zenika
ARCHITECTURE INFORMATIQUE

HTML5

XMLHttpRequest Level 2

Plan

- Rappels sur l'API standard
- Les nouveautés du Level 2
- Meilleure gestion des données
- Transfert de fichier
- Monitoring du transfert
- Gestion du timeout
- Requête sur des domaines différents
- Sécurité avec les credentials

L'API XMLHttpRequest

- Principe :
  - Envoyer des requêtes au serveur Web et charger la réponse dans les scripts
  - On modifie le DOM du document courant
  - Pas de rechargement complet de la page
  - Un rôle important dans le développement Ajax

- Cas d'utilisation standard
  - Récupération de données additionnelles
  - Envoi de formulaire

L'API XMLHttpRequest
Rappels

- Objet **`window.XMLHttpRequest`**
  - Envoi d'une requête GET

```
var xhr = new XMLHttpRequest();
xhr.open('GET','/page.php');
xhr.send(null);
```

  - Envoi d'une requête POST

```
var xhr = new XMLHttpRequest();
xhr.open('POST','/page.php');
xhr.setRequestHeader("Content-Type", "application/x-www-form-
urlencoded");
xhr.send(data);
```

  - État de la requête et réponse

```
xhr.onreadystatechange = function() {
  if (xhr.readyState==4 && (xhr.status == 200 || xhr.status == 0)) {
      alert(xhr.responseText); // ou xhr.responseXML
  }
};
```

L'API XMLHttpRequest
Limitations

- Envoi de données uniquement en format texte, html, xml
  - Pas de possibilités d'envoi de fichier

- Restriction cross-domain : same origin policy
  - Pas de requêtes cross-origin
  - On avait recours à des solutions de contournement comme passer par un serveur proxy, modifier document.domain …

- Progression de requête
  - OnReadyStateChange ne permet pas de connaître la progression de la requête

Les nouveautés du Level 2

- Meilleure gestion des données
- Transfert de données
- Monitoring du transfert
- Requête sur différent noms de domaine
- Sécurité avec les credentials

- Spécification : http://www.w3.org/TR/XMLHttpRequest/

Meilleure gestion des données

- Dans les précédentes implémentations, les données envoyées devaient être sous forme de texte, on recourrait à l'URL-encoding

```
var msg = 'field1=foo&field2=bar';
var xhr = new XMLHttpRequest();
xhr.open('POST','/server');
xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xhr.send(msg);
```

- On ne peut pas envoyer/recevoir des données binaires sans hack
- Difficulté d'envoyer des données d'un formulaire

Meilleure gestion des données ResponseType

- On peut spécifier le format de réponse
  - xhr.responseType
  - "text", "arraybuffer", "blob", ou "document"
  - Par défaut "text"
- xhr.response devient DOMString, ArrayBuffer, Blob, ou Document

```
var txt = 'data';
var xhr = new XMLHttpRequest();
xhr.open('POST', '/server');
xhr.responseType = 'text';
xhr.onload = function(e) {
    if (this.status == 200) {
        console.log(this.response);
    }
};
xhr.send(txt);
```

Meilleure gestion des données FormData

- Envoyer des données clé / valeur avec l'objet FormData
- Donnée envoyée comme dans un formulaire HTML normal
- FormData utilise le type multipart/form-data
  - L'envoi de fichier est possible

```
var xhr = new XMLHttpRequest();
var dataToSend = new FormData(); // créer un objet FromData

xhr.open('POST','/server');

dataToSend.append('name','John Doe'); //ajouter des données (clé,val)
dataToSend.append('age','40');
//...

xhr.send(dataToSend); //envoyer
```

Meilleure gestion des données FormData

- L'entête content-type est ajouté par le navigateur
  - Plus besoin de la spécifier

```
POST /fileUpload HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:16.0)
Gecko/20100101 Firefox/16.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en,fr;q=0.8,fr-fr;q=0.5,en-us;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://localhost:8080/inscription.html
Content-Length: 595843
Content-Type: multipart/form-data;
boundary=---------------------------15329212382
Pragma: no-cache
Cache-Control: no-cache
```

Meilleure gestion des données FormData

- Si vous avez un formulaire déjà existant dans votre document

```
<form id="myform" name="myform" action="/server">
  <input type="text" name="username" value="johndoe">
  <input type="number" name="age" value="40">
  <input type="submit" onclick="return sendForm(this.form);">
</form>
```

```
function sendForm(form) {
  var formData = new FormData(form);

  formData.append('other_data', '...');  //ajout de données
                                         //avant envoi

  var xhr = new XMLHttpRequest();
  xhr.open('POST', form.action, true);
  xhr.onload = function(e) { ... };

  xhr.send(formData);

  return false; // Pour stopper l'événement submit
}
```

Transfert de fichier Download

- Récupérer un fichier du serveur
  - Blob responses
  - xhr.responseType = 'blob';

- var blobURL = window.URL.createObjectURL(blob);
  - Créé une String URL qui référence l'objet blob dans le DOM

```
blob:http://localhost/c745ef73-ece9-46da-8f66-ebes574789b1
```

- Les BlobURL sont uniques mais c'est toujours une bonne pratique de les déréférencer.

```
window.URL.revokeObjectURL(blobURL);
```

Transfert de fichier Download

- Exemple complet

```
window.URL = window.URL || window.webkitURL;

var xhr = new XMLHttpRequest();
xhr.open('GET', '/path/to/image.png');
xhr.responseType = 'blob';

xhr.onload = function(e) {
  if (this.status == 200) {
    var blob = this.response;

    var img = document.createElement('img');
    img.onload = function(e) {
      window.URL.revokeObjectURL(img.src);
    };
    img.src = window.URL.createObjectURL(blob);
    document.body.appendChild(img);
  }
};

xhr.send();
```

Transfert de fichier Download

```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'doodle.png');

xhr.responseType = 'arraybuffer';

xhr.onload = function(e) {
  if (this.status == 200) {
    var uInt8Array = new Uint8Array(this.response);
    var i = uInt8Array.length;
    var binaryString = new Array(i);
    while (i--)
    {
      binaryString[i] = String.fromCharCode(uInt8Array[i]);
    }
    var data = binaryString.join('');

    var base64 = window.btoa(data);


document.getElementById("myImage").src=""
+base64;
  }
};

xhr.send();
```

Transfert de fichier
Upload

- Upload de fichier
  - Devient facile avec FormData

```
function uploadFiles(url, files) {
  var formData = new FormData();

  for (var i = 0, file; file = files[i]; ++i) {
    formData.append(file.name, file);
  }

  var xhr = new XMLHttpRequest();
  xhr.open('POST', url, true);
  xhr.onload = function(e) { ... };

  xhr.send(formData);  // multipart/form-data
}
```

- Envoyer un blob ou un file

```
xhr.send(new Blob(['hello world'], {type: 'text/plain'}));
```

Transfert de fichier
Upload

- Envoyer des bytes (Arraybuffer)

```
function sendArrayBuffer() {
  var xhr = new XMLHttpRequest();
  xhr.open('POST', '/server', true);
  xhr.onload = function(e) { ... };

  var uInt8Array = new Uint8Array([1, 2, 3]);

  xhr.send(uInt8Array.buffer);
}
```

ProgressEvent Interface

attribute	type	Explanation
onloadstart	loadstart	When the request starts.
onprogress	progress	While loading and sending data.
onabort	abort	When the request has been aborted.
onerror	error	When the request has failed.
onload	load	When the request has successfully completed.
ontimeout	timeout	When the author specified timeout has passed before the request could complete.
onloadend	loadend	When the request has completed, regardless of whether or not it was successful.

Monitoring du transfert

- Suivre l'évolution d'un upload

```
<progress min="0" max="100" value="0">0% complete</progress>
```

- xhr.upload.onprogress

```
// Listen to the upload progress.
var progressBar = document.querySelector('progress');
xhr.upload.onprogress = function(e) {
  if (e.lengthComputable) {
    progressBar.value = (e.loaded / e.total) * 100;
    progressBar.textContent = progressBar.value;
  }
};
```

- Pour suivre un download utiliser `xhr.onprogress`

Gestion du timeout

- Si une requête met trop de temps à s'exécuter, que faire?
- On peut définir un timeout et un handler sur une requête

```
xhr.timeout = 3000;
xhr.ontimeout = onTimeOutHandler;
```

```
var onTimeOutHandler = function(event){
      // afficher un message

     // renvoyer la requête
     event.target.open('GET','data.json');

     // On peut définir un timeout plus long
     event.target.timeout = 6000;
     event.target.send();
 }
```

Cross Origin Resource Sharing (CORS)

- La précédente version du XHR limitait les requêtes aux mêmes origines : protocoles (http, https), nom de domaine et port
- XMLHttpRequest 2 supporte les requêtes cross-origin
- Le navigateur web ajoute un origin header lors d'une XHR
  - Ne peut pas être définie ou modifié par setRequestHeader()
- Sur le serveur on peut ajouter un header pour activer CORS:

```
Access-Control-Allow-Origin: http://example.com
```

- Pour accepter toutes les requêtes :

```
Access-Control-Allow-Origin: *
```

- On fait une requête XHR classique avec l'url du serveur

```
xhr.open('GET','http://other.server/path/script');
```

Cross Origin Resource Sharing (CORS)

- Exemple serveur php

```
<?php
header("Access-Control-Allow-Origin: *");
```

- Exemple serveur avec nodeJS

```
headers["Access-Control-Allow-Origin"] = "*";
```

CORS with credentials

- Par défaut CORS n'envoie pas les « user credentials »
  - Cookies
  - HTTP authentication
  - client-side SSL certificates

- Si on veut les envoyer au serveur
  - xhr.withCredentials = true;

- Nécessaire seulement pour CORS
- Access-Control-Allow-Origin header ne contient pas de *
- Access-Control-Allow-Credentials header doit être « true »

XMLHttpRequest 2 Support

- Navigateurs
  - Chrome 7.0+
  - Firefox 4.0+
  - Safari 5.0+
  - Opera 12.0+
  - IE 10
- Tester le support

```
var xhr = new XMLHttpRequest();
if (typeof xhr.withCredentials === undefined &&
    typeof xhr.responseType === undefined &&
    typeof xhr.upload){ //... (tester toutes les proprietés)
    //le navigateur ne supporte pas
} else {
    //...
}
```

TP