

Technologies Web

Javascript

Alexandre Pauchet

INSA Rouen - Département ASI

BO.B.RC.18, pauchet@insa-rouen.fr

Plan

- 1 Introduction
- 2 Inclusion dans le code
- 3 Syntaxe générale
- 4 Les prototypes
- 5 Programmation évènementielle
- 6 Interaction avec HTML/XHTML : le DOM
- 7 Conseils de programmation

Introduction (1/6)

Généralités

- Description

- Créé en 1995 par Netscape et Sun Microsystems
- **But** : interactivité dans les pages HTML/XHTML, traitements simples sur le poste de travail de l'utilisateur
- **Moyen** : introduction de scripts dans les pages HTML/XHTML
- Norme : <http://www.ecma-international.org/publications/standards/Ecma-262.htm>

- Programmation locale

- sans javascript : programmation exécutée sur le serveur
- avec javascript : inclusion de programmes dans les pages HTML/XHTML afin de les exécuter sur le poste client

Introduction (2/6)

Motivations

- Insertion d'instructions de programmation directement dans le code des pages HTML/XHTML
- Exécution de code sur le poste client pour
 - améliorer l'interactivité (temps de réponse plus court)
 - améliorer les débits des réseaux (éviter les envois erronés, limiter les requêtes)
 - proposer des pages dynamiques (animation, personnalisation)
- Exemples :
 - test d'un formulaire avant envoi
 - animation type texte défilant
 - affichage dynamique
- **Javascript \neq scripts Java !**

Introduction (3/6)

Comparaison Javascript/Java

Comparatif

Javascript

- interprété
- orienté prototypes (héritage de propriétés)
- code intégré (visible) ⇒ à offusquer
- typage faible
- langage à part entière
- débogage difficile, maintenant facilité par l'arrivée de nouveaux outils

Java

- compilé
- orienté objets (classes, héritage)
- code dans applets (invisible)
- typage fort
- langage à part entière
- environnements de développement

Remarque

Communication possible entre Java et Javascript (plugin LiveConnect (Netscape), contrôles Active X (Microsoft), ...)

Introduction (4/6)

Avantages et inconvénients

- Points forts :
 - langage de programmation structurée ; de nombreuses applications sont maintenant développées uniquement en Javascript, côté serveur (en utilisant par exemple Node.js)
 - il enrichit le HTML/XHTML (intégré \Rightarrow interprété par le client),
 - il partage Les prototypes DOM des documents HTML/XHTML \Rightarrow manipulation dynamique possible
 - gestionnaire d'événements
- Limitations/dangers :
 - c'est un langage de script (interprété), très permissif
 - typage faible
 - ce n'est pas un langage orienté objet, mais par prototypage

Il est recommandé de bien suivre les bonnes pratiques !

Introduction (5/6)

Domaines d'application

- Javascript permet
 - de programmer des actions en fonction d'événements utilisateurs (déplacements de souris, focus, *etc.*)
 - d'accéder aux éléments de la page HTML/XHTML (traitement de formulaire, modification de la page)
 - d'effectuer des calculs sans recours au serveur
- Domaines d'application historiques :
 - petites applications simples (calculatrice, conversion, *etc.*)
 - aspects graphiques de l'interface (événements, fenêtrage, *etc.*)
 - tests de validité sur des interfaces de saisie
- Exemples de nouvelles applications possibles en Javascript :
 - Vidéos affichées en HTML5 sans Flash (ex : Youtube)
 - Jeux
 - Bureautique (ex : Google Docs)

Introduction (6/6)

Normalisation

La norme, mais...

- ECMA (European Computer Manufactures Association) a défini un standard ECMAScript (Mozilla et Adobe)
- <http://www.ecma-international.org/publications/standards/Ecma-262.htm>
- Ce standard, repris par l'ISO, définit les caractéristiques du noyau du langage
- Javascript 2.0 est conforme à cette norme mais a
 - ses propres extensions
 - des différences au niveau du modèle objet du navigateur

Inclusion dans le code (1/6)

Exemple de Javascript inséré dans du code XHTML 1.0

Exemple1.xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//FR" "DTD/xhtml1-strict.dtd"
">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Exemple de page XHTML contenant du Javascript</title>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <script type="text/javascript">
      // <![CDATA[
        function fenetre() {
          alert("Message dans une fonction.");
        }
      // ]]>
    </script>
  </head>
  <body onload="alert('Après le chargement.')">
    <script type="text/javascript">
      // <![CDATA[
        alert("Message dans le corps du document.");
      // ]]>
    </script>
    <p>
      Corps du document. <a href="Javascript:fenetre()">Message d'alerte</a>.
    </p>
  </body>
</html>
```

Inclusion dans le code (2/6)

Exemple de Javascript inséré dans du code HTML

Exemple1.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html;charset=utf-8" />
    <title>Exemple de page HTML contenant du Javascript</title>
    <script type="text/javascript">
      // <!--
      function texte() {
        document.write("Texte généré.");
      }
      // -->
    </script>
  </head>
  <body>
    <script type="text/javascript">
      // <!--
      document.write("Code javascript dans le corps du document.");
      // -->
    </script>
    <p>
      Dans une fonction appelée en cliquant <a href="javascript:texte()">ici</a>,
    <p>
      ou en passant dessus <a href="" onmouseover="javascript:texte()">cela</a>.
    </body>
  </html>
```

Inclusion dans le code (3/6)

Insertion de code Javascript

- Utilisation de la balise `<script>...</script>` :
 - déclaration de fonctions dans l'entête ou dans le footer HTML/XHTML
 - appel d'une fonction ou exécution d'une commande Javascript dans `<body>...</body>`
 - insertion d'un fichier externe (usuellement '.js')
- Utilisation dans une URL, en précisant le protocole
Ex : `Texte`
- Utilisation des attributs de balise pour la gestion événementielle :
`<balise onEvenement="instructionJavascript">...</balise>`

Remarque

Traitement séquentiel des pages par un navigateur ⇒ Placer les scripts dans le footer permet de charger les éléments visuels avant les scripts.

Inclusion dans le code (4/6)

Exemples d'inclusion

Exemple2.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>Exemple de page HTML contenant du JavaScript</title>
    <script type="text/javascript">
      // <!--
      function message() {
        alert("Ceci est un message d'alarme.");
      }
      // -->
    </script>
  </head>
  <body>
    <script type="text/javascript">
      // <!--
      prompt("Saisissez du texte", "");
      // -->
    </script>
  </body>
</html>
```

...

Inclusion dans le code (5/6)

Exemples

```
...  
  
<p onmouseover="javascript:message();" >  
  Corps du document. <a href="javascript:alarme();" >Message d'alerte</a>.  
</p>  
<footer>  
  <script type="text/javascript" src="alarme.js"></script>  
</footer>  
</body>  
</html>
```

alarme.js

```
function alarme() {  
  alert("Alerte ! Alerte ! Alerte !");  
}
```

Inclusion dans le code (6/6)

Utilisation correcte de la balise <script>

Compatibilité avec la majorité des navigateurs

- En XHTML :

```
<script type="text/javascript">  
// <br/>Code Javascript<br/>// ]&gt;<br/>&lt;/script&gt;</pre></div><div data-bbox="64 528 244 566" data-label="List-Group"><ul><li>● En HTML :</li></ul></div><div data-bbox="198 579 476 707" data-label="Text"><pre>&lt;script type="text/javascript"&gt;<br/>// &lt;!--<br/>Code Javascript<br/>// --&gt;<br/>&lt;/script&gt;</pre></div><div data-bbox="64 752 303 791" data-label="List-Group"><ul><li>● Code alternatif :</li></ul></div><div data-bbox="198 805 650 880" data-label="Text"><pre>&lt;noscript&gt;<br/>Message à afficher en cas d'absence de Javascript<br/>&lt;/noscript&gt;</pre></div><div data-bbox="628 962 988 986" data-label="Page-Footer"><img alt="Navigation icons"/></div>
```

Syntaxe générale (1/7)

Caractéristiques

- Description de la syntaxe
 - Variables faiblement typées
 - Opérateurs et instructions identiques au C/C++/Java
 - Des méthodes
 - globales (associées à tous les objets)
 - fonctions définies par l'utilisateur
 - Des "objets" (des prototypes)
 - prédéfinis (String, Date, Math, etc.)
 - liés à l'environnement
 - définis par l'utilisateur
 - Commentaires : // ou /*...*/
 - Séparateur d'instruction : ';'

Syntaxe générale (2/7)

Opérateurs

- Opérateurs identiques à ceux du C/C++/Java
 - opérateurs arithmétiques : + - * / %
 - in/décrémentation : `var++` `var--` `++var` `--var`
 - opérateurs logiques : `&&` `||` `!`
 - comparaisons : `==` `===` `!=` `!==` `<=` `<` `>=` `>`
 - concaténation de chaîne de caractères : +
 - affectation : `=` `+=` `-=` `*=` ...

Syntaxe générale (3/7)

Variables

- Utilisation de variables
 - Déclaration : `var nom[=valeur];`
 - déclaration optionnelle mais fortement conseillée
 - 'undefined' si aucune valeur à l'initialisation
 - aucun type!
 - Distinction de la localisation des variables (locale ou globale)
 - Sensible à la casse
 - Typage dynamique (à l'affectation) ⇒ transtypage

Exemples

```
var philosophe      = "Diogene";  
var anneeNaissance = -413;  
var nonRien;      // vaut undefined
```

Syntaxe générale (4/7)

Tests et boucles

Si-sinon-alors :

```
if(condition) {
  instructions
}
[else if(condition) {
  instructions
}]
[else {
  instructions
}]
```

Switch-case :

```
switch(variable) {
  case 'valeur1':
    Instructions
    break;
  ...
  default:
    Instructions
    break;
}
```

Boucles for :

```
for(i=0; i<N; i++) {
  Instructions
}

for(p in objet) {
  Instructions
}
```

Boucles while :

```
while(condition) {
  Instructions
}

do {
  Instructions
} while(condition);
```

Syntaxe générale (5/7)

Fonctions/Procédures

- Déclaration d'une fonction/procédure

```
function nom(arg1, ..., argN) {  
    Instructions  
    [return valeur;]  
}
```

- Remarques
 - Arguments et valeur en retour non typés
 - Nombre d'arguments non fixé par la déclaration
 - **Passage des paramètres par référence**

Syntaxe générale (6/7)

Les tableaux

- Déclaration :
 - `var nom = new Array([dimension]);`
 - `var nom = new Array(o1, ..., on);`
- Accession avec `[]` (ex : `tableau[i]`)
 - les indices varient de 0 à N-1
 - les éléments peuvent être de type différent
 - la taille peut changer dynamiquement
 - les tableaux à plusieurs dimensions sont possibles
- Propriétés et méthodes : `length`, `reverse()`, `sort()`, `toString()`, `push(element)`, etc.
- Tableaux associatifs : `tableau['nom']`, équivalent à `tableau.nom`
⇒ `for(var in tableau)`

Syntaxe générale (7/7)

Boîtes de dialogue et fenêtres

- Boîte de dialogue type “pop-up” :

```
window.alert("Message à afficher");
```

- Boîte de saisie simple :

```
reponse = window.prompt("texte", "chaine par défaut");
```

- Ouverture d'une fenêtre fille :

```
fenetre = window.open("page", "titre");
```

NB : si plusieurs fois le même titre, ouverture dans la même fenêtre

Remarque : ces 3 fonctions sont des méthodes de la classe window.

Les prototypes (1/10)

Notion d'objet en Javascript

- Le Javascript 'objet'
 - Pas de véritable classe, uniquement des créations d'objet et la possibilité de définir des propriétés
 - Création d'une 'classe' d'objets par la définition de son constructeur
 - Accession aux champs et méthodes : '.'
 - Un objet est un prototype et stocké comme un tableau associatif
 - Chaque "champ" peut être une variable d'instance ou une méthode :
`objet.methode()` \Leftrightarrow `objet["methode"]`
 - fonctions = objets de premier ordre
 - Héritage par prototype
 - Polymorphisme partiellement supporté

Les prototypes (2/10)

Constructeur

- Déclaration et utilisation d'une classe

```
function Rectangle(l0, l1) {  
    this.longueur = l0;  
    this.largeur = l1;  
}
```

...

```
var unRectangle = new Rectangle(20, 10);
```

...

```
var cote = unRectangle.longueur;
```

Les prototypes (3/10)

La propriété prototype

- Les classes Javascript ont une propriété `prototype` permettant d'ajouter de nouvelles propriétés ou méthodes à une classe :

```
classe.prototype.nouvellePropriété = valeur;
```

Exemple

```
Rectangle.prototype.couleur = "rouge";  
Rectangle.prototype.surface = function() {  
    return this.largeur*this.longueur;  
}  
var surf = unRectangle.surface();
```


Les prototypes (4/10)

Exemple

Prototype.html

```
<p onmouseover="javascript:message();">
  Corps du document.
</p>
</footer>
<script type="text/javascript" src="alerte.js"></script>
<script type="text/javascript">
  // <!--
  function Rectangle(lo, la) {
    this.longueur = lo;
    this.largeur = la;
  };

  Rectangle.prototype.couleur = "rouge";

  Rectangle.prototype.surface = function () {
    return this.longueur*this.largeur;
  };

  function message() {
    var unRectangle = new Rectangle(10,20);
    unRectangle.couleur = "vert";
    alert("unRectangle: "+unRectangle.couleur + " / " + unRectangle.
      surface());
  };
  // -->
</script>
</footer>
```

Les prototypes (5/10)

L'héritage de prototype

- Héritage de classe \neq Héritage de prototype
 - Dans le constructeur du prototype fils, appeler la méthode `call` du prototype parent, avec les arguments nécessaires ;
⇒ Cela permet de créer le prototype fils
 - Faire hériter attributs et méthodes :

```
ProtoFils.prototype = new ProtoParent();
```
 - Réassigner le constructeur :

```
ProtoFils.prototype.constructor = ProtoFils;
```

Les prototypes (6/10)

Exemple

Inheritance.html

```
<script type="text/javascript">
  // <!--
  function Rectangle(lo, la) {
    this.longueur = lo;
    this.largeur = la;
  };

  Rectangle.prototype.surface = function () {
    return this.longueur*this.largeur;
  };

  function Carre(c) {
    Rectangle.call(this, c, c);
  };

  Carre.prototype = new Rectangle(); // l'héritage se fait ici
  Carre.prototype.constructor = Carre;

  function message() {
    var unCarre = new Carre(10);
    alert("unCarre: " + unCarre.surface());
  };
  // -->
</script>
```

Les prototypes (7/10)

Les prototypes prédéfinis

- L'objet `Global` définit les propriétés et méthodes communes à tous les objets
 - Les méthodes et propriétés de cet objet n'appartiennent à aucune classe et cet objet n'a pas de nom
 - La seule façon de faire référence à cet objet est `this`
 - Chaque variable ou fonction globale est propriété de `Global`
 - Propriétés de `Global` : `Infinity` `NaN` `undefined`
 - Quelques méthodes : `parseFloat(s)`, `parseInt(s,base)`, `isNaN(expression)`, `eval(expression)`, `escape(URL)` et `unescape(URL)` (Codage des URL)
- Classes prédéfinies : `Array`, `Boolean`, `Date`, `Function`, `Math`, `Number`, `Image`, `Option`, `RegExp`, `String`, `Navigator`, `Window`, `Screen`

Les prototypes (8/10)

L'objet Date

- Objet Date
 - Pas de propriété
 - Liste des méthodes :

<code>getDate()</code>	<code>setDate()</code>
<code>getDay()</code>	
<code>getHours()</code>	<code>setHours()</code>
<code>getMinutes()</code>	<code>setMinutes()</code>
<code>getMonth()</code>	<code>setMonth()</code>
<code>getSeconds()</code>	<code>setSeconds()</code>
<code>getTime()</code>	<code>setTime()</code>
<code>getFullYear()</code>	<code>setYear()</code>
<code>getTimezoneOffset()</code>	<code>...</code>

Les prototypes (9/10)

String et Math

- Objet String

- Lorsqu'on définit une constante ou une variable chaîne de caractères, Javascript crée d'une façon transparente une instance String
- 1 propriété : `length`
- Les balises HTML/XHTML ont leur équivalent en méthode
- Liste (non exhaustive) des méthodes :

```
bold()      italics()   fontcolor()  fontsize()   small()     big()
toUpperCase()  toLowerCase()  sub()      sup()      substring()
charAt()     indexOf()   ...
```

- Objet Math

- Propriétés : `Math.PI` et `Math.E`
- Méthodes :

```
atan()      acos()      asin()      tan()      cos()      sin()      abs()
exp()       max()       min()       pow()      round()    sqrt()     floor()
random()    log()
```

Les prototypes (10/10)

L'objet Window

- L'objet Window
 - Cet objet représente la fenêtre courante
 - Propriétés : `defaultStatus` `frames` `length` `name` `parent` `status` `top` `window`
 - Méthodes :

<code>alert()</code>	<code>confirm()</code>
<code>prompt()</code>	<code>clear()</code>
<code>open()</code>	<code>close()</code>

Programmation événementielle (1/4)

Événements reconnus par Javascript

- `onclick` : un clic du bouton gauche de la souris sur une cible
- `onmouseover` : passage du pointeur de la souris sur une cible
- `onblur` : une perte de focus d'une cible
- `onfocus` : une activation d'une cible
- `onselect` : sélection d'une cible
- `onchange` : une modification du contenu d'une cible
- `onsubmit` : une soumission d'un formulaire
- `onload` : à la fin du chargement d'un élément
- `onunload` : la fermeture d'une fenêtre ou le chargement d'une page autre que la courante

Programmation événementielle (2/4)

Fonctionnement

- Le navigateur intercepte les événements (interruptions) et agit en conséquence
- Action → Événement → Capture → Action
- Actions associées aux cibles par les balises HTML :

```
<balise onEvenement="Action">
```

Exemple

```
<a href="" onclick="alert('Merci !');">
```

Programmation événementielle (3/4)

Événements liés à Window

- L'objet `Window` possède plusieurs méthodes spécifiques pour la gestion d'un compte à rebours :
 - `setTimeout(instruction , temps)` permet de spécifier un compteur de millisecondes associé à une instruction. Après l'intervalle de temps spécifié, une interruption est produite et l'instruction est évaluée.
 - `setInterval (instruction , temps)` permet de spécifier un compteur de millisecondes associé à une instruction. L'instruction est évaluée à intervalles réguliers.
 - `clearTimeout()` et `clearInterval ()` annulent un compte à rebours.

Exemple

```
setTimeout("window.alert(' Hello ! ');", 1000);
```

Programmation événementielle (4/4)

Fonction wait

La fonction wait n'existe pas par défaut !

⇒ Nécessité de la définir dans une fonction propre

Exemple

```
function wait(delay) {  
  var date = new Date();  
  var curDate = null;  
  do {  
    curDate = new Date();  
  } while(curDate-date<delay);  
}
```

Interaction avec HTML/XHTML : le DOM (1/4)

Objectifs

- DOM : Document Object Model.
- API pour du HTML et XML (donc XHTML) valide
- En suivant le DOM, les programmeurs peuvent construire des documents, naviguer dedans, ajouter, modifier ou effacer des éléments de ces documents
- En tant que recommandation du W3C, l'objectif du DOM est de fournir une interface de programmation standard pour être utilisée par tous (applications, OS)
- Suivant le DOM, un document a une structure logique d'arbre (ou de forêt)
- Le DOM a pour but d'uniformiser la manipulation des documents "web", notamment par les scripts.

Interaction avec HTML/XHTML : le DOM (2/4)

Liste (non exhaustive) des éléments du DOM

- Document
 - createElement(tagName)
 - getElementsByTagName(tagname)
 - getElementById(elementId)
 - createTextNode(texte)
- Document HTML/XHTML (hérite de document)
 - title : titre de la page
 - referrer, domain, URL
 - body
 - images, applets, links, forms, anchors, cookie [des collections]
- NodeList
 - length
 - item(index)

...

Interaction avec HTML/XHTML : le DOM (3/4)

Quelques éléments du DOM

...

- node
 - nodeName, nodeValue, nodeType, parentNode
 - insertBefore(newChild, refChild), replaceChild(newChild, oldChild), appendChild(newChild), removeChild(oldChild)
- element
 - tagName
- Élément HTML/XHTML
 - id
 - lang
 - dir (sens de lecture : ltr/rtl)
 - style (font-family, font-size, color, outline, etc.)

Interaction avec HTML/XHTML : le DOM (4/4)

Exemple

Exemple3.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript</title>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  </head>
  <body onload="javascript:depart();">
    <p id="paragraphe">Ceci est un paragraphe. </p>
    <footer>
      <script type="text/javascript">
        // <!--
        function depart() {
          var body = document.getElementsByTagName("body")[0];
          var texte1 = document.createTextNode("Hello !");
          body.appendChild(texte1);
          var paragraphe = document.getElementById("paragraphe");
          var texte2 = document.createTextNode("Un mot est ajouté.");
          paragraphe.appendChild(texte2);
        }
        // -->
      </script>
    </footer>
  </body>
</html>
```

Conseils de programmation

- RTFM : <http://www.ecma-international.org/publications/standards/Ecma-262.htm>
- Programmer TRÈS proprement
- Penser *Flux* et *DOM* : tous les éléments sont-ils chargés ? Ai-je modifié des noeuds ? Quels événements pour quels actions ?
- Utiliser les Plugins liés aux développement Web
 - Firebugs
 - Web Developer