

jQuery - AJAX

Concepts et Technologies XML

Année 2011-12

JavaScript - L'exemple Aalm

```
<script type="text/javascript">
function initialisation(){
    lesSections =
        document.getElementsByClassName("section");
    for(i = 0; i < lesSections.length; i++){
        var uneSection = lesSections.item(i);
        uneSection.
            setAttribute("onMouseOver", "onMouseOver(this)");
        uneSection.
            setAttribute("onMouseOut", "onMouseOut(this)");
    }
}
```

JavaScript - L'exemple Aalm (2)

```
function onMouseOver(elt){
    elt.style.backgroundColor = "red";
}
function onMouseOut(elt){
    elt.style.backgroundColor = "white";
}
</script>
</head>
<body onload="initialisation()">
    <!-- plus de javascript !! -->
</body>
</html>
```

JavaScript - L'exemple Aalm (3) Sur l'écran



Le DOM de la page

```
<div class="principale">
  <h1>Les Cours </h1>
  > <div class="section" onmouseover="onMouseOver(this)" onmouseout="onMouseOut(this)" style="background-color: white;">
  > <div class="section" onmouseover="onMouseOver(this)" onmouseout="onMouseOut(this)" style="background-color: red;">
  > <div class="section" onmouseover="onMouseOver(this)" onmouseout="onMouseOut(this)" style="background-color: white;">
  > <div class="section" onmouseover="onMouseOver(this)" onmouseout="onMouseOut(this)" style="background-color: white;">
```

JavaScript - Séparer Structure et Comportement

- Bonne conception :
 - séparation de la structure et de la mise en forme
 - séparation de la structure et du comportement
- robustesse, maintenabilité
- bonne gestion de l'accessibilité!

De nombreuses bibliothèques : *jQuery*, *Prototype*, *script.aculo.us*, *MooTools* ...

- permettent de coder facilement en Javascript
- de mettre en oeuvre naturellement la séparation code/structure
- étendent les fonctionnalités, allègent la syntaxe, ...

La bibliothèque jQuery

- bibliothèque très utilisée
- très légère
- sous licences GPL et MIT
- permet de faciliter
 - l'accès aux éléments du document
 - la gestion des événements
 - l'animation des pages
 - les interactions Ajax

Utiliser la bibliothèque jQuery

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <script src="chemin/vers/jquery.js"></script>
  </head>
  <body>
    ...
  </body>
</html>
```

La classe jQuery

jQuery. ou son raccourci *\$*. permet de manipuler les objets, les tableaux, les chaînes de caractères :

- méthodes booléennes : *\$. isArray(uneVar)*, *\$. isEmptyObject(uneVar)*...
- *\$. extend(obj1, obj2)* : ajoute toutes les propriétés de *obj2* à *obj1*
- *obj3 = \$. extend({}, obj1, obj2)* : retourne un nouvel objet fusion de *obj1* et *obj2*
- *\$. each(chaineDuObjet, uneFonction(cle, valeur)...)* applique *uneFonction* sur chaque propriété ou méthode d'un objet, ou sur chaque caractère d'une chaîne
- ...

Les objets jQuery

Pour Traverser le DOM avec jQuery :

- utilise la syntaxe des CSS (et de XPath, on en reparlera plus tard) pour accéder aux éléments d'un document
- passe par l'instanciation d'un objet jQuery *jQuery(selecteur)*. ou son raccourci *\$(selecteur)*.

```
$("#document")
$("#orderedElt")
$(".urgent")
$(".section.exercice p:first-child")
```

Modifier un document

Un document jQuery possède les méthodes

- *load()* : pendant le chargement du document
- *ready()* : pour spécifier la fonction à exécuter lorsque le document est complètement chargé

```
<!doctype html>
<html>
  <head>
    <script src="chemin/vers/jquery.js"></script>
    <script>
      $(document).ready(function(){
        // le code à écrire
      });
    </script>
  </head>
  <body> ... </body>
</html>
```

Sur les éléments ...

Les éléments auxquels on accède par *\$(selecteur)*. possède des méthodes pour accéder aux attributs, à la css, pour réagir aux événements...

- attributs :
 - `$("#image").attr("src")` retourne la valeur de l'attribut `src` de l'élément d'identifiant `image`
 - `$("#image").attr("src", "Chat.gif")` affecte l'attribut `src` à la valeur `Chat.gif`
- css
 - `$("#li").css({border : "solid 0.1em red", color : "purple"})` : affecte des propriétés de mise en forme
 - `$("#li").css("border", "solid 0.1em red").css("color", "purple")` : idem
 - `$("#li").css("border")` retourne la chaîne `"solid 0.1em red"`
 - `$("#section").addClass("rubrique")` ajoute la valeur `rubrique` à l'attribut `class`
 - `$("#section").removeClass("rubrique")` supprime la valeur `rubrique` de l'attribut `class`

Accéder au contenu, ajouter du contenu

- `$("#p:first-child").html()` retourne une chaîne de caractères avec le contenu du premier paragraphe. Par exemple `"ici on parle de <code>jQuery</code>"`
- `$("#p:first-child").html("Voici un premier paragraphe")`
- ou bien avec une fonction, lorsque plusieurs éléments sont sélectionnés :


```
$("#p").html(function(indice){
    return "ceci est le <em>"+indice+"ème</em> paragraphe";
});
```
- même chose avec la méthode `text()`
- on retrouve les méthodes habituelles de manipulation du DOM (`appendTo()`, `insertBefore()`) avec quelques adaptations
- plus quelques méthodes spécifiques (`prependTo()`, `wrap()`, `wrapAll()`...

Gérer des événements

On peut gérer les événements sur des objets jQuery et leur associer des fonctions :

```
$("#h1").click(function(event){ ... })
```

ou bien

```
$("#h1").bind("mouseout",function(event){ ... })
$("#h1").bind("mouseover",function(event){ ... })
```

Un exemple complet (1)

```
<!DOCTYPE html>
<html>
  <head>
    <script src="../../jquery-1.6.3.min.js"></script>
```

```

    <script type="text/javascript"> ... </script>
</head>
<body>
  <p> Choisissez parmi les animaux proposés :</p>
  <div style="float: left; width: 30%;">
    <ul><li>Chat</li>
      <li>Chien</li>
      <li>Perroquet</li>
      <li>Cochon</li>
    </ul>
  </div>
  <div></div>
</body>
</html>

```

Un exemple complet (2)

```

<script type="text/javascript">
  $(document).ready(function()
    $("#image").css("visibility","hidden");
    $("li").mouseover(function(event)
      img = $("#image");
      nomFich = "./"+$(this).text()+".gif";
      img.attr("src",nomFich);
      img.css("visibility","visible");
      $(this).css("color","green");
    );
    $("li").mouseout(function(event)
      $(this).css("color","blue");
      $("#image").css("visibility","hidden");
    );
    $("li").css("color","purple");
  );
</script>

```

Les liste d'éléments

Les sélecteurs peuvent retourner une liste d'éléments :

`$("#p")` retourne ainsi la liste des éléments p d'un document.

jQuery propose des méthodes pour manipuler ces listes d'éléments :

- `$("#p").each(function(indice)$ (this).appendTo(" p numéro "+indice);)`
- `alert($("#p').length");`
- `$("#p")[0]`
- et des méthodes pour ajouter des éléments à la liste, supprimer des éléments, les filtrer, leur appliquer une fonction ...

AJAX

AJAX : Asynchronous Java And Xml

- client riche : une meilleure répartition de la charge de travail entre le serveur, le réseau et le client
- recharger uniquement les parties nécessaires d'une page en fonction des événements

AJAX

- ensemble de techniques qui utilisent
 - HTML, CSS
 - DOM
 - HTTP
 - JavaScript
 - l'objet XMLHttpRequest
 - et éventuellement un autre langage sur le serveur

L'objet XMLHttpRequest

Pour effectuer une requête HTTP

```
interface XMLHttpRequest {
// pour établir la connexion
void open(méthode DOMString, uri DOMString,
          boolean async);

void setRequestHeader(en-tête DOMString,
                     valeur DOMString) raises(DOMException);
// pour envoyer la requête (params. uniqu. pour POST)
void send(donnée DOMString);
void send(donnée Document);
void abort();
};
```

L'objet XMLHttpRequest

Pour traiter la réponse du serveur

```
interface XMLHttpRequest {
DOMString getAllResponseHeaders();
DOMString getResponseHeader(en-tête DOMString);
attribut DOMString responseText;
attribut Document responseXML;
// 200 : OK, 404 : File Not Found, ...
attribut unsigned short status;
attribut DOMString      statusText;
};
```

L'objet XMLHttpRequest

Les différents états de l'objet requête :

```
interface XMLHttpRequest {
// change de valeur, de 0 à 4
readonly attribute unsigned short readyState;
// on lui associe une fonction : activée à chaque
// changement de valeur pour readyState
attribut EventListener onreadystatechange;

// les états possibles
const unsigned short UNSENT = 0;
const unsigned short OPEN = 1;
const unsigned short SENT = 2;
const unsigned short LOADING = 3;
const unsigned short DONE = 4;
};
```

Schéma général

Lorsqu'un événement sur la page justifie d'en rafraîchir une partie, on appelle une fonction qui :

- crée un objet XMLHttpRequest
- associe à sa propriété `onreadystatechange` une fonction qui traitera le retour de la requête
- crée une connexion HTTP
- envoie la requête

Un premier exemple

```
<script language="JavaScript">
function submitForm(){
    var req = null;
    try {
        req = new XMLHttpRequest();
    }
    catch (e) { // spécial I.E.
        req = new ActiveXObject(Microsoft.XMLHTTP);
    }
    req.onreadystatechange = function() {
        // beaucoup de travail
    };
    req.open("GET", "data.xml", true);
    req.send(null);
}
</script>
```

La fonction qui traite le retour

Schéma général :

- si le statut est différent de 4 ([DONE](#)) :
 - ne rien faire! ou informer l'internaute
- sinon, il faut étudier le retour de la requête
 - si son statut est 200 : récupérer le résultat ([responseText](#) ou [responseXml](#))
 - sinon, traiter l'échec.

Exemple de fonction de retour

```
req.onreadystatechange = function(){
    document.getElementById("info").value="Wait server...";
    if(req.readyState == 4) {
        if(req.status == 200) {
            document.getElementById("retour").value=
                req.responseText;
        }
        else {
            document.getElementById("retour").value=
                "Error: returned status code " + req.status
                + " " + req.statusText;
        }
    }
};
```

Et la page HTML

```
<body>
    <form name="ajax" method="POST" action="">
        <input type="button"
            value="Submit"
            onClick="submitForm()" />
        <input type="text"
            id="retour"
            size="32" value="" />
    </form>
</body>
```

Et les informations demandées ?

Le fichier [data.xml](#) peut simplement contenir :

Un message de bienvenue!

mais il peut aussi contenir un document XML :

```
<h3>Un message de bienvenue!</h3>
```

Dans ce cas on récupèrera le résultat de la requête avec :

```
attribut Document responseXML;
```

Traiter des données XML

Avec les méthodes de DOM! :

```
// on récupère le document XML
```

```
result = req.responseXML;
```

```
// on accède à sa racine
```

```
racine = result.documentElement;
```

```
// on fait une copie du noeud
```

```
// dans le document courant
```

```
nd = document.importNode(racine, true);
```

```
// on l'insère dans le document courant
```

```
document.body.appendChild(nd);
```

Traiter des données XML

Mais `importNode()` n'est pas supporté par tous les navigateurs...

Une autre proposition :

```
// on récupère le document XML
```

```
// sous forme texte
```

```
result = req.responseText;
```

```
// si on veut ajouter à la fin...
```

```
// on créé un élément
```

```
boite = document.createElement("div");
```

```
// on insère la réponse dans cette boite
```

```

boite.innerHTML = result;

// et on ajoute la boite dans le document courant

document.body.appendChild(boite);

```

Ajax et jQuery (1)

C'est la méthode `$.ajax()` qui permet de

- créer un objet `XmlHttpRequest`
- créer la connexion
- envoyer la requête
- récupérer le résultat

Le programmeur jQuery doit paramétrer ce qu'il veut :

- l'URL choisie
- les données envoyées
- le type de la requête
- le type des données attendu
- les traitements à effectuer
- ...

Ajax et jQuery (2)

Le paramétrage de la requête Ajax se fait par :

```

$.ajax({
  option1 : valeur_ou_fonction,
  option2 : valeur_ou_fonction,
  ...
});

```

avec les options possibles :

- `url` : URL du programme qui va traiter la requête
- `data` : les données à transmettre, sous la forme "nom1=val1&nom2=val2" ou d'un objet jQuery
- `type` : GET ou POST
- `dataType` : `text`, `xml`, `html`, ... type attendu des données
- `context` : l'élément qui sera concerné par le résultat (objet jQuery). C'est le `$(this)` dans les fonctions de traitement
- `timeout` : délai en millisecondes maximal d'attente

Ajax et jQuery (3)

et pour les traitements, options possibles :

- `success` : fonction de la forme `function(reponse, statut, xhr)`
- `error` : fonction de la forme `function(xhr, statut)`
- `complete` : fonction de la forme `function(xhr, statut)`

Ces fonctions indiquent le traitement à effectuer avec la réponse. Elles peuvent modifier le DOM du document à partir de `$(this)` qui est le `context` défini.

On peut toujours obtenir les réponses à la requête à partir de l'objet `xhr` : `xhr.responseText`, `xhr.responseXML`

Exemple

```
<script>
function submitForm(){
  $.ajax({
    url : "data.xml",
    type : "POST",
    datatype : "text",
    context : $("#retour"),
    success : function(rep,statut,xhr){
      $(this).text(rep);},
    error : function(xhr,statut){
      $(this).text("Erreur: " +
        statut + xhr.statusText);}});}
</script>
<form method="POST" action="">
  <input type="button" value="Submit" onClick="submitForm()" />
  <input type="text" id="retour" size="32" value="" />
</form>
```