JavaScript: Introduction et Intégration au HTML

Quentin Bouillaguet <quentin.bouillaguet@u-psud.fr>

2019-2020

Introduction

Il était une fois...

Des débuts chaotiques (1995-2008) ...

- Créé en 10 jours par Brendan Eich pour le navigateur de Netscape.
- Confusion avec Java (similitudes syntaxiques mais concepts différents)
- **Incompatibilité** entre navigateurs
- **Failles de sécurité** ($=>$ `JS` souvent désactivé)
- Scripts interprétés par le navigateur (**peu performant**)
- D'autres cadres pour faire du web interactif (`Adobe Flash`, ...)

... mais parti pour durer (2008-aujourd'hui)

- **Interopérabilité**: normalisation (**ECMAScript**) et uniformisation
- **Performance**: course à la performance entre navigateurs (*JIT*)
- **Sécurité**: scripts isolés des autres programmes (*sandboxing*)
- Au cœur du Web d'aujourd'hui !

Le langage

Propriétés

- Langage de script interprété
- Orienté objet, **faiblement** typé
- Exécuté dans différentes configurations
  - navigateur côté **client**
  - côté **serveur** (`Node.js`, ...)
  - applications **multiplateformes** (`Electron`, `React`, ...)

Limitations

- Développement difficile (erreurs silencieuses, typage, ...)
- Pas d'entrées/sorties de fichiers (pour la sécurité)
- S'exécute sur un fil d'exécution (absence de *multi-threading*)

JavaScript: Langage

Instructions

- Syntaxe similaire à `C/C++/Java/C#`
- Différences majuscules/minuscules
- Noms de variables ou fonctions (**identifiants**):
  - lettres, '_', '$' ou chiffre (sauf en début)
  - mots-clés du langage sont réservés
- Instructions séparées par ';' (optionnel mais à mettre!)
- Commentaires en fin de ligne `//` ou multiligne `/* ... */`

```
var x = 10; // x vaut 10
var y = 5;  // y vaut 5
/* on assigne x + y à num
   num vaut 15 */
var num = x + y;
```

Variables

Variables

- Typage **faible**: *automatique* et *dynamique*

```
var foo = 42;    // foo est un nombre
    foo = "bar"; // foo est une chaine de caractere
    foo = false; // foo est un booleen
```

- 3 niveaux de visibilité: **global**, local à la **fonction** ou au **bloc**

Déclaration et Initialisation

- `var n;` déclare une variable `n` locale au **contexte d'exécution** courant (fonction ou global)
- `let x = 10;` déclare une variable `x` locale au **bloc** courant et l'initialise à `10` (comme en `C`)

Variables Globales

- Assignation à une variable non déclarée

Types

Types et valeurs primitifs

- Nombres (**`number`**): *décimal* (`6.02e+23`) ou *entier* (32, `0x1e`)
- Chaines de caractère (**`string`**): entre `"guillemets"` ou `'apostrophes'`
- Booléens (**`boolean`**): `true` et `false`
- Valeur **`null`**: aucune valeur
- Valeur et type indéfini (**`undefined`**): valeur ou type d'une variable non affectée

Attention: `undefined` et `null` sont deux valeurs différentes !

Objets

- Collection de propriétés (type **object**)

Opérateurs

Usuels

- Arithmétiques: `+ - * / %`
- Logiques: `&&` (et) `||` (ou) `!` (non)
- Binaires: `&` (et) `|` (ou exclusif)
- Ordre: `< <= >= >` (nombre ou chaîne de caractère)
- Concaténation: `+`

Affectations

- Affectations: `= += *= ...`

Égalité

- Égalité: `== !=` (y compris avec les chaînes de caractères)
- Identité: `=== !===` (égalité + type identique)

Conditions

Conditionnelle

```
if (n >= 0) {
  ...
} else {
  ...
}
```

Branchement multiple

```
switch (fruit) {
  case "Strawberry":
    color="red"; break;
  case "Grape":
    color="purple"; break;
  default:
    color="orange";
}
```

Boucles

Répétitions avec incrément

```
for (i = 0; i < 10; i++) {
  sum += i;
}
```

Tant que ...

```
while (i < 10) { ... }
```

Faire ... tant que ...

```
do { ... } while (a == "");
```

Rupture de séquence

- `break` et `continue` comme en C

Fonctions

Déclaration par le mot-clef `function`

```
function ajoute(x, y) {
  return x + y;
}
```

- Renvoi du résultat par `return`
- En JavaScript, une fonction est une **valeur** !
  On peut donc assigner une fonction à une variable:

```
var ajoute = function(x, y) {
  return x + y;
}
```

Appel de fonction

```
var s = ajoute(1, 2);
```

- Toujours déclarer les variables locales à la fonction avec `var` ou `let` (sinon on modifie des variables globales !)

JavaScript: Objets

Structure de données

Déclaration et création d'un objet vide

```
var o = {};
```

- On peut l'utiliser comme un `struct` en langage `C`
- Contient des données de n'importe quel type (**propriétés**), **sans déclaration préalable**

Propriété (= nom et valeur d'un champ de la structure)

- Accessible par l'opérateur `.`

```
o.x = 1; o.y = 2;
```

- Accessible par le nom du champ (comme pour un dictionnaire)

```
o["x"] = 1; o["y"] = 2;
```

- Les propriétés qui n'ont pas été affectées auront la valeur `undefined`

```
o.z; // vaut 'undefined'
```

Structure de données

Déclaration et initialisation d'un objet

- Valeurs des propriétés définies par :

```
var o = { x: 1, y: 2 };
```

Méthodes

- Une **méthode** est une fonction qui est la propriété d'un objet

```
o.ajoute = function () {
  return this.x + this.y;
}
```

Mot-clef `this`

- Dans une méthode d'un objet, `this` correspond à l'objet possédant la méthode qu'on appelle.
- Dans d'autres contextes, voir la documentation de `this`

Opérateurs spéciaux

Opérateurs de type

- `typeof`: renvoie le type de l'opérande (`"number"`, `"string"`, `"boolean"`, `"object"`, `"function"`, `"undefined"`)
- `instanceof`: vrai si l'objet a été créé avec le constructeur donné
- `void`: renvoi `undefined`

Opérateurs d'objet

- `new`: création d'un objet
- `delete`: supprime une propriété d'un objet
- `in`: vrai si la propriété appartient à l'objet

Exemple

```
if (typeof o == "object" && x in o) {
  delete o.x
}
o.x; // undefined
```

JavaScript: Objets Standards

Objets standards

JavaScript fournit nativement un ensemble d'objets accessibles globalement

Objets fondamentaux

- `Object`, `Function`, `Boolean`, `...`

Manipulation de nombres

- `Number`: conversion et constantes
- `Math`: constantes et fonctions mathématiques

Manipulation de texte

- `String`: conversion et manipulation

Collections

- `Array`: tableaux

Référence

- Voir la référence sur les objets globaux

Tableaux (`Array`) – Principes

Déclaration et initialisation

- Peut mélanger des valeurs de types différents
- Tableau vide ou avec initialisation des valeurs :

```
var a = [];  // vide
var b = [42, "foo", false]; // 3 éléments
```

Accès aux éléments (opérateur `[]`)

```
a[0] = 1; a[1] = 2;
```

Taille dynamique

- Position du dernier élément donné par la propriété `length`
- Le tableau s'agrandit/rétrécit avec la création/suppression de nouveaux éléments

```
a[2] = 4;
a.length; // vaut 3
```

- Attention: valeur indéfinie (`undefined`) pour les cases vides

Tableaux (`Array`) – Propriétés

Propriétés

- `length`: taille du tableau

Méthodes

- `push(valeur...)`, `pop()`: gestion en pile
- `shift()`, `unshift(valeur...)`: gestion en file
- `sort(fonctionCmp)`: tri du tableau
- `splice(i, n)`: suppression de `n` éléments à partir de l'indice `i`
- `join(separateur)`: concatène les éléments
- `slice(début, fin)`: retourne un sous-tableau
- Voir la documentation pour `Array`

Exemple sur le tableau `a`

```
a.push(0); // rajoute 0 comme 4eme élément
a.join('-'); // retourne "1-2-4-0"
a.sort(); // a = [0, 1, 2, 4]
a.slice(1, 2); // a = [0, 4]
```

Objets Primitifs (`Number`, `Boolean` et `String`)

Conversion

- `Number(valeur)`: convertit la valeur (dont les chaînes) en nombre
- `Boolean(valeur)`: toutes les valeurs converties à `true` à part `0`, `NaN`, `null`, `undefined`, `""`
- `String(valeur)`: convertit la valeur en chaîne de caractères

Attention aux conversions implicites !!!

```
5 + 3 // 8
'5' + '3' // "53"
5 + '3' // "53"
```

Constantes

- `Number.MAX_VALUE`, `Number.MIN_VALUE`, `Number.NaN`, ...

String

Propriétés

- `length`: longueur de la chaîne

Méthodes

- `charAt(n)`: accès à un caractère
- `indexOf(sous-chaîne,début)`: recherche de chaîne
- `slice/substring(debut,fin)`: sous-chaîne
- `split(délimiteur,limite)`: découpe en tableau de chaînes
- `toUpperCase(),toLowerCase()`: conversion de casse
- Voir la documentation pour `String`

Exemple

```
var s = "Polytech";
var lastchar = s.charAt(s.length - 1); // lastchar = 'h'
```

`Math`

Constantes

- `Math.E`: base e des logarithmes naturels
- `Math.PI`: π
- `...`

Méthodes

- `Math.abs(x)`: valeur absolue
- `Math.sin(x)`,`cos(x)`,`tan(x)`, `...`: fonctions trigonométriques
- `Math.ceil(x)`,`floor(x)`,`round(x)`:
  arrondi au-dessus, au-dessous, au plus proche
- `Math.sqrt(x)`,`log(x)`,`exp(x)`,`pow(x,y)`:
  racine, logarithme, exponentielle et puissance
- `Math.min(...)`,`max(...)`: plus petite ou plus grande valeur
- `Math.random()`: nombre aléatoire compris dans [0, 1[

Documentation

- Voir la documentation pour `Math`

Intégration au HTML

HTML: Inclusion de scripts JavaScript

Balise `<script>`

- Référence à un fichier JavaScript externe (préférable)
- Dans l'entête `<head>` pour un pré-chargement du JS
- Dans le corps `<body>` pour une génération dynamique de code HTML

Gestionnaire d'événement

```
<input type="button" value="Oui" onclick="update()">
```

Pseudo-URL

```
<a href="javascript: void update()">
```

Exemple d'interaction HTML/JavaScript

Document HTML:

```
<html>
  <head>
    <title>Exemple de page dynamique</title>
    <script type="text/javascript" src="script.js"> </script>
  </head>
  <body>
    <button id="ici" onclick="update()">Cliquer ici</button>
  </body>
</html>
```

Fichier JavaScript `script.js`:

```
function update() {
  document.getElementById("ici").innerHTML = "Merci";
}
```

JavaScript: Accès aux éléments/balises

Recherche par identifiant

- HTML:
  - les attributs `id` identifient de manière **unique** un élément

```
<button id="ici" onclick="update()">Cliquer ici</button>
```

- JavaScript:
  - `getElementById(id)` retourne un **objet** dont on peut lire ou modifier les propriétés
  - retourne `null` si l'élément n'existe pas ou n'a pas été construit

```
document.getElementById("ici")
```

Recherche par type de balise

- Retourne un **tableau** d'objets éléments

```
document.getElementByTagName("button")
```

Retarder l'exécution du code

Ordre d'exécution du code

- Le script est **exécuté au moment de la lecture** de la ligne
- Lors de la lecture de l'entête, la page web n'est **pas encore créée**
- Il faut donc **retarder** l'exécution du code

Trois solutions

- Mettre le code JavaScript en fin de fichier
- Script externe à exécution retardée dans le HTML

```
<script type="text/javascript" src="script.js" defer> </script>
```

- Lancement programmée à la fin du chargement de la page depuis le JavaScript

```
window.onload = function() {
   ...
}
```

Modifier le document HTML depuis JavaScript

Génération dynamique de HTML

- dans le corps du document:

```
document.write("<h1>"+titre+"</h1>");
```

Modification du texte interne aux balises

- HTML:

```
<button id="ici">Cliquer ici</button>
```

- JavaScript (propriété `innerHTML` d'un élément):

```
document.getElementById("ici").innerHTML = "Merci";
```

- Modification plus complexe: utiliser le DOM
  Au prochain cours !

Interaction avec un clic

```html
<button onclick="update()">Cliquer ici</button>
```

Événement

- L'attribut `onclick` permet de répondre à un clic de l'utilisateur sur cet élément (ici un bouton)
- Le script donné en valeur de cet attribut (ici `update()`) est **exécuté** à chaque fois que cet **événement** se produit
- D'autres événements permettent de gérer les différentes entrées de l'utilisateur !

  Au prochain cours !

Autres interactions

Afficher un message dans une fenêtre

```
alert('Tout se passe bien.');
```

Saisir une chaîne de caractères dans une fenêtre

```
var nom = prompt('Quel est votre nom?', 'Personne');
if (nom == null) return;
```

Récupérer le contenu d'un champ dans un formulaire

- HTML

```
<input type="text" id="moninput">
```

- JavaScript

```
var m = document.getElementById("moninput").value;
// la valeur obtenue est une chaîne de caractères
```