Programmation Web - Client riche

Gaëtan Rey
Gaetan.Rey@unice.fr

DUT Informatique – 2016

Objectifs

- Introduction / Rappel sur le web
- Navigateurs et DOM
- JavaScript
- Requêtes asynchrones et formats d'échanges de données
- Exemples de bibliothèques et autres langages

JAVASCRIPT

Généralités

- Langage de scripts
  - Interprété par le navigateur
- Langage sensible à la casse
- Langage orienté objet
- Langage « non/faiblement typé » (« loose typing »)
  - Attention, ça ne veut pas dire qu'il n'y a pas de types
  - Mais plus qu'on n'est pas obligé de déclarer les variables

JavaScript et HTML

- Intégration directement dans le HTML

```
<script type="text/javascript">
 /*<![CDATA[*/
  ... Insérez ici votre code JavaScript ...
 /*]]>*/
</script>
```

- Intégration dans un fichier séparé

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


Les variables et le typage (1)

- Utiliser le mot-clé var pour une déclaration explicite d'une variable
- Le typage est fait automatiquement dès la première affectation

```
var monText;                    // déclaration explicite, type undefined
var monCompteur = 0;            // déclaration explicite, type numérique
maValeur = monCompteur ;        // déclaration implicite, type numérique
monText = "0123";               // typage en chaîne de caractères
```

Les variables et le typage (2)

- Le type d'une variable n'est pas immuable, il peut être redéfini
- On parle alors de transtypage

```
var monText = "0123";          // déclaration explicite, type chaîne de caractères

i = monText + "456";           // déclaration implicite, type chaîne de caractères,
                               // i prend la valeur "0123456"

monText = i * 1;               // transtypage chaîne de caractères -> nombre,
                               // entier prend la valeur numérique 123456

var i = monText / 2;           // redéfinition explicite & transtypage
                               // chaîne de caractères -> nombre,
                               // i prend la valeur numérique 61728
```


Les variables et leur portée

- Les variables déclarées avec « var » ont une portée
  - Locale si elles sont déclarées dans une fonction
  - Globale si elles sont déclarées en dehors d'une fonction
- Les variables implicitement déclarées (sans var) sont globales (utilisable dans tout le programme)

```
var x = "0123";                // Une variable globale
y = "456";                     // Une variable globale

function maFonction() {
        var z = 23;            // Une variable locale
        t = 789;               // Une variable globale
}
```


Présentation du langage (1)

- Commentaires

```
// commentaire
/* commentaire
   multiligne */
```

- Les conditions

```
If (cond) { ... } else { ... }
```

```
cond ? expVrai : expFaux
```

Egal à	Différent de	Inférieur à	Supérieur à	Inférieur ou égal à	Supérieur ou égal à	Identiques	Non-identiques
==	!=	<	>	<=	>=	===	!==

ET PUIS (logique)	OU ALORS (logique)	NON (logique)
&&	||	!

ET (bit à bit)	OU (bit à bit)
&	|

Présentation du langage (2)

- Les boucles

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

for (var Propriété in Objet) { ... }        //on utilisera Objet[Propriété]

while (cond) { ... }

do { ... } while (cond)
```

```
var estBleu = false;
monLabel:
    for (var i=0; i<10; i++) {
        for (var j=0; j<5; j++){
            if (maVar[i][j] == "bleu" ){ estBleu = true; break monLabel;}
        }
    }
```


Présentation du langage (3)

- Les tableaux
  - Taille dynamique, peut être connue avec la propriété *length*
  - Les tableaux multidimensionnés sont possibles

```
monTab = [0,1,,,4,5];          // crée un tableau de longueur 6 avec 4 éléments
monTab = new Array(0,1,2,3);   // crée un tableau de longueur 4 avec 4 éléments
monTab = new Array(1000);      // crée un tableau vide de longueur 1000
Taille = monTab.length;
monTab = = [["essai",3],{h:3,v:12},3.7];

monTab[1];        // accès aux 2eme élément du tableau
monTab["1"];      // accès aux 2eme élément du tableau
```


Présentation du langage (4)

- Les tableaux sont aussi des objets
  - **tab.join(sep)** : retourne une chaine de caractères composé des éléments de *tab* avec *sep* comme séparateur entre élément du tableau
  - **tab.splice(i,nb)** : supprime de tab *nb* éléments à partir de l'indice *i* (i inclus dans la suppression)
  - **tab.unshift(e)** : insert l'élément *e* au début de *tab*
  - **tab.push(e)** : insert l'élément *e* à la fin de *tab*
  - **tab.concat(t)** : retourne un nouveau tableau qui est la concaténation des tableaux *tab* et *t*

Présentation du langage (5)

- Les objets
  - En JavaScript, tout est objet !
  - Deux types d'objets
    - objets de type primitif (manipulés par valeur) : nombres, booléens et chaînes
    - objets de type composé (manipulés par référence) : tableaux, fonctions ou… objets
      - ils comportent plusieurs valeurs (propriétés)

```
monObj = new Object(); // création d'un objet à l'aide du constructeur prédéfini
monObj = {};                                    // création littérale d'un objet
monObj = {nom:"Rey", prenom :"Gaetan" ,age:37};     // création littérale d'un objet

sonAge = monObj.age;          //lecture notation objet
sonNom = monObj["nom"];   //lecture notation tableau
```


Javascript et la POO

- Ce n'est pas un vrai langage orienté objet
  - Au sens java, C++, …
  - On parle de langage à prototype
- L'opérateur *this*
  - Accès aux propriétés/fonctions de l'instance d'un objet
- L'opérateur *new*
  - Construit une instance d'un objet
- Les fonctions
  - Elles sont des objets

Présentation du langage (6)

- Définitions de fonctions

```
function maFonction(arg1, …) {
            …; return exp;
}
```

- Appel de fonctions

```
var res = maFonction(arg1,…);
```

  - Omission de paramètres
    - Les paramètres omis (ceux de la fin), on la valeur *undefined*
  - Accès aux paramètres dans la fonction
    - Via leur nom
    - Via l'objet *arguments :* arguments[0], …
      - *Attention ce n'est pas un tableau*

Prototype et fonction

- Définition d'un prototype

```
Rectangle.prototype = {
  surface: function() {
    return this.largeur * this.hauteur;
  },
  perimetre: function() {
    return (this.largeur + this.hauteur) * 2;
  }
};
```

- Ajout de méthodes / extension d'un objet

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


Exemple

- La modification du prototype impacte même les instance déjà créée

```
function ExObj (){
    this.exProp = 10;
}
var ob1 = new ExObj ();
var ob2 = new ExObj ();
ExObj.prototype.newProp = 20;  // modification de toutes les instances
ob1.instProp = 30; // modification de l'instance ob1
console.log(ob1.newProp); //affiche 20
console.log(ob2.newProp); //affiche 20
console.log(ob1.instProp); //affiche 30
console.log(ob2.instProp); //undefined
```


Présentation du langage (7)

- Définitions de fonctions anonymes

```
var maVar = function (arg1, …) {
            …; return exp;
}
```

- Appel de fonctions anonymes

```
var res = maVar(arg1,…);
```

- Les types d'invocation / contexte d'invocation

```
function f () { return this; }
f();          // this === window (f est invoquée comme une fonction globale)
var obj = {'f': f };
obj.f();    // this === obj (f est invoqué comme une méthode de l'objet obj)
```

Les méthodes call et apply

- Call et Apply permettent d'appeler une fonction en passant en paramètre
  - la valeur de *this* (contexte)
  - les arguments de la fonction
    - call : une liste de paramètres
    - apply : un tableau de paramètres

```
laFonction.call(contexte, p1, p2, p3, ...);
laFonction.apply(contexte, tableau);
```

```
Math.max(8, 14, 32, 54, 2); // retourne 54
tab = [8, 14, 32, 54, 2]; // comment faire avec un tableau ?
Math.max.apply(Math, tab); //this est égal à Math, mais ça n'a pas d'impact
```


Les Fermetures (closures)

- Les fermetures sont des fonctions qui utilisent des variables libres
  - les variables de la fonction parente de la fermeture restent liées à la portée parente [Mozilla]
- Une fermeture est un objet spécial qui combine deux éléments :
  - une fonction et
  - l'environnement dans lequel la fonction a été créée
    - toutes les variables locales de la portée présente lorsque la fermeture a été créée [Mozilla]

Les Fermetures (exemples)

```
function init() {
    var nom = "Mozilla";     // nom est une variable locale créée par init
    function affNom() {      // affNom() est une fonction interne, une closure
```

```
function creerFonction() {
  var nom = "Mozilla";
  function afficheNom() {
    console.log( nom);
  }
  return afficheNom; // attention ne pas mettre () à la fin du nom de la fonction
}

var maFonction = creerFonction();
console.log(typeof maFonction);  // function
maFonction();
```


Les évènements

- Déclenche l'exécution d'une méthode sur une condition particulière
- Quelques exemples

```
// intégration simple dans le HTML
<body onload="update()">…</body>

// intégration dans le HTML avec passage de paramètre
<input type="submit" name="Submit" value="Ok" onclick="valide(this.value)" />

// intégration dans du JavaScript (attachement direct à l'objet)
document.onload=alert("message à afficher au chargement de la page");
```


Les évènements depuis le code

- Comment ajouter/supprimer la gestion d'un évènement de manière dynamique et propre ?

```
EventTarget.addEventListener(DOMString type, EventListener fonct, boolean)
EventTarget.removeEventListener(DOMString type, EventListener fonct, boolean)
```

- Paramètres :
  - type de l'évènement
    - click, mousedown, mouseup, mouseover, mousemove, mouseout, keydown, keypress, keyup, …
  - nom de la fonction de callback
    - fonction implémentant l'interface EventListener
  - true = mode capture ou false = mode propagation
    - False si le paramètre est omis

La fonction de callback

```
document.getElementById("titre").onclick = function myCallback(evt) {
    console.log(evt);
}
```

```
function myCallback(evt) {
    console.log(evt);
}
elt = document.getElementById("titre");
elt.addEventListener("click", myCallback);
```

```
elt = document.getElementById("titre");
elt.addEventListener("click", function() {
    console.log(evt);
}); // on peut omettre le dernier paramètre qui sera false par défaut
```

Event

- L'évènement passé par défaut à la callback
- Quelques propriétés
  - **event.altKey** : retourne si la touche <alt> était enfoncée au cours de l'évènement
  - **event.charCode** : retourne la valeur Unicode de la touche caractère qui a été enfoncée dans le cadre d'un évènement *keypress*
  - **event.keyCode** : retourne la valeur Unicode de toute touche non caractère dans un évènement *keypress* ou de toute touche dans les autres types d'évènement clavier
  - **event.type** : retourne le nom de l'évènement

Fermetures et évènements Exemple

- Ecrivons une fonction qui
  - ajoute n boutons dans une page

```
var b = document.createElement("button");
document.getElementById(id).appendChild(b);
```

  - affiche dans la console le numéro du bouton sur lequel on vient de cliquer

```
b.addEventListener("click", function() {
   myLog(a[i]);
});
...
function myLog(m) {
   console.log("clic sur bouton "+m);
}
```

Examinons l'exemple complet

L'objet document (1)

- Dans le DOM, l'objet document
  - fournit une manière de représenter les documents HTML, XHTML et XML.
  - implémente l'interface DOM Core Document
- Les documents HTML implémentent l'interface DOM HTMLDocument,
  - interface spécialisée pour le traitement des documents HTML
- L'objet document présente donc l'union de ces deux interfaces

L'objet document (2)

- Il se présente comme un arbre composé de nœuds
- Un nœud c'est soit
  - une balise : body, div, h1, p, …
  - Un texte : le contenu des balises mais aussi les retours à la ligne
- Chaque nœud a
  - Des propriétés le décrivant
  - Des méthodes
    - Pour se déplacer dans l'arbre
    - Pour faire d'autres actions

L'objet document (3)

- Quelques propriétés
  - **document.body** : le nœud body du document
  - **document.cookie** : la liste des cookies du document
  - **document.documentElement** : le fils direct du document. Pour les documents HTML : le nœud HTML
  - **document.head** : le nœud head du document
  - **document.images** : la liste des images du document
  - **document.links** : la liste des liens (balises « area » et « a ») du document

L'objet document (4)

- Quelques propriétés
  - **document.location** : un objet Location, contenant les informations sur l'URL du document
  - **document.referrer** : l'URI de la page de provenance
  - **document.styleSheets** : liste d'objets StyleSheet qui représentent les feuilles de style
  - **document.title** : le titre de la page (visible dans l'onglet)
  - **document.URL** : l'URL de la page

Les nœuds en détails

- Propriétés
  - **childNodes** : nœuds enfants
  - **firstChild** : premier nœud enfant
  - **lastChild** : dernier nœud enfant
  - **nextSibling** : prochain nœud de même niveau (frère suivant)
  - **parentNode** : nœud parent
  - **previousSibling** : nœud précédent de même niveau (frère précédent)
  - **nodeName** : nom du nœud
  - **nodeValue** : valeur / contenu du nœud
  - **nodeType** : type du nœud (12 types différents)
    - 1 - Nœud élément
    - 2 - Nœud attribut
    - 3 - Nœud texte
    - 4 - Nœud pour CDATA

Les méthodes de gestion des nœuds

- **document.createElement(« balise »)** : crée un nouvel élément HTML (balise) dans le document
- **document.createTextNode(« texte »)** : crée un nœud texte
- **node.appendChild(e)** : ajoute le nouvel élément dans le document. L' élément *e* sera ajouté comme étant le dernier enfant de *node*.
- **node.insertBefore(e,node)** : ajoute le nouvel élément *e* avant le nœud *node*
- **node.removeChild(e)**: supprime l'élément *e* qui est fils de *node*
- **node.replaceChild(e,old)** : remplace le fils *old* de *node* par l'élément *e*

Les méthodes de sélection de nœuds

- **document.getElementById()** : retourne l'objet correspondant à l'id spécifié
- **document.getElementsByClassName()** : retourne une liste d'objets correspondant à la classe spécifiée
- **document.getElementsByName() :** retourne une liste d'objets correspondant à la classe spécifiée
- **document.getElementsByTagName()**: retourne une liste d'objets correspondant à la classe spécifiée
- **document.querySelector() :** retourne le 1er éléments correspondant au sélecteur CSS spécifié
- **document.querySelectorAll()** : retourne une liste de tous les éléments correspondant au sélecteur CSS spéfifié

HTMLCollection

- Attention, à ne pas confondre avec la liste des nœuds
  - Ici on a la liste des « objets HTML »
- Quelques propriété (en lecture seule !!)
  - **ParentNode.childElementCount** : nombre d'enfants
  - **ParentNode.children** : liste des enfants
  - **ParentNode.firstElementChild** : premier enfant
  - **ParentNode.lastElementChild** : dernier enfant

L'objet String (1)

- Attention, il y a une différence entre
  - Le type primitif pour les chaînes de caractères
  - L'objet String

```
var sTP = "2 + 2";
var sO = new String("2 + 2");

console.log(typeof sTP); // string
console.log(typeof sO); // object

eval(sTP); // 4 (eval : évalue une expression)
eval(sO); // String {0: "2", 1: " ", 2: "+", 3: " ", 4: "2", length: 5, [[PrimitiveValue]]: "2 + 2"}
eval(sO.valueOf()); // 4 (valueOf() : retourne la valeur primitive d'un objet String
```


L'objet String (2)

- Propriétés
  - **s.length** : longueur de la chaîne de caractères
- Quelques méthodes
  - **s.charAt(i)** : retourne le caractère à la position *i*
  - **s.indexOf(c)** : retourne la 1ère position de la chaîne *c*
  - **s.lastIndexOf(c)** : retourne la dernière position de la chaîne *c*
  - **s.substring(d,f)** : Retourne la sous-chaine située entre la position d et la position f-1.
  - **s.toLowerCase()** : Transforme les lettres en minuscules
  - **s.toUpperCase()** : Transforme les lettres en majuscules

L'objet Number

```
var n = new Number("145.27");
console.log(typeof n); // object
n.toFixed(); // 145
n.toFixed(1); // 145.2
n.toFixed(2); // 145.27
n.toFixed(5); // 145.27000
n.toPrecision(); // 145.27
n.toPrecision(1); // 1e+2
n.toPrecision(2); // 1.5e+2
n.toPrecision(5); // 145
n.toExponential(); // 1.4527e+2
n.toExponential(1); // 1.5e+2
n.toExponential(2); // 1.45e+2
n.toExponential(5); // 1.45270e+2
```


L'objet Date

- Constructeurs

```
// Sans paramètre, la date est celle de la création de l'objet
new Date();
// Nombre de milliseconde depuis le 1 janvier 1970
new Date(millisecondes);
// Attention aux plages de valeurs de chaque paramètres
// Exemple le mois de 0 à 11 !
// Les arguments sont des nombres entiers et certains sont optionnels
new Date(année, mois[, jour[, heures[, minutes[, secondes[, millisecondes]]]]]);
// Chaîne représentant une date
// Attention au format de la chaîne : RFC 1123 de l'IETF ou à l'ISO8601
// Exemple : "December 17, 1995 03:24:00"
new Date(texte);
```


L'objet Date

- Quelques méthodes
  - **Date.now()** : Retourne le nombre de millisecondes depuis le 1 janvier 1970 par rapport au temps courant
  - **Date.parse(s)** : Retourne le nombre de millisecondes depuis le 1 janvier 1970 par rapport à la représentation textuelle de la date *s*
  - **d.getDate()** : Retourne le jour du mois (1-31) pour la date *d*
  - **d.getDay()** : Retourne le jour de la semaine (0-6) pour la date *d*
  - **d.getFullYear()** : Retourne l'année (avec 4 chiffres pour une année à 4 chiffres) pour la date *d*
  - **d.getHours()** : Retourne l'heure (0-23) pour la date
  - ...

L'objet Math

- Objet natif pour l'utilisation de constantes et fonctions mathématiques
- Quelques propriétés
  - **Math.E** : la valeur de la constante d'Euler ≈ 2,718
  - **Math.PI** : la valeur de Pi ≈ 3,14159
- Quelques méthodes
  - **Math.abs(n)** : retourne la valeur absolue du nombre *n*
  - **Math.ceil(n)** : retourne le plus petit entier supérieur ou égal au nombre *n*
  - **Math.floor(n)** : retourne le plus grand entier inférieur ou égal au nombre *n*
  - **Math.random()** : retourne un nombre pseudo-aléatoire compris entre 0 (inclus) et 1 (exclu)

L'objet window

- Représente le navigateur contenant l'objet document
  - Contient les propriétés et les méthodes de gestion de la fenêtre
- Les méthodes sont appelable sans spécifier l'objet window
  - window.alert(« attention !! »);
  - alert(« attention !! »);
- Quelques propriétés
  - window.name : nom de la fenêtre
  - window.status : texte de la barre de statut
- Quelques méthodes
  - print() : imprime la page
  - stop() : arrête le chargement de la page

L'objet history

- Permet de parcourir l'historique du navigateur
  - C'est un des objets de l'objet window
    - Window.history
  - N'est pas accessible en lecture (sécurité/confidentialité)
- Propriété
  - length : le nombre d'urls dans l'historique
- Méthodes
  - back() : Retourne à la page précédente
  - forward() : Avance à la page suivante
  - go() : Redirige vers une page de l'historique

Cookies en JavaScript

- Rappel
  - Un cookie n'est rien d'autre qu'un petit fichier texte stocké par le navigateur.
  - Il contient
    - Une paire nom / valeur contenant les informations
    - Une date d'expiration au-delà de laquelle il n'est plus valide
    - Un domaine et une arborescence qui indiquent quel répertoire de quel serveur y aura accès
  - Lorsque vous demandez une page Web à laquelle le cookie est associé, celui-ci est ajouté dans l'en-tête HTTP de votre REQUÊTE
    - Les programmes côté serveur peuvent alors le lire

Comment gérer des cookies en JavaScript

- Via document.cookie
  - Attention ce n'est pas une vrai chaine de caractères
  - Ajout ou modification d'un cookie

```
document.cookie = "nom1=val1; expires=Sun, 22 Feb 2015 00:00:00 UTC; path=/"
```

  - Lire un cookie (attention on a juste la paire nom=valeur)

```
var myCookList = document.cookie;
console.log(myCookList); //nom1=val1
document.cookie = "test2=val2; expires=Sun, 22 Feb 2015 00:00:00 UTC; path=/"
var myCookList = document.cookie;
console.log(myCookList); //nom1=val1; test2=val2
```


Comment gérer des cookies en JavaScript

- Via document.cookie
  - Attention ce n'est pas une vrai chaine de caractères
  - Ajout ou modification d'un cookie

```
document.cookie = "nom1=val1; expires=Sun, 22 Feb 2015 00:00:00 UTC; path=/"
```

  - Lire un cookie (attention on a juste la paire nom=valeur)

```
var myCookList = document.cookie;
console.log(myCookList); //nom1=val1
```

  - Supprimer un cookie
    - Il suffit de recréer/définir le cookie avec une date expirée

Le minimum à connaître

- Les principaux éléments du langage JavaScript
  - Objets, Fonctions, Fermetures, …
- Comment intégrer le JavaScript dans une page HTML
- Le fonctionnement des principaux objets du DOM et les types JavaScript
  - document, window, Math, Date, …

Quelques liens utiles

- Google Hosted Libraries
  - https://developers.google.com/speed/libraries/
- Microsoft Ajax Content Delivery Network
  - http://www.asp.net/ajax/cdn#jQuery_Releases_on_the_CDN_0
- Mozilla Developer Network
  - https://developer.mozilla.org/fr/
- Developpez.com
  - http://web.developpez.com

Bibliographie / Remerciement

- Merci
  - A la communauté Developpez.com
  - A la communauté Wikipédia
- Cours sur JavaScript utilisés pour monter ce module
  - Introduction au JavaScript de Serge P.
  - Développement Web : « Zone Grand Débutant » de Guillaume Rossolini
  - Cours de JavaScript de Jacques Guizol
  - Apprendre le JavaScript de Van Lancker Luc
  - Syntaxe JavaScript de [Wikipédia]
  - JavaScript sur le Mozilla Developer Network
- Autres supports utilisés
  - Comment fonctionnent les navigateurs