Chapitre 3 : les langages de scripts coté client

Introduction :

Le Web a toujours été basé sur un principe simple : tous les types de contenu sont fournis par les serveurs Web et peuvent être récupérés par les clients via HTTP ou FTP. Les clients sont des **navigateurs** dont les plus connus sont Mozilla Firefox ou Google Chrome. Ceux-ci peuvent être installés et utilisés sur le système de l'utilisateur. Les **serveurs Web** comme Apache et NGINX sont en revanche des composants de projets Web ; ils sont aussi installés et exécutés dans cet environnement et permettent au client respectif d'accéder au contenu.

Alors qu'un contenu statique, comme par exemple les éléments HTML classiques ou les images, est simplement et facilement transmis et affiché, **le contenu dynamique** lui, comme par exemple un wiki, une liste déroulante (*drop-down list*) ou une application Web, fonctionne uniquement **à l'aide de scripts.** Ils doivent être exécutés et interprétés avec un langage de script approprié, et peuvent être réalisés côté serveur ou côté client. C'est pour cette raison que l'on distingue le script côté serveur (*server side scripting*) du script côté client (*client side scripting*).

Les langages de scripts coté client :

La technique de script côté client est utilisée par les développeurs Web pour réaliser des projets avec un contenu dynamique, les scripts programmés sont exécutés et traités par le client. Pour cela, il est nécessaire soit d'intégrer les scripts dans le document HTML ou XHTML, soit de les écrire dans un fichier séparé qui est associé au document. Si désormais l'utilisateur appelle une page Web ou une application avec un tel script côté client, le serveur Web envoie le document HTML ainsi que le script au navigateur qui exécute et présente le résultat final. Les scripts côté client peuvent également contenir des **instructions concrètes pour le navigateur Web**, comme déterminer comment il doit réagir aux actions de l'utilisateur, par exemple en cliquant sur un bouton. Souvent, le client n'a pas besoin de rétablir une connexion au serveur Web.

Etant donné que les scripts sont exécutés dans le navigateur de l'utilisateur, l'utilisateur a la possibilité de voir le code source contrairement aux scripts côté serveur. En contrepartie, l'interprétation des scripts requiert que le langage de script correspondant soit compris par le navigateur Web. Il existe plusieurs extensions de navigateur disponibles qui peuvent bloquer les scripts puisque par exemple les pop-ups et les outils de web-tracking qui sont basés sur le script côté client, peuvent avoir un impact négatif sur les temps de chargement.

Il existe plusieurs langages de script coté clients par exemple : java script, Vbscript , jquery …

Le langage de script côté client le plus important est **JavaScript**. Il a été développé par Netscape qui est le prédécesseur de Mozilla, et publié en 1995 avec la version précédente du navigateur Netscape Navigator 2.0, alors connu à cette époque sous le nom de LiveScript. Il a été diffusé rapidement et est devenu le **langage de script universel de tous les navigateurs Web pertinents.**

Le JavaScript :

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

Points forts :

- langage de programmation structurée ; de nombreuses applications sont maintenant développées uniquement en JavaScript.
- Il enrichit le HTML/XHTML (intégré ⇒ interprété par le client),
- Il partage les prototypes DOM des documents HTML/XHTML ⇒ 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

Fonctionnalité : le 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 (calculette, 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)

Inclusion dans le code : on peut insérer le code JavaScript directement dans le code html, comme on peut le maitre dans des fichiers enregistrés avec l'extension « .js ».

Exemple1 : du JavaScript inséré dans code HTML :

```
<! DOCTYPE html >
 <html >
 <head >
<title > ma page avec JAVAScript </title >
</head >
<body >
<h1>  bonjour tout le monde </h1>
<script >
        alert("alarme");
</script>
</body >
</html >
```

L'insertion du code JavaScript dans le code html se fait soit par :

1. Utilisation des balises <**script>...</script**> :
   - déclaration de fonctions dans l'entête ou dans le footer HTML
   - Appel d'une fonction ou exécution d'une commande JavaScript dans <body>...</body>
   - Insertion d'un fichier externe (usuellement '.js')
2. Utilisation dans une URL, en précisant le protocole
   Ex : Texte
3. Utilisation des attributs de balise pour la gestion événementielle :
   <**balise** onEvenement="instructionJavascript">...</**balise**>

Exemple 2 : du JavaScript inséré dans code 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("voici le code javascript1.") ;
}
</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>
```

Exempel 3 :exemple de javascript enregistré dans fichier « .js » et appelé dans du code HTML

```
<! DOCTYPE html >
<html >
<head >
<title > example d'appele   de fichier javascript </title >
</head >
<body >
<h1> bonjour tous
 <script src="Alarm.js">

</script>
</body >
</html >
```

Alarm.js

```
Alert ("ceci est une alarme");
```

Syntaxe générale du JavaScript :

Caractéristiques :

- Variables faiblement typées.
- Opérateurs et instructions identiques au C/C++/Java.
- Des fonctions/procédures
  1. globales (méthodes associées à tous les objets)
  2. fonctions/procédures/méthodes 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 : ';'

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 : = += −= ∗= ...

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 – déclarée en dehors d'une fonction-)
- Sensible à la casse
- Typage dynamique (à l'affectation) ⇒ transtypage

Exemple

```
var philosof = " soucrat " ;
var anneeNaissance = −413 ;
var nonRien ; // vaut u n d e f i n e d
```

Tests et boucles :

Les tests :

1. **Test SI- SINON-ALORS**

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

                     Instructions }]
```

2. **SWICH CASE :**

```
swich  ( variable  ) {
 case ' valeur1 ' :
                    Instructions1
break ;
……
case ' valeur n ' :
                    Instructions n
break ;

default  :
                     Instructions
break ;
}
```

3. **Boucle for:**

```
 for ( i=0 ; i<N ; i ++) {
                Instruction}
 for  ( p in tableau ) {
        Instructions }
```

4. **Boucles while et do while**

```
While  ( condition ) {
Instructions
}
do{
      instructions
} while ( condition )
```

Déclaration d'une fonction/procédure

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

Remarque :

- Arguments et valeur en retour non typés
- Nombre d'arguments non fixé par la déclarationPassage des paramètres par référence

Exemple : déclaration et appel d'une fonction

```
<script >
function message()
{
alert("ceci est message d'alarme ")
}
</script>
```

```
<script >
Message() ;
</script>
```

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

Boite 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 / d'un onglet fils :
  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.

L'objet global et les classes prédéfinis :

- **Global** définit : 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 (ou rien)
- 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 prototype :

L'objet date :

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

L'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 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 ( ).

L'objet Windows :

- Cet objet représente la fenêtre courante
- Propriétés : default, Status, frames, length, name, parent, status, top, window.

Méthodes :

alert ()	confirm ()
prompt ()	clear ()
open ()	close ()

Programmation évènementiel :

- 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 contenue 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

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 !') ">  ici <a/>
```

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.

```
window.setTimeout(alert('Merci'), 1000);
```

Les prototypes (1/6)

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 est stocké comme un tableau associatif
  - Chaque "champ" peut être une variable d'instance ou une méthode :
    objet .methode() ⇔ objet ["methode"]
  - fonctions = objets de premier ordre

- Héritage par prototype
- Polymorphisme partiellement supporté

Constructeur

Déclaration et utilisation d'une classe

```
function Rectangle (lo , la) {
this . longueur = lo;
this . largeur = la;
}
...
var unRectangle = new Rectangle (20 , 10);
...
var cote = unRectangle . longueur ;
```

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

Exemple

```
Rectangle . p r o t o t y p e . c o u l e u r = " rouge " ;
Rectangle . p r o t o t y p e . s u r f a c e = f u n c t i o n ( ) {
return this.largeur ∗ this. longueur ;
}
var surf = unRectangle . surface ( ) ;
```

Exemple :

```
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv="content-type" content="text/html;charset=utf-8" />
  <title>Exemple de page HTML contenant du JavaScript</title>
 </head>
 <body>
  <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 = "rose";
          alert("unRectangle: "+unRectangle.couleur+ " /  c moi"
+unRectangle.surface()  );
          };
    </script>
  </footer>
 </body>
</html>
```

L'héritage de prototype

Héritage de classe ≠ 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;

Exemple :

```
function Carre ( c ) {
Rectangle.call (this, c , c ) ;
} ;
Carre .prototype = new Rectangle ( ) ; // l ' héritage se fait ici
```

Carre. prototype. constructor = Carre ;