AJAX

Asynchronous Javascript And XML

J.-M. Pecatte

Dept MMI – IUT Castres

jean-marie.pecatte@iut-tlse3.fr

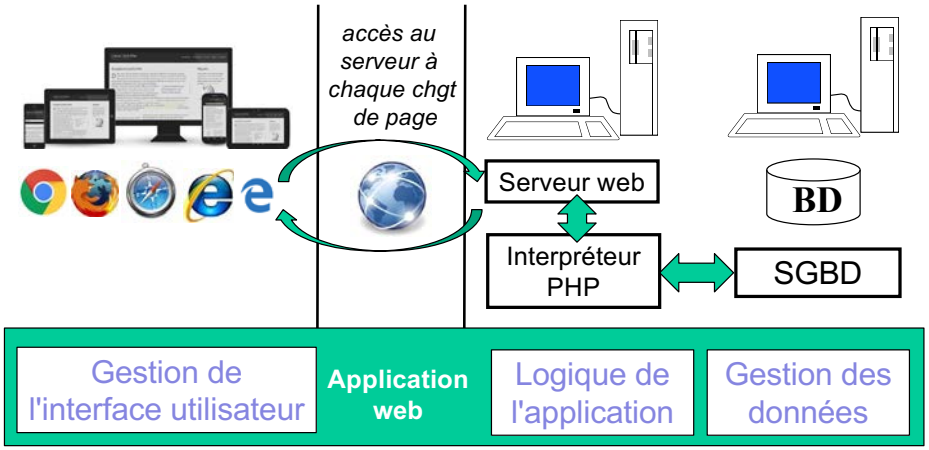
AJAX

- AJAX : **A**synchronous **J**avascript **A**nd **X**ML
- Fonction XMLHttpRequest créée par Microsoft en 1999 : récupération par un script d'un texte distant via le protocole HTTP, au format XML ou non, réalisée en mode asynchrone.
- Généralisation dans tous les navigateurs
- Normalisation par le W3C
- Avantage principal (pour une application web) : récupérer une information distante sans (re)chargement de page, et donc faire des économies en réduisant le volume des échanges avec le serveur.

AJAX - exemples

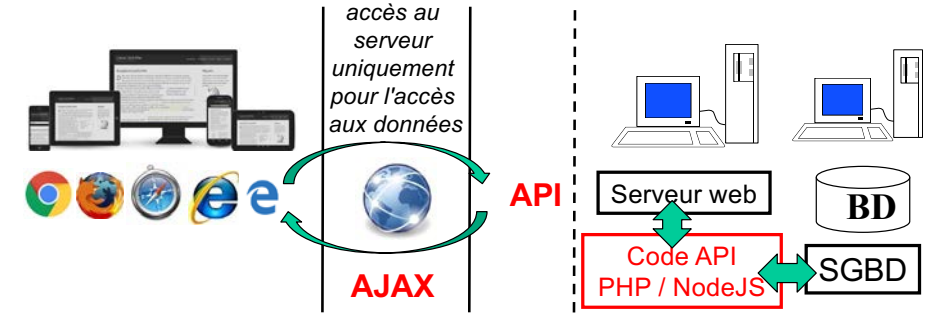
- Google maps
- Gmail
- Auto complétion (par exemple dans google.fr)
- Eléments d'interface avancée
- …

Application web classique



→ Le navigateur ne gère que l'interface avec l'utilisateur
→ La logique de l'application est codée coté serveur (en PHP) qui gère aussi les données

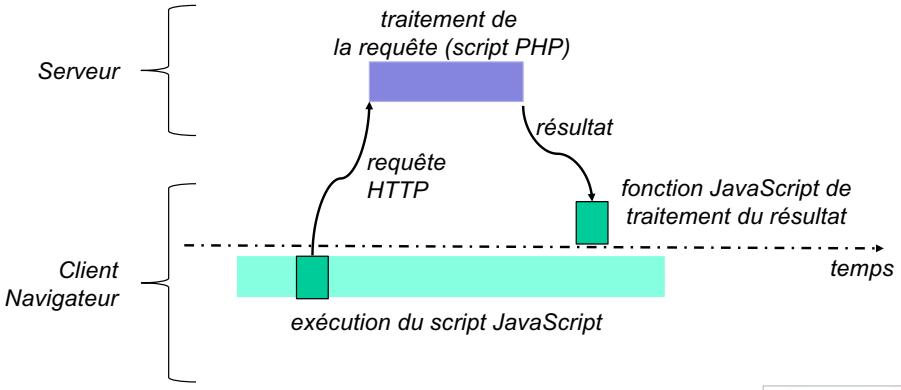
Single Page Application



→ Le navigateur gère l'interface utilisateur mais La logique de l'application (codée en JS)
→ Le serveur (en PHP ou NodeJS) gère uniquement les données, il les rend accessible à travers une API ; l'accès à l'API se fait en utilisant AJAX.

AJAX

- Exécution en mode asynchrone : les échanges avec le serveur via le protocole http se font en parallèle de l'exécution du script JavaScript.



AJAX

- Historiquement, pour des problèmes de sécurité, une requête AJAX ne pouvait se faire que sur le même domaine.
- Le w3c a recommandé 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 *entête* Access-Control-Allow-Origin

XML

eXtended Markup Language

XML

- XML est un langage de structuration de documents
- Dans le cadre d'AJAX, le serveur renvoie le résultat (les données) au format XML ou au format texte
- La méthode responseXML de l'objet XMLHttpRequest renvoie ce document XML
- XML abandonné au profit de JSON, plus simple à manipuler en javascript.

un exemple de doc XML

```
<?xml version="1.0" encoding="utf-8"?>
<!-- editeurs.xml -->
<root>
   <soft name="Adobe Dreamweaver" />
   <soft name="Microsoft Expression Web" />
   <soft name="Notepad++" />
   <soft name="gedit" />
   <soft name="Emacs" />
</root>
```


Structure d'un document XML

Tout document XML se compose :

d'un prologue (facultatif mais conseillé)

d'un arbre d'éléments (le contenu proprement dit du document)

de commentaires (facultatifs et qui peuvent apparaître aussi bien dans le prologue que dans l'arbre d'éléments) et d'instructions de traitements

Structure d'un document XML

- **Eléments**

Balise d'ouverture : **<** suivi immédiatement du nom de l'élément, suivi éventuellement d'attributs (précédé par au moins un séparateur), suivi éventuellement de séparateurs, suivi de **>**

<Titre> <Auteur nom="Dupont">

Balise de fermeture : **</** suivi immédiatement du nom de l'élément, suivi éventuellement de séparateurs, suivi de **>**

</Titre> </Auteur>

Rq : par rapport à HTML, un élément XML a forcément une balise de fermeture

Structure d'un document XML

Un élément peut contenir d'autres éléments, des données, des références à des entités, des sections littérales et des instructions de traitement

Exemple:

```
<Un>texte un
  <Deux>texte deux</Deux>
  <Trois attr1="1" attr2="deux">
   </Trois>
  texte trois
</Un>
```

Rq : les éléments doivent s'imbriquer correctement

Structure d'un document XML

- **Elément vide**

  <element_vide></element_vide>

ou plus simplement

  <element_vide />

Rq : pas de séparateurs entre < et / ni entre / et >

Exemple :

la balise HTML
 serait représentée en XML par
</BR> ou

Structure d'un document XML

- Attributs
  - paire *nom = valeur*
  - à l'intérieur d'une balise ouvrante
  - le nom d'un attribut doit respecté les règles de nommage.
  - la valeur est une chaîne de caractères encadrée par des guillemets (‘ ou ").
  - cette chaîne ne peut pas inclure les caractères %, & et ^.

Document bien formé/valide

Pour un document XML, on distingue 2 niveaux de vérifications:

- le document est bien formé s'il satisfait les critères de conformité
- le document est valide si il est bien formé et si il respecte un modèle de document type (DTD)

Document bien formé

Majuscules/minuscules : XML différencie strictement les majuscules et minuscules dans les noms d'éléments et d'attributs contrairement à HTML.

Guillemets pour les affectations de valeur : toutes les affectation de valeur à des attributs doivent être placées entre guillemets (‘ ou ").

Eléments : en XML, contrairement à HTML, les éléments sont constitués de repères d'ouverture et de repères de fermeture (sauf les éléments vides)

Nommage : les noms d'éléments et d’attributs doivent respecter des règles précises

Structure arborescente : un document XML doit présenter un seul élément racine qui englobe tous les autres.

DTD (Document Type de Document)

- XML introduit la notion de définition de type (de modèle) de document (DTD)
- Langage spécifique (non XML) permettant de définir un modèle que devront vérifier tous les documents XML y faisant référence
- Un document XML est valide s’il respecte une DTD
- Une DTD est composée de définitions d’éléments et d’attributs

DTD

- la DTD est définie dans un fichier et est référencée par tous les documents XML basés sur ce modèle

```
<!DOCTYPE nom SYSTEM "urldeladtd">
```

Rq : le nom du DOCTYPE doit être le même que le nom de l’élément racine

DTD : élément

- Syntaxe générale :
  <!ELEMENT nom_modele definition>
- Elément vide
  <!ELEMENT nom_modele EMPTY>
- Elément de données (textuelles car pas de typage)
  <!ELEMENT nom_modele (#PCDATA)>
- Elément de contenu quelconque
  <!ELEMENT nom_modele ANY>

DTD : élément

- Structure arborescente :
  un élément est composé d'éléments fils
- Dans un ordre imposé :
  <!ELEMENT elem_père (fils1, fils2, fils3)>
- De manière alternative :
  <!ELEMENT elem_père (fils1 | fils2)>
- Indicateurs d'occurrences pour éléments fils
  * : 0, 1 ou plusieurs fois
  ? : 0 ou 1 fois maximum
  + : 1 (minimum) ou plusieurs fois
  rien : 1 fois et une seule

DTD : élément

Exemple:

<!ELEMENT livre (titre, auteur+, editeur?, tome*) >

L'élément livre est composé de 4 éléments fils dans l'ordre :
- un titre et un seul (aucun indicateur d'occurrence),
- un auteur minimum mais éventuellement plusieurs (indicateur +),
- un éditeur maximum éventuellement aucun (indicateur ?),
- aucun, un ou plusieurs tomes (indicateur *)

DTD exemple

DTD : livre.dtd

```
<?xml version="1" encoding="UTF-8"?>
<!ELEMENT livre (titre, auteur+, editeur?, tome*) >
<!ELEMENT titre (#PCDATA)>
<!ELEMENT auteur (#PCDATA)>
<!ELEMENT editeur (#PCDATA)>
<!ELEMENT tome (#PCDATA)>
```

Document XML minimum valide relatif à la DTD livre

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE livre SYSTEM "livre.dtd">
<livre>
        <titre/>
        <auteur/>
</livre>
```


DTD exemple

Document XML relatif à la DTD livre

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE livre SYSTEM "livre.dtd">
<livre>
  <titre>XML langage et applications</titre>
  <auteur>Alain Michard</auteur>
  <auteur>G. Gardarin</auteur>
  <editeur>Eyrolles></editeur>
</livre>
```

DTD : attributs

Un attribut est un couple *nom=valeur*

Un attribut caractérise forcément un élément

Il n'y a pas d'ordre pour les attributs d'un élément

Syntaxe générale :

<!ATTLIST nom-élément nom-attribut type-attribut mode>

Un nom d'attribut doit vérifier les mêmes règles qu'un nom d'élément

DTD : attributs

Le mode d'un attribut correspond à la définition de la valeur implicite de l'attribut.

- #REQUIRED : attribut obligatoire dans l'élément
- #IMPLIED : attribut facultatif dans l'élément
- #FIXED valeur : l'attribut prend toujours la valeur fixée (constante)
- Valeur : valeur par défaut de l'attribut si aucune valeur n'est précisée.

DTD : attributs

Exemple :

```
<!ELEMENT hotel (#PCDATA)>

<!ATTLIST hotel
   nom            CDATA              #REQUIRED
    categorie          (I|II|III|IV|V)     #REQUIRED
    chambre_simple    (oui|non)          #IMPLIED
    chambre_double    (oui|non)          "oui"
>
```


JSON

JavaScript Object Notation

JSON

- JSON (JavaScript Object Notation) est aussi un langage de structuration de document (de données plus exactement)
- JSON est souvent utilisé (comme XML) pour transmettre de l'information structurée
- Même structure qu'XML : Imbrication hiérarchique d'éléments, mais syntaxe basée sur les littéraux JavaScript
- Json est moins verbeux qu'XML, documents moins volumineux
- Json étant basé sur javascript, il est facile à traiter par ce langage !

JSON

- Un document JSON comporte :
  - des couples clés (propriété) / valeurs
  - des listes ordonnées de valeurs
- Trois types de données pour les valeurs :
  - objet
  - tableau
  - type simple : chaîne, nombre, booléen, null

→ Normalement les noms des propriétés devraient toujours être en ""

JSON

- Exemple de déclaration d'un objet JSON en JavaScript :

```
// un objet avec 4 propriétés
let pers = {
        "nom" : "Pecatte", // une chaine de caractères
        "taille" : 1.75, // un nombre
        "prof" : true, // un booléen
        "langages" : [ "html", "css", "javascript"] // un tableau
};
```

- Accès aux informations : notation . et [ ]
  pers.nom pers.langages[0]

JSON

- Pour transformer un objet JSON en chaîne de caractère, on utilise l'expression JSON.stringify(json) avec le paramètre json représentant l'objet à transformer. Cette opération s'appelle une sérialisation.

```
var st = JSON.stringify(pers) ;
console.log(st);
```

Résultat :

{"nom":"Pecatte","prenom":"Jean-Marie","langages":["html","css","javascript"]}

JSON

- L'opération inverse qui consiste à transformer en un objet JSON une chaine de caractère s'appelle une désérialisation. L'expression qui réalise cela est JSON.parse(jsonstring).

```
var st = "{"nom":"Pecatte","prenom":"Jean-Marie","langages":["html","css","javascript"]}";
var personne = JSON.parse(st);
console.log(personne);
```

```
▼ Object
  ▶ langages: Array[3]
    nom: "Pecatte"
    prenom: "Jean-Marie"
```


AJAX

Trois manières de coder :

1) JS ancienne - XMLHttpReqest
2) JS nouvelle - API fetch
3) jQuery - $.ajax() , $.get(), …

Exemple de résultat attendu

Exemple : récupérer des données (au format JSON) d'un fichier stocké sur un serveur pour les afficher dans un navigateur

Fichier http://monserveur.fr/..../editeur.json

```
{"soft" : [ "Adobe Dreamweaver" , "Microsoft Expression Web", "Notepad++", "gedit", "Emacs"] }
```

Résultat à obtenir : les noms des logiciels sont affichés dans une liste HTML

- Adobe Dreamweaver
- Microsoft Expression Web
- Notepad++
- gedit
- Emacs

AJAX avec jQuery – doc JSON

Exemple :

Code HTML avant : une 'div' d'id « editeurs » vide

```
▶<div id="editeurs">…</div>
```

Code HTML après : la 'div' est remplie par une liste 'ul' contenant les noms des logiciels 'li'

```
▼<div id="editeurs">
  ▼<ul>
      <li>Adobe Dreamweaver</li>
      <li>Microsoft Expression Web</li>
      <li>Notepad++</li>
      <li>gedit</li>
      <li>Emacs</li>
    </ul>
  </div>
```

AJAX en javascript

Version ancienne

XMLHttpReqest

Une classe XMLHttpReqest avec des propriétés et des méthodes ; cette classe permet d'envoyer une requête http vers un serveur puis de récupérer les données.

- Gestionnaire d'événement :

onreadystatechange ..... fonction appelée lors de chaque étape de l'échange

- Etat : permet de connaître l'état d'avancement de la requête

readyState ..... état de l'échange, 5 valeurs :
UNSET (0, non initialisé),
OPENED (1, prêt),
HEADERS_RECEIVED (2, envoyé),
LOADING (3, en cours),
DONE (4, achevé)

- Requête :

open ..... code
setRequestHeader…..*ajouter un entête (header)*
timeout……..*temp max d'exécution de la requête*
upload……*données à envoyer*
send……*envoyer la requête*
abort……...*stoppe la requête*

- Réponse :

status ..... code du résultat de l'échange
bon si valeur dans [200,299]
(valeur nulle en cas d'accès local)
statusText ..... libellé du résultat de l'échange
responseType……*arraybuffer, blob, document, json, text*
response……..
responseText ..... texte du contenu reçu via HTTP
responseXML .....élément (DOM) du contenu reçu

AJAX en JavaScript

```
// fonction qui initialise correctement un objet XMLHTTPRequest
//   selon les navigateurs
function getXMLHttpRequest() {  var xhr = null;
  if (window.XMLHttpRequest || window.ActiveXObject) {  // cas particulier d'IE
     if (window.ActiveXObject) {  // selon la version d'IE...
          try { xhr = new ActiveXObject("Msxml2.XMLHTTP"); }
          catch(e) { xhr = new ActiveXObject("Microsoft.XMLHTTP"); }
     } else {  // tous les autres navigateurs
          xhr = new XMLHttpRequest(); }
   } else { alert("Votre navigateur ne supporte pas l'objet XMLHTTPRequest...");
            return null; }
   return xhr;
}
```


AJAX en JavaScript – doc JSON

```
var xhr =  getXMLHttpRequest();  // un objet de type XMLHttpRequest
// ajout d'un écouteur d'événement sur l'objet xhr
xhr.addEventListener("readystatechange", affichage);
function affichage() {
    // si l'état est 4 (Done, achevé) et si y a pas d'erreur
    if (xhr.readyState == 4 && (xhr.status == 200 || xhr.status == 0)) {
        afficheRes(JSON.parse(xhr.response)); // affichage du résultat
     }
};
// paramètres de la requête AJAX  : fichier JSON à charger
xhr.open("GET", "http://monserveur.fr/…./editeur.json", true);
// lancement de la requête
xhr.send(null);
```


AJAX en JavaScript – doc JSON

```
// fonction qui traite l'objet JSON envoyé par la requête AJAX
function afficheRes(dataJSON) {
   var editeurs = document.createElement("ul");
   for (var i=0 ; i < dataJSON.soft.length ; i++) {
          // création d'une liste HTML
          ol = document.createTextNode(dataJSON.soft[i]);
          li = document.createElement("li");
          li.appendChild(ol);
          editeurs.appendChild(li);
          }
   document.getElementById("editeurs").appendChild(editeurs);
}
```


AJAX en javascript

Version actuelle

fetch remplace XMLHttpRequest

Nouvelle API Fetch

- Remplacer XMLHttpRequest et $.ajax ?
- Une nouvelle API propose une fonction native pour cela : fetch()

  https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch
- La méthode fetch accepte 2 paramètres :
  - l'url de la ressource
  - des options comme la méthode (GET, POST, …)
    (cf. la doc pour la liste des options)
    https://developer.mozilla.org/fr/docs/Web/API/WindowOrWorkerGlobalScope/fetch
- Elle renvoie une promesse avec un objet de type Response; cet objet n'est pas le résultat de l'API mais la réponse HTTP ; response.json() permet d'en extraire le résultat au format JSON

Nouvelle API Fetch

- Algorithme général

```
const url = "…" // l'url de la ressource
let fetchOptions = { … } // les options de l'API fetch
// executer la req AJAX
fetch(url, fetchOptions)
.then( (response) => { return response.json() })
.then( (dataJSON) => { // dataJSON = les données renvoyées
  console.log(dataJSON) // ici le traitement des données …
                        // pour affichage dans le navigateur
}
.catch( (error) => {  // gestion des erreurs
        console.log(error)
 } )
```


Exemple API fetch

- Code de l'exemple

```
// l'url du fichier sur le serveur
const url = 'http://monserveur.fr/..../softs.json';
// les options de l'API fetch
let fetchOptions = { method: 'GET' }
// executer la req AJAX
fetch(url, fetchOptions)
.then( (response) => {
    return response.json()
 })
```

→ suite…

Exemple API fetch

```
.then( (dataJSON) => {
  let softs = dataJSON.soft

  let texteHTML = "<ul> »

  for (let v of softs) {
    texteHTML += `<li>${v}</li>`
  }
  texteHTML += "</ul> »

  document.getElementById('editeurs')
      .innerHTML = texteHTML

})
.catch( (error) => {
   console.log(error)
})
```

softs = tableau, contient les noms des logiciels
texteHTML = string, doit contenir le texte HTML …
…

la div editeurs reçoit le texteHTML

gestion des erreurs

Exemple API fetch

➔Variante sans boucle → Utilisation de « reduce »

```
.then( (dataJSON) => {
   let softs = dataJSON.soft
   let texteHTML = "<ul>"
      + softs.reduce( (res, element) =>
            `${res}<li>${element}</li>`, "")
      + "</ul>"
   document.getElementById('editeurs')
               .innerHTML = texteHTML
})
.catch( (error) => { console.log(error) } )
```


AJAX en jQuery

AJAX avec jQuery

Manipulations de XMLHttpRequest en jQuery

- opérations simplifiées : $.get(), $.post() et .load()
- opération avec contrôle complet : $.ajax()

Opérations simplifiées

$.get(url, données, callback, datatype)

ou $.post(url, données, callback, datatype)

url : url du contenu à récupérer

données : tableau associatif ou null

callback : fonction appelée à la fin de l'échange ; elle a en paramètre le résultat au format texte et l'état

datatype : xml, json, script, or html

AJAX avec jQuery – doc JSON

```
$.get("http://monserveur.fr/.../editeur.json",null, // appel AJAX
                                        afficheRes, "json");

function afficheRes(dataJSON) {  // fonction d'affichage
   var res = $("<ul>"); // creation d'une liste d'éléménts
   var softs =  dataJSON.soft;  // le tableau des softs
   for (var i=0 ; i < softs.length ; i++) {
         txt = softs[i];
         res.append("<li>"+txt+"</li>"); // ajout du nom à la liste
   }
   $("#editeurs").append(res); // ajout de la liste dans la div d'id
editeurs
}
```

Opération avec contrôle complet

$.ajax(paramètres sous la forme d'un tableau associatif)

url : *adresse* ……..url du contenu à récupérer

type : *méthode* ….."GET" par défaut ou "POST"

dataType : *type*….."html", "xml", "text","json"

cache : *booléen*…. récupérer la copie du contenu si déjà dans le cache du navigateur (vrai par défaut)

error : *fonction*……fonction appelée en cas d'erreur

success : *fonction*..fonction appelée en cas de succés


```
// appel AJAX avec la fonction $.ajax
$.ajax( {
        url: "editeur.json",
        type : "get",
        success : afficheRes,
        dataType: "json"
        });
// la fonction afficheRes est la même
```