Institut Paul Lambin

BAC 2 Informatique de gestion

JavaScript

Synthèse JavaScript

Auteurs :
Christopher Sacré

Professeur :
D. Grolaux

24 décembre 2016

Table des matières

1 Présentation du JavaScript

1.1 Langage d'avenir

Le JavaScript est un langage créé pour le front-end Web, il peut également être utilisé au niveau back-end Web notamment à l'aide de Node.js.

1.2 Mauvaise réputation

Argument 1 : Performances médiocres

Argument 2 : Le langage est considéré comme sale, ceci est du à certaines spécificités du langage parfois inhabituelles. (Beaucoup de manières de faire la même chose, et ceci de manière très différentes).

1.3 Les bons côtés

Il existe divers moyens afin de réaliser la même opération et de ce fait parmi toutes ces manières, il en existe forcément une simple à comprendre et à utiliser

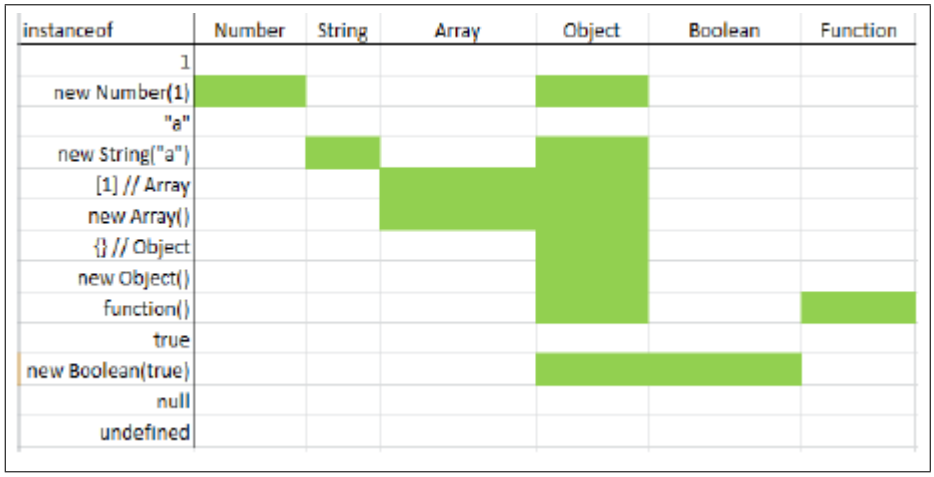
1.4 Langage interprété

Il n'y a pas de phase de compilation, c'est donc le code source qui sera directement utilisé lors de la compilation.

1.5 Langage dynamiquement typé

Le type des variables est déterminé à l'exécution, le mot clef var sert à déclarer les variables (Attention, le type d'une variable peut changer lors de l'exécution du programme).

1.6 Opérateur instanceOf



instanceof	Number	String	Array	Object	Boolean	Function
1						
new Number(1)	■			■		
"a"						
new String("a")		■		■		
[1] // Array			■	■		
new Array()			■	■		
{} // Object				■		
new Object()				■		
function()				■		■
true						
new Boolean(true)				■	■	
null						
undefined						

FIGURE 1 – Tableau montrant le résultat renvoyé par l'opérateur instanceOf

1.7 Opérateur typeOf

Fonction permettant de tester une variable et de retourner une chaîne de caractères décrivant le type de cette dernière.

	typeof
1	"number"
new Number(1)	"object"
"a"	"string"
new String("a")	"object"
[1] // Array	"object"
new Array()	"object"
{} // Object	"object"
new Object()	"object"
function()	"function"
true	"boolean"
new Boolean(true)	"object"
null	"object"
undefined	"undefined"

FIGURE 2 – Tableau montrant le résultat renvoyé par l'opérateur typeOf

1.8 Différence entre "==" et "==="

"==" effectue un transtypage faiblement typé. Ceci peut parfois donner des résultats surprenant, on évitera donc de l'utiliser. équivalent de "==" pour démontrer une inégalité : " !="

"===" n'effectue pas de transtypage, on le privilégiera donc à "==", équivalent de "===" pour démontrer une inégalité : " !==".

1.9 Les fonctions

Une fonction est considéré comme une méthode sans classe, par ailleurs les fonctions peuvent très bien être assignées au sein de variables.

Étant donné qu'une fonction peut etre affectée à des variables de noms différents, le nom de la fonction n'a pas réellement d'importance. Dans certains cas on ne nommera même pas la fonction (elle sera donc considérée comme une fonction anonyme)

C'est la présence ou l'absence de parenthèses qui détermine si on parle de la référence de la fonction, ou de son exécution.

1.10 Portée lexicale des variables

Une variable ne se restreint pas au bloc où elle est déclarée, elle se restreint plutôt à la fonction qui l'englobe.

Il faut faire attention aux variables globales car si celles sont utilisées pour définir une autre variable ce sera la référence de la variable globale qui sera retenue au sein de celle-ci et non sa valeur.

1.11 OO en JavaScript

JavaScript supporte aussi la programmation orientée objet, mais la manière dont elle fonctionne est fortement différente du java.

Les Objets JavaScript indépendamment de la programmation OO fonctionne comme des Maps. On associe une clé à une valeur, on parle alors d'objets associatifs.

1.12 Gestion des attributs

Listing 1 – Valeur d'un attribut

```
.nom // renvoie "Atreides"
["nom"] // renvoie "Atreides
```

Listing 2 – Modification d'un attribut

```
.nom="Muad'dib";
["nom"]="Kwisatz Haderach";
```

Listing 3 – Suppresion d'un attribut

```
delete o.nom;
delete o["prenom"];
```

1.13 Objets et héritages

Les attributs d'un objet peuvent provenir d'un parent de l'objet, par contre la méthode Object.keys ne renverra pas les attributs parents, si l'on désire tout de même y accéder on peut toujours utiliser un for(... in ...). On peut également savoir si un objet possède un attribut a l'aide de la méthode hasOwnProperty("attribut").

1.14 Pseudo Classe

Listing 4 – Exemple de déclaration d'une pseudo classe

```
var createPerson=function(name,surname) {
        var age,address;
        function setAddress(a) { address=a;}
        function setAge(a) { age=a;}
        var self={
                getName:function() {return name;},
                getSurname:function() {return surname;},
                getAge:function() {return age;},
                getAddress:function() {return address;},
                setAge:setAge,
                setAddress:setAddress
        }
        return self;
};
```

Listing 5 – Exemple d'utilisation d'une pseudo classe

```
var toto=createPerson("Toto","Blague");
toto.setAddress("rue de la blague, 10");
toto.setAge(7);
var jean=createPerson("Jean","Valjean");
jean.setAge(42);
```

Listing 6 – Exemple d'utilisation de l'héritage sur une pseudo classe

```
var createEtudiant=function(nom,prenom) {
```

```
        var self=createPerson(nom,prenom);
        var noma;
        self.setNoma=function(n) {noma=n;}
        self.getNoma=function() {return noma;}
        return self;
}
```

2 JavaScript dans le browser

2.1 Ajout du JavaScript au sein de notre page HTML

Listing 7 – Directement au sein de la page HTML

```
<html>
...
<script type="application/javascript">
console.log('test');
</script>
...
</html>
```

Listing 8 – Au sein d'un fichier séparé du reste de la page HTML

```
<html>
...
<script type="application/javascript" src="test.js"></script>
...
</html>
```

2.2 Méthodes / Variables utiles

- console.log : affiche au sein de la console ce qui lui est passé en paramètre (si variable affiche son contenu).
- navigator : objet contenant de l'information sur le browser lui-même, permet de l'identifier.
- window : objet manipulant la fenêtre du browser.
- document : objet manipulant la fenêtre du browser (document.writeln("argument") remplace le contenu de la pge HTML par celui passé en paramètre).
- location : objet représentant l'URL de la page. (location.href = cette URL, location.reload() recharge la page, location.replace("URL") navigue vers l'url passée en paramètre).

2.3 Document Object Model (DOM)

2.3.1 Présentation générale du DOM

Le navigateur crée de lui même une représentation objet de la page. Lorsque cet objet est manipulé par JavaScript, ce dernier se mets automatiquement à jour.

Le DOM a une structure en arbre où chaque nœud représente un élément HTML, par ailleurs certains attributs correspondent aux attributs HTML. Certaines fonctions permettent de manipuler de diverses manières le nœud et/ou d'accéder aux autres nœuds.

En pratique, traverser un arbre pour trouver un nœud en particulier est très peu souple (Si on change la moindre chose dans notre structure HTML on devra modifier l'intégralité de notre code en conséquence. Il existe donc des raccourcis facilitant l'accès à des nœuds en particulier.

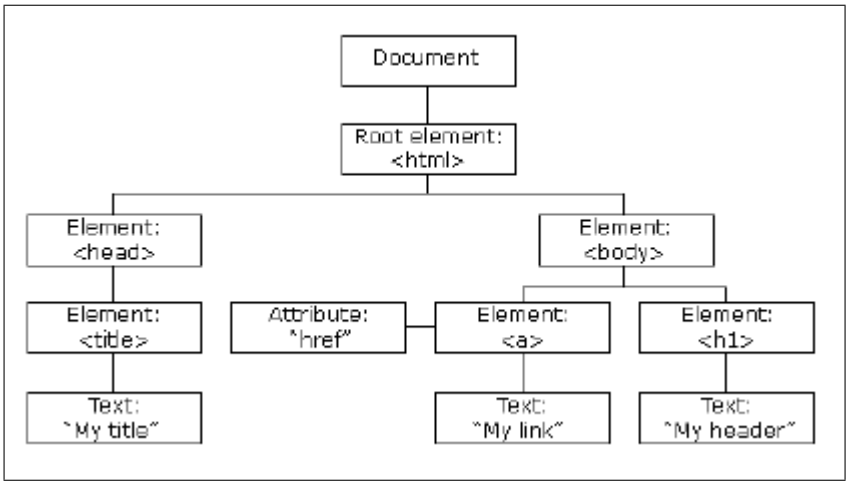


FIGURE 3 – Exemple de représentation du DOM d'une page HTML

2.3.2 Quelques méthodes utiles du DOM

- *document.getElementById(...)* : retourne l'objet portant l'id entré en paramètre
- *document.getElementsByName(...)* : retourne les objets ayant l'attribut name égal à celui passé en paramètre. (Cette méthode renvoie donc un tableau d'éléments).
- *document.getElementsByClassName(...)* : retourne les objets ayant une classe css égale à celle passée en paramètre de la méthode).
- *document.getElementsByTagName(...)* : retourne les objets dont le tag HTML est équivalent à celui entré en paramètre.

2.4 Les événements

Les événements permettent de pouvoir réagir aux actions de l'utilisateur, on enregistre alors les fonctions qui seront appelées lors d'actions spécifiques de l'utilisateur (action utilisateur = un événement sur un élément du DOM).

Les éléments HTML possèdent des attributs affectables à une fonction JavaScript.

Event	Description
onclick	The event occurs when the user clicks on an element
oncontextmenu	The event occurs when the user right-clicks on an element to open a context menu
ondblclick	The event occurs when the user double-clicks on an element
onmousedown	The event occurs when the user presses a mouse button over an element
onmouseenter	The event occurs when the pointer is moved onto an element
onmouseleave	The event occurs when the pointer is moved out of an element
onmousemove	The event occurs when the pointer is moving while it is over an element
onmouseover	The event occurs when the pointer is moved onto an element, or onto one of its children
onmouseout	The event occurs when a user moves the mouse pointer out of an element, or out of one of its children
onmouseup	The event occurs when a user releases a mouse button over an element

FIGURE 4 – Pour exemple voici les événements rattachés à la souris

Par contre, la valeur de l'attribut est unique, si on change cette valeur, la fonction précédemment affectée à cet événement ne sera plus appelée. L'HTML peut quant à lui référencé une fonction JavaScript qui n'est pas encore chargée, Ceci peut conduire a certains disfonctionnements si l'utilisateur réagit trop vite. (Afin de se prévenir de ces problèmes, on enregistre via le DOM, *document.body.addEventListener("click", function())*).

Les navigateurs ne permettent d'effectuer qu'une seule tâche à la fois (on dit qu'ils sont monothreadés). Ainsi, quand le JS s'exécute, rien d'autre ne s'exécute (pas de chargement de page, pas de rafraîchissement de la page, pas d'autres

JS, pas d'interactions avec l'utilisateur, ...).

Cas spécial, la fonction : *setTimeout(f,t)*, utilisée si l'on souhaite effectuer des actions dans le futur, .., f sera une fonction appelée après t millisecondes. (Il existe une alternative si l'on désire créer un intervalle : *setInterval(f,t)*

3 JQuery

3.1 Pourquoi le JQuery

La manipulation du DOM en JavaScript est lourde et donc vite pénible, Elle pose également des problèmes de compatibilité entre les différents navigateurs. à l'aide de JQuery, on va simplifier et normaliser tout cela.

Le JQuery permet notamment une bibliothèque JavaScript, d'assurer une compatibilité multi-navigateurs, de simplifier l'écriture de scripts, de demander au serveur de mettre à jour une partie de la page en utilisant une requête AJAX. Le JQuery est notamment utiliser par de nombreux acteurs du premier plan du Web.

Attention, il faut toujours attendre que le DOM soit entièrement obtenus avant d'appliquer du code sur notre page.

```
$(function(){
        //ici le dom est entierement defini
});
```

. Une méthode s'utilisera souvent comme suit : selecteur.méthode(paramètres).

3.2 Méthodes d'écritures

- *selecteur.html("mon message")* : va remplacer le contenu HTML de mon sélecteur et va y écrire mon message à la place.
- *selecteur.text("mon message")* : va remplacer la valeur textuelle stockée dans l'élément par le message passé en paramètre.
- *selecteur.val("mon message"* : écrit le message dans la zone texte, au contraire si il n'y a pas de paramètres : obtient la valeur entrée dans la zone texte.

Il est intéressant de noter que ces méthodes utilisée sans paramètres renvoie uniquement ce qu'elle représente sans en remplacer le contenu.

3.3 Sélection d'éléments

On peut sélectionner les nœuds du DOM qui nous intéressent à l'aide de

```
$('Selecteur')
```

où sélecteur représente un sélecteur CSS ou un sélecteur JQuery.Cette fonction, renvoie un objet JQuery qui représente un ou plusieurs éléments du DOM. (C'est une sorte de tableau, on peut notamment utiliser

```
$('element:first')
```

afin de recevoir la premier élément de la page, toutes les caractéristiques normalement applicables à un tableau le sont également pour celui-ci.

Sélecteur	$('Sélecteur') renvoie sous forme d'objet(s) jQuery
#texteJQ	la balise d'identifiant texteJQ
.bleu	toutes les balises de classe bleu
h1	toutes les balises <h1>
h1,h2	toutes les balises <h1> et <h2>

FIGURE 5 – Exemples de sélecteurs d'éléments

Sélecteur jQuery	qui renvoie sous forme d'objet(s) jQuery toutes les balises
$('ul')	
$('ul.bleu')	de classe bleu
$('div ul')	contenues dans une
$('li[class]')	qui ont un attribut class
$('li[class="impair"]')	qui ont un attribut class de valeur impair
$('div img[width="40"]')	<img> qui ont un attribut width de valeur 40 contenues dans une

FIGURE 6 – Suite des exemples de sélecteur d'éléments

3.4 Modifier le css

- *selecteur.css('attribut')* : récupère la valeur d'un attribut css.
- *selecteur.css('attribut','valeur')* : définit un attribut css.
- *selecteur.css("attribut1" :"valeur1", "attribut2" :"valeur2"* : définit plusieurs attributs CSS.

3.5 Parcourir les éléments sélectionnés

```
$('Selecteur').each(function(index) {
        // $(this) donne l'element courant
});
```

où index est une variable JavaScript qui représente la position de l'élément dans la sélection.

3.6 Listener

On peut attacher un listener sur un event à l'aide d'un sélecteur :

```
$('.a').click(function(){ alert("Click sur une balise de class a!"); });
```

On peut utiliser une manière de lier de façon générique : le *on*

```
$('.c').on("mouseenter", function(){
        $('.c').text("Mouse au-dessus");
```

});

3.7 Insertion d'éléments dans le DOM

- *.append()* : ajoute l'élément après ceux déjà présent au sein de celui passé en paramètre.
- *.prepend()* : ajoute l'élément avant ceux déjà présent au sein de celui passé en paramètre.
- *.after()* : ajoute l'élément après celui auquel la méthode est attachée.
- *.before()* : ajoute l'élément avant celui auquel la méthode est attachée.

3.8 Sélecteurs spécifiques aux formulaires

Expression	Éléments sélectionnés
:input	Tous les éléments de type input, textarea, sele
:button	Éléments de type button
:checkbox	Éléments de type checkbox
:checked	Éléments qui sont cochés
:radio	Éléments de type radio
:reset	Éléments de type reset
:image	Tous les boutons de type image
:submit	Éléments de type submit
:text	Éléments de type text
:password	Éléments de type password
:selected	Éléments sélectionnés
:focus	Sélectionne l'élément qui a le focus
:enabled	Éléments activés

FIGURE 7 – Exemple de sélecteurs de formulaires

Quelques remarques Le contraire de focus est blur, on peut générer des clicks nous mêmes : selecteur.click().

4 Chapitre 5 : JavaScript Object Notation (JSON)

4.1 JSON = variable sérialisable

Sérialisable car elle peut être transformée en une suite de bytes équivalent. (Les valeurs doivent avoir un type reconnus (booléen, null, objet, ...). Les clefs sont entourées de guillemets (obligatoire en JSON, facultatif en JavaScript). Les valeurs sont des chaînes des caractères, pas des fonctions.

4.2 Utilité de JSON

— Il s'agit d'un format standardisé.
— Il est sérialisable donc il peut être sauvé dans un fichier.
— Des librairies sont disponibles dans de nombreux langages.
— Il s'agit d'un support natif du JavaScript.

4.3 Support JavaScript pour JSON

Toute entité JSON est une valeur JavaScript, rétrospectivement, toute valeur JavaScript sérialisable est une entité JSON. *JSON.parse* transforme une chaîne de caractère en l'objet JSON correspondant, dans l'autre sens, *JSON.stringify* va transformer un objet JSON en la chaîne de caractère correspondante.

```
{
    "menu": {
        "id": "file",
        "value": "File",
        "popup": {
            "menuitem": [
                { "value": "New", "onclick": "CreateNewDoc()" },
                { "value": "Open", "onclick": "OpenDoc()" },
                { "value": "Close", "onclick": "CloseDoc()" }
            ]
        }
    }
}
```

FIGURE 8 – Exemple structure de JSON

5 HTTPServlet

5.1 Subdivisions Application Web

Une application Web est subdivisées en deux parties : le front-end (gestion du navigateur, permet d'émettre des requêtes (Au Format HTTP)) et le back-end (gestion des fichiers et des requêtes).

Le passage du front-end au back-end est une question composée de :
- Une URL
- une méthode
- une adresse d'origine
- des cookies
- des données à envoyer
- d'un mymetype pour les données à envoyer
- des en-têtes

Le passage du back-end -¿ front-end est une réponse composée de :
- Un code d'état (200 = OK, ...)
- Des données
- Un mime-type pour ces données
- Des modifications de cookies
- Des en-têtes, des informations sur la manière de cacher les entrées

5.2 Les méthodes des requêtes

GET : sert à obtenir une ressource (L'URL entrée dans le navigateur effectue un GET sur cette adresse). Elle est bookmarkable, en générale elle peut être mise en cache.

POST : sert à soumettre une réponse (Soumission d'un formulaire, les données sont envoyées en POST). Elle est non-bookmarkable, en général elle ne peut pas être mise en cache (La réponse à la soumission d'un formulaire dépend de ce formulaire et change donc à chaque fois).

Les autres méthodes *DELETE, PUT* : utilisée dans les API Webs, dans le standard REST (il est peu voir pas du tout utilisé à partir du navigateur).

5.3 Liaison avec le Java

Il nous faut un serveur pour écouter les requêtes Web et y répondre. Nous utiliserons un composant Java dédié au traitement des requêtes : Le HTTPServlet (s'utilise au sein d'un serveur Web, traite beaucoup de requêtes différentes : donc de multiples HTTPServlet sur un seul serveur).

5.4 Configuration des HTTPServlet

Il faut configurer la ou les servlets pour expliquer à java leur comportement (quels ports écouter, quels chemins dans l'URL correspondent à quels servlets, si il y a un fichier "index.html" à afficher en cas d'absence de précision, ...).

5.5 HTTPServlet

Cette classe a pour but d'être étendue, chaque requête HTTP servlet appellera la méthode :

Listing 9 – Méthode Service

```
service(HttpServletRequest i,HttpServletResponse o)
//HttpServletRequest : contient tout ce qui concerne la requete (son chemin
//HttpServletResponse : possede les methodes pour preparer la reponse a ren
//En general on ne redefinit pas directement service :
doGet(HttpServletRequest i,HttpServletResponse o)
doPost(HttpServletRequest i,HttpServletResponse o)
doPut(HttpServletRequest i,HttpServletResponse o)
doDelete(HttpServletRequest i,HttpServletResponse o)
```

5.6 HttpServletRequest

Il s'agit de tout ce qui concerne la requête, voici les méthodes que l'on peut appliquer dessus :

- *getPathInfo()* : renvoie la partie d'URL supplémentaire à ce qui est configuré pour le servlet.
- *getParameter(String)* : obtient un des paramètres de la requête, c'est à dire des données transmises.
- *getCookies()* : retourne les cookies.

5.7 HttpServletResponse

Il s'agit de tout ce qui concerne la réponse, voici les méthodes que l'on peut appliquer dessus :

- *setStatus(int)* : définit le statut de la réponse.
- *setContentLength(int)* : définit la longueur en byte de la réponse.
- *setCharacterEncoding(String)* : définit l'encodage de la réponse.
- *setContentType(String)* : définir le mimetype de la réponse
- *getOutputStream()* : retourne l'Output Stream permettant d'écrire la réponse.

5.8 Jetty

En java, sur un unique serveur Web on peut exécuter plusieurs applications distinctes (On appelle cela un 'Application Server', chaque application y est dis-

tinguée par son URL, on déploie par ailleurs son application sur un application server).

5.8.1 Jetty : Nomenclature

- Server : ce qui écoute sur un port TCP.
- WebAppContext : ce qui configure le serveur en tant qu'application server.
- ServletHolder : retient le nom et la configuration d'une instance de servlet.
- HTTPServlet : classe à étendre pour répondre aux requêtes HTTP.

5.8.2 Principe

Listing 10 – Principe Jetty

```
// lie le server a un port
Server server = new Server(8080);
// instancie un WebAppContext pour configurer le server
WebAppContext context = new WebAppContext();
// Ou se trouvent les fichiers (ils seront servis par un DefaultServlet)
context.setResourceBase("c://web");
// MaServlet repondra aux requetes commencant par /chemin/
context.addServlet(new ServletHolder(new
MaServlet()), "/chemin/*");
// Le DefaultServlet sert des fichiers (html, js, css, images, ...). Il e
context.addServlet(new ServletHolder(new DefaultServlet()),
"/");
// ce server utilise ce context
server.setHandler(context);
// allons-y
server.start();
```

5.8.3 WebAppContext

Listing 11 – Quelques méthodes du WebAppContext

```
context.setContextPath("/");
// Chemin de l'URL pour lequel ce contexte s'applique.
context.setWelcomeFiles(new String[] { "index.html" });
//Quel est le fichier a servir si l'utilisateur va a l'URL racine sans
plus de precision. www.monsite.com affichera www.monsite.com/index.html
context.setInitParameter("cacheControl","no-store,nocache, must-revalidat
//Dans le protocole HTTP, le serveur dicte le comportement du cache du
navigateur. Ici on dit que par defaut, il ne faut pas stocker ni
retenir en cache les reponses aux requetes.
```

```
context.setInitParameter("redirectWelcome", "true");
//Quand c'est 'true', la page de bienvenue est affichee par redirection p
context.setClassLoader(Thread.currentThread().getCont
extClassLoader());
//Pour des raisons de securite, il faut preciser sous quelle autorite do
context.setMaxFormContentSize(50000000);
//Specifie la taille limite des donnees qu'un frontend peut soumettre a c
context.addServlet(new ServletHolder(new MyServlet()), "/url1");
//Enregistre une servlet pour repondre a l'url /url1 exactement.
context.addServlet(new ServletHolder(new MyOtherServlet ()), "/url2/*");
//Enregistre une servlet pour repondre a toute url commencant par
/url2, Si plusieurs servlets sont ajoutees, chaque requete est traitee da
context.addServlet(new ServletHolder(new DefaultServlet()), "/");
// La DefaultServlet sert des fichiers, Ici toutes les URLs seront traite
general.
```

5.9 Configuration d'une application Web

web.xml : Description XML de la configuration de l'application. Sur un Application Server, chaque application possède son propre web.xml la concernant.

Embedded A la place d'avoir un application server sur lequel on déploie les applications, on va utiliser un processus Java qui embarque son propre application server pour y déployer son unique application. Même si cela est moins souple qu'un application server, cela est plus pratique lors du développement(Pas de phase de déploiement, c'est une pure application Java qui s'exécute, par ailleurs, les outils de debugging habituels restent fonctionnels).

5.10 Exception

5.10.1 Exceptions dans doGet ou doPost

Lorsqu'un fichier est non-trouvé : Error 404, Ceci est un traitement normal du coté Java et non une exception jetée.

Listing 12 – Gestion de l'erreur 404

```
resp.setStatus(404);
resp.setContentType("text/html");
String msg="<html><body>Fichier non
trouve</body></html>";
byte[] msgBytes=msg.getBytes("UTF-8");
resp.setContentLength(msgBytes.length);
resp.setCharacterEncoding("utf-8");
resp.getOutputStream().write(msgBytes);
```

5.10.2 Exception en Java

Si une exception s'échappe, c'est Jetty lui-même qui la gérera (redirection vers une page d'erreur ou envoi d'un message standard). En général on évitera de laisser s'échapper les exceptions.

Listing 13 – Gestion des Exceptions

```
@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
try {
....
} catch (Exception e) {
e.printStackTrace(); // pour logger l'erreur
resp.setStatus(500); // erreur au niveau du serveur
resp.setContentType("text/html");
byte[] msgBytes=e.getMessage().getBytes("UTF-8");
resp.setContentLength(msgBytes.length);
resp.setCharacterEncoding("utf-8");
resp.getOutputStream().write(msgBytes);
}
}
```

6 Chapitre 7 : Genson

6.1 Traitement du JSON en java ?

En java, il n'y a pas de support natif pour le JSON, par contre plusieurs librairies permettent de le traiter, dont Genson.

6.2 Présentation du Genson

On appellera dé sérialisation le passage d'un JSON vers du Java. Au contraire, on appellera sérialisation la transformation d'un objet Java en un objet JSON.

6.3 instanciation du Genson

Listing 14 – Directement

```
Genson genson=new Genson();
```

Listing 15 – Via un builder pour une configuration plus précise

```
Genson genson=new GensonBuilder()
.useDateFormat(new SimpleDateFormat("yyyy-MM-dd"))
.useIndentation(true)
```

```
.useConstructorWithArguments(true)
.create();
```

6.4 Genson avec des collections standards

Liste des transformations :
— *JSON objects* : Une map java ayant des chaînes de caractères pour clés.
— *JSON arrays* : Une liste ou un tableau Java.
— *JSON numbers* : Un long ou un double Java.
— *JSon String* : Une chaîne de caractère Java.
LA transformation de Java vers JSON est totalement automatique. La transformation du Java vers le JSON ne l'est par contre pas, il faut préciser vers quelle classe désérialiser (En effet l'information de classe est nécessaire en Java mais n'existe pas en JSON).

6.5 MODE Plain Old Java Object (POJO) / JavaBean

Listing 16 – Exemple de POJO

```
public class Person{
        privateString name;
        privateintage;
        privateAddressaddress;
        public Person() {} // constructeur sans argument
        public Person(String name, intage, Addressaddress) {
                this.name = name;
                this.age= age;
                this.address= address;
        }
        // getters & setters
}
        public class Address{
                public intbuilding;
                public String city;
                public Address() {}
                public Address(intbuilding, String city) {
                this.building= building;
                this.city= city;
        }
}
```

Java vers JSON

Listing 17 – Transformation Java vers JSON

```
Person someone= new Person("Eugen", 28, new Address(157, "Paris"));
```

```
String json= genson.serialize(someone);
==>
{"address":{"building":157,"city":"Paris"},"age":28,"name":"Eugen"}
```

JSON vers JAVA , Il faut préciser la classe cible.

Listing 18 – Transformation JSON vers Java

```
Person person= genson.deserialize(json, Person.class);
```

JavaBean : un JavaBean est un POJO qui est sérialisable, a un constructeur sans arguments, et permet l'accès à des propriétés utilisant des méthodes getter et setter dont les noms sont déterminés par une convention simple.

6.6 Untyped Java Structures

Pendant la sérialisation si il n'y a pas d'informations de typage (type Object), Genson va utiliser le type lors de l'exécution.

Listing 19 – Exemple de Structure Java Untyped

```
Map<String, Object> p1 = new HashMap<String, Object>() {{
        put("name", "Foo");
        put("age", 28);
}};
String json= genson.serialize(p1);
// {"age":28,"name":"Foo"}
Map<String, Object> p2 = genson.deserialize(json, Map.class);
```

6.7 Classe anonyme

Listing 20 – Exemple de classe anonyme

```
//instanciation d'une classe avec extension ce sera une classe anonyme
Personne etudiant=new Personne() {
        public String toString() {
                return "Etudiant "+super.toString();
        }
}
//equivalent a, sauf que ci-dessus la classe n'a pas le nom Etudiant mais e
class Etudiant extendsPersonne {
        public String toString() {
                return "Etudiant "+super.toString();
        }
}
Personne etudiant=new Etudiant();
```

Les classes anonymes permettent de déclarer et d'instancier une classe en même temps, elle permettent d'écrire un code plus concis et elles sont souvent utilisée dans le contexte d'écrire un Listener.

7 Chapitre 8 : Ajax

7.1 Utilité des requêtes Ajax

Avec des requêtes "normales", la réponse est de l'HTML fournie directement à partir de l'implémentation Java. Il est peu pratique de travailler de la sorte, on finis par écrire de l'HTML long et compliqués dans des Strings en Java.

7.2 JSP

Il existe une approcha classique qui permet de gérer cette manière de travailler. Utilisation de fichiers templates, les données à injecter dans le template sont retenues dans un modèle Java, des tags JSP permettent de faire le lien entre le modèle et l'HTML à intégrer et le servlet s'occupe de tout faire fonctionner ensemble.

Listing 21 – Exemple JSP

```
<HTML>
<HEAD>
<TITLE>Test</TITLE>
</HEAD>
<BODY>
<%!
int minimum(int val1, int val2) {
        if (val1 < val2) return val1;
        else return val2;
}
%>
<% int petit = minimum(5,3);%>
<p>Le plus petit de 5 et 3 est <%= petit %></p>
</BODY>
</HTML>
```

Critique du JSP : On mélange du Java dans de l'HTML, cela est difficile à maintenir, et requiert beaucoup de discipline. De plus il s'agit d'une navigation à l'ancienne (l'unité de communication est la page entière,correspond au Web classique, pas aux applications Webs modernes.), donc très verbeux et pas amusant à utiliser.

7.3 Appel Ajax

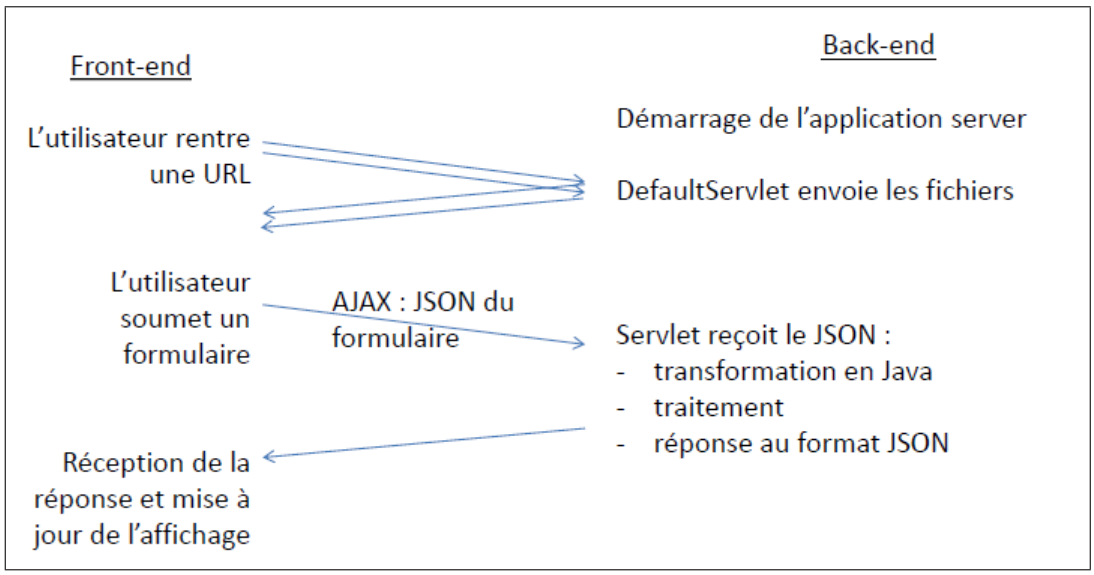


FIGURE 9 – Design Ajax

Un appel Ajax est donc une requête HTTP, mais qui ne navigue pas vers une nouvelle page. étant donné que nos requêtes Ajax communiqueront en JSON, ce dernier n'est pas à destination de l'utilisateur et donc ces requêtes seront envoyées avec la méthode POST.

Listing 22 – Exemple de requête Ajax

```
$.ajax({
        url: 'getInfo',
        type: 'POST',
        data: 'id=5',
        success: function(reponse) {
                // traitement de la reponse
        },
        error: function(e) {
                // en cas d'erreur
                console.log(e.message);
        }
});
```

Attention car uen requête HTML transporte du texte pas du JSON, il faudra donc transformer la chaîne de caractère renvoyée par le serveur en un objet JSON (on utilisera pour cela la méthode JSON.parse). Au contraire si l'on

souhaite envoyer un objet JSON au sein d'une requête il faudra le transformer d'abord en chaîne de caractères (on utilisera pour cela : JSON.stringify).

8 Chapitre 9 : L'authentification

8.1 Principe de l'authentification

L'authentification consiste à identifier un utilisateur.

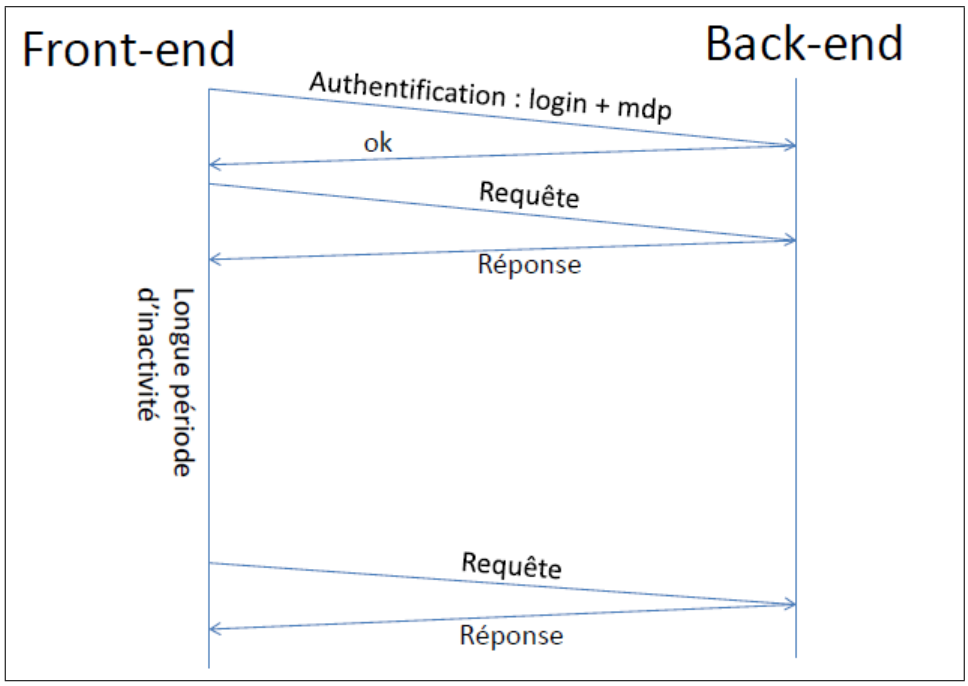


FIGURE 10 – Cycle de vie Front-end / Back-end

Le problème de l'authentification résident du fait que les requêtes du front-end dépendent de l'utilisateur authentifié, que le serveur connaît cet utilisateur au moment de son authentification et que les requêtes sont facilement manipulables. Le serveur doit donc trouver un moyen de retenir l'utilisateur une fois qu'il est authentifié pour répondre aux requêtes ultérieures correctement.

8.2 Session

L'application Server va retenir une conversation (plusieurs requêtes successives du même front-end) entre un front-end et un back-end (Attention, une conversation n'est pas une connexion TCP, l'application server fais de son mieux

pour ne pas se tromper mais des erreurs peuvent survenir).

Lors d'une authentification réussie, la session est remplie avec l'id de l'utilisateur.

```
req.getSession().setAttribute("id", idUser);
```

Ainsi, lors des requêtes ultérieures, le serveur récupère l'id à partir de la session.

```
Object idUser=req.getSession().getAttribute("id");
if (idUser!=null) {...}
```

Inconvénients de la session

- La session utilise de la mémoire pour chaque utilisateur
- Il peut y avoir de longues périodes d'inactivités du front-end (Deux possibilités : soit les sessions n'expirent jamais mais alors la mémoire n'est pas récupérée et au dans le cas contraire, les sessions expirent mais alors l'utilisateur devra se réauthentifier).
- En cas de redémarrage du serveur, toutes les données sont perdues

Ces inconvénients sont dû à une seule chose : le serveur possède un état dépendant du front-end.

La solution à cela serait un serveur Stateless : Cela serait fonctionnel et la réponse à une requête ne dépendrait que de la requête elle-même et non pas d'un état retenu sur le serveur.

Mais une requête est trop facilement modifiable, le serveur ne peut avoir confiance qu'en lui-même (on va dès lors utiliser la cryptographie).

8.3 Httpservlet HttpSession

On peut obtenir une HttpSession depuis la requête HTTP (HttpServletRequest) à l'aide de la méthode *getSession()*, cette instance sera une Map¡String, Object¿ sur lequel on peut obtenir un attribut (*getAttribute()*) et en rajouter un (*setAttribute()*)

8.4 Gestion de l'authentification : Json Web Token (JWT)

Il s'agit d'une chaîne de caractères qui retient les informations d'authentification de l'utilisateur, elle est signée cryptographiquement par le serveur. De ce fait même, seul le serveur peut générer et valider un JWT valide.

Listing 23 – Création de la chaîne JWT lors d'une authentification réussie

```
Map<String, Object> claims = new HashMap<String, Object>();
claims.put("username", email);
```

```
claims.put("id", id);
claims.put("ip", req.getRemoteAddr());
String ltoken = new JWTSigner(Config.JWTSecret).sign(claims);
```

Dans ce bout de code, JWTSecret est une chaîne de caractères connue uniquement par le serveur.

On peut mettre un id, un nom, une date de péremption, l'IP du front-end, mais on ne peut surtout pas mettre le mot de passe de l'utilisateur. De plus une chaîne JWT ne peut être créé que par le serveur (sur base de son secret caché), de ce fait même sa signature ne peut être validée que par le serveur (toujours sur base de son secret caché), par contre elle est facilement décryptable, le front-end peut en lire le contenu facilement et cela sans même connaître le secret.

JWT et cookies Le navigateur devra donc fournir la chaîne de caractères à chacune de ses requêtes. Il vaudrait donc mieux le retenir au sein d'un cookie, cela rendrait le processus automatique.

Les cookies font partie du protocole HTTP. Elles son,t placées au niveau d'un domaine, ont un nom, un contenu et peuvent avoir une date de péremption. Le front-end et le back-end peuvent ajouter/supprimer/modifier les cookies tant que le domaine est correct. Toutes les requêtes HTTP émises par le front-end contiennent tous les cookies placés sur l'URL de la page.

Listing 24 – Création d'un cookie contenant la chaîne JWT

```
Cookie cookie = new Cookie("user", ltoken);
cookie.setPath("/");
cookie.setMaxAge(60 * 60 * 24 * 365);
resp.addCookie(cookie);
```

Lors des requêtes ultérieures, on pourra aller chercher la chaîne JWT dans les cookies.

Listing 25 – Vérification de la présence du cookie

```
String token = null;
Cookie[] cookies=req.getCookies();
if (cookies != null) {
        for (Cookie c : cookies) {
                if ("user".equals(c.getName()) && c.getSecure()) {
                        token = c.getValue();
                } else if ("user".equals(c.getName()) && token == null){
                        token = c.getValue();
                }
        }
}
```

L'étape finale sera donc de valider la chaîne JWT avant de récupérer l'utilisateur.

Listing 26 – Vérification de la chaîne JWT

```
Object userID = null;
try {
        Map<String, Object> decodedPayload = new JWTVerifier(JWTSecret).ver
        userID = decodedPayload.get("id");
        if (!remoteHost.equals(decodedPayload.get("ip"))) userID=null;
} catch (Exception exception) {
// ignore
}
if (userID!=null) { // ici on a pu recuperer un utilisateur valide
...
} else { // ici pas : envoi d'une erreur ou redirection vers page d'authent
...
}
```

Authentification avec JWT , lorsque l'utilisateur rentre son login/mdp, une requête est envoyé sur une Servlet (c'est là que l'on procèder a à une vérification de son login/mdp, en cas de succès, on créera une chaîne JWT qui contient l'id/login de l'utilisateur et on placera cela dans un cookie en renvoyant un message de succès, si au contraire on est dans le cas d'un échec, on renvoie un message d'erreur). Sur base de message renvoyée par le serveur, le front-end décidera de la marche à suivre.

Dès que l'utilisateur est authentifié , chaque requête contiendra le cookie avec la chaîne JWT (Les servlets vont valider le JWT et vont récupérer l'id / le login de l'utilisateur, et effectueront par la suite leur traitement habituel).

8.5 Session et JWT

Valider la signature JWT prend du temps, on va donc le faire une fois par session uniquement. Le processus deviendra donc : Est ce que la session est authentifiée, si oui alors on peut traiter la requête, sinon on prends alors la chaîne JWT et on valide la signature. On retiendra cette authentification dans la session pour la prochaine fois et on peut traiter la demande dans le cas ou celle celle ci ne serait pas validée, on redirige vers la page d'authentification.

8.6 Chapitre Bonus : L'Orienté Objet en JavaScript

Listing 27 – Exemple d'orienté objet en JavaScript

```
function Person(name,surname) {
```

```
  var age,address;
  function setAddress(a) { address=a;}
  function setAge(a) { age=a;}
  this.getName=function() {return name;};
  this.getSurname=function() {return surname;};
  this.getAge=function() {return age;};
  this.getAddress=function() {return address;};
  this.setAge=setAge;
  this.setAddress=setAddress;
};
var john=new Person("John", "Snow");
```

L'utilisation du new crée un this lors de l'exécution de la fonction (this et new sont des mots-clés du langage, l'appel d'une fonction avec un new renvoie un this (le this représentera l'instance), la fonction sert donc de constructeur à cette instance, elle définit les propriétés de ce this notamment les fonctions utilisables.

Listing 28 – Exemple de prototype JavaScript

```
function Person(name,surname) {
        this.name=name; this.surname=surname;
}
Person.prototype.setAddress=function(a) { this.address=a;}
Person.prototype.setAge=function(a) { this.age=a;}
Person.prototype.getName=function() {return this.name;};
Person.prototype.getSurname=function() {return
this.surname;};
Person.prototype.getAge=function() {return this.age;};
Person.prototype.getAddress=function() {return
this.address;};
};
var john=new Person("John", "Snow");
```

Ici, john.getage() continue à renvoyer son âge, mais this.age est accessible via john.age également. LE principe du prototype lors de l'accès à une propriété : Si on ne trouve pas la propriété directement sur l'objet, alors JS cherche cette dernière sur le prototype de la fonction pour l'appeler.

Listing 29 – Exemple d'Héritage par prototype

```
// definition de Lanister qui herite de Person :
Lanister.prototype=new Person("Lanister","");
// redirection du constructeur par defaut (sinon celui de Person est utilis
Lanister.prototype.constructor=function(surname) {
        this.surname=surname;
}
```

9 Lecture optionnelle mais conseillée

9.1 Chapitre 1

— Enquête sur l'utilisation des différents langages
— Différences d'utilisation entre le Java, le JavaScript ainsi que le C
— Différents types de données en JavaScript
— Les Fonctions de closure
— Documentation sur la fonction push

9.2 Chapitre 2

— Documentation sur les évènements surgissant aus sein du DOM
— Documentation sur le EventListener

9.3 Chapitre 3 et 4

— Différences entre la fonction text() et html().
— Documentation sur les sélecteurs.
— Documentation sur la fonction .each().
— Documentation sur la fonction .eq().
— Documentation sur la fonction .val().
— Documentation sur les différents événements du DOM.
— Documentation sur les événements.
— Documentation sur la fonction unbind().
— Documentation globale JQuery.

9.4 Chapitre 5

— Documentation sur le JSON

9.5 Chapitre 6

— Documentation protocole HTTP
— Documentation Jetty
— Liste complètes des mime-types
— Liste des codes de statut
— Documentation sur la HttpServletRequest
— Documentation sur la HttpServletResponse

9.6 Chapitre 7

— Documentation Genson
— Guide d'utilisateur Genson Vue Globale Genson

9.7 Chapitre 8

— Documentation sur les requêtes Ajax

9.8 Chapitre 9

— Documentation sur la HttpSession
— Github pour l'utilisation des JWT au sein d'un code Java
— Documentation sur l'utilisation des cookies au sein de Java

9.9 Chapitre Bonus

— Documentation sur l'orienté objet en JavaScript

9.10 Documentation supplémentaire

— Documentation sur Bootstrap
— Documentation sur JQuery
— Documentation sur les DataTables
— Documentation sur les SELECT2
— Documentation sur les Canvas