Programmation WEB

Ajax

Programmation licence

IUT de Fontainebleau

1er mars 2015

Sommaire

Sommaire

Introduction

- AJAX se base sur l'objet JavaScript XMLHttpRequest qui permet de requêter dynamiquement une url via le protocole HTTP(S).
- L'un des avantages est de pouvoir échanger des données entre la page et un serveur sans avoir à recharger entièrement la page.

- Intéraction "traditionnelle" avec l'utilisateur

  click > attente > rafraîchissement de la page

- Avec Ajax : rafraîchissement partielle de la page

- Seuls les éléments d'interface qui contiennent de nouvelles informations sont rafraîchis de manière asynchrone.
- Le reste de l'interface reste visible (pas de perte de contexte opérationnel)

- IHM orientée "données" par opposition à une IHM orientée "page".

L'IHM est gérée par le client, tandis que les données sont calculées et fournit par le serveur.

- Le modèle asynchrone remplace le modèle synchrone requête/réponse.

⇝ L'application reste utilisable pendant que le client demande des infos au serveur en arrière plan.

⇝ Séparation de la présentation et de l'accès aux données.

- La programmation côté serveur reste la même :

⇝ cgi qui recoivent des requêtes http : servlet, jsp, php, python, ruby, asp, etc ...

⇝ et qui génére une réponse http avec un contenu pouvant être de type xml, json, javascript, texte brute, html, etc ...

Quelques cas d'utilisations. Il y a en beaucoup d'autres.

Google map

- L'utilisateur draggue une partie de la carte.
- Le nouveau morceau de carte est chargé de manière asynchrone avec Ajax.
- Le reste de l'interface utilisateur est inchangée.

Auto-complétion

Emails, noms, code postaux, etc peuvent être auto-complétés quand l'utilisateur fait sa saisie.

Eléments d'interfaces avancées

Menus, arbres, barre de progression, date picker peuvent être utilisés sans rafraîchir toute l'interface.

Un exemple qui utilise des données d'un document XML récupéré par le réseau :

```
function processData(data) {
  // taking care of data
}

function handler() {
  if(this.readyState == this.DONE) {
    if(this.status == 200 &&
       this.responseXML != null &&
       this.responseXML.getElementById('test').textContent) {
      // success!
      processData(this.responseXML.getElementById('test').textContent);
      return;
    }
    // something went wrong
    processData(null);
  }
}

var client = new XMLHttpRequest();
client.onreadystatechange = handler;
client.open("GET", "unicorn.xml");
client.send();
```

Log sur un serveur

```
function log(message) {
  var client = new XMLHttpRequest();
  client.open("POST", "/log");
  client.setRequestHeader("Content-Type",
              "text/plain;charset=UTF-8");
  client.send(message);
}
```

Récupérer le statut d'un document sur le serveur

```
function fetchStatus(address) {
  var client = new XMLHttpRequest();
  client.onreadystatechange = function() {
    if(this.readyState == this.DONE)
      returnStatus(this.status);
  }
  client.open("HEAD", address);
  client.send();
}
```

Sommaire

Spécification W3C

```
interface XMLHttpRequest : XMLHttpRequestEventTarget {
  // event handler
  attribute EventHandler onreadystatechange;

  // states
  const unsigned short UNSENT = 0;
  const unsigned short OPENED = 1;
  const unsigned short HEADERS_RECEIVED = 2;
  const unsigned short LOADING = 3;
  const unsigned short DONE = 4;
  readonly attribute unsigned short readyState;

  // request
  void open(ByteString method, [EnsureUTF16] DOMString url);
  void open(ByteString method, [EnsureUTF16] DOMString url, boolean async,
      optional [EnsureUTF16] DOMString? username = null,
      optional [EnsureUTF16] DOMString? password = null);
  void setRequestHeader(ByteString header, ByteString value);
           attribute unsigned long timeout;
           attribute boolean withCredentials;
  readonly attribute XMLHttpRequestUpload upload;
  void send(optional (ArrayBufferView or Blob or Document
        or [EnsureUTF16] DOMString or FormData)? data = null);
  void abort();
```

Spécification W3C

```
// response
  readonly attribute unsigned short status;
  readonly attribute ByteString statusText;
  ByteString? getResponseHeader(ByteString header);
  ByteString getAllResponseHeaders();
  void overrideMimeType(DOMString mime);
  attribute XMLHttpRequestResponseType responseType;
  readonly attribute any response;
  readonly attribute DOMString responseText;
  readonly attribute Document? responseXML;
};
```

Spécification W3C

Sur les navigateurs qui implantent les recommandations récentes, XMLHttpRequest hérite de XMLHttpRequestEventTarget et rajoutent les handlers

```
// event handlers
  attribute EventHandler onloadstart;
  attribute EventHandler onprogress;
  attribute EventHandler onabort;
  attribute EventHandler onerror;
  attribute EventHandler onload;
  attribute EventHandler ontimeout;
  attribute EventHandler onloadend;
```

Création

Sur tous les navigateurs "modernes" :

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

Navigateurs anciens

```
function getXMLHttpRequest(){
  var xmlhttp;
  try{
    xmlhttp = new XMLHttpRequest();
  }
  catch (e1){
    try {
      xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
    }
    catch (e2) {
      try {
        xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
      }
      catch (e3) {
        xmlhttp=null;
      }
    }
  }
```

L'objet XMLHttpRequest

Attributs

onreadystatechange	fonction réflexe appelée lors de l'événement onreadystatechange , qui se produit, en mode asynchrone, lors d'un changement d'états de la requête.
readyState	état courant de la requête
responseText	Réponse sous forme de texte
responseXML	Réponse sous forme XML
status	code du statut de retour HTTP de la réponse
statusText	texte du statut de retour HTTP de la réponse
responseType	Type de la réponse : "", "arraybuffer", "blob", "document", "json", "text"

La classe XMLHttpRequest

- `abort`

annule la requête et réinialise l'objet

- `getAllResponseHeaders`

Retourne tous les entêtes HTTP de la réponse sous forme d'une chaîne de caractères. Valable uniquement pour la valeur 3 et 4 de `readyState`

- `getResponseHeader(nom)`

renvoie la valeur de l'entête dont le nom est spécifié en paramètre. (idem ci-dessus)

- `setRequestHeader(header,value)`

positionne un entête HTTP pour la requête.

- `open(methode,url,async,[user],[password])`

initialise l'objet pour une requête.

  - méthode http : GET, POST, etc.
  - url : url de la requête.
  - async : Le type de fonctionnement détermine la manière dont est traitée la réponse à la requête (synchrone ou asynchrone par défaut).
  - Enfin, des informations (user et password) de sécurité peuvent être utlisées.

- `send(string/document)`

envoie une requête à l'adresse spécifiée avec la méthode HTTP souhaitée. Si le mode de réception est synchrone, la méthode est bloquante jusqu'à ce moment. On peut ajouter une chaîne ou un document xml en paramètre en cas de post.

- Synchrone, c'est à dire séquentiellement.

```
var request=getXMLHttpRequest();
request.open("get","data.js",false); // false => synchrone
requete.send(null);
vat test=request.reponseText;
```

- asynchrone : réception par une fonction réflexe qui est appelée à chaque changement d'états :

Mode asynchrone

Valeur	Etat	description
0	UNSENT	état initial
1	OPENED	méthode open exécutée avec succés
2	HEADERS_RECEIVED	headers http reçus, aucune donnée reçue
3	LOADING	en cours de reception
4	DONE	Le traitement requête est terminé

Fonctionnement asynchrone

```
var request=getXMLHttpRequest();
function changement(){
  if (request.readyState==4 && request.status == 200){
    var texte=request.reponseText;
    ...
  }
}
request.onreadystatechange=changement;
requete.open("get","data.js",true);
requete.send(null);
```

La fonction reflexe onreadystatechange est appelée 5 fois.

Envoi

Il existe deux façons de spécifier des données lors de l'envoie :

- Dans l'url de la requête avec la méthode get :

```
requete.open("get","requete.php?a="+vala+"&b="+valb,true);
requete.send(null);
```

- comme paramètre de la méthode send, avec la méthode post :

```
requete.open("POST", "script.php", true);
var params = "parametres brutes";
requete.send(params);
```

On peut préciser, dans l'entête http de la requête, le type des données envoyées à l'aide la méthode setRequestHeader

- Données url-encodées (données d'un formulaire par exemple) :

```
var url = "get_data.php";
var params = "lorem=ipsum&name=binny";
http.open("POST", url, true);
http.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
http.setRequestHeader("Content-length", params.length);
http.setRequestHeader("Connection", "close");
http.onreadystatechange = function() {
  if(http.readyState == 4 && http.status == 200)
    alert(http.responseText);
}
http.send(params);
```

- Données au format xml

```
<form name="create" id="create"
  action="javascript:createObject(document.create)">
  <input type="text" id="field1" name="field1" size="20"/>
  <input type="text" id="field1" name="field2" size="20"/>
  <input type="text" id="field3" name="field3" size="20"/>
  <input type="submit" id="submit" name="submit" value="Enregistrer"/>
</form>
```

```
function createObject(data)
{


  var request="<createObject>";
  request=request+"<field1>"+data.field1.value+"</field1>";
  request=request+"<field2>"+data.field2.value+"</field2>";
  request=request+"<field3>"+data.field3.value+"</field3>";
  request=request+"</createObject>";

  var _xmlHttp=getXMLHTTP();
  if(_xmlHttp){
    //appel a l'url distante
    _xmlHttp.open("POST",_createService,true);
    _xmlHttp.onreadystatechange=function() {
      if(this.readyState==4&&this.responseXML) {
        var xmldoc=this.responseXML;
        // Do what you need !
      }
    };
    // envoi de la requete
    _xmlHttp.setRequestHeader('Content-Type','text/xml');
    _xmlHttp.send(request);
  }
```

Sommaire

XML

Outre le format texte (pas très pratique à parser..), AJAX offre nativement un support pour XML :

- à l'envoi avec la méthode send :

```
var parametres=document.createElement("parametres");
var item=document.createElement("parametre");
item.setAttribute("nom","le_nom");
var valeur=document.createTextNode("la_valeur");
item.appendChild(valeur);
parametres.appendChild(item);
...
requete.send(parametres);
```

qui envoie la structure

```
<parametres>
  <item nom="le_nom">la_valeur</item>
</parametres>
```

- à la réception avec l'attribut responseXML

```
_xmlHttp.onreadystatechange=function() {

  if(this.readyState==4&&this.responseXML) {

    var xmldoc=this.responseXML;
    var donnees=xmldoc.childNodes[0];
    for(i=0;i<donnees.childNodes.length;i++)
    {
      ....
      ....
    }
  }
```

Il s'agit en fait de texte, représentant du code html déstiné à rafraîchir la partie d'une page :

```
<html>
  <head>
    <script>
      ....
      requete.onreadystatechange=function (){
        if (this.readyState==4){
          var html=this.responseText;
          document.getElementById("reponse").innerHTML=html;
        }
      }
    </script>
  </head>
  <body>
    <div id="reponse"></div>
  </body>
</html>
```

JSON (Javascript Object Notation) permet de déclarer une donnée sous forme de collection d'éléments. Il utilise deux structures de données :

- l'objet, sous forme de (clé valeur) :

```
{
  cle1:valeur1,
  cle2:function(){},
  cle3:valeur3
}
```

- le tableau simple : liste de valeurs

```
[
valeur,
function(){},
valeur
]
```

On peut les imbriquer :

```
[
{name:"Gary", age:"24"},
{name:"Shane", age:"29"},
{name:"Kay", age:"54"},
{name:"Sarah", age:"30"},
{name:"Jerry", age:"56"}
]
{"menu": {
  "id": "file",
    "value": "File",
    "popup": {
      "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
      ]
    }
     }
```

La méthode parse de l'objet JSON permet de parser une chaîne de caractères représentant un objet.

```
var text = '{ "employees" : [' +
  '{ "firstName":"John" , "lastName":"Doe" },' +
  '{ "firstName":"Anna" , "lastName":"Smith" },' +
  '{ "firstName":"Peter" , "lastName":"Jones" } ]}';

var obj = JSON.parse(text);
```

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

<script>
  document.getElementById("demo").innerHTML =
  obj.employees[1].firstName + " " + obj.employees[1].lastName;
</script>
```

Sommaire

Historiquement, pour des problèmes de sécurité, une requête ajax ne pouvait se faire que sur le même domaine.
Le w3c a recommendé le nouveau mécanisme de Cross-Origin Resource Sharing qui fournit un moyen aux serveurs web de contrôler les accès en mode cross-site et aussi d'effectuer des transferts de données sécurisés en ce mode.

Le standard de partage de ressources d'origines croisées fonctionne grâce à l'ajout d'entêtes HTTP qui permettent aux serveurs de décrire l'ensemble des origines permises.

Un exemple avec php et l'entête Access-Control-Allow-Origin

```php
<?php
header("Access-Control-Allow-Origin: *");
header("content-type: application/json");
mysql_connect("localhost","root","password");
mysql_select_db("communes");
$code=$_GET['code'];
$res=mysql_query("SELECT Commune,Departement
  FROM ville
  WHERE Codepos='$code';");
$t=[];
while($row=mysql_fetch_object($res))
{
  $t[]=$row;
}
echo json_encode($t);
?>
```