TP4 - JSon

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

JSON ... keskecé ?

JSON (**J**ava**s**cript **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 librairie **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

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"
      }
   ]
```

```
}
```

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

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 *servlet*. L'exploitation de ces données est facilitée en **Javascript** puisque ce format est nativement interprété, contrairement au format **XML**.

Comankonfé ?

Pour produire des données au format JSON depuis une *servlet*, nous allons utiliser la librairie 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 librairie **JSON.simple**, un tableau JSON est manipulable à l'aide d'une instance de JSONArray.

JSONArray

Éalorkeskondoifère ?

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

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

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 *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 *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.

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("chaine")
null pour une commande GET

Mise en œuvre

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");
}
```

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...
}

};
```

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

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++) {
      …
      …
   }
}
```

Alléhoboulo !

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("…");
```

Kestionpratiks ...

La librairie Java **JSON.simple** est téléchargeable à l'adresse http://www.pencouelo.fr/download

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.

On veillera à respecter l'ordre des étapes décrites pour la mise en œuvre d'une requête Ajax en Javascript.