultat ?

Variables spéciales :

- `this`: **dans le code d'une méthode**
- `arguments`: **dans le code d'une fonction**
  - **tableau des arguments,**
  - **utile au cas où la fonction est appelée avec trop de paramètres,**
  - **sert à implanter des arguments optionnels.**

Modèle d'héritage par **prototypes** :

- Chaque objet a un champ caché `__prototype__`,
- ce champ est lui-même un objet,
- si un champ `o.m`n'est pas trouvé, `o.__prototype__.m`est recherché,
- et récursivement,
- jusqu'à la fin de la chaîne de prototypes : `Object`.

Le prototype d'un objet :

- Est fixé à sa création,
- dépend de son constructeur (cf. suite),
- peut être modifié dynamiquement.

Définition de constructeur personnalisé :

- **Mot-clef `function` pour définir une fonction de construction,**
- **mot-clef de `new` devant l'appel de fonction,**
- **crée un nouvel objet (`this` dans le corps du constructeur),**
- **le prototype des futurs objets est accessible via le champ `prototype` du constructeur.**

Exemple :

```
1 : function Vec(u, v) {
2 :   this.u = u ;
3 :   this.v = v ;
4 : }
5 : Vec.prototype.norm = function () {
6 :   return Math.sqrt (this.u * this.u + this.v * this.v) ;
7 : }
8 : v = new Vec (2, 3) ;
9 : alert (v.norm()) ;
```

La semaine prochaine : Javascript II

http://benjamin.canou.fr/Cours/ENSTA/INE11

benjamin@ocamlpro.com