Prénom :
Nom :
Groupe TD :
Date :

Programmation répartie

M4102

François Merciol, Didier Lesage

DUT Informatique – 2nd année

Vannes – 2021/2022

Support de cours **étudiant**

http://m4102.merciol.fr/

(page gauche vide)

Table des matières

Table des figures

Liste des tableaux

Listings

1 TD-TP 3 : Servlets

Nous entamons une série de 3 TD (3, 4 et 5) qui nous permettrons d'expérimenter le développement de pages HTML dynamique avec AJAX. Pour nous donner un but et un résultat motrant notre réussite, nous réaliserons un outil de consultation d'une arborescence de répertoire à distance. Nous procéderons en 3 étapes :

- La mise en place d'un serveur web excétutant des programmes Java (aspect serveur).
- La mise en place d'une communication AJAX avec une application de calculatrice en exemple (aspect client).
- La réalisation d'une application importante de consultation de répertoire à distance.

Rappels :

- n'oubliez pas qu'il y a un contrôle de présence. Les absences sont remontées et gérés par le secrétariat. Les retards seront comptabilisés pour être intégrés dans la note de contrôle continue.

1.1 Un peu de vocabulaire

1.1.1 *Servlet...* keskecé ?

Une *servlet* est un composant logiciel, utilisé dans un serveur *web*, qui peut être invoqué par les navigateurs clients au travers d'une URL. Le protocole de communication est dans ce cas HTTP. Même si le *web dynamique* est l'utilisation majeure de l'*API Servlet*, elle permet théoriquement de couvrir d'autres domaines d'application.

Le principe de fonctionnement est très simple : ce composant logiciel reçoit une requête, et il envoie une réponse.

Techniquement, l'*API Servlet* est un ensemble d'interfaces et de classes Java, rangées dans les packages `javax.servlet` et `javax.servlet.http`.

Dans la mesure où les *servlets* sont des composants programmés entièrement en Java, elles sont portables sur toutes les architectures munies d'une machine Java.

L'*API Servlet* a subi de nombreuses évolutions et améliorations depuis qu'elle est apparue. La première de ces évolutions est maintenant placée dans l'API JSP (Java Server Pages).

1.1.2 *web container...* et ça, kessdonc ?

Un *web container*, ou *servlet container*, est un logiciel qui est en capacité d'assurer l'exécution de *servlets*. Ces logiciels sont parfois appelés moteur web ou moteur de *servlets*.

1.1.3 *Web Application Archive...* encore un nouveau format ?

Un fichier WAR (pour *Web Application Archive*) est un fichier JAR organisé pour contenir un ensemble de *Java-Server Pages*, de *servlets*, de classes Java, de fichiers XML, de pages *Web* statiques (et fichiers joints), le tout constituant une **application web**.

Ces fichiers doivent obligatoirement contenir certains répertoires et fichiers.

Le répertoire racine d'une application *web* (celui dans lequel seront déposées toutes les ressources de l'application) est appelé **document root** de l'application. Il correspond à l'attribut **docBase** d'un **Context**, dans la configuration du serveur.

Dans le **document root**, on doit obligatoirement trouver un répertoire nommé **WEB-INF**. Ce répertoire doit en contenir deux autres, **WEB-INF/classes** et **WEB-INF/lib**. Ceux-ci contiendront respectivement les classes et les bibliothèques accessibles uniquement à cette application *web*. Ce répertoire ne doit pas nécessairement contenir que des *servlets*. Toutes les classes Java utilisées par l'application doivent s'y trouver.

Attention toutefois à respecter les packages Java.

Si la classe PremiereServlet était située dans le package `org.test.servlet`, la classe compilée devrait se trouver dans **WEB-INF/classes/org/test/servlet**.

Le document root doit également contenir un fichier `web.xml`, appelé descripteur de déploiement de l'application *web*. Il contient les différentes caractéristiques et paramètres de l'application, notamment la description des *servlets* ou des paramètres d'initialisation.

1.2 Un peu de technique

1.2.1 Le format WAR... pour livrer...

La plupart des applications J2EE sont livrées dans un fichier **WAR** et sont destinées à être déployées par une application de type *web container*. En pratique, il suffit de déposer le fichier *war* dans un répertoire spécifique du *web container* pour qu'un déploiement soit réalisé (soit au lancement du *web container*, soit automatiquement pendant son exécution, le *web container* vérifiant régulièrement la disponibilité de nouveaux fichiers *war* ou la mise à jour de fichiers déjà présents).

La plupart des *web containers* gèrent un répertoire nommé `webapps` contenant l'ensemble des applications déployées.

L'installation d'une application sur un *web container* respecte donc généralement le déroulé suivant :

- Un fichier *war* caractérisant une application est déposé dans le répertoire `webapps` du serveur
- Le contenu du fichier *war* (de même structure qu'un fichier *jar* et donc *zip*) est décompressé dans un sous-répertoire du répertoire `webapps`, dont le nom sera généralement le nom du fichier *war* sans suffixe (le dépôt du fichier `MonAppli.war` dans le répertoire `webapps` va par exemple permettre la création d'un sous-répertoire `MonAppli` contenant l'arborescence complète des fichiers contenus dans `MonAppli.war`).

1.2.2 Une fois déployée, comment accéder à une application ?

Une URL permet d'accéder aux ressources statiques et dynamiques d'une application *web*. Par exemple, une application contenue dans le sous-répertoire `MonAppli` du répertoire `webapps` sera accessible à travers une URL de la forme...

```
http://localhost/MonAppli
```

Pour une ressource statique (une page HTML), il suffit de préciser dans l'URL le chemin d'accès à la ressource dans la `WebApp`, ainsi que son nom. Par exemple...

```
http://localhost/MonAppli/rubrique1/page2.html
```

... permet d'accéder à la page HTML caractérisée par le fichier `page2.html` et rangée dans le sous-répertoire rubrique1 de l'application `MonAppli`.

Pour accéder à une *servlet*, ressource dynamique, le chemin et la ressource utilisées dans l'URL doivent correspondre au "mapping" (correspondance) qui est décrit dans le fichier de configuration `web.xml` et qui définit un lien entre classe Java et URL.

1.2.3 Web.xml... keskecé ?

Le fichier `web.xml` est essentiel au fonctionnement d'une *web* application. Appelé descripteur de déploiement, ce fichier est à installer dans le répertoire **WEB-INF** de l'application. Il contient les caractéristiques et paramètres de l'application. Cela inclut le description des *servlets* utilisés, ou les différents paramètres d'initialisation.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  id="M4201C"
  version="2.5">

<display-name>M4102C</display-name>

<description>Cours M4102C - François MERCIOL</description>

<servlet>
  <servlet-name>ServletAction</servlet-name>
  <servlet-class>com.iut.servlets.Action</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>ServletAction</servlet-name>
  <url-pattern>/action</url-pattern>
</servlet-mapping>

</web-app>
```

Listing 1 – Le fichier web.xml

Deux balises relèvent de la description de l'application. La première, `display-name` permet de donner un nom à l'application. Certains *web containers* utilisent cet identifiant pour reconnaître l'URL utilisé pour l'accès aux ressources de l'application (et ignorent donc le nom du sous-répertoire de déploiement). La balise `description` permet de fournir une description plus détaillée de l'application, tout en étant techniquement inopérante.

Les balises `servlet` permettent de décrire les *servlets* mis à disposition de l'application. Ces *servlets* sont potentiellement accessibles par les clients, au travers d'une URL. Ainsi, la balise *servlet* permet d'affecter un nom à une *servlet* et de préciser la classe Java qui en assurera la gestion (classe Java dérivée de *HttpServlet*), et la balise *servlet-mapping* permet d'indiquer au serveur quelle *servlet* charger pour telle requête du client (telle URL demandée) – les URL des *servlets* sont relatives à l'URL du `Context` (la `WebApp`) auquel elles appartiennent. Aussi, en utilisant la configuration ci-dessus, l'appel de la *servlet* nommée *ServletAction* sera par exemple possible par une utilisation de l'URL `http://localhost/M4102C/action`.

1.3 Un peu de pratique

1.3.1 Mettre en œuvre une servlet...

Les fonctionnalités d'une *servlet* sont donc gérées par une classe Java surchargeant `HttpServlet`. Pour gérer une requête http de type `GET` en provenance d'un client, une *servlet* devra donc fournir le code de la méthode doGet de sa super classe.

```
1  public void doGet(HttpServletRequest req, HttpServletResponse resp) {
     /*
     analyse de la requête

5    réalisation des traitements adéquats

     définition de la réponse fournie

     */
10 }
```

Listing 2 – Le squelette doGet

D'autres méthodes de la classe `HttpServlet` peuvent être surchargées pour gérer différents types de requêtes `HTTP` :

- `doPost()` : pour les requêtes http de type `POST`
- `doHead()` : pour les requêtes http de type `HEAD`
- `doPut()` : pour les requêtes http de type `PUT`
- `doDelete()` : pour les requêtes http de type `DELETE`
- `doOptions()` : pour les requêtes http de type `OPTIONS`
- `doTrace()` : pour les requêtes http de type `TRACE`

1.3.2 doGet... komenkonfé ?

Les opérations réalisées dans une méthode doGet sont généralement...

- La lecture des données associées à la requête (les paramètres de la requête)
- La génération des headers de la réponse
- La récupération d'une instance de `OutputStream` ou de `PrintWriter` pour permettre l'émission des données de la réponse
- L'émission des données de la réponse

1.3.3 Tapahinexampe ?

Ecrivons une simple *servlet* retournant à l'appelant un texte (avec quelques balises HTML) fournissant l'heure courante...

```
1  import java.io.*;
   import javax.servlet.*;
   import javax.servlet.http.*;

5  public class HorlogeServlet extends HttpServlet {
     public void doGet(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {
       SimpleDateFormat dateFormat = new SimpleDateFormat("HH:mm:ss");
       PrintWriter pw;
```

```
10
        response.setContentType("text/html");
        pw=response.getWriter();
        pw.println("<h1>"+ dateFormat.format(new Date())+"</h1>");
      }
15  }
```

Listing 3 – Exemple de *servlet* : HorlogeServlet

1.4 Allez... au boulot !

Tiny Java Web Server (*tjws*) est une application de type *servlet container portable*, permettant d'exploiter des *servlets*. Et comme toute application de ce type, *tjws* est en capacité de déployer des applications distribuées sous la forme de fichiers respectant le standard WAR.

Pour installer et exécuter *tjws*...

- Téléchargez l'archive d'installation
- Décompressez cette archive (un répertoire `Tjws` est créé)
- Dans le répertoire `Tjws`, éditez le fichier `tjws.sh` et complétez la déclaration de la variable d'environnement `JDK_HOME` (ou supprimez-la si votre environnement la définie déjà)
- Exécutez `./tjws.sh`

Par défaut, *Tjws* écoute sur le port TCP 8080. Vérifiez son bon fonctionnement en utilisant l'URL `http://127.0.0.1:8080`sur un navigateur.

Pour stopper le serveur *Tjws*, tapez `q`, puis validez.

1.5 Yapluka !

Installez sur votre serveur *Tjws* une application proposant une page statique (une page de bienvenue, intitulée par exemple index.html) et une *servlet* permettant de retourner la racine carrée d'un nombre entier passé en paramètre.

La *servlet* devra être en capacité d'interpréter une URL de type...

`http://127.0.0.1:8080/MonAppli/racine?entier=45`

... et de fournir une réponse texte de type...

```
1  la racine carrée de 45 est 6,7
```

1.6 Les sessions... keskecéhenkor ?

Une application *web* est basée sur le protocole http, qui est un protocole dit « sans état ». Cela signifie que le serveur, une fois qu'il a envoyé une réponse à la requête formulée par un client, ne conserve pas les données le concernant. Autrement dit, le serveur traite chaque requête qui lui parvient dans le même environnement vide d'historique... pour lui, chaque nouvelle requête provient d'un nouveau client.

Pour pallier cette lacune, qui peut être très gênante dans certaines situations, le concept de session a été créé... il permet au serveur de mémoriser des informations relatives à un client, entre deux requêtes émises par ce client.

La session représente un espace mémoire alloué pour chaque utilisateur et identifié par une valeur (un identifiant) générée par les serveurs, valeur transportée dans un *cookie* http dont le nom est `JSESSIONID`.

La méthode `getSession()` de la classe HttpServletRequest permet d'obtenir une instance de HttpSession caractérisant la session associée à l'utilisateur courant. La méthode `getID()` de HttpSession permet d'obtenir l'identifiant associé à la session.

Le contenu d'une session (une instance de `HttpSession`) est conservée par le serveur tant qu'un utilisateur n'est pas resté inactif trop longtemps ou que la session n'a pas été désactivée volontairement par la *servlet*.

Les méhodes setAttribute et getAttribute de HttpSession permettent la mise en place d'instances d'objets au sein d'une session et leur récupération.

Inkapratik ?

Modifiez-la *servlet* que vous avez développé précédemment pour faire en sorte qu'elle numérote les réponses fournies dans la cadre d'une même session utilisateur, et génère des réponses du type...

```
Calcul n° 12 : la racine carrée de 45 est 6,7
```

1.7 Kestionpratiks...

- Précisez, dans le classpath Java l'utilisation de la bibliothèque `Tjws/lib/servlet.jar`
- Organisez votre espace de travail. Par exemple, à partir du répertoire TP3...

```
TP3
├── src
│   └── com
│       └── iut
│           └── servlets
│               └── MonServlet.java
├── WEB-INF
│   ├── classes
│   ├── lib
│   └── web.xml
└── index.html
```

- Dans le répertoire TP3, générez un script de compilation intégrant une commande de type...

```
java -cp ../Tjws/lib/servlet.jar -d WEB-INF/classes src/com/iut/servlets/*
```

- Toujours dans ce répertoire, écrivez un script de génération du fichier WAR permettant de déployer l'application sur votre *web container Tjws*. Un fichier WAR respectant le format JAR, utilisez la commande `jar`.
  Exemple :

```
jar cf MonAppli.war WEB-INF
jar uf MonAppli.war index.html
```

A la fin de la séance, vous devez avoir installé un seveur *tjws* sur votre poste. vous devez être capable de créer et modifier un *servlet*. Vous devez être capable de voir le résultat dans un navigateur.

Page de notes personnelles : ✍

2 TD-TP 4 : JSon

Rappels :

- n'oubliez pas qu'il y a un contrôle de présence. Les absences sont remontées et gérés par le secrétariat. Les retards seront comptabilisés pour être intégrés dans la note de contrôle continue.

Aujourd'hui... mettons en œuvre JSON...

2.1 JSON... keskecé ?

JSON (**J**avascript **O**bject **N**otation), est une syntaxe d'échange et de stockage de données. JSON est souvent considéré comme une alternative simple à XML. Ce format est reconnu nativement par Javascript, ce qui simplifie les procédures d'échanges de données entre navigateurs et serveurs.

JSON est donc un format de données de type « texte ». Il est spécifié dans la RFC 4627. Il permet facilement de sérialiser une structure de données au format texte et est largement utilisé pour stocker des données ou échanger des données, notamment sur Internet.

JSON connaît un fort engouement, car il possède quelques points forts :

- Standard ouvert
- Syntaxe simple et compacte
- Facile à parser et à écrire
- Format offrant une structure de données compacte

La syntaxe de JSON est très simple, ce qui explique une partie de son succès. Elle ne définit que deux types de structures...

- Un objet, qui est un ensemble de paires clé/valeur
- Un tableau

JSON définit 6 types de données que la bibliothèque **JSON.simple** sait mapper nativement :

string	=>	java.lang.String
number	=>	java.lang.Number
true | false	=>	java.lang.Boolean
array	=>	java.util.List
object	=>	java.util.Map
null	=>	null

2.2 Akoissaressemble ?

Voici un exemple de structure de données au format JSON...

```
{
  "menu": "Fichier",
  "commandes": [
      {
          "titre": "Nouveau",
          "action":"CreateDoc"
      },
      {
          "titre": "Ouvrir",
          "action": "OpenDoc"
      },
      {
          "titre": "Fermer",
          "action": "CloseDoc"
      }
  ]
}
```

Listing 4 – Le source exmple.json

2.3 Akoissasserre ?

Comme nous l'avons évoqué, JSON est un format d'échange de données très facile et pratique à mettre en œuvre, notamment dans le cadre d'une exploitation du protocole HTTP. Ainsi, il est très courant de recevoir des données au format JSON en provenance d'une it servlet. L'exploitation de ces données est facilitée en **Javascript** puisque ce format est nativement interprété, contrairement au format **XML**.

2.4 Comankonfé ?

Pour produire des données au format JSON depuis une it servlet, nous allons utiliser la bibliothèque Java **JSON.simple** qui propose des classes de construction et de manipulation de données au format JSON. Deux classes principales sont à utiliser pour élaborer une structure JSON :

JSONObject
: Cette classe étend la classe **HashMap** permettant de définir facilement l'ensemble des paires nom/valeur ou listes de valeurs caractérisant un objet. Les types de base du format JSON sont les chaînes de caractères, les booléens, les nombres (entiers ou flottant), les tableaux, la valeur null... et les objets JSON. A l'aide de la bibliothèque **JSON.simple**, un tableau JSON est manipulable à l'aide d'une instance de `JSONArray`.

JSONArray
: Cette classe étend la classe ArrayList. Chaque élément d'un tableau JSON peut être l'un des types de base décrit ci-dessus.

Les deux classes possèdent chacune un constructeur sans paramètre permettant d'instancier un objet ou un tableau JSON. Elles possèdent également chacune une méthode `toJSONString()` retournant une instance de *String* dont le contenu est une représentation au format JSON de l'objet associé.

Lorsqu'une it servlet retourne des données au format JSON, le type *MIME* précisé dans l'entête de la réponse HTTP est **application/json**.

2.5 Éalorkeskondoifère ?

La première étape de ce TP consiste à mettre en œuvre une *web application* proposant une it servlet retournant une structure de données quelconque au format JSON.

L'application doit être déployée sur un serveur **Tjws**.

2.6 Éhenssuite ?

Et bien... il ne nous reste plus qu'à gérer, sur un navigateur, la réception de ces données...

Dans une page HTML, un bouton doit permettre d'émettre une requête interrogeant la it servlet de notre serveur et d'analyser les données reçues. Pour cette première étape côté client, nous nous contenterons de générer une requête, de réceptionner les données émises par la it servlet et de les produire sur la console *JavaScript* du navigateur... (ouf !).

La pratique consistant à exécuter, au sein d'une page HTML, une requête HTTP dont la finalité est l'exécution de tâches sur le serveur et/ou l'obtention de données destinées à faire évoluer le contenu de la page porte couramment le nom d'AJAX, **Asynchronous Javascript + XML**.

Ajax est seulement un nom donné à un ensemble de techniques préexistantes. Il dépend essentiellement de **XMLHttpRequest**, un objet coté client utilisable en *JavaScript*, qui est apparu avec Internet Explorer 4.0. `XMLHttpRequest` a été conçu par Mozilla sur le modèle d'un objet ActiveX nommé *XMLHTTP* créé par Microsoft. Il s'est généralisé sur les navigateurs après que le nom Ajax ait été lancé par un article de J. J. Garrett.

Ajax est une technique qui fait usage des éléments suivants :

- HTML pour l'interface.
- CSS (*Cascading Style-Sheet*) pour la présentation de la page.
- *JavaScript* pour les traitements locaux, et DOM (*Document Object Model*) qui accède aux éléments de la page ou du formulaire ou aux éléments d'un fichier XML chargé sur le serveur.
- L'objet `XMLHttpRequest` lit des données ou fichiers sur le serveur de façon asynchrone.
- PHP ou un autre langage de scripts peut être utilisé coté serveur.

Le terme “asynchronous”, asynchrone en français, signifie que l’exécution de *JavaScript* continue sans attendre la réponse du serveur qui sera traitée quand elle arrivera. Tandis qu’en mode synchrone, le navigateur serait “gelé” en attendant la réponse du serveur.

Pour recueillir des informations sur le serveur, l’objet XHR (**X**ML**H**ttp**R**equest) dispose de deux méthodes :

open établit une connexion.

send envoie une requête au serveur.

Les données retournées par le serveur seront récupérées dans les champs de l’objet XHR :

responseXml pour un fichier XML ou

responseText pour un fichier de texte brut.

La disponibilité des données et le statut de la requête sont fournis par l’attribut **readyState** de XMLHttpRequest.

Les états de **readyState** sont les suivants (seul le dernier est vraiment “utile”) :

- 0 : non initialisé.
- 1 : connexion établie.
- 2 : requête reçue.
- 3 : réponse en cours.
- 4 : terminé.

2.7 L’objet XMLHttpRequest

Attributs

readyState
ce code d’état passe successivement de 0 à 4

status
Code retour de la requête HTTP
ex : 200 est ok, 404 si la page n’est pas trouvée (voir protocole HHTP)

responseText
contient les données chargées dans une chaîne de caractères.

responseXml
contient les données chargées sous forme XML, les méthodes de DOM servent à les extraire.

onreadystatechange
propriété activée par un évènement de changement d’état. On lui assigne une fonction.

Méthodes

open(mode, url, boolean)
mode : type de requête, GET ou POST
url : l’endroit ou trouver les données, un fichier avec son chemin sur le disque.
boolean : true (asynchrone) / false (synchrone).
en option on peut ajouter un login et un mot de passe.

send("chainen")
null pour une commande GET

send("chaine")
null pour une commande GET

2.8 Mise en œuvre

2.8.1 Création d'une instance XHR

Il s'agit d'une instance de classe classique, mais deux cas de figure sont à considérer pour assurer un fonctionnement sur différents navigateurs.

```
if (window.XMLHttpRequest) {    //  Objet standard

  xhr = new XMLHttpRequest();   //  Firefox, Safari...
} else if (window.ActiveXObject) {     //  Internet Explorer

  xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
```

Listing 5 – Adaptation au navigateur

2.8.2 Gestion de la réponse

Le traitement de la réponse et les traitements qui suivent sont inclus dans une fonction, et la valeur de retour de cette fonction sera assignée à l'attribut **onreadystatechange** de l'objet précédemment créé.

```
xhr.onreadystatechange = function() {
  if (xhr.readyState == 4) {
    // Reçu, OK
  } else {
    // Attendre...
  }
};
```

Listing 6 – capture d'évènement

2.8.3 Emission de la requête

Deux méthodes de **XMLHttpRequest** sont utilisées :

- **open** : commande `GET` ou `POST`, URL du document, true pour asynchrone.
- **send** : avec `POST` seulement, données à envoyer au serveur.

La requête ci-dessous lit un document sur un serveur.

```
xhr.openn('GET', 'http://www.xul.fr/fichier.xml', true);
xhr.send(null);
```

2.9 Ajax et JSON

Si on considère qu'une structure JSON est transportée entre le serveur et le client (le navigateur) sous sa forme texte, les données retournées par le serveur au travers d'une requête Ajax seront disponibles dans l'attribut *responseText* de l'objet XHR.

Javascript met à disposition une fonction de "parsing" assurant la transformation d'une chaîne de caractères (respectant le format JSON) en une structure très facilement manipulable.

Exemple (reprenant la structure "menu" présentée avant) :

```
obj = JSON.parse (xhr.responseText);
if (obj.menu == 'Fichier') {
  for (i = 0; i < obj.commandes.length; i++) {
    ...
    ...
  }
}
```

2.9.1 Aidons-nous avec jQuery...

Nous avons présenté des fonctions JavaScript standard, et nous avons évoqué l'existence de librairies JavaScript permettant, par exemple, de s'affranchir des particularités du navigateur sur lequel les pages que nous avons conçues seront exploitées.

jQuery est l'une de ces librairies JavaScript...

2.9.2 jQuery... keskecé ?

jQuery est une librairie JavaScript libre et multiplateforme créée par *John Resig* pour faciliter le développement de scripts côté client dans le code HTML des pages web.

Cette bibliothèque propose de nombreux objets et fonctions JavaScript qui permettent, notamment...

- Le parcours et la modification du DOM
- La gestion d'évènements
- La gestion d'effets visuels
- La manipulation des feuilles de style en cascade
- L'exploitation de fonctionnalités AJAX
- La gestion de plugins
- L'accès à des fonctions utilitaires

Depuis sa création en 2006 et à cause de la complexité croissante des interfaces Web, jQuery a connu un large succès auprès des développeurs. Son apprentissage est devenu aujourd'hui l'un des fondamentaux de la formation aux technologies du Web.

2.9.3 Komenksamarche ?

jQuery se présente comme un fichier JavaScript contenant toutes les fonctions de base. Il doit être inclus à l'aide d'une balise de type...

```
1 <script src="/chemin/jQuery.js"></script>
```

Listing 7 – inclusion de code `jQuery`

La bibliothèque jQuery peut alors être appelée dans un script de deux manières différentes :

- soit via la fonction jQuery (ou la fonction $) – Cette fonction est chaînable (elle retourne l'objet appelant)
- soit via l'objet $

2.9.4 Exemples :

```
1 $("div.test").addClass("jaune").slideDown("slow");

$.each([1,2,3],function() {
   console.log(this);
5 });

$(document).ready(function() {
   console.log("Ca y est ... la page est entierement chargee !");
});
```

Listing 8 – exemple de code `jQuery`

2.9.5 Manipuler le DOM

L'une des grandes forces de jQuery est d'intégrer la syntaxe des sélecteurs CSS. Par cet intermédiaire, il est facile de sélectionner les nœuds du DOM qui nous intéressent en utilisant la syntaxe $\$(selection)$ où selection représente un sélecteur CSS.

Quelques sélecteurs CSS...

- Un nom de balise (par exemple : *div*)
- L'attribut *id* d'une balise
- La classe d'une balise

2.9.6 Exemples :

```
$("div") ...

$("#nom") ...

$(".jaune") ...
```

Listing 9 – exemple de code CSS

Les combinaisons de sélecteurs sont couramment utilisées et permettent d'affiner les sélections d'éléments du DOM.

Le résultat retourné par la fonction $\$()$ est un objet jQuery. Cet objet ressemble à un tableau. Il possède une propriété *length* et les éléments sélectionnés sont accessibles par un indice.

```
$("a")[0]
```

Listing 10 – accès au retour de jQuery

. . . retourne le premier lien hypertexte de la page courante

Une fois qu'un ou plusieurs éléments ont été sélectionnés, il est possible de leur appliquer un traitement ex exécutant une fonction jQuery.

```
$("#resultat").html('Texte a ecrire dans la balise');
```

Listing 11 – chaînage d'appel jQuery

2.9.7 Exploiter AJAX

La fonction $\$.ajax()$ de la librairie jQuery peut être utilisée pour émettre une requête http vers un serveur. Cette fonction exploite les objets JavaScript *XMLHttpRequest* ou leurs équivalents et propose une utilisation identique quelle que soit la plateforme.

Le paramètre principal de la fonction $\$.ajax()$ est l'URL qui caractérise la ressource ciblée par la requête.

```
$.ajax({
   url : 'http://localhost:8080/Demo/myServlet'
});
```

Listing 12 – appel AJAX dans HTML

D'autres paramètres sont couramment utilisés. . .

type
: Permet de préciser le type de requête HTTP à émettre (GET/POST)

data
: Données à émettre au serveur. Il peut s'agir de données complémentaires aux éventuellement déjà précisées dans l'URL (pour une requête GET)

dataType
: Type des données attendues. Par défaut, jQuery tente de déterminer automatiquement le type des données reçues en réponse à la requête.

success
: Fonction appelée en cas de succès de la requête. Des arguments spécifiques sont passés à la fonction permettant la manipulation des données reçues et l'obtention d'informations complémentaires sur les conditions d'exécution de la requête.

error
: Fonction appelée en cas d'erreur retournée par le serveur.

complete
: Fonction appelée après que la fonction success ou error ait été appelée.

2.9.8 Exemple :

```
$.ajax({
   url : 'http://localhost:8080/Demo/myServlet',
   dataType : 'json',
   type : 'GET',
   data : { name : 'John', location : 'Boston' },
   success : function(data) {
      if (data.firstname=='Bill') console.log("C'est Bill !");
```

```
    },
    error : function() {
       console.log('Grave erreur !');
    }
});
console.log("Une petite trace ...");
```

2.9.9 Petit exercice simple...

Considérant le code ci-dessus et sachant que le serveur contacté va mettre plus de 5 secondes à retourner une réponse, indiquez les diverses possibilités d'affichage de messages dans la console JavaScript.

2.9.10 Le travail à réaliser...

Il vous est demandé de mettre en œuvre une *servlet* qui doit être en capacité de calculer la racine carrée d'un entier qui lui est fourni en paramètre (vous avez normalement déjà fait cela la semaine dernière...).

La *servlet* sera sollicitée par votre navigateur lorsque le bouton "calcul" de la page *index.html* sera enfoncé par le curseur de votre souris. L'entier à utiliser par le serveur pour le calcul de la racine carrée sera lu dans un champ de saisie (balise HTML *input*) de la page. Le résultat retourné par la *servlet* sera intégré dans la page.

2.9.11 Le rendu du jour

Il vous est demandé...

- De fournir un texte explicatif sur le fonctionnement de votre application.
- De fournir une documentation sur l'utilisation de votre *servlet*. Cette documentation simple doit permettre à un développeur de la solliciter.
- De fournir le code source de votre *servlet*.
- De fournir le code source de votre page HTML cliente, ainsi que le code JavaScript développé (éventuellement intégré à la page). La syntaxe d'utilisation de la librairie *jQuery* pouvant vite devenir obscure pour un néophyte, vous ne manquerez pas de commenter votre travail.

2.9.12 Kestionpratiks...

Les librairies Java **JSON.simple** et JavaScript **jQuery** sont téléchargeables à l'adresse `http://www.pencouelo.fr/download`

Les navigateurs couramment utilisés proposent des outils intégrés d'aide au développement. Ceux-ci permettent généralement d'analyser la structure HTML d'une page, d'accéder à la console JavaScript ou encore d'analyser les échanges réalisés entre le navigateur et les serveurs interrogés.

La publication de traces sur la console Javascript du navigateur est possible à l'aide de...

```
console.log("...");
```

Pour exécuter une fonction Javascript sur le *click* d'un bouton, associer un gestionnaire d'événement *onClick* sur l'image (ou plus généralement le container) caractérisant le bouton.

Page de notes personnelles : ✍

3 TD-TP 5 : Répertoir distant

Rappels :

- N'oubliez pas qu'il y a un contrôle de présence. Les absences sont remontés et gérés par le secrétariat. Les retards seront comptabilisés pour être intégrés dans la note de contrôle continue.
- N'oubliez pas qu'un compte-rendu sera demandé en fin de dernier créneau de la semaine (donc jeudi à 13h).
- Ce travail est à faire en binôme (un seul compte-rendu par binôme) dans le cas de groupe TD (environt 28).
- Si le travail est réalisé en TP (environt 14 étudiants) vous aurez
  - un bonus à rendre seul
  - un malus si des éléments sont copier/coller d'une autre copie.

  En d'autres termes si votre travail est collectif, assumez le en rendant en groupe, mais vous êtes invitez à travailler seul pour mieux vous investir.
- Vous ne devez pas rendre un simple listing, mais un document répondant aux questions posées (qui peut contenir des partie de listing).
- Vous rappelerez vos prénoms et noms, groupe TD, date et identifierez clairement le TP.
- Vous recopierez le texte des questions (et numéro) à chaque fois (pensez au copier/coller depuis l'énoncé).
- Vous répondrez par des phrases.
- Vous recopierez les commandes que vous avez saisies pour obtenir un résultat et plus généralement ce que vous saisissez dans un terminal.
- Quand c'est nécessaire, vous pouvez faire une copie d'écran (pas l'écran entier mais uniquement la fenêtre concernée).
- Le rendu est un programme qui devra fonctionner sur une autre machine que la votre. Il contiendra
  - les fichiers binaires (war)
  - les sources Java
  - un jeu de test (des documents à consulter)
  - un document explicatif contenant des traces d'utilisation
- Pensez à ne pas mettre de chemin absolu : vous ne savez pas où l'enseignant installera votre fichiers war.

3.1 Allons plus loin...

A l'occasion des deux derniers TD/TP, nous avons pu mettre en œuvre un *servlet container*, expérimenter un format de données facilitant les échanges avec un client de type *web browser*, pour lequel nous avons d'ailleurs mis en œuvre la gestion de requêtes asynchrones au sein même du code d'une page HTML.

A l'aide de la librairie Javascript jQuery, nous avons facilement réalisé un appel AJAX depuis une page HTML.

Essayons d'aller un peu plus loin...

3.1.1 Gros exercice un peu plus compliqué...

Vous avez besoin de pouvoir accéder à distance à la hiérarchie des répertoires et fichiers de votre compte utilisateur. Vous souhaitez, depuis un navigateur...

- Lister les fichiers disponibles dans un répertoire du serveur
- Charger le contenu d'un fichier sélectionné
- Changer de répertoire (descendre/monter d'un niveau)

3.1.2 Vous avez décidé que...

- Votre navigation débuterait dans un répertoire dont le chemin est fixé arbitrairement sur le serveur (vous le nommerez répertoire racine).
- La liste des fichiers et répertoires du répertoire courant serait affichée sur le navigateur (nom, type, date de dernière modification, taille).
- Un clic sur un nom de répertoire permettrait un changement de répertoire courant (gérer le répertoire « .. »). Il ne serait pas possible de monter dans un répertoire de niveau supérieur au répertoire d'origine.
- Un clic sur un nom de fichier permettrait le chargement de ce fichier sur le navigateur (gestion de types MIME en fonction des suffixes des fichiers afin de bénéficier des fonctionnalités éventuelles de visualisation du navigateur).

3.2 Ayayaille... Parkelbouhonkomensse ?

Vous allez développer une web application qui devra être déployée sur votre serveur *Tjws*. Vraisemblablement, vous allez avoir deux parties à gérer...

3.2.1 Côté serveur

Une *servlet* doit être conçue pour répondre à, au moins, trois types de requêtes :

- L'obtention de la liste des fichiers d'un répertoire courant
- Le changement de répertoire courant (navigation relative) et la fourniture de la liste des fichiers de ce nouveau répertoire courant
- Le chargement du contenu d'un fichier du répertoire courant

3.2.2 Côté client

Une unique page HTML doit permettre de répondre à toutes les fonctionnalités. Elle doit intégrer du code JavaScript permettant l'interrogation de la *servlet*, la restitution des données transmises et la gestion des clicks de l'utilisateur sur les noms de fichiers et répertoires présentés.

3.2.3 Pour faciliter votre travail, une proposition de travail par étapes...

Décrivez, sur le papier, les différentes requêtes (précisez l'usage de chaque paramètre) auxquelles la *servlet* doit pouvoir répondre.

Déduisez-en le format des données (JSON semble une excellente solution) retournées par ces requêtes.

Prenez un moment pour consulter la documentation Java concernant les classes de manipulation de fichiers.

Développez une première version de la *servlet*, que vous testerez en utilisant directement une URL dans la barre d'adresse de votre navigateur. Cette première version fournira par exemple la liste des fichiers et répertoires du répertoire racine.

Créez la page HTML de présentation des données et de navigation dans les répertoires. Contentez vous d'une première version permettant une simple visualisation des noms de fichiers et répertoires retournés par la *servlet*.

Rendez ensuite ces noms de fichiers et répertoire *clickables*. Vérifiez l'efficacité de votre travail avec des traces sur la console JavaScript. Associez ensuite à ces *clicks* des actions d'émission de requête demandant une navigation vers un sous-répertoire (ne vous souciez pas du type de fichier dans un premier temps) et affichant le contenu du nouveau répertoire courant (si la navigation a pu être effectuée côté serveur).

Complétez la présentation de vos données sur le navigateur et le type de requête à émettre au serveur lors des *clicks* réalisés sur le navigateur.

Sur le serveur, mettre en œuvre la réponse à une demande de téléchargement de fichier reçue. Gérer une liste de types Mime associés à des suffixes caractéristiques de fichiers. En réponse à une demande de téléchargement :

- Initialiser certains headers de la réponse *HTTP (Content-Type, Content-Disposition*[1]*)*
- Récupérer le stream de sortie associé à l'instance de gestion de la réponse
- Ouvrir le fichier concerné par la demande de téléchargement et transférer son contenu dans le stream d'émission

1. Content-Disposition : voir RFC 6266

3.3 Oulalaékeskondoirendre ?

Il vous est demandé...

- De fournir un texte explicatif sur le fonctionnement de votre application. Vous exposerez les choix fonctionnels réalisés et les avantages et/ou contraintes qui en ont découlés au niveau de la programmation.
- De fournir une documentation sur l'utilisation de votre (ou vos) *servlet*. Cette documentation simple doit permettre à un développeur de réaliser votre partie client.
- De fournir le code source de votre *servlet*. Ce code devra être commenté de manière intelligente.
- De fournir le code source de votre page HTML cliente, ainsi que le code JavaScript développé (éventuellement intégré à la page). La syntaxe d'utilisation de la librairie *jQuery* pouvant vite devenir obscure pour un néophyte, vous ne manquerez pas de commenter votre travail.
- De fournir votre *web application* sous la forme d'un fichier WAR pouvant être déployé et exploité sur n'importe quelle instance de *Tjws*. Ce fichier WAR doit joint à un message envoyé aux adresses `francois.merciol@univ-ubs.fr` et `m4102@zandolite.com`, message précisant bien les noms des étudiants ayant participé (normalement, un binôme) à sa réalisation.

3.3.1 Kestionpratiks...

Les librairies Java **JSON.simple** et JavaScript **jQuery** sont téléchargeables à l'adresse `http://www.pencouelo.fr/download`.

Page de notes personnelles : ✍