Formation JavaScript avancé

Pierre Bretéché

Dawan

pbreteche@dawan.fr

October 30, 2017

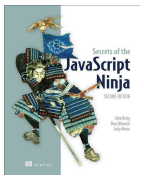
Plan d'intervention I

À propos de ce document

Pierre Bretéché <pbreteche@dawan.fr>
Consultant, Formateur en développement web
Addict du SOLID DRY KISS

Référence

- Mozilla Developper Network: `developer.mozilla.org`
- 

You don't know JS, Kyle Simpson
- 

Secrets of JavaScript Ninja, Bear Bibeault et John Resig

Présentation du langage

Débuts chez Netscape

- 1993 NSCA Mosaic (NSCA $\in$ University of Illinois)
- 1994 fondation de Mosaic Communications, Mountain View: débauche de la majorité des dev originaux de NSCA
- 1994 (fin) Mosaic Netscape (nom Projet: Mozilla = «Mosaic killer»), 3/4 marché en 4 mois
- 1944 (fin) renommage Netscape Navigator & Netscape Communications

Naissance du langage

1995

- selon Marc Andreessen (fondateur), besoin d'un langage «ciment» pouvant être écrit directement dans le HTML
- recrutement Brendan Eich pour embarquer Scheme (Lisp)
- collaboration avec Sun Microsystems, support de Java
- ⇒langage de script doit respecter la syntaxe Java
- mai: premier prototype écrit en 10j
- nom interne Mocha, premières bétas Netscape Navigator 2.0 LiveScript puis Javascript en septembre

Browser war

- 1996, nov: candidature de la spéc JS auprès d'Ecma
- 1997, juin: ECMA-262, première édition du standard
- 1998, juin: ECMAScript 2
- 1999, déc: ECMAScript 3
- 2003 ECMAScript 4, mis en attente dû à la non-coopération de Microsoft
- 2005 Eich rejoint Ecma, redémarrage du projet ECMAScript 4 avec Macromedia (E4X et ActionScript 3)

Standardisation

- 2008, juil: négociation entre les parties à Oslo, accord probable début 2009, ECMAScript 3.1 est renommé ECMAScript 5
- 2009, déc, ECMAScript 5
- 2011, juin ECMAScript 5.1
- 2015, juin ECMAScript 2015
- 2016, juin ECMAScript 2016
- 2017, juin ECMAScript 2017

Popularité

JavaScript is an easy-to-use object scripting language [...]. While Java is used by programmers to create new objects and applets, JavaScript is designed for use by HTML page authors and enterprise application developers [...]. JavaScript is analogous to Visual Basic in that it can be used by people with little or no programming experience to quickly construct complex applications.

Popularité

quelques moments ayant poussé à la popularité du JavaScript

- AJAX (autour de 2005)
- CommonJS 2009
- SPA

- augmentation du nombre d'applications et bibliothèques
- augmentation du nombre d'environnement
- augmentation du nombre de développeurs
- professionnalisation de la compétence

Rappels

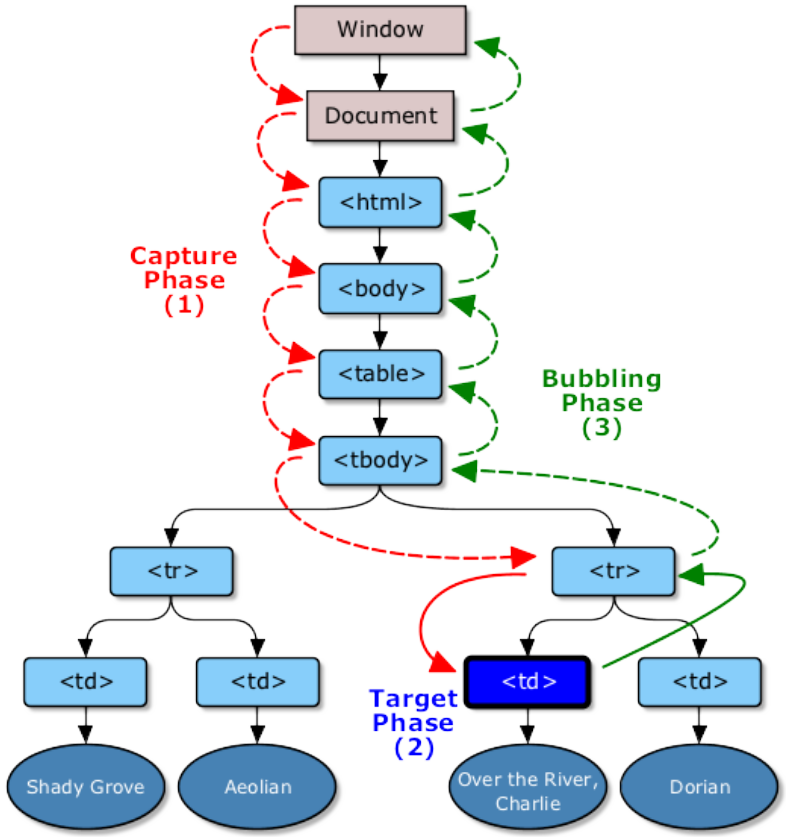
- $\simeq$ 1995 DOM-0 (Legacy DOM), fonctionnalités limités, pas de spec, partiellement décrit dans HTML4
- 1998 DOM Level 1: modélisation complet du document HTML (ou XML)
- 2000 DOM Level 2: getElementById, modèle événementiel, support xmlns
- 2004 DOM Level 3: XPath, événement clavier
- 2015 DOM Level 4: part du WHATWG HTML Living Standard

Écouteurs

```
<nav>
        <a>Open menu</a>
        <ul class="menu">
                <li>
                        <a href="#">one link</a>
                </li><li>
                        <a href="#">another link</a>
                </li>
        </ul>
</nav>
```

```
document.querySelector('.menu a[href]')
        .addEventListener('click', function(){
        // do some stuff
})
```

Flux de l'événement



Exploiter les phases

```
target.addEventListener(type, listener [, useCapture]);
```

```
var phase = event.eventPhase;
```

```
return false // DOM Level 0
event.preventDefault() // DOM Level 2
```

```
event.stopPropagation()
```

return false

Attention ! Certaines bibliothèques modifie ce comportement par défaut (ex: jQuery utilise "return false" pour annuler le comportement par défaut ET stopper la propagation)

Types

Types

- number: nombre à virgule flottante, double précision
- string: chaîne de caractères
- boolean: booléen
- null: nul
- undefined: indéfini

Initialisation par défaut

Attention, aucune valeur par défaut pour une variable non-initialisée (déclarée ou non)! Undefined !

Tableaux

- Objet de type tableau
- propriétés natives:
  - length (lecture seule)
- syntaxe littérale: [1, 'banane', -123.45e67, true]
- accès à une position par son index avec crochet
- ajout, retrait, nombreuses fonctions: push, pop, shift, unshift, slice, splice...

Itération

Structure itérative sur l'index

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

Structure itérative sur file (pile)

```
while(var elem = tableau.pop()) {}
while(var elem = tableau.unshift()) {}
```

Appel sur chaque élément

```
tableau.forEach(function(currentValue, index, array){})
tableau.map(function(currentValue, index, array){})
tableau.filter(function(currentValue, index, array){})
// [...]
```

Itération

Attention au for..in !

Accède aux propriétés personnalisée, cette structure est d'avantage conçue pour obtenir les propriétés d'un objet !

```
var tableau = ['pomme', 'banane', 'poire']
tableau.famille = 'fruits'
for (var fruit in tableau) {
        console.log(tableau[fruit])
}
// pomme banane poire fruits
```

Objets

Attention objets!

ne pas confondre objet JavaScript et Objet POO !

- est une référence en mémoire
- tout élément nommable via un symbole est un objet (à l'exception des variables primitives)
- un objet est simplement une collection de propriétés associées à un symbole
- une propriété est un association clé: valeur
- l'opérateur d'accès à la propriété est le «.»

Objets

```
var monChat = new Object()
monChat.nom = 'Medor'
monChat.miaule = function(){ console.log('meeow') }
monChat['se fait marcher sur la queue'] = function() {
        console.log('MEEEeeeEEEeeeEEEW ! ! ! ')
}

monChat.miaule()
monChat['se fait marcher sur la queue']()
```

Littéraux objets

```
var monChat = {
        nom: 'Medor'
        miaule: function(){ console.log('meeow') }
        'se fait marcher sur la queue': function() {
                console.log('MEEEeeeEEEeeeEEEW ! ! ! ')
        }
}
monChat.miaule()
monChat['se fait marcher sur la queue']()
```

JSON

```
var jsonString = '{ "nom": "Medor"}'
monChat = JSON.parse(jsonString)

monChat.miaule = function(){ console.log('meeow') }

JSON.stringify(monChat)
```

hasOwnProperty

- méthode disponible sur tout objet
- permet de vérifier si:
  1. une propriété existe
  2. cette propriété est propre à l'objet (non-héritée)

Tester l'égalité

Quatre types d'égalités

- Comparaison d'égalité abstraite (==)
- Comparaison d'égalité stricte (===)
- SameValueZero: utilisé par certaines fonctions natives depuis ES2016
- SameValue: utilisé partout ailleurs

Comparaison d'égalité abstraite

http://www.ecma-international.org/ecma-262/5.1/#sec-11.9.3

1. converti les opérandes s'ils ne sont pas du même type
2. applique une comparaison d'égalité stricte

Comparaison d'égalité stricte

http://www.ecma-international.org/ecma-262/5.1/#sec-11.9.6
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Using_the_Equality_Operators

1. si les types sont différents le résultat est faux
2. Algorithme SameValue http://www.ecma-international.org/ecma-262/5.1/#sec-9.12

le String, un objet bien primitif

String: à la fois primitif et objet

Un objet String n'aura pas le même type suivant s'il est déclaré via un littéral ou l'instruction new String!

```
// true as both operands are type String (i.e. string primitives):
'foo' === 'foo'

var a = new String('foo');
var b = new String('foo');

// false as a and b are type Object and reference different objects
a == b

// false as a and b are type Object and reference different objects
a === b

// true as a and 'foo' are of different type and, the Object (a)
// is converted to String 'foo' before comparison
a == 'foo'
```

Scope

Scope

en JavaScript ECMAScript 5.1 Pas d'espace de nommage, pas de mécanisme de paquetage, pas de module, pas de bundle
⇒ faire attention aux collision dans l'espace de nommage !

scope

Contrôle le cycle de vie de la variable en mémoire
Permet d'éviter des collisions

scope

Un symbole est conservé au sein de sa fonction de définition, sinon est considérée comme global
Une donnée en mémoire ne peut être libérée par le ramasse-miette (garbage collector) que si elle ne peut plus être accédée d'aucune manière que ce soit ⇒ attention au partage de références!

Fonctions

Définition

3 façons de définir une fonction:

- déclaration de fonction
- constructeur Function
- expression de fonction

Déclaration de fonction

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function

```
function gratterLAcoudoir() {
        console.log('scriiitch')
}
```

Expression de fonction

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/function

```
var gratterLAcoudoir = function() {
console.log('scriiitch')
}
```

Constructeur Function

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function

```
var gratterLAcoudoir = new Function('console.log(\'scriii
```

Hissage des déclarations de fonctions

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function#Function_declaration_hoisting

- les déclaration de fonctions sont «hissées»
- l'interpréteur remonte les déclaration de symbole lors de l'analyse syntaxique
- fonctionne même avec le JIT !
- sépare la déclaration de l'initialisation, mais ne remonte pas l'initialisation

Appel

4 façons de d'appeler une fonction:

- en tant que fonction
- en tant que méthode (fonction membre)
- en tant que constructeur
- via call ou apply

En tant que fonction

```
gratterLAcoudoir()
```

la pseudo variable this n'est pas définie par le contexte d'appel.
En mode non strict, elle prend la valeur par défaut de l'objet global (window dans une page web)

En tant que méthode

```
monChat.gratterLAcoudoir()
```

la fonction doit être associé à un objet par le biais d'une propriété
la pseudo variable this est le contexte d'appel, donc la variable possédant la propriété depuis laquelle la fonction est appelée

En tant que constructeur

```
new gratterLAcoudoir()
```

la fonction est précédée de l'opérateur new

un contexte anonyme par est créé «new» et sera référencé par «this»

via call ou apply

```
gratterLAcoudoir.call(monChat)
gratterLAcoudoir.apply(monChat)
```

call et apply sont similaire en tout point à la différence du passage d'argument
le premier paramètre permet de choisir arbitrairement un contexte pour this

IIFE

Les expressions de fonctions immédiatement invoquées sont:

- des expressions de fonction comme vu précédemment
- invoquées en tant que fonction par l'ajout de parenthèses pour le passage d'argument à la suite
- une seconde paire de parenthèses sert à protéger l'expression

```
(function(){
        console.log('parenthesis wrap expression')
})()

(function(){
        console.log('parenthesis wrap function call')
}())
```

Fermetures

Définition

Une fonction interne a accès à l'ensemble des variables du contexte d'appel
Les fermetures sont transitives

- des expressions de fonction comme vu précédemment
- invoquées en tant que fonction par l'ajout de parenthèses pour le passage d'argument à la suite
- une seconde paire de parenthèses sert à protéger l'expression

```
var externalScope = 123
(function(){
        var internalScope = 456
        console.log(externalScope, internalScope)
})
```