Soutien informatique

Langages Web côté client (2)

Lors de la séance précédente, vous avez vu comment interagir avec le contenu d'une page HTML en utilisant du code JavaScript. Dans l'exemple que vous avez développé, l'ensemble de la page HTML était envoyée par le serveur au client, et l'interaction (survol de la souris) permettait à ce client de sélectionner les éléments à afficher.

Selon cette approche, l'ensemble des données est envoyé au client qui peut ensuite les manipuler. Ceci ne pose pas de problèmes lorsque cet ensemble n'est pas très grand comme dans le cas du CV. Mais prenons l'exemple de Google Map. Les cartes sont découpées en tuiles (images carrées) et il existe une carte pour chaque niveau de zoom. Dans ce cas, il parait difficile d'envoyer toutes les images correspondant à toutes les cartes que peut afficher l'utilisateur : d'une part, le temps de chargement de l'application serait énorme et d'autre part, toutes ces cartes encombreraient inutilement l'ordinateur client. Vous avez dû aussi sans doute remarquer que lorsque l'on se déplace dans une carte Google Map, les tuiles ne s'affichent pas instantanément, il y a un petit temps d'attente. C'est parce qu'elles ne sont pas présentes sur le client. Lorsque vous vous déplacez, un code JavaScript demande au serveur les tuiles à afficher, et ce dernier les renvoie.

Cette pratique de programmation est appelée **AJAX** (*Asynchronous JavaScript And XML*). Son principe est le suivant. D'abord, le serveur envoie une première page au client contenant la structure HTML, une partie du contenu, le code JavaScript et les mises en forme. Ensuite, lorsque le client interagit avec l'application, des événements JavaScript sont déclenchés et demandent au serveur de leur renvoyer des nouvelles données. Lorsque ces données sont reçues, un nouvel évènement JavaScript est déclenché, et l'on peut ajouter un code pour modifier le contenu (ou la forme) de la page HTML en fonction de ces nouvelles données. Cette pratique est qualifiée d'**asynchrone** car l'utilisateur peut continuer d'interagir avec l'application entre l'envoi au serveur de la demande de nouvelles données et leur réception.

Exercice 1 : AJAX

Vous allez commencer par mettre en pratique ce principe de programmation en affichant un petit texte de présentation dans votre page HTML. Commencez par créer un fichier presentation.txt contenant un petit texte sur vous, que vous voudriez voir apparaitre sur votre CV.

Ensuite, avant le titre « Expériences professionnelles », ajoutez un paragraphe vide muni d'un identifiant. C'est dans ce paragraphe que vous allez ajouter votre présentation.

```
<p id="presentation"></p>
```

Il vous faut maintenant un événement permettant de déclencher la demande du fichier presentation.txt au serveur. Vous pourriez par exemple faire la demande lorsque l'utilisateur clique sur ou survole votre nom. Ici, vous allez plutôt voir un nouvel événement, `onload`, qui est déclenché par le client lorsqu'il a fini de charger la page HTML. Voici donc ce qu'il va se passer : le client va recevoir la page HTML, il va l'afficher puis déclencher la demande. Même si dans notre contexte, il n'y a pas vraiment d'intérêt à l'utiliser (on pourrait directement envoyer le texte dans le HTML) cet événement est très utilisé pour le chargement de données et vous serez

souvent amené à le manipuler. Vous allez donc modifier la création de votre `body` de la façon suivante :

```
<body onload="demanderPresentation()">
```

Traduction en français : la fonction `demanderPresentation()` est appelée automatiquement à la fin du chargement de la page HTML. Pour vous en convaincre, dans votre fichier .js, implémentez cette fonction de la façon suivante :

```
function demanderPresentation (){
  alert("Fichier chargé !")
}
```

Ouvrez votre page dans votre navigateur. Voyez-vous le *popup* ?

Vous pouvez maintenant supprimer le `alert` et le remplacer par le code permettant de demander au serveur de renvoyer au client le fichier de présentation que vous avez créé :

```
xhttp = new XMLHttpRequest(); Création de l'objet permettant la demande.
xhttp.onreadystatechange = afficherPresentation; Nom de la fonction
                                                 appelée à la réception du fichier.
xhttp.open("GET", "presentation.txt", true); Nom du fichier demandé.
xhttp.responseType = 'text'; Type du fichier demandé.
xhttp.send(); Envoi de la demande.
```

Ce code permet donc de demander le fichier presentation.txt au serveur, et d'appeler la fonction `afficherPresentation()` lorsque ce fichier est récupéré. Vous allez donc devoir créer cette fonction. Mais avant, il faut déclarer la variable `xhttp` contenant l'objet `XMLHttpRequest` comme étant globale. En effet, cette variable est nécessaire à la fonction `afficherPresentation()`pour récupérer le contenu du fichier presentation.txt. Or, pour l'instant, elle se trouve uniquement dans la fonction `demanderPresentation()` et n'est donc pas accessible dans les autres fonctions. Pour la rendre globale, il suffit de la déclarer en haut de votre fichier .js comme suit :

```
var xhttp;
```

Vous pouvez maintenant créer la fonction `afficherPresentation()` :

```
function afficherPresentation(){
  if (xhttp.readyState == 4 && xhttp.status == 200) {
    alert(xhttp.response);
  }
}
```

Dans cette fonction, les données chargées se trouvent dans le champ `response` de la variable `xhttp`. La condition du `if` permet de vérifier que le transfert s'est effectué correctement.

Testez votre code. Est-ce que le contenu de votre fichier presentation.txt apparait bien dans un *popup* ? Si oui, alors vous avez réussi à charger un fichier dynamiquement avec AJAX. Remarque : pour que votre code fonctionne, vous devez mettre vos fichiers dans le répertoire approprié de votre serveur local (MAMP, WAMP, LAMP...) et appeler cv.html avec une URL de la forme « localhost :8080/.../cv.html ».

Pour finir cet exercice, il ne reste plus qu'à ajouter le texte chargé dans le paragraphe que vous avez créé au début. Pour cela, vous allez utiliser le champ `innerHTML` disponible sur les objets

définis par des balises HTML. Il contient le contenu, en HTML, de la boite correspondant à la balise (vide du coup pour l'instant car le paragraphe ne contient rien). Remplacez donc le `alert` par le code suivant :

```
var mapres = document.getElementById("presentation");
mapres.innerHTML = xhttp.response;
```

Testez !

Exercice 2 : JSON

Dans l'exemple d'introduction (Google Map), ce sont des images qui sont demandées au serveur. Dans l'exercice 1, c'est un fichier texte. Cependant, il arrive souvent que l'on veuille charger des données avec un format plus structuré. Les deux formats les plus utilisés en Web sont **XML** et **JSON**. En voici deux exemples contenant les mêmes données :

```
<employees>
  <employee>
    <firstName>Arnaud</firstName>
    <lastName>Sallaberry</lastName>
  </employee>
  <employee>
    <firstName>Sandra</firstName>
    <lastName>Bringay</lastName>
  </employee>
  <employee>
    <firstName>Sylvain</firstName>
    <lastName>Durand</lastName>
  </employee>
</employees>
```

```
{
  "employees": [
    {
      "firstName":"Arnaud",
      "lastName":"Sallaberry"
    },
    {
      "firstName":"Sandra",
      "lastName":"Bringay"
    },
    {
      "firstName":"Sylvain",
      "lastName":"Durand"
    }
  ]
}
```

Dans ce TP, vous allez utiliser du JSON, qui, comme vous pouvez l'observer, est plus léger (moins verbeux) et plus lisible.

Vous allez commencer par récupérer le fichier exppro.json et le compléter avec vos informations. Vérifiez ensuite que votre fichier est bien formaté et le copiant à cette adresse : https://jsonlint.com/. Si ce n'est pas le cas, corrigez les erreurs si vous voulez pourvoir réaliser la suite de l'exercice.

Vous allez maintenant modifier les fonctions `afficher()` et `cacher()` pour qu'elles affichent/cachent les données provenant de fichiers JSON. Pour l'instant, vous vous contenterez de la faire pour la partie « Expériences professionnelles », donc ne vous souciez pas des erreurs qui seront générées par les autres parties.

Commencez par supprimer le contenu de ces deux fonctions dans le fichier .js et modifiez le nom de la fonction `afficher()` en `demanderContenu()`. Dans le fichier HTML, supprimez les items (boites `li`) de la liste de la section « Expériences professionnelles », mais gardez la boite `ul`. Enfin, en plus de l'identifiant de la boite `ul`, passez comme paramètre de la fonction `demanderContenu()` (ex `afficher()`) le nom du fichier à demander :

```
demanderContenu('idExPro', 'exppro.json')
```

En vous inspirant du code de l'exercice 1, modifiez la fonction `demanderContenu()` pour qu'elle demande au serveur le fichier dont le nom est passé en paramètre. Appelez la fonction appelée lors de la réception du fichier `afficherContenu`.

Créez la fonction `afficherContenu()` avec la condition vue dans l'exercice 1 et munissez la d'un `alert` affichant le contenu récupéré du serveur. Testez afin de vous assurer que vous récupérez bien le fichier.

Une commande très utile en JavaScript est `console.log()`. Elle permet d'afficher le contenu d'une variable dans la console JavaScript. Remplacez le `alert` de la fonction `afficherContenu()` par `console.log(xhttp.response);`. Ensuite, dans votre navigateur, ouvrez la console JavaScript (cf. TP précédent dernière page) et rechargez votre page HTML. Au survol de la section « Expériences professionnelles » par la souris, le contenu de la variable `xhttp.response` s'affiche dans la console sous la forme d'un texte.

Dans la fonction `demanderContenu`, remplacez maintenant `xhttp.responseType = 'text';` par `xhttp.responseType = 'json';`. Cela vous permet de créer un objet JavaScript à partir du contenu du fichier JSON, au lieu de simplement le récupérer sous forme d'une chaine de caractères. Regardez dans la console comment se présente cet objet à présent quand il est récupéré dans `afficherContenu()`.

Vous allez maintenant pouvoir remplir la liste de la section « Expériences professionnelles » avec les valeurs contenues dans l'objet. Comme nous l'avons vu dans le premier exercice, il faut pour cela récupérer la boite dans laquelle vont se trouver les items (ici c'est une boite de type `ul`) à l'aide de la méthode `getElementById`, et ensuite compléter son champ `innerHTML`. Nous sommes ici confrontés à une difficulté. En effet, la variable contenant l'identifiant de la boite `ul` n'existe pas dans la fonction `afficherContenu()`. En revanche, elle est passé en paramètre de la fonction `demanderContenu()`. Vous allez donc commencer par créer une variable globale nommée `idboite`, puis lui donner la valeur de l'identifiant de la boite dans la fonction `demanderContenu()`. Puisque cette nouvelle variable est globale, vous pouvez vous en servir pour récupérer la boite dans la fonction `afficherContenu()` :

```
var l = document.getElementById(idboite);
```

Il suffit maintenant de parcourir l'objet JavaScript pour remplir la liste. Voici comment ajouter les intitulés :

```
for(i=0 ; i<xhttp.response.length ; i++){
  l.innerHTML = l.innerHTML + "<li><strong>" +
                xhttp.response[i]["intitule"] + "</strong></li>";
}
```

Testez, sans oublier de supprimer la ligne de votre CSS qui cache les `ul` ! Si vous voyez apparaitre les intitulés au survol du titre par la souris, modifiez l'instruction du for pour ajouter aussi le descriptif, et re-testez.

Il vous reste maintenant à modifier la fonction cacher afin qu'elle supprime le contenu (`innerHTML` de la boite `ul`) lorsque la souris ne survole plus le titre :

```
var l = document.getElementById(idboite);
l.innerHTML = "";
```

A vous de jouer ! Faites la même chose pour toutes les autres sections, excepté « Projets et réalisations ». Il vous suffit pour cela de créer les fichiers JSON correspondant et de passer leur nom en paramètre de la fonction `afficher()`.

Pour finir cet exercice, il reste à gérer la section « Projets et réalisations » pour qu'elle apparaisse/disparaisse elle aussi au survol de la souris. Commencez par créer un fichier JSON adapté. La fonction `cacher()` peut être utilisée telle quelle. En revanche, les fonctions `demanderContenu()` et `afficherContenu()` ne sont pas compatibles car elles créent des `li` et non des `h3` et des `p`. Définissez de nouvelles fonctions pour résoudre ce problème, par exemple `demanderContenuProj()` et `afficherContenuProj()`. Après les avoir créées, n'oubliez pas de tester que l'affichage de la section se fait bien dynamiquement au survol de la souris.

Exercice 3 : jQuery

jQuery est une librairie JavaScript. Son utilisation présente 2 avantages :

- Le code est plus rapide à écrire et plus simple à lire que du JavaScript « pur ».
- Le code est portable sur tous les navigateurs. En effet, certaines fonctions JavaScript sont différentes selon les navigateurs. jQuery a été développée de façon à harmoniser cela.

Commencez par télécharger la dernière version de jQuery (https://jquery.com/download/), mettez-là dans votre répertoire de développement et chargez-la dans votre page HTML.

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

Voici un exemple montrant comment modifier le contenu d'un paragraphe dont l'identifiant est `monP` :

Sans utiliser jQuery :

```
var monP = document.getElementById ("monP");
monP.innerHTML = ("Un petit texte.");
```

En utilisant jQuery :

```
var monP = $("#monP");
monP.html("Un petit texte.");
```

Les deux codes ci-dessus sont équivalents, pourtant, le second est plus compact et plus facile à lire. Il en est de même pour un code permettant de changer la couleur du texte d'un paragraphe :

Sans utiliser jQuery :

```
var monP = document.getElementById("monP");
monP.style.color ="rgb(255, 0, 0)";
```

En utilisant jQuery :

```
var monP = $("#monP");
monP.css("color"," rgb(255, 0, 0)");
```

Modifiez toutes les fonctions de votre fichier monscript.js de façon à utiliser jQuery lors des modifications de style et de contenu. Précision : si l'on veut connaitre la valeur d'une propriété CSS en jQuery, par exemple la couleur du texte d'un paragraphe `monP`, il suffit d'écrire `monP.css("color")`. Vous en aurez besoin pour changer la couleur du titre lorsque l'on clique dessus (cf. début de l'exercice 3 de la séance précédente). Testez.

Pour finir, vous allez voir comment charger un fichier à l'aide de jQuery. C'est beaucoup plus simple qu'en JavaScript « pur », plus besoin de créer une fonction à part appelée lors de la réception du fichier :

```
$.ajax({
  url: "exemple.json",  Nom du fichier demandé.
  dataType: "json",  Type du fichier demandé.
  success: function(res){
    Le code à réaliser lors de la réception du fichier se place ici.
    Le contenu du fichier se trouve dans la variable res.
  }
});
```

Supprimez vos fonctions `afficherPresentation()`, `afficherContenu()` et `afficherContenuProj()` et modifiez le code de vos fonctions `demanderPresentation()`, `demanderContenu()` et `demanderContenuProj()` de façon à utiliser jQuery. Testez : normalement, tout doit fonctionner comme avant.