Développement Web 2

Bertrand Estellon

Aix-Marseille Université

April 1, 2014

JavaScript

JavaScript est un langage de programmation initialement utilisé pour créer des pages web interactives. Il peut être inclus dans un document HTML :

```
<!DOCTYPE html>
<html>
 <head>
  <script type="text/javascript" src="script.js"></script>
  <script type="text/javascript">
  //<![CDATA[
  /* code JavaScript */
  //]]>
  </script>
 </head>
 <body></body>
</html>
```


JavaScript – Les fonctions

Il est possible de définir des fonctions en JavaScript :

```
function additionner(a,b,c) {
  a = b + c;
  return a;
}

function multiplier(a,b) {
  return a*b;
}

alert(additionner(1,2,3));
alert(mutltiplier(2,4));
```


JavaScript – Les variables

Par défaut, les variables non-déclarées sont globales :

```
function ajouter(a) { total += a; }
total = 0; ajouter(2); ajouter(4); alert(total);
```

Pour déclarer une variable, il faut utiliser le mot-clé *var* :

```
function ajouter(a,b) {
  var total = a + b;
  return b;
}
total = 0;
total = ajouter(total,2); total = ajouter(total,4);
alert(total);
```

JavaScript – Les types

En JavaScript, les types sont associés aux valeurs :

```
var v = "toto";                 // variable contient une chaîne
v = 'toto';                     // variable contient une chaîne
v = 22;                         // variable contient un nombre
v = 22.12;                      // variable contient un nombre
v = true;                       // variable contient un booléen
v = ["toto", 22];               // variable contient un tableau
v = {name:"toto", age:22}; // variable contient un objet
```

Une variable qui n'a jamais été affecté est "undefined" :

```
var v; /* v est considéré comme "undefined". */
```

La valeur "null" peut être affectée à une variable. Cela signifie que la variable existe mais sa valeur ne désigne aucun objet :

```
var v = null;
```


JavaScript – Les types

Il est possible de connaître le type de la valeur contenue dans une variable à l'aide du mot-clé *typeof* :

```
var v; alert(typeof v);                          // "undefined"
v = "toto"; alert(typeof v);                     // "string"
v = 'toto'; alert(typeof v);                     // "string"
v = 22; alert(typeof v);                         // "number"
v = 22.12; alert(typeof v);                      // "number"
v = true; alert(typeof v);                       // "boolean"
v = null; alert(typeof v);                       // "object"
v = ["toto", 22]; alert(typeof v);               // "object"
v = {name:"a", age:22}; alert(typeof v);  // "object"
```


JavaScript – Les booleans

Affectation et utilisation des booleans :

```
var a = true;
var b = false;

var c = a || b;

if (c) {
  alert(a && b);
}
```


JavaScript – Les nombres

Les différentes opérations arithmétiques :

```
var v = 12;
var v = 2 + 4;
var v = 2 * 4;
var v = 4 / 2;
var v = 123 % 10;
v++; v--;
```

Les différentes affectations :

```
v=2; v+=2; v-=2; v*=2; v/=2; v%=2;
```

Les conversions entre chaînes et nombres :

```
var v = parseInt("123");
var v = parseFloat("123.12");
var s = v.toString();
```

JavaScript – Les comparaisons et opérateurs logiques

égal	`a == b`	vrai si a est égal à b
identique	`a === b`	vrai si a et b sont égaux et ont le même type
différent	`a != b`	vrai si a est différent de b
non identique	`a !== b`	vrai si a et b sont différents ou n'ont pas le même type
plus petit	`a < b`	vrai si a est strictement plus petit que b
plus grand	`a > b`	vrai si a est strictement plus grand que b
inférieur ou égal	`a <= b`	vrai si a est plus petit ou égal à b
supérieur ou égal	`a >= b`	vrai si a est plus grand ou égal à b

non	`!a`	vrai si a n'est pas vrai
et	`a && b`	vrai si a et b sont vrais
ou	`a || b`	vrai si a ou b sont vrais

```
var v = (test)?"Vrai":"Faux";
```


JavaScript – If/For/While/Continue/Break

```
var x = 2, y = 5;
if (x < y) document.write("<");
else document.write(">");

for (var i = 0; i < 10; i++) {
  if (i == 4) continue;
  document.write(i);
  if (i > 7) break;
}

var i = 0;
while(i < 10) {
  document.write(i);
  i++;
}
```


JavaScript – Switch

```
function (x) {
  switch (x) {
    case 0 : alert("zéro"); break;
    case 1 : alert("un"); break;
    case 2 : alert("deux"); break;
    case 3 : alert("trois"); break;
    case 4 : alert("quatre"); break;
    default : alert("nombre"); break;
  }
}
```


JavaScript – Les tableaux

Déclaration d'un tableau :

```
var fleurs = new Array();
fleurs[0] = "Rose";
fleurs[1] = "Tulipe";
fleurs[2] = "Coquelicot";
/* ou */
fleurs = ["Rose", "Tulipe", "Coquelicot"];
```

Les méthodes et les propriétés:

```
var length = fleurs.length;
var position = fleurs.indexOf("Tulipe");
```

JavaScript – Les tableaux

Parcourir un tableau :

```
for (var i = 0; i < fleurs.length; i++)
  document.write(i+"->"+fleurs[i]);

for (var i in fleurs)
  document.write(i+"->"+fleurs[i]);

for (var fleur of fleurs) /* Expérimental */
  document.write(fleur);
```


JavaScript – Les chaînes de caractères

```
var fleur="Tulipe";
var c=fleur[2];                          // 'l'
var length = fleur.length;               // 6
var index = fleur.indexOf("ip");         // '3'
var s = fleur.match("[^i]*");           // 'Tul'
var s = "Machin Truc";
var r = s.replace("Truc", "Bidule"); // "Machin Bidule"
var s = "a,b,c";
var r = s.split(",");                    // ['a','b','c']
var c = "Machin"+ " Truc" + 5            // "Machin Truc5"
```

Autres méthodes utiles sur les chaînes de caractères :

charAt, charCodeAt, concat, contains, endsWith, indexOf, lastIndexOf, match, replace, search, slice, split, startsWith, substr, substring, toLowerCase, toString, toUpperCase, trim.

JavaScript – Les fonctions

Il est possible d'affecter le code d'une fonction à une variable :

```
var sum = function(a,b) { return a+b; }
alert(sum(2,3)); // affiche 5
alert(typeof sum); // affiche "function"
```

Les fonctions peuvent utiliser des variables externes à la fonction :

```
var a;
var b = 2;
var assign = function() { a = b; }
assign(); alert(a);          // affiche 2
b = 3; assign(); alert(a);   // affiche 3
```

Il est important de noter que le code est lié au variable (et non aux valeurs des variables au moment de la définition de la fonction).

JavaScript – Les fonctions

Les paramètres des fonctions sont passés par valeur donc chaque paramètre est une variable locale à la fonction :

```
var sum2 = function(n) { n += 2; return n; }
var n = 2;
alert(sum2(n)); // affiche 4
alert(n);     // affiche 2
```

Une fonction peut retourner une fonction :

```
var getDisplayFunction = function(a) {
                return function() { alert(a); }
              }
var display3 = getDisplayFunction(3);
var display4 = getDisplayFunction(4);
display3(); // affiche 3
display4(); // affiche 4
```

JavaScript – Les fonctions

Une fonction peut retourner une fonction :

```
var getDisplayFunction = function(a) {
                return function() { alert(a); }
              }
var display3 = getDisplayFunction(3);
display3(); // affiche 3
```

Le code précédent est équivalent au code suivant :

```
var getDisplayFunction3 = function() { var a = 3;
                return function() { alert(a); }
              }
var display3 = getDisplayFunction3();
display3(); // affiche 3
```


JavaScript – Les objets

En JavaScript, il n'y a pas de classe. Les instance sont créée directement :

```
person=new Object();
person.name="Bob";
person.age=22;
```

On peut également utiliser une description littérale de l'objet :

```
person = {name : "Bob", age : 22};
```

Nous sommes en présence de références :

```
var bob = {name: "Bob", age:22}
var jim = {name: "Jim", age:23}
var joe = {name: "Joe", friends : [bob, jim]}
jim.age = 43;
alert(joe.friends[1].age); // affiche 43
```


JavaScript – Les objets

En JavaScript, il n'y a pas de classe.

Une fonction permet de construire un objet :

```
function Person(name, age) {
  this.name = name;
  this.age = age;
}

var bob = new Person("bob", 23);
var joe = new Person("joe", 42);
```

Il est possible d'ajouter des propriétés après la création de l'objet :

```
var jim = new Person("jim", 45);
jim.friends = [bob, joe];
```


JavaScript – Les objets

Des fonctions (ou méthodes) peuvent être associées à un objet :

```
function Counter() { this.count = 0; }
var counter = new Counter();
counter.notify = function() { this.count++; }
counter.add(v) = function() { this.count += v; }
counter.toString = function() { return "counter :"
                                    +this.count; }
counter.notify();
counter.add(3);
alert(counter.toString()); // "counter : 4"
```

JavaScript – Les objets

Il est également possible d'associer des méthodes dans le constructeur :

```
function Counter() {
  this.count = 0;
  this.notify = function() { this.count++; }
  this.add = function(v) { this.count += v; }
  this.toString = function() { return "counter :"
                                      +this.count; }
}

var counter = new Counter();
counter.notify();
counter.add(3);
alert(counter.toString()); // "counter : 4"
```


JavaScript – Les objets

Attention, le code suivant ne fonctionne pas car, dans la méthode *click*, la fonction *notify* n'est appelée sur un objet (dans ce cas, *this* contient une référence vers l'objet *Window*) :

```
function Counter() {
  ...
  this.notify = function() { this.count++; }
  ...
}

function click(callback) {
  callback();
}

var counter = new Counter();
click(counter.notify);
```


JavaScript – Les objets

Le code suivant fonctionne :

```
function Counter() {
  ...
  this.notify = function() { this.count++; }
  ...
}

function click(callback) {
  callback();
}

var counter = new Counter();
click(function() { counter.notify(); });
```


JavaScript – Les objets

Un autre exemple avec deux objets :

```
function Button(listener) {
  this.listener = listener;
  this.click = function() { listener.notify(this); }
}

function Counter() {
  this.count = 0;
  this.notify = function() { this.count++; }
}

var counter = new Counter();
var button = new Button(counter);
button.click();
alert(counter.count);
```

JavaScript – Les objets

Le code suivant ne fonctionne pas. Pourquoi ?

```
function Button() {
  this.setCallback = function(c) { this.callback = c; }
  this.click = function() { this.callback(this); }
}

function Counter(notifier) {
  this.count = 0;
  notifier.setCallback(function() { this.notify(); });
  this.notify = function() { this.count++; }
}

var button = new Button();
var counter = new Counter(button);
button.click(); alert(counter.count);
```


JavaScript – Les objets

Correction du code précédent :

```
function Button() {
  this.setCallback = function(c) { this.callback = c; }
  this.click = function() { this.callback(this); }
}

function Counter(notifier) {
  this.count = 0;
  var self = this;
  notifier.setCallback(function() { self.notify(); });
  this.notify = function() { this.count++; }
}

var button = new Button();
var counter = new Counter(button);
button.click(); alert(counter.count);
```


JavaScript – Les prototypes

Le constructeur suivant associe des méthodes identiques à chaque objet :

```
function Counter() {
  this.count = 0;
  this.notify = function() { this.count++; }
  this.add = function(v) { this.count += v; }
  this.toString = function() { return "counter :"
                                      +this.count; }
}
```

Notez que ce n'est pas forcement le cas :

```
function Counter(step) {
  this.count = 0;
  this.notify = function() { this.count+=step; }
}
var c1 = new Counter(1); c1.notify(); alert(c1.count); // "1"
var c2 = new Counter(2); c2.notify(); alert(c2.count); // "2"
```


JavaScript – Les prototypes

- Sans optimisation et analyse du code, il est obligatoire de conserver en mémoire toutes les fonctions associés à toutes les objets.
- Cependant, on va pouvoir associer à des objets un prototype contenant un ensemble de fonctions (et de propriétés) partagées.
- Lors d'une invocation, la méthode est recherchée dans l'objet puis dans le prototype (de façon récursive -> simulation de l'extension).
- Voici une mauvaise façon d'affecter un prototype à un objet :

```
function CounterPrototype() {
  this.notify = function() { this.count++; }
}

var prototype = new CounterPrototype();
function Counter() {
  this.count = 0; this.__proto__ = prototype;
}
```

JavaScript – Les prototypes

La bonne façon de faire :

```
function Counter() { this.count = 0; }

Counter.prototype.notify = function() { this.count++; }
Counter.prototype.add = function(v) { this.count += v; }
Counter.prototype.toString = function() { return "counter :"
                                    +this.count; }

var counter = new Counter();
counter.notify();
counter.add(2);
alert(counter.toString()); // "3"
alert(counter.__proto__ == Counter.prototype); // true
```


JavaScript – L'objet Math

- x = Math.abs(-2.23); // 2.23
- x = Math.ceil(6.05); // 7
- x = Math.floor(6.23); // 6
- x = Math.round(3.8); // 4
- x = Math.max(2,6); // 6
- x = Math.min(2,6); // 2
- x = Math.pow(3,3); // 27
- x = Math.random(); // 0.2323...
- x = Math.sqrt(9); // 3
- Math.E, Math.exp(v), Math.LN2, Math.LN10, Math.log(v), Math.LOG2E, Math.SQRT1_2, Math.SQRT2, Math.PI, Math.sin(v), Math.asin(v), Math.cos(v), Math.acos(v), Math.tan(v), Math.atan(v);

JavaScript – L'objet Date

Création d'un objet Date :

```
var today = new Date()
var d1 = d1 = new Date("February 10, 2013 10:12:12");
var d2 = new Date(2013,1,10);
var d3 = new Date(2013,1,10,10,12,12);
```

Utilisation :

```
var today = new Date()
var year = today.getFullYear(); alert(year);
today.setMonth(4);
...
```


JavaScript – Les exceptions

La gestion des exceptions est similaire à Java :

```
function validate(x) {
  if(x=="")     throw "empty";
  if(isNaN(x)) throw "not a number";
  if(x=="0")    throw "equal zero";
  if(x>50)      throw ["too high", x, 50];
}

try {
  validate("12");
  validate("123");
} catch(err) {
  alert(err);
}
```

JavaScript – JSON

JSON (JavaScript Oject Notation) est un format de données dérivé de la notation des objets et tableaux de ECMAScript (donc de JavaScript) :

```
{"machin": {
  "taille": 12,
  "style": "gras",
  "bidule": {
    "machin": [
      {"style" : "italique" },
      {"style" : "gras" }
    ]
  }
}}
```

L'avantage de JSON est qu'il est reconnu nativement par JavaScript.

JavaScript – JSON

Les éléments en JSON :

- Les objets : `{chaîne : valeur, chaîne : valeur...}`
- Les tableaux : `[valeur, valeur, ...]`
- Les valeurs : chaîne, nombre, objet, tableau, true, false, null
- Les chaînes : `"abcdef"` ou `"abcd\n\t"`
- Les nombres : `-1234.12`

```
{"unObjet": {
  "unTableau": [12, 13, 53],
  "unNombre" : 53,
  "unChaîne" : "truc\n"
  "unObjet" : { "style" : "gras" }
}}
```


JavaScript – JSON

En JavaScript, il est très facile de sérialiser une valeur en JSON :

```
var bob = {name: "Bob", age:22}
var jim = {name: "Jim", age:23}
var joe = {name: "Joe", friends : [bob, jim]}
JSON.stringify(joe);
```

Le code précédent génère la chaîne de caractères suivante :

```
{
  "name":"Joe",
  "friends":[
      {"name":"Bob","age":22},
      {"name":"Jim","age":23}
  ]
}
```


JavaScript – JSON

Inversement, il est facile de construire une valeur à partir d'une chaîne de caractères respectant le format JSON en utilisant la fonction *eval* :

```
var json = '{'+
             '"name":"Joe",'+
             '"friends":['+
                '{"name":"Bob","age":22},'+
                '{"name":"Jim","age":23}'+
             ']'+
           '}';
var joe = eval('('+json+')');
for (var i = 0; i < joe.friends.length; i++)
  alert(joe.friends[i].name);
```