TD 7 : AJAX et JSON

18 novembre 2015

Objectif: Ce TD a pour but de decouvrire les syntaxes AJAX et JSON, ainsi que leur liens avec jQuery.

1 AJAX

AJAX (acronyme de ***A****synchronous* ***J****avascript* ***A****nd* ***X****ML*), n'est pas un langage, mais une façon de concevoir des applications web.

AJAX permet, en Javascript, d'appeler de façon asynchrone (c'est-à-dire indépendamment du chargement de la page) le serveur [1], qui renverra des informations qui pourront être traitées via Javascript. À la base, AJAX devait s'agir de nœuds (éléments) XML (d'où la dernière lettre du nom). En fait, on peut récupérer n'importe quel type de données : code HTML, objets JSON, du texte, etc. Une fois les données récupérées, on peut les injecter dans une partie de la page web.

Ainsi, on peut envisager modifier le contenu d'une page HTML lors d'une interaction de l'utilisateur (par exemple, lors d'un clic sur un bouton) sans avoir à la recharger entièrement. Le framework Javascript jQuery, qu'on a étudié le TD précédent, permet d'implémenter facilement les requêtes AJAX. Ce qui prendrait une dizaine de lignes en Javascript « de base » est simplifié au maximum.

Attention : En raisons de sécurité, le navigateur Google Chrome ne charge pas les fichers locals par défaut. Si vous ignorez cela, AJAX ne va pas pouvour télécharger des fichiers qui sont sur votre ordinateur. Pour résoudre ce problème :

- ouvrez avec un editeur de texte le *Unity Launcher* de Chrome, c'est-à-dire le fichier « `google-chrome.destkop` » (en principe il est situé dans le répertoire `/usr/share/applications`) ;
- changez la valeur de la propriété « `Exec` » de « `/usr/bin/google-chrome %U` » à « `/usr/bin/google-chrome --allow-access-from-files` ».

1. Dans ce cours on se concentre sur le *côté client* de la Programmation Web. Tous (ou presque) les concepts concernants le *côte serveur* qu'on aborde dans ce TD seront repris le prochain semestre.

1.1 Bases de syntaxe AJAX

AJAX utilise l'objet `XMLHttpRequest` pour échanger des données avec un serveur "dèrriere les coulisses". Cela signifie qu'il est possible de mettre à jour des parties d'une page Web, sans recharger la page entière. La syntaxe pour créer un `XMLHttpRequest` est la suivante :

```
var x = new XMLHttpRequest();
```

Pour envoyer une demande à un serveur, nous utilisons les méthodes « `open()` » et « `send()` » de nôtre `XMLHttpRequest`:

```
x.open("GET", "monTexte.txt", true);
x.send();
```

La première méthode prend trois paramètres :

- la méthode d'envoi les donnés ("GET" ou "POST") ;
- l'adresse url du fichier ;
- une valeur booleen (`true` ou `false`) correspondant à l'utilisasion asynchrone [2]

Lors de l'utilisation `asynchrone=true`, il faut aussi spécifier une fonction à exécuter lorsque la réponse est prête, c'est à dire, à l'événement « `onreadystatechange` » :

```
x.onreadystatechange = function() {
  if (x.readyState == 4 && x.status == 200) {
    // Code à executer quand la réponse est prête
  }
};
x.open("GET", "mesInfos.txt", true);
x.send();
```

Le « `readyState` » correspond au status de l'`XMLHttpRequest`. En particulier sa valeur sera :

- **0**, si la requête n'a pas (encore) été initialisée ;
- **1**, si une connexion au server a été établié ;
- **2**, si la requête a été réçue ;
- **3**, si la requête est en traitement et
- **4**, si la réponse à la requête est prête.

Le « `status` », lui, prend comme valeur :

- **200**, si tout va bien et
- **404**, si la page n'a pas été trouvée.

Pour acceder à la réponse du serveur on utilisera la méthode « `responseText()` ».

```
y = x.responseText;
```

1.2 AJAX en jQuery

Le code AJAX que l'on a vu dans la subsection precédénte peut aussi bien être écrit en jQuery. La syntaxe est la suivante :

2. Dans ce TD on le considera toujours `true`, si non ce n'est pas intéressant...

```
$.ajax({
  type: 'GET', // Type de la requête : GET ou POST
  url: 'monTexte.txt', // URL à appeler
  dataType:'text' // S'il n'est pas spécifié, le navigateur va penser que le code
                  // est en XML
  success: function(responseText){ // Fonction de callback
  responseText // contient les données renvoyées (comme avant)
  }
});
```

Cependant, ils existent des méthodes déjà implementés qui simplifient des choses plus souvent utilisées, notamment « `get()` » et « `load()` ».

Pour manipuler les données, on peut utiliser la méthode `load()`. Elle charge les données d'un serveur et met les données renvoyées directement dans l'élément sélectionné. La syntaxe est la suivante :

```
$(sélécteur).load(url,données,callback);
```

Seulement le première paramètre est obligatoire. Par exemple,

```
("div").load("monTexte.txt")
```

va ajouter le contenu du fichier « `monTexte.txt` » dans tous les éléments `div`.

La méthode « `get()` » charge les données d'un serveur, et ces données peuvent ensuite être utilisées et manipulées. La syntaxe est :

```
$.get(url,données,function(...),typeDeDonnées)
```

Par exemple,

```
$.get("monTest.txt",function(mesDonnees) {
  alert(mesDonnees);
},"text");
```

Pour plus d'information, voir aussi `http://api.jquery.com/jquery.ajax/`.

▸ **Exercice 1:** Créez un dossier « `ajax` » et dedans un fichier texte « `avignon.txt`» contenant les paroles (en format HTML) de la chanson " Sur le pont d'Avigon ".

Dans ce dossier créez une page HTML (ou copiez-collez et modifiez une déjà crée) avec un bouton ayant ID « `bouton-ajax` » et un `div` d'ID « `succes-ajax` ».

À l'aide de jQuery et d'Ajax remplissez, lors d'un clic de l'utilisateur sur un bouton, tous le `div` « `succes-ajax` » avec les paroles de la comptine.

2 JSON

JSON (***J**ava**S**cript **O**bject **N**otation*) est une syntaxe pour stocker et échanger des données. Il s'agit d'un format texte permettant d'architecturer des données. En gros, un objet JSON comprend une liste de plusieurs nœuds (ensemble « `"clé":"valeur"` ») qui peuvent s'imbriquer ou non.

Les données JSON sont sauvegardés dans un fichier `.json`, et ont le type MIME (ou *Internet media type*) `application/json`.

Les *objets* JSON sont écrits entre accolades. Ces objets peuvent contenir plusieurs paires des clés/valeurs.

Les *arrays* JSON sont écrits entre crochets. Un array JSON peut contenir plusieurs objets.

Par exemple,

```
[
  {
    "prenom": "David",
    "nom": "Beckham",
    "fils": [
      {
        "nomFils": "Brooklyn Joseph",
        "age": "16"
       },
       {
         "nomFils": "Romeo James",
         "age": "13"
       }
    ]
  },
  {
    "prenom": "Scarlett",
    "nom": "Johanson",
    "fils": [
      {
        "nomFils": "Rose Dorothy",
        "age": "1"
      }
    ]
  }
]
```

Dans l'exemple ci-dessus, l'objet JSON est un array contenant deux objets. Chaque objet correspond à une personne (avec un prénom et un nom de famille).

Attention : Pour tester si votre code JSON est valide, vous pouvez le valider en utilisant le site `http://jsonlint.com/`

▸ **Exercice 2:** Reprenez l'Exercice 10 du TD 5 (galerie d'images).

Créez maintenant un fichier « `galerie.json` ». Dans ce fichier seront stockés les objets de type « `maGallerie` ». Chaque objet contiendra comme information le titre de la galerie, une déscription, una date de création ed une liste d'images. De chaque image on connaît le titre, la source, un texte alternatif, sa date de création et les pérsonnes présentes (une liste des *tags*).

Pour avoir accès aux données JSON dans nos scripts `.js`, on utilise `getJSON()`.

Par exemple, si on suppose que les données dans exemple sont stockées dans un ficher appellé « `mesEmployes.json` », on écrira :

```
$.getJSON("mesEmployes.json", function(employes) {
  // A faire
});
```

Comme `employes` est un array, on peut accéder à sa première entrée avec `employees[0]`. Par exemple, le code

```
$.getJSON("mesEmployes.json", function(employes) {
  alert(employes[0].prenom + " " + employes[0].nom);
});
```

montre les noms et prenoms de l'employé 1.

▸ **Exercice 3:** Modifiez votre site du TD3 pour que les parties contenu (« HTML&CSS », « JavaScript », « jQuery », etc.) soient remplies à partir d'un fichier `mesContenus.json` en utilisant des requêtes AJAX. Cela sera fait par trois boutons dans votre page HTML. Un clic sur chacun de ces boutons appellera via AJAX le même fichier, mais il affichera differents contenus selon le bouton cliqué : un pour le titre, un autre pour le texte et le dernière pour le liens " Voir le TD ".

Pensez à utiliser la méthode « `html()` » (voir, par exemple, `http://www.w3schools.com/jquery/jquery_dom_get.asp`).

3 Each iteratively

La méthode « `each()` » permet de récupérer toutes les valeurs des attributs d'un objet JavaScript.

Prenons comme exemple une liste d'objets HTML :

```
<ul>
  <li>Sonja</li>
  <li>Marjatta</li>
  <li>Hiltunen</li>
</ul>
```

L'exemple ci-dessous montre comment manipuler cette liste avec la méthode « `each()` »:

```
$("#monBouton").click(function(){
  $("li").each(function(){
    alert($(this).text())
  });
});
```

Notre bouton « `monBouton` » va, quand il est cliqué, utiliser la méthode `each()` et le mot-clé « `this` » pour faire un `alert` de tous les éléments `li`.

La méthode `each()` peut aussi être utilisée pour manipuler les objets d'une facon itérative. De cetta façon, on peut accéder à tous les objets dans le fichier `.json`. Cela est applicable si, par exemple dans l'exemple avant, on veut accéder aux informations des enfants des employes.

```
$.getJSON("ajax/famille.json", function(mesDonnes) {
  $.each(mesDonnes, function(dataID,mesDonnes) {
    // Code à completer
    $.each(mesDonnes.donnesFils, function(IDFils,fils) {
      // Code à completer
    });
  });
});
```

Notez que dans l'exemple ci-dessus, `mesDonnes.donnesFils` correspond, dans le cas des employés, à `employes.fils`.

▸ **Exercice 4:** Créez un fichier `.json` qui contient un array des objets `personne` avec trois noms, prénoms, âges et les mêmes informations sur tous les i enfants de la personne i (la première personne a un enfant, la deuxième a deux enfants, etc).

Ajouter un bouton " Montrer informations ". En cliquant sur ce bouton, les informations de toutes les personnes, y inclus les information des enfants, seront montrées.

▸ **Exercice 5:** L'idée est maintenant d'utiliser le fichier `.json` que vous avez crées dans l'Eexercice 2. Écrivez un script « `scriptGalerie.js` » qui récupere l'information dans votre fichier « `galerie.jspn` » et l'utilise pour former un code HTML pour la gallerie d'images.