Notes relatives au sous-projet 4

D.Moreaux

15 octobre 2024

1 Appels AJAX

Pour une requête AJAX qui retourne des données en JSON, on utilisera la structure suivante :

```
function MakeRequest() {
    var xhr = new XMLHttpRequest();
    var param =      "xxx=" + encodeURIComponent(123);
    param = param + "&yyy=" + encodeURIComponent("abc");
    [...]
    xhr.onreadystatechange = function() {
        if (xhr.readyState == 4) {
            if (xhr.status == 200) {
                ProcessRequest(xhr.responseText);
            }
        }
    };

    xhr.open("POST","rpc/phpfunct.php");
    xhr.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
    xhr.send(param);
}

function ProcessRequest(str) {
    var result = JSON.parse(str);
    [...]
}
```

Pour une requête GET, on mettra GET à la place de POST, on supprimera le setRequestHeader, on remplacera `param` par `null` dans le send et on ajoutera les paramètres à la fin de l'URL en les séparant par un "`?`"

Côté PHP, on utilisera un tableau associatif et `json_encode()`

```
$a= array("aaa" => '123", "bbb" => 'xyz');
echo json_encode($a);
```

Il est possible d'encoder un tableau ou un objet Javascript en JSON avant de l'envoyer au PHP.

Côté Javascript, on utilisera `JSON.stringify()` pour convertir le tableau ou l'objet en une chaîne JSON qui sera envoyée dans un paramètre POST et côté PHP, on utilisera `json_decode(str,true)` pour la décoder en un tableau [1]

2 Javascript : arbre DOM

Si il est possible de modifier le contenu d'un élément HTML à l'aide de son `.innerHTML`, ce n'est pas recommandé en général car toute balise HTML présente dans le texte sera interprétée, au risque d'injecter un script ou d'avoir certains caractères interprétés (`<>&"'`). De plus, cette méthode ne permet pas d'associer une fonction à un événement sur un élément qui serait inclus dans la chaîne HTML en question.

A la place, il est conseillé de vider l'élément et de créer la structure comme suit :

```
el=document.getElementById("maDiv");
el.innerHTML=""; // On commence par vider
tbl=document.createElement("TABLE");
for(i=0,i<a.length;i++) { // lignes de la table
    tr=document.createElement("TR");
    for(j=0;j<a[0].length;j++) { // colonnes de la table
        td=document.createElement("TD");
        tn=document.createTextNode(a[i][j]); // Un TextNode avec le contenu
        // td.onclick= ...
        td.appendChild(tn); // Le TextNode dans le TD
        tr.appendChild(td); // Le TD dans le TR
    }
    tbl.appendChild(tr) // Le TR dans la TABLE
}
el.appendChild(tbl); // la table dans l'élément (DIV)
```

1. sans le true, un objet Javascript sera décodé en un objet PHP

On peut évidemment créer autre chose que des tables par cette méthode. A noter qu'il n'est pas nécessaire de mettre des ID sur les cases de la table (ou tout autre élément créé ainsi) pour y accéder vu qu'on dispose déjà de l'élément dans une variable.

3 Javascript : les Closures

Quand une fonction est créée dans le javascript, elle est attachée à l'environnement de variables dans laquelle elle se trouve. Il ne s'agit pas d'une copie de l'environnement mais bien d'un lien vers l'environnement. Si ce dernier continue d'évoluer, les variables peuvent changer et ce sera la dernière valeur de ces variables qui seront utilisée quand la fonction sera exécutée.

```
function test() {
    var div=document.createElement("div");
    document.getElementByTagName("body")[0].appendChild(div);
    var txt="Hello";

    div.onclick=function(ev) {
        div.innerHTML=txt;
    };

    txt="GoodBye";
}
```

Lorsque l'on cliquera sur la division ajoutée en fin de body, ce sera le texte "GoodBye" qui sera inséré et non le texte "Hello".

Pour éviter cela, il faudra créer un nouvel environnement de variables. Cela peut par exemple fait en utilisant une fonction "helper" extérieure ou autre

```
function test() {
    var div=document.createElement("div");
    document.getElementByTagName("body")[0].appendChild(div);
    var txt="Hello";

    div.onclick=clickhelper(txt);

    txt="GoodBye";
}
```

```
function clickhelper(txt) {
    return function(ev) {
        div.innerHTML=txt;
    };
}
```

L'environnement utilisé en closure sera celui de la fonction `clickhelper()` et plus celui de la fonction `test()`.

On peut aussi utiliser une fonction anonyme appelée immédiatement

```
function test() {
    var div=document.createElement("div");
    document.getElementByTagName("body")[0].appendChild(div);
    var txt="Hello";

    div.onclick=(function(txt) {
                    return function(ev) {
                        div.innerHTML=txt;
                    };
                })(txt);
    txt="GoodBye";
}
```

Une tendance actuelle sera d'utiliser `let` au lieu de `var` qui crée un environnement limité au bloc en cours mais cela n'est pas applicable tout le temps (par exemple, cela ne pourrait être utilisé dans l'exemple ci-dessus qui ne crée par de bloc de code intermédiaire).